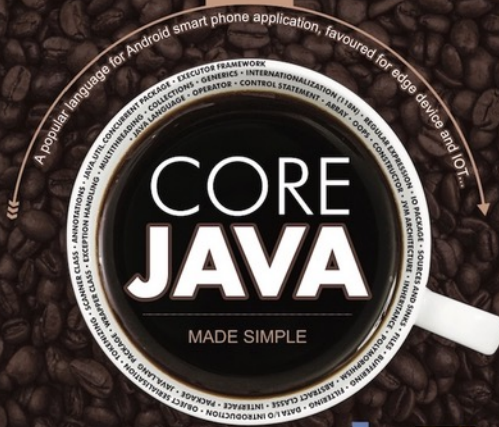


**BPB PUBLICATIONS**

Core Java

Made Simple

By
Som Prakash Rai



BPB PUBLICATIONS

20 Ansari Road, Daryaganj, New Delhi-110002

FIRST EDITION 2018

Copyright © BPB Publication, INDIA

ISBN: 978-93-8655-197-9

All Rights Reserved. No part of this publication can be stored in a retrieval system or reproduced in any form or by any means without the prior written permission of the publishers

LIMITS OF LIABILITY AND DISCLAIMER OF WARRANTY

The Author and Publisher of this book have tried their best to ensure that the programs, procedures and functions described in the book are correct. However, the author and the publishers make no warranty of any kind, expressed or implied, with regard to these programs or the documentation contained in the book. The author and publisher shall not be liable in any event of any damages, incidental or consequential, in connection with, or arising out of the furnishing, performance or use of these programs, procedures and functions. Product name mentioned are used for identification purposes only and may be trademarks of their respective companies.

All trademarks referred to in the book are acknowledged as properties of their respective owners

Distributors:

BPB PUBLICATIONS

20, Ansari Road,
Darya Ganj New Delhi-110002
Ph: 23254990/23254991

BPB BOOK CENTRE

376 Old Lajpat Rai Market,
Delhi-110006
Ph: 23861747

MICRO MEDIA

Shop No. 5, Mahendra Chambers 150
DN Rd. Next to Capital Cinema, V.T.
(C.S.T.) Station, MUMBAI-400 001
Ph: 22078296/22078297

DECCAN AGENCIES

4-3-329, Bank Street,
Hyderabad-500195
Ph: 24756967/24756400

Preface

The author is confident that the present work will come as a relief to the students wishing to go through a comprehensive material for understanding Java concepts in layman's language, offering a variety of analogical understanding and code snippets.

This book promises to be a very good starting point for beginners and an asset to advance users too. Difficult concepts of Core Java are given in an easy way, so that the students are able to understand them in an efficient manner.

It is said "To err is human, to forgive is divine". In this light, we wish that the shortcomings of the book will be forgiven. At the same, the author is open to any kind of constructive criticisms and suggestions for further improvement. All intelligent suggestions are welcome and the author will try its best to incorporate such invaluable suggestions in the subsequent editions of the book.

Table of Content

Chapter 1 Introduction to Core Java

1.1	Introduction.....	1
1.2	Difference Between C Programming and Java Programming.....	1
1.3	Important Terms to Remember	2
1.4	Install The JDK	3
1.4.1	Setting Path	6
1.4.2	How to Compile Java Program.....	8
1.4.3	Running The Java Program	9

Chapter 2 Java Language

2.1	Character Set.....	12
2.2	Data Types.....	12
2.3	Keywords.....	15
2.4	Identifiers or User Define Words	15
2.5	Variable	17
2.6	Constant.....	19
2.7	Literals.....	20

Chapter 3 Operator

3.1	What is Operators.....	37
3.2	Types of Operators	37
3.3	Type Casting	54

Chapter 4 Control Statement

4.1	Introduction	59
4.2	Types of Control Statement	59
4.2.1	Conditional Control Statement.....	59
4.2.2	Looping Control Statement.....	67

Chapter 5 ARRAY

5.1	Introduction	74
5.2	Types of Array	74
5.3	Multidimensional Array	81

Chapter 6 OOPS Concept

6.1	Introduction	86
6.2	Class	87
6.3	Object	87
6.4	Variable	88
6.5	Blocks	95
6.6	Internally by the JVM	99
6.7	Class Loader Concept	101
6.8	Methods.....	103
6.9	Volatile Modifier	114
6.10	Method Overloading.....	114

Chapter 7 Constructor

7.1	What is Constructor	123
7.2	Final Variable and Static Final Variable	128

Chapter 8 JVM Architecture

8.1	Virtual Machine	133
8.2	JVM	133
8.3	“this” Keyword.....	134
8.4	VAR-ARG Method.....	153
8.5	Local Variable.....	159
8.6	Local Block	159
8.7	Assertion.....	164

Chapter 9 Inheritance

9.1	What is Inheritance.....	168
9.2	Difference between this and super keyword	175
9.3	S-A Relationship	176
9.4	Composition vs Aggregation	176
9.5	Coupling	176

Chapter 10 Polymorphism

10.1	What is Polymorphism	178
10.2	Method Overriding	181

Chapter 11 Abstract Class

11.1	What is Abstract Class.....	194
------	-----------------------------	-----

Chapter 12 Interface

12.1	What is an Interface.....	204
12.2	Multiple Inheritance Through Interface.....	207

12.3	Inner Class.....	215
12.4	Adapter Class	233

Chapter 13 Package

13.1	What is Package	235
13.2	Import Package	239

Chapter 14 JAVA.LANG Package

14.1	Introduction	248
14.1.1	Java.Lang.Object Class	248
14.1.2	Java.Lang.String Class	279
14.1.3	Java.Lang.StringBufferclass.....	295
14.1.4	Java.Lang.System Class	301

Chapter 15 Wrapper Class

15.1	What are Wrapper Class	309
15.2	Utility Method.....	310
15.2.1	ValueOf() Method	310
15.2.2	xxxValue() Method	311
15.2.3	ParseXxx() Method	313
15.2.4	toString() Method	314

Chapter 16 Exception Handling

16.1	Introduction	320
16.2	Exception	320
16.2.1	Handling Exception by Writing the Multiple Try and Catch	323
16.3	Finally Block	325
16.4	Types Of Exceptions	331
16.5	“throw” Keyword	334
16.6	“throws” Keyword	335
16.8	Method	339
16.9	Difference between class not found Exception & No Class Deffound Error	345

Chapter 17 Multithreading

17.1	What is Multithreading	349
17.2	Java Thread	350
17.3	User Define Thread	351
17.4	Thread Priority	355
17.5	Thread Life Cycle	357
17.6	Synchronization.....	364
17.7	Inter Thread Communication	370
17.8	Lazy Thread	373

17.9	Deadlock vs Starvation	373
17.10	How to kill a thread in the middle of the line	373
17.11	Suspend and Resume Method	374
17.12	RACE Condition	374
17.13	Thread Local (1.2 v)	374
17.14	Green Thread	374
17.15	Thread Local	375
17.16	Constructor	375
17.17	Java.util.concurrent.lock package	376
17.18	Lock(l)	377
17.19	Reentrantlock	378
17.20	Thread Pools	383
17.21	Callable and Future	384

Chapter 18 Collections

18.1	Java.util package or Collection Framework	391
18.2	Collection	393
18.3	List Interface	393
18.4	SET	400
18.5	Map	409
18.6	Difference Between Comparable and Comparator	411
18.7	String Tokenizer	413
18.8	Has More Token	414
18.9	Difference Between Enumeration and Iterator	418
18.10	Difference Between Iterator and List Iterator	419
18.11	AddElements	419
18.12	Timer and Timer Task	419
18.13	Methods.....	419
18.13.1	Stack	421
18.13.2	Null Acceptance	421
18.14	Entry Interface	422
18.15	Identity Hash Map.....	422
18.16	Week Hash Map	423
18.17	Sorted Map	424
18.20	Properties	424
18.21	Queue	426
18.22	Priority Queue.....	427
18.23	Navigable Set	429
18.24	Navigable Map	430

Chapter 19 Generics

19.1	Introduction	440
19.2	Generics Classes.....	441

19.3	Bounded Types.....	444
19.4	Generic method and wild-card character	444

Chapter 20 Internationalization (118N)

20.1	Introduction	450
20.2	Locale	450
20.3	Number Format	452
20.4	Setting Maximum,Minimum,Fraction and Integer digits	454
20.5	Data Format	454
20.6	System Properties	456

Chapter 21 Regular Expression

21.1	Introduction	458
21.2	Patten Class	458
21.3	Matcher	459
21.4	Character Classes	459
21.5	Quantifiers	460
21.6	Pattern Class Split() Method	460

Chapter 22 IO Package

22.1	Introduction	462
22.2	Stream.....	462
22.3	What to use.....	465
22.4	Filtering	465
22.5	Storing data records	466
22.6	Exceptions	466
22.7	What are Readers	470
22.8	What are Writers.....	471

Chapter 23 Sources and Sinks

23.1	Introduction	473
23.2	Byte Arrays	473
23.3	Pipes in Java code	473
23.4	Char Arrays.....	477

Chapter 24 Files

24.1	Introduction	481
24.2	The File Classes	481
24.3	Constructors.....	481
24.4	Test Methods	481
24.5	Action Methods	482
24.6	List Methods	482

Chapter 25 Buffering

25.1	What is Buffering	488
25.2	Buffered Reader and Buffered Writer	489

Chapter 26 Filtering

26.1	What is Filtering	494
26.2	Pushback	494
26.3	Checksumming	496
26.4	Digesting	498
26.5	Checksum Versus MessageDigest	500
26.6	Inflating and Deflating	500
26.7	ZipInputStream	503
26.8	ZipOutputStream	504

Chapter 27 Data I/O Introduction

27.1	What is Data I/O	507
27.2	Unicode Text Format	507
27.3	Random Access File	509

Chapter 28 Object Serialization

28.1	What is Object Serialization	512
28.2	How Object Serialization Works	512
28.3	Reading an Object From a File	513
28.4	Alternative Ways to read back	514
28.5	Add Serializable	514
28.6	Custom Serialization	515
28.7	Externalizable	516
28.8	Customized Serialization	521
28.9	Serialization With Respect to Inheritance	522
28.10	Externalization	524
28.11	Difference between Serialization & Externalization	526
28.12	Serial Version UID	527

Chapter 29 Tokenizing

29.1	Introduction	529
29.1.1	Using String Tokenizer	529
29.1.2	Stream Tokenizer	530

Chapter 30 Scanner Class

30.1	Introduction	533
30.2	Class Declaration	533
30.3	Examples of Interactive Input/Output in Java	534

30.4	Class Constructors	535
30.5	Class Methods	536

Chapter 31 Annotations

31.1	Introduction	544
31.2	Types of Annotations	545
31.3	Built-In Annotations	547

Chapter 32 Java.util.concurrent package

32.1	The Java.util.concurrent	558
32.2	Other Atomic Classes are	560

Chapter 33 Executor Framework

33.1	Introduction	562
------	--------------------	-----

Chapter 1

Introduction to Core Java

1.1 Introduction

Java is a high-level programming language which was originally developed by Sun Microsystems which was initiated by James Gosling and released in 1995 as core component of Sun Microsystems' Java platform (Java 1.0 [J2SE]). It runs on variety of platforms, such as Windows, Mac OS, and the various versions of UNIX. The latest release of the Java Standard Edition is Java SE 9. With the advancement of Java and its widespread popularity, multiple configurations were built to suit various types of platforms.

1.2 Difference between C Programming and Java Programming

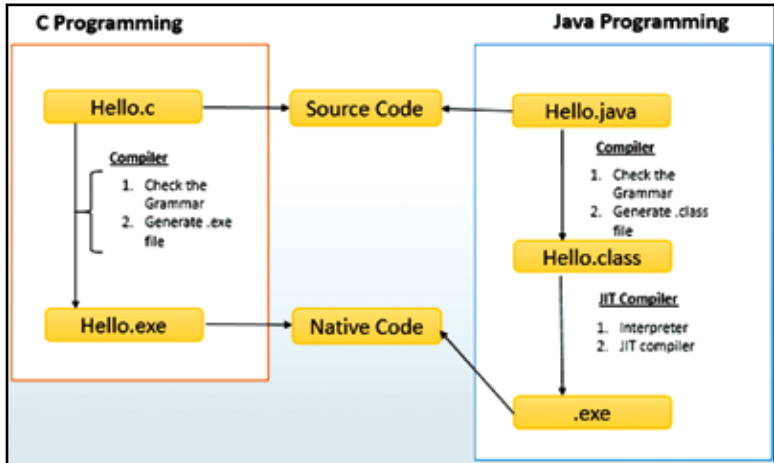


Fig 1.1: C Programming and Java Programming

C program will be saved with .c extension. After successful compilation of 'c' program native code or .exe file will be generated which is platform dependent and is cannot be used on any other platform. This .exe file is platform dependent because compatibility issue related to corresponding platform compatibility will be added. When we are writing java program, it must be saved with java extension. After successful compilation of java program .class file will be generated which is

known as byte code, and it will be platform independent despite of corresponding platform JDK/JRE available. (Fig 1.1)

Q 1: Why in 'C' programming language native code is available where as in java native code is not available?

Ans: 'C' programming supports static loading which means if the corresponding member of program is required in further program then everything will not be loaded into the main memory where as Java supports dynamic loading which means if any method is required then it will be allocated in main memory and it will get immediately destroy after the use.

1.3 Important terms to remember

1. **Source Code:** The program which is written by the developer (Human Readable Format).
2. **Byte Code:** After successful compilation of java program byte code will be generated i.e. .class file, which is called as byte code. That byte code will be processed further while running the program.
3. **Java Compiler:** It is a program with the name of javac.exe which is mainly responsible for checking the grammar of the program language. It will also generate '.class' file after the successful compilation of the program file with the name of class.
4. **Interpreter:** It is a program with the extension of java.exe. Interpreter is mainly responsible for generating byte code to the native code line by line. Converting byte code to native code is time consuming process. So in order to overcome this problem from java2 JIT compiler has been introduced.
5. **JIT (Just In Time) Compiler:** It is used to convert byte code to the native code at a time by adding corresponding platform compatibly.
6. **JVM (Java Virtual Machine):** It provides platform independent execution environment.
Note: JVM itself is platform dependent. Byte Code is platform independent because of JVM available corresponding for that machine.
7. **JRE (Java Runtime Environment):** It provides many JVM in order to run the program. It comes in different version associated with the corresponding JDK.
8. **JDK (Java Development Kit):** It comes into different versions according to Sun Oracle released.

- **JDK 1.2 (Playground)**
- **JDK 1.4 (Merline)**
- **JDK 1.5 (Tiger)**

- **JDK 1.6 (Mustang)**
- **JDK 1.7 (Dolphin)**
- **JDK 1.8 (Spider)**

Note: JDK consist of different types of tools JRE, JVM in order to develop the java application.

Q 2: State whether following is True or False.

1. 'C' programs are platform independent. (False)
2. Java programs are platform independent. (True)
3. 'C' compiler is platform dependent. (True)
4. Java Compiler is platform Independent. (False)
5. JDK is platform Independent. (False)

1.4 Install the JDK

Steps

1. Open JTC DVD.
2. Open folder Java8.
3. Double click on JDK 1.8. **(Fig 1.2)**



Fig 1.2: Java SE Development Kit 8 Setup

4. Click on next.
5. Provide the destination folder where exactly you wanted to install JDK 1.8 or

else if you do not want to specify any specific location then click on next. **(Fig 1.3, 1.4)**

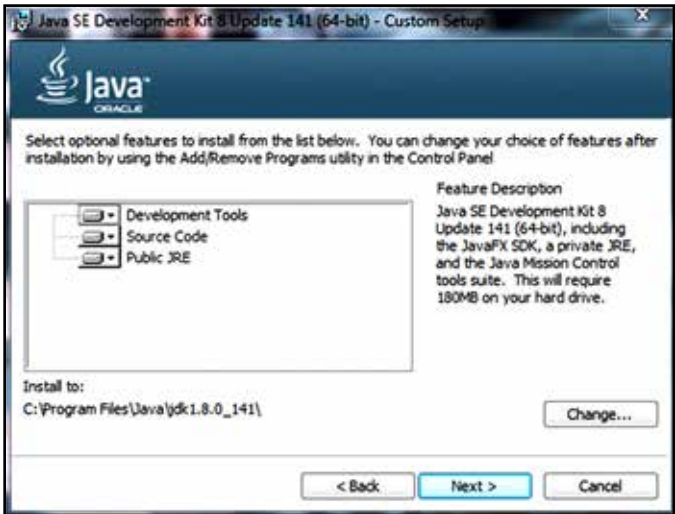


Fig 1.3: Custom Setup

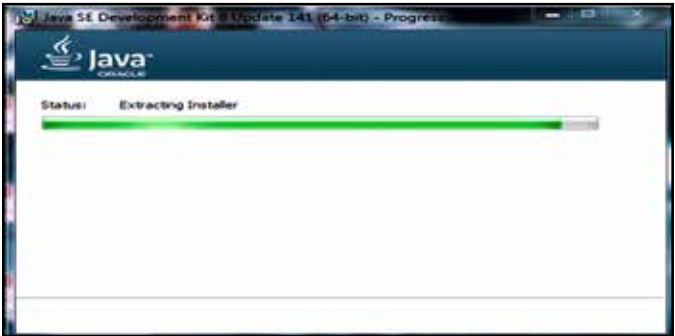


Fig 1.4: Progress

6. Click on next. **(Fig 1.5, 1.6)**



Fig 1.5: Destination Folder



Fig 1.6: Java Setup Progress

7. Click Finish. (**Fig 1.7**)



Fig 1.7: Java SE Development Kit 8 - Complete

After successful installation of JDK, you need to set path for the 'bin' of the JDK. It is not sufficient just to install JDK in your machine but parallel you need to set the JDK of bin.

1.4.1 Setting Path

Path can be set in two different ways:

1. Temporary path setting.

To set the Temporary path

1. Go to your corresponding working directory in command prompt.
2. Copy JDK/Bin folder location.
3. Type on command prompt. **(Fig 1.8)**

Set path =%path%;C:\Java\Jdk1.7\bin

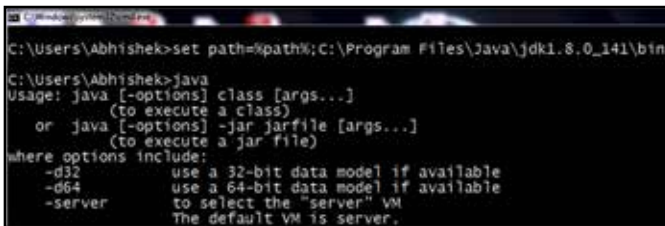


Fig 1.8: Command Prompt

When you set path in temporary manner, then each time when you try to compile and run java program or when you open command prompt then you need to set the path, which is time consuming process.

In order to avoid the problem with temporary path setting we can set the path in permanent manner.

2. Permanent path setting.

To set the permanent path:

1. Right click on My Computer Icon.
2. Click on properties.
3. Click on advance tab. **(Fig 1.9)**

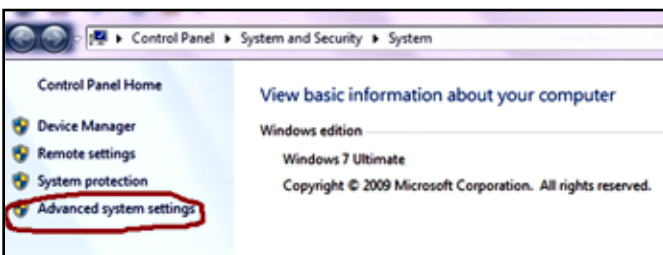


Fig 1.9: Advance System Settings

4. Click on Environment variable. **(Fig 1.10)**

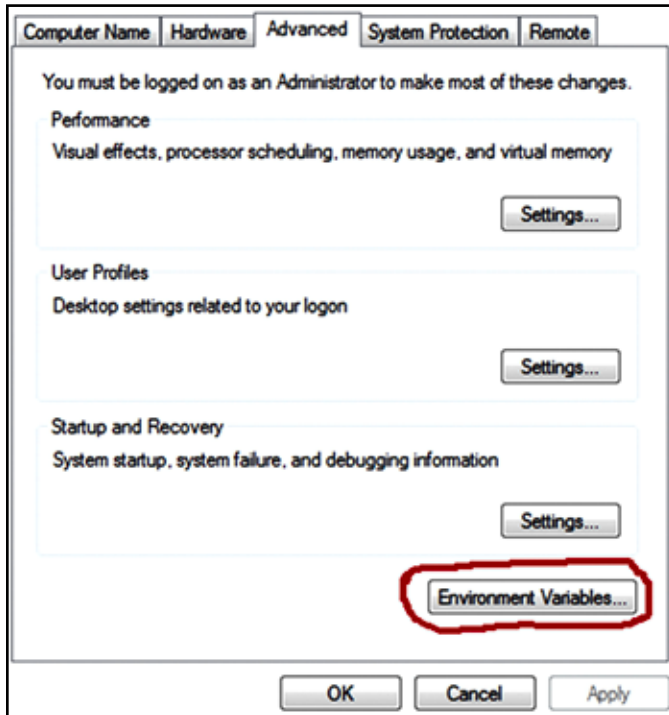


Fig 1.10: Environment Variables

5. Click on new under user variable. **(Fig 1.11)**

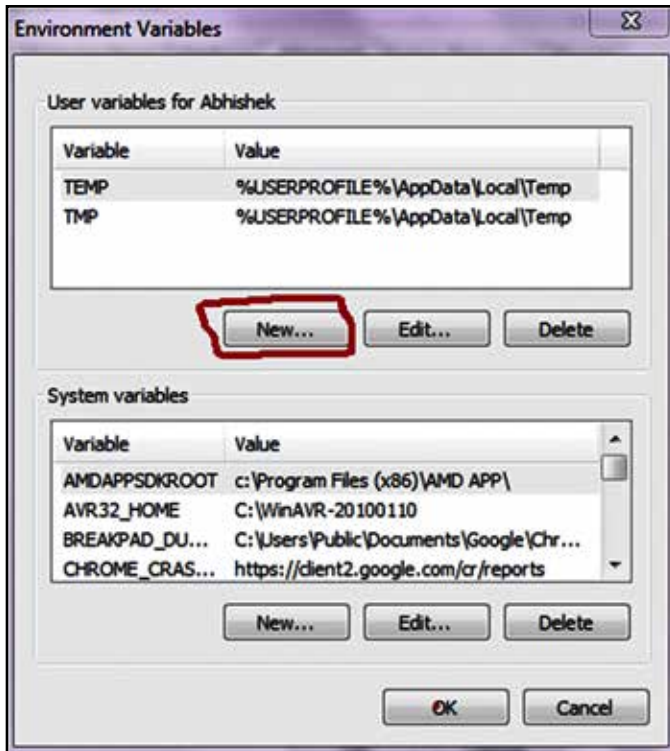


Fig 1.11: New Variables

6. Provide variable name as path and variable value as **path%;C:\Java\Jdk1.7\bin** (bin folder location)
7. Click on Ok. (Fig 1.12)

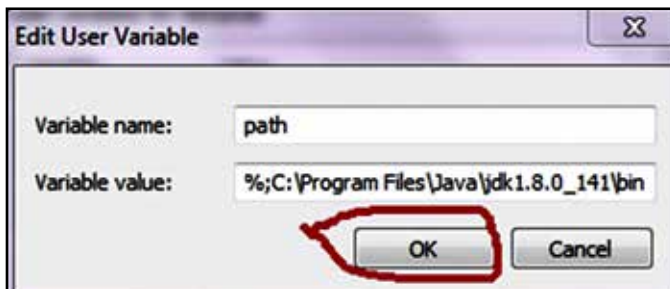


Fig 1.12: Edit User Variable

After setting the permanent path you need not to set the path again and again.

1.4.2 How to Compile Java Program

1. Write your java program and save that program with .java extension.

2. After successful compilation of java program (.class) file will be generate which is called **“byte code”** with the name of class.
3. The entire java program must be written inside the class.
4. To compile java program open your command prompt.
5. Go to the corresponding working directory.
Syntax: javac filename (by which you have saved java program). E.g.: javac Jtc1.java
6. At the time of compiling the java program you need to provide the file name by which your java program has been saved.
7. You can save your java program by any name followed by ‘.java’ extension.
8. .class file will be generated with the name of class not with the name of the file name.
9. It is always recommendable to save the java program with the name of class which contains main method.

1.4.3 Running the java program

At the time of running the java program you need to provide the .class file name which contains main method. E.g.:- java Jtc1

Java -verbose } Commands
jconsole

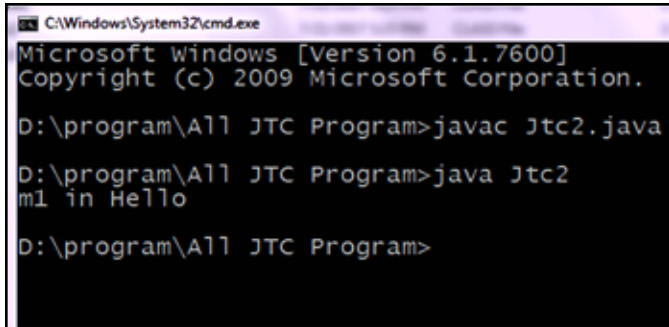
Program 1.1

```

class Hello{
    void m1(){
        System.out.println("m1 in Hello");
    }
}

public class Jtc2 {
    public static void main(String[] args) {
        Hello h1=new Hello();
        h1.m1();
    }
}

```



```

C:\Windows\System32\cmd.exe
Microsoft Windows [Version 6.1.7600]
Copyright (c) 2009 Microsoft Corporation.

D:\program\All JTC Program>javac Jtc2.java

D:\program\All JTC Program>java Jtc2
m1 in Hello

D:\program\All JTC Program>

```

Output:

Compile: javac Jtc2.java – Two .class file will be generated first with Hello and second with Jtc2.

Run : java Jtc2
 M1 in Hello – output

Run : java Hello

Error : Main method not found in class Hello please define the main method as: Public static void main(String args[])

Above program can be saved by any name followed by .java extension while compiling the program you need to provide the file name by which you have saved the program.

After successful compilation of above program two .class file will be generated i.e. Jtc2 and Hello.

Note

- The .class file generation does not depend upon by which you have saved program or main method. As many number of classes available inside your source file that many number of .class file will be generated.
- While running the program you need to provide the (.class) file name which contains main method that is Jtc2.

Q 3: Let's consider that you are saving a java program by some name followed by .java extension but within that editor you have not written any single thing then what will happen when we compile it, will be considered as Java program or not?

Ans: It will be considered as java program but in order to compile successfully some context must be there inside, but if corresponding editor is not having any content then there is no question of compiling the program. So, even we cannot

expect any .class file will be generated.

Q 4: Can we perform any kind of changes or any kind of replacement in byte code by the keyboard.

Ans: No, you cannot. If you are doing any changes in existing bytecode then you will not be able to run that program successfully because whatever input you are providing from the keyboard, it will be there in the form of input stream which is having some different meaning then the existing one.

Chapter 2

Java Language

2.1 Character Set

There are mainly following types of character set in java.

- 1. A to Z
- 2. a to z
- 3. 0 to 9

Basically, these are all the special symbol available on your keyboard. If you are developing any application in ‘C’ programming language then it can support maximum English language because ‘C’ supports 8-bit ASCII character set. Whereas Java supports 16-bit UNICODE Character Set. So, by using Java as an application, it can support many languages.

When you are developing any application in Java then with the help of Internationalization (I18N) or localization. Till UNICODE 4.0 it was supporting almost all the languages but it was not supporting Sanskrit and some other languages. But from UNICODE 5.0 it supports Sanskrit ₹ and many more new inclusions are there.

2.2 Data Types

There are mainly two types of data type.

a) Primitive Data Type:

Data Type	Size (byte)	Default Value	Range
Byte	1	0	-2^7 to 2^7-1
short	2	0	-2^{15} to $2^{15}-1$
int	4	0	-2^{31} to $2^{31}-1$
long	8	0	-2^{63} to $2^{63}-1$
float	4	0.0f	
double	8	0.0, 0.0d	
boolean	-	false	
char	-	blank space	0 to 65535

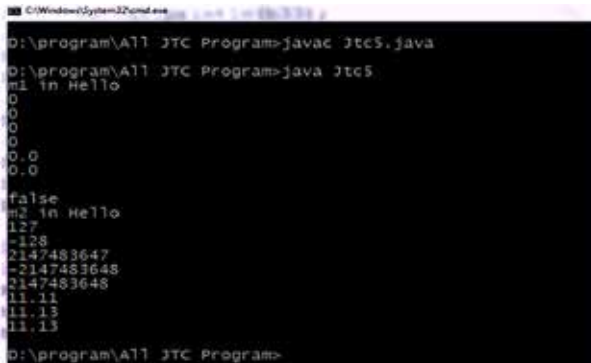
Program 2.1

```
class Hello{
    byte b1;
    short s1;
    int i1;
    long l1;
    float f1;
    double d1;
    boolean b11;
    char c1;
    void m1(){
        System.out.println("m1 in Hello");
        System.out.println(b1);
        System.out.println(s1);
        System.out.println(i1);
        System.out.println(l1);
        System.out.println(f1);
        System.out.println(d1);
        System.out.println(c1);
        System.out.println(b11);
    }
    void m2(){
        System.out.println("m2 in Hello");
        byte b11=127;
        byte b22=-128;
        //byte b33=128;
        //byte b44=-129;
        int i11=2147483647;
        int i12=-2147483648;
        //int i13=2147483648;
        //long l14=2147483649;
        long l12=2147483648;
        //float f11=11.11;
        float f12=11.11f;
        double d11=11.12;
        double d12=11.13d;
        System.out.println(b11);
    }
}
```

```

        System.out.println(b22);
        //System.out.println(b33);
        //System.out.println(b44);
        System.out.println(i11);
        System.out.println(i12);
        //System.out.println(i13);
        System.out.println(l12);
        //System.out.println(f11);
        System.out.println(f12);
        System.out.println(d12);
        System.out.println(d12);
    }}
public class Jtc5 {
    public static void main(String[] args) {
        Hello h1=new Hello();
        h1.m1();
        h1.m2();
    }
}

```

Output:


```

C:\Windows\System32\cmd.exe
D:\program\All JTC Program>javac Jtc5.java
D:\program\All JTC Program>java Jtc5
m1 in Hello
0
0
0
0
0.0
0.0
false
m2 in Hello
127
-128
2147483647
-2147483648
2147483648
11.11
11.13
11.13
D:\program\All JTC Program>

```

b) User Define Data Type:

There are mainly two types of user define data types:

- I. Class
- II. Interface

From JDK 1.5 two more data types have been introduced:

- I. Enum
- II. Annotation

2.3 Keywords

Keywords are having some predefined meaning which cannot be changed. In your further programme as keywords cannot be used as identifiers or user defined words.

List of Keywords:

Abstract	Assert	Boolean	Break	Throw	Byte
Case	Transient	Catch	Char	Class	Const
Continue	Default	InstanceOf	Do	double	Else
Enum	Extends	Final	finally	Float	For
Goto	If	implements	import	Int	Interface
Long	Native	New	package	while	private public
Short	static	strictfp	super	switch	Synchronized
This	protected	Try	void	volatile	Return

2.4 Identifiers or User Define Words

Identifiers will be used as class variables methods etc. While naming the identifiers or user define words remember the following paths:

1. Keywords cannot be used as identifiers.
2. Identifier must not start with any numeric or it is always recommendable to starts some of the identifiers with small case.
3. It must not contain any special symbol other than (`_` or `$`).
4. Identifier must not contain any blank space in between.
5. Identifiers may contain numeric but it should not be in starting.

e.g.:

```
int ab; //ok
```

```
int a_b; //ok
```

```
int a b; // not ok
```

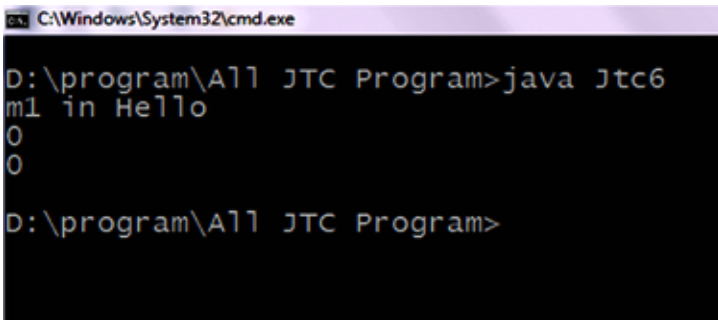
Program 2.2

```
class Hello{
    int a;
    int ab;
    int ab1;
    //int 1ab; - not ok
    int a1b;
    int a_b;
```

```

int ab_;
int _ab;
int a$b;
//int _; - identifier might not be supported in releases after Java SE 8
int _$;
//int ab*; - not ok
//int ()_; - not ok
//int for; - not ok
//int true; - not ok
int Integer;
int INT;
int For;
void m1(){
    System.out.println("m1 in Hello");
    System.out.println(Integer);
    System.out.println(INT);
}
}
public class Jtc6 {
    public static void main(String[] args) {
        Hello h1=new Hello();
        h1.m1();
    }
}

```

Output:


```

C:\Windows\System32\cmd.exe
D:\program\All JTC Program>java Jtc6
m1 in Hello
0
0
D:\program\All JTC Program>

```

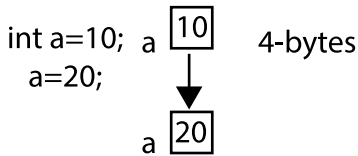
Note: int Integer; Integer INT; int java.lang.Integer;

- When we are writing int Integer, then it is being considered as variable of type int. This “Integer” will not be considered as keyword.

- When we are writing `java.lang.Integer` then in this case it is invalid declaration because special symbols are not allowed in identifiers name.
- In java corresponding to the primitive data type wrapper classes has been designed which is available in `java.lang` package. Therefore in “Integer int” , here Integer will be considered as wrapper class and ‘int’ will be considered as a reference variable of type Integer.

2.5 Variable

Variables will be of some specific data type which contains some value according to the compatibility of corresponding data type. The variable value can be changed in further program. The value which is been assign to any variable will be loosely coupled with that specific memory location.



The variable will be mainly of two types

- I. **Class Level Variable:** It can be accessed anywhere within the class. If you are not assigning any value to the class level variable corresponding to data type, it will be assigned internally by the JVM.
 - Two class level variables cannot have same name.
 - The variable which is directly define inside the class is called as class level variable.
- II. **Local Variable:** The variable which is define inside any block, method, constructor or inside any local context is called as local variable. Local variables will not be initialized implicitly by the JVM i.e. that local variable must be initialised explicitly by the developer before the use.

Local variable memory will be allocated only when that corresponding context is being processed and immediately after processing the context that memory will be destroyed. That is the region you cannot assess the local variable outside context.

You can name local variable and class level variable with the same name but when you are accessing that local variable in that context itself then that local variable only will be accessed. In order to access the class level variable within that local context you need use the “this” keyword.

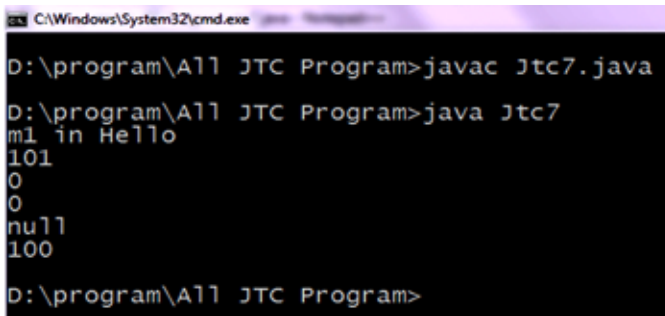
Program 2.3

```

class Hello{
    int a=10;
    int b;
    int Hello;
    Hello h1;
    void m1(){
        int ab=100;
        int bc;
        int a=101;
        //int b;
        System.out.println("m1 in Hello");
        System.out.println(a);
        System.out.println(b);
        System.out.println(Hello);
        System.out.println(h1);
        System.out.println(ab);
        //System.out.println(bc);
    }
}

public class Jtc7 {
    public static void main(String[] args) {
        Hello h1=new Hello();
        h1.m1();
    }
}

```

Output:


```

C:\Windows\System32\cmd.exe
D:\program\All JTC Program>javac Jtc7.java
D:\program\All JTC Program>java Jtc7
m1 in Hello
101
0
0
null
100
D:\program\All JTC Program>

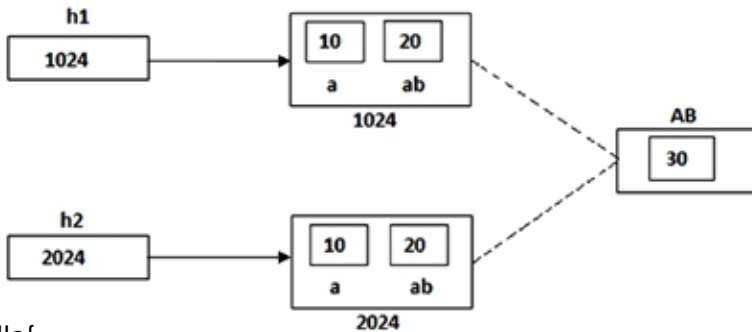
```

2.6 Constant

If you wanted to declare any variable with the fixed value which must not change in further program. So declare that variable as “final”.

1. final in tab=20;
2. static final int ab1=20;
3. static final int AB=30;

Program 2.4



```
class Hello{
    int a=10;
    final int ab=20;
    static final int AB=30;
}
Hello h1= new Hello();
Hello h2= new Hello();
```

When you are writing final int ab then for ab, memory allocation depends upon number of objects creation. (Figure 2.1) In case of static final, int AB memory will be allocated only once for n number of object also.

According to JLS (Java Language Specification) static final variable must be in caps.

Final variable will not be initialized automatically by JVM explicitly, you must have to initialize final variable.

Q 1: Can we change the value of the final variable?

Ans: There are some conditions before changing the value of the final variable, such as if it is instance final variable and it being initialize there itself when you have declared then you will not be able to change the value of the final variable and If it is instance final variable and its not been initialized at the time of declaration then in different objects context you will be able to have the different values for the same final variable.

Q 2: Is it always necessary to initialize the value of the final variable at the time of declaration?

Ans: No, it is not necessary to initialize the value of final variable at the time of declaration. It can be initialized in following different ways:

- I. At the time of declaration, With the help of block (if it is instance variable then we need to initialize it by using instance block whereas in the case of static variable we need to initialize explicitly in static block.
- II. Also we can initialize it by using constructor.

2.7 Literals

Literals are the value which can be assigned to the corresponding type of data type variables. There are mainly following types of literals:

a) **Integer Literal:** Integer literal is further divided into following parts:

- I. **Decimal Literal:** Decimal literal must not start with 0 i.e. it can be anything within 0 to 9.

e.g. `int i1=1234;`
 `int i2=89746;`
 `int i3=0123456; // not ok`

In case of i3 it will give error i.e. if it is starting with zero then it will be considered as octal but octal must be within the range of 0 to 7.

- II. **Octal Literal:** Octal literals must be within 0 to 7 but it must start with 0.

e.g. `int a=01234;`
 `int b=01267;`
 `int c=01248; // not ok`

- III. **Hexadecimal literals:** Hexadecimal literal must start with zerox (0X) or 0x and it must be within the range of 0 to 9 and A to F.

Note: a to f or A to F both are having the same meaning.

e.g. `int i1=0x1234A;`
 `int i2=0X1234a;`
 `int i3=1234a0X //not ok`

It is not necessary that Hexadecimal literal must have 0X somewhere in between but it is parallelly important that the Hexadecimal no must start with 0X.

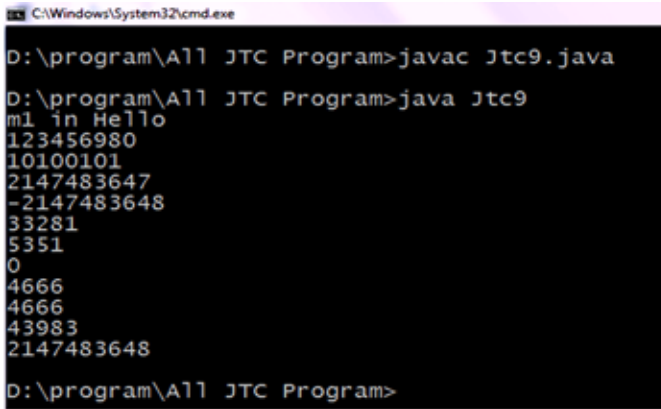
Program 2.5

```
class Hello1{
    int i1=123456980;
    int i2=10100101;
```

```

int i3=2147483647;
int i4=-2147483648;
//int i5=2147483648;
int i6=0101001;
int i7=012347;
//int i8=0123478;
int i9=000;
int i10=0x123a;
int i11=0X123A;
int i12=0XABCF;
//int i13=123A0X;
long l1=2147483648l;
void m1(){
    System.out.println("m1 in Hello");
    System.out.println(i1);
    System.out.println(i2);
    System.out.println(i3);
    System.out.println(i4);
    //System.out.println(i5);
    System.out.println(i6);
    System.out.println(i7);
    //System.out.println(i8);
    System.out.println(i9);
    System.out.println(i10);
    System.out.println(i11);
    System.out.println(i12);
    //System.out.println(i13);
    System.out.println(l1);
}
}
public class Jtc9 {
    public static void main(String[] args) {
        Hello1 h1=new Hello1();
        h1.m1();
    }
}

```

Output:


```

C:\Windows\System32\cmd.exe
D:\program\All JTC Program>javac Jtc9.java
D:\program\All JTC Program>java Jtc9
m1 in Hello
123456980
10100101
2147483647
-2147483648
33281
5351
0
4666
4666
43983
2147483648
D:\program\All JTC Program>

```

b) **Floating point Literal:** In order to increase the readability from jdk1.7 underscore(_) is introduced. This (_) can be use to increase the readability of a numerical literal. Remember following points while using (_) in numeric literal:

- I. Underscore(_) must not be used for any numeric literal must not start with underscore(_) and not end with (_).
- II. Underscore must not be used just before or just after the decimal.

NOTE: This underscore is just for the developer site which will not reflect in output.

Program 2.6

```

class Hello2{
    int i1=11234;
    int i2=11_234;
    int i3=11__234;
    int i4=11_23_353_3;
    //int i5=_1123;
    //int i6=1123_;
    int i7=0x123_a;
    //int i8=0x_123_A;
    int i9=01_01_01_01_0;
    //int i10=0_X12233A;
    int i11=0_101010;
}

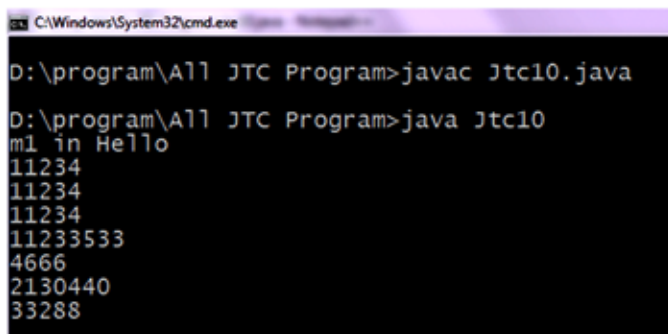
```

```

void m1(){
    System.out.println("m1 in Hello");
    System.out.println(i1);
    System.out.println(i2);
    System.out.println(i3);
    System.out.println(i4);
    //System.out.println(i5);
    //System.out.println(i6);
    System.out.println(i7);
    //System.out.println(i8);
    System.out.println(i9);
    //System.out.println(i10);
    System.out.println(i11);
}
}
public class Jtc10 {
    public static void main(String[] args) {
        Hello2 h1=new Hello2();
        h1.m1();
    }
}

```

Output:



```

C:\Windows\System32\cmd.exe
D:\program\All JTC Program>javac Jtc10.java
D:\program\All JTC Program>java Jtc10
m1 in Hello
11234
11234
11234
11233533
4666
2130440
33288

```

Floating point literal can be represented in two different ways:

1. Floating point representation.
e.g.- float f1=10.11; // not ok

```
float f2=10.11f;
double d1=99.99;
double d2=99.88d;
double d3=99.66D; }
```

// ok

2. Exponential representation.

```
double d1=9.9E+5;
double d2=9.9E-5;
```

Program 2.7

```
class Hai{
    //float f1=11.22; not ok
    float f2=11.22f;
    float f3=11.33F;
    float f4=Float.MAX_VALUE;
    float f5=Float.MIN_VALUE;
    //float f6=11.22;
    float f7=11.22f;
    float f8=99.999f;
    float f9;
    float f10;
    double d1=11.555;
    double d2=526.566D;
    double d3=11.333d;
    double d4=Double.MAX_VALUE;
    double d5=Double.MIN_VALUE;
    double d6=22.22;
    double d7=221.378D;
    double d8=99.99e+5;
    double d9=9.9E-5;
    void show(){
        System.out.println("m1 in Hai");
        System.out.println(f2);
        System.out.println(f3);
        System.out.println(f4);
        System.out.println(f5);
        System.out.println(f7);
        System.out.println(f8);
    }
}
```



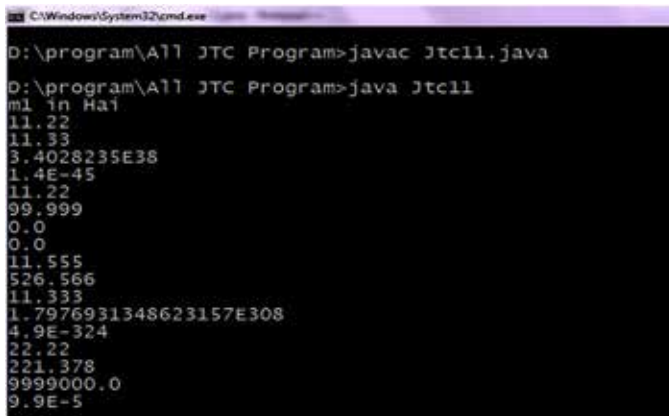
```

        System.out.println(f9);
        System.out.println(f10);
        System.out.println(d1);
        System.out.println(d2);
        System.out.println(d3);
        System.out.println(d4);
        System.out.println(d5);
        System.out.println(d6);
        System.out.println(d7);
        System.out.println(d8);
        System.out.println(d9);
    }
}

public class Jtc11 {
    public static void main(String[] args) {
        Hai h1=new Hai();
        h1.show();
    }
}

```

Output:



```

C:\Windows\System32\cmd.exe
D:\program\All JTC Program>javac Jtc11.java
D:\program\All JTC Program>java Jtc11
m1 in Hai
11.22
11.33
3.4028235E38
1.4E-45
11.22
99.999
0.0
0.0
11.555
526.566
11.333
1.7976931348623157E308
4.9E-324
22.22
221.378
9999000.0
9.9E-5

```

F8 print Null as default (wrapper class) f9 print 0.0 because of primitive float type variable. Here f8 is called as reference variable and default type of a reference variable is null.

In case of f7 assignment allow from jdk 1.5 because of jdk 1.5 auto boxing and un-boxing. It automatically or internally performs the required casting. But if the same assignment you are doing in JDK1.4 then it will give the error.

- c) **Character literal:** Anything which is enclosed with in single quotes will be considered as character literal.

```
e.g.- char ch1='A';
      char ch2='a';
      char ch3='AB' //not ok
      char ch4=' '; //not ok
```

In case of character literal whenever you are enclosing any escape sequence character then it will show its original value for what it has been created.

Program 2.8

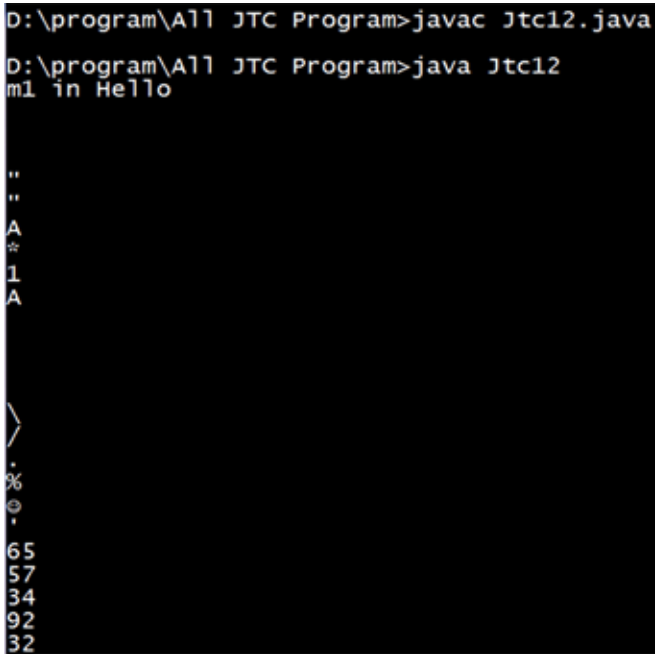
```
class Hello{
    char ch1;
    char ch2=' ';
    char ch3=' ';
    //char ch4=''; not ok
    //char ch5='.'; not ok
    char ch6='''';
    //char ch7=''''; not ok
    char ch8='\u0022';
    char ch9='A';
    //char ch10='AB'; not ok
    char ch11='*';
    //char ch12=' '; not ok
    //char ch13='123'; not ok
    char ch14='1';
    char ch15=65;
    char ch16='\n';
    char ch17='\t';
    char ch18='\r';
    char ch19='\\';
    //char ch20='\'';
    //char ch21='//';
    char ch22='/';
    char ch23=':';
    char ch24='%';
    char ch25='\u0001';
    char ch26='\"';
    //char ch27='\u000ABC';
```

```
int i1='A';
//int i2=A;
//int i3='AB';
int i4='9';
int i5='\u0022';
int i6='\\';
//int i7='\';
int i8=' ' ;
//int i9='123';
void m1(){
    System.out.println("m1 in Hello");
    System.out.println(ch1);
    System.out.println(ch2);
    System.out.println(ch3);
    //System.out.println(ch4);
    //System.out.println(ch5);
    System.out.println(ch6);
    //System.out.println(ch7);
    System.out.println(ch8);
    System.out.println(ch9);
    //System.out.println(ch10);
    System.out.println(ch11);
    //System.out.println(ch12);
    //System.out.println(ch13);
    System.out.println(ch14);
    System.out.println(ch15);
    System.out.println(ch16);
    System.out.println(ch17);
    System.out.println(ch18);
    System.out.println(ch19);
    //System.out.println(ch20);
    //System.out.println(ch21);
    System.out.println(ch22);
    System.out.println(ch23);
    System.out.println(ch24);
    System.out.println(ch25);
    System.out.println(ch26);
    //System.out.println(ch27);
```

```

        System.out.println(i1);
        //System.out.println(i2);
        System.out.println(i4);
        System.out.println(i5);
        System.out.println(i6);
        System.out.println(i8);
    }
}
public class Jtc12 {
    public static void main(String[] args) {
        Hello h1=new Hello();
        h1.m1();
    }
}

```

Output:


```

D:\program\All JTC Program>javac Jtc12.java
D:\program\All JTC Program>java Jtc12
m1 in Hello

"
"
A
*
1
A

\
.
%
@
'
65
57
34
92
32

```

Program 2.9

```

public class Jtc13 {
    public static void main(String[] args) {
        char ch1;
    }
}

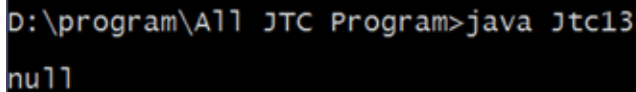
```

```

char ch2=' ';
//char ch3="";
//char ch4='NULL';
//char ch5=null;
//char ch6=null;
Character ch7=null;
//System.out.println(ch1);
System.out.println(ch2);
//System.out.println(ch3);
//System.out.println(ch4);
//System.out.println(ch5);
//System.out.println(ch6);
System.out.println(ch7);
    }
}

```

Output:



```

D:\program\All JTC Program>java Jtc13
null

```

Exception in thread "main" java.lang.Error: Unresolved compilation problem:
 The local variable ch1 may not have been initialized
 System.out.println(ch1);

NOTE

- In java there is no empty character literal but when you are assigning any character literal without giving any space in single quotes then it will be considered as empty character literal and it will give error at compile time.
- Commenting line doesn't mean that the compiler is going to ignore that. It reads that line by converting in corresponding Unicode but again it is not going to be verified from compiler library.

In jtc13 we have written character ch6=null it doesn't mean that it is a character literal but in facts ch6 is a reference variable which is available inside main method & in that explicitly we have assigned default value of any reference variable.

- d) String Literal:** Anything which is enclosed with in double quotes (") will be treated as string literal. In java string is not primitive data type but it is a special class which is available inside java.lang package.

Whatever you are enclosing with in double quotes, it will be printed on your console when you are trying to do so. But in case of escape sequence character it will show its original value for what it has been designed.

Program 2.10

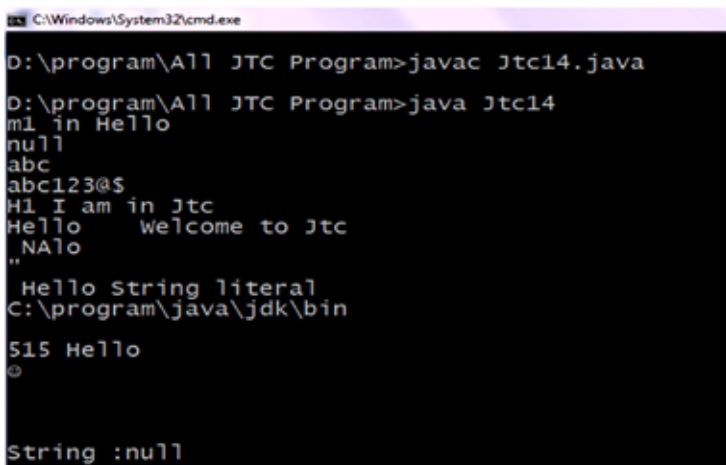
```
class Hello{
    String s1;
    String s2="abc";
    String s3="abc123@$";
    String s4="H1 I am in Jtc";
    String s5="Hello \t Welcome to Jtc";
    String s6="Hello \r NA";
    //String s7="""; //not ok
    //String s8="\u0022"; not ok
    String s9="\\";
    String s10="\u0022 Hello String literal\u0022;
    //String s11="\u0022\u0022; not ok
    String s12="C:\\program\\java\\jdk\\bin";
    //String s13="C:\\program\\java\\jdk\\bin"; not ok
    String s14="";
    String s15=515+" Hello";
    String s16="\u0001";
    String s17="        "; //tab ok
    String s18="        "; //tab & space ok
    //String s19="""Hello""; not ok
    //String s20="""String""; not ok
    String s22=" ";
    String String;
    void m1(){
        System.out.println("m1 in Hello");
        System.out.println(s1);
        System.out.println(s2);
        System.out.println(s3);
        System.out.println(s4);
        System.out.println(s5);
```

```

System.out.println(s6);
//System.out.println(s7);
//System.out.println(s8);
System.out.println(s9);
System.out.println(s10);
//System.out.println(s11);
System.out.println(s12);
//System.out.println(s13);
System.out.println(s14);
System.out.println(s15);
System.out.println(s16);
System.out.println(s17);
System.out.println(s18);
//System.out.println(s19);
//System.out.println(s20);
//System.out.println(s21);
System.out.println(s22);
System.out.println("String :"+String);
}}
public class Jtc14 {
    public static void main(String[] args) {
        Hello h1=new Hello();
        h1.m1();
    }}

```

Output:



```

C:\Windows\System32\cmd.exe
D:\program\All JTC Program>javac Jtc14.java
D:\program\All JTC Program>java Jtc14
m1 in Hello
null
abc
abc123@$
H1 I am in Jtc
Hello welcome to Jtc
NALo
"
Hello String literal
C:\program\java\jdk\bin
515 Hello
String :null

```

e) Boolean Literal: There are mainly two types of Boolean literal:

- I. true
- II. false

This true and false will be specifically assigned to the corresponding Boolean type of variables.

Program 2.11

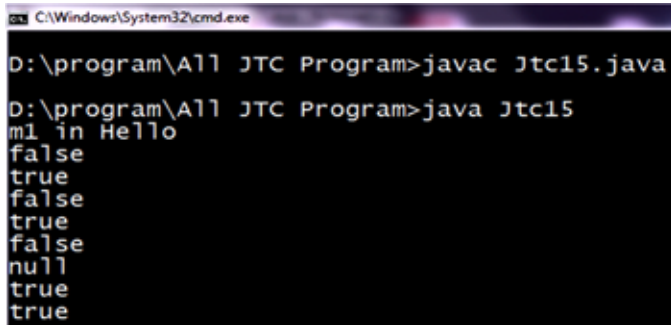
```
class Hello{
    boolean b1;
    boolean b2=true;
    boolean b3=false;
    //boolean b4=TRUE;
    //boolean b5=FALSE;
    //boolean b6=0;
    //boolean b7=1;
    Boolean b8=true;
    Boolean b9=false;
    Boolean b10;
    //Boolean b11=new Boolean();
    Boolean b12=new Boolean(true);
    boolean b13=Boolean.TRUE;
    boolean b14=Boolean.FALSE;
    void m1(){
        System.out.println("m1 in Hello");
        System.out.println(b1);
        System.out.println(b2);
        System.out.println(b3);
        //System.out.println(b4);
        //System.out.println(b5);
        //System.out.println(b6);
        //System.out.println(b7);
        System.out.println(b8);
        System.out.println(b9);
        System.out.println(b10);
        //System.out.println(b11);
        System.out.println(b12);
        System.out.println(b13);
    }
}
```



```

public class Jtc15 {
    public static void main(String[] args) {
        Hello h1=new Hello();
        h1.m1();
    }
}

```

Output:


```

C:\Windows\System32\cmd.exe
D:\program\All JTC Program>javac Jtc15.java
D:\program\All JTC Program>java Jtc15
m1 in Hello
false
true
false
true
false
null
true
true

```

f) Null Literal: null is the default value of any reference variable. This null is not equivalent to zero (0) or anything else.

e.g.: Hello h1;
 System.out.println(h1); //null
 Hello h2=null;
 System.out.println(h2); //null
 Hello h3=Null;
 System.out.println(h3); //not ok

g) Binary Literal: Before jdk 1.7 there was no concept of binary literal in java but from jdk 1.7 binary literal is introduced. While writing the binary literal remember the following points.

Binary literal must start with 0B/0b (Zero B).

Binary Literal representation should be only 0 or 1. If you are writing any number other than binary no's it will give compile time error.

For eg. int i1 = 0b1010101110

 Int i2 = 0B01011101

When you try to print variable which is being assigned by the binary literals then it will be converted to the decimal and then it will print some values.

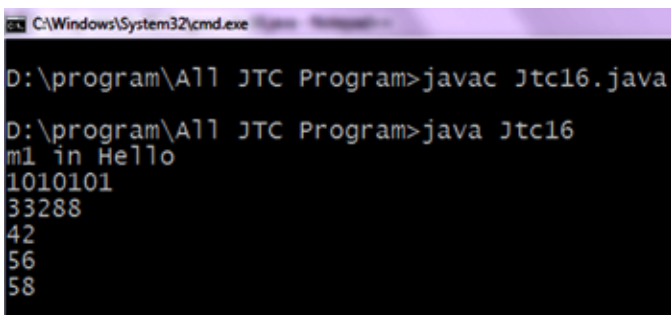
Program 2.12

```

class Hello{
    int i1=1010101;
    int i2=0101010;
    int i3=0b101010;
    int i4=0B111000;
    //int i5=0b01010120101;
    //double d1=0b111.01010;
    //double d2=0B101010.1101;
    //byte b1=0b111111010;
    byte b2=0b111010;
    void m1(){
        System.out.println("m1 in Hello");
        System.out.println(i1);
        System.out.println(i2);
        System.out.println(i3);
        System.out.println(i4);
        //System.out.println(i5);
        //System.out.println(d1);
        //System.out.println(d2);
        //System.out.println(b1);
        System.out.println(b2);
    }
}

public class Jtc16 {
    public static void main(String[] args) {
        Hello h1=new Hello();
        h1.m1();
    }
}

```

Output:


```

C:\Windows\System32\cmd.exe
D:\program\All JTC Program>javac Jtc16.java
D:\program\All JTC Program>java Jtc16
m1 in Hello
1010101
33288
42
56
58

```

Error:

```
E:\> javac Jtc15.java
```

```
Jtc15.java:12: error: no suitable constructor found for Boolean()
b11=new Boolean(); constructor Boolean.Boolean(String) is not
applicable (actual and formal argument lists differ in length)
```

```
E:\> javac Jtc16.java
```

```
Jtc. Java:10:error: possible loss of precision
```

```
Byte b1=0b1111111010
```

```
Required : byte
```

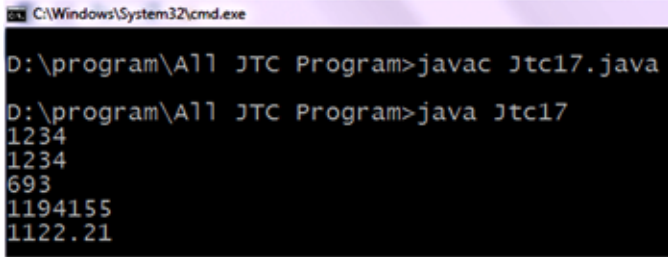
```
Found : int
```

Program 2.13

```
class Hello{
    int i1=1234;
    int i2=1_234;
    int i3=0b10101_10101;
    //int i4=0B_10101011;
    //int i5=0B10101010_;
    int i6=0X1238A_B;
    //double d1=11.22d;
    double d2=11_22.21;
    //double d3=11._272;
    //double d4=111_.2828;
    void m1(){
        System.out.println(i1);
        System.out.println(i2);
        System.out.println(i3);
        //System.out.println(i4);
        //System.out.println(i5);
        System.out.println(i6);
        //System.out.println(d1);
        System.out.println(d2);
        //System.out.println(d3);
        //System.out.println(d4);
    }
}

public class Jtc17 {
    public static void main(String[] args) {
```

```
        Hello h1=new Hello();  
        h1.m1();  
    }  
}
```

Output:

A screenshot of a Windows command prompt window titled "C:\Windows\System32\cmd.exe". The prompt shows the directory "D:\program\All JTC Program". The user enters "javac Jtc17.java" to compile the program. Then, they enter "java Jtc17" to run it. The output of the program is displayed on the following lines: "1234", "1234", "693", "1194155", and "1122.21".

```
C:\Windows\System32\cmd.exe  
D:\program\All JTC Program>javac Jtc17.java  
D:\program\All JTC Program>java Jtc17  
1234  
1234  
693  
1194155  
1122.21
```

Chapter 3

Operator

3.1 What is Operators

Operators are used to perform operation after being used with operands.

3.2 Types of Operators

Various types of Operators are:

1. Arithmetic Operator (+, -, *, /, %)
2. Assignment Operator (=, +=, -=, *=)
3. Relational Operator (==, <=, >=, <>)
4. Logical Operator (&&, ||, !)
5. Bitwise Operator (&, |, ~, <<, >>)
6. Ternary Operator (: ?)
7. Increment/Decrement Operator (++ , --)
8. new Operator
9. instanceof Operator

Operator Precedence

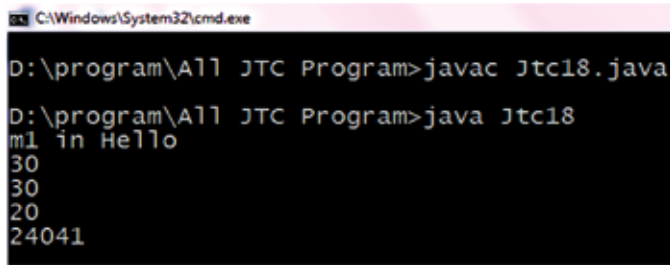
Operators	Precedence
postfix	expr++ expr--
unary	++expr --expr +expr -expr ~ !
multiplicative	* / %
additive	+ -
shift	<< >> >>>
relational	< > <= >= instanceof
equality	== !=
bitwise AND	&
bitwise exclusive OR	^
bitwise inclusive OR	
logical AND	&&
logical OR	
ternary	? :
assignment	= += -= *= /= %= ^= = <<= >>= >>>=

- 1. Arithmetic Operators:** Arithmetic operators are the binary operator which requires two operands. The result of arithmetic operations depends upon type of operand used. By using arithmetic operators we can form the expression.

Program 3.1

```
class Hello{
    byte b1=10;
    byte b2=20;
    int i1=10;
    int i2=20;
    long l1=111112;
    long l2=12929l;
    void m1(){
        System.out.println("m1 in Hello");
        //byte b11=b1+b2;
        //byte b12=i1+i2;
        //byte b13=b1+i1;
        int i11=b1+b2;
        int i12=i1+i2;
        int i13=b1+i1;
        long l11=l1+l2;
        //System.out.println(b11);
        //System.out.println(b12);
        //System.out.println(b13);
        System.out.println(i11);
        System.out.println(i12);
        System.out.println(i13);
        System.out.println(l11);
    }
}

public class Jtc18 {
    public static void main(String[] args) {
        Hello h1=new Hello();
        h1.m1();
    }
}
```

Output:


```

C:\Windows\System32\cmd.exe

D:\program\All JTC Program>javac Jtc18.java

D:\program\All JTC Program>java Jtc18
m1 in Hello
30
30
20
24041

```

Note:

```

final byte b3=10;
final byte b4=20;
final byte b5=127;
final byte b6=10;
final int i3=10;
final int i4=20;
final int i5=2147483647;
final int i6=1234;
void m1(){
byte b14=b3+b4;
//byte b15=b5+b6;
byte b16=i3+i4;
int i14=i3+i4;
int i5=i5+i4;
System.out.println(b14);
System.out.println(b15);
System.out.println(b16);
System.out.println(i14);
System.out.println(i15);
} }

```

+	Byte	short	int	long	float	double	char
Byte							
short							
Int							
Long							
Float							
double							
Char							

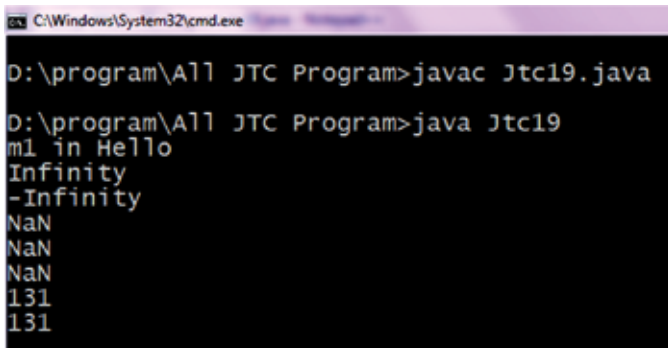
Program 3.2

```

class Hello{
    float f1=Float.POSITIVE_INFINITY;
    float f2=Float.NEGATIVE_INFINITY;
    float f3=Float.NaN;
    float f4=f1+f2;
    float f5=f2+f3;
    char ch1='A';
    char ch2='B';
    int i11=ch1+ch2;
    int ch3=ch1+ch2;
    void m1(){
        System.out.println("m1 in Hello");
        System.out.println(f1);
        System.out.println(f2);
        System.out.println(f3);
        System.out.println(f4);
        System.out.println(f5);
        System.out.println(i11);
        System.out.println(ch3);
    }
}

public class Jtc19 {
    public static void main(String[] args) {
        Hello h1=new Hello();
        h1.m1();
    }
}

```

Output:


```

C:\Windows\System32\cmd.exe
D:\program\All JTC Program>javac Jtc19.java
D:\program\All JTC Program>java Jtc19
m1 in Hello
Infinity
-Infinity
NaN
NaN
NaN
131
131

```


E:\> javac Jtc19.java

Jtc19. Java:10:error: possible loss of precision

Char ch3=ch1+ch2;

Required : char

Found : int

2. **Assignment Operator:** It is used to assign some value to the corresponding destination data type.

Destination = Source

1. byte b1=20;
2. int i1=20;
3. byte b11=b1;
4. byte b12=i1;
5. int i11=b1;
6. int i12=i1;

In case of assign values we should remember the following points:

- a. If the source and destination both are the of same type. Then the assignment is possible.
- b. When destination data type is larger than source data type.

When destination is smaller than Source this assignment is not possible directly but in order to make it possible we need to perform the type casting to the required destination type.

3. **Relational Operator:** Relational operator is used to establish the relation between two expressions or by using this relational operator we can form a relational expression. The result of relational expression always returns some Boolean value that is either true or false.

e.g.: int a=10;
int b=20;
(a>b)

4. **Logical Operator:** Logical operator is used to form the logical expression. Logical expression always gives result in Boolean value either true or false. Logical operation will always be used between two relational expressions i.e. between two Boolean values.

Truth Table

A	B	A&&B	A B	!A
F	F	F	F	T
F	T	F	T	T
T	F	F	T	F
T	T	T	T	F

e.g. - int a=10;
 int b=20;
 (a>b) && (a<b)

5. **Bitwise Operator:** Bitwise operator is used to work on individual bit. There are mainly two types of bit shifting operator. They are:

a. **Left Shift Operator:** This operator is used to work on individual bit i.e. it shifts the bit position towards left side by 1 for specified no. of times. When you are shifting bit position towards left then the value get exactly double of its original one.

Given: int i=16;
 i<<2

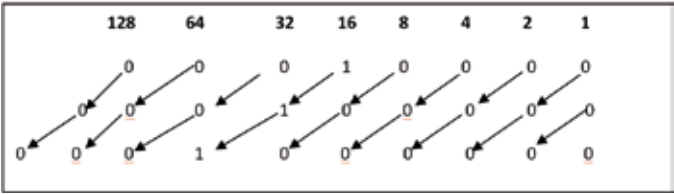


Fig 3.1: Left Shift Operator

b. **Right Shift Operator:** When you are shifting bit position towards right then the individual bit will shift towards right side by 1 for specified number of times. When it is shifting one bit position towards right then the value get exactly half of its original one.

Note: In case of precision part, precision part will be ignoring.
e.g.: int i=12
 i>>3

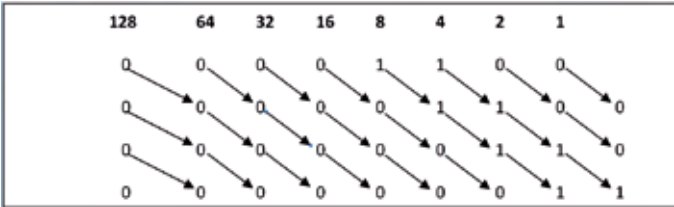


Fig 3.2: Right Shift Operator

Q 1: Find the negation of the number int i=16.

Ans:	128	64	32	16	8	4	2	1
	0	0	0	1	0	0	0	0
	1	1	1	0	1	1	1	1 (1's Compliment)

Negation of no. = (Sum of 1's bit position after 1'compliment) – (Sum of max bit representation)

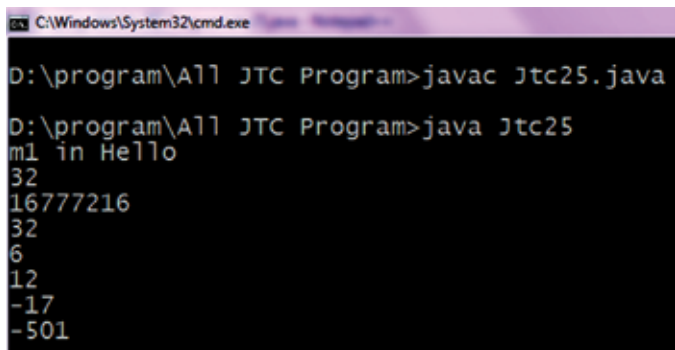
=239-256

=-17

Q 2: Write a Program to find the negation of the number int i=500. (H/W)

```
class Hello{
    int i1=16;
    int i2=12;
    void m1(){
        System.out.println("m1 in Hello");
        System.out.println(i1<<1);
        System.out.println(i1<<20);
        System.out.println(i1<<33);
        System.out.println(i2>>1);
        System.out.println(i2>>32);
        System.out.println(~16);
        System.out.println(~500);
    }
}

public class Jtc25 {
    public static void main(String[] args) {
        Hello h1=new Hello();
        h1.m1();
    }
}
```

Output:


```

C:\Windows\System32\cmd.exe
D:\program\All JTC Program>javac Jtc25.java
D:\program\All JTC Program>java Jtc25
m1 in Hello
32
16777216
32
6
12
-17
-501

```

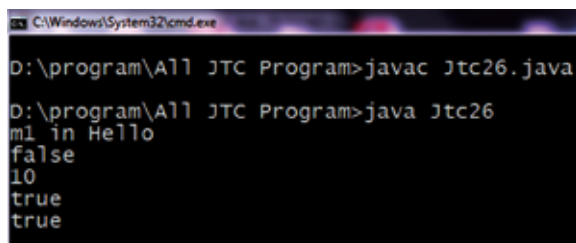
Program 3.3

```

class Hello{
    int i1=10;
    int i2=11;
    int i3=0b101010;
    int i4=0B101010;
    void m1(){
        System.out.println("m1 in Hello");
        System.out.println(i1>i2);
        System.out.println(i1&i2);
        //System.out.println((i1)&&(i2));
        //System.out.println(i3&&i4);
        System.out.println((i1<i2)&&(i2>i1));
        System.out.println((i1<i2)&(i2>i1));
    }
}

public class Jtc26 {
    public static void main(String[] args) {
        Hello h1=new Hello();
        h1.m1();
    }
}

```

Output:


```

C:\Windows\System32\cmd.exe
D:\program\All JTC Program>javac Jtc26.java
D:\program\All JTC Program>java Jtc26
m1 in Hello
false
10
true
true

```

Difference between Logical AND (&&) and Bitwise AND (&):

1. Logical AND can be applied only between relational expression where as bitwise AND can be applied between numeric as well as relational expression also.

e.g.: (i1 < i2) && (i2 > i1) (i1 < i2) & (i2 > i1)

2. When you have applied logical AND(&&) between two relational expression then if the first expression returns true then only second expression will be evaluated or else not i.e. the evaluation of second expression is dependent on the result of first one. Whereas in the case of bitwise AND (&) second expression evaluation is independent of first one.
3. In case of expression evaluation it is always recommendable to use logical and(&&).

```
jtc26: java:8 error: bad operand type for binary operator '&&'
```

```
System.out.println(i1&&i2);
```

```
First type: int
```

```
Second type: int
```

6. **Increment or decrement operator:** This operator is used to increase/decrease the value of the variable by 1. There are mainly two type of increment or decrement operator.

a. Pre Increment operator: The processing of pre increment or decrement operator takes place by following two steps:

- I. Increment or decrement the value of the variable by 1,
- II. Perform the assignment to the corresponding destination.

```
Ex:  -int i1=10;
      int i2=11;
      int i11=++i1;//i1=11;
```

b. Post increment or decrement operator: Processing of post increment/decrement operator, first it perform the assignment and then the value of the variable by one,

```
For ex:-  int i1=10;
           int i2=11;
           int i11=i++;
           int i12=i--;
```

Program 3.4

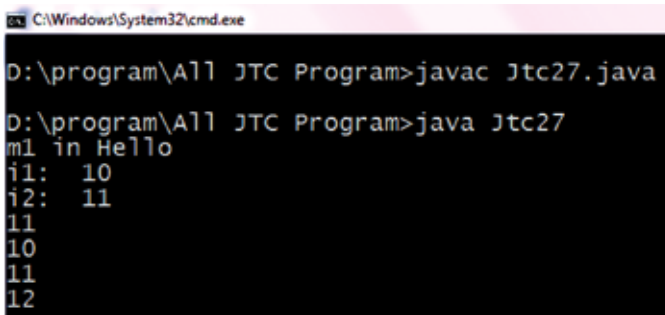
```
class Hello{
int i1=10;
int i2=11;
```

```

void m1(){
    System.out.println("m1 in Hello");
    int i11=++i1;
    int i12=--i1;
    int i13=i2++;
    int i14=i2--;
    System.out.println("i1: "+i1);
    System.out.println("i2: "+i2);
    System.out.println(i11);
    System.out.println(i12);
    System.out.println(i13);
    System.out.println(i14);
}
}

class Jtc27{
public static void main(String[] args) {
    Hello h1=new Hello();
    h1.m1();
}
}

```

Output:


```

C:\Windows\System32\cmd.exe
D:\program\All JTC Program>javac Jtc27.java
D:\program\All JTC Program>java Jtc27
m1 in Hello
i1: 10
i2: 11
11
10
11
12

```

Program 3.5

```

class Hello{
    int i1=10;
    int i2=11;
    int i3=2147483647;
    int i4=Integer.MAX_VALUE;
    int i5=Integer.MIN_VALUE;
}

```

```

int i6=0b1010101;
int i7=0x122A;
float f1=10.22f;
float f2=Float.POSITIVE_INFINITY;
//String s1=Float.POSITIVE_INFINITY;
char ch1='A';
void m1(){
    System.out.println("m1 in Hello");
    int i11=++i1;
    int i12=i2++;
    int i13=++i1+i1++;
    int i14=++i1+i1++;
    int i15=i11--i1;
    //int i16=++i1++;//not ok;
    //int i17=++i1--;//not ok;
    //int i18=++10;
    inti19=++i3;
    //int i20=++IntegerMax_value;
    inti21=++i14;
    //int i22=++i5;
    //int i23=i1+++++i2;
    int i24=(i1++)(++i2);
    int i25=++i6;
    int i26=++i7;
    float f11=++f1;
    float f12=++f2;
    char ch11=++ch1;
    System.out.println(i12);
    System.out.println(i13);
    System.out.println(i14);
    System.out.println(i15);
    System.out.println(i19);
    System.out.println(i21);
    System.out.println(i24);
    System.out.println(i25);
    System.out.println(i26);
    System.out.println(f11);
    System.out.println(f12);
}

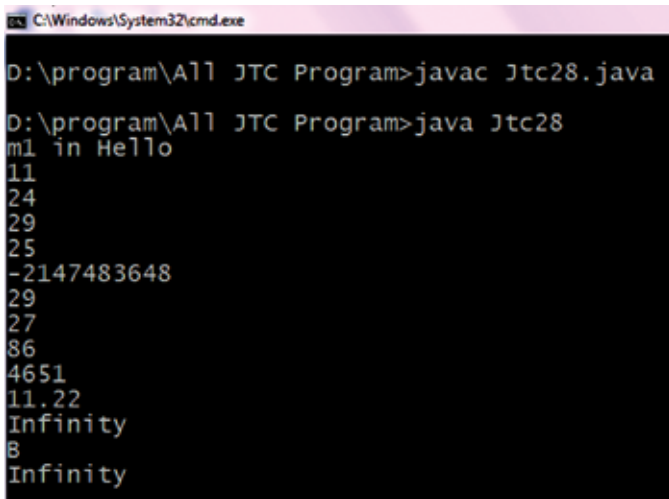
```

```

        System.out.println(ch11);
        System.out.println(f2);
    }
}

public class Jtc28 {
    public static void main(String[] args) {
        Hello h1=new Hello();
        h1.m1();
    }
}

```

Output:


```

C:\Windows\System32\cmd.exe
D:\program\All JTC Program>javac Jtc28.java
D:\program\All JTC Program>java Jtc28
m1 in Hello
11
24
29
25
-2147483648
29
27
86
4651
11.22
Infinity
8
Infinity

```

Note: Increment/ Decrement operator cannot be applied on any value or any final variable.

```

int i16=++i++;
    p1 p2

```

In case of i16 consider p1 is begin process first so after processing of p1 it is giving some value to the p2 that is increment/decrement cannot be applied on any value.

7. new Operator: new operator is used to allocate some memory inside heap or with the help of new operator we can create the object of the class.

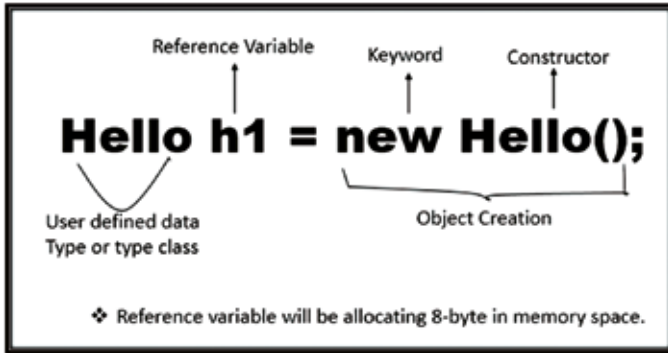


Fig 3.3: new Operator

- 8. Ternary Operator:** Ternary operator is used to check the small condition and based on that it returns the result (it works somehow like if-else)

Syntax:-

Datatype variable_name=(Condition) ? true part : false part

For e.g :-
 int a=10;
 int b=20;
 int c=(a<b)?a:b;

Program 3.6

```
public class Jtc29 {
    public static void main(String[] args) {
        /*int a=10;
        * int b=20;
        * int c=(a<b)?a:b;
        * System.out.println(c);
        */
        int a=Integer.parseInt(args[0]);
        int b=Integer.parseInt(args[1]);
        //int c=(a<b)?a:b;
        //System.out.println(c);
        //String result=(a<b)?"TRUE":"FALSE";
        //System.out.println(result);
        String result1=(a<b)? a:"FALSE";
        System.out.println(result1);
        /*System.out.println("Length of Array :"+args.length);
        System.out.println("\n");
        for(int i=0;i<args.length;i++){
```

```

        System.out.println(args[i]);
    }*/
}
}

```

Error

Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: 0
 at Source.Jtc29.main(Jtc29.java:10)

Exception in thread "main" java.lang.Error: Unresolved compilation problem:
 Type mismatch: cannot convert from int to String
 at Source.Jtc29.main(Jtc29.java:16)

Exception in thread "main" java.lang.NumberFormatException: For input string:
 "a"
 at java.lang.NumberFormatException.forInputString(Unknown Source)
 at java.lang.Integer.parseInt(Unknown Source)
 at java.lang.Integer.parseInt(Unknown Source)
 at Source.Jtc29.main(Jtc29.java:10)

Taking values from the command line:

In main method String args[] as a parameter in which all the values which we have taking from the command line will be stored. According to our requirement we can fetch all that values one by one and can be type casted accordingly.

There is no fixed value which you can take from the command line that is n number of value we can take from the command line.

Integer.parseInt() :- integer is a class in which parseInt method is available which is used to parse or type cast the string to integer type within its compatible range.

Q 3: Can we write nested ternary operator?

Ans: Yes

Example

```

public class Jtc30 {
    static String arg1[];
    public static void main(String[] args) {
        System.out.println(args.length);
        System.out.println(arg1.length);
    }
}

```

Exception in thread "main" java.lang.NullPointerException
at Source.Jtc30.main(Jtc30.java:7)

Whenever requested resource is not available then in that case you are going to get the null pointer exception.

In the case of main method parameter String args[] whatever inputs you are providing from the command line that no. of inputs internally it will be considered by the JVM to the size of Array.

Here String Array size is not being predefined because we don't know how many inputs the user is going to provide from command line.

9. instanceof Operator: instanceof operator is used to check whether that corresponding reference or not. If that reference variable contain then it returns true or else it return false.

Note: In order to apply instanceof operator between two different type there must be the super and sub relation in between. In java java.lang Object class for all types of Classes in java.

Difference between instanceof and isInstance()

InstanceOf	isInstance()
instanceof an operator which can be used to check whether the given object is particular type or not	isInstance() is a method , present in class Class, we can use isInstance() method to checked whether the given object is particular type or not
We know at the type at beginning it is available	We don't know at the type at beginning it is available Dynamically at Runtime.

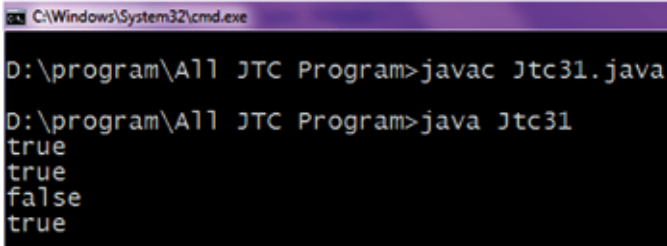
Program 3.7

```
class Hello{
}
class Hai extends Hello{
}
public class Jtc31 {
    public static void main(String[] args) {
        Hello h1=new Hello();
        Hai hai= new Hai();
        System.out.println(h1 instanceof Hello);
        System.out.println(hai instanceof Hai);
    }
}
```

```

        System.out.println(h1 instanceof Hai);
        System.out.println(hai instanceof Hello);
    }
}

```

Output:


```

C:\Windows\System32\cmd.exe
D:\program\All JTC Program>javac Jtc31.java
D:\program\All JTC Program>java Jtc31
true
true
false
true

```

Program 3.8

```

import java.util.ArrayList;
class Hello {
    void m1() {
        System.out.println("m1 in Hello");
    }
}
class Hai {
    void m2() {
        System.out.println("m2 in Hai");
    }
}
public class Jtc32 {
    public static void main(String[] args) {
        Hello h1 = new Hello();
        Hai hai = new Hai();
        if (h1 instanceof Hello) {
            System.out.println("If...");
            h1.m1();
        } else if (hai instanceof Hai) {
            System.out.println("Else...");
            hai.m2();
        }
        System.out.println("ARRAY LIST");
    }
}

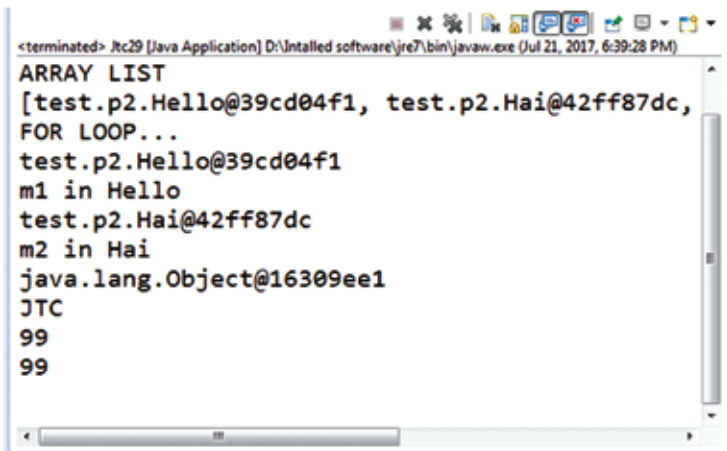
```

```

ArrayList al = new ArrayList();
al.add(new Hello());
al.add(new Hai());
al.add(new Object());
al.add(new String("JTC"));
al.add(99);
System.out.println(al);
System.out.println("FOR LOOP...");
for (Object o : al) {
    System.out.println(o);
    if (o instanceof Hello) {
        Hello h11 = (Hello) o;
        h11.m1();
    } else if (o instanceof Hai) {
        Hai hai1 = (Hai) o;
        hai1.m2();
    } else if (o instanceof Integer) {
        Integer i = (Integer) o;
        System.out.println(i);
    }
}
}
}

```

Output:



```

<terminated> Jtc29 [Java Application] D:\Installed software\jre7\bin\javaw.exe (Jul 21, 2017, 6:39:28 PM)
ARRAY LIST
[test.p2.Hello@39cd04f1, test.p2.Hai@42ff87dc,
FOR LOOP...
test.p2.Hello@39cd04f1
m1 in Hello
test.p2.Hai@42ff87dc
m2 in Hai
java.lang.Object@16309ee1
JTC
99
99

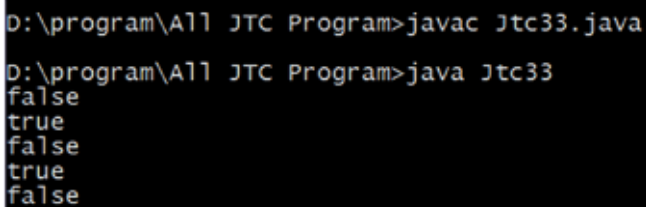
```

Program 3.9

```

class Hello extends Object{
}
public class Jtc33 {
    public static void main(String[] args) {
        Hello h1 = null;
        Object o= new Object();
        String s1= new String("JTC");
        Hello h2 = new Hello();
        System.out.println(h1 instanceof Hello);
        System.out.println(h2 instanceof Object);
        System.out.println(o instanceof Hello);
        System.out.println(s1 instanceof Object);
        //System.out.println(s1 instanceof Hello);
        System.out.println(o instanceof String);
    }
}

```

Output:


```

D:\program\All JTC Program>javac Jtc33.java
D:\program\All JTC Program>java Jtc33
false
true
false
true
false
false

```

3.3 Type Casting

Type Casting is the process of converting one data type to the corresponding compatible destination data type. The type casting mainly can be written in two different types:

1. **Implicit type casting:** When destination is larger than source showing in that case you need not to perform any type casting explicitly it will automatically be converted to the corresponding destination type.

Syntax :

Datatype variable_Name=(Datatype) variable_2;

2. **Explicit type casting:** Explicit type casting is narrowing when destination data type is smaller than source then you need to perform the explicit type casting to corresponding destination type.

Ex. byte b12 = (byte) i1;

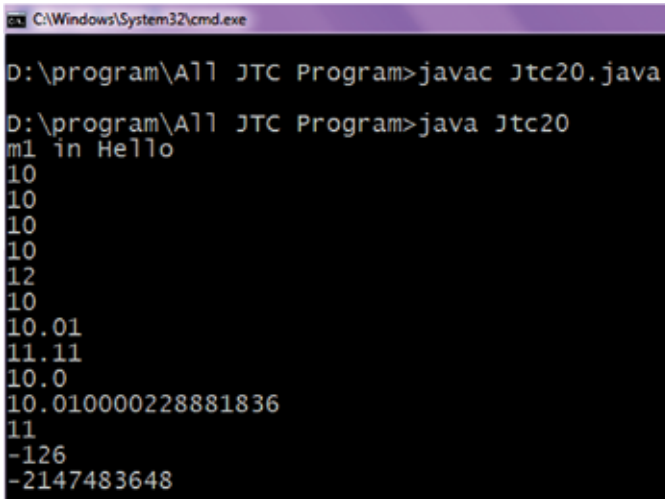
Program 3.10

```
class Hello{
    int i1=10;
    byte b1=10;
    int i2=130;
    long l1=12l;
    long l2=2147483648l;
    float f1=10.01f;
    double d1=11.11;
    void m1(){
        System.out.println("m1 in Hello");
        byte b11=b1;
        byte b12=(byte)i1;
        int i11=b1;
        int i12=i1;
        int i13=(int)l1;
        long l11=i1;
        float f11=f1;
        float f12=(float)d1;
        double d11=i1;
        double d12=f1;
        int i14=(int)d1;
        byte b13= (byte)i2;
        int i15=(int)l2;
        System.out.println(b11);
        System.out.println(b12);
        System.out.println(i11);
        System.out.println(i12);
        System.out.println(i13);
        System.out.println(l11);
        System.out.println(f11);
        System.out.println(f12);
        System.out.println(d11);
        System.out.println(d12);
        System.out.println(i14);
        System.out.println(b13);
    }
}
```

```

        System.out.println(i15);
    }
}
public class Jtc20 {
    public static void main(String[] args) {
        Hello h1=new Hello();
        h1.m1();
    }
}

```

Output:


```

C:\Windows\System32\cmd.exe
D:\program\All JTC Program>javac Jtc20.java
D:\program\All JTC Program>java Jtc20
m1 in Hello
10
10
10
10
10
12
10
10.01
11.11
10.0
10.010000228881836
11
-126
-2147483648

```

Program 3.11

```

class Hello{
    byte b1=10;
    int i1=10;
    byte b12=(byte)(char)(double)(float)(byte)i1;
    byte b13=-1;
    char ch1=(char)(byte)(int)(-1); //ch1=1;
    void m1(){
        System.out.println("m1 in Hello");
        //b1=b1+10;
        b1+=10;
        System.out.println(b12);
    }
}

```

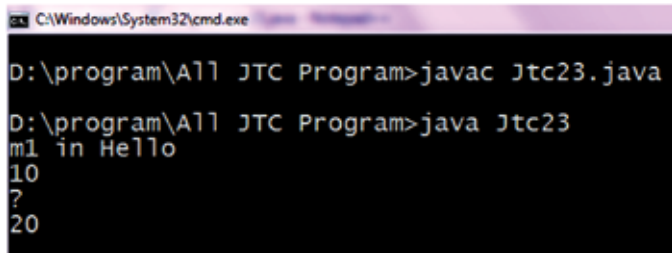


```

        System.out.println(ch1);
        System.out.println(b1);
    }
}
public class Jtc23 {
    public static void main(String[] args) {
        Hello h1=new Hello();
        h1.m1();
    }
}

```

Output:



```

C:\Windows\System32\cmd.exe
D:\program\All JTC Program>javac Jtc23.java
D:\program\All JTC Program>java Jtc23
m1 in Hello
10
?
20

```

Note: In a single line we are performing different types of casting which is called multiple type casting. In case of $b1+=10$; this is called as **compound assignment** in which you need not to perform explicit type casting to the corresponding compatible destination type will be done automatically (Implicitly).

Program 3.12

```

class Hello{
    Byte b1=10;
    Byte b2=10;
    Short s1=11;
    Short s2=22;
    void m1(){
        System.out.println("m1 in Hello");
        Byte b11=(Byte)(b1+b2);
        byte b12=b1+b2;
        Byte b13=b1+s1;
        byte b14=b1+s1;
        short s11=b1+b2;
        System.out.println(b11);
        System.out.println(b12);
    }
}

```

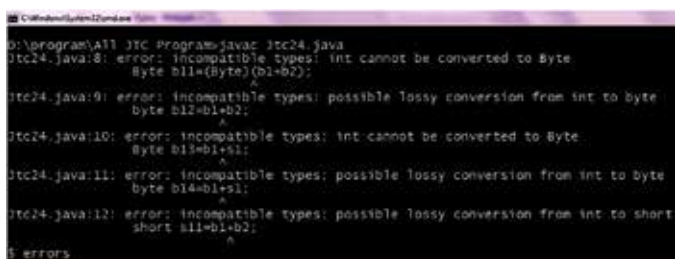
```

        System.out.println(b13);
        System.out.println(b14);
        System.out.println(s11);
    }
}

public class Jtc24 {
    public static void main(String[] args) {
        Hello h1=new Hello();
        h1.m1();
    }
}

```

Output:



```

D:\Program\All Jtc Program>javac Jtc24.java
Jtc24.java:8: error: incompatible types: int cannot be converted to Byte
    Byte b11=(Byte)(b1+b2);
                ^
Jtc24.java:9: error: incompatible types: possible lossy conversion from int to byte
    byte b12=b1+b2;
           ^
Jtc24.java:10: error: incompatible types: int cannot be converted to Byte
    Byte b13=b1+b1;
           ^
Jtc24.java:11: error: incompatible types: possible lossy conversion from int to byte
    byte b14=b1+b1;
           ^
Jtc24.java:12: error: incompatible types: possible lossy conversion from int to short
    short s11=b1+b2;
           ^
6 errors

```

Note

1. Incompatible type's error represents the source and destinations are of not same type or the source and destination data type are not compatible to each other.
2. The Inconvertible data type represents that the source cannot be type cast to corresponding destination type

Chapter 4

Control Statement

4.1 Introduction

Control statement is used to control the flow of execution depending on some specific condition.

4.2 Types of Control Statement

There are mainly following types of control statement.

1. Conditional Control Statement
2. Looping Control Statement

4.2.1 Conditional Control Statement

In case of conditional control statement based on some specific condition execution will be done.

There are mainly following types of conditional control statement.

1. if

- a)

```
if(condition){
    statements;
}
```
- b)

```
if(condition){
    Statement;
}else{
    Statement ;
}
```
- c)

```
if(condition 1){
    Statement;
}else if(condition 2)
    Statement;
}else{
    Statement;
}
```
- d)

```
if(condition){
```

```

        Statement;
    }else{
        if(condition){
            Statement;
        }else{
            Statement ;
        }
    }
}

```

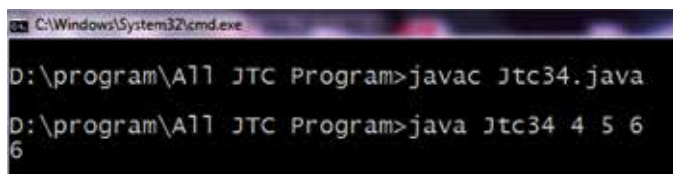
Program 4.1

```

public class Jtc34 {
    public static void main(String[] args) {
        int i1=Integer.parseInt(args[0]);
        int i2=Integer.parseInt(args[1]);
        int i3=Integer.parseInt(args[2]);
        if((i1>i2)&&(i1>i3)){
            System.out.println(i1);
        }else if((i2>i1)&&(i2>i3)){
            System.out.println(i2);
        }else if((i3>i1)&&(i3>i2)){
            System.out.println(i3);
        }
    }
}

```

Output:



```

C:\Windows\System32\cmd.exe
D:\program\All JTC Program>javac Jtc34.java
D:\program\All JTC Program>java Jtc34 4 5 6
6

```

Program 4.2

```

public class Jtc35 {
    public static void main(String[] args) {
        int i=0;
        if(i){

```

```

        System.out.println("if");
    }else{
        System.out.println("else");
    }
}
}

```

Error: java.lang.Error: Unresolved compilation problem:
 Type mismatch: cannot convert from int to boolean at
 Source.Jtc35.main(Jtc35.java:6)

Note: In java 0 and 1 is not equivalent to either true or false. In case of if you must have to provide some relation which returns either true or false.

2. Switch:

```

switch(...){
    case 1: statement 1;
        break;
    case 2: statement 1;
        break;
    case 3: statement 1;
        break;
    default: statement 1;
}

```

It is not necessary to write "break" after each case but if you are not writing break then you will not be able to get the formatted or expected output.

Program 4.3

```

public class Jtc36 {
    public static void main(String[] args) {
        int i = Integer.parseInt(args[0]);
        switch (i) {
            case 3:
                System.out.println("THREE");
                break;
            case 2:
                System.out.println("TWO");
                break;
            case 1:
                System.out.println("ONE");

```

```

        break;
    case 3:
        System.out.println("THREE");
        break;
    default: System.out.println("Default Statement");
        break;
    }
}
}

```

Error : java.lang.Error: Unresolved compilation problems:

Duplicate case

Duplicate case

at Source.Jtc36.main(Jtc36.java:7)

Q1: Can you write default statement anywhere inside switch?

Ans: Yes

Q2: Can I write the default statement more than one?

Ans: No

Q3: Can I duplicate the case labels?

Ans: No

Program 4.4

```

public class Jtc37 {
    public static void main(String[] args) {
        int a = 3;
        switch (a) {
            case 1:
                System.out.println("Case 1");
                break;
            case 2:
                System.out.println("Case 2");
                break;
            case 3:
                System.out.println("Case 3");
                break;
        }
    }
}

```

```

        default: System.out.println("Invalid Case no");
            break;
    }
}
}

```

Output:

Till jdk1.4 byte, short, int and char is allowed inside the switch from Jdk.5 enum is also allowed inside switch. From Jdk 1.7 String is also allowed inside switch.

```

D:\program\All JTC Program>javac Jtc37.java
D:\program\All JTC Program>java Jtc37
Case 3

```

Remember following points while using enum inside switch : The case number's you are using in switch must be the void type of enum if not then it will give the compilation error.

Program 4.5

```

enum MONTH {
    JAN,FEB,MARCH,APR
}
class Hello{
    void m1(char ch){
        System.out.println("char in Switch in m1");
        switch(ch){
            case 'A':
                System.out.println("Case A");
                break;
            case 'B':
                System.out.println("case B");
                break;
            case 'C':
                System.out.println("case C");
                break;
            default:
                System.out.println("invalid case in m1");
            case 'D':
                System.out.println("case D");

```

```

    }
}

void m2(){
    System.out.println("Enum in switch in m2");
    MONTH m[]=MONTH.values();
    for(MONTH m1:m){
        System.out.println(m1+"-----"+m1.ordinal());
    }
    MONTH m1=MONTH.FEB;
    switch(m1){
    case JAN: System.out.println("JAN");
        break;
    case FEB: System.out.println("FEB");
        break;
    case JULY: System.out.println("JULY");
        break;
    default: System.out.println("Invalid Enum Type");
    }
}

void m3(String str){
    System.out.println("String from JDK1.7 in m3");
    switch(str){
    case "AB": System.out.println("AB");
        break;
    case "BC": System.out.println("BC");
        break;
    case "CD": System.out.println("CD");
        break;
    default: System.out.println("Invalid String case");
    }
}

}

class Jtc38{
    public static void main(String arg[]){
        Hello h1=new Hello();
        h1.m1('A');
    }
}

```



```

        h1.m2();
        h1.m3(arg[0]);
    }
}

```

Error: java.lang.Error: Unresolved compilation problem:

JULY cannot be resolved or is not a field

at Source.Heello.m2(Jtc38.java:37)

at Source.Jtc38.main(Jtc38.java:59)

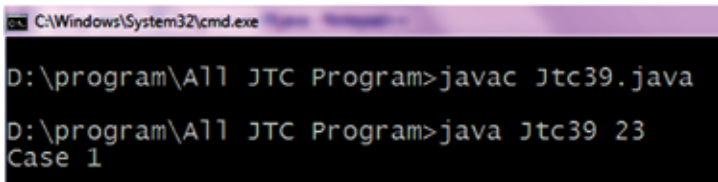
Program 4.6

```

public class Jtc39 {
    public static void main(String[] args) {
        int i=Integer.parseInt(args[0]);
        switch(1){
            case 1:
                System.out.println("Case 1");
                break;
            case 2:
                //break;
            case 3:
                ;;;
                ;
            case 4:
                System.out.println("case 4");
            default:
                System.out.println("Default");
        }
    }
}

```

Output:



```

C:\Windows\System32\cmd.exe
D:\program\All JTC Program>javac Jtc39.java
D:\program\All JTC Program>java Jtc39 23
Case 1

```

Program 4.7

```

public class Jtc40 {
    public static void main(String[] args) {
        char ch='A';
        switch(ch){
            case 1:
                System.out.println("Case 1");
                break;
            case 'A':
                System.out.println("A");
                break;
            default:
                System.out.println("Default");
        }
    }
}

```

Output:


```

C:\Windows\System32\cmd.exe
D:\program\All JTC Program>javac Jtc40.java
D:\program\All JTC Program>java Jtc40
A

```

Program 4.8

```

public class Jtc41 {
    public static void main(String[] args) {
        int i=3;
        final int i1=3;
        final int i2;
        i2=3;
        switch(1){
            case 1:
                System.out.println("ONE");
                break;
            case 2:
                System.out.println("TWO");
                break;
        }
    }
}

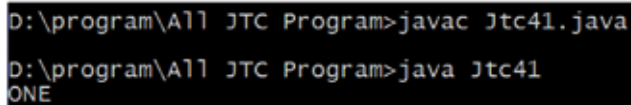
```

```

/*case i:
    System.out.println("THREE");
    break;*/
case i1:
    System.out.println("THREE i1");
    break;
/*case i2:
    System.out.println("THREE i2");
    break;*/
default:
    System.out.println("Default");
}}}

```

Output:



```

D:\program\All JTC Program>javac Jtc41.java
D:\program\All JTC Program>java Jtc41
ONE

```

Note: In case of switch cases must be the constant that is there should not be any chances that the value of the corresponding cases must be variable.

In case of Jtc40 we have provide the case 1 and case 'A', in that the parameter you are passing to the switching must be of same type of the cases enclosed inside the switch.

4.2.2 Looping Control Statement

Looping control statement is used to do some task repeatedly for specified number of times based on some specific condition. There are mainly following types of looping control statement.

1. for Loop

For is used to iterate the loop for the specified number of times based on some specific condition.

Syntax: for(initialization; condition; Increment/ Decrement)

```

{
    Statement;
}

```

Processing of for loop: In processing of for loop first initialization will be done then

the condition check or condition will be evaluated if its satisfied that it returns true, the statement enclosed inside for loop will be processed and then increment or decrement will be done. Then again it evaluates the condition if it returns true then this process will continue until the conditional part does not return false.

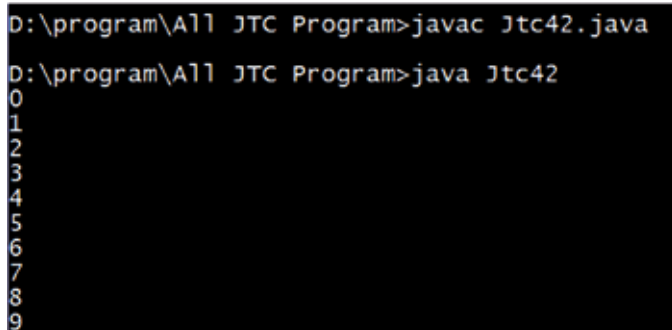
Program 4.9

```
public class Jtc42 {
    public static void main(String[] args) {
        for(int i=0;i<10;i++){
            System.out.println(i);
        }
        int i=0;
        for(;i<10;){
            System.out.println(i);
            i++;
        }
        /*for(System.out.println(i);System.out.println("ABC");i++){
            System.out.println(i);
        }*/
        /*
        for(System.out.println(i); ;i++){
            if(i<10){
                System.out.println(i);
            }
        }*/
        for(int j=0;j<20;j++){
            System.out.println("IN FOR : "+j);
            if(i==3)
                break;
            System.out.println("After Loop");
        }
        for(int j=0;j<20;j++){
            System.out.println("IN FOR : "+j);
            if(i==3)
                System.exit(0);
            System.out.println("After Loop");
        }
    }
}
```

```

        for(int j=0;j<10;j++){
            if(i==3)
            {
                System.out.println("In FOR LOOP If "+i);
                System.exit(0);
            }
            System.out.println(i);
        }
        int k=0;
        for(k=0;k<10;k++)
            i=i++;
        System.out.println(i);
    }
}

```

Output:


```

D:\program\All JTC Program>javac Jtc42.java
D:\program\All JTC Program>java Jtc42
0
1
2
3
4
5
6
7
8
9

```

2. while loop

while loop is descriptive form of for loop. In case of while loop condition will be evaluated first if it returns true then statements enclosed inside will be processed and then increment or decrement will be done. It repeats the same process again and again until the condition is not being unsatisfied or return false.

Syntax: while(Condition){
 Statement;
 Incr/Decr
 }

3. do-while loop

Syntax: do{

```

        Statement;
        Incr/Decr
    } while(Condition);

```

In case of do-while without checking the condition itself it will process the statement once and second iteration onward it checks the condition specified in while. If it returns true then the loop will be continued or else it will come out of the do-while loop.

Q4: In comparison to while and do-while which one we should used mostly in sense of better control.

Ans: It is always recommendable to use while loop in comparison to do-while because we are going to get the better control with the help of while.

Example

```

public class Jtcloop1 {
    public static void main(String[] args) {
        for(byte b=Byte.MIN_VALUE;b<Byte.MAX_VALUE;b++){
            if(b==0x90)
                System.out.println("JTC!");
        }
    }
}

```

```

public class Jtcincr {
    public static void main(String[] args) {
        int j=0;
        for(int i=0;i<100;i++){
            j=j++;
            System.out.println(j);
        }
    }
}

```

```

public class Jtc {
    public static final int END=Integer.MAX_VALUE;
    public static final int START=Integer.MIN_VALUE;
    public static void main(String[] args) {
        int count=0;

```

```

        for(int i=START;i<END;i++){
            count++;
            System.out.println(count);
        }
    }
}

```

```

public class Jtc {
    public static void main(String[] args) {
        int i=0;
        while(-1<<i!=0){
            i++;
            System.out.println(i);
        }
    }
}

```

```

public class Jtc {
    public static void main(String[] args) {
        //Place youe declaration for i here
        while(i==i+1){

        }
    }
}

```

```

public class Jtc{
    public static void main(String[] args) {
        //Place youe declaration for i here
        while(i!=i){
        }
    }
}

```

```

public class Jtc {
    public static void main(String[] args) {
        //Place youe declaration for i here
        while(i!=i+0){
        }}
    }

```

```

public class Jtc {
    public static void main(String[] args) {
        //Place youe declaration for i here
        while(i!=0){
            i>>=1;
        }
    }
}

```

```

public class Jtc {
    public static void main(String[] args) {
        //Place youe declaration for i here
        while(i<=j&& j<=i&&i!=1){
        }    }    }
    }

```

```

public class Jtc {
    public static void main(String[] args) {
        //Place youe declaration for i here
        while(i!=0 && i==i){
        }
    }
}

```

```

public class Jtc {
    public static void main(String[] args) {
        final int START =20000000;
        int count =0;
        for(float f=START;f<START+50;f++)

```



```
        count++;  
        System.out.println(count);  
    }  
}
```

```
public class Jtc {  
    public static void main(String[] args) {  
        int minutes=0;  
        for(int ms=0;ms<60*60*1000;ms++)  
            if(ms%60*1000==0)  
                minutes++;  
        System.out.println(minutes);  
    }  
}
```

Chapter 5

Array

5.1 Introduction

Array is the collection of similar type of data type. Array is static in nature that is one's the size of the array is being defined then it cannot be changed further.

5.2 Types of Array

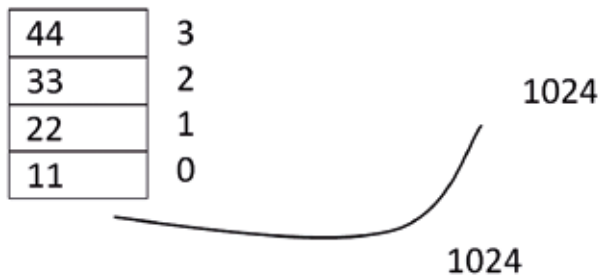
There are mainly two type of array. Array uses indexive type representation to store the element.

1. Static array

Syntax: Datatype array_name[]={val1,val2,.....valn};

for ex: `int a[]={11,22,33,44};`

Note: In java array names are the reference variable and array is the object.



2. Dynamic array / Anonymous Array

Syntax: Datatype array_name[]=new Data type[size];

for ex: `int a[]=new int[4];`

0

0

0

0

`a[3]=303;`

`a[2]=303;`

`a[1]=202;`

`a[0]=101;`

```
int a[]=new int[3]{11,22,33,44};
```

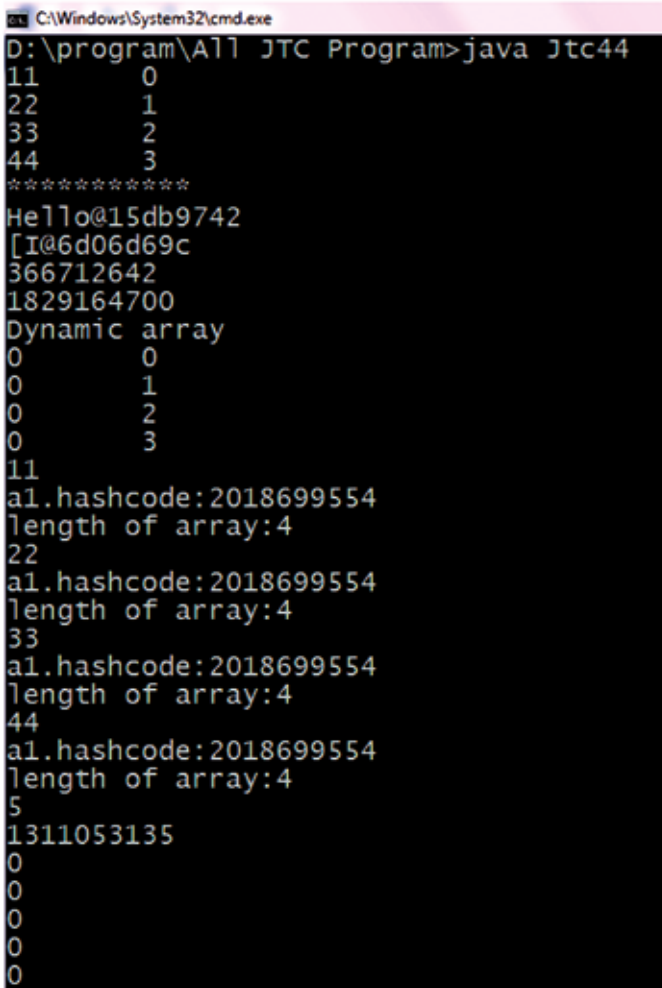
Program 5.1

```
class Hello{
}
class Jtc44 {
public static void main(String[] args) {
    int a[]={11,22,33,44};
    for(int i=0;i<a.length;i++){
        System.out.println(a[i]+"\\t"+i);
    }
    System.out.println("*****");
    Hello h1=new Hello();
    System.out.println(h1);
    System.out.println(a);
    System.out.println(h1.hashCode());
    System.out.println(a.hashCode());
    System.out.println("Dynamic array");
    int a1[]=new int[4];
    for(int i=0;i<a1.length;i++){
        System.out.println(a1[i]+"\\t"+i);
    }
    a1[0]=11;
    a1[1]=22;
    a1[2]=33;
    a1[3]=44;
    for(int i=0;i<a1.length;i++){
        System.out.println(a1[i]);
        System.out.println("a1.hashCode:"+a1.hashCode());
        System.out.println("length of array:"+a1.length);
    }
    //a1.length=10;
    a1=new int[5];
    System.out.println(a1.length);
    System.out.println(a1.hashCode());
    for(int i=0;i<a1.length;i++){
        System.out.println(a1[i]);
    }
}
```

```

    }
}

```

Output:


```

C:\Windows\System32\cmd.exe
D:\program\All JTC Program>java Jtc44
11      0
22      1
33      2
44      3
*****
Hello@15db9742
[I@6d06d69c
366712642
1829164700
Dynamic array
0      0
0      1
0      2
0      3
11
a1.hashCode:2018699554
length of array:4
22
a1.hashCode:2018699554
length of array:4
33
a1.hashCode:2018699554
length of array:4
44
a1.hashCode:2018699554
length of array:4
5
1311053135
0
0
0
0
0
0
0

```

Program 5.2

```

public class Jtc45 {
    public static void main(String[] args) {
        char ch[]=new char[10];
        for(int i=0;i<ch.length;i++){

```

```

        System.out.println(ch[i]);
    }

    System.out.println("*****");
    System.out.println(ch.hashCode());
    System.out.println(ch.length);

    /*for(char ch1='\u0000';ch1<='\u00ff';ch1++){
        System.out.println(ch1);
    }*/

    Hello h1[]=new Hello[5];
    Hello h11=new Hello();
    System.out.println(h11);
    for(int i=0;i<h1.length;i++){
        System.out.println(h1);
    }
    String str1[]={"abc","xyz","mno"};
    for(int i=0;i<str1.length;i++){
        System.out.println(str1[i]+" "+str1[i].length());
    }
}
}}

```

Output:

```

D:\program\All JTC Program>javac Jtc45.java
D:\program\All JTC Program>java Jtc45

*****
366712642
10
Hello@6d06d69c
[LHello;@7852e922
[LHello;@7852e922
[LHello;@7852e922
[LHello;@7852e922
[LHello;@7852e922
abc      3
xyz      3
mno      3

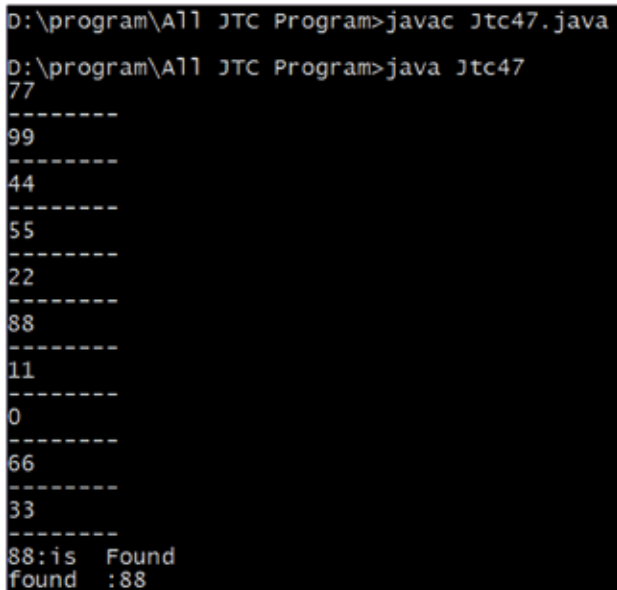
```

Program 5.3

```

class Hello{
    public int hashCode(){
        return 4;
    }
    public String toString(){
        return this.getClass().getName()+":@:"+Integer.toHexString(hashCode());
    }
}
class Hai{
}
public class Jtc46 {
    public static void main(String[] args) {
        Hello h1[]=new Hello[2];
        h1[0]=new Hello();
        //h1[1]=new Hai();
        Object o[]=new Object[2];
        o[0]=new Hello();
        o[1]=new Hai();
        for(int i=0;i<o.length;i++){
            System.out.println(o[i]);
        }
    }
}

```

Output:


```

D:\program\All JTC Program>javac Jtc47.java
D:\program\All JTC Program>java Jtc47
77
-----
99
-----
44
-----
55
-----
22
-----
88
-----
11
-----
0
-----
66
-----
33
-----
88:is Found
Found :88

```

Program 5.4

```

class Hello {
    int a[] = new int[50];
    int b = 0;
    public boolean find (long searchKey){
        int j;
        for(j=0;j<b;j++)
            if(a[j]==searchKey){//found item
                System.out.println(a[j]+":is Found");
                break;
            }
        if(j==b)
            return false;
        else
            return true;
    }
    public void insert(int value)// put element into array
    {
        a[b] = value;
        b++;
    }
    public boolean delete(int value) {
        int j;
        for (j = 0; j<b; j++)
            if (value == a[j])
                break;
        if (j == b)
            return false;
        else {
            for (int k = j; k<b; k++)
                a[k] = a[k + 1];
            b--;
            return true;
        }
    }
    public void display() {
        for (int j = 0; j<b; j++) {

```

```

        System.out.println(a[j]);
        System.out.println("-----");
    }
}

public class Jtc47 {
    public static void main(String[] args) {
        Hello arr=null;
        arr = new Hello();
        arr.insert(77);
        arr.insert(99);
        arr.insert(44);
        arr.insert(55);
        arr.insert(22);
        arr.insert(88);
        arr.insert(11);
        arr.insert(00);
        arr.insert(66);
        arr.insert(33);
        arr.display();// display items
        int SearchKey = 88;// search for item
        if (arr.find(SearchKey)) {
            System.out.println("found :" + SearchKey);
        } else {
            System.out.println("can`t find :" + SearchKey);
            arr.delete(00);
            arr.delete(55);
            arr.delete(99);
            arr.display();
        }
    }
}

```

Output:

```

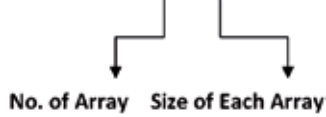
D:\program\All JTC Program>java Jtc46
Hello:@:4
Hai@15db9742

```


5.3 Multidimensional Array

Array of Array is called multidimensional Array.

Syntax: **Datatype array_Name[][]=new Datatype[size][size]**



For e.g.: `int a[][]=new int[3][3]`

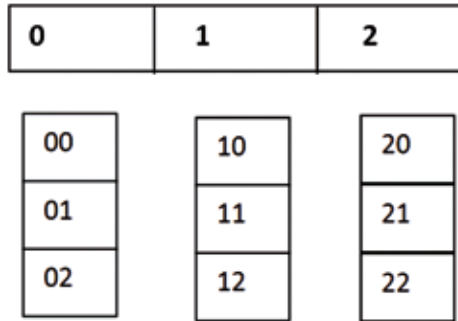
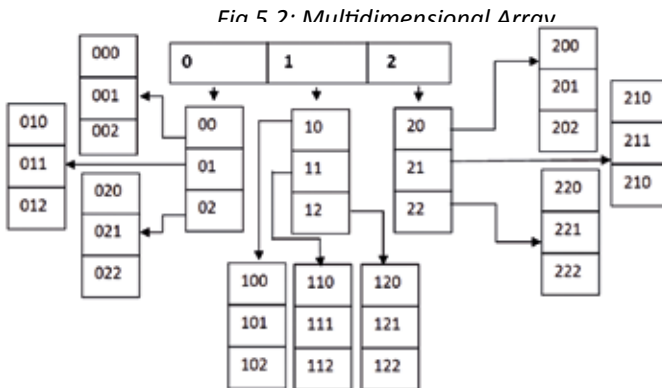


Fig 5.1: Multidimensional Array

Three dimensional Array:

For e.g.: `int a[][][]=new int[3][3][3]`



Static Three Dimension Array

For eg...:- `int a[][]={{11,22,33},{44,55,66}}`

Program 5.5

```
public class Jtc49 {
    public static void main(String arg[]){
```

```

System.out.println("Dynamic Array");
int a[][]=new int[3][3];
for (int i=0;i<a.length;i++) {
    for(int j=0;j<a[i].length;j++){
        System.out.println(a[i][j]);
    }
    System.out.println("\n");
}
a[0][0]=101;
a[0][1]=202;
a[0][2]=303;
a[1][0]=404;
a[1][1]=505;
a[1][2]=606;
a[2][0]=707;
a[2][1]=808;
a[2][2]=909;
for(int i=0;i<a.length;i++){
    for(int j=0;j<a[i].length;j++){
        System.out.println(a[i][j]);
    }
    System.out.println("          ");
}
/* System.out.println("*****");
int b[][][]=new int[][][];
for(int i=0;i<a.length;i++){
    for(int j=0;j<a[i].length;j++){
        for(int k=0;k<a[j].length;k++){
            System.out.println(a[i][j][k]+",");
        }
        System.out.println("\n");
    }
}*/
System.out.println("*****Static Array*****");
double d1[][]={{11.11,22.22},{33.33,44.44},{55.55,66.66}};
for(int i=0;i<d1.length;i++){

```

```

        for(int j=0;j<d1[i].length;j++){
            System.out.println(d1[i][j]+" , ");
        }
        System.out.println("\n");
    }

    System.out.println("*****VAR_ARRAY*****");
    int c[][]=new int[3][];
    c[0]=new int[3];
    c[0][0]=11;
    c[0][1]=22;
    c[0][2]=33;
    c[1]=new int[2];
    c[1][0]=44;
    c[1][1]=55;
    c[2]=new int[1];
    c[2][0]=66;
    for(int i=0;i<c.length;i++){
        for(int j=0;j<c[i].length;j++){
            System.out.println(c[i][j]);
        } } } }

```

Output:

```

D:\program\All JTC Program>java Jtc49
Dynamic Array
0
0
0

0
0
0

0
0
0

101
202
303

404
505
606

707
808
909

```

```

*****Stat ic Arry*****
11.11 ,
22.22 ,

33.33 ,
44.44 ,

55.55 ,
66.66 ,

*****VAR_ARRAY*****
11
22
33
44
55
66

```

NOTE

1. `int a[][]=new int[2][2];`
 2. `int a[][]=new int[0][0];`
 3. `int a[][]=new int[3][];`
 4. `int a[][];`
 5. `a=new int[2][2];`
 6. `int a[][]=new int[2][-1];` //NOT OK
 7. `int a[3][2];` //NOT OK
 8. `int a[][]=new int[2.5][3.56];` //NOT OK
- `int a[][]=new int[3][];`
`a[0]=new int[3];`
`a[1]=new int[2];`
`a[2]=new int[1];`

Program 5.6

```
public class Jtc50 {
    public static void main(String arg[]){
        int a[][]=new int[3][2];
        a[0][0]=99;
        a[0][1]=88;
        a[1][0]=77;
        a[1][1]=66;
        a[2][0]=55;
        a[2][1]=44;
        for(int i=0;i<a.length;i++){
            for(int j=0;j<a[i].length;j++){
                System.out.println(a[i][j]+"\\t"+a[i].hashCode());
            }
        }
        System.out.println("*****\\n");
        a[0]=new int[3];
        for(int i=0;i<a.length;i++){
            for(int j=0;j<a[i].length;j++){
                System.out.println(a[i][j]+"\\t"+a[i].hashCode());
            }
        }
    }
}
```

Output:

```
D:\program\All JTC Program>javac Jtc50.java
D:\program\All JTC Program>java Jtc50
99      366712642
88      366712642
77      1829164700
66      1829164700
55      2018699554
44      2018699554
*****
0       1311053135
0       1311053135
0       1311053135
77      1829164700
66      1829164700
55      2018699554
44      2018699554
```

Chapter 6

OOPS Concept

6.1 Introduction

There are mainly four types of OOPs concept:

1. Abstraction

Abstraction is the process of defining properties and operations of an entity. Here properties relate to variables, operations relate to methods and entity relates to class.

For Example Consider a mobile phone. There is one green and one red button. Green one is there to receive the calls and red one is there to disconnect the calls. End users just know this only and they do not have to worry about what all internal processes have been applied in order to provide that functionality.

NOTE: Abstraction is a design level concept that is not having any equivalent programming implementation.

2. Encapsulation

Encapsulation is the process of defining properties and operations for an entity. In other words encapsulating all the variable methods inside a class is called encapsulation. Defining methods and variables with the proper access modifier in a class is called encapsulation.

```
class Hello{  
private int a=10;  
void m1(){  
SOP(a);  
}}
```



Data Hiding.

The private int a variable cannot be accessed outside the class which we can call it as data hiding. This is also a form of encapsulation.

3. Inheritance

Inheritance is the process of defining a new class from existing class functionality. In order to use existing class to the newly defined class, we need to use the keyword 'extends'.

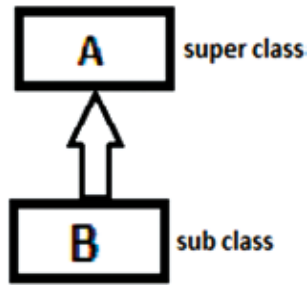


Fig. 6.1: Inheritance

4. Polymorphism

It is the process of one form behaving differently in different cases. There are mainly two types of polymorphism:

- Static polymorphism or (compile time polymorphism)
- Dynamic polymorphism or (runtime polymorphism)

6.2 Class

In java, class is a user defined datatype in which we can write the following different members:

- class
- Variables
- Block
- Method
- Constant
- Class
- Enum
- Interface

Note: class is keyword & `java.lang.Class` is a class

6.3 Object

Object is a blue-print of class. In other words, object contains information about class.

Example

```
class Hello{
```

```
int a=10;
int b=20;
}
```

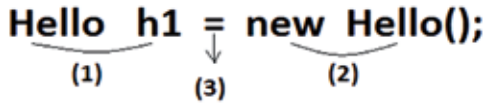


Fig. 6.2: Object

Steps: For reference variable h1, memory will be allocated of 8bytes and to which null will be assigned.



Object will be created in which memory for the class level variable, i.e., instance or static variable will get memory.



Object reference will be assigned to reference variable h1 at the place of null.

NOTE

- For any type of reference variable 8 bytes of memory will be allocated.
- For all types of reference variable default value is null.

6.4 Variable

Variable will be of some specific data type which contains some value that can be changed further in your program. There are mainly two types of variables:

1. **Instance variable:** The variable which is directly defined inside class without any static keyword.

```
class Hello{
    int a=10;    //instance var
    int b=20;
}
```
2. **Static variable:** The variable which is directly defined inside class with static keyword.


```

class Hello{
int a=10; //instance var
int b=20; //instance var
static int c=30; //static var
}

```

Difference between instance variable and static variable

INSTANCE VARIABLE	STATIC VARIABLE
1) Directly defined in the class without any static keyword.	1) Directly defined inside class with static keyword.
2) For instance variable, memory will be allocated at the time of object creation	2) For static variable, memory will be allocated at the time of class loading.
3) Instance variable member cannot be accessed directly inside any static context.	3) Static variable can be accessed directly inside any static context as well as instance context.
4) Instance var can be accessed only with the object reference of corresponding class.	4) Static var can be accessed in the following different ways: a) with the reference var which contains null. <pre>Hello h=null; SOP(h.c);</pre> b) directly by the class name. <pre>SOP(Hello.c);</pre> with the reference var which contains the corresponding class object reference. <pre>Hello h1=new Hello(); SOP(h1.c);</pre>

NOTE

- Static and instance variable both can be declared as final variable.
- Instance and static variable cannot be declared inside any local context.

Program 6.1

```

class hello
{
    int a=10;
    int b=20;
    static int c=30;
    void m1()
    {
        System.out.println("m1 in hello");
        System.out.println("a:\t"+a);
        System.out.println("b:\t"+b);
        System.out.println("c:\t"+c);
    }
    static void m2()
    {
        System.out.println("m2 in hello");
        //System.out.println("a:\t"+a);
        //System.out.println("b:\t"+b);
        System.out.println("c:\t"+c);
    }
}

class Jtc51
{
    public static void main(String arg[])
    {
        //hello h1=new hello();
        //h1.m1();
        //h1.m2();
        System.out.println("instatce variable");
        hello h2=new hello();
        System.out.println(h2.a);
        System.out.println(h2.b);
        System.out.println(h2.c);
        System.out.println("static variable");
        System.out.println("1.....");
        hello h3=new hello();
        System.out.println(h3.a);
        System.out.println(h3.b);
    }
}

```

```

        System.out.println(h3.c);
        System.out.println("2.....");
        //System.out.println(hello.a);
        System.out.println(hello.c);
    }
}

```

Output:

```

C:\Windows\System32\cmd.exe
D:\program\All JTC Program>javac Jtc51.java
D:\program\All JTC Program>java Jtc51
instance variable
10
20
30
static variable
1.....
10
20
30
2.....
30

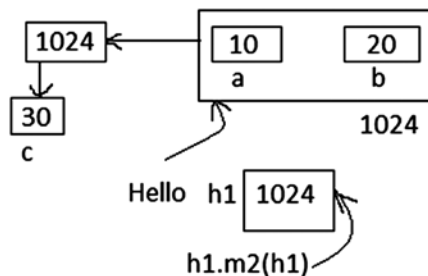
```

Q1 : When exactly a class can be loaded into the main memory?

Ans: Class can be loaded in the main memory in three ways:

At the time of object creation.

- When you are accessing any static member of class with the name of the class.
- When you are accessing any static member of the class with the reference variable which contains null.
- At the time of loading the class first it will check whether the corresponding class has already been loaded into the main memory or not. If the class is not loaded then first it will load it. While loading it allocates memory for the static variables of the class and the static block will be processed.



Q2 : How many times a class can be loaded into the main memory?**Ans:** 1 time**NOTE:** At the time of object creation first instance variable will get memory and then further other processing will be done.**Program 6.2**

```

class Hello{
    int a=10;
    int b=20;
    static int c=30;
    void m1(){
        System.out.println("m1 in hello");
        System.out.println(a);
        System.out.println(b);
        System.out.println(c);
    }
    static void m2(Hello h1){
        System.out.println("M2 in Hello");
        //System.out.println(a);
        //System.out.println(b);
        System.out.println(h1.a);
        System.out.println(h1.b);
        System.out.println(h1.c);
    }
}

class Jtc52{
    public static void main(String args[]){
        Hello h1=new Hello();
        h1.m1();
        h1.m2(h1);
    }
}

```

Output:

```
D:\program\All JTC Program>java Jtc52
m1 in hello
10
20
30
M2 in Hello
10
20
30
```

Program 6.3

```
class Hello{
    int a=10;
    int b=200;
    static int c=30;
    void m1(){
        int aa=11;
        //static int bb=22;
        System.out.println("m1 in Hello");
        System.out.println(a);
        System.out.println(b);
        System.out.println(c);
        System.out.println(aa);
        //System.out.println(bb);
    }
    static void m2(){
        int ab=101;
        //static int bc=202;
        System.out.println("m2 in Hello");
        //System.out.println(a);
        //System.out.println(b);
        System.out.println(c);
        System.out.println(ab);
        //System.out.println(bc);
    }
}

class Jtc53{
    public static void main(String args[]){
        Hello h1=new Hello();
        h1.m1();
    }
}
```

```

        h1.m2();
    }
}

```

Output:

```

D:\program\All JTC Program>javac Jtc53.java
D:\program\All JTC Program>java Jtc53
m1 in Hello
10
200
30
11
m2 in Hello
30
101

```

Note:

- Variable aa inside the method m1 will not be considered as instance variable, it is just a local variable.
- Variable ab declared inside method m2 will not be equivalent to static variable. It is just a local variable.

Program 6.4

```

class Hello{
    void m1(){
        System.out.println("m1 in Hello");
    }
}

class Jtc54{
    int a=10;
    static int b=20;
    Hello h1=new Hello();
    static Hello h2=new Hello();
    public static void main(String ars[]){
        System.out.println("In main method");
        //System.out.println(a);
        System.out.println(b);
        //System.out.println(h1);
        System.out.println(h2);
        //h1.m1();
    }
}

```

```

        h2.m1();
    }
}

```

Output:

```

D:\program\All JTC Program>java Jtc54
In main method
20
Hello@15db9742
m1 in Hello

```

6.5 Blocks

A pair of curly braces defined inside a class is called a block. There are mainly two types of blocks:

1. **Static Block:** A block which is directly defined inside a class with 'static' keyword.

For Example

```

class Hello{
static{
//static block
}}

```

2. **Instance Block:** A block which is directly defined inside the class without any static keyword.

For Example

```

class Hello{
static{
//static block
}
{
//instance block
}}

```

Difference between instance block and static block:

Instance Block	Static Block
1. A block which is directly defined inside any class without static keyword.	1. A block which is directly defined inside any class with static keyword.
2. Inside the instance block we can access instance and static both the members of the class.	2. Inside the static block, we can access only static members of the class directly.
3. The variable defined inside any instance block will not be equivalent to instance variable. It will just be a local variable.	3. The variable defined inside static block will not be equivalent to static variable.
4. Instance block will be processed only at the time of object creation.	4. Static block will be processed at the time of class loading.
5. Instance block will be processed as many times as object of the class is created.	5. Static block will be processed only once that too at the time of class loading.

NOTE

- Block which is defined inside any local context will not be equivalent to instance and static block. That will just be considered as local block.
- The block which is being defined inside any local context will be processed only when that corresponding context is being processed.
- Unlike methods, blocks will be processed automatically, i.e., you cannot call any block explicitly.

Program 6.5

For Instance and static block Processing:

```
class Hello{
    int a=0;
    int b=20;
    static int c=30;
    {
        System.out.println("IB in Hello");
        System.out.println(a);
        System.out.println(b);
        System.out.println(c);
    }
}
```



```

static {
    System.out.println("SB in Hello");
    //System.out.println(a);
    //System.out.println(b);
    System.out.println(c);
}
void m1(){
    System.out.println("M1 in Hello");
    System.out.println(a);
    System.out.println(b);
    System.out.println(c);
}
static void m2(){
    System.out.println("M2 in Hello");
    //System.out.println(a);
    //System.out.println(b);
    System.out.println(c);
}
}
class Jtc55{
    public static void main(String args[]){
        System.out.println(Hello.c);
        System.out.println(Hello.c);
        Hello h1=new Hello();
        Hello h2=new Hello();
    }
}

```

Output:

```

D:\program\All JTC Program>java Jtc55
SB in Hello
30
30
30
30
IB in Hello
0
20
30
IB in Hello
0
20
30

```

Instance And Static Block Processing (with Number of IB and SB).

Program 6.6

```

class Hello{
    int a=10;
    int b=20;
    static int c=30;
    {
        System.out.println("IB1 in Hello");
    }
    static{
        System.out.println("SB1 in Hello");

    }
    {
        System.out.println("IB2 in Hello");
    }
    /*Static{
        System.out.println("Sb2 in Hello");
    }*/
}

class Jtc56{
    int aa=111;
    static int bb=222;
    {
        System.out.println("IB1 in Jtc56");
        System.out.println(aa);
        System.out.println(bb);
    }
    static{
        System.out.println("SB1 in Jtc56");
        //System.out.println(aa);
        System.out.println(bb);
    }
}

```

```

public static void main(String args[]){
    System.out.println("In Main");
    System.out.println(Hello.c);
    Hello h1=new Hello();
    Hello h2=new Hello();
    Jtc56 j56=new Jtc56();
}
}

```

Output:

```

D:\program\All JTC Program>java Jtc56
SB1 in Jtc56
222
In Main
SB1 in Hello
30
IB1 in Hello
IB2 in Hello
IB1 in Hello
IB2 in Hello
IB1 in Jtc56
111
222

```

6.6 Internally by the JVM

- When you are running the program by the name of the class which contains the main method then internally JVM takes that class name and with the help of that class name it calls the main method.
- When JVM is accessing static member of the class then first it will check whether the corresponding class (Jtc56) has been loaded into the main memory or not. If it has not been loaded then first it allocates the memory for static variable of Jtc56 class.

bb 222

- Then if any static block is available, then that block will be processed.
- If you have accessed any instance variable of the class directly then it will give an error because the memory for that instance variable has not been allocated while processing of static block.
- Then after processing the static block it will access the main method. In main method you are accessing static variable of class Hello directly by the name of the class, i.e., Hello.c

- While accessing the static variable it will check whether class Hello has been loaded into the main memory or not. If class Hello has not been loaded into the main memory then it will load that class, while loading the class first allocates the static variable memory and then the static block will be processed. This process will be done only once.
- If you are not creating any object then any instance variable or any instance block will not be processed.

Loading Class Dynamically By using Class.forName(...)

Program 6.7

```
class Hello{
    int a;
    static int b;
    {
        System.out.println("IB in Hello");
    }
    static{
        System.out.println("Sb in Hello");
    }
    void m1(){
        System.out.println("m1 in Hello");
    }
    static void m2(){
        System.out.println("m2 in Hello");
    }
}

class Jtc57{
    {
        System.out.println("IB in Jtc57");
    }
    static {
        System.out.println("Sb in Hello");
        try{
            Jtc57 j57=(Jtc57)Class.forName("Jtc57").newInstance();
        }
        catch(Exception e){
```

```

    }}

    public static void main(String args[])throws Exception{
        Class cls=Class.forName("Hello");
        String str=cls.getName();
        System.out.println(str);
        if(str.equals("Hello")){
            System.out.println("Hello is loaded");
        }
        else{
            System.out.println("Hello is not found");
        }
        Hello h1=(Hello)cls.newInstance();
        System.out.println(Hello.b);
        //Hello h1= new Hello();
    }
}

```

Output:

```

D:\program\All JTC Program>javac Jtc57.java
D:\program\All JTC Program>java Jtc57
Error: Could not find or load main class Jtc57
D:\program\All JTC Program>javac Jtc57.java
D:\program\All JTC Program>java Jtc57
Sb in Hello
IB in Jtc57
Sb in Hello
Hello
Hello is loaded
IB in Hello
0

```

6.7 Class Loader Concept

```

class Hello{
    int a=10;
    static int b=20;
    {
        System.out.println("IB in Hello");
    }
    static{
        System.out.println("SB in Hello");
    }
}

```

```

}
class hai{
    int a1=111;
    static int b1=222;
    {
        System.out.println("Ib in Hai");
    }
    static{
        System.out.println("SB in Hai");
    }
}
class hai1{
    {
        System.out.println("IB in Hai1");
    }
    static{
        System.out.println("SB in Hello");
    }
}
public class Jtc58{
    public static void main(String args[])throws Exception{
        Class cls=Class.forName("Hello");
        Class cls1=Class.forName("Hai",true,cls.getClassloader());
        String str=cls1.getName();
        System.out.println(str);
        System.out.println("*****");
    }
}

```

NOTE

- 'forName()' method is just used to load the class into the main memory.
- When we are defining a block inside another block then the block will not be the instance or static block. That is just a local block.

OR

- The block defined inside any instance or static block inside any method will not be treated as instance or static block. This block will be processed automatically whenever that corresponding context is being processed.
- Inside any block you cannot define any static declaration.

Program 6.8

```

class Hello{
    {
        System.out.println("IB in Hello");
        {
            System.out.println("local bloock in IB");
        }
    }
    static{
        System.out.println("SBI in Hello");
        {
            System.out.println("local block in Sb");
        }
        /*static{
            System.out.println("local in SB1");
        }*/
    }
}
class Jtc59{
    public static void main(String args[])throws Exception{
        Hello h1=(Hello)Class.forName("Hello").newInstance();
    }
}

```

Output:

```

D:\program\All JTC Program>java Jtc59
SBI in Hello
local block in Sb
IB in Hello
local bloock in IB

```

6.8 Methods

Method is of some specific name and for some specific task.

Syntax:

```

<access modifier> <return type> [method name](parameter1, ...)
{
    Stmt1;
    Stmt2;
    //return stmt;
}

```

```
}
```

Example

```
public void m1(){
}
```

- Method must have some return type, i.e., according to the return type of the method return statement must also be available.
- If your method is not returning any kind of value then you can specify method return type as “void”.
- void does not represent zero, null or any value. It just represents that your method does not return any value.
- Method may or may not have any parameters.
- You can define method only directly inside a class, i.e., you cannot define method inside any local context.

There are mainly 2 types of methods:

1. **Instance Method/ Non-Static Method:** The method that is directly defined inside a class without any ‘static’ keyword.

For Example

```
class Hello{
void m1(){
//instance method
}
}
```

2. **Static Method:** A method which is directly defined inside a class with ‘static’ keyword.

For Example

```
class Hello{
static void m1(){
//static method
}
}
```


Difference between instance method and static method

Instance Method	Static Method
1. The method which is directly defined inside a class without any static keyword.	1. The method which is directly defined inside a class with static keyword.
2. Inside instance method we can access instance and static both members of the class directly.	2. Inside the static method you can only access the static members of the class directly.
3. Instance method can be accessed with the reference variable which contains corresponding class object reference.	3. Static method can be accessed in three different ways: with the reference variable which contains null a. Example Hello h1=null; h1.m2(); b. with the reference variable which contains the corresponding class object reference c. directly by the name of the class Example Hello.m2();
4. The variable defined inside any instance method will not be equivalent to instance variable.	4. The variable defined inside any static method will not be equivalent to static variable.

NOTE:

- Instance and static both method of a class can be declared as “final”.
- Both methods have to be called explicitly in order to process, i.e., it will not be processed automatically by the blocks.

Program 6.9

```
class Hello{
    int a=10;
    int b=20;
    static int c=30;
    void m1(){
        System.out.println("m1 in Hello.....instance method");
    }
}
```

```

        System.out.println(a);
        System.out.println(b);
        System.out.println(c);
    }
    static void m2(){
        System.out.println("m2 in Hello.....Static method");
        //System.out.println(a);
        //System.out.println(b);
        System.out.println(c);
    }
}
class Jtc60{
    public static void main(String args[]){
        Hello h1=null;
        h1.m2();
        //h1.m1();
        Hello.m2();
        //Hello.m1();
        Hello h2=new Hello();
        h2.m2();
        h2.m1();
    }
}

```

Output:

```

D:\program\All JTC Program>java Jtc60
m2 in Hello.....Static method
30
m2 in Hello.....Static method
30
m2 in Hello.....Static method
30
m1 in Hello.....instance method
10
20
30

```

NOTE

- We can access method inside another method, but we cannot define a method inside method.
- In order to call any method we should remember:

If you are calling method outside the class and if it is an instance method, then you must have that class object reference, method name, number of parameters (also the sequence of parameters).

Program 6.10

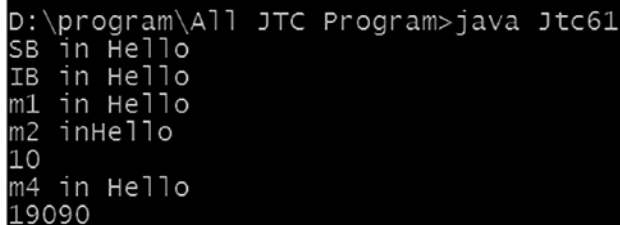
```
class Hello{
    int a=10;
    int b=20;
    static int c=30;
    {
        System.out.println("IB in Hello");
    }
    static{
        System.out.println("SB in Hello");
    }
    int m1(){
        System.out.println("m1 in Hello");
        return 10;
    }
    void m2(){
        System.out.println("m2 inHello");
        //return 10.22;
    }
    static void m3(){
        System.out.println("m3 In hELLO");
    }
    static int m4(int a1){
        System.out.println("m4 in Hello");
        return a1;
    }
}

class Jtc61{
    public static void main(String args[]){
        Hello h1=new Hello();
        int i1=h1.m1();
        h1.m2();
        System.out.println(i1);
    }
}
```

```

        int i2= Hello.m4(19090);
        System.out.println(i2);
    }
}

```

Output:


```

D:\program\All JTC Program>java Jtc61
SB in Hello
IB in Hello
m1 in Hello
m2 inHello
10
m4 in Hello
19090

```

Program 6.11

```

class Hello{
    int a=10;
    static int b=20;
    void m1(){
        System.out.println("m1 in Hello");
    }
    static void m2(){
        System.out.println("m2 in Hello");
    }
    void m11(){
        System.out.println("m11 in Hello");
        System.out.println(a);
        System.out.println(b);
        m1();
        m2();
    }
    static void m22(Hello h1){
        System.out.println("M22 in Hello");
        //System.out.println(a);
        System.out.println(h1.a);
        System.out.println(b);
        //m1();
        m2();
    }
}

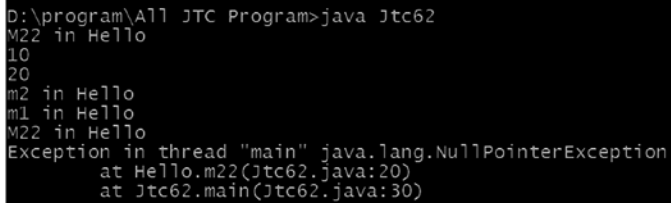
```

```

        h1.m1();
    }
}
class Jtc62{
    public static void main(String args[]){
        Hello.m22(new Hello());
        Hello.m22(null);
    }
}

```

Output:



```

D:\program\All JTC Program>java Jtc62
M22 in Hello
10
20
m2 in Hello
m1 in Hello
M22 in Hello
Exception in thread "main" java.lang.NullPointerException
    at Hello.m22(Jtc62.java:20)
    at Jtc62.main(Jtc62.java:30)

```

Program 6.12

```

class Hai{
    void m1(){
        System.out.println("m1 in Hai");
    }
    static void m2(){
        System.out.println("m2 in hello");
    }
}
class Hello{
    int m11(){
        System.out.println("m11 in Hello");
        return 10;
    }
    Hai m12(){
        System.out.println("M22 inHello");
        return new Hai();
    }
    Hai m13(Hai hai){
        System.out.println("M13 (Hai hai)in Hello");
    }
}

```

```

        //hai.m1();
        hai.m2();
        return m12();
    }
    Object m14(){
        System.out.println("m14 in Hello");
        return 11;
    }
    Object m15(){
        System.out.println("m15 in Hello");
        return new Hai();
    }
    static Hai m16(){
        System.out.println("m16() in Hello");
        return new Hai();
    }
}

class Jtc63{
    public static void main(String args[]){
        Hello h1=new Hello();
        int i=h1.m11();
        System.out.println(i);
        Hai hai=h1.m12();
        hai.m1();
        Hai hai1=h1.m13(null);
        System.out.println(hai.hashCode());
        System.out.println(hai1.hashCode());
        Hai hai2=Hello.m16();
        Hai hai3=Hello.m16();
        System.out.println(hai2.hashCode());
        System.out.println(hai3.hashCode());
    }
}

```

Output:

```

D:\program\All JTC Program>java Jtc63
m11 in Hello
10
M22 inHello
m1 in Hai
M13 (Hai hai)in Hello
m2 in hello
M22 inHello
366712642
1829164700
m16() in Hello
m16() in Hello
2018699554
1311053135

```

Program 6.13

```

class Hai{
    void m1(){
        System.out.println("m1 in Hello");
    }
}
class Hello{
    static Hai hai=new Hai();
    static Hai getHai(){
        System.out.println("getHAi() in Hello");
        return hai;
    }
}
class Jtc64{
    public static void main(String args[]){
        Hai hai=Hello.getHai();
        Hai hai1=Hello.getHai();
        System.out.println(hai.hashCode());
        System.out.println(hai1.hashCode());
    }
}

```

Output:

```
D:\program\All JTC Program>javac Jtc64.java
D:\program\All JTC Program>java Jtc64
getHAi() in Hello
getHAi() in Hello
366712642
366712642
```

NOTE: You cannot specify method parameter as 'void'.

Program 6.14

```
class Hello{
    int m1(){
        System.out.println("M1 in Hello");
        return 10;
    }
    static int m2(){
        System.out.println("m2 in Hello");
        return 20;
    }
    //static int i1=m1();
    static int i2= m2();
    int i3=m1();
    int i4=m2();
    static int m3(){
        System.out.println("m3 in Hello");
        return 100;
    }
    static void m4(int a){
        System.out.println("m4(int a) in Hello");
        System.out.println(a);
    }
}

class Jtc65{
    public static void main(String args[]){
        //Hello h1=new Hello();
        Hello.m2();
        Hello.m4(Hello.m3());
    }
}
```


Output:

```
D:\program\All JTC Program>java Jtc65
m2 in Hello
m2 in Hello
m3 in Hello
m4(int a) in Hello
100
```

Program 6.15

```
class Hello{
    int m1(){
        System.out.println("m1 in Hello");
        return 10;
    }
    int m2(){
        System.out.println("m2 in Hello");
        return 20;
    }
    void m3(){
        System.out.println("m3 in Hello");
    }
}
class Jtc66{
    public static void main(String args[]){
        Hello h1=new Hello();
        System.out.println(h1.m1());
        System.out.println(h1.m2());
        //System.out.println(h1.m3());
        System.out.println(h1.m1()+" "+h1.m2());
    }
}
```

Output:

```
D:\program\All JTC Program>java Jtc66
m1 in Hello
10
m2 in Hello
20
m1 in Hello
m2 in Hello
1020
```

6.9 Volatile Modifier

Volatile is the modifier applicable only for variables but not for classes and methods. If the value of variable keeps on changing such type of variables we have to declare with volatile modifier.

If a variable is declared as volatile then for every thread a separate local copy will be created by the jvm, all intermediate modifications performed by the thread will take place in the local copy instead of master copy. Once the value got finalized before terminating the thread that final value will be updated in master copy.

The main advantage of volatile modifier is that we can resolve data inconsistency problems, but creating and maintaining a separate copy for every thread increases complexity of the Programming and affects the performance of the system. Hence if there is no specific requirement then we do not use volatile modifier and it's almost outdated. Volatile means the value keeps on changing whereas final means the value never changes hence final volatile combination is illegal for variables.

6.10 Method Overloading

Writing more than one method with same name inside a class is called as method overloading.

When you are writing more than one method inside a class with the same name then all these methods will be having different tasks to perform.

While overloading the method remember the following points:

- Method name must be the same.
- Method return type can be anything.
- Method parameter must be different in:
 1. Type of parameters
 2. Sequence of parameters
 3. Number of parameters

Program 6.16

```
class Arith{
    void sum1(){
        System.out.println("sum1 in Arith");
    }
    void sum1(int a){
        System.out.println("sum1(int a) in Arith");
    }
}
```

```

void sum1(byte b1){
    System.out.println("sum1(byte b1) in Arith");
}
void sum1(int a,byte b1){
    System.out.println("sum1(int a,byte b1) in Arith");
}
void sum1(byte b1,int a){
    System.out.println("sum1(byte b1,int a) in Arith");
}
void sum1(int a, int b){
    System.out.println("sum1(int a, int b) in Arith");
}
void sum1(int a,double d){
    System.out.println("sum1(int a,double d) in Arith");
}
void sum1(double d,int a){
    System.out.println("sum1(double d,int a) in Arith");
}
}
public class Jtc67{
    public static void main(String arg[]){
        Arith ar=new Arith();
        ar.sum1();
        ar.sum1(111);
        ar.sum1((byte)111);
        ar.sum1(111,22.22);
        ar.sum1(22.22,111);
    }
}

```

Output:

```

D:\program\All JTC Program>java Jtc67
sum1 in Arith
sum1(int a) in Arith
sum1(byte b1) in Arith
sum1(int a,double d) in Arith
sum1(double d,int a) in Arith

```

Program 6.17

```

class Hai{
}
class Hello{
    void m1(){
        System.out.println("m1 in Hello");
    }
    void m1(Hai hai){
        System.out.println("m1(Hai hai) in Hello");
    }
    void m1(int a[]){
        System.out.println("m1(int a[]) in Hello");
    }
    void m1(Object O){
        System.out.println("m1(m1(Object O) in Hello");
    }
    void m1(String str){
        System.out.println("m1(string str) in Hello");
    }
}
class Jtc68{
    public static void main(String arg[]){
        Hello h1=new Hello();
        h1.m1();
        h1.m1(new Hai());
        //h1.m1(null);
        h1.m1("abc");
    }
}

```

Output:

```

D:\program\All JTC Program>java Jtc68
m1 in Hello
m1(Hai hai) in Hello
m1(string str) in Hello

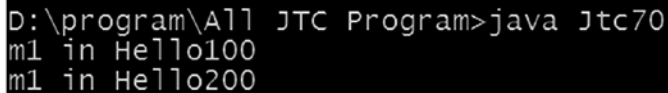
```

Error: reference to m1 is ambiguous, both method m1(int []) in Hello and method m1(String) in Hello match – h1.m1(null);

Program 6.18

```
class Hello{
    void m1(final int a){
        System.out.println("m1 in Hello"+a);
    }
}
class Jtc70{
    public static void main(String arg[]){
        Hello h1=new Hello();
        h1.m1(100);
        h1.m1(200);
    }
}
```

Output:



```
D:\program\All JTC Program>java Jtc70
m1 in Hello100
m1 in Hello200
```

NOTE

- In Jtc70, we have declared the m1 method parameter as a final variable. Whenever we are calling the m1 method by passing some parameter then only memory for variable 'a' will be allocated and immediately after processing the method allocated memory will be destroyed. So, if the memory allocated for the variable 'a' will be destroyed or released then there will be no existence of 'a' the next time we are calling this method of final variable 'a' then new memory allocation will be done.

Program 6.19

```
class Hello{
    void m1(byte b1){
        System.out.println("m1(byte b1) in Hello");
    }
}
```

```

void m1(short s){
    System.out.println("m1(short s) in Hello");
    return;// Empty Return Statement
}
void m1(int i){
    System.out.println("m1(int i) in Hello");
}
/* m1(){
    error: invalid method declration, return type required.
*/
void m1(long l){
    System.out.println("m1(long l) in Hello");
}
void m1(char ch){
    System.out.println("m1(char ch):"+ch);
}
}
class Jtc71{
    public static void main(String arg[]){
        Hello h1=new Hello();
        h1.m1(111);
        h1.m1(65);
        h1.m1((byte)111);
        h1.m1((short)111);
    }
}

```

Output:

```

D:\program\All JTC Program>java Jtc71
m1(int i) in Hello
m1(int i) in Hello
m1(byte b1) in Hello
m1(short s) in Hello

```

NOTE

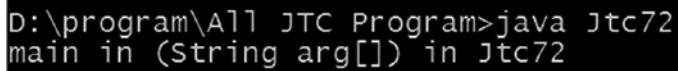
When you are calling m1() by passing some integer then the m1() with the integer parameter will be processed. If the integer parameter is not available then it will check the higher compatible datatype. If it is available then it will be called.

Program 6.20

```

class Hello{
}
class Jtc72{
    public static void main(String arg[]){
        System.out.println("main in (String arg[]) in Jtc72");
    }
    public static void main(String arg){
        System.out.println("main(String arg) in Jtc72");
    }
    public static void main(int a){
        System.out.println("main(int a) in Jtc72");
    }
    public static void main(short s){
        System.out.println("main(short s) in Jtc72");
    }
}

```

Output:


```

D:\program\All JTC Program>java Jtc72
main in (String arg[]) in Jtc72

```

NOTE

- JVM identifies the main method signature with the “public static void main(String arg[])”. When you are running the program, JVM will call this method automatically and rest of the other main methods has to be called explicitly.
- We can write more than one main methods in a class but it must be overloaded main methods.

Q3 : Can I call the main method?

Ans: Yes, we can call the main method according to the requirement.

Program 6.21

```

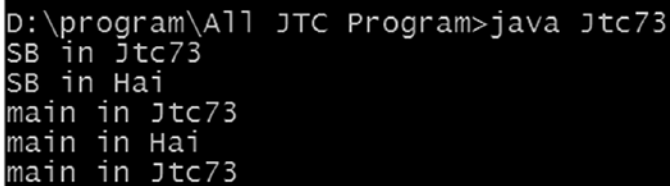
class Hai{
    static{

```

```

        System.out.println("SB in Hai");
        String str[]{"abc"};
        Jtc73.main(str);
    }
    public static void main(String arg[]){
        System.out.println("main in Hai");
    }
}
class Jtc73{
    static{
        System.out.println("SB in Jtc73");
        String str[]{"abcd"};
        Hai.main(str);
    }
    public static void main(String arg[]){
        System.out.println("main in Jtc73");
    }
}

```

Output:**Note**


```

D:\program\All JTC Program>java Jtc73
SB in Jtc73
SB in Hai
main in Jtc73
main in Hai
main in Jtc73

```

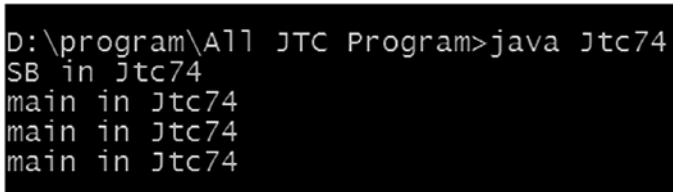
- In case of Jtc73, output depends upon by what name we are running the program. If we are running the program by the name Jtc73 then first JVM writes one code internally (Jtc73.main()). Now in order to call all these main methods Jtc73 class has to be loaded into the main memory. While loading first it will process the static block of Jtc73 and in that itself we have called Hai.main() method. Then further processing will go accordingly.

Program 6.22

```

class Hai {
    public static void main(String arg[]) {
    }
}
class Jtc74 {
    static {
        System.out.println("SB in Jtc74");
        String str[] = { "abc" };
        Jtc74.main(str);
        Jtc74.main(str);
    }
    public static void main(String agr[]) {
        System.out.println("main in Jtc74");
    }
}

```

Output:


```

D:\program\All JTC Program>java Jtc74
SB in Jtc74
main in Jtc74
main in Jtc74
main in Jtc74

```

Program 6.23

```

class Hello {
    {
        System.out.println("IB in Hello");
    }
    static {
        System.out.println("SB in Hello");
    }
    void m1() {
        System.out.println("m1 in Hello");
    }
    static void m2() {
        System.out.println("m2() in Hello");
    }
}

```

```

    }
}
class Jtc75 {
    Hello h1 = new Hello();
    static Jtc75 jtc75 = new Jtc75();
    {
        System.out.println("IB in Jtc75");
        Hello h2 = new Hello();
        h1.m1();
        h2.m1();
    }
    static {
        System.out.println("SB in Jtc75");
    }
    public static void main(String arg[]) {
        System.out.println("main in Jtc75");
        Hello.m2();
    }
}

```

Output:

```

D:\program\All JTC Program>java Jtc75
SB in Hello
IB in Hello
IB in Jtc75
IB in Hello
m1 in Hello
m1 in Hello
SB in Jtc75
main in Jtc75
m2() in Hello

```

Chapter 7

Constructor

7.1 What is Constructor

- It is a special type of method with the name of class.
- Constructor's main task is to initialize instance variables of the class. Even we can initialize static variables but it is not recommendable.
- Constructor must not have any return type like in the case of methods (not even void).
- Constructors may or may not have any parameters.
- Constructors will be processed automatically at the time of object creation; you cannot call a constructor explicitly.
- If the constructor is having any return type then that will be treated as a normal method, i.e., it will be processed only when you are calling that or else it will not be processed automatically at the time of object creation.
- You can write more than one constructors but all that must be different on the following ways:
 1. Number of parameters
 2. Type of parameters
 3. Sequence of parameters

Example

```
class Hello{
Hello(){
// default constructor
}}
```

Program 7.1

```
class Hello {
    int a;
    int b;
    static int c = 30;
    Hello() {
        System.out.println("Default constructor in Hello");
    }
}
```

```

Hello(int a1) {
    System.out.println("1 param constructor in Hello");
    a = a1;
}
Hello(int a1, int b1) {
    System.out.println("2 param constructor in Hello");
    a = a1;
    b = b1;
}
void m1() {
    System.out.println("m1 in Hello");
    System.out.println(a);
    System.out.println(b);
    System.out.println(c);
}
}
class Jtc76 {
    public static void main(String arg[]) {
        Hello h1 = new Hello();
        h1.m1();
        Hello h2 = new Hello(111);
        h2.m1();
        Hello h3 = new Hello(111, 222);
        h3.m1();
    }
}

```

Output:

```

D:\program\All JTC Program>java Jtc76
Default constructor in Hello
m1 in Hello
0
0
30
1 param constructor in Hello
m1 in Hello
111
0
30
2 param constructor in Hello
m1 in Hello
111
222
30

```

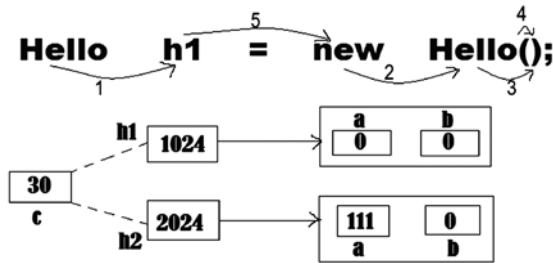


Fig. 7.1: Creating an object

When we are creating the object of a class then: (Fig. 7.1)

- For the reference variable `h1` memory will be allocated and in that null will be assigned.
- Instance variable of a class will get memory inside heap.
- Instance block of a class will be processed.
- Corresponding constructor will be processed.
- At the last, object reference will be assigned to reference variable.

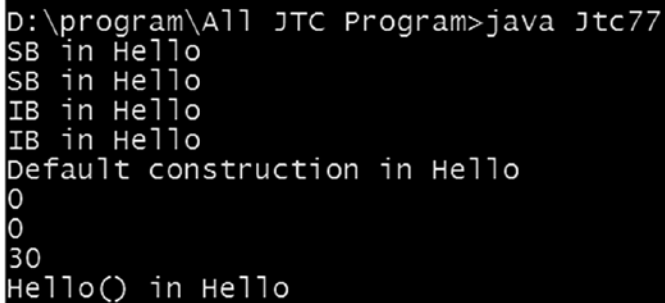
Program 7.2

```
class Hello {
    int a;
    int b;
    static int c = 30;
    {
        System.out.println("IB in Hello");
    }
    static {
        System.out.println("SB in Hello");
    }
    Hello() {
        System.out.println("Default construction in Hello");
    }
    /*
    * Hello(){ }
    */
    void Hello() {
        System.out.println("Hello() in Hello");
    }
}
```

```

Hello(int a1) {
    System.out.println("1 param constructor in Hello");
    a = a1;
}
Hello(int a1, int b1) {
    a = a1;
    b = b1;
}
{
    System.out.println("IB in Hello");
}
static {
    System.out.println("SB in Hello");
}
void show() {
    System.out.println(a);
    System.out.println(b);
    System.out.println(c);
}
}
class Jtc77 {
    public static void main(String arg[]) {
        Hello h1 = new Hello();
        h1.show();
        h1.Hello();
    }
}

```

Output:


```

D:\program\All JTC Program>java Jtc77
SB in Hello
SB in Hello
IB in Hello
IB in Hello
Default construction in Hello
0
0
30
Hello() in Hello

```

Note: Constructor cannot be defined inside any local context.

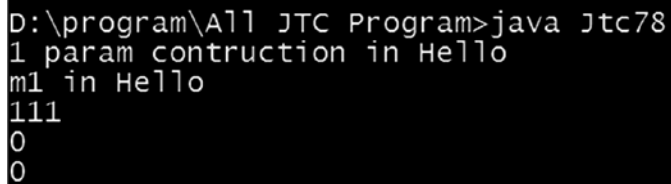
Program 7.3

```

class Hello {
    int a;
    int b;
    static int c;
    Hello(int a1) {
        System.out.println("1 param construction in Hello");
        a = a1;
    }
    void m1() {
        System.out.println("m1 in Hello");
        System.out.println(a);
        System.out.println(b);
        System.out.println(c);
    }
}

class Jtc78 {
    public static void main(String arg[]) {
        // Hello h1=new Hello();
        // h1.m1();
        Hello h2 = new Hello(111);
        h2.m1();
    }
}

```

Output:


```

D:\program\All JTC Program>java Jtc78
1 param construction in Hello
m1 in Hello
111
0
0

```

ERROR

- Java17: constructor Hello in class Hello cannot be applied to given types:
Hello h2=new Hello(111);
Required: no arguments
Found: int
Reason: actual and formal arguments list differ in length

- Constructor Hello in class Hello cannot be applied to given types:
Hello h2=new Hello();
Required: no arguments
Found: int
Reason: actual and formal arguments list differ in length

Note

- When you are not writing any parameterized constructor inside the class then default constructor will internally or automatically be inserted by JVM.
- When you are creating the object of a class then first JVM checks whether that corresponding constructor is available inside that class or not. If you have created object without passing any parameter, i.e., default constructor then JVM checks whether that default constructor is available inside that class or not.
- If default constructor is not available then again it checks whether any parameterized constructor is available or not.
- If parameterized constructor is also not available then internally it inserts default constructor inside the class and process it. If parameterized constructor is available then no default constructor will be inserted by the JVM internally.

Q1 : Can I declare constructor of a class as final?

Ans: No, we cannot declare constructor of a class as final because it will not be allowed to change.

7.2 Final Variable and Static Final Variable

The final variable can be declared in three ways:

- At the time of declaration
- With the help of block
- Constructor

For static final variable should be initialized at:

- At the time of declaration
- Static block

Otherwise it will give error.

Program 7.4

```
class Hello {
    int a;
```



```

final Hello(int a1) {
    System.out.println("1 param constructor in Hello");
    a = a1;
}
void m1() {
    System.out.println("m1 in Hello");
    System.out.println(a);
}
}
class Jtc79 {
    public static void main(String arg[]) {
        Hello h1 = new Hello();
        h1.m1();
        Hello h2 = new Hello(222);
        h2.m1();
    }
}

```

ERROR: modifier final not allowed here:

```
Final Hello(int a1){
```

1 error

Q2 : Can I declare constructor of a class as static?

Ans: No, we cannot declare the constructor of a class as static.

Program 7.5

```

class Hello {
    int a;
    final int b;
    static final int c;
    /*
    {System.out.println("IB in Hello"); b=200; c=300;}
    */
    static {
        System.out.println("SB in Hello");
        c = 100;
    }
    /*

```

```

Hello(){ System.out.println("Default constructor in Hello"); }
*/
Hello(int a1) {
    System.out.println("1 param constructor in Hello");
    b = a1;
}
void m1() {
    System.out.println("m1 in Hello");
    System.out.println(a);
    System.out.println(b);
    System.out.println(c);
}
}
class Jtc80 {
    public static void main(String arg[]) {
        // Hello h1=new Hello();
        // h1.m1();
        Hello h2 = new Hello(999);
        h2.m1();
        Hello h3 = new Hello(888);
        h3.m1();
    }
}

```

Output:

```

D:\program\All JTC Program>javac Jtc80.java
D:\program\All JTC Program>java Jtc80
SB in Hello
1 param constructor in Hello
m1 in Hello
0
999
100
1 param constructor in Hello
m1 in Hello
0
888
100

```

Note: Hello h1=new Hello();
Hello h2=new Hello();

ERROR: variable b might already been initialized. (Fig. 7.2)

b=a1;

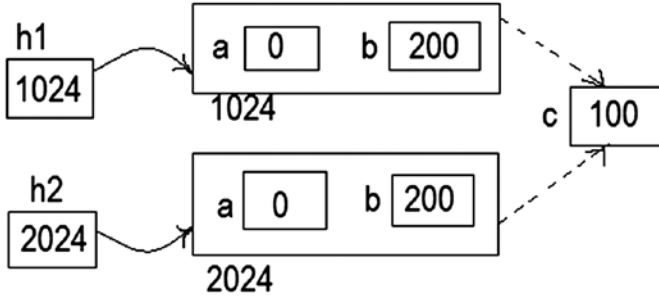


Fig. 7.2: Error

Program 7.6

```
class Hello {
    {
        System.out.println("IB in Hello");
    }
    static {
        System.out.println("SB in Hello");
    }
    Hello() {
        System.out.println("default constructor in Hello");
    }
    Hello(Hello h) {
        System.out.println("1 param constructor in Hello");
    }
    Hello(Hello h1, Hello h2) {
        System.out.println("2 param constructor in Hello");
    }
}

class Jtc81 {
    public static void main(String arg[]) {
        // Hello h1=new Hello();
        // Hello h2=new Hello(null);
        Hello h3 = new Hello(new Hello(new Hello(new Hello(new Hello(),new
        Hello()))));
    }
}
```

Output:

```
D:\program\All JTC Program>java Jtc81
SB in Hello
IB in Hello
default constructor in Hello
IB in Hello
default constructor in Hello
IB in Hello
2 param constructor in Hello
IB in Hello
1 param constructor in Hello
IB in Hello
1 param constructor in Hello
IB in Hello
1 param constructor in Hello
```

Q3 : Can we have the class level variable and local variable with the same name?

Ans: Yes, we can have it.

Q4 : Can we declare the default constructor of a class as private?

Ans: Yes, we can declare the constructor of a class as private. But when you are declaring constructor of a class as private then you will not be able to create the object of that class from outside the class because private constructor of a class will not be accessible from outside the class.

Q5 : What is the main benefit of constructor or use of constructor?

Ans: Constructor is mainly used to initialize the instance variables of the class in different object context because constructor will be processed only at the time of object creation. But just for the initialization each time you are allocating some new memory allocation which is quite an expensive process, i.e., the initialization for the class level variables can be done by the “setter” method also and we can get the initialized value by using the “getter” method.

Note

- In most of the cases it is recommendable to use setters and getters instead of using the constructor which saves lot of memory in compare to constructor.

Chapter 8

JVM Architecture

8.1 Virtual Machine

It is a Software Simulation of a Machine which can Perform Operations Like a Physical Machine.

There are two types of Virtual Machines:

1. Hardware Based OR System Based Virtual Machines

It Provides Several Logical Systems on the Same Computer with Strong Isolation from Each Other.

Examples

- KVM (Kernel Based Virtual Machine) for Linux Systems
- VMware (Virtual Machine ware)
- Xen
- Cloud Computing

The main advantage of Hardware based Virtual Machines is for effective utilization of hardware resources.

2. Software Based OR Application Based OR Process Based Virtual Machines

These Virtual Machines Acts as Runtime Engines to Run a Particular Programming Language Application.

Examples

- JVM Acts as Runtime Engine to Run Java Applications
- PVM (Parrot VM) Acts as Runtime Engine to Run Scripting Languages Like PERL.
- CLR (Common Language Runtime) Acts as Runtime Engine to Run .Net Based Applications.

8.2 JVM

JVM is the Part of JRE. It is Responsible to Load and Run Java Applications. This

Process will be continued for Every Method. Once if any Method Count Reaches Threshold (The Starting Point for a New State) Value, then JIT Compiler Identifies that Method Repeatedly used Method (HOT SPOT).

Immediately JIT Compiler Compiles that Method and Generates the corresponding Native code. Next Time JVM Come Across that Method Call then JVM Directly Use Native Code Executes it Instead of interpreting Once Again. So that Performance of the System will be Improved. The Threshold Count Value varied from JVM to JVM. Some Advanced JIT Compilers will Re-compile generated Native Code if Count Reaches Threshold Value Second Time, So that More optimized Machine Code will be generated. Profiler which is the Part of JIT Compiler is Responsible to Identify HOT SPOTS.

Note

- JVM Interprets Total Program Line by Line at least Once.
- JIT Compilation is Applicable Only for Repeatedly invoked Methods. But Not for Every Method.

Java Native Interface (JNI)

JNI Acts as Bridge (Mediator) between Java Method Calls and corresponding Native Libraries.

Eg: hashCode()

8.3 “this” KEYWORD

- “this” is the instance variable.
- “this” is the instance reference variable which holds the current working object reference.
- “this” is used to point or refer the class level members.
- By using “this” we can refer the instance and static variable both.
- Since “this” is the instance reference variable thus it cannot be accessed from any static context.
- By using “this” we cannot refer local variables of a class.
- “this” is the final instance reference variable.

Program 8.1

```

class Hello {
    int a;
    int b;
    static int c = 30;
    Hello() {
        System.out.println("default constructor in Hello");
    }
    Hello(int a) {
        System.out.println("1 param constructor in Hello");
        this.a = a;
    }
    Hello(int a, int b) {
        System.out.println("2 param constructor in Hello");
        this.a = a;
        this.b = b;
    }
    Hello(int a, int b, int c) {
        System.out.println("3 param constructor in Hello");
        this.a = a;
        this.b = b;
        this.c = c;
    }
    void show() {
        int a = 101;
        int b = 202;
        int c = 303;
        System.out.println("show in Hello");
        System.out.println(a);
        System.out.println(b);
        System.out.println(c);
        System.out.println(this.a);
        System.out.println(this.b);
        System.out.println(this.c);
    }
}

class Jtc82 {
    public static void main(String arg[]) {

```

```

    Hello h1 = new Hello();
    h1.show();
    Hello h2 = new Hello(111);
    h2.show();
    Hello h3 = new Hello(222, 333);
    h3.show();
    Hello h4 = new Hello(444, 555, 666);
    h4.show();
}}

```

Output:

```

D:\program\All JTC Program>java Jtc82
default constructor in Hello
show in Hello
101
202
303
0
0
30
1 param constructor in Hello
show in Hello
101
202
303
111
0
30
2 param constructor in Hello
show in Hello
101
202
303
222
333
30

```

```

3 param constructor in Hello
show in Hello
101
202
303
444
555
666

```

Program 8.2

```

class Hello {
    int a;
    int b;
    static int c = 30;
    {
        int a = 101;
        int b = 202;
        int c = 303;
        System.out.println("IB in Hello");
        System.out.println(this);
        System.out.println(this.hashCode());
    }
}

```



```

        System.out.println(a);
        System.out.println(b);
        System.out.println(c);
        System.out.println(a);
        System.out.println(b);
        System.out.println(c);
    }
    static {
        int a = 101;
        int b = 202;
        int c = 303;
        System.out.println("SB in Hello");
        // System.out.println(this);
        System.out.println(a);
        System.out.println(b);
        System.out.println(c);
        System.out.println(a);
        System.out.println(b);
        System.out.println(c);
    }
    Hello() {
        System.out.println("default constructor in Hello");
        System.out.println(this);
        System.out.println(this.hashCode());
    }
    Hello(int a) {
        System.out.println("1 param constructor in Hello");
        System.out.println(this);
        System.out.println(this.hashCode());
        this.a = a;
    }
    Hello(int a, int b) {
        System.out.println("2 param constructor in Hello");
        System.out.println(this);
        System.out.println(this.hashCode());
        this.a = a;
        this.b = b;
    }
}

```

```

void show() {
    int a = 100;
    int b = 200;
    int c = 300;
    System.out.println("show in Hello");
    System.out.println(this);
    System.out.println(a);
    System.out.println(b);
    System.out.println(c);
    System.out.println(this.a);
    System.out.println(this.b);
    System.out.println(this.c);
}

static void show1() {
    int a = 999;
    int b = 888;
    int c = 777;
    System.out.println("show1 in Hello");
    // System.out.println(a);
    // System.out.println(b);
    System.out.println(c);
    System.out.println(a);
    System.out.println(b);
    System.out.println(c);
    // System.out.println(this.a);
    // System.out.println(this.b);
    // System.out.println(this.c);
}

}

class Jtc83 {
    public static void main(String arg[]) {
        Hello h1 = new Hello();
        System.out.println(h1);
        System.out.println(h1.hashCode());
        h1.show();
        System.out.println("***2nd object creation***");
        Hello h2 = new Hello(999);
        h2.show();
    }
}

```

```

    }
}

```

Output:

```

D:\program\All JTC Program>java Jtc83
SB in Hello
101
202
303
101
202
303
IB in Hello
Hello@15db9742
366712642
101
202
303
101
202
303
default constructor in Hello
Hello@15db9742
366712642
Hello@15db9742
366712642
show in Hello
Hello@15db9742
100
200
300
0
0
30

```

```

***2nd object creation***
IB in Hello
Hello@6d06d69c
1829164700
101
202
303
101
202
303
1 param constructor in Hello
Hello@6d06d69c
1829164700
show in Hello
Hello@6d06d69c
100
200
300
999
0
30

```

Note:

- By using “this”, we can invoke the current class constructor from the constructor itself.
- Remember the following points while invoking current class constructor from the constructor:
 - o “this” must be the first statement inside the constructor.
 - o From a constructor maximum of only one constructor can be invoked.
 - o Cyclic invocation of constructor is not allowed.

Program 8.3

```

class Hello {
    int a;
    int b;
    static int c = 30;
    {
        System.out.println("IB in Hello");
    }
}

```

```

    }
    static {
        System.out.println("SB in Hello");
    }
    Hello(int a, int b, int c) {
        // this(5451);
        System.out.println("3 param constructor in Hello");
        System.out.println(a);
        System.out.println(b);
        System.out.println(c);
    }
    Hello(int a, int b) {
        this(22, 33, 44);
        System.out.println("2 param constructor in Hello");
        this.a = a;
        this.b = b;
    }
    Hello(int a) {
        this(888, 777);
        // this(26723);
        System.out.println("1 param constructor in Hello");
        this.a = a;
    }
    Hello() {
        this(999);
        System.out.println("default constructor in Hello");
    }
    void show() {
        System.out.println("show in Hello");
        System.out.println(a);
        System.out.println(b);
        System.out.println(c);
    }
}

class Jtc84 {
    public static void main(String arg[]) {
        Hello h1 = new Hello();
        h1.show();
    }
}

```

```

    }
}

```

Output:

```

D:\program\All JTC Program>java Jtc84
SB in Hello
IB in Hello
3 param constructor in Hello
22
33
44
2 param constructor in Hello
1 param constructor in Hello
default constructor in Hello
show in Hello
999
777
30

```

ERROR: recursive constructor invocation:

```
Hello(int a,int b){
```

1 error.

Remember following points:

- 'this' must be first statement inside constructor.
- From a constructor you can invoke only one constructor.
- Cyclic invocations of constructor are not allowed.
- You cannot invoke constructor from any method other than constructor.

Try yourself

```

class Hello{
    int a;
    int b;
    static int c=30;
    Hello(){
        System.out.println("Default Constructor");
    }
    Hello(int a){
        System.out.println("1 Parameterised Constructor");
        this.a=a;
    }
}

```

```

Hello(int a,int b){
    System.out.println("2 Parameterised Constructor");
    this.a=a;
    this.b=b;
}
Hello(int a,int b,int c){
    System.out.println("3 Parameterised Constructor");
    this.a=a;
    this.b=b;
    this.c=c;
}
void show(){
    System.out.println("Show method inside Hello Class");
    System.out.println(a);
    System.out.println(b);
    System.out.println(c);
    System.out.println(this.a);
    System.out.println(this.b);
    System.out.println(this.c);
}
}

class Jtc29{
    public static void main(String arg[]){
        Hello h1=new Hello();
        h1.show();
        Hello h2=new Hello(10);
        h2.show();
        Hello h3=new Hello(11,13);
        h3.show();
        Hello h4=new Hello(101,201,301);
        h4.show();
    }
}

class Hello{
    int a;
    int b;
    static int c=30;
    {
        System.out.println("Instance Block");
    }
}

```

```

    }
    static{
        System.out.println("Static Block");
    }
    Hello(int a,int b){
        System.out.println("2 Parameterised Constructor");
        this.a=a;
        this.b=b;
    }
    Hello(int a1){
        this(301,201);
        System.out.println("1 Parameterised Constructor");
        this.a=a;
    }
    Hello(){
        this(1010);
        System.out.println("Default Constructor");
    }
    void show(){
        System.out.println("Show method inside Hello Class");
        System.out.println(a);
        System.out.println(b);
        System.out.println(c);
    }
}

class Jtc30{
    public static void main(String arg[]){
        Hello h1=new Hello();
        h1.show();
        //Hello h2=new Hello(10);
        //h2.show();
        //Hello h3=new Hello(11,13);
        //h3.show();
    }
}

class Hello{
    int a;
    int b;
    static int c=30;
    {

```

```

    System.out.println("Instance Block");
    System.out.println(this);
}
static{
    System.out.println("Static Block");
    //System.out.println(this);
}
Hello(){
    System.out.println("Default Constructor");
}
Hello(int a1){
    System.out.println("1 Parameterised Constructor");
}
Hello(int a,int b){
    System.out.println("2 Parameterised Constructor");
}
void show(){
    System.out.println("Show method inside Hello Class");
    System.out.println(this);
}
}

class Jtc31{
public static void main(String arg[]){
    Hello h1=new Hello();
    System.out.println("-----");
    System.out.println(h1);
    h1.show();
    Hello hh=new Hello();
    System.out.println("-----");
    Hello h2=new Hello(111);
    System.out.println(h2);
    System.out.println("-----");
    //h2.show();
    //Hello h3=new Hello(11,13);
    //h3.show();
}
}

class Hello{

```



```

int a;
int b;
static int c=30;
{
    System.out.println("Instance Block");
    System.out.println(this);
}
static{
    System.out.println("Static Block");
    //System.out.println(this);
}
Hello(){
    System.out.println("Default Constructor");
}
Hello(int a){
    System.out.println("1 Parameterised Constructor");
}
Hello(int a,int b){
    System.out.println("2 Parameterised Constructor");
}
Hello(int...a){
    System.out.println("Var-Arg Constructor Invoke");
    for(int i=0;i<a.length;i++){
        System.out.println(i+" : "+a[i]);
    }
}
void show(){
    System.out.println("Show method inside Hello Class");
    System.out.println(this);
}
}

class Jtc32{
public static void main(String arg[]){
    Hello h1=new Hello();
    System.out.println("-----");
    System.out.println(h1);
    h1.show();
    Hello hh=new Hello();
}
}

```

```

        System.out.println("-----");
        Hello h2=new Hello(111);
        System.out.println(h2);
        System.out.println("-----");
        //h2.show();
        Hello h3=new Hello(11,13,14,78,45);
        System.out.println("-----");
        Hello h4=new Hello(213,234,3565,555);
        System.out.println("-----");
        //h3.show();
    }
}

class Hello{
    int a;
    int b;
    static int c=30;
    Hello(){
        System.out.println("Default constructor in Hello");
    }
    Hello(int a){
        System.out.println("1 parameter constructor in Hello");
        this.a=a;
    }
    Hello(int a,int b){
        System.out.println("2 parameter constructor in Hello");
        this.a=a;
        this.b=b;
    }
    Hello(int a,int b,int c){
        System.out.println("3 parameter constructor in Hello");
        this.a=a;
        this.b=b;
        this.c=c;
    }
    void show(){
        int a=101;
        int b=202;
        int c=303;
    }
}

```

```

System.out.println("show in Hello");
System.out.println(a);
System.out.println(b);
System.out.println(c);
System.out.println(this.a);
System.out.println(this.b);
System.out.println(this.c);
}
}
class Jtc33{
public static void main(String arg[]){
Hello h1=new Hello();
h1.show();
Hello h2=new Hello(444);
h2.show();
Hello h3=new Hello(111,222);
h3.show();
Hello h4=new Hello(777,888,999);
h4.show();
}
}
class Hello{
int a;
int b;
static int c=30;
{
int a=111;
int b=222;
int c=333;
System.out.println("IB in Hello");
System.out.println(this);
System.out.println(this.hashCode());
System.out.println(a);
System.out.println(b);
System.out.println(c);
System.out.println(a);
System.out.println(b);
System.out.println(c);
}
}

```

```

}
static{
int a=101;
int b=202;
int c=303;
System.out.println("SB in Hello");
//System.out.println(this);
System.out.println(a);
System.out.println(b);
System.out.println(c);
System.out.println(a);
System.out.println(b);
System.out.println(c);
}
Hello(){
System.out.println("Default constructor in Hello");
System.out.println(this);
System.out.println(this.hashCode());
}
Hello(int a){
System.out.println("1 parameter constructor in Hello");
System.out.println(this);
System.out.println(this.hashCode());
this.a=a;
}
Hello(int a,int b){
System.out.println("2 parameter constructor in Hello");
System.out.println(this);
System.out.println(this.hashCode());
this.a=a;
this.b=b;
}
void show(){
int a=10;
int b=20;
int c=30;
System.out.println("show in Hello");
System.out.println(this);

```

```

System.out.println(a);
System.out.println(b);
System.out.println(c);
System.out.println(this.a);
System.out.println(this.b);
System.out.println(this.c);
}
static void show1(){
int a=110;
int b=220;
int c=330;
System.out.println("show1 in Hello");
//System.out.println(this);
//System.out.println(a);
//System.out.println(b);
System.out.println(c);
System.out.println(a);
System.out.println(b);
System.out.println(c);
//System.out.println(this.a);
//System.out.println(this.b);
//System.out.println(this.c);
}
}
class Jtc34{
public static void main(Strin arg[]){
Hello h1=new Hello();
System.out.println(h1);
System.out.println(h1.hashCode());
h1.show();
System.out.println("***** 2nd Object Creation*****");
Hello h2=new Hello(999);
h2.show();
}
}

```

Note

- When you are creating the object of the class while processing the corresponding constructor internally JVM checks whether any “this” or “super” statement is available or not.
- If any “this” statement is not available then it will process that constructor but if any “this” statement is available then first it will try to process that constructor. After processing that constructor it will come to the current class constructor.
- From a constructor when we are using two this statements then it will give error at compile time that in order to call the constructor “this” must be the first statement inside the constructor.
- In case of Jtc84, when we are creating the object on the basis of default constructor then first it will try to process default constructor of the class Hello.
- While processing the default constructor it is getting one statement “this” with one parameter so that control will be immediately transferred to the one parameterized constructor.
- While processing one parameterized constructor it is getting “this” statement with two parameters so before processing the one parameterized constructor control will be transferred to the two parameterized constructor and then similarly to the three parameterized constructor.
- While processing it first the three parameterized constructor will be processed then 2 then 1 and at last default will be processed.

Program 8.4

```
class Student {
    int sid;
    String name;
    double fee;
    static Student s1 = null;
    private Student() {
    }
    static {
        s1 = new Student();
    }
    public static Student getStudent() {
        return s1;
    }
}
```

```

public void setSid(int sid) {
    this.sid = sid;
}
public void setName(String name) {
    this.name = name;
}
public void setFee(double fee) {
    this.fee = fee;
}
public int getSid() {
    return sid;
}
public String getName() {
    return name;
}
public double getFee() {
    return fee;
}
/*
Student(){System.out.println("default constructor in Hello");}
*/
/*
Student(int sid,String name,double fee){this.sid=sid; this.name=name;
* this.fee=fee; }
*/
public String toString() {
    return "Name \t:" + name + "\n ID \t:" + sid + "\n Fee \t:" + fee;
}
/*
* void display(){ System.out.println("displaying details of student");
* System.out.println(sid); System.out.println(name);
* System.out.println(fee); }
*/
}

```

```

public class Jtc85 {
    public static void main(String arg[]) {
        // Student s1=new Student();
        // s1.display();
        /*
        System.out.println(s1); System.out.println(s1.toString); Student
        * s2=new Student(101,"som",9000.00); System.out.println(s2); Student
        * s3=new Student(202,"rai",8888.99); System.out.println(s3);
        */
        Student s11 = null;
        s11 = Student.getStudent();
        System.out.println(s11);
        s11.setName("jtcrai");
        s11.setSid(101);
        s11.setFee(9999.99);
        System.out.println(s11);
        System.out.println(s11.getName());
        Student s12 = Student.getStudent();
        System.out.println(s12);
        System.out.println(s11.equals(s12));
        System.out.println(s11 == s12);
        System.out.println(s11.hashCode());
        System.out.println(s12.hashCode());
    }
}

```

Output:

```

D:\program\All JTC Program>java Jtc85
Name      :null
ID        :0
Fee       :0.0
Name      :jtcrai
ID        :101
Fee       :9999.99
jtcrai
Name      :jtcrai
ID        :101
Fee       :9999.99
true
true
366712642
366712642

```


8.4 VAR-ARG Method

- VAR-ARG Method concept is introduced from JDK1.5
- VAR-ARG refers to variable argument.
- VAR-ARG is implemented on the basis of single dimensional array.
- While writing the VAR-ARG method remember the following:
 - o VAR-ARG must be the last parameter of the method.
 - o A method cannot contain more than one VAR-ARG.
 - o VAR-ARG must not be the first parameter.
 - o While calling the VAR-ARG method, first preference will be given to the VAR-ARG method.
 - o If the exactly matching method parameter is not available then reference will be given to the VAR-ARG method.
 - o In case of method without any parameter if it is not available then also reference will be given to the VAR-ARG method only.

Syntax

- 1) `void m1(int... a){`
 `//OK`
 `}`
- 2) `void m1(int ...a){`
 `//OK`
 `}`
- 3) `void m1(int ... a){`
 `//OK`
 `}`
- 4) `void m1(int...a){`
 `//OK`
 `}`
- 5) `void m1(int .._.a){`
 `//NOT OK`
 `}`
- 6) `void m1(int ...a,double d){`
 `//NOT OK`
 `}`
- 7) `void m1(double d, int ...a){`
 `//OK`
 `}`
- 8) `void m1(double...d,int...a){`

```
//NOT OK
}
```

Program 8.5

```
class Hello {
    int a = 10;
    int b = 20;
    void m1() {
        System.out.println("m1 in Hello");
    }
    void m1(int a) {
        System.out.println("m1(int a) in Hello");
    }
    void m1(int... a) {
        System.out.println("m1(int... a) in Hello");
        System.out.println(a.length);
        for (int i = 0; i < a.length; i++) {
            System.out.println(a[i]);
        }
    }
    void m1(double d, int a) {
        System.out.println("m1(double d,int a) in Hello");
    }
    void m2(int... a) {
        System.out.println("m2 in Hello");
        // System.out.println(Modifier.PUBLIC);
        System.out.println(a.length);
        for (int i = 0; i < a.length; i++) {
            System.out.println(a[i]);
        }
    }
    void m2(int a, int b) {
        System.out.println("m2(int a,int b) in Hello");
    }
}

class Jtc86 {
```

```

public static void main(String arg[]) {
    Hello h1 = new Hello();
    h1.m1();
    h1.m1(111);
    h1.m1(11, 22, 33, 44, 55);
    h1.m1(11.22, 123);
    h1.m2();
}
}

```

Output:

```

D:\program\All JTC Program>java Jtc86
m1 in Hello
m1(int a) in Hello
m1(int... a) in Hello
5
11
22
33
44
55
m1(double d,int a) in Hello
m2 in Hello
0

```

Program 8.6

```

class Hai {
}
class Hello {
    void m1() {
        System.out.println("m1 in Hello");
    }
    void m1(int... a) {
        System.out.println("m1(int...a) in Hello");
        System.out.println(a);
    }
    /*
    * void m1(int a[]){ System.out.println("m1(int a[]) in Hello"); }
    */
    void m1(Hai hai) {

```

```

        System.out.println("m1(Hai hai) in Hello");
    }
    void m2(Object O) {
        System.out.println("m2(Object O) in Hello");
    }
    void m2>Hello h) {
        System.out.println("m2>Hello h) in Hello");
    }
    void m2(String arg) {
        System.out.println("m2(String arg)");
    }
    /*
    * void m2>Hello h1){ System.out.println("m2>Hello h1"); }
    */
}
class Jtc87 {
    public static void main(String arg[]) {
        Hello h1 = new Hello();
        h1.m1(11, 22);
        // h1.m1(null);
        h1.m2(null);
    }
}

```

Output:

```

D:\program\A11 JTC Program>javac Jtc87.java
Jtc87.java:35: error: reference to m2 is ambiguous
        h1.m2(null);
            ^
    both method m2>Hello) in Hello and method m2(String) in Hello match
1 error

```

Error

Cannot declare both m1(int[]) and m1(int...a) in Hello{ void m1(int a[]){

Refermce to m1 is ambiguous both method method m1(int...) in Hello &method m1(Hai) in Hello match h1.m1(null);

Note

- When the method parameter are type of reference of class type and by calling that overloaded method if it is creating ambiguity problem then it needs

to comment and it is not in any super and sub relation with other method parameter.

- If the overload method parameter have super and sub relation then preference will be given to sub class parameter.

Program 8.7

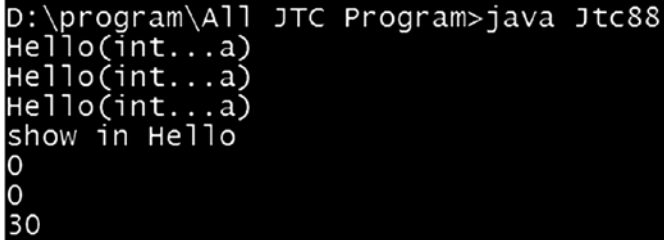
```
class Hello {
    int a;
    int b;
    static int c = 30;
    Hello(int a) {
        System.out.println("Hello(int a)");
        this.a = a;
    }
    Hello(int a, int b) {
        this(a);
        System.out.println("2 param constructor");
        this.a = a;
        this.b = b;
    }
    Hello(int... a) {
        //this(a[0], a[1]);
        System.out.println("Hello(int...a)");
        //this.a = a[0];
        //this.b = a[1];
    }
    void show() {
        System.out.println("show in Hello");
        System.out.println(a);
        System.out.println(b);
        System.out.println(c);
    }
}

class Jtc88 {
    public static void main(String arg[]) {
        Hello h = new Hello();
        Hello h1 = new Hello();
    }
}
```

```

        Hello h2 = new Hello();
        h2.show();
    }
}

```

Output:


```

D:\program\All JTC Program>java Jtc88
Hello(int...a)
Hello(int...a)
Hello(int...a)
show in Hello
0
0
30

```

Program 8.8

```

class Hai {
    static Hello h1 = new Hello();
}

class Hello {
    Hello() {
        // this(new Hello());
        // this(new Hello(null));
        // this(this);
        this(Hai.h1);
        System.out.println("Default constructor in Hello");
        System.out.println(this);
    }

    Hello(Hello h1) {
        System.out.println("1 param constructor in Hello");
    }

    Hello(Hello h1, Hello h2) {
        System.out.println("Hello(Hello h1,Hello h2) in Hello");
    }
}

class Jtc89 {
    public static void main(String arg[]) {
        Hello h1 = new Hello();
        System.out.println(h1);
    }
}

```

```

    }
}

```

Output:

```

D:\program\All JTC Program>java Jtc89
1 param constructor in Hello
Default constructor in Hello
Hello@15db9742
1 param constructor in Hello
Default constructor in Hello
Hello@6d06d69c
Hello@6d06d69c

```

Q1: Can we declare the constructor of a class as final?

Ans: NO

8.5 Local Variable

The variable which is defined inside any local context is called as local variable or the variable which is defined inside any method, constructor, blocks, etc are also called as local variables. Any local variables cannot be declared as “static”. Local variables will not be initialized automatically or internally by the JVM. Developer has to initialize that explicitly before the use or else it will give the compile time error.

Local variables can be accessed only within that context where it has been declared. It cannot be accessed outside that local context. In a same context you cannot have two local variables with the same name. Memory for the local variable will be allocated only when that corresponding context is being processed and immediately after processing that local context memory for the local variable will be destroyed automatically. The variable which is defined inside any local context (instance or static context) will not be equivalent to instance or static variable.

8.6 Local Block

A block which is defined inside any block, method or constructor is called as local block. Local blocks processing will be done only when the corresponding context where it has been declared is being processed. Local block processing cannot be done independently. We cannot declare any local block as static block or the block which is defined inside any static or instance context will not be treated as instance or static block. We can define n number of blocks inside the block.

Program 8.9

```

class Hello {
    int a;
    int b;
    static int c = 30;
    {
        System.out.println("IB in Hello");
        int a = 101;
        int b = 202;
        // static int c=303;
        {
            System.out.println("local block in IB");
            int a1 = 111;
            int b1 = 222;
            System.out.println(a);
            System.out.println(b);
            System.out.println(c);
            System.out.println(this.a);
            System.out.println(this.b);
            System.out.println(a1);
            System.out.println(b1);
            System.out.println("local block in IB");
        }
        // System.out.println(a1);
        // System.out.println(b1);
    }
    static {
        System.out.println("SB in Hello");
        int a = 101;
        int b = 202;
        // static int c=303;
        {
            System.out.println("local block in SB");
            int a1 = 111;
            int b1 = 222;
            System.out.println(a);

```



```

        System.out.println(b);
        // System.out.println(this.a);
        // System.out.println(this.b);
        System.out.println(c);
        System.out.println(a1);
        System.out.println(b1);
        System.out.println("local block in SB");
    }
    // System.out.println(a1);
    // System.out.println(b1);
}
Hello() {
    System.out.println("default constructor in Hello");
}
Hello(int a1, int b1) {
}
void m1() {
    System.out.println("m1 in Hello");
}
}
class Jtc90 {
    public static void main(String arg[]) {
        Hello h1 = new Hello();
        h1.m1();
    }
}

```

Output:

```

D:\program\All JTC Program>java Jtc90
SB in Hello
local block in SB
101
202
30
111
222
local block in SB
IB in Hello
local block in IB
101
202
30
0
0
111
222
local block in IB
default constructor in Hello
m1 in Hello

```

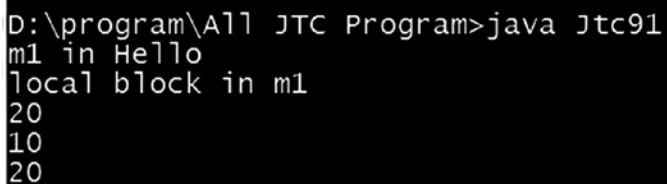
Q2: Can we declare local variable of a class as final?**Ans:** Yes, we can declare.**Q3: Can we declare the local variable as private?****Ans:** No, we cannot declare or we cannot use any access modifier with the local members of the class.**Program 8.10**

```

class Hello {
    int a = 10;
    void m1() {
        System.out.println("m1 in Hello");
        int a = 20;
        {
            System.out.println("local block in m1");
            // int a=30;
            System.out.println(a);
            System.out.println(this.a);
            System.out.println(a);
        }
    }
}

class Jtc91 {
    public static void main(String arg[]) {
        Hello h1 = new Hello();
        h1.m1();
    }
}

```

Output:


```

D:\program\All JTC Program>java Jtc91
m1 in Hello
local block in m1
20
10
20

```

```

/*error: variable a is already defined in method m1() – int a=30; */

```

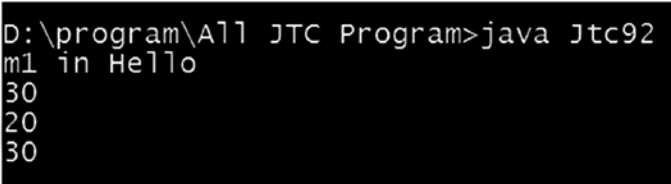
Program 8.11

```

class Hello {
    int a = 30;
    private int b = 20;
    void m1() {
        System.out.println("m1 in Hello");
        // private int aa=101;
        //System.out.println(aa);
        System.out.println(a);
        System.out.println(b);
    }
}

class Jtc92 {
    public static void main(String arg[]) {
        Hello h1 = new Hello();
        h1.m1();
        System.out.println(h1.a);
        // System.out.println(h1.b);
    }
}

```

Output:


```

D:\program\All JTC Program>java Jtc92
m1 in Hello
30
20
30

```

Program 8.12

```

class Hello{
    static int s;
    static final int c=30;
    static final int b;
    static{
        System.out.println("SB in Hello");
        b=10;
    }
}

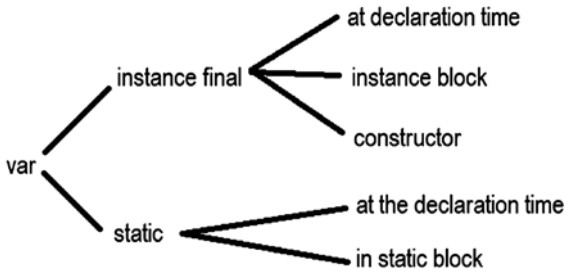
class Jtc93{
    public static void main(String arg[]){
        System.out.println("in main");
    }
}

```

```
//System.out.println(Hello.a);
//System.out.println(Hello.b);
System.out.println(Hello.c);
}}
```

Output:

```
D:\program\All JTC Program>java Jtc93
in main
30
```



Note

- In Jtc93 when you are accessing static variable a & b directly by the name of class then static block is being processed whereas when you are accessing only static final variable c with the name of class then static block is not being processed because “static final int c=30” is being initialized at the time of declaration that is the variable c value is available there itself so that there is no need to process the static block.

8.7 Assertions

The most common way of debugging is uses of sops. But the main disadvantage of sops is after fixing the bug, we should delete extra added sops otherwise these sops also will be executed at runtime which impacts performance of the system and disturbs logging mechanism.

To overcome these problems sun people introduced assertions concept in 1.4 versions. The main advantage of assertions over sops is based on our requirement we can enable or disable assertions and by default assertions are disable hence after fixing the bug it is not required to delete assert statements explicitly.

Hence, the main objective of assertions is to perform debugging. Usually we can perform debugging either in development environment or Test environment but not in production environment hence assertions concept is applicable for the development and test environments but not for the production.

Assert as keyword and identifier

assert keyword is introduced in 1.4 version hence from 1.4 version onwards we can't use assert as identifier but until 1.3 we can use assert as an identifier.

Example

```
class Hai{
public static void main(String[] args){
int assert=10;
System.out.println(assert);
}}
```

Note: It is always possible to compile a java program according to a particular version by using -source option.

Types of assert statements

There are two types of asset statements.

1. Simple Version

Syntax:

assert(b); //b should be boolean type.

1. If 'b' is true then our assumption is correct and continue rest of the program normally.
2. If 'b' is false our assumption fails and hence stop the program execution by raising assertion error.

2. Argumented Version

By using argumented version we can argument some extra information with the assertion error.

Syntax:

assert(b):e;

'b' should be boolean type.

'e' can be any type.

Example

```

class Hello{
    public static void main(String args[])
    {
        int a=10;
        assert(a>10):"Less";
        System.out.println("value of a is : "+a);
    }
}

```

Various runtime flags

1. -ea: To enable assertions in every non system class(user defined classes).
2. -enableassertions: It is exactly same as -ea.
3. -da: To disable assertions in every non system class.
4. -disableassertions: It is exactly same as -da.
5. -esa: To enable assertions in every system class(predefined classes or application classes).
6. -enablesystemassertions: It is exactly same as -esa.
7. -dsa: To disable assertions in every system class.
8. -disablesystemassertions: It is exactly same as -dsa.

Appropriate and inappropriate use of assertions

It is always inappropriate to mix programming logic with assert statements, because there is no guaranty for the execution of assert statement always at runtime.

For validating public method arguments usage of assertions is always inappropriate, because outside person is not aware whether assertions are enabled or disabled in our local system. While perform debugging if any place where the control is not allow to reach then that is the best place to use assertions.

- It is always inappropriate to use assertions for validating public method arguments.
- It is always appropriate to use assertions for validating private method arguments.
- It is always inappropriate to use assertions for validating command line arguments because these are arguments to public method main.

Assertion Error

- It is the child class of Error and is unchecked.
- Raised explicitly whenever assert statement fails.
- Even though it is legal but it is not recommended to catch AssertionError.

Example

```
class Test{
public static void main(String[] args){
int x=10;
try {
assert(x>10);
}
catch (AssertionError e){
System.out.println("not a good programming practice to catch AssertionError");
}
System.out.println(x);
}}
```

Example

```
class Test{
public static void main(String[] args){
boolean assertOn=true;
assert(assertOn):assertOn=true;
if(assertOn){
System.out.println("assert is on");
}}}
```

Chapter 9

Inheritance

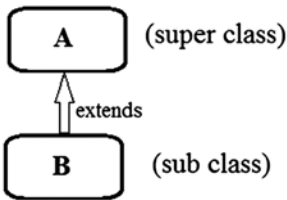
9.1 What is Inheritance?

It is the process of defining a new class from its existing class functionality.

Types of Inheritance

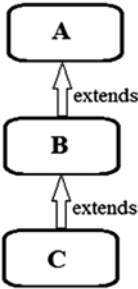
1. Simple Inheritance

In case of simple inheritance, there will be only one direct class and only one direct subclass.



2. Multilevel Inheritance

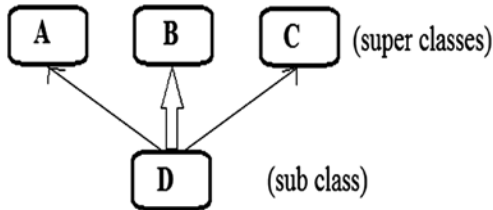
In this case, there will be only one immediate super class but many indirect super and subclasses.



	Immediate super class	Immediate subclass	Indirect super class	Indirect subclass
A	-	B	-	C
B	A	C	-	-
C	B	-	A	-

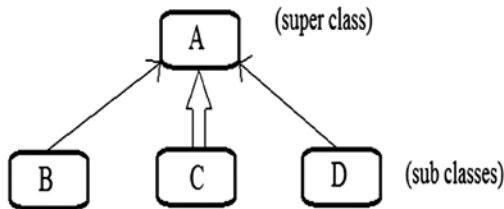
3. Multiple Inheritance

In this case, there will be only one immediate sub class and many immediate super classes.



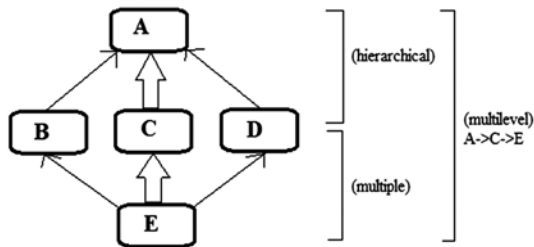
4. Hierarchical Inheritance

In this case, there will be only one immediate super class and many immediate subclasses.



5. Hybrid Inheritance

Hybrid inheritance is the mixture of all the above types of inheritance.



6. Cyclic Inheritance

It is the process of extending a class by itself.



Note:

- In java, extending more than one classes at a time is not allowed. So, the multiple inheritance is not allowed in java through classes but we can achieve multiple inheritance by using interfaces.

- If the multiple inheritance is not allowed through classes, so Hybrid inheritance is also not allowed.
- For all types of classes in java, java.lang.Object class is the super class.
- When you are extending a class, then almost all the members of the class will be inherited to the subclass.
- 'private' and 'local' members of a super class will not be inherited to the subclass.
- Final members of a class can be inherited but final class cannot be inherited.

Program 9.1

```
class Hello {
    int a = 10;
    int b = 20;
    static int c = 30;
    {
        System.out.println("IB in Hello");
    }
    static {
        System.out.println("SB in Hello");
    }
    void m1() {
        System.out.println("mi in Hello");
        System.out.println(a);
        System.out.println(b);
        System.out.println(c);
    }
    static void m2() {
        System.out.println("m2 in Hello");
        // System.out.println(a);
        // System.out.println(b);
        System.out.println(c);
    }
}

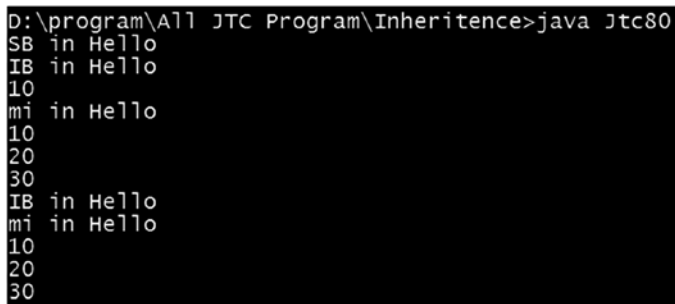
class Hai extends Hello {
    void showAll() {
        System.out.println("showAll in Hai");
        System.out.println(a);
    }
}
```

```

        System.out.println(b);
        System.out.println(c);
        m1();
        m2();
    }
}
class Jtc80 {
    public static void main(String arg[]) {
        Hai hai = new Hai();
        System.out.println(hai.a);
        hai.m1();
        Hello h1 = new Hello();
        h1.m1();
        // h1.showAll(); -cannot access subclass method from super class ref
    }
}

```

Output:



```

D:\program\All JTC Program\Inheritance>java Jtc80
SB in Hello
IB in Hello
10
mi in Hello
10
20
30
IB in Hello
mi in Hello
10
20
30

```

Q1: Can we use super inside static context?

Ans: No, we cannot.

Q2: Can we write super.super.a ?

Ans: No, it means we cannot give super reference to super.

For Example

the following is wrong:

```
/* class A{ int a, int b}
class B extends A{}
class C extends B{ S.O.P(super.super.a); }
*/
```

Q3: Can we access the variables of class A from class C by using super keyword?

Ans: No, we cannot.

Q4: What will be the order of constructor invocation in class A, B and C?

Ans: sub to super...

```
class A{}
class B extends A{}
class C extends B{ c(){ super(); } }
C c1=new C();
-----> C → B → A
```

Q5: What will be the order of constructor processing?

Ans: super to sub.

Program 9.2

```
class Hai {
    int a;
    Hai(int a) {
        System.out.println(" 1 param contr in Hai");
        this.a = a;
    }
}

class Hello extends Hai {
}

class Jtc84 {
    public static void main(String arg[]) {
        Hello h1 = new Hello();
    }
}
```

Output:

```

D:\program\all JTC Program\Inheritance>javac Jtc84.java
Jtc84.java:8: error: constructor Hai in class Hai cannot be applied to given types;
class Hello extends Hai {
^
  required: int
  found: no arguments
  reason: actual and formal argument lists differ in length
1 error

```

Error

```

/* Jtc84.java:8:error: constructor Hai in class Hai cannot be applied
class Hello extends Hai{
required : int
found : no argument
reason : actual and formal argument list differ in length
1 error
*/

```

<pre> class Hello extends Hai{ Hello() { super(); } } </pre>	automatically by JVM
--	-------------------------------------

Note

- When you are creating the object of class Hello, then it checks whether the default constructor is available inside the class hello or not. If the default constructor is not available, then JVM inserts one default constructor automatically and inside that default constructor, JVM inserts one super statement without any parameter. That super statement will try to invoke default constructor from class Hai and inside class Hai there is no default constructor available. Also, one parameterized constructor is available. If one parameterized constructor is available then JVM will not insert any default constructor automatically, so it gives the above error.

Q6: What is the proof that java.lang.Object class itself is the super class for all types of classes in JAVA?

Ans: write in command prompt:

```
javap java.lang.Object
```

(javap is java profiler used to view about the class)

Program 9.3

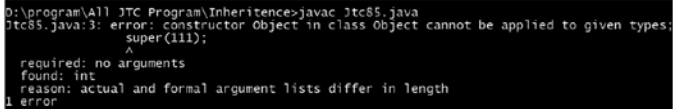
```

class Hai {
    Hai(){
        super(111);
        System.out.println("default constructor in Hai");
    }
}

class Hello extends Hai {
    Hello(){
        super();
        System.out.println("default constructor in Hello");
    }
}

class Jtc85 {
    public static void main(String arg[]) {
        Hello h1 = new Hello();
    }
}

```

Output:


```

D:\program\All JTC Program\Inheritance>javac Jtc85.java
Jtc85.java:3: error: constructor Object in class Object cannot be applied to given types;
    super(111);
        ^
    required: no arguments
    found:    int
    reason:   actual and formal argument lists differ in length
1 error

```

Error

```

/* Jtc85.java:3:error: constructor Object in class Object cannot be applied.
super(111);
required : no argument
found : int
reason : actual and formal argument lists differ in length
1 error    */

```

Q7: Can we declare default constructor of super class as private?

Ans: Yes, we can declare the default constructor of super class as private but if we do so then we will not be able to create the object of any of its subclasses or any class which is extending that class directly or indirectly (but object can be created based on parameterized constructor).

For example

```

class A{
    private A(){

```

```

}
class B extends A{
  B(){ }
}
class C extends B{
  C(){ }
}
C c=new C();    //NOT POSSIBLE

```

Errors

class B extends A : A has private access in A()
 A a=new A() : A has private access in A()

Q8: Can we declare any class as private?

Ans: No, a class can only be either “public” or “default”.

Note

- In a source file only one class can be public and that public class should contain the main method.
- You can invoke the constructor of super class from the subclass but you cannot invoke the constructor of sub class from the super class.

9.2 Difference between This and Super Keyword

THIS	SUPER
This is the instance reference variable which is used to access the class level members.	It is also an instance reference variable that is used to access the super class members.
Can call another constructor explicitly using this()	Every constructor by default calls super() which is a call to argument constructor of parent class
‘this’ points to current object of the same class.	‘super’ access super class methods and variables.
‘this’ invokes current class constructor	‘super’ invokes super class constructor.
‘this’ keyword refers to an object	‘super’ keyword refers to super class variable or methods

9.3 S-A Relationship

- HAS-A relationship is also known as composition (or) aggregation.
- There is no specific keyword to implement HAS-A relationship but mostly we can use new operator.
- The main advantage of HAS-A relationship is reusability.
- The main dis-advantage of HAS-A relationship is that it increases the dependency between the components and creates maintains problems.

9.4 Composition vs Aggregation

Composition

Without existing container object if there is no chance of existing contained objects then the relationship between container object and contained object is called composition which is a strong association.

Aggregation

Without existing container object if there is a chance of existing contained objects such type of relationship is called aggregation. In aggregation objects have weak association.

Note

- In composition container, contained objects are strongly associated, but container object holds contained objects directly
- But in Aggregation container and contained objects are weakly associated and container object just now holds the reference of contained objects.

Static control flow parent to child relationship

9.5 Coupling

The degree of dependency between the components is called coupling.

Example

```
class A{
static int i=B.j;
}
```



```
class B extends A{
static int j=C.methodOne();
}
class C extends B{
public static int methodOne(){
return D.k;
}}
class D extends C{
static int k=10;
public static void main(String[] args){
D d=new D();
}}
```

- The above components are said to be tightly coupled to each other because the dependency between the components is more.
- Tightly coupling is not a good programming practice because it has several serious disadvantages.
- Without effecting remaining components we can't modify any component hence enhancement (development) will become difficult.
- It reduces maintainability of the application.
- It doesn't promote reusability of the code.
- It is always recommended to maintain loosely coupling between the components.

Chapter 10

Polymorphism

10.1 What is Polymorphism?

One form behaving differently in different cases is called as polymorphism.

It is of mainly two types

1. Static OR (Compile time) polymorphism:-It can be achieved by Method Overloading. In case compile time polymorphism method call depends on method parameters which you are passing while calling.

2. Dynamic Polymorphism

Dynamic Polymorphism can be achieved by:

- a. Method Overriding
- b. Dynamic Dispatch.

If you will miss any one of that then you will not be able to achieve runtime polymorphism.

Dynamic Dispatch:

It is the process of assigning the Sub-class Object ref. to Super class Ref. variable.

For Example

```
Class A{
}
Class B extends A{
}
```

1. A a1=new A(); // ok
2. B b1=new B();//ok.
3. A a2=new B();// Here Sub-class(B) Object ref. Assigning to Super class Ref. Variable(a2)
4. B b2=new A();// This is not possible because Super class Object ref. Can't be assigned to Sub-class Ref. var.

In above cases 1 and 2 is fine because source and destination both are of same type. In cases of 3 and 4 will not be able perform this assignment because source and destination both are not having any relation in between. In order to perform this kind of assignment there must super and sub relation between the classes.

(Very important to understand this example to Understand Dynamic Dispatch).

```
class Hai{
}
class Hello extends Hai{
}
class Jtc107{
public static void main(String arg[]){
```

```

Hai hai1=new Hai();
Hello h1=new Hello();
Hai hai2=new Hello();
//Hello h2=new Hai();//not ok
//Hello h2=(Hello)new Hai();
Hai hai3=hai1;
Hai hai4=h1;//
Hai hai5=hai2;//
//Hello h3=(Hello)hai1;//not ok
Hello h4=h1;
Hello h5=(Hello)hai2;//not ok
}
}
/*

```

Exception in thread "main" java.lang.ClassCastException: Hai cannot be cast to Hello

at Jtc107.main(Jtc107.java:13)

*/

Note

- In case of h3 and h5 both source is of super type itself but after type casting in case of h5 it is not giving any error or exception where as in case of h3 it is giving exception at runtime that is java.lang.ClassCastException why?
- In case of h3 source is of super type i.e. hai1
- If super class ref. variable contains the super class object ref. then it can't be type casted to Sub-class ref. Var.
- If Super class ref. variable contains Sub-class Object Ref. then we can typecast to the compatible Sub-type.

```

//Hello h2=(Hello)new Hai();
//Hello h3=(Hello)hai1;//not ok

```

Program 10.1

```

class A{
    void m1(){

```

```

        System.out.println("m1 in A");
    }
}
class B extends A{
    void m2(){
        System.out.println("m2 in B");
    }
}
class C{}
class Hello{
    void m11(){
        System.out.println("m11 in Hello");
    }
    int m12(){
        System.out.println("m12 in Hello");
        return 100;
    }
    A m13(){
        System.out.println("m13 in Hello");
        return new A();
    }
    A m14(){
        System.out.println("m13 in Hello");
        return new B();
    }
    B m15(){
        System.out.println("m15 in Hello");
        return new B();
    }
    /*B m16(){
        System.out.println("m16 in Hello");
        return (B)new A();
    }*/
    /*A m17(){
        System.out.println("m17 in Hello");
        return new C();
    }*/
}

```

```

    /*B m18(){
        System.out.println("m18 in Hello");
        return (B)m13();
    }*/
    B m19(){
        System.out.println("m19 in Hello");
        return (B)m14();
    }
}
public class Jtc1 {
    public static void main(String[] args) {
        Hello h1=new Hello();
        int i=h1.m12();
        System.out.println(i);
        //h1.m16();
        //h1.m18();
        h1.m19();
        A a1=h1.m14();
        B b1=(B)h1.m14();
        b1.m1();
        b1.m2();
    }
}

```

Output:

```

D:\program\All JTC Program\Polymorphism>java Jtc1
m12 in Hello
100
m19 in Hello
m13 in Hello
m13 in Hello
m13 in Hello
m1 in A
m2 in B

```

10.2 Method Overriding

Method overriding can be done in sub class. In order to override the method remember:

1. Method name must be same.
2. Method return type also should be same as it is like super class.

3. Method parameter also must be the same like the super class.
4. Access modifier of the method should be equal or higher than the super class.



Super Class	Sub Class
Private	Private, Default, Protected, Public
Default	Default, Protected, Public
Protected	Protected, Public
Public	Public

Note

1. Private method of a class cannot be overridden.
2. Final method of a class cannot be overridden.
3. Static method of a class cannot be overridden. It can only hide it.
4. In case of instance method, method call depends upon the content of the reference variable by which you are calling.
5. In case of static method, method call depends upon on the basis of type which we are calling.

Program 10.2

```
class Hello{
    void m1(){
        System.out.println("m1 in Hello");
    }
    void m2(int a){
        System.out.println("m2 in Hello");
    }
    private void m3(){
        System.out.println("m3 in Hello");
    }
    protected int m4(){
        System.out.println("m4 in Hello");
        return 111;
    }
}

class Hai extends Hello{
    void m1(){
        System.out.println("m1 in Hai");
    }
}
```

```

    /*int m2(int a){
        System.out.println("m2(int a) in Hai");
        return 10;
    }*/
    void m2(int a){
        System.out.println("m2 (int a) in Hai");
    }
    void m2(int a,int b){
        System.out.println("m2 (int a,int b) in Hai");
    }
    public void m3(){
        System.out.println("m3 in Hai");
    }
    /*int m4(){
        System.out.println("m4 in Hai");
        return 101;
    }*/
    public int m4(){
        System.out.println("m4 in Hai");
        return 101;
    }
    private void m5(){
        System.out.println("m5 in Hai");
    }
}

public class Jtc109 {
    public static void main(String[] args) {
        Hai hai =new Hai();
        hai.m1();
        hai.m2(11);
        hai.m2(11,22);
    }
}

```

Output:

```
D:\program\All JTC Program\Polymorphism>java Jtc109
m1 in Hai
m2 (int a) in Hai
m2 (int a,int b) in Hai
```

Error: m2(int) in Hai cannot override m2(int) in Hello

```
Int m2(int a){
```

Return type int is not compatible with void. M4() in Hai cannot override m4() in Hello
 Hello int m4(){

Attempting to assign weaker access privilege was protected.

Program 10.3

```
class Hello{
    final void m1(){
        System.out.println("m1 in Hello");
    }
    static void m2(){
        System.out.println("m2 in Hello");
    }
    void m3(){
        System.out.println("m3 in Hello");
    }
}

class Hai extends Hello{
    /*void m1(){
        System.out.println("m1 in Hai");
    }*/

    /*void m2(){
        System.out.println("m2 in Hai");
    }*/

    /*static void m3(){
        System.out.println("m3 in Hai");
    }*/
    static void m2(){
        System.out.println("m2 in Hai");
    }
}
```



```

        void m3(){
            System.out.println("m4 in Hai");
        }
    }
}

public class Jtc110 {
    public static void main(String[] args) {
        Hai hai = new Hai();
        hai.m1();
        hai.m2();
        hai.m3();
    }
}

```

Output:

```

D:\program\All JTC Program\Polymorphism>java Jtc110
m1 in Hello
m2 in Hai
m4 in Hai

```

Error

m1() in Hai cannot override m1() in Hello. void m1(){ } overridden method is final.
 m2() in Hai cannot override m2() in Hello void m2(){ } overridden method is static.
 m3() in Hai cannot override m3() in Hello. static void m3() overriding method is static

Note: Static method can be hidden by static method only.

Instance method can be overridden by instance method only.

Or

Instance method cannot be overridden by static method.

Program 10.4

```

class A{
}
class B extends A{
}
class Hello{
    A m1(){
        System.out.println("m1 in Hello");
        return new A();
    }
}

```

```

    }
    B m2(){
        System.out.println("m2 in Hello");
        return new B();
    }
    A m3( A a1){
        System.out.println("m3 (A a1)in Hello");
        return a1;
    }
    A m4(B b1){
        System.out.println("m4(B b1) in Hello");
        return b1;
    }
}
class Hai{
    B m1(){
        System.out.println("m1 in Hai");
        return new B();
    }
    A m2(){
        System.out.println("m2 in Hai");
        return new B();
    }
    A m3(B b1){
        System.out.println("m3 (B b1) in Hai");
        return b1;
    }
    A m4(A a1){
        System.out.println("m3 (A a1) in Hai");
        return a1;
    }
}
public class Jtc111 {
    public static void main(String[] args) {
        Hai hai=new Hai();
        hai.m1();
    }
}

```

Output:

```
D:\program\All JTC Program\Polymorphism>java Jtc111
m1 in Hai
```

Error

m2() in Hai cannot override in Hello A m2(){
Return type is not compatible with B

Program 10.5

```
class Animal {
    int a = 10;
    int b = 20;
    static int c = 30;
    void eating() {
        System.out.println("Animal is eating");
    }
    int sleeping() {
        System.out.println("Animal is sleeping");
        return 10;
    }
    void thinking() {
        System.out.println("Animal is thinking");
    }
    static void seeing() {
        System.out.println("Animal is seeing");
    }
}

class Dog extends Animal {
    int a = 101;
    int b = 202;
    static int c = 303;
    void eating() {
        System.out.println("Dog is eating");
    }
    int sleeping() {
        System.out.println("Dog is sleeping");
        return 10;
    }
    static void seeing() {
```

```

        System.out.println("Dog is seeing");
    }
    void barking() {
        System.out.println("Dog is barking");
    }
}

class Cat extends Animal {
    void eating() {
        System.out.println("Cat is eating");
    }
    static void seeing() {
        System.out.println("cat is seeing");
    }
    void drinking() {
        System.out.println("Cat is drinking");
    }
}

class Jtc112 {
    public static void main(String arg[]) {
        Animal ani1 = new Animal();
        ani1.eating();
        ani1.thinking();
        ani1.seeing();
        Dog dog1 = new Dog();
        dog1.eating();
        dog1.thinking();
        dog1.seeing();
        dog1.barking();
        System.out.println("*****\n");
        Animal ani2 = null;
        ani2 = new Dog();
        ani2.eating();
        ani2.thinking();
        ani2.seeing();
        // ani2.barking();
        System.out.println(ani2.a);
        System.out.println(ani2.c);
    }
}

```

```

        Animal ani3 = new Cat();
        ani3.eating();
        ani3.seeing();
        // ani3.drinking();
    }
}

```

Output:

```

D:\program\All JTC Program\Polymorphism>java Jtc112
Animal is eating
Animal is thinking
Animal is seeing
Dog is eating
Animal is thinking
Dog is seeing
Dog is barking
*****
Dog is eating
Animal is thinking
Animal is seeing
10
30
Cat is eating
Animal is seeing

```

Note

- When you are calling any method with the super class reference variable which contains sub class object reference then that method signature must be available inside super class. If it is not available then it will give error.
- When you are calling any method with the super class reference then internally first JVM will verify that whether that method signature is available inside super class or not. If it is available inside super class then it checks for the sub class implementation. If sub class implementation is available then it will be processed.
- In case of static method when you call by super class reference variable which contains sub class object reference then in that case always super class implementation will be called.
- By using the static method we will not be able to achieve the Runtime polymorphism.
- We cannot override static method but we can only hide that static method because static method supports the early binding concept and instance method supports late binding concept.

Program 10.6

```
class A{
    void m11(){
        System.out.println("m11 in A");
    }
    void m12(){
        System.out.println("m12 in A");
    }
}

class B extends A{
    void m11(){
        System.out.println("m11 in B");
    }
    void m13(){
        System.out.println("m13 in B");
    }
}

class Hello{
    void m1(){
        System.out.println("m1 in Hello");
    }
    A m2(){
        System.out.println("m2 in Hello");
        return new B();
    }
    B m3(){
        System.out.println("m3 in Hello");
        return (B)m2();
    }
    void m4(int a){
        System.out.println("m4 in Hello");
    }
}

class Hai extends Hello{
    void m1(){
        System.out.println("m1 in Hai");
    }
}
```

```

    B m2(){
        System.out.println("m2 in Hai");
        return (B)super.m2();
    }
    void m4(){
        System.out.println("m4 in Hai");
    }
    B m5(){
        System.out.println("m5 in Hai");
        return new B();
    }
}
public class Jtc113 {
    public static void main(String[] args) {
        Hello h1=new Hello();
        h1.m1();
        A a1=h1.m2();
        a1.m11();
        a1.m12();
        //a1.m13();
        //(h1.m2()).m13();
        new Hai().m5().m13();
    }
}

```

Output:

```

D:\program\All JTC Program\Polymorphism>java Jtc113
m1 in Hello
m2 in Hello
m11 in B
m12 in A
m5 in Hai
m13 in B

```

Q1: Can we override static method?**Ans:** No, we cannot.**Q2: Can we override final method?****Ans:** No, we cannot

Q3: Can we override private method?

Ans: No, we cannot override but if you declare any method inside the super class as private then that super class private method will not be visible to the sub class so inside sub class if you are writing that method with the same name with any access modifier it is not going to create any kind of problem in further.

Example

```
package Polymorphism5;
class Hello{
    void m1(){
        System.out.println("m1 in Hello");
    }
    void m2(){
        System.out.println("m2 in Hello");
    }
    void m3(){
        System.out.println("m3 in Hello");
    }
}
class Hai extends Hello{
    void m1(){
        System.out.println("m1 in Hai");
    }
    void m2(){
        System.out.println("m2 in Hai");
        super.m2();
    }
}
public class Jtc114 {
    public static void main(String[] args) {
        Hai hai=new Hai();
        hai.m1();
        hai.m2();
        hai.m3();
    }
}
```


Output:

```
D:\program\All JTC Program\Polymorphism>java Jtc114
m1 in Hai
m2 in Hai
m2 in Hello
m3 in Hello
```

- When you do not want to use the super class implementation as it is then override that method inside the sub class like m1() method.
- If you want to use super class as well as sub class implementation both then override that method inside the sub class and access super class implementation by using the super keyword like m2() method.
- If you want to use super class implementation as it is then do not override that inside the sub class like m3() method.

Q4: Can we override main method inside the main method?

Ans: No, we cannot override main method but we can overload “main()” method.

Chapter 11

Abstract Class

11.1 What is Abstract class?

Abstract class is special type of class which contain “method with body or method with implementation and method without implementation both. When you have written a method without any implementation or without any body then that method must be declare as an “abstract method”. If a class contain one or more abstract method then that class must be declares as abstract. Abstract class may or may not contain any “abstract method”. Abstract class object cannot be created. Only reference of the abstract class can be created. When you are extending any abstract class remember the following points:

- When any class is extending any abstract class then the sub class must be override all the abstract method of super or abstract class.
- If you have not provided any implementation or not overridden abstract method of super class then sub class must declare as abstract class.

Program 11.1

```
abstract class Hello{
    int a=10;
    int b=20;
    static int c=30;
    void m1(){
        System.out.println("m1 in Hello");
    }
    abstract void m2();
    public abstract void m3();
}

abstract class Hai extends Hello{
    void m2(){
        System.out.println("m2 in Hai");
    }
}

class Hai1 extends Hai{
    /*void m3(){
```

```

        System.out.println("m3 in Hai1");
    }*/
    public void m3(){
        System.out.println("m3 in Hai1");
    }
}
public class Jtc115 {
    public static void main(String[] args) {
        //Hello h=new Hello(); -Abstract class cannot be declare their own object
        Hello h=null;
        h=new Hai1();
        h.m2();
        h.m3();
    }
}

```

Output:

```

D:\program\All JTC Program>java Jtc115
m2 in Hai
m3 in Hai1

```

Program 11.2

```

abstract class Hello{
    int a=10;
    int b=20;
    static int c=30;
    {
        System.out.println("IB in Hello");
    }
    static{
        System.out.println("SB in Hello");
    }
    Hello(){
        System.out.println("Default Constructor in Hello");
    }
    Hello(int a,int b){
        System.out.println("2 para. constr. in Hello");
    }
}

```

```

        this.a=a;
        this.b=b;
    }
    void m1(){
        System.out.println("m1 in Hello");
        System.out.println(a);
        System.out.println(b);
        System.out.println(c);
    }
    abstract void m2();
    /*private abstract void m3();
    final abstract void m4();
    static abstract void m5();*/
}

class Hai extends Hello{
    /*private void m2(){
        System.out.println("m2 in Hai");
    }*/
    public void m2(){
        System.out.println("m2 in Hai");
    }
    //void m3(){        }
    public void m3(){
        System.out.println("m3 in Hai");
    }
    void show(){
        System.out.println("show in Hai");
        System.out.println(a);
        System.out.println(b);
        System.out.println(c);
    }
}

public class Jtc116 {
    public static void main(String[] args) {
        //Hello h1=new Hello(); -Abstract class cannot be declare their own
        object
        Hello h2=null;
        h2=new Hai();
    }
}

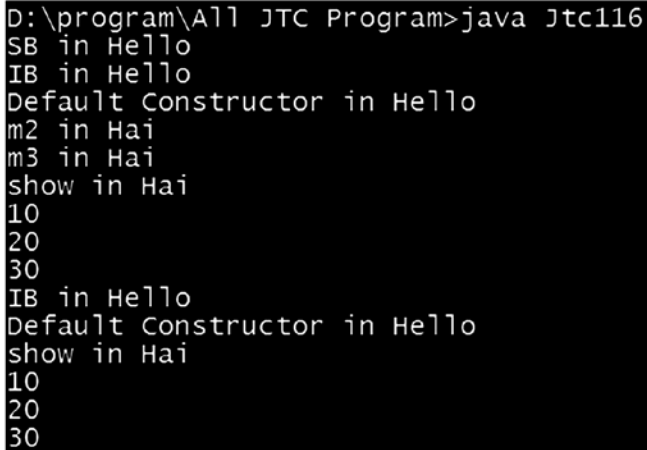
```

```

        h2.m2();
        //h2.m3();
        //h2.show();
        Hai hai=(Hai)h2;
        hai.m3();
        hai.show();
        new Hai().show();
    }
}

```

Output:



```

D:\program\All JTC Program>java Jtc116
SB in Hello
IB in Hello
Default Constructor in Hello
m2 in Hai
m3 in Hai
show in Hai
10
20
30
IB in Hello
Default Constructor in Hello
show in Hai
10
20
30

```

Error: m2() in Hai cannot override m2 in Hello

```
private void m2(){
```

Jtc116:java:27:illegal combination of modifier abstract and final private
abstract void m3();

Note

- We cannot use “static abstract”, “final abstract” or “private abstract” combination because it is called illegal combination.
- In case of h2.show method, it is given error because whenever you are calling any method which contain super class reference variable which contain sub class object reference then that method signature must be available inside the super class. If not available then you will not able to call that.
- Constructor of a class will be process only when the object of a class is being created, but in the case of abstract class we cannot create any object.

So the contractor object class will not be process directly but when you are creating the sub class object then internally it will check whether any default constructor available inside sub class or not and even parameterize constructor is not available then JVM insert one default constructor in sub class and default super () statement will also be inserted and due to that super statement abstract class constructor will be process.

Program 11.3

```
abstract class Animal{
    public abstract void eating();
    public abstract void sleeping();
}

abstract class Dog extends Animal{
    public void eating(){
        System.out.println("Dog is eating");
    }
}

class MyDog extends Dog{
    public void sleeping(){
        System.out.println("MyDog is sleeping");
    }
}

abstract class Cat extends Animal{
    public void eating(){
        System.out.println("Cat is eating");
    }
    public void sleeping(){
        System.out.println("Cat is sleeping");
    }
}

class MyCat extends Cat{
}

class Animal_Factory{
    static final int DOG=1;
    static final int CAT=2;
    static Animal getAnimal(int a){
        Animal ani=null;
        if(a==1){
```

```

        ani=new MyDog();
    }else if(a==2){
        ani=new MyCat();
    }
    return ani;
}
}
public class Jtc117 {
    public static void main(String[] args) {
        Animal ani=null;
        ani=Animal_Factory.getAnimal(Animal_Factory.DOG);
        show(ani);
        ani=Animal_Factory.getAnimal(Animal_Factory.CAT);
        show(ani);
    }
    static void show(Animal ani){
        ani.eating();
        ani.sleeping();
    }
}

```

Output:

```

D:\program\All JTC Program>java Jtc117
Dog is eating
MyDog is sleeping
Cat is eating
Cat is sleeping

```

Program 11.4

```

abstract class Hello{
    int a=10;
    int b=20;
    static int c=30;
    {
        System.out.println("IB in Hello");
        System.out.println(this);
    }
    Hello(){
        System.out.println("Default Constructor in Hello");
    }
}

```

```

    }
    Hello(int a,int b){
        System.out.println("2 para. constr. in Hello");
        this.a=a;
        this.b=b;
    }
    protected abstract void m1();
    void m2(){
        int a=10;
        int b=20;
        int c=30;
        System.out.println("m2 in Hello");
        System.out.println(a);
        System.out.println(b);
        System.out.println(c);
        System.out.println(this.a);
        System.out.println(this.b);
        System.out.println(this.c);
        System.out.println(this);
    }
    static void m3(){
        System.out.println("use of static method in Abstract class");
    }
}

class Hai extends Hello{
    int a;
    int b;
    static int c=60;
    {
        System.out.println("IB in Hai");
        System.out.println(this);
    }
    Hai(){
        System.out.println("Default constr in Hai");
    }
    Hai(int a,int b){
        super(777,666);
        System.out.println("2 para constr. in Hai");
    }
}

```



```

        this.a=a;
        this.b=b;
    }
    void show(){
        int a=101;
        int b=202;
        int c=303;
        System.out.println("show in Hai");
        System.out.println(a);
        System.out.println(b);
        System.out.println(c);
        System.out.println("Hai class level variable");
        System.out.println(this.a);
        System.out.println(this.b);
        System.out.println(this.c);
        System.out.println("Hello class level variable");
        System.out.println(super.a);
        System.out.println(super.b);
        System.out.println(super.c);
    }
    public void m1(){
        System.out.println("m1 in Hai");
    }
    static void m3(){
        System.out.println("m3 in Hai");
    }
    static void m4(){
        System.out.println("m4 in Hai");
    }
}

public class Jtc118 {
    public static void main(String[] args) {
        Hello h1=null;
        h1=new Hai();
        //h1.show();
        h1.m2();
        System.out.println(h1);
        Hello h2=new Hai(999,888);
    }
}

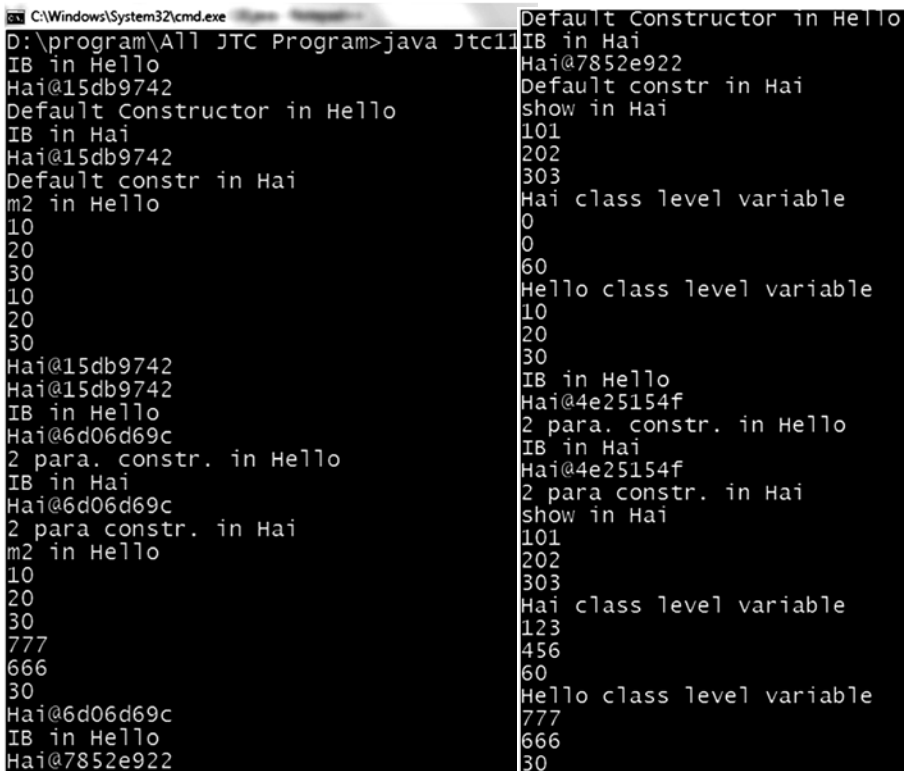
```

```

        h2.m2();
        Hai hai=new Hai();
        hai.show();
        Hai hai1=new Hai(123,456);
        hai1.show();
        Hello h3=new Hai();
        h3.m3();
        //h3.m4();
    }
}

```

Output:



```

C:\Windows\System32\cmd.exe
D:\program\All JTC Program>java Jtc11
IB in Hello
Hai@15db9742
Default Constructor in Hello
IB in Hai
Hai@15db9742
Default constr in Hai
m2 in Hello
10
20
30
10
20
30
Hai@15db9742
Hai@15db9742
IB in Hello
Hai@6d06d69c
2 para. constr. in Hello
IB in Hai
Hai@6d06d69c
2 para constr. in Hai
m2 in Hello
10
20
30
777
666
30
Hai@6d06d69c
IB in Hello
Hai@7852e922
Default Constructor in Hello
IB in Hai
Hai@7852e922
Default constr in Hai
show in Hai
101
202
303
Hai class level variable
0
0
60
Hello class level variable
10
20
30
IB in Hello
Hai@4e25154f
2 para. constr. in Hello
IB in Hai
Hai@4e25154f
2 para constr. in Hai
show in Hai
101
202
303
Hai class level variable
123
456
60
Hello class level variable
777
666
30

```

Note

In order to achieve the Runtime polymorphism it is always recommendable to use instance members inside the class. By using static method we cannot achieve runtime polymorphism.

Q1: Can I declare abstract class as a final?

Ans: No, we cannot declare because if it is abstract class then it must be inherited somewhere and where as final represents that this class cannot be inherited. So both are contradict to each other.

Example

```
final abstract class A{  
}  
class B extends A{  
}  
public class Jtc119 {  
    public static void main(String[] args) {  
    }  
}
```

Chapter 12

Interface

12.1 What is an interface?

In order to declare an interface, we need to use the keyword “interface”. Interface is fully abstract. An interface can contain only two types of members:

1. public abstract method
2. public static final variable

If you are declaring any method inside an interface then by default it will be a public abstract method and if it is a variable inside any interface then by default it will be public static final variable. Inside an interface you cannot write the following members:

- Instance variable
- Instance block
- Static block
- Constructor

Any concrete method (concrete methods are methods with implementation). We cannot instantiate any interface. If any class is trying to use any interface then that subclass needs to use the keyword “implements”. When a class is implementing any interface, then that subclass must have to override all the methods of the interface. If it is not overriding all the methods of the interface then that subclass must be declared as abstract.

Program: 12.1

```
interface I1{
    public static final int a=10;
    static final int b=20;
    final int c=30;
    int d=40;
    public abstract void m1();
    abstract void m2();
    void m3();
    /* {
        System.out.println("IB is not allowed in an Interface");
    }
```

```

void mm1(){ Abstract methods do not specify a body }
    */
}
abstract class Hello implements I1{
    public void m1(){
        System.out.println("m1 in Hello");
    }
    public void m2(){
        System.out.println("m2 in Hello");
    }
    /* without using 'public' error: reduce the visibility of the inherited method
from I1 */
    /* public void m3(){
        System.out.println("m3 in Hello");
    }
    */
}
class Hai extends Hello{
    public void m3(){
        System.out.println("m3 in Hai");
    }
    void m4(){
        System.out.println("m4 in Hai");
    }
}
public class Jtc98{
    public static void main(String arg[]){
        // I1 i1=new Hello();
        I1 i1=new Hai();
        Hello h1=new Hai();
        i1.m1();
        i1.m2();
        i1.m3();
        //i1.m4();
        h1.m1();
        h1.m3();
        //new Hai.m4();      //anonymous object (cannot be reused)
    }
}

```

```

    }
}

```

Output:

```

D:\program\All JTC Program\Interface>java Jtc98
m1 in Hello
m2 in Hello
m3 in Hai
m1 in Hello
m3 in Hai

```

From JDK1.8

- Default method concept got introduced in interface, with method implementation.
e.g.:

```
interface Car{
    default void driver()           //it is public by default
    {
        System.out.println("driver");
    }
    steering();
    headlights();
}
```
- Static methods inside interface are allowed.
e.g.:

```
interface I1 {
    static void m1() {}           //it is public by default
}
```
- Lambda expression concept is introduced.

- From JDK1.8, you can write the method implementation inside the interface, but you must have to declare that method as default.
- Even you can write static methods with implementation
- This default and static methods written inside an interface is by default public.
- You cannot use this default keyword inside any class or abstract class other than interface.

Program 12.2

```

interface I1{
    int a=10;
    void m1();
    void m2();
    default void m3(){
        System.out.println("m3 in I1");
    }
    static void m4(){
        System.out.println("m4 in I1");
    }
}

```

```

class Hello implements I1{
    public void m1(){
        System.out.println("m1 in Hello");
    }
    public void m2(){
        System.out.println("m2 in Hello");
    }
    public void m3(){
        System.out.println("m3 in Hello");
    }
    public void m4(){
        System.out.println("m4 in Hello");
    }
}
class Jtc99{
    public static void main(String arg[]){
        I1 i1=new Hello();
        i1.m1();
        i1.m2();
        i1.m3();
    }
}

```

Output:

```

D:\program\All JTC Program\Interface>java Jtc99
m1 in Hello
m2 in Hello
m3 in Hello

```

12.2 Multiple Inheritance through Interfaces**Program 12.3**

```

interface I1{
    public static int a=10;
    public abstract void m1();
}
interface I2{
    int b=20;
    void m2();
}

```

```

}
interface I3 extends I1,I2{
    int c=30;
    void m3();
}
interface I4{
    int d=40;
    void m4();
}
interface I5{
    int e=50;
    void m5();
}
interface I6 extends I4,I5{
    int c=60;
    void m3();
}
class Hello implements I3,I6{
    public void m1(){
        System.out.println("m1 in Hello");
    }
    public void m2(){
        System.out.println("m2 in Hello");
    }
    public void m4(){
        System.out.println("m4 in Hello");
    }
    public void m5(){
        System.out.println("m5 in Hello");
    }
    public void m3(){
        System.out.println("m3 in Hello");
    }
    void show(){
        System.out.println("show in Hello");
        System.out.println(a);
        System.out.println(b);
    }
}

```



```

        System.out.println(d);
        System.out.println(e);
        System.out.println(a);
        //System.out.println(c);
        System.out.println(l3.c);
        System.out.println(l6.c);
    }
}
class Jtc100{
    public static void main(String arg[]){
        l3 i3=null;
        i3=new Hello();
        i3.m1();
        i3.m2();
        i3.m3();
        //i3.m4();
        l6 i6=null;
        i6=new Hello();
        //i6.m1();
        i6.m4();
        i6.m5();
        i6.m3();
        //i6.show();
        Hello h1=new Hello();
        h1.show();
    }
}

```

Output:

```

D:\program\All JTC Program\Interface>java Jtc100
m1 in Hello
m2 in Hello
m3 in Hello
m4 in Hello
m5 in Hello
m3 in Hello
show in Hello
10
20
40
50
10
30
60

```

```
/* ERROR: reference to c is ambiguous, both var c in I3 and var c in I6 match for
System.out.println(c);
*/
```

Program 12.4

```
interface I1 {
    void m1();
}
interface I2 {
    int m1();
}
interface I3 {
    int m1(int a);
}
/*
 * class Hello implements I1,I2{ public void m1(); }
 */
class Hai implements I1, I3 {
    public void m1() {
        System.out.println("m1 in Hai");
    }
}
class Jtc101 {
    public static void main(String arg[]) {
        I1 i1 = new Hai();
        I3 i3 = new Hai();
        i1.m1();
        i3.m3();
    }
}
/* ERROR: Hello is not abstract and does not override abstract method m1() in I2
ERROR: m1() in Hello cannot implement m1() in I2
'public void m1(){'
Return type is not compatible with int.
*/
```

Program 12.5

```

interface I1{
    public abstract void m1();
    /* void m1(){
    System.out.println("m1 in I1");
    */
    default void m2(){
        System.out.println("m2 in I1");
    }
    static void m3(){
        System.out.println("m3 in I1");
    }
    /* default static void m4(){
    System.out.println("m4 in I1");
    */
}

class Hello implements I1{
    public void m1(){
        System.out.println("m1 in Hello");
    }
    /*default void m2(){
    System.out.println("m2 in Hello");
    }
    */
    public void m2(){
        System.out.println("m2 in Hello");
    }
    public void m3(){
        System.out.println("m3 in Hello");
    }
}

class Jtc102{
    public static void main(String arg[]){
        I1 i1=null;
        i1=new Hello();
        i1.m1();
        i1.m2();
    }
}

```

```

        //i1.m3();
        I1.m3();
    }
}

```

Output:

```

D:\program\All JTC Program\Interface>java Jtc102
m1 in Hello
m2 in Hello
m3 in I1

```

```

/*ERROR: illegal combination of modifiers : static and default in
default static void m4()
*/

```

Note

- When a class is trying to use any interface then it has to use the keyword “implements”.
- If an interface is trying to use some other interfaces then it should use the keyword “extends”
- A class cannot extend more than one class, but an interface can extend more than one interfaces.
- A class can implement more than one interface.
- When a class is implementing more than one interface, and two same variable names is coming from two different interfaces, then you cannot access that directly. It will give the ambiguity problem.
- In this case, you need to access the variables with their interface name.
- In case of Jtc100, we cannot call I6.m1() method because m1() method signature is not available inside interfaces I4,I5 and I6. I3.m4() method also cannot be called because m4() method signature is not available inside I1, I2 and I3 interfaces.

Program 12.6

```

interface I1{
    void m1();
    int m2();
}

interface I2{
    void m1();
    void m2();
}

```

```

}
class A implements I1, I2{
public void m1(){
System.out.println("m1 in A");
}
public void m2(){
/* multiple marker at this line
-return type is incompatible with I1.m2()
-implements com.p2.I1.m2
*/
}
public int m2(){
return 10;
}
}

```

Q1: Can I write interface without any member?

Ans: Yes, an interface without any member is called as marker interface.

Q2: What is a marker interface?

Ans: It is used to mark the eligibility of a class. For the processing of the marker interface, you need to write it explicitly by using the reflection mechanism. Sun has already given some marker interfaces, i.e., `Cloneable`, `RandomAccess`, `Serializable` etc.

Q3: Can a class can extend a class and implement an interface together?

Ans: Yes, e.g.:

```
class Hello extends A implements I1, I2
```

Q4: What is the difference between extends and implements keywords?

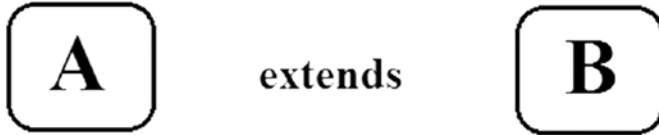
Ans: Extends keyword is used for the same type, i.e. either with classes or with interfaces whereas implements keyword is used by class for implementing interfaces.

The class implementing any interface has the responsibility to override all the methods or else it is declared as abstract.

Q5: Why multiple interface is not allowed through class but allowed through interfaces?

Ans: Possibilities of “extends” :

- Class extends only one class - ✓



- Class extends one or more classes - ✗
 - Interface extends one or more interfaces - ✓
 - Interface extends one or more classes - ✗
-

Possibilities of “implements”:

- Interface implements one or more interfaces - ✗



- Interface implements one or more classes - ✗
- Class implements one or more interfaces - ✓
- Class implements one or more classes - ✗

Note

- If one try with no catch is there, then one finally block is needed.
e.g.:

```
try{
}
finally{
}
```
- One try with multiple catch can be used.
- From JDK1.7, try with resource is being used that does not require the use of finally block.

- JAVAC Jtc1.java can get compiled even though java is a case sensitive language.
- javac -verbose Jtc1.java
- You can write main method as final because it is already static, both final and static methods cannot be overridden
e.g.: public final static void main(String arg[]){}
- If a class is declared as final then it cannot be extended to its sub class, its members are not final if method is final is cannot be overridden but if a class is final then no need to define final methods because if class cannot be extended then there will be no need of overriding of methods.

INNER CLASSES

12.3 Inner class

A class which is defined inside another class is called as inner class. There are mainly following types of inner class:

1. Non static Inner class or Instance inner class
2. Static Inner class
3. Method Local inner class.
4. Anonymous Inner class

Non static Inner Class

- A class which is defined inside another class without any static keyword is called as non-static Inner class.
- Inside non-static Inner class all the members of outer class can be accessed directly. You cannot have static declaration but static member of outer class can be accessed inside that.
- In order to access the members of inner class inside, you need to create the object of inner class and through that reference it can be accessed.
- In order to create the Inner class object outside outer class first you need to create the object of outer class and through that reference you can create the object of Inner class also.
- By using object reference of the outer class you cannot access the - the Inner class.
- By creating the inner class object you cannot access the outer class members.

Program 12.7

```
class Outer{
    int a=10;
    int b=20;
```

```

static int c=30;
void m1(){
    System.out.println("m1 in outer class");
}
static void m2(){
    System.out.println("m2 in outer class");
}
class Inner{
    int a1=101;
    //static int b1=202;
    static final int c1=303;
    void m11(){
        System.out.println("m11 in inner class");
    }
    /*static void m12(){
        System.out.println("m12 in Inner class");
    }*/
    void showInner(){
        System.out.println("showInner in Inner");
        System.out.println(a);
        System.out.println(b);
        System.out.println(c);
        System.out.println(a1);
        System.out.println(c1);
        m1();
        m2();
    }
} //Inner class
void showOuter(){
    System.out.println("showOuter in Outer");
    System.out.println(a);
    System.out.println(b);
    System.out.println(c);
    //System.out.println(a1);
    //System.out.println(c1);
    Inner inr=new Inner();
    System.out.println(inr.a1);
    System.out.println(inr.c1);
}

```



```

    }
}
public class Jtc124 {
    public static void main(String[] args) {
        Outer otr=new Outer();
        otr.m1();
        //otr.m11();
        //Inner inr=new Inner();
        //Outer.Inner oi1=new Outer.Inner();
        //Outer.Inner oi2=new Outer().Inner();
        //Outer.Inner oi3=Outer.new Inner();
        Outer.Inner oi4 =new Outer().new Inner();
        oi4.m11();
        //oi4.m1();
    }
}

```

Output:

```

D:\program\All JTC Program\Inner Class>java Jtc124
m1 in outer class
m11 in inner class

```

Q6: For above program how many (.class) files can be generate?

Ans: For the above program three .class file will be generated which is named as Outer.class,

Jtc124.class and Outer\$Inner.class. Here '\$' represents anonymous class.

Q7: Can I write the Inner class inside Inner class.

Ans: Yes, we can create a Inner class inside Inner class.

Note:

The variables or members declare inside Inner class will not be considered as local members.

Program 12.8

```

class Outer{
    int a=10;
    static int b=20;
}

```

```

{
    System.out.println("lb in outer");
}
static{
    System.out.println("Sb in outer");
}
class Inner1{
    int a1=101;
    //static int b1=202;
    static final int ab1=300;
    class Inner2{
        int ab=202;
        static final int bc=303;
        void showInner2(){
            System.out.println(a);
            System.out.println(b);
            System.out.println(a1);
            System.out.println(ab);
            System.out.println(bc);
        }
    }
    //close Inner2
    void showInner1(){
        System.out.println("showInner1 in Inner");
        //System.out.println(ab);
        System.out.println(new Inner2().ab);
    }
}
//close Inner1
void showOuter(){
    System.out.println("showOuter() in Outer");
    Inner1 inr=new Inner1();
    System.out.println(inr.a1);
    System.out.println(Inner1.ab1);
    Inner1.Inner2 inr12=new Inner1().new Inner2();
    System.out.println(inr12.ab);
}
}

```

```

public class Jtc125 {
    public static void main(String[] args) {
        Outer.Inner1.Inner2 oinr2=new Outer().new Inner1().new Inner2();
        //System.out.println(oinr2.a);
        oinr2.showInner2();
    }
}

```

Output:

```

D:\program\All JTC Program\Inner Class>java Jtc125
Sb in outer
Ib in outer
10
20
101
202
303

```

Q8: Can I write a class within any Interface?

```

interface I1{
    int a=10;
    void m11();
    class Inner{
        int a1=10;
        void m1(){
            System.out.println("m1 in Inner");
        }
    }
}

interface I2{
    static int b=20;
    void m11();
    class Inner2 implements I1,I2{
        public void m11(){
            System.out.println("m11 in Inner2");
        }
    }
}

class HelloTest extends I1.Inner{
    public void m1(){

```

```

        System.out.println("m1 in HelloTest");
    }
}

public class Jtc126 {
    public static void main(String[] args) {
        I1 i1=null;
        i1=new I2.Inner2();
        i1.m11();
        I1.Inner ii1=null;
        ii1=new HelloTest();
        ii1.m1();
    }
}

```

Output:

```

D:\program\All JTC Program\Inner Class>java Jtc126
m11 in Inner2
m1 in HelloTest

```

Program 12.9

```

class Outer{
    private int ab=1010;
    class Inner{
        public int a1=10;
        private int b1=202;
        void m1(){
            System.out.println("m1 in Inner");
        }
    }
}

//close Inner
void showOuter(){
    Inner inr=new Inner();
    System.out.println(ab);
    System.err.println(inr.a1);
    System.out.println(inr.b1);
}

}

public class Jtc127 {
    public static void main(String[] args) {
        Outer otr=new Outer();
    }
}

```

```

        //System.out.println(otr.ab);
        Outer.Inner oi1=otr.new Inner();
        //System.out.println(oi1.b1);
        otr.showOuter();
    }
}

```

Output:

```

D:\program\All JTC Program\Inner Class>java Jtc127
1010
10
202

```

Q9: Can I declare Inner class as private?

Ans: Yes, we can declare inner class as private.

Program 12.10

```

class Outer{
    int a=10;
    static int b=20;
    private class Inner{
        int ab=101;
        void m1(){
            System.out.println("m1 in Inner");
        }
    } //close Inner
    void show(){
        System.out.println("Show in Inner");
        System.out.println(a);
        System.out.println(b);
        Inner inr=new Inner();
        System.out.println(inr.ab);
    }
}

public class jtc128 {
    public static void main(String[] args) {
        //Outer.Inner oi1=new Outer().new Inner();
    }
}

```

```
}
```

When we are declaring any Inner class as private then it can only be accessed within that class where it has been declared and mainly Inner class is being designed for this kind of usage only.

Program 12.11

```
class Outer{
    interface I1{
        int a=10;
        void m1();
    }
    interface I2{
        static int b=20;
        public void m2();
    }
    interface I3 extends I1,I2{
        int c=11;
        void m3();
    }
    interface I4{
        int d=30;
        void m4();
    }
    interface I5{
        int e=40;
        void m5();
    }
    interface I6 extends I4,I5{
        int c=33;
        void m3();
    }
    abstract class Inner1 implements I3,I6{
        public void m1(){
            System.out.println("m1 in Inner1");
        }public void m2(){
            System.out.println("m2 in Inner1");
        }
        public void m4(){
```

```

        System.out.println("m4 in Inner1");
    }
    public void m5(){
        System.out.println("m5 in Inner1");
    }
    class Inner2 extends Inner1{
        public void m3(){
            System.out.println("m3 in Inner2");
        }
    }
} //close Inner2
} //close Inner1
}
public class Jtc129 {
    public static void main(String[] args) {
        Outer otr=new Outer();
        //Outer.Inner1 oi1=new Outer().new Inner1().new Inner2();
        //Outer.Inner1 oi1=new Outer().Inner1().new Inner2();
        Outer.Inner1 oi4=null;
        //oi4=oi4.new Inner2();
        /*Outer.Inner1.Inner2 oii2=otr.new Inner1(){
        }*/
    }
}

```

Static Inner Class

- A class which is defined inside another class with static keyword is called as static inner class.
- Inside static Inner class we can have Instance and static declaration both.
- Inside static Inner class only static member of outer class can be accessed.
- In order to create the object of static Inner class we need not to create any object of Outer class.

Program 12.12

```

class Outer{
    int a=10;
    static int b=20;
    void m1(){
        System.out.println("m1 in Outer");
    }
}

```

```

    }
    static void m2(){
        System.out.println("m2 in Outer");
    }
    static class Inner{
        int a11=101;
        static int b11=202;
        void m11(){
            System.out.println("m11 in Inner");
        }
        static void m22(){
            System.out.println("m22 in Inner");
        }
    }
} //close Inner
void show(){
    System.out.println("show in Outer");
    Inner inr =new Inner();
    inr.m11();
    inr.m22();
    Inner.m22();
}
}
}
public class Jtc130 {
    public static void main(String[] args) {
        //Outer.Inner oi1=new Outer().new Inner();
        //Outer.Inner oi2=Outer.new Inner();
        Outer.Inner oi3=new Outer.Inner();
        oi3.m11();
        Outer.Inner.m22();
        System.out.println(Outer.Inner.b11);
    }
}
}

```

Output:

```

D:\program\All JTC Program\Inner Class>java Jtc130
m11 in Inner
m22 in Inner
202

```


All the member of the static inner class will not be considered as a static member. It will execute successfully and has a .class file name Outer\$\$Outer\$Inner.

Q10: Can I declare Outer class as static?

Ans: No, we cannot declare.

Q11: Where exactly memory will be allocated for Instance and static variable of static inner classes.

Program 12.13

```
class Outer{
    int a=99;
    static int b=88;
    {
        System.out.println("Ib i Outer");
    }
    static{
        System.out.println("Sb in Outer");
    }
    static class Inner{
        int a1=101;
        static int b1=202;
        {
            System.out.println("Ib in Inner");
        }
        static{
            System.out.println("Sb in Inner");
        }
        static class Inner1{
            {
                System.out.println("Ib in Inner... Inner1");
            }
            static{
                System.out.println("Sb in Inner... Inner1");
            }
        }
    } //close Inner1
} //close Inner
}
```

```

public class Jtc131 {
    public static void main(String[] args) {
        Outer.Inner oi1=new Outer.Inner();
        //Outer.Inner oi2=new Outer().new Inner();
        Outer o1=new Outer();
        Outer.Inner.Inner1 oii1=new Outer.Inner.Inner1();
        //Outer.Inner.Inner1 oii2=new Outer().Inner().Inner1();
    }
}

```

Output:

```

D:\program\All JTC Program\Inner Class>java Jtc131
Sb in Inner
Ib in Inner
Sb in Outer
Ib i Outer
Sb in Inner... Inner1
Ib in Inner... Inner1

```

Method Local Inner Class

- A class which is defined inside any method or inside any local context are called as method local inner class.
- You can access all the members of Outer inside the method local inner class.
- Method local inner class members can be accessed inside the method only that too by creating the object of that method local inner class.
- Outside method, Method local inner class object cannot be created.
- Method local Inner class will not be processed until you are not creating the object of that class inside that method.

Program 12.14

```

class Outer{
    int a=10;
    static int b=20;
    void m1(){
        System.out.println("m1 in Outer");
        int aa=101;
        final int bb=202;
        class Inner{

```

```

        int ab=99;
        //static int bb1=88;
        {
            System.out.println("Ib in Inner");
        }
        void m2(){
            System.out.println("m2 in Inner");
            System.out.println(a);
            System.out.println(b);
            System.out.println(aa);
            System.out.println(bb);
            System.out.println(ab);
            //System.out.println(bb1);
        } //close m2() method
    } //close Inner class
    Inner inr=new Inner();
    System.out.println(inr.ab);
    inr.m2();
} //close m1() method
}
public class Jtc132 {
    public static void main(String[] args) {
        Outer otr=new Outer();
        otr.m1();
    }
}

```

Output:

```

D:\program\All JTC Program\Inner Class>java Jtc132
m1 in Outer
Ib in Inner
99
m2 in Inner
10
20
101
202
99

```

Program 12.15

```

interface I1{
    void m1();
}
class Outer{
    I1 mm(){
        class Inner implements I1{
            public void m1(){
                System.out.println("m1 in Inner");
            } //close m1() method
        } //close inner class
        return new Inner();
    } //close mm() method

    void show(){
        System.out.println("show in Outer");
        I1 i1=mm();
        i1.m1();
    }
}
public class Jtc133 {
    public static void main(String[] args) {
        Outer otr=new Outer();
        otr.show();
    }
}

```

Output:

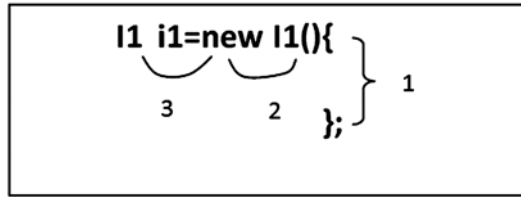
```

D:\program\All JTC Program\Inner Class>java Jtc133
show in Outer
m1 in Inner

```

Anonymous Inner Class:

- Anonymous Inner class is not having any name.
- Anonymous Inner class object can be created only once.
- Anonymous Inner class will always be the sub class of any Interface, abstract class or concrete class



Interface I1{

Void m1();

- Anonymous Inner class takes the following steps:
 - a. **Step 2:** Create the object of Anonymous Inner Class.
 - b. **Step 1:** Declaring the Anonymous Inner Class.
 - c. **Step 3:** Assigning the sub class object reference in the super class reference variable.

Q12: Is that object class is the super class for all inner classes also.

Ans: Yes, it is the super class of all types of Inner class.

Program 12.16

```
class Outer{
    class Inner{
    }
}
public class Jtc134 {
    public static void main(String[] args) {
        Outer otr=new Outer();
        Outer.Inner oinr=new Outer().new Inner();
        System.out.println(otr instanceof Object);
        System.out.println(oinr instanceof Object);
    }
}
```

Output:

```
D:\program\All JTC Program\Inner Class>java Jtc134
true
true
```

Q13: Suppose if you are getting requirement to use Interface and abstract class then which you should use and why?

Ans: It is always recommendable to use interface because when we are using Interface in our project implementation then chance are there to use multiple Inheritance properties then we can easily achieve that whereas with the help of abstract class we will not be able to achieve the multiple Inheritance.

Program 12.17

```
interface Animal{
    void eating();
    int sleeping();
}

class Hello{
    Animal ani=new Animal() {
        public int sleeping() {
            System.out.println("Animal in Hello sleeping");
            return 10;
        }
        public void eating() {
            System.out.println("Animal in Hello eating");
        }
        void walking(){
            System.out.println("walking in Hello");
        }
    };
    Animal ani2=new Animal() {
        public int sleeping() {
            System.out.println("Animal is sleeping in Hello");
            return 10;
        }
        public void eating() {
            System.out.println("Animal is eating in Hello");
        }
        {
            System.out.println("Ib in Animal2");
        }
        Animal ani3=new Animal(){
```

```

        public void eating(){
            System.out.println("Animal is eating in Hello");
        }
        public int sleeping(){
            System.out.println("Animal is sleeping in Hello");
            return 0;
        }
        {
            System.out.println("Ib in Animal 3");
        }
    };
};

void show1(){
    System.out.println("show1 in Hello");
    Animal ani4=new Animal(){
        public void eating(){
            System.out.println("Animal is eating in Show1");
        }
        public int sleeping(){
            System.out.println("Animal is sleeping in Show1");
            return 0;
        }
    };
    ani4.eating();
    ani4.sleeping();
}

Animal show2(){
    System.out.println("show2 in Hello");
    return new Animal() {
        public void eating(){
            System.out.println("Animal is eating in Show2");
        }
        public int sleeping(){
            System.out.println("Animal is sleeping in show2");
            return 10;
        }
    };
};
};

```

```

void showAll(){
    System.out.println("ShowAll in Hello");
    ani.eating();
    ani.sleeping();
    //ani.walkiing();
}
}

abstract class All{
    abstract void thinking();
    abstract Animal doingAll();
}

class Hai{
    All all=new All() {
        void thinking() {
            System.out.println("All thinking in Hai");
        }
        Animal doingAll() {
            System.out.println("All doingAll in Hai");
            return new Animal() {
                public int sleeping() {
                    System.out.println("All doingAll sleeping in Hai");
                    return 0;
                }
                public void eating() {
                    System.out.println("All doingAll eating in Hai");
                }
            };
        }
    };
}

public class Jtc135 {
    public static void main(String[] args) {
        Hello h1=new Hello();
        h1.showAll();
        System.out.println("*****In MAIN*****");
        h1.ani.eating();
        h1.ani.sleeping();
        h1.ani2.eating();
    }
}

```



```

        h1.ani2.sleeping();
        h1.show1();
        Animal ani1=h1.show2();
        ani1.eating();
        ani1.sleeping();
        Hai hai=new Hai();
        hai.all.thinking();
        Animal ani3=hai.all.doingAll();
        ani3.eating();
        ani3.sleeping();
    }
}

```

Output:

```

D:\program\All JTC Program\Inner Class>java Jtc135
Ib in Animal2
Ib in Animal 3
ShowAll in Hello
Animal in Hello eating
Animal in Hello sleeping
*****In MAIN*****
Animal in Hello eating
Animal in Hello sleeping
Animal is eating in Hello
Animal is sleeping in Hello
show1 in Hello
Animal is eating in Show1
Animal is sleeping in Show1
show2 in Hello
Animal is eating in Show2
Animal is sleeping in show2
All thinking in Hai
All doingAll in Hai
All doingAll eating in Hai
All doingAll sleeping in Hai

```

12.4 Adapter Class

Adapter class is a simple java class that implements an interface only with empty implementation for every method. If we implement an interface directly for each and every method compulsory we should provide implementation whether it is required or not. This approach increases length of the code and reduces readability.

Example

```

interface X{
void m1();

```

```

void m2();
void m3();
void m4();
//.
//.
//.
void m5();
}

```

Example

```

class Test implements X{
public void m3(){
System.out.println("m3() method is called");
}
public void m1(){}
public void m2(){}
public void m4(){}
public void m5(){}
}

```

- We can resolve this problem by using adapter class.
- Instead of implementing an interface if we can extend adapter class we have to provide implementation only for required methods but not for all methods of that interface.
- This approach decreases length of the code and improves readability.

Example

```

abstract class AdapterX implements X{
public void m1(){}
public void m2(){}
public void m3(){}
public void m4(){}
//
public void m1000(){}
}

```

Chapter 13

Package

13.1 What is Package?

In java package will be used to avoid the naming conflicts between the classes. In java, if you are using any package explicitly then by default your current working directory will be considered as default package.

Program 13.1

```
class Hello{
    void m1(){
        System.out.println("m1 in Hello");
    }
}
class Hai{
    void show(){
        System.out.println("show in Hai");
    }
}
public class Jtc138 {
    public static void main(String[] args) {
        Hello h1=new Hello();
        Hai hai=new Hai();
        h1.m1();
        hai.show();
    }
}
```

In above program Hai and Hello is not available inside the Jtc sources file. Even though it will not give any compilation error because Hai and Hello class are available inside the same folder Jtc, which will be considered as default package.

In java, java.lang is the built in default package whose members will be available for your class.

In java you can create your own package also depending on the requirement.

While creating the package remember the following points:

- Package must be the first starts of your program.
- In a source file you cannot have more than one package declaration.

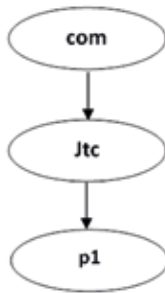
Syntax

package <package name>;

For e.g. :

package com.Jtc.p1;

It is not necessary to write the package name separated by the (.)Dot, but package name separated by (.) represents the sub directory under the main directory.



Program 13.2

```

package com.jtc.p1;
class Hello{
    void m1(){
        System.out.println("m1 in Hello");
    }
}
public class Jtc139 {
    public static void main(String[] args) {
        Hello h1=new Hello();
        h1.m1();
    }
}
  
```

Notes

- javac -d. Jtc139.java
- javac -d d:\package Jtc139.java

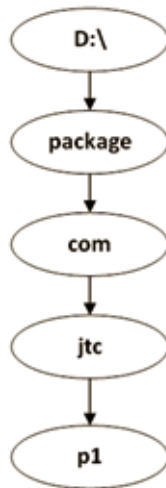
- `javac -d d:\package2 *.java`

With the `-d` folder structure is created according to the specified package (.) that represents the package structure, and it will be generated inside the current working directory.

When you are compiling package program in first way then folder structure will be generated inside the current working directory and only `Jtc139` will be compiled.

When you are compiling package programming in second way then the package structure will be generated inside the current working directory. The package structure will be generated at the specified location.

In case of third one all the program available inside current working directory will be compiled.



Program 13.3

```

package com.jtc.p12;
class Hai{
    public int b=20;
    public void show(){
        System.out.println("show in Hai");
        Hello h1=new Hello();
        h1.m1();
    }
}

```

Program 13.4

```

package com.jtc.p12;

public class Hello{
    public int a=10;
    public void m1(){
        System.out.println("m1 in Hello");
    }
}

```

Program 13.5

```

package com.jtc.p13;
import com.jtc.p12.*;
public class HelloHai{
    public int c=30;
    public void show(){
        Hai hai=new Hai();
        Hello h1=new Hello();
        System.out.println("show in Hai");
        System.out.println(h1.a);
        System.out.println(hai.b);
        System.out.println(c);
    }
}

```

Program 13.6

```

package com.jtc.p14;
import com.jtc.p12.*;
import com.jtc.p13.*;
public class Jtc140 {
    public static void main(String[] args) {
        Hello h1=new Hello();
        Hai hai=new Hai();
        HelloHai hh=new HelloHai();
        h1.m1();
        hai.show();
        hh.show();
    }
}

```

In order to run the program, we need to specify the fully qualified class name.

13.2 Import package

In order to use the existing package to same other program then in that case we need to declare the import statement. In a source file you can write number of import starts where as only one package declaration statement.

Syntax

Import package name;

Ex. Import com.jtc p1.hello

Import com.jtc.p1 *;

When you are writing the first import statement then from the package com.jtc p1 only hello class will be imported and in second import statement all the classes available inside com jtc p1 will be imported.

Program 13.7

```
package com.jtc.p1;
class Hello{
}
class Jtc57{
public static void main(String arg[]){
    System.out.println("Main in Jtc57");
}
}
package com.jtc.a1;
class A{
void m1(){
System.out.println("m1 in A");
}
}
package com.jtc.a1;
class B{
void m2(){
System.out.println("M2 in B");
}
}
package org.jtc.p1;
```

```

public class Hai{
    public void m2(){
        System.out.println("m2 in Hai");
    }
}

```

```

package org.jtc.p1;
public class Hello{
    public void m1(){
        System.out.println("m1 in Hello");
    }
    public static void main(String arg[]){
        System.out.println("main in Hello");
    }
}

```

```

package org.jtc.p2;
public class HelloA{
    public void m3(){
        System.out.println("m3 in HelloA");
    }
    public static void main(String arg[]){
        System.out.println("main in HelloA");
    }
}

```

```

package org.jtc.p4;
import org.jtc.p1.*;
import org.jtc.p2.HelloA;
class Jtc91{
    public static void main(String arg[]){
        Hello h1=new Hello();
        h1.m1();
        Hai hai=new Hai();
        hai.m2();
        HelloA ha=new HelloA();
        ha.m3();
        Hello.main(arg);
    }
}

```


Static Import

Static Import is introduced from jdk-1.5. By using the static import we can import only static member of a class.

Note

- a) `import static java.*;`
 - b) `import static java.lang.System.*;`
 - c) `import static java.lang.System.out;`
- I. When you are writing the first static import statement because from java.lang package it is trying to import static member & in a package. We can have many classes and a class cannot be static so it will give the compile time error.
 - II. When you are writing the second static import statement then all the static member of java.lang.System class will be available.
 - III. When you are writing the third static import statement then only out will be available for following class `import static java.lang.System.out`

Programs of Static Import

1) Hello.java

```
package com.Jtc.p1;
public class Hello {
    public static void m1() {
        System.out.println("-- m1() in com.Jtc.p1.Hello class --");
    }
    static void m2() {
        System.out.println("-- m2() in com.Jtc.p1.Hello class --");
    }
    public static void m3() {
        System.out.println("-- m3() in com.Jtc.p1.Hello class --");
    }
    public static void m4() {
```

```

System.out.println("*** m4 in com.Jtc.p1.Hello class ***");
}
public static void m5() {
System.out.println("== m5() in com.Jtc.p1.Hello class ==");
}
}

```

2) Abc.java

```

package com.Jtc.p2;

public class Abc {
public static void mm1() {
System.out.println("-- mm1() in com.Jtc.p2.Abc class --");
}
public static void mm2() {
System.out.println("-- mm2() in com.Jtc.p2.Abc class --");
}
static void m1Msg() {
System.out.println("-- m1Msg() in com.Jtc.p2.Abc class --");
}
public static void m5() {
System.out.println("~~ m5() in com.Jtc.p2.Abc class ~~");
}
}

```

3) Jtc1.java

```

package org.wb.test;
import static
com.Jtc.p1.Hello.m1; import
static com.Jtc.p1.Hello.m4;
import static com.Jtc.p2.Abc.*;

```

```

public class Jtc1 {
    public static void main(String[] args) {
        // Hello.m1();
        m1();
        m4();
        // m3();
        mm2();
        mm1();
        // m1Msg()
    }
    static void m4() {
        System.out.println("*** m4 in Test class **");
    }
}

```

4) Jtc2.java

```

package org.wb.test;
import static com.Jtc.p1.Hello.m1;
import static com.Jtc.p1.Hello.m4;
import com.Jtc.p1.Hello;

public class Jtc2 {
    public static void main(String[] args) {
        m1();
        m4();
        Hello.m4();
    }
    static void m4() {
        System.out.println("*** m4 in Test class **");
    }
}

```

```
}
```

5) Jtc3.java

```
package org.wb.test;
import static com.Jtc.p1.Hello.*;
import static com.Jtc.p2.Abc.*;

public class Jtc3 {
    public static void main(String[] args) {
        m1();
        m4();
        mm2();
        mm1();
        //m5();
    }
}
```

6) Jtc4.java

```
package org.wb.test;
import static
com.Jtc.p1.Hello.*; import
static com.Jtc.p2.Abc.*; import
com.Jtc.p1.Hello; import
com.Jtc.p2.Abc; import static
java.lang.System.*;

public class Jtc4 {
    public static void main(String[] args) {
        m1();
```

```

m4();
mm2();
mm1();
// m5();
Hello.m5();
Abc.m5();
out.println("Main Completed");
}
}

```

Static Import

When we are using the static import and if more than one classes are having same static member then in that case we need to access the static member by the fully qualified class name or else it will give the problem.

Createing the jar file

Create jar file

- `jar -cvf som.jar abc`
- `jar -cf som.jar *.java`

Extract jar file

- `jar -xvf som.jar`
- `jar -xf som.jar`

In order to create the jar file link represent that by what name you want to create jar file and the source file name is something which has to be included inside the jar file.

Here if you are writing cvf file createing the jar file, then it will show all the internal step which it is takeing to creating the jar file. If you are not writing 'v' then also jar file will be create but its will not show the steps which it is taking to create the jar file.

In order to extract the jar file we are writing xvf which show all the internal process which is it extrat even in this also you are not writing 'v' then it will not show the internal steps which it is the taking problem.

The jar file contains a collection of class file or package. The jar file can be use in your program in order to use to set the class path to there corresponding directory.

Note: Source file

- I. package declaration start only once.
- II. One or many import statement
- III. Only one public class
- IV. One or many default class.

Access Modifier: It can be use with the different class member. It specifies that which member and where can be access depending on the type of modifier used. There are only following type of access modifier.

- 1. private
- 2. default
- 3. protected
- 4. public

package com.p1

Aa.java
<pre>private int a=10; int b=20; protected int c=30; public int d=40; System.out.println(a); System.out.println(b); System.out.println(c); System.out.println(d);</pre>

BB.java	
System.out.println(a);//not ok	A a1=new A();
System.out.println(b);//not ok	System.out.println(a1.a);//not OK
System.out.println(c);//not ok	System.out.println(a1.b);//OK
System.out.println(d);//not ok	System.out.println(a1.c);//OK
	System.out.println(a1.d);//OK

CC extends AA.java	
System.out.println(a); //not OK	A a1=new A();
System.out.println(b);// OK	System.out.println(a1.a);//not OK
System.out.println(c);//OK	System.out.println(a1.b);//OK
System.out.println(d);//OK	System.out.println(a1.c);//OK
	System.out.println(a1.d);//OK

package com.p2

DD.java
A a1=new A();
System.out.println(a1.a);//not OK
System.out.println(a1.b);//not OK
System.out.println(a1.c);//not OK
System.out.println(a1.d);//OK

EE extends AA.java	
System.out.println(a); //not OK	A a1=new A();
System.out.println(b);// not OK	System.out.println(a1.a);//not OK
System.out.println(c);//OK	System.out.println(a1.b);//not OK
System.out.println(d);//OK	System.out.println(a1.c);//not OK
	System.out.println(a1.d);//OK

Chapter 14

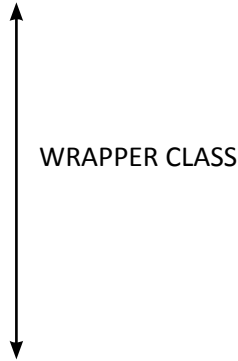
Java.Lang Package

14.1 Introduction

java.lang package is the default package which will automatically be available for your class without importing it. Even the members of java.lang package can be accessed directly without using any kind of import statement.

java.lang package contains many classes:

1. java.lang.Object
2. java.lang.String
3. java.lang.StringBuffer
4. java.lang.StringBuilder
5. java.lang.Byte
6. java.lang.Short
7. java.lang.Integer
8. java.lang.Long
9. java.lang.Float
10. java.lang.Double
11. java.lang.Boolean
12. java.lang.Character
13. java.lang.Class
14. java.lang.System
15. java.lang.Runtime
16. java.lang.Math



14.1.1 java.lang.Object class

Object is a class which is available inside java.lang package. It is by default a class e.g. without extending explicitly. It is the immediate super class for all types of classes in java by default. As it is the super class for all the class, all the member of object class can be accessed inside any class.

Object class contains the following members

Public class object{

1. public final native Class getClass();
2. public native int hashCode();
3. public Boolean equals(Object);
4. protected native Object clone() throws CloneNotSupportedException;
5. public String toString();
6. public final native void notify();
7. public final native void notifyAll();
8. public final native void wait(long) throws InterruptedException;
9. public final native void wait(long,int) throws InterruptedException;
10. public final void wait() throws InterruptedException;
11. protected void finalize() throws InterruptedException;

Used In Multi-Threading.

Why these methods are available in Object class?

public final native class getClass() method

- getClass() is a method which is available inside the java.lang.Object class. getClass() method is a native method which means it is a non-java programming implementation.
- And it cannot be overridden inside the subclass because it is a final method. It is public method which can be accessed inside or outside the package. That class method is used to get the class name which object reference is available inside corresponding reference variable.
- The return type of this method is java.lang.Class

Example

```
package com.p1;
class Hello {
}
class A {
}
class B extends A {
}
public class Jtc {
    public static void main(String arg[]) {
        System.out.println("in main");
        Hello h1 = new Hello();
        Class cls = h1.getClass();
    }
}
```

```

        System.out.println(cls);
        String cname = cls.getName();
        System.out.println(cname);
        Hello h2 = null;
        // System.out.println(h2.getClass().getNmae());
        System.out.println("*****\n");
        A a1 = new A();
        A a2 = new B();
        Object o = new String("abc");
        System.out.println(a1.getClass().getName());
        System.out.println(a2.getClass().getName());
        System.out.println(o.getClass().getName());
    }
}

```

public native int hashCode() method

- When you are creating any object, for all the objects hash code will be assigned internally by the JVM. On the basis of that hash code, JVM identifies each object uniquely. For each object there will be unique generation of hashCode(), and this method is mainly responsible to generate the hashCode () and retunes in form of integer. According to requirements you can override hashCode method inside your class and provide your own implementation to generate the hashCode().
- If you want to change the hash code generation algorithm then you can override the hashCode() method inside your class with the following signature by using your own algorithm:

Syntax

```

public int hashCode()
{
    own implementation
    return...;
}

```

The hashCode() method implementation can be given in two different ways:

- I. Always returns the unique hashCode.
- II. Static implementation which returns same hashCode again and again.

It is always recommendable to write the hashCode() method implementation in a such way that return a unique hashCode for each object.

public boolean equals(Object) method:

- Equals() method which is available inside java.lang.Object class. It is used to compare two objects in the following ways:
 - a) On the basis of reference
 - b) On the basis of contents
- If you are using equals(Object) method default implementation which is available inside java.lang.Object class, then it will compare on the basis of reference.
- But if you want to compare two Objects on the basis of contents then override the equals(Object) method in the class with your own implementation with the following **signature**:

```
public boolean equals(Object o)
{
return...;
}
```

Contract between equals() method and hashCode() method:

1. Two equivalent Object should be placed in the same bucket but all the object available in the same bucket may not be equals also.
2. Two equivalent Object: must have same hashCode() i.e. h.equals (h2) is true the h1.hashCode()==h2.hashCode() should return true.
3. If two objects are not equal by equals(..)method then there is no restriction on their hashCode, may be same or may not be same.
4. If hashCode of two object are equals then these object may or may not equal by .equals method.
5. If hashCode of two object are not equals then these object are always not equal by .equals method.

To satisfy above contract between equals() method and hashCode() method whenever we are overriding equals() method compulsory we have to override hashCode method otherwise we won't get any compile time error or runtime error but it is not a good programming practice.

Consider the following Code:

```

Class Hello{
Public boolean equals(Object o){
If(o instanceof Hello){
Hello h=(Hello)o;
If(name.equals(p.name)&& age==p.age)
Return true;
}else false;
Return false;
}

```

public String toString() method:

- toString() is the method which is available inside java.lang.Object class. It is used to give some information about the class. If you want to get the custom information about the class then follow the following signature:
 - o By default internal implementation of toString() method in a class is giving the hash code in the form of hexadecimal implementation.
 - o If you want to give some other information using toString() method then you can override the toString() method by writing your own implementation.

Example

```

public String toString(){
returnthis.getClass().getName()+"@"+Integer.toHexString(hashCode());
}

```

Example

```

package com.p2;
class Student {
    int sid;
    String name;
    String city;
    Student() {
    }
    Student(int sid, String name, String city) {
        this.sid = sid;
        this.name = name;
        this.city = city;
    }
}

```

```

    }
    void show() {
        System.out.println("show in Student");
        System.out.println(sid);
        System.out.println(name);
        System.out.println(city);
    }
    /*
    * public String toString(){
    * returnthis.getClass().getName()+"@"+Integer.toHexString(hashCode()); }
    */
    public String toString() {
        return "sid\t:" + sid + "Name\t:" + name + "city\t:" + city;
    }
    public int hashCode() {
        return 33;
    }
    public boolean equals(Object o) {
        Student s1 = (Student) o;
        if ((s1.sid == this.sid) && (s1.name.equals(this.name) && (s1.city.
equals(this.city))))
            return true;
        return false;
    }
}
public class Jtc1 {
    public static void main(String arg[]) {
        Student s1 = new Student(101, "Som", "noida");
        Student s2 = new Student(101, "Som", "noida");
        Student s3 = new Student(102, "Som", "noida");
        s1.show();
        s2.show();
        s3.show();
        System.out.println("*****\n");
        System.out.println(s1);
        System.out.println(s2.toString());
        System.out.println(s3);
        System.out.println("***COMPARE***\n");
    }
}

```

```

    /*
    * boolean bn=s1.equals(s2); System.out.println(bn);
    */
    System.out.println(s1.hashCode());
    System.out.println(s2.hashCode());
    System.out.println(s1.hashCode() == s2.hashCode());
    System.out.println(s1 == s2);
    System.out.println(s1.equals(s2));
    System.out.println(s1.equals(s3));
}
}

```

Garbage Collection

In C and C++ by using malloc and calloc we can allocate some memory inside heap but whenever it is required you can reclaim the memory allocated by free or delete function. In Java in order to perform memory cleaning process of unused object, it will be done automatically with the help of garbage collection. Then you are using any resources in java so immediately after the use that resource has to be closed or resource clean up has to be performed.

Resources cleanup can be done under the finalize() method by overriding it with the following signature:

```

    public void finalize(){
    }

```

Garbage Collection is a service thread which is used to provide the service to the main thread. Garbage Collection task is to reclaim the memory allocated by unused or un-referenced object. There are following ways in which JVM will identify the object that are unused or unreferenced object.

- i. When you are creating object inside any local context and then immediately after processing that local context object reference stored inside the unreferenced which will be eligible for Garbage Collection.

```

class Hello{
}
class Jtc{
    static void m1(){
        Hello h1=new Hello();
        Hello h2=new Hello();
    }
}

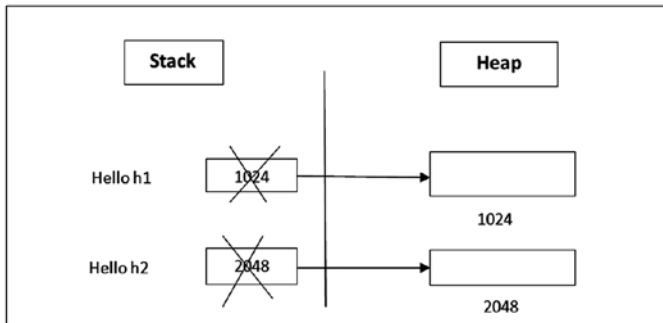
```

```

public static void main(String[] args){
    m1();
}
}

```

In above program once the m1() processing is completed, local reference variable h1 and h2 memory will be destroyed and then corresponding reference object will be unreferenced or unused and it is eligible for Garbage Collection.



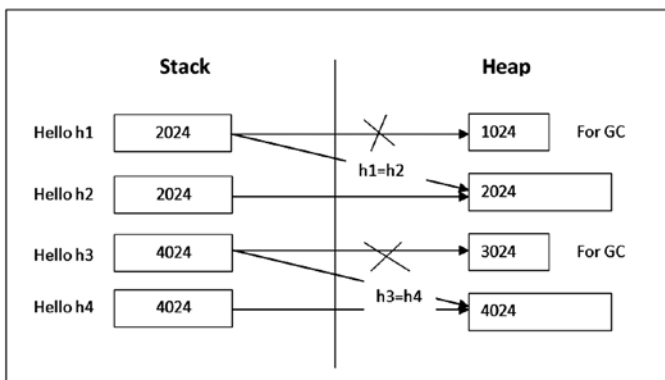
When you are creating the object and storing that object reference inside corresponding reference variable and later inside that reference variable some other object reference will be assigned so that previously referenced object will be unreferenced and is eligible for Garbage Collection.

Then the object which was being referenced by the left hand side referenced variable will be unreferenced.

```

class Hello{
}
class Jtc{

```



```

public static void main(String[] args){
    Hello h1=new Hello();
    Hello h2=new Hello();
    Hello h3=new Hello();
    Hello h4=new Hello();
    h1=h2;
    h3=h4;
}
}

```

When you are creating a object and in that reference variable we are assigning the null and then the object which is being referenced by that corresponding reference variable will eligible for Garbage Collection.

```

class Hello{
}
class Jtc{
    public static void main(String[] args){
        Hello h1=new Hello(); →for GC
        h1=null;
        new Hello(); →for GC
    }
}

```

Internal JVM task, JVM prepares a list of, unused or unreferenced object. Whenever JVM feel any heap memory related problem them it makes a call to Garbage Collection to reclaim or clean the memory allocated by unused or unreferenced object. While calling the Garbage Collection, JVM prepares the list of unused object and hand over to the gc(). gc() is trying to reclaim the memory allocated by unused object then in that some of the object may refuse gc() not to come with because that objects are holding some resources connection or file connection etc. so in order to reclaim the memory allocated by that corresponding objects we need to make sure that allocated resource is unused an resource cleanup.

JVM Task

- JVM prepares a list of, unused or unreferenced object.
- Whenever JVM feel any heap memory related problem them it makes a call to GC to reclaim or clean the memory allocated by unused or unreferenced object calls
 - System.runFinalization() ;

- `System.gc();`
- There is no guarantee that is when gc exactly will be call. `gc()` call depends on CPU scheduler.

System.gc()

`gc()` method is a static method which is available inside the `System` class. When you are calling or writing `System.gc()` explicitly then `gc()` will try to invoke garbage collector.

Cloning

Cloning is the process of creating the copy of existing object. Any Object can be clone in two different ways:

1. Shallow cloning / Shallow copy
2. Deep cloning / deep copy

When you are trying to clone any object then the corresponding class object must implement the **“Cloneable interface”**.

Shallow Cloning

- In the case of shallow cloning, only top-level objects will be cloned, and all the deep level objects will be used as it is. The default implementation of clone method available inside `Object` class is there for the shallow cloning which will clone object in the shallow manner. In order to clone any class object that corresponding class object must implement the **“Cloneable interface”**. The existing object and the cloned object, both must be having the different addresses (top-level only). (`Clone()` method is an instance method and cannot be called in static context.) (Fig. 14.1)

Example

```
package com.jtc2;
class A {
    int p;
    A(int p) {
        this.p = p;
    }
}
class B {
    int q;
```

```

    A a1;
    B(int q, A a1) {
        this.q = q;
        this.a1 = a1;
    }
}

class C {
    int r;
    B b1;
    C(int r, B b1) {
        this.r = r;
        this.b1 = b1;
    }
    {
        System.out.println("IB in C");
    }
}

class Hello implements Cloneable {
    int x;
    C c1;
    {
        System.out.println("IB in Hello");
    }
    Hello(int x,C c1){
        this.x=x;
        this.c1=c1;
    }

    public Object myClone() {
        Hello h1 = null;
        try {
            h1 = (Hello) this.clone();
        } catch (CloneNotSupportedException e) {
            e.printStackTrace();
        }
        return h1;
    }

    void show() {
        System.out.println("show in Hello");
    }
}

```

```

        System.out.println(x);
        System.out.println(c1.r);
        System.out.println(c1.b1.q);
        System.out.println(c1.b1.a1.p);
    }
}

public class Jtc1 {
    public static void main(String arg[]) {
        A a1 = new A(111);
        B b1 = new B(222, a1);
        C c1 = new C(333, b1);
        Hello h1 = new Hello(444, c1);
        // Hello h2=h1.clone();
        // Hello h2=clone();
        Hello h2 = (Hello) h1.myClone();
        System.out.println(h1 == h2);
        System.out.println(h1.c1 == h2.c1);
        System.out.println(h1.c1.b1 == h2.c1.b1);
        System.out.println(h1.c1.b1.a1 == h2.c1.b1.a1);
        h1.show();
        h2.show();
        System.out.println(h1.hashCode());
        System.out.println(h2.hashCode());
        System.out.println(h1.c1.hashCode());
        System.out.println(h2.c1.hashCode());
        System.out.println("*****\n");
        h1.show();
        h2.show();
        h1.x = 9999;
        h1.c1.r = 8888;
        System.out.println("*****\n");
        h1.show();
        h2.show();
    }
}

```

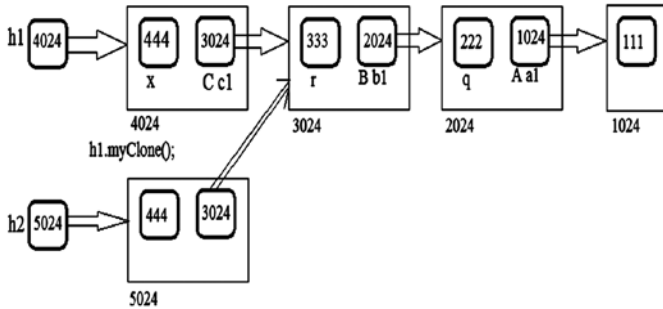


Fig. 14.1: Shallow cloning

Deep cloning

In the case of deep cloning, all the top level objects and deep level objects will be cloned. The `clone()` method available inside object class is for the shallow cloning. (Fig. 14.2) If you want to implement deep cloning then you must have to override your own implementation of the `clone()` method with following signature:

```
public Object clone(){
    return...;
}
```

Example

```
package com.jtc3;
class A {
    int p;
    A(int p) {
        this.p = p;
    }
}
class B {
    int q;
    A a1;
    B(int q, A a1) {
        this.q = q;
        this.a1 = a1;
    }
}
```

```

class C {
    int r;
    B b1;
    C(int r, B b1) {
        this.r = r;
        this.b1 = b1;
    }
    {
        System.out.println("IB in C");
    }
}

class Hello implements Cloneable {
    int x;
    C c1;
    {
        System.out.println("IB in Hello");
    }
    Hello(int x,C c1){
        this.x=x;
        this.c1=c1;
    }
    public Object clone() {
        Hello h1 = null;
        Class cls[] = this.getClass().getInterfaces();
        try {
            if (cls[0].getName().equals("java.lang.Cloneable")) {
                A a1 = new A(c1.b1.a1.p);
                B b1 = new B(c1.b1.q, a1);
                C c11 = new C(c1.r, b1);
                h1 = new Hello(this.x, c11);
            }
        } catch (Exception e) {
        }
        return h1;
    }
}

```

```

void show() {
    System.out.println("show in Hello");
    System.out.println(x);
    System.out.println(c1.r);
    System.out.println(c1.b1.q);
    System.out.println(c1.b1.a1.p);
}
}

public class Jtc1 {
    public static void main(String arg[]) {
        A a1 = new A(111);
        B b1 = new B(222, a1);
        C c1 = new C(333, b1);
        Hello h1 = new Hello(444, c1);
        Hello h2 = (Hello) h1.clone();
        System.out.println(h1 == h2);
        System.out.println(h1.c1 == h2.c1);
        System.out.println(h1.c1.b1 == h2.c1.b1);
        System.out.println(h1.c1.b1.a1 == h2.c1.b1.a1);
        h1.show();
        h2.show();
        System.out.println(h1.hashCode());
        System.out.println(h2.hashCode());
        System.out.println(h1.c1.hashCode());
        System.out.println(h2.c1.hashCode());
        System.out.println("*****\n");
        h1.show();
        h2.show();
        h1.x = 9999;
        h1.c1.r = 8888;
        System.out.println("*****\n");
        h1.show();
        h2.show();
    }
}

```

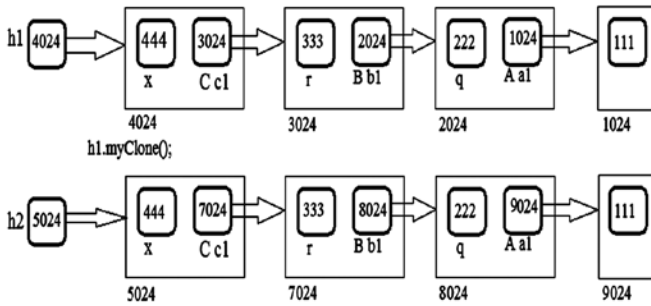


Fig. 14.2: Deep Cloning

Example

```
package com.jtcindia.p1;
public class Jtc1 {
    public static void main(String arg[]) throws CloneNotSupportedException {
        Employee emp1 = new Employee(88, "Manish");
        emp1.showClone();
        System.out.println("\n-----");
        LoginInfo log = new LoginInfo(101, "somsree", "JTCIndia");
        Address ad = new Address("c-29", "noida", 201301);
        Student st = new Student(999, "somPrakash", 6526668, ad, log);
        System.out.println(st);
        Student stud = (Student) st.clone();
        System.out.println("\n*****After Clonign the Object*****");
        System.out.println(st == stud);
        System.out.println(st.studAdd == stud.studAdd);
        System.out.println(st.login == stud.login);
        System.out.println(st);
        System.out.println(stud);
        System.out.println("\n===MODIFYING THE DATA===");
        stud.sid = 909090;
        stud.name = "Manish";
        stud.phone = 557628;
        stud.studAdd.street = "delhi";
        stud.login.username = "JTCuser";
        System.out.println(st);
        System.out.println(stud);
    }
}
```

```

}
class Student implements Cloneable {
    int sid;
    String name;
    long phone;
    Address studAdd;
    LoginInfo login;
    public Student(int sid, String name, long phone, Address studAdd, LoginInfo
login) {
        this.sid = sid;
        this.name = name;
        this.phone = phone;
        this.studAdd = studAdd;
        this.login = login;
    }
    public String toString() {
        return "\nStud Info\t:" + sid + "\t" + name + "\t" + phone + "\t" + "\nAdd Info\t:" + studAdd + "\nLogin Info\t:" + login;
    }
    public Object clone() throws CloneNotSupportedException {
        Object obj = null;
        if (this instanceof Cloneable) {
            Address ad = new Address(this.studAdd.aid, this.studAdd.street,
this.studAdd.pin);
            LoginInfo info = new LoginInfo(this.login.loginId, this.login.
uname, this.login.pwd);
            obj = new Student(this.sid, this.name, this.phone, ad, info);
        } else {
            throw new
CloneNotSupportedException(this.getClass().getName());
        }
        return obj;
    }
}
class Employee implements Cloneable {
    int eid;
    String name;
    public Employee(int eid, String name) {

```



```

        this.eid = eid;
        this.name = name;
        System.out.println("----Employee(int,String)----");
    }
    public String toString() {
        return eid + "\t" + name;
    }
    void showClone() throws CloneNotSupportedException {
        Employee ep = (Employee) clone();
        System.out.println("this\t\t:" + this);
        System.out.println("Cloned Obj\t" + ep);
        System.out.println(this == ep);
    }
}

class Address {
    String aid;
    String street;
    int pin;
    public Address(String aid, String street, int pin) {
        super();
        this.aid = aid;
        this.street = street;
        this.pin = pin;
    }
    public String toString() {
        return aid + "\t" + street + "\t" + pin;
    }
}

class LoginInfo {
    int loginId;
    String uname;
    String pwd;
    public LoginInfo(int loginId, String uname, String pwd) {
        this.loginId = loginId;
        this.uname = uname;
        this.pwd = pwd;
    }
    public String toString() {

```

```

        return loginId + "\t" + uname + "\t" + pwd;
    }
}

```

Reflection

- Reflection is a process of analyzing the class without seeing them.
- By using reflection API which has been given by sun microsystem. i.e. java.lang.reflect API. We can come to know most of the class level properties like classes, interfaces, fields, and methods at the runtime without knowing the name of corresponding entity and operation at compile time. Here entity relates to a class, operation relates to a methods. Also we can instantiate the new objects invoke the methods and inject the values.
- Reflection API is the very powerful, which is not having use in general programming or in basic programming but based on reflection many concept of java and J2EE has been design. There are many framework and tools which uses the reflection properly.
 - ✓ Struts
 - ✓ Hibernate
 - ✓ Spring
 - ✓ JUnit
 - ✓ Tomcat
 - ✓ Eclipse
 - And many more...
- It is recommendable not to use reflection in general programming because of some issues.
 - ✓ Poor performance
 - ✓ Security Restriction
 - ✓ Security Issue
 - ✓ High Maintenance

Program 14.1

```

package com.jtc;
import java.lang.reflect.*;

```

```

public class Jtc1 {
public static void main(String[] args) {
try{
Hello hh=new Hello();
hh.show();
Class
cls=Class.forName("com.jtc.Hello");
Hello h=(Hello)cls.newInstance();
h.show();
System.out.println(cls);
System.out.println(cls.getName());
System.out.println(cls.getPackage());
System.out.println(cls.getPackage().getName(
)); System.out.println(cls.getSuperclass());
System.out.println(cls.getSuperclass().getNa
me()); Class c[]=cls.getInterfaces();
for(int i=0;i<c.length;i++)
{ System.out.println(c[i].getName());
}
System.out.println(cls.getModifiers());
System.out.println(Modifier.PUBLIC);
System.out.println(Modifier.FINAL); }
catch(Exception e){
e.printStackTrace();
}}}
interface i1{}
interface i2{}
class A{}
final class Hello extends A implements i1,i2{
int a=10;
int b;
Hello(){ }
Hello(int b){
this.b=b;
}
void show(){
System.out.println(a);
System.out.println(b);
}
}

```

```
}}
```

Program 14.2

```
package com.jtc;
import java.lang.reflect.*;

public class Jtc2 {
    public static void main(String[] args) {
        try{
            Class cls=Class.forName("com.jtc.Hello");
            Hello h=(Hello)cls.newInstance();
            h.show();
            System.out.println(Modifier.PUBLIC);
            System.out.println(Modifier.FINAL);
            System.out.println(Modifier.STATIC);
            // fields
            System.out.println("fields");
            Field f[]=cls.getFields();
            for(int i=0;i<f.length;i++){ System.out.println(f[i].getModifiers()
            + "..."+f[i].getType()+"..." +f[i].getName());
            }
            System.out.println("Declared fields");
            // Declared fields
            Field ff[]=cls.getDeclaredFields();
            for(int i=0;i<ff.length;i++){ System.out.println(ff[i].getModifiers()
            + "..."+ff[i].getType()+"..." +ff[i].getName()); }
            // methods
            System.out.println("methods");
            Method m[]=cls.getMethods();
            for(int i=0;i<m.length;i++){ System.out.println(m[i].getModifiers()
            + "..."+m[i].getReturnType()+"..." +m[i]. getName());
            }
```

```

}
// Declared methods
System.out.println("Declared methods");
Method mm[]=cls.getDeclaredMethods();
for(int i=0;i<mm.length;i++){
System.out.println(mm[i].getModifiers()+"..." +mm[i].
getReturnType()+"..." +mm[i].getName());
}
// constructors
System.out.println("constructors");
Constructor c[]=cls.getConstructors();
for(int i=0;i<c.length;i++){
System.out.println(c[i].getModifiers()+"..." +c[i].getName());
}
//invoking methods dynamically
mm=cls.getDeclaredMethods();
for(int i=0;i<mm.length;i++){
if(mm[i].getName().equals("show"))
mm[i].invoke(h,null);
}
}catch(Exception e){
e.printStackTrace();
}}
}
class Hai {
public int x=200;
public void hai(){
System.out.println("hai");
}
}
class Hello extends Hai{
public static final int a=10;
public static int b=20;
public Hello(){ }
public void show(){
System.out.println("show");
System.out.println(a);
System.out.println(b);
}
}
public void m1(String s1,String s2){

```

```

System.out.println("m1");
System.out.println(s1);
System.out.println(s2);
}
public void m2(int x,int y){
System.out.println("m2");
System.out.println(x);
System.out.println(y);
}}

```

Program 14.3

```

package com.jtc;
import java.lang.reflect.*;

public class Jtc3 {
public static void main(String[] args) {
try{
Class cls=Class.forName("com.jtc.Hello");
Hello h=(Hello)cls.newInstance(); //
invoking methods dynamically
Method []mm=cls.getDeclaredMethods();
for(int i=0;i<mm.length;i++)
{ if(mm[i].getName().equals("show"))
{ mm[i].invoke(h,null);
}
if(mm[i].getName().equals("m1")){
Object o[]={ "jtc1", "jtc2"};
mm[i].invoke(h,o);
}
if(mm[i].getName().equals("m2")){
Object o[]={new Integer(99),new
Integer(88)}; mm[i].invoke(h,o);
}
}
}
}

```

```

}
}catch(Exception e){
e.printStackTrace();
}}
class Hello {
public void show(){
System.out.println("show");
}
public void m1(String s1,String s2){
System.out.println("m1");
System.out.println(s1);
System.out.println(s2);
}
public void m2(int x,int y){
System.out.println("m2");
System.out.println(x);
System.out.println(y);
}
}
}

```

Program 14.4

```

import java.lang.reflect.*;
import java.io.*;

public class Jtc4{
public static void main(String str[]) throws Exception{
Class cls=Class.forName("Student");
System.out.println("**** INTERFACES ****");
Class inters[]=cls.getInterfaces();
for(int i=0;i<inters.length;i++)
{ System.out.println(inters[i]);
}
System.out.println("**** PUBLIC CONSTRUCTOR
****");
}
}

```

```

Constructor pubCons[]=cls.getConstructors();
for(int i=0;i<pubCons.length;i++){
System.out.println(pubCons[i]);
}
System.out.println("**** Declared CONSTRUCTOR ****");
Constructor decCons[]=cls.getDeclaredConstructors();
for(int i=0;i<decCons.length;i++){
System.out.println(decCons[i]);
}
System.out.println("**** PUBLIC FIELD ****");
Field pubFlds[]=cls.getFields();
for(int i=0;i<pubFlds.length;i++){
System.out.println(pubFlds[i]);
}
System.out.println("**** DECLARED FIELD ****");
Field decFlds[]=cls.getDeclaredFields();
for(int i=0;i<decFlds.length;i++){
System.out.println(decFlds[i]);
}
System.out.println("**** PUBLIC INNER CLASSES ****");
Class innClass[]=cls.getClasses();
for(int i=0;i<innClass.length;i++){
System.out.println(innClass[i]);
}}}}
interface Inter1{}
interface Inter2{}
interface Inter3{}
class User{
public String username;
String password;
public String name;
public class Ab{}
class Bc{}
}
class Student extends User implements Inter1,Inter2,Inter3,Cloneable,Serializable{
private String batchId;
public String batchTimings;

```



```

int sid;
Student(int ab){}
private Student(int ab,String nm){}
public Student(int ab,long phone){}
protected Student(int ab,boolean registered){}
public Student(int ab,String nm,long phone){}
protected Student(String nm,float fee){}
Student(String nm){}
class A{}
public class B{}
}

```

Program: Access Method Information

```

import java.lang.reflect.*;
import java.io.*;

public class Jtc5{
public static void main(String str[]) throws Exception{
Class cls=Class.forName("Student");
Object st=new Student();
Object test=new RefTest5();
System.out.println(cls.isInstance(st));
System.out.println(cls.isInstance(test));
Method methods[]=cls.getDeclaredMethods();
for(int i=0;i<methods.length;i++){
Method m=methods[i];
System.out.println("\n=====");
System.out.println(m); System.out.println("***NAME***");
\t:"+m.getName());
Class retType=m.getReturnType();
System.out.println("***RET TYPE***");
\t:"+retType.getName()); System.out.println("Interface");
\t:"+retType.isInterface()); System.out.println("Primitive");
\t:"+retType.isPrimitive());

```

```

System.out.println("Arrays\t:" + retType.isArray());
Class params[] = m.getParameterTypes();
for (int j = 0; j < params.length; j++) {
    System.out.print(params[j] + " , ");
}
int x = m.getModifiers();
System.out.println("\nPUBLIC\t:" + Modifier.isPublic(x));
System.out.println("FINAL\t:" + Modifier.isFinal(x));
System.out.println("STATIC\t:" + Modifier.isStatic(x));
System.out.println(Modifier.toString(x));
}}
}

class Student {
    public synchronized void show(int ab, String name, boolean valid, long phone ) {
    }

    static final public String getName(int id, String email) {
        return "";
    }

    public int showInformation(String email) {
        return 0;
    }

    int[] getStudentIds() {
        return null;
    }
}

```

Program: Invoke the Method Dynamically

```

package com.jtc.ref.obj;

import java.io.*;
import java.lang.reflect.*;

public class Jtc6 {
    public static void main(String[] args) throws Exception {
        if (args.length == 1) {

```

```

String cName = args[0];
Class cl1 = Class.forName(cName);
DataInputStream dis = new DataInputStream(System.in);
Method ms[] = cl1.getDeclaredMethods();
for (int i = 0; i < ms.length; i++) {
    Method m = ms[i];
    System.out.println("\nCalling the method " + m.getName());
    Object objReq = null;
    int mod = m.getModifiers();
    boolean st = Modifier.isStatic(mod);
    if (!st) {
        objReq = cl1.newInstance();
    }
    Class params[] = m.getParameterTypes();
    Object[] reqArgs = new Object[params.length];
    for (int j = 0; j < params.length; j++) {
        String type = params[j].getName();
        System.out.println("Enter Value of type\t:" + type);
        String val = dis.readLine();
        if (type.equals("boolean"))
            reqArgs[j] = new Boolean(val);
        else if (type.equals("byte"))
            reqArgs[j] = new Byte(val);
        else if (type.equals("char"))
            reqArgs[j] = new Character(val.charAt(0));
        else if (type.equals("short"))
            reqArgs[j] = new Short(val);
        else if (type.equals("int"))
            reqArgs[j] = new Integer(val);
        else if (type.equals("long"))
            reqArgs[j] = new Long(val);
        else if (type.equals("float"))
            reqArgs[j] = new Float(val);
        else if (type.equals("double"))
            reqArgs[j] = new Double(val);
        else if (type.equals("java.lang.String"))
            reqArgs[j] = val;
    }
}

```

```

Object res = m.invoke(objReq, reqArgs);
System.out.println("Result is\t:" + res);
}
} else {
System.out.println("Provide fully qualified class name & Method Name as CLA");
}}}
class Student {
private void showDetails() {
System.out.println("-- showDetails () of Student\t:" + this);
}
static int searchId(int ab, String bc, boolean b1) {
System.out.println("-- searchId () of Student\t:");
System.out.println("ab\t:" + ab);
System.out.println("bc\t:" + bc);
System.out.println("b1\t:" + b1);
return 99;
}
boolean verifyStudent(char ch, String bc, boolean b1, long val, float f1) {
System.out.println("-- verifyStudent () of Student\t:" + this);
System.out.println("ch\t:" + ch);
System.out.println("bc\t:" + bc);
System.out.println("b1\t:" + b1);
System.out.println("val\t:" + val);
System.out.println("f1\t:" + f1);
return true;
}
}
}

```

Program: Creating the Object Dynamically using Various Constructor

```

package com.jtc.ref.obj;
import java.io.*;
import java.lang.reflect.*;
public class Jtc7 {
public static void main(String[] args) throws Exception {
if (args.length != 1) {
System.out.println("Provide the fully qualified class name as CLA");
System.exit(0);
}
}
}

```

```

DataInputStream dis = new DataInputStream(System.in);
String className = args[0];
Class cl = Class.forName(className);
Constructor cons[] = cl.getConstructors();
for (int k = 0; k < cons.length; k++) {
    Constructor c = cons[k];
    Class cParams[] = c.getParameterTypes();
    Object objArg[] = new Object[cParams.length];
    for (int i = 0; i < cParams.length; i++) {
        Class p = cParams[i];
        String type = p.getSimpleName();
        System.out.println("Enter value of type\t:" + type);
        String val = dis.readLine();
        if (type.equals("boolean")) {
            objArg[i] = new Boolean(val);
        } else if (type.equals("byte")) {
            objArg[i] = new Byte(val);
        } else if (type.equals("short")) {
            objArg[i] = new Short(val);
        } else if (type.equals("char")) {
            objArg[i] = new Character(val.charAt(0));
        } else if (type.equals("int")) {
            objArg[i] = new Integer(val);
        } else if (type.equals("long")) {
            objArg[i] = new Long(val);
        } else if (type.equals("float")) {
            objArg[i] = new Float(val);
        } else if (type.equals("double")) {
            objArg[i] = new Double(val);
        } else if (type.equals("String")) {
            objArg[i] = val;
        } else {
            objArg[i] = p.newInstance();
        }
    }
    Object obj = c.newInstance(objArg);
    System.out.println(obj);
}
}
}

```

2) Employee.java

```

package com.jtc.ref.obj;
public class Employee {
    private int eid;
    private String eName;
    private String email;
    private long phone;
    private boolean permanent;
    private Address empAdd;
    public Employee(int eid, String email, long phone, Address empAdd) {
        this.eid = eid;
        this.email = email;
        this.phone = phone;
        this.empAdd = empAdd;
    }
    public Employee(int eid, String eName, String email, long phone,
        boolean permanent, Address empAdd) {
        this.eid = eid;
        this.eName = eName;
        this.email = email;
        this.phone = phone;
        this.permanent = permanent;
        this.empAdd = empAdd;
    }
    public String toString() {
        return "Employee [eid=" + eid + ", eName=" + eName + ", email=" + email + ",
        phone=" + phone + ", permanent=" + permanent + ", empAdd=" + empAdd + "]";
    }
}

```

3) Address.java

```

package com.jtc.ref.obj;
public class Address {
    public String toString() {
        return "Emp Address";
    }
}

```

14.1.2 java.lang.String class

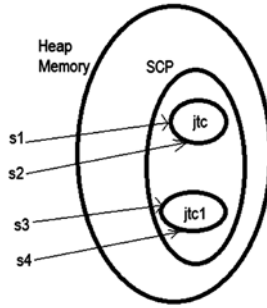
- java.lang.String is not a data type , it is a special class which is available inside java.lang package..
- String is a final class which cannot be extended to some other class.
- String objects are immutable in nature in which any change cannot be done.
- Whenever you concatenate any string with another string that will always create a new object.
- String uses special memory location, i.e., String-Constant-Pool for the reusability of String objects
- String object can be created in two different ways:
 1. Without new operator
 2. With new operator

Creating a String object without new operator

- When we are creating a String object without new operator then first it will check whether that string literal is available inside string-constant-pool or not.
- If it is not available inside the string-constant-pool then it will create a new string literal inside the string-constant-pool and it will point to the corresponding reference variable.
- If that string literal is already available inside that string-constant-pool, then it is not going to create a new string literal and it will point to the same.

Example

```
package com.jtc4;
public class Jtc124{
public static void main(String arg[]){
String s1="jtc";
String s2="jtc";
String s3="jtc1";
String s4="jtc1";
    System.out.println(s1==s2); //true
    System.out.println(s1==s3); //false
    System.out.println(s3==s4); //true
}}
```



Creating a String object with new operator

- When you are creating String object with new operator then first it will check inside the SCP whether that string literal is available inside SCP or not.
- If it is not available inside SCP, it will create one new literal with SCP and then it comes out of the SCP and creates one new literal outside the SCP and will point to that.
- If the String literal is available inside SCP (String-Constant-Pool), then it will not create any new literal inside SCP; it comes out of the SCP and creates a String literal inside the Heap and will point to that;

Example

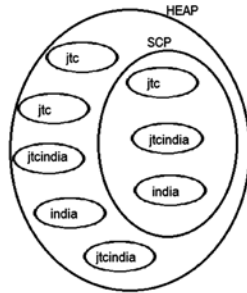
```
package com.jtc5;

public class Jtc125{
    public static void main(String arg[]){
        String s1=new String("jtc");
        String s2=new String("jtc");
        String s3=new String("jtcindia");
        String s4=new String("india");
        String s5=s1+s4;
        System.out.println(s1==s2);
        System.out.println(s2==d3);
        System.out.println(s3==s5);
    }
}

package com.Jtc6;
public class Jtc126{
    public static void main(String arg[]){
        String s1="jtc";
```



```
String s2="jtc";
String s3="jtcnoida";
String s4=new String("jtc");
```



FORMULA:

Reference + Reference = false	(outside SCP)
Reference + Literal = false	(outside SCP)
Literal + Reference = false	(outside SCP)
Literal+Literal = true	(inside SCP)

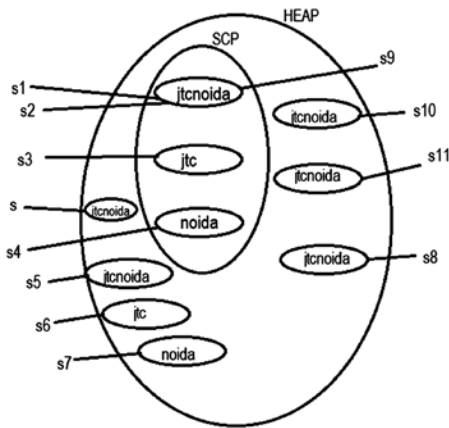
```
String s5=new String("jtcnoida");
System.out.println(s1==s2);
System.out.println(s1==s3);
System.out.println(s3==s5);
}
}
```

```
package com.jtc7;
public class Jtc127{
    public static void main(String arg[]){
        String s=new String("jtcnoida");
        String s1="jtcnoida";
        String s2="jtcnoida";
        String s3="jtc";
        String s4="noida";
        String s5=s3+s4;
        String s6=new String("jtc");
        String s7=new String("noida");
        String s9="jtc"+"noida";
```

```

String s10="jtc+s4;
String s11=s3+"noida";
String s8=s6+s7;
System.out.pritnln(s==s1);
System.out.pritnln(s1==s2);
System.out.pritnln(s1==s5);
System.out.pritnln(s5==s8);
System.out.pritnln("****CHECKING****");
System.out.pritnln(s1==s9);
System.out.pritnln(s1==s10);
System.out.pritnln(s1==s11);
System.out.pritnln(s5==s8);
}}

```



intern() method

- intern() method is used to point the string literal inside the String-Constant-Pool, if that string literal is available inside the SCP.
- If it is not available then it will create a new literal inside the pool and it will point to that.

Example

```

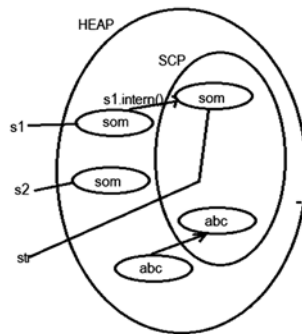
package com.jtc8;
public class Jtc128{
public static void main(String arg[]){
String s1=new String("som");

```

```

String s2=new String("som");
String str="som";
System.out.println(s1==s2);
System.out.println(s1==str);
System.out.println(str==s1.intern());
System.out.println("abc"==new String("abc").intern());
System.out.println(new String("abc").intern()=="abc".intern());
System.out.println(s1==s2.intern());
System.out.println("*****");
System.out.println("Checking:"+s2.intern()==str);
}
}

```



Writing immutable class

Immutable class means we should not be able to create or modify any existing object in order to create any class immutable declare that class as final and all the field of that class should be private.

To create immutable class do the following:

- (a) Final for class declaration
 - (b) Declare its members as private
- Once we created an object we can't perform any changes in the existing object.
 - If we are trying to perform any changes with those changes a new object will be created. If there is no change in the content then existing object will be reused. This behavior is called immutability.

Program 14.5

```

final class CreateImmutable{
private int i;
CreateImmutable(int i){
this.i=i;
}
public CreateImmutable modify(int i){
if(this.i==i)
return this;
else
return (new CreateImmutable(i));
}
public static void main(String[] args){
CreateImmutable c1=new CreateImmutable(10);
CreateImmutable c2=c1.modify(100);
CreateImmutable c3=c1.modify(10);
System.out.println(c1==c2); //false
System.out.println(c1==c3);//true
CreateImmutable c4=c1.modify(100);
System.out.println(c2==c4);//false }
}

```

- Once we create a CreateImmutable object we can't perform any changes in the existing object, if we are trying to perform any changes with those changes a new object will be created.
- If there is no change in the content then existing object will be reused

Example

```

package com.jtc9;
final class Student {
    private final int sid;
    private final String name;
    private final long phone;
    private final Address stuAdd;
    public Student(int sid, String name, long phone, Address stuAdd) {
        this.sid = sid;
        this.name = name;
        this.phone = phone;
        this.stuAdd = new Address(stuAdd.aid, stuAdd.street, stuAdd.city,

```

```

stuAdd.pin);
    }
    public String toString() {
        return "Stud Info\t:" + sid + "\t" + name + "\t" + phone + "\n Addr
Info\t:" + stuAdd;
    }
    public int getSid() {
        return this.sid;
    }
    public Address getStudAddress() {
        return (Address) stuAdd.clone();
    }
    class Address {
        int aid;
        String street;
        String city;
        int pin;
        public Address(int aid, String street, String city, int pin) {
            this.aid = aid;
            this.street = street;
            this.city = city;
            this.pin = pin;
        }
        public Object clone() {
            return new Address(this.aid, this.street, this.city, this.pin);
        }
        public String toString() {
            return aid + "\t" + street + "\t" + city + "\t" + pin;
        }
    }
}
public class Jtc1 {
    public static void main(String arg[]) {
        Address add = new Address(101, "c29Sector2", "noida", 201301);
        Student stud = new Student(99, "SomPrakash", 6526668, add);
        System.out.println(stud);
        System.out.println("--modifying the address reference in main--");
        // stud.studAddr.add=1234;
    }
}

```

```

        add.aid = 145673;
        add.street = "sector15";
        System.out.println(stud);
        System.out.println(stud.getSid());
        Address ref = stud.getStudAddress();
        ref.aid = 1111;
        // Student stud2=new Student(sid,name,phone,studAddr);
    }
}

```

- hashCode() method inside the String class had been overridden on the basis of ASCII value.
- equals() method has been overridden on the basis of contents.

Final Vs Immutability

- final modifier applicable for variables where as immutability concept applicable for objects
- If reference variable declared as final then we can't perform reassignment for the reference variable it doesn't mean we can't perform any change in that object.
- That is by declaring a reference variable as final we won't get any immutability nature.
- final and immutability both are different concepts .

Example

```

class Hello{
public static void main(String[] args){
final StringBuffer sb=new StringBuffer("java");
sb.append("training");
System.out.println(sb);//javatraining
sb=new StringBuffer("center");//C.E: cannot assign a value to final
    variable sb
}}

```

In the above example even though "sb" is final we can perform any type of change in the corresponding object. That is through final keyword we are not getting any immutability nature.

Q1: Which of the following are meaningful?

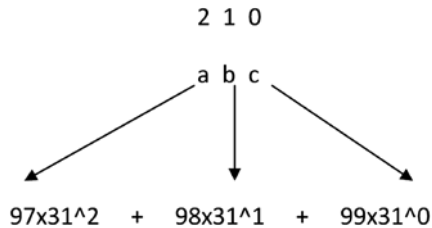
- final variable (valid)

- final object (invalid)
- immutable variable (invalid)
- immutable object (valid)

Example

String s1="abc";

String s2="abc";



s1.equals(s2) → true

s1.hashCode()==s2.hashCode() → true

Example

```

package com.p3;
class Hello extends Object{
}
public class Jtc130{
public static void main(String arg[]){
String s1="jtc";
String s2="jtc";
String s3=new String("jtc");
Hello h1=new Hello();
String s4="abc";
String s5=new String("abc");
System.out.println(s1);
System.out.println(h1);
//System.out.println(s1==s2);
System.out.println(s1.equals(s2));
System.out.println(s1.hashCode()==s2.hashCode());
System.out.println(s4.hashCode());
System.out.println(s5.hashCode());
System.out.println(s5.equals(s4));
}
}

```

```

System.out.println("*****\n");
String s11="aaBBcddef";
System.out.println(s11.matches("a*Bcd*ef")); //false
System.out.println(s11.matches("a*B*c*d*ef")); //true
System.out.println(s11.matches("a*B*c*d*e*f*")); //true
String s12="Hi I am in Java Training Center, sleeping in core Java class;
System.out.println(s12.indexOf("Java"));
System.out.println(s12.lastIndexOf("Java"));
System.out.println(s12.indexOf("Java",16));
System.out.println("****java\n");
String str1="jtcindia";
System.out.println(str1.regionMatches(0,"jtcindia",0,8)); //true
System.out.println(str1.regionMatches(3,"india",0,5) //true
System.out.println(str1.regionMatches(true,3,"india",0,5)); //true
}
}

```

compareTo() method

- This method is used to compare two strings character by character on the basis of its ASCII value.
- If any one of the characters is not matching then it will return the difference of ASCII value and further comparisons will not be done.
- (It will return 0 when no difference is found; when it finds the first difference then further characters are not compared.)
- The return type of compareTo() method is available inside "java.lang.Comparable" interface.
- Your string class is implementing Comparable interface and it has overridden the compareTo() method inside the String class.
- compareToIgnoreCase() method is used to compare two string without considering the case of the letters.

Example

```

package com.jtc10;
public class Jtc131{
public static void main(String arg[]){
String s1="abc";
String s2="aBc";
System.out.println(s1.compareTo("abc")); //0

```



```

System.out.println(s2.compareTo("abc")); //-32
System.out.println(s1.compareTo(s2)); //32
System.out.println(s1.compareToIgnoreCase(s2)); //0
}
}

```

split () method

- This method is used to split()String on the basis of some specified criteria but we are splitting the string by using the split() method which unnecessary create the many different object. The split() method when you are specifying the criteria on which basis you split the string so that specific criteria will not be included into resultant. While splitting the string by using the split() method, it spilt on the basis of that specified one but after spitting it. It will store inside a string Array which unnecessary going to create the n number of object.
- When it is creating unnecessary String objects, this may create heap memory related problems.
- So it is always recommended not to use the split() method.

Example

```

String str="Hi I am in java training center for java classes";
String s2[]=str.split("java");
for(int i=0;i<s2.length;i++){
System.out.println(s2[i]);
}

```

→output:

```

Hi I am in
training center for
classes

```

Q2 : Write a program to split a string on the basis of wild characters

```

class Jtc{
public static void main(String arg[]){
String s1="Hi, help. How are you? Bye! Ok";
String s2[]=s1.split("[\\.,?!]");
for(int i=0;i<s2.length;i++){
System.out.println(s2[i]);
}}}

```

startsWith()

This method is available inside the String class. It is used to check whether that corresponding String literal is internally startsWith() by the specified or not. Internally startswith() method works on the ASCII value. If that specified string starts with that specific string literal then it return the true else false.

indexOf()

This method is used to point the index position of the specified character, if the specified character is not there in the corresponding string then it return the -1 and if found then it returns the index position.

indexOf(char, int)

When you are specifying (char ,int) then it will not perform the checking of that specific character till that index position after that index position it will try to search the specified position if it is matching then it return the index position else it returns -1. If some index position which you are providing is not available then it also it returns the -1.

regionMatches()

In the regionMatches there are mainly four parameters as:

Syntax:

regionMatches(int offset,String,int onset,int)

in regionMatches the first parameter represent the index position of through which reference you are calling that, second parameter string is to compare with the index position which you have specified in the first parameter, third parameter int represents the index position of String which you have specified in the regionMathces parameter. The next int parameter represent the number of character has to compared in both string i.e. onset or offset string from the specified position.

regionMatches(boolean,int,String,int,int)

When you not specifying any Boolean parameter then by default it is false which means that while matching the region specified it compares the Strings by considering there cases. Also if explicitly you have written the false as a method parameter then also it will consider the cases of the specified String. If the

parameter is true then ignore the cases of the String.

matches()

This method is used to match the pattern. While matching the pattern * represents the continuation of the character till the time next will not be encountered.

```
public class Jtc{
    public static void main(String[] arg){
        String s1= "aaaabbbcddeee";
        System.out.println(s1.matches("a*b*cd*e"));
        System.out.println(s1.matches("ab*cd*e"));
    }
}
```

public char charAt(int index);

Returns the character locating at specified index.

```
class StringDemo{
    public static void main(String[] args){
        String s="prakash";
        System.out.println(s.charAt(3));
        System.out.println(s.charAt(100));
        StringIndexOutOfBoundsException
    }
}
```

public String concat(String str);

Use to join two String

Example

```
class StringDemo{
    public static void main(String[] args) {
        String s="som";
        s=s.concat("prakash");
        System.out.println(s);//somprakash
    }
}
```

- The overloaded "+" and "+=" operators are meant for concatenation purpose only.

public boolean equalsIgnoreCase(String s);

For content comparison where case is not important.

Example

```
class StringDemo {
    public static void main(String[] args){
        String s="java";
        System.out.println(s.equals("JAVA")); //false
        System.out.println(s.equalsIgnoreCase("JAVA")); //true
    }
}
```

Note:

We can validate username by using `.equalsIgnoreCase()` method where case is not important and we can validate password by using `.equals()` method where case is important.

public String substring(int begin);

Return the substring from begin index to end of the string.

Example

```
class StringDemo{
    public static void main(String[] args){
        String s="javatrainingcenter";
        System.out.println(s.substring(4)); //trainingcenter
    }
}
```

public String substring(int begin, int end);

Returns the substring from begin index to end-1 index.

Example

```
class StringDemo{
    public static void main(String[] args){
        String s="somprakash";
        System.out.println(s.substring(3)); //prakash
        System.out.println(s.substring(3,7)); //prak
    } }
}
```

public int length();

Returns the number of characters present in the string.

Example

```
class StringDemo{
public static void main(String[] args)    {
String s="javatrainingcenter";
System.out.println(s.length());
//System.out.println(s.length);
}}
```

Note

length is the variable applicable for arrays where as length() method is applicable for String object.

public String replace(char old, char new);

To replace every old character with a new character.

Example

```
class StringDemo{
public static void main(String[] args){
String s="ababab";
System.out.println(s.replace('a','b'));//bbbbbb
}}
```

public String toLowerCase();

Converts the all characters of the string to lowercase.

Example

```
class StringDemo{
public static void main(String[] args){
String s="SOM";
System.out.println(s.toLowerCase());//som
}}
```

public String toUpperCase();

Converts the all characters of the string to uppercase.

Example

```
class StringDemo{
```

```

    public static void main(String[] args) {
        String s="som";
        System.out.println(s.toUpperCase());//SOM }
    }

```

public String trim();

We can use this method to remove blank spaces present at beginning and end of the string but not blank spaces present at middle of the String.

Example

```

class StringDemo{
    public static void main(String[] args) {
        String s=" som prakash ";
        System.out.println(s.trim());//som prakash
    }
}

```

public int indexOf(char ch);

It returns index of 1st occurrence of the specified character if the specified character is not available then return -1.

Example

```

class StringDemo {
    public static void main(String[] args){
        String s="somprakash";
        System.out.println(s.indexOf('m'));
        System.out.println(s.indexOf('z'));
    }
}

```

public int lastIndexOf(Char ch);

It returns index of last occurrence of the specified character if the specified character is not available then return -1.

Example

```

class StringDemo{
    public static void main(String[] args){
        String s="somprakash";
        System.out.println(s.lastIndexOf('a'));
        System.out.println(s.indexOf('z'));
    }
}

```

```
}}
```

Note

- Because during runtime operation if there is a change in content then a new object will be created only on the heap but not in SCP.
- If there is no change in content no new object will be created then the same object will be reused.
- This rule is same whether object is present on the Heap or on SCP

14.1.3 java.lang.StringBuffer class

- StringBuffer is a class which is available inside java.lang package. It is a final class which cannot be extended to the subclasses. StringBuffer objects are mutable in nature, i.e., if you will perform any concatenation or other operation it will modify in the source. Almost all the member of stringBuffer is synchronized so it affects concurrency of the application. (Uses of the StringBuffer member affect the concurrency of the application). StringBuffer objects are eligible for serialization. StringBuffer does not use any special memory location like String. Constant pool of String for the reusability of objects. String buffer use capacity concept in order to store the String (capacity=16). StringBuffer object cannot be created without “new” operator.
- If the content change frequently then never recommended to go for String object because for every change will create a new object internally.
- To handle this type of requirement we should go for StringBuffer concept.
- The main advantage of StringBuffer over String is, all required changes will be performed in the existing object only instead of creating new object. (won't create new object)
- StringBuffer does not use any SCP or any special memory location for the reusability of string literals.
- Creates an empty StringBuffer object with default initial capacity “16”. Once StringBuffer object reaches its maximum capacity a new StringBuffer object will be created.

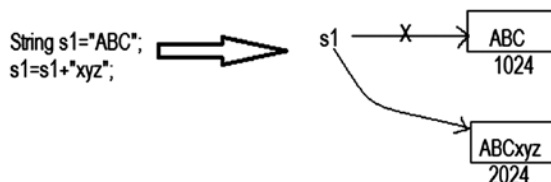
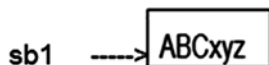


fig.: Showing immutability of String objects

- StringBuffer uses the “**capacity concept**” to store the string literals.

```
StringBuffer sb1="ABC";
sb1.append("xyz");
```



```
public class Jtc132{
public static void main(String arg[]){
StringBuffer sb1=new StringBuffer();
System.out.println(sb1.capacity()); //16
System.out.println(sb1.length()); //0
sb1.append("jtcindia");
System.out.println(sb1.capacity()); //16
System.out.println(sb1.length()); //8
sb1.append("jtcindia1");
System.out.println(sb1.capacity()); //34
System.out.println(sb1.length()); //17
}
}
```

Formula:

$$(initial\ capacity + 1) * 2$$

Important method of StringBuffer

- `public int length();`
Return the number of characters present in the StringBuffer.
- `public int capacity();`
Returns the total number of characters StringBuffer can accommodate(hold).
- `public char charAt(int index);`
It returns the character located at specified index.
- `public void setCharAt(int index, char ch);`
To replace the character locating at specified index with the provided character.
- `public StringBuffer append(String s);`
- `public StringBuffer insert(int index,String s);`
- `public StringBuffer delete(int begin,int end);`

To delete characters from begin index to end n-1 index.

- `public StringBuffer deleteCharAt(int index);`
To delete the character locating at specified index.
- `public void setLength(int length);`
Consider only specified number of characters and remove all the remaining characters.
- `public void trimToSize();`
To deallocate the extra allocated free memory such that capacity and size are equal
- `public void ensureCapacity(int initialcapacity);`
To increase the capacity dynamically(fly) based on our requirement

Note

Every method present in `StringBuffer` is synchronized hence at a time only one thread is allowed to operate on `StringBuffer` object, it increases waiting time of the threads and creates performance problems, to overcome this problem we should go for `StringBuilder`.

Program 14.6

```
package com.p33;
public class Jtc133{
    public static void main(String arg[]){
        /* char x='X';
        int i=0;
        System.out.println(true?x:0);
        System.out.println(false?i:x);
        */
        System.out.println("JTC"+"J");
        System.out.println("S"+"J");
    }
}

package com.jtc12;
public class Jtc134{
    public static void main(String arg[]){
        StringBuffer sb1=new StringBuffer("jtc");
        System.out.println(sb1.capacity());
        System.out.println(sb1.length());
        sb1.append("jtcindiajtcindia");
```

```
System.out.println(sb1.capacity());
System.out.println(sb1.length());
sb1.append("jtc");
System.out.println(sb1.capacity());
System.out.println(sb1.length());
sb1.append("jtcindiajtcindia123");
System.out.println(sb1.capacity());
System.out.println(sb1.length());
String s1="jtc";
String s2="jtc";
StringBuffer sb11=new StringBuffer("jtc");
StringBuffer sb22=new StringBuffer("jtc");
System.out.println(s1.equals(s2));
System.out.println(s1.hashCode()==s2.hashCode());
System.out.println(sb11.hashCode());
System.out.println(s1.hashCode());
System.out.println(s1.hashCode()==sb11.hashCode());
System.out.println(sb11.equals(sb22));
System.out.println(sb11.equals(s2));
}
}
```

/* This example proves that in StringBuffer class the method hashCode() and equals() is not overridden on the basis of ASCII value and content representation.

StringBuffer class contains all methods as synchronized. It means only one thread will process at a time. StringBuilder does not uses synchronized methods.*/

String	StringBuffer
1. String Object are immutable in nature	1. String Buffer objects are mutable in nature.
2. String Objects can be created with and without new operator.	2. Its object can only be created with new operator.
3. String uses the special memory location, String Content Pool for the reusability of String literal.	3. It's uses only one memory location.
4. In case of String you cannot delete any specific position character.	4. In String Buffer by using the built-in method you can delete any specific position character.

5. In case of String if you want to do a reverse then first you need to split the string and after splitting we need to store it inside String Array which is unnecessarily creating n number of object. Which is not recommendable.	5. In case of String Buffer by using the built in method we can reverse the string easily.
--	--

StringBuilder

- Every method present in StringBuffer is declared as synchronized hence at a time only one thread is allowed to operate on the StringBuffer object due to this, waiting time of the threads will be increased and effects performance of the system.
- To overcome this problem sun people introduced StringBuilder concept in 1.5v

StringBuffer Vs StringBuilder

StringBuffer	StringBuilder
Every method present in StringBuffer is synchronized	No method present in StringBuilder is synchronized.
At a time only one thread is allow to operate on the StringBuffer object hence StringBuffer object is Thread safe.	At a time Multiple Threads are allowed to operate simultaneously on the StringBuilder object hence StringBuilder is not Thread safe.
It increases waiting time of the Thread and hence relatively performance is low	Threads are not required to wait and hence relatively performance is high.
Introduced in 1.0 version.	Introduced in 1.5 versions.

String Vs StringBuffer Vs StringBuilder

- If the content is fixed and won't change frequently then we should go for String.
- If the content will change frequently but Thread safety is required then we should go for StringBuffer.
- If the content will change frequently and Thread safety is not required then we should go for StringBuilder.

Method Chaining

For most of the methods in String, StringBuffer and StringBuilder the return type is same type only. Hence after applying method on the result we can call another method which forms method chaining.

Example

```
sb.m1().m2().m3().....
```

In method chaining all methods will be evaluated from left to right.

Example

```
class StringBufferDemo{
public static void main(String[] args){
sb.append("som").insert(5,"prakash").delete(11,13)reverse()
.append("java").insert(18,"abcdf").reverse();
System.out.println(sb);
}}
```

Singleton Design Pattern

- Usually, you cannot create n-number of objects for a class, i.e., there is such types of restrictions on creating the objects, but in some cases it may give heap memory related problems.
- In order to avoid number of objects creation, we can implement the singleton design pattern.
- When we are writing the singleton design pattern then for that class only one object can be created.

Steps to implement singleton design pattern

- Declare the default constructor of a class as private, so that object of this class cannot be created from outside the class.
- Declare one static reference variable.
- Declare one static block and create the object of the class within that block itself.
- Write the static getter() method with the return type of the class.

Example

```
package com.jtc13;
class Hello{
```

//1. Declare default constructor of the class as private

```
private Hello(){
}
```

//2. static reference variable

```
static Hello h1=null;
```

//3. static block

```
static{
h1=new Hello();
}
```

//4. static getter method

```
static Hello getHello(){
return h1;
}

void m1(){
System.out.println("m1 in Hello");
}
}
```

```
public class Jtc134{
public static void main(String arg[]){
//Hello h1=new Hello();
Hello h1=Hello.getHello();
h1.m1();
Hello h2=Hello.getHello();
h2.m1();
System.out.println(h1);
System.out.println(h2);
System.out.println(h1.hashCode());
System.out.println(h2.hashCode());
}
}
```

/* The object h1 of Hello class will be created when the class will be loaded. The reference variable will be processed first then the static block.*/

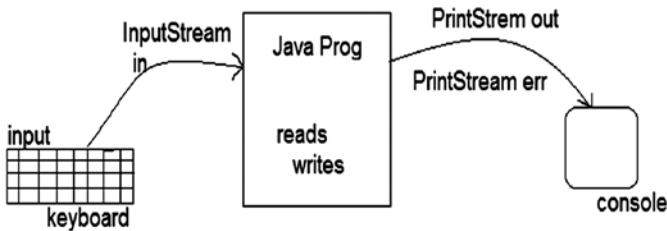
14.1.4 java.lang.System Class

System is a final class which is available in java.lang package. Default constructor of System class has been declared as private and also you will not get a requirement

to create System class object because almost all the members of System class is static. So, it can be accessed directly by the name of the class.

Variables

1. `public static final java.io.InputStream in;`
2. `public static final java.io.PrintStream out;`
3. `public static final java.io.PrintStream err;`



Methods

1. `public static void setIn(java.io.InputStream)`
2. `public static void setOut(java.io.PrintStream)`
3. `public static void setErr(java.io.PrintStream)`

System.in default implementation is of input device i.e. keyboard so the input which is being supplied to your java program will be given by the “in”.

In is the reference variable of “input stream” instead of taking input from keyboard if you want to take input from some other input device then in that case you need to change the implementation of in. Since in is the static final variable so it will not be able to change the value like this but at machine level we can change the value.

PrintStream out:

out is also a static final variable whose implementation cannot be changed. By default implementation of out is your output device i.e. console. If you wanted to change this then also you can specify.

java.io.PrintStream err

err is also the reference of PrintStream.err. Default implementation is the console as same like the out.

- **`public static native long currentTimeMillis()`**

This is static method which is available inside System class. currentTimeMillis is

used to return the time differences from 1st Jan midnight 1970 to the current time.

- **public static void setIn(java.io.InputStream)**
 - i. public static void setIn(java.io.InputStream)
 - ii. public static void setOut(java.io.PrintStream)
 - iii. public static void setErr(java.io.PrintStream)
 - o As in, out, err are the static final reference variables.
 - o in is responsible to take input from your keyboard. By default 'in' is pointing to the keyboard.
 - o out& err are the references of PrintStream. Both of them is pointing to the console itself.
 - o System.out.println() is used to display the general message on your console.
 - o System.err.println is used to display the error message on your console.
 - o By using setIn(), setOut(), setErr() methods we can change the implementation of in, out and err respectively.
- **public static native void arraycopy(java.lang.Object, int, java.lang.Object, int,int):**

arraycopy is used to copy the one array element to the another array. The first parameter java.lang.object is the source from which array has to copied. First int parameter specify the index position of the source array from where it has to be copied, next java.lang.Object parameter is a destination array in which array has to be copied. Next int parameter specifies that in the destination array from which index you need to copy the array, last int parameter specifies that how many number of parameter not to be copied.

Program 14.7

```
import java.io.FileOutputStream;
import java.io.PrintStream;
public class Jtc3 {
    public static void main(String[] args) throws Exception{
        System.out.println("---array copy---");
        int a[]={11,22,33,44,55,66,77};
        int b[]=new int [15];
        for (int i=0;i<a.length;i++){
            System.out.println("A array"+a[i]+"\\t"+i);
```

```

}
for (int i=0;i<b.length;i++){
    System.out.println("B Array"+b[i]+"\\t"+i);
}
System.arraycopy(a, 2, b, 1, 3);
System.out.println("after copy");
for(int i=0;i<a.length;i++){
    System.out.println(a[i]);
}
System.out.println("system class");
System.out.println("for error msg");
System.out.println("system.in");
System.out.println("system.out");
System.out.println("system.err");
System.out.println("system.out==system.err");
System.setOut(new PrintStream(new FileOutputStream("ABC.txt")));
System.out.println("i am going to file");
System.out.println("i am err msg");
}
}

```

System.exit()

System.exit() is used to terminate the JVM normally or abnormally. The normal and abnormal termination of JVM depends upon exit() method parameter. If exit() parameter is '0'(zero) then it terminates the JVM normally and other then '0' in the parameter of exit() then it terminates JVM abnormally.

The normal and abnormal termination means if the JVM is being terminated abnormally then also the system resources are not being closed. If it is the normal termination then all the system resources are being closed properly.

System.gc()

gc() method is a static method which is available inside System class. If it is a static method then it can be called directly by the name of class. gc() is used to invoke the garbage collector explicitly i.e. when you are trying to call garbage collector explicitly then in that case you need to write the System.gc().

You can't call gc() explicitly or when you are calling gc() explicitly then there is no guarantee that gc() will call or not.

System.runFinalization()

runFinalization is the static method which can be call directly by the name of class. runFinalization() will be invoke just before the gc() which will internally invoke finalize() in order to perform resource cleanup inside your class.

- **public static void setProperties(Properties)**

This method is used to set system properties like path, classpath, OS name etc.

- **public static String getProperty(String)**

This method is used to get the specified system property.

- **public static void load(String)**

This method is used to load the other libraries which has been implemented in some other programming languages; those methods and functions are called as native.

- **public static void loadLibrary(String)**

Example

Hai.c after compilation becomes Hai.exe

Hello.c after compilation becomes Hello.exe

```
classJtc{
static{
System.load("Hai.exe");
System.loadLibrary("Abc.lib");
}
public static void main(String arg[]){
//...
}
}
```

Runtime Class

- Runtime class is a class which is available inside java.lang package. You cannot create the Runtime class object because Runtime class default constructor has been declare as private. So you cannot create any object outside the class. Runtime class is mainly implemented on the basis of singleton design pattern. So you will not able to create the Runtime class object more than one.

- `Runtime rt=new Runtime(); //NOT OK`
- Runtime class object can be created in the following ways:
`Runtime rt=Runtime.getRuntime(); //OK`

- Runtime Object creation process:

```
class Runtime{
    private Runtime(){
    }

private static Runtime rt=null;
static{
    rt=new Runtime();
}

public static Runtime getRuntime(){
    return rt;
}
}

Runtime rt=Runtime.getRuntime(); //OK
Runtime rt=new Runtime();      //NOT OK
```

METHODS

1. **`public native int availableProcessors();`**
 2. **`public native long freeMemory();`**
 3. **`public native long totalMemory();`**
 4. **`public native long maxMemory();`**
- **`availableProcessors()`**: returns the number of processors available in your machine.
 - **`freeMemory()`**: returns the amount of unused heap memory.
 - **`totalMemory()`**: returns the total amount of memory which is allocated for heap.
 - **`maxMemory()`**: return the maximum amount of memory which can be used for heap.
 - **`exec()`**: It is used to execute the process specified at runtime.

`public java.lang.Process exec(String);`

Program 14.8

```
import java.io.IOException;
public class Jtc {
    public static void main(String[] args) throws IOException{
```

```
//Runtime rt=new Runtime();
Runtime rt=null;
Runtime.getRuntime();
Runtime rt1=Runtime.getRuntime();
System.out.println(rt);
System.out.println(rt1);
System.out.println(rt.hashCode()==rt1.hashCode());
System.out.println(rt.availableProcessors());
System.out.println(rt.totalMemory());
System.out.println(rt.freeMemory());
System.out.println(rt.maxMemory());
System.out.println(rt1.totalMemory()-rt1.freeMemory());
Process p1=rt.exec("notepad");
System.out.println(p1);
rt.exec("mspaint");
rt.exec("calc");
    }
}
```

Chapter 15

Wrapper Class

15.1 What are Wrapper class

The main objectives of wrapper classes are:

- To wrap primitives into object form so that we can handle primitives also just like objects.
- To define several utility functions which are required for the primitives.
- All most all wrapper classes define the following two constructors one can take corresponding primitive as argument and the other can take String as argument.

Example

```
Integer i=new Integer(10);
Integer i=new Integer("10");
```

If the String is not properly formatted i.e., if it is not representing number then we will get runtime exception saying "NumberFormatException"

Example

```
class WrapperClassDemo{
public static void main(String[] args)throws Exception{
Integer i=new Integer("ten");
System.out.println(i);//NumberFormatException
}}
```

Float class defines 3 constructors with float, String and double arguments.

```
Float f=new Float (10.5f);
Float f=new Float ("10.5f");
Float f=new Float(10.5);
Float f=new Float ("10.5");
```

Character class defines only one constructor which can take char primitive as argument there is no String argument constructor.

```
Character ch=new Character('a');//valid
Character ch=new Character("a");//invalid
```

- Boolean class defines two constructors with boolean primitive and String arguments.

- If we want to pass boolean primitive the only allowed values are true, false where case should be lower case.

Example

```
Boolean b=new Boolean(true);
Boolean b=new Boolean(false);
Boolean b1=new Boolean(True);//C.E
Boolean b=new Boolean(False);//C.E
Boolean b=new Boolean(TRUE);//C.E
```

If we are passing String argument then case and content is not important. If the content is case insensitive String of true then it is treated as true in all other cases it is treated as false.

Example

```
class WrapperClassDemo {
public static void main(String[] args)throws Exception{
Boolean b1=new Boolean("true");
Boolean b2=new Boolean("True");
Boolean b3=new Boolean("false");
Boolean b4=new Boolean("False");
Boolean b5=new Boolean("som");
Boolean b6=new Boolean("TRUE");
System.out.println(b1);//true
System.out.println(b2);//true
System.out.println(b3);//false
System.out.println(b4);//false
System.out.println(b5);//false
System.out.println(b6);//true
}}
```

Example (for exam purpose)

```
class WrapperClassDemo{
public static void main(String[] args)throws Exception{
Boolean b1=new Boolean("yes");
Boolean b2=new Boolean("no");
System.out.println(b1);//false
System.out.println(b2);//false
System.out.println(b1.equals(b2));//true
```

```
System.out.println(b1==b2);//false
}}
```

Note

In all wrapper classes `toString()` method is overridden to return its content.

In all wrapper classes `.equals()` method is overridden for content comparison.

Example

```
Integer i1 = new Integer(10) ;
Integer i2 = new Integer(10);
System.out.println(i1); //10
System.out.println(i1.equals(i2)); //true
```

15.2 Utility Methods

- `valueOf()` method.
- `XXXValue()` method.
- `parseXxx()` method.
- `toString()` method.

15.2.1 `valueOf()` method

- We can use `valueOf()` method to create wrapper object for the given primitive or String.
- This method is alternative to constructor.

Form 1

Every wrapper class except Character class contains a static `valueOf()` method to create wrapper object for the given String.

public static wrapper valueOf(String s);

Example

```
class WrapperClassDemo {
public static void main(String[] args)throws Exception {
Integer i=Integer.valueOf("10");
Double d=Double.valueOf("10.5");
Boolean b=Boolean.valueOf("som");
System.out.println(i);//10
```

```
System.out.println(d);//10.5
System.out.println(b);//false
}}
```

Form 2

Every integral type wrapper class (Byte, Short, Integer, and Long) contains the following `valueOf()` method to convert specified radix string to wrapper object.

Example

```
class WrapperClassDemo{
public static void main(String[] args){
Integer i=Integer.valueOf("100",2);
System.out.println(i);//4
}}
```

Form 3

Every wrapper class including Character class defines `valueOf()` method to convert primitive to wrapper object.

public static wrapper valueOf(primitive p);

Example

```
class WrapperClassDemo {
public static void main(String[] args)throws Exception{
Integer i=Integer.valueOf(10);
Double d=Double.valueOf(10.5);
Boolean b=Boolean.valueOf(true);
Character ch=Character.valueOf('a');
System.out.println(ch); //a
System.out.println(i);//10
System.out.println(d);//10.5
System.out.println(b);//true
}}
```

15.2.2 xxxValue() method

- We can use `xxxValue()` methods to convert wrapper object to primitive.
- Every number type wrapper class (Byte, Short, Integer, Long, Float, Double) contains the following six `xxxValue()` methods to convert wrapper object to

primitives.

1. public byte byteValue()
2. public short shortValue()
3. public int intValue()
4. public long longValue()
5. public float floatValue()
6. public double doubleValue();

Example

```
class WrapperClassDemo{
public static void main(String[] args)throws Exception{
Integer i=new Integer(130);
System.out.println(i.byteValue());//-126
System.out.println(i.shortValue());//130
System.out.println(i.intValue());//130
System.out.println(i.longValue());//130
System.out.println(i.floatValue());//130.0
System.out.println(i.doubleValue());//130.0
}}
```

charValue() method

Character class contains charValue() method to convert Character object to char primitive.

Example

```
class WrapperClassDemo{
public static void main(String[] args{
Character ch=new Character('a');
char c=ch.charValue();
System.out.println(c);//a
}}
```

booleanValue() method

Boolean class contains booleanValue() method to convert Boolean object to boolean primitive.

public boolean booleanValue();

Example

```
class WrapperClassDemo{
public static void main(String[] args){
Boolean b=new Boolean("som");
boolean b1=b.booleanValue();
System.out.println(b1);//false
}}
```

In total there are 38(= 6*6+1+1) xxxValue() methods are possible.

15.2.3 parseXxx() method

We can use this method to convert String to corresponding primitive.

Form1

- Every wrapper class except Character class contains a static parseXxx() method to convert String to corresponding primitive.

Example

```
class WrapperClassDemo{
public static void main(String[] args){
int i=Integer.parseInt("10");
boolean b=Boolean.parseBoolean("som");
double d=Double.parseDouble("10.5");
System.out.println(i);//10
System.out.println(b);//false
System.out.println(d);//10.5
}}
```

Form 2

Integral type wrapper classes(Byte, Short, Integer, Long) contains the following parseXxx() method to convert specified radix String form to corresponding primitive.

public static primitive parseXxx(String s,int radix);

The allowed range of radix is : 2 to 36

Example

```
class WrapperClassDemo{
```

```
public static void main(String[] args){
    int i=Integer.parseInt("100",2);
    System.out.println(i);//4
}
```

15.2.4 toString() method

We can use toString() method to convert wrapper object (or) primitive to String.

Form 1

public String toString();

- Every wrapper class (including Character class) contains the above toString() method to convert wrapper object to String.
- It is the overriding version of Object class toString() method.
- Whenever we are trying to print wrapper object reference internally this toString() method only executed.

Example

```
class WrapperClassDemo {
    public static void main(String[] args) {
        Integer i=Integer.valueOf("10");
        System.out.println(i);//10
        System.out.println(i.toString());//10
    }
}
```

Form 2

Every wrapper class contains a static toString() method to convert primitive to String.

public static String toString(primitive p);

Example

```
class WrapperClassDemo{
    public static void main(String[] args){
        String s1=Integer.toString(10);
        String s2=Boolean.toString(true);
        String s3=Character.toString('a');
        System.out.println(s1);           //10
    }
}
```

```
System.out.println(s2);          //true
System.out.println(s3);          //a
}}
```

Form 3

Integer and Long classes contains the following static toString() method to convert the primitive to specified radix String form.

public static String toString(primitive p, int radix);

Example

```
class WrapperClassDemo{
public static void main(String[] args){
String s1=Integer.toString(7,2);
String s2=Integer.toString(17,2);
System.out.println(s1);//111
System.out.println(s2);//10001
}}
```

Form 4

Integer and Long classes contains the following toXxxString() methods.

public static String toBinaryString(primitive p);

public static String toOctalString(primitive p);

public static String toHexString(primitive p);

Example

```
class WrapperClassDemo {
public static void main(String[] args) {
String s1=Integer.toBinaryString(7);
String s2=Integer.toOctalString(10);
String s3=Integer.toHexString(20);
String s4=Integer.toHexString(10);
System.out.println(s1);//111
System.out.println(s2);//12
System.out.println(s3);//14
System.out.println(s4);//a
}}
```

AutoBoxing

Example

```
public class Jtc7 {

    public static void main(String[] args) {
        Integer in = 123; // AutoBoxing
        in++;
        System.out.println(in + 10);
        int ab = in; // AutoUnBoxing
        // int ab=in.intValue();
        long val = 123;
        // Long ref=new Integer(123);
        // Long obj=1234; // Widening and Auto Boxing
        Long obj2 = 1234L; // Auto Boxing
        Object obj4 = 1234; // Auto Boxing and widening
        Number num = 1234; // Auto Boxing and widening
        System.out.println("-----");
        Integer in1 = 123;
        Integer in2 = in1;
        System.out.println(in1 + "\t" + in2);
        System.out.println(in1 == in2);
        in1++;
        // int t=in1.intValue();
        // t++;
        // in1=new Integer(t);
        System.out.println(in1 + "\t" + in2);
        System.out.println(in1 == in2);
    }
}
```

Example

```
public class Jtc8 {
```

```
public static void main(String[] args) {
    Integer in1 = new Integer(123);
    Integer in2 = new Integer(123);
    System.out.println(in1 == in2);
    Boolean b1 = new Boolean(true);
    Boolean b2 = new Boolean(true);
    System.out.println(b1 == b2);
    System.out.println("-----");
    Boolean b3 = false;
    Boolean b4 = false;
    Boolean b5 = false;
    System.out.println(b3 == b4);
    System.out.println(b4 == b5);
    System.out.println("----BYTE-----");
    Byte by1 = 127;
    Byte by2 = 127;
    System.out.println(by1 == by2);
    System.out.println("-- CHARACTER
--"); Character ch1 = 'A';
    Character ch2 = 'A';
    System.out.println(ch1 == ch2);
    Character ch3 = 128;
    Character ch4 = 128;
    System.out.println(ch3 == ch4);
    System.out.println("-- INTEGER -");
    Integer in3 = 127;
    Integer in4 = 127;
    System.out.println(in3 == in4);
    Integer in5 = 128;
    Integer in6 = 128;
    System.out.println(in5 == in6);
    System.out.println("-- LONG -");
```

```

Long ref1 = 127L;
Long ref2 = 127L;
System.out.println(ref1 == ref2);
Long ref3 = 128L;
Long ref4 = 128L;
System.out.println(ref3 == ref4);
System.out.println("--- float ----");
Float f1 = 12.0F;
Float f2 = 12.0F;
System.out.println(f1 == f2);
System.out.println("=====");
Integer in11 = 123;
Integer in12 = 122;
Integer in13 = 122;
in12++;
in13 = in13 + 1;
System.out.println(in11 + "\t" + in12 + "\t" + in13);
System.out.println(in11 == in12);
System.out.println(in11 == in13);
}
}

```

Example

```

public class Jtc9 {

    public static void main(String[] args) {
        BoxingService.m1(123);
        BoxingService.m2(123);
        BoxingService.m2(123);
    }
}

class BoxingService {
    static void m1(long val) {

```

```

System.out.println("-- m1(long)---");
}
static void m2(int... is) {
System.out.println("-- m2(int...)---");
}
static void m2(Integer in) {
System.out.println("-- m2(Integer)---");
}
}

```

Example

```

public class Jtc10 {

    public static void main(String[] args)
    { BoxingLoading.m1(123);
      BoxingLoading.m2(123);
    }
}

class BoxingLoading {
    static void m1(long val)
    { System.out.println("-- m1(long)---");
    }
    static void m1(Integer in)
    { System.out.println("--
      m1(Integer)---"); }
    static void m2(int... is)
    { System.out.println("-- m2(int...)---");
    }
    static void m2(Integer val)
    { System.out.println("--
      m2(Integer)---"); }
}

```

Chapter 16

Exception Handling

16.1 Introduction

When you are writing any Java program, you may get the following types of problems:

1. **Error**

2. **Exception**

1. **ERROR:** There are mainly the following types of errors:

- a. **Compile-time error**

- In the case of compile time error, if any syntactical mistakes are there, then it is called as compile-time error which will be identified at the time of compilation.
- Compile-time errors show that the compiler is inefficient to compile that specific program without correcting that.

- b. **Runtime Error**

- The error which is being identified at runtime with the help of JVM is called runtime error.
- Runtime errors show the inefficiency of the JVM.

For eg.

- o Main method is not available (compiler can compile without the main method).
- o If any stack memory related problem is identified, that will be represented by `java.lang.StackOverflowError` (memory required by the JVM to process).

16.2 Exception

- Some unexpected operation which occurs in a program is called as exception.
- Exception is of two types:
 1. Compile-time exception / Checked exception
 2. Runtime exception / Unchecked exception

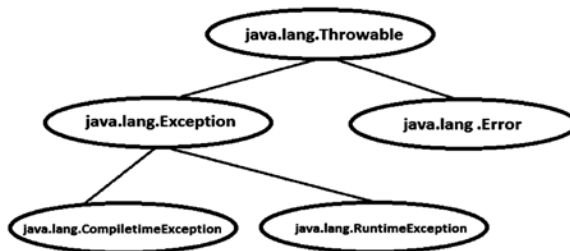
Example

```
class Jtc103{
    public static void main(String arg[]){
        System.out.println("in Jtc103");
    }
}
```

/* errors:

1. java Jtc103 (enter)
In Jtc103
Exception in thread "main"
Java.lang.ArrayIndexOutOfBoundsException : 0
2. java Jtc103 abc (enter)
In Jtc103
Exception in thread "main"
java.lang.NumberFormatException : for input string "abc"
3. java Jtc103 0 (enter)
in Jtc103
Exception in thread "main"
java.lang.ArithmeticException : 1 by zero */

- For all types of exceptions and errors, "java.lang.Throwable" is the super type.
- For all types of exceptions "java.lang.Exception" is the super class.
- For all the types of Runtime Exceptions, "java.lang.RuntimeException" is the super class.



Example

```
class Jtc104{
    public static void main(String arg[]){
        System.out.println("in Jtc104");
    }
}
```

```

try{
int i=Integer.parseInt(arg[0]);
int i1=10/i;
System.out.println(i);
}
catch(Exception e){
System.out.println("in catch...");
if(e instanceof ArrayIndexOutOfBoundsException){
System.out.println("plz provide the value.." +e);
}
else if(e instanceof NumberFormatException){
System.out.println("plz provide numeric value.." +e);
}}}}

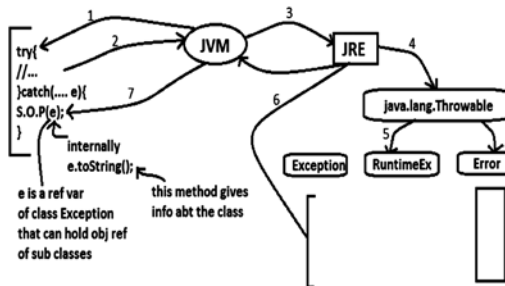
```

Output:

1. java Jtc104 (enter)
 in Jtc104
 in catch...
 plz provide the value...:ArrayIndexOutOfBoundsException:0

2. java Jtc104 2abc (enter)
 in Jtc104
 in catch...
 plz provide numeric value...:NumberFormatException:for input string "2abc"

Working



- Whenever any problem is identified, following things will be done internally by the JVM:
 - If any problem is identified, immediately control will be transferred to the JVM.

2. Now, JVM will identify the exact type of problem.
3. After identifying the exact type of problem, it will create the object of that and will throw it to the program.
4. In the program, it will try to search matching catch block.
5. If the matching catch block is found, then the corresponding catch block will be processed.

16.2.1 Handling exception by writing the multiple try and catch

1. Enclose that piece of code which you want to monitor, inside the try block
2. Write the catch block associated with that try with the proper exception type.
3. By using a catch you can handle only one exception at a time.
4. You can write multiple catch associated with a try to handle the multiple types of exceptions.

- **While writing multiple try and catch remember the following points**

1. There should not be any statement between try and catch.
2. There should not be any statement between catch and catch.
3. You can write try without any catch but you cannot write catch without any try.
4. When you are writing the catch blocks, the catch parameter should be sub to super type. It cannot have parameter from super to sub type.

Example

```
package com.jtc.p1;
public class Jtc105{
public static void main(String arg[]){
int i1=10;
try{
int i2=Integer.parseInt(arg[0]);
int ii=i1/i2;
}

/*catch(exception e){ ... }
Catch(Throwable e){ ... }
*/
Catch(NumberFormatException e){
System.out.println("plz provide number value..." +e);
}
```

```

catch(ArithmeticException e){
    System.out.println("plz provide non zero value..." +e);
}
catch(ArrayOutOfBoundsException e){
    System.out.println("plz provide some value..." +e);
}
catch(NullPointerException e){
    System.out.println("trying exception type..." +e);
}
catch(Exception e){
    System.out.println("boss of all exceptions..." +e);
}
}
}
}

```

Example

```

package com.jtc.p2;
public class Jtc106{
    public static void main(String arg[]){
        int i1=10;
        try{
            int i2=Integer.parseInt(arg[0]);
            int ii=i1/i2;
        }
        catch(NumberFormatException | ArithmeticException |
            ArrayIndexOutOfBoundsException | NullPointerException e)
        {
            System.out.println(e);
            /* exception not allowed sub to super type...error: catch statement cannot be
            related by subclassing */
        }
        catch(Exception e){
            System.out.println(e);
        }
    }
}

```

- From JDK1.7 you can handle the multiple exceptions by writing a single try and catch.

- In catch you can write multiple exceptions separated by a pipe operator (|).
- But make sure not to write any super type of exception in that.

16.3 Finally Block

- When any return statement is encountered, any other statements will not be processed after that. Immediately control will be transferred to the caller of the method.
- In some cases, some statements are very important to be processed whether return statement is processed or not.
- For eg: closing the resources. If you have used the resources and it's not closed properly then it may create some problem. In order to make sure that the statement must be processed in any case in that you need to use the finally block.
- When you are enclosing any statement inside the finally block then whether any return statement is encountered or not, any exception occurred or not, the finally block will always be processed.

Note

Finally block will not be processed only when `System.exit()` is used.

Example

```
package com.jtc.p3;
class Hello{
int m1(int a){
System.out.println("m1 in hello");
try{
System.out.println("in try of m1");
int x=10/a;
System.out.println("x= "+x);
//return 88;
}
catch(Exception e){
System.out.println("in catch of m1");
e.printStackTrace();
}finally{
System.out.println("in finally...very important code");
}
System.out.println("after catch block in m1");
}
```

```

return 99;
}
int m2(int a){
System.out.println("m2(int a) in Hello");
try{
System.out.println("in m2(int a) try block");
int x=10/a;
return 10;
}catch(ArithmeticException e){
System.out.println("in catch of m2");
e.printStackTrace();
}finally{
System.out.println("very imp code in finally of m2");
}
System.out.println("out of catch in m2");
return 20;
}
}

public class Jtc107{
public static void main(String arg[]){
Hello h1=new Hello();
h1.m1(Integer.parseInt(arg[0]));
h1.m2(Integer.parseInt(arg[1]));
}
}

```

Remember the following points while writing try and catch

- There should not be any statement between try-catch-finally.
- You can write try without catch.
- You should not write try-catch after the finally.
- You cannot write finally without try.
- finally will be the last processing inside try-catch.

NOTE

when `System.exit(0)` is encountered in that case, finally will not be processed.

Whenever any return is encountered, following things will be done

internally by the JVM

1. Whenever any return statement is encountered, JVM checks whether any finally block is available or not.
2. If finally block is available then before transferring the control to the caller of the method, it will try to process the finally block and immediately after processing the finally block, control will be transferred to the caller of the method.

Q1: Can we write return statement in try-catch-finally?

Ans : Yes, but it has no benefit.

(if you are writing return statement in finally, then no need to write return in try and catch)

Example

```
package com.jtc.p4;
class Hello{
    int m1(){
        System.out.println("m1 in Hello");
        try{
            int x=10/0;
            return 10;
        }
        catch(Exception e){
            e.printStackTrace();
            return 20;
        }
        finally{
            System.out.println("in finally");
            return 30;
        }
    }
}

public class Jtc108{
    public static void main(String arg[]){
        Hello h1=new Hello();
        int i=h1.m1();
        System.out.println(i);
    }
}
```

```

}
}

```

Example of finally block use

```

import com.sun.org.apache.regexp.internal.recompile;
class Window{
    public Window(){
        System.out.println("Open Window");
    }
    public void close(){
        System.out.println("close window");
    }
}
class Door{
    Door(){
        System.out.println("open door");
    }
    public void close(){
        System.out.println("close door");
    }
}
class UseRes{
    int m1(){
        Door dr=null;
        Window wn=null;
        try{
            dr=new Door();
            wn=new Window();
            System.out.println("perform your task");
        }
        catch(Exception e){
            e.printStackTrace();
        }
        finally{
            dr.close();
            wn.close();
        }
        return 20;
    }
}

```



```

class Jtc109{
public static void main(String arg()){
UseRes ur=new UseRes();
ur.m1();
}}

```

AUTOCLOSEABLE INTERFACE (from JDK1.7)

```

class Window implements AutoCloseable{
public Window(){
System.out.println("Open Window");
}
public void close(){
System.out.println("Close Window");
}
}

class Door implements AutoCloseable{
Door(){
System.out.println("Open Door");
}
public void close(){
System.out.println("Close Door");
}
}

public class Jtc110{
public static void main(String arg[]){
try(Door dr=new Door(); Window wn=new Window()){
System.out.println("Use Resource");
}
catch(Exception e){
e.printStackTrace();
}
}
}

```

More examples

```

import java.io.*;
public class Jtc111{
public static void main(String arg[]){

```

```

System.out.println("in main");
try(FileOutputStream fos=new FileOutputStream("abc.txt");
BufferedOutputStream bos=new BufferedOutputStream(fos);
DataOutputStream dos=new DataOutputStream(bos))
{
dos.writeUTF("HELLO JTC");
}
catch(Exception e){
e.printStackTrace();
}
}
}

```

```

import java.io.*;
public class Jtc112{
public static void main(String arg[]){
FileOutputStream fos=null;
BufferedOutputStream bos=null;
DataOutputStream dos=null;
try{
fos=new FileOutputStream("abc.txt");
bos=new BufferedOutputStream(fos);
dos=new DataOutputStream(bos);
dos.writeUTF("HELLO JTC STUDENTS");
}
catch(Exception e){
e.printStackTrace();
}
finally{
try{
fos.close();
bos.close();
dos.close();
}
catch(Exception e){
}}}}

```

- If you want to develop your custom resources then write your resource class by implementing the `AutoCloseable` interface.

- In AutoCloseable interface there is only one method, close().
- This close() method must be overridden inside your class according to the resource requirement.
- If you are using resource inside try with resource then your resource must implement AutoCloseable interface, otherwise it will give the compile-time error.
- If we are using try with resource and it is satisfying all the constraints then there is no use of writing any finally block, you can easily write it with that.

16.4 Types Of Exceptions

- There are mainly two types of exceptions:
 1. Checked exception / compile-time exception
 2. Unchecked exception / runtime exception

Checked Exception or Compile-time Exception

- The exception being caught by the compiler at the time of compilation or caught by the compiler is called compile time exception.
- Also, the exceptions which are the sub class of java.lang.Exception except java.lang.RuntimeException is called compile time exception.
- **Note:**
The compile-time exception can be of two types:
 - I. User-defined
 - II. Built-in type
- Compile time exception must be handled or reported either by writing try and catch or by propagating.
- For eg: java.lang.IOException, FileNotFoundException, SQLException, JSPException.

Unchecked Exception or Runtime Exception

- The exceptions that are the sub class of java.lang.RuntimeException are called as Unchecked or runtime exceptions.
- These exceptions will be called by the JVM.
- It need not to be reported by writing try and catch or by propagating. Even if you want, you can handle it.
- **Runtime exceptions are also of two types:**
User-defined: the exceptions that are being created by the developer according to the application requirements are called as user-defined exceptions.
 Eg: InsufficientFoundException, AccountNumberDoesNotExist.

Built-in: the exceptions given by the technology vendors is called as built-in exception. These also can be of checked and unchecked types. Eg: ServletException, JSPEException, EJBException.

Steps To Write User Defined Exceptions

1. Write your exception class by extending java.lang.Exception or java.lang.RuntimeException.
2. Declare some instance variable inside exception class.
3. Write default constructor inside exception class.
4. Write one parameterized constructor if required.
5. Override toString() method (it internally calls getMessage() method).
6. Override getMessage() method (it is available inside throwable class and gives exact reason of the exception).

e.printStackTrace() gives the following information

=> type of exception occurred

=> Reason of exception

=> location of exception)

- Throw sends the exception to the next level to handle; it depends on the type of exception.
 - o Checked exceptions must be handled by either
 - o Try n catch blocks, or
 - o Transfer to another level(method header propagation)
- Unchecked exceptions can be handled by the JVM

Example

```
class InvalidAccNoException extends Exception{
    int accno;
    public InvalidAccNoException(){
    }
    InvalidAccNoException(int accno){
    }
    public String toString(){
        return "..."+getMessage();
    }
    public String getMessage(){
        return "...";
    }
}
```

```

class ServerNotRespondingException extends RuntimeException{
}
public class Jtc113{
public static void main(String arg[]){
try{
int i=Integer.parseInt(arg[0]);
}
catch(Exception e){
System.out.println(e);
}
}
}
import java.io.IOException;
import java.sql.SQLException;
class Hello{
void m1(int a)throws ArithmeticException{
System.out.println("in m1");
throw new ArithmeticException();
}
void m2(boolean b)throws IOException,InstantiationException{
System.out.println("in m2");
if(!b){
throw new IOException();
}
else{
throw new InstantiationException();
}
}
void m3(){
System.out.println("in m3");
try{
throw new SQLException();
}
catch(SQLException e){
}
}
void m4() throws IOException{
System.out.println("m4 in Hello");
}
}

```

```

try{
m2(true);
}
catch(InstantiationException e){
}
}
void m5(){
System.out.println("m5 in Hello");
m3();
}
}
public class Jtc114{
public static void main(String arg[])throws IOException{
Hello h1=new Hello();
h1.m4();
h1.m5();
}
}

```

16.5 “throw” keyword

- Throw keyword is used to throw the exception from the body of the method.
- If you are throwing any unchecked exception then there is no hard and fast rule that you will have to report about that exception using try and catch or by propagating.
- But, it is always recommendable to handle throw from the body of the method using try and catch.

Syntax

```

void m1(){
ArithmeticException ae=new ArithmeticException();
throw ae;
}
OR
void m1(){
throw new ArithmeticException();
}

```

- If you are throwing any checked exception from the body of the method then

you must have to report about that exception by writing try and catch or by propagating at method header.

16.6 “throws” keyword

- throws is used to throw the exception from the header of the method.
- Whenever you are throwing any exception by writing throws keyword, it means you are directing caller of the method to report about the exception in its own way (i.e., either by using try and catch or by propagating).
- If you are throwing any checked exception then the caller of the method must have to report about the exception by writing try and catch or by propagating.
- If you are not doing that then it will give compilation error.
- If you are throwing any unchecked exception by writing throws keyword then you may or may not handle that.
- If you are not handling it, it will automatically be handled by the JVM.

(eg: void m1() throws IOException

```
{
    try{
        throw new IOException();
    }
    catch(Exception e){
        //...
    }
}
```

Using both ‘throws’ and ‘throw’ will not give any compile-time error but the caller of m1() will have to report IOException by either using try and catch or by propagating)

- **NOTE:** It always recommendable to handle the exception using try and catch.

Example

```
import java.io.FileNotFoundException;
import java.io.IOException;
import java.sql.SQLException;
class Hello{
    void m5()throws NumberFormatException{
        System.out.println("in m5");
        try{
            int i=10/0;
            System.out.println(i);
```

```

}
catch(ArithmeticException e){
System.out.println(e);
}
}
void m4()throws ArithmeticException,SQLException{
System.out.println("in m4");
try{
throw new IOException();
}
catch(IOException e){
System.out.println(e);
}
}
void m3()throws SQLException,FileNotFoundException{
System.out.println("m3 in Hello");
m5();
m4();
}
void m2()throws FileNorFoundException,IOException{
System.out.println("m2 in Hello");
try{
m3();
}
catch(SQLException e){
System.out.println(e);
}
}
void m1()throws ArithmeticException{
System.out.println("m1 in Hello");
}
}
public class Jtc115{
public static void main(String arg[]){
Hello h1=new Hello();
try{
h1.m1();
h1.m2();

```



```

}
catch(ArithmeticException | FileNotFoundException e){
System.out.println(e);
}
}
}
import java.io.Exception;
class CheckedException extends Exception{
}
class UncheckedException extends RuntimeException{
}
class Hello{
void m1(int a)throws UncheckedException,CheckedException,IOException{
System.out.println("m1 in Hello");
if(a==0){
throw new Uncheckedexception();
}
else{
throw new CheckedException();
}
}
void m2(){
System.out.println("m2 in Hello");
try{
m1(11);
}
catch(UncheckedException | CheckedException | IOException e){
e.printStackTrace();
}}
public class Jtc117{
public static void main(String arg[]){
new Hello().m2();
}
}
class InvalidUserIdException extends Exception{
String userId;
public InvalidUserIdException(){
}
}

```

```

InvalidUserIdException(String userId){
this.userId=userId;
}
public String toString(){
return this.getClass.getName+":getMessage();
}
public String getMessage(){
return "userId \t"+userId+" is invalid";
}
}
class ServerBusyException extends RuntimeException{
}
class ChatRoom{
void startChat(String userId){
System.out.println("connecting to the chatroom");
try{
if(userId.equals("som123")){
System.out.println("Welcome to chatroom...");
}
else{
throw new InvalidUserIdException e){
System.out.println(userId);
}
}
catch(InvalidUserIdException e){
System.out.println(e);
}}}
public class Jtc118{
public static void main(String arg[]){
ChatRoom cr=new ChatRoom();
try{
cr.startChat(arg[0]);
}
catch(Exception e){
if(e instanceof ArrayIndexOutOfBoundsException){
System.out.println("provide user id to chat");
}
else if(e instanceof NumberFormatException){

```

```
System.out.println("provide valid user id");
}}}
```

- When super class method is throwing method-level exception then subclass overridden method can do the following tasks :
 1. Subclass overridden method can ignore the exception which is being thrown from the super class method as it is.
 2. Subclass overridden method can report about the exception which is coming from the super class as it is.
 3. Subclass overridden method can report about the exception which is the subclass of the super class method-level exception.
 4. Subclass overridden method cannot report about the super class of the super class method-level exception
 5. Subclass overridden method cannot report about the new type of exception which is not coming from the super class method.

16.7 Methods

printStackTrace()

- it is available inside java.lang.Throwable
- it is used to give the following 3 informations about any exceptions:
- type of exception
 - o reason of exception
 - o line number of exception where it occurred

getMessage()

- it is available inside java.lang.Throwable
- this is used to give exact reason of the exception that why exactly it occurred.

Example

```
(1)
class Hello{
public static void main(String arg[]){
System.out.println(m1());
}
static boolean m1(){
try{
return true;
}
```

```
finally{
return false;
}
}
}
```

(2)

```
import java.io.IOException;
public class Hello{
public static void main(String arg[]){
try{
System.out.println("hell Jtc");
}
catch(IOException e){
System.out.println("i have never seen println fail");
}
}
}
```

(3)

```
public class XYZ{
public static void main(String arg[]){
try{
if you have nothing nice to say then say nothing
}
catch(Exception e){
System.out.println("this cannot happen");
}
}
}
```

(4)

```
interface Type1{
void f()throws CloneNotSupportedException{
}
interface Type2{
void f()throws InterruptedException{
}
interface Type3 extends Type1,Type2{
}
```

```

public class Jtc3 implements Type3{
    public void f(){
        System.out.println("Hello World");
    }
    public static void main(String arg[]){
        Type3 t3=new Jtc3();
        t3.f();
    }
}

```

(5)

```

public class Hello{
    public static final long GUEST_USER_ID=-1;
    private static final long USER_ID;
    static{
        try{
            USER_ID=getUserIdFromEnvironment();
        }
        catch(IdUnavailableException e){
            USER_ID=GUEST_USER_ID;
            System.out.println("logging in as guest");
        }
    }
    private static long getUserIdFromEnvironment()throws IdUnavailableException{
        throw new IdUnavailableException();    //simulate an error
    }
    public static void main(String arg[]){
        System.out.println("user Id:"+USER_ID);
    }
}

class IdUnavailableException extends Exception{
}

```

(6)

```

public class HelloGoodbye{
    public staic void main(String arg[]){
        try{
            System.out.println("hello world");
            System.exit(0);
        }
    }
}

```

```

}
finally{
System.out.println("goodbye world");
}
}
}

```

(7)

```

public class Reluctant{
public Reluctant inetrnalInstance=new Relactant();
public Reluctant() throws Exception{
throw new Exception("i am not coming out");
}
public static void main(String arg[]){
try{
Reluctant b=new Reluctant();
System.out.println("surprise!!");
}
catch(Exception ex){
System.out.println("i told u so");
}
}
}
}

```

(8)

```

import java.io.*;
public class Copy{
static void Copy(String src,String dest)throws Exception{
InputStream in=null;
OutputStream out=null;
try{
in=new FileInputStream(src);
out=new FileOutputStream(dest);
byte[] buf=new byte[1024];
int n;
while((n=in.read(buf))>0)
out.write(buf,0,n);
}
finally{

```

```

if(in!=null) in.close();
if(out!=null) out.close();
}
}
public static void main(String arg[])throws IOException{
if(args.length!=2)
System.out.println("usage:java copy <src><dest>");
else
copy(arg[0],arg[1]);
}
}

```

(9)

```

public class Loop{
public static void main(String arg[]){
int[][] tests={{1,2,3,4,5,6},{1,2},{1,2,3},{1,2,3,4,5},{5}};
int n=0;
try{
int i=0;
while(true){
if(thirdElementIsThree(tests[i++]))
n++;
}
}
catch(ArrayIndexOutOfBoundsException e){
//No more test to process
}
System.out.println(n);
}
public static boolean thirdElementIsThree(int[] a){
return a.length>=3&a[2]==3;
}
}

```

(10)

```

public class Thrower{
public static void main(String arg[]){
sneakyThrow(new Exception("this is a checked exception"));
}
}

```

/* provide a body for this method to make it throw the specified exception.

you must not use any deprecated methods

*/

```
public static void sneakyThrow(throwable t){
}
}
```

(11)

```
class Missing{
```

```
Missing(){}
}
```

```
public class Strange1{
```

```
public class void main(String arg[]){
```

```
try{
```

```
Missing m=new Missing();
```

```
}
```

```
catch(java.lang.NoClassDefFoundError ex){
```

```
}
```

```
}
```

```
}
```

```
public class Strange2{
```

```
public static void main(String arg[]){
```

```
Missing m;
```

```
try{
```

```
m=new Missing();
```

```
}
```

```
catch(java.lang.NoClassDefFoundError ex){
```

```
System.out.println("got it");
```

```
}
```

```
}
```

```
}
```

(12)

```
public class Workout{
```

```
public static void main(String arg[]){
```

```
Workout();
```

```
System.out.println("its a nap time");
```

```
}
```



```
private static void workHard(){
try{
workHard();
}
finally{
workHard();
}}}
```

16.8 Difference between Class Not Found Exception & No Class DefFound Error

For hard coded class names at Runtime in the corresponding .class files not available we will get NoClassDefFoundError , which is unchecked

```
Test t = new Test( );
```

In Runtime Test.class file is not available then we will get NoClassDefFoundError

For Dynamically provided class names at Runtime , If the corresponding class files is not available then we will get the RuntimeException saying ClassNotFoundException

```
Ex : Object o=Class.forName("Test").newInstance( );
```

At Runtime if Test.class file not available then we will get the ClassNotFoundException , which is checked exception

Example

```
public class Jtc5 {
public static void main(String[] args)
{ System.out.println("-- Main Started --");
System.out.println("No of Arg\t:" +
args.length); try {
int ab = Integer.parseInt(args[0]);
System.out.println("-- ab value is\t:" + ab);
```

```

int res = 123 / ab;
System.out.println("-- Result is\t:" + res);
} catch (ArithmeticException e) {
System.out.println("\n* Value should not be 0 *");// Task 1
} catch (NumberFormatException | ArrayIndexOutOfBoundsException ex) {
System.out.println("\n* Provide one int Value as CLA *");// Task 2
} catch (Exception e) {
System.out.println("\n* Other Exception Occured *");// Task 3
}
System.out.println("\nAfter Catch Block");
System.out.println("Main Completed");
}
}

```

Example - Try with resource file

```

import java.io.FileInputStream;

public class Jtc6 {
public static void main(String[] args) {
long res1 = ResourceUser.readFile("info.txt");
System.out.println("In main Result\t:" + res1);
System.out.println();
long res3 = ResourceUser.readFile("Empty.java");
System.out.println("In main Result\t:" + res3);
}
}

class ResourceUser {
static long readFile(String fileName) {
System.out.println("-- File Name in Method\t:" + fileName);
long length = 0;
try (JtcResource resource = new JtcResource());
FileInputStream fis = new FileInputStream(fileName);
/* JtcAnotherResource other=new JtcAnotherResource(); */) {

```

```

while (true) {
    int x = fis.read();
    if (x == -1)
        break;
    length++;
}
long res = 13 / length;
System.out.println("Result\t:" + res);
} catch (Exception e) { e.printStackTrace(); }
return length;
} }

class JtcResource implements AutoCloseable {
    public void close() throws Exception {
        System.out.println("** close method() of JtcResource **\t:" + this);
    } }

class JtcAnotherResource { }

```

Example - Rethrowing Exceptions with Improved Type Checking.

Before Java 7

```

public class ObjectCreator {
    public static Object getInstance(String className)
        throws Exception {
        Object ref = null;
        try {
            Class cl = Class.forName(className);
            ref = cl.newInstance();
        } catch (Exception e) {
            // Send mail to Mail ID
            throw e;
        }
        return ref;
    }
}

```

From Java 7

```

public class ObjectCreator {

```

```
public static Object getInstance(String className)
throws ClassNotFoundException, IllegalAccessException,
InstantiationException {
    Object ref = null;
    try {
        Class cl = Class.forName(className);
        ref = cl.newInstance();
    } catch (Exception e) {
        // Send mail to Mail ID
        throw e;
    }
    return ref; }}
```

Chapter 17

Multithreading

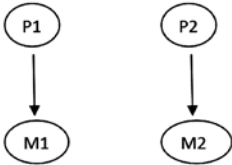
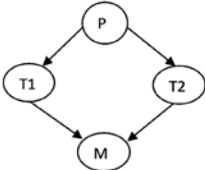
17.1 What is Multitasking?

Performance or executing more than one task at the same time is called as multitasking.

Note: As of now there is no machine or processors which can perform more than one task at the same time but the time duration required to complete the task is very less, which we will not be able to analyse whether the task being done at the same time or different time.

There are mainly two types of multitasking

- 1)Process based multitasking.
- 2)Thread based multitasking.

Process Based	Thread Based
1) Process is a program and if multiple process or being done or process at the same time then it utilize the separate memory location and separate resources also.	1) Thread is the unit of a program in a process multiple threads that might be running & which all leads to the competition of a process by sharing the same memory location as well as the recourse.
2) A process may or may not be implemented in a same program so all way utilizes different recourses which could be quiet expensive.	2) In case of thread execution, it is not expensive as like process.
3) 	3) 
4) In case of process based multitasking context switching is quiet difficult.	4) In case of thread based context switching is very easy.

- **Note**

While threads will share the resources which will be decided by CPU scheduling algorithm and is managed by thread life cycle.

Each thread in java will be managed by a special life cycle mechanism at the processing of all these will completely depend upon the scheduling algorithm.

17.2 Java thread

Sun has given following API to manage the thread:

1) **Java.lang.Thread**

2) **Java.lang.ThreadGroup**

3) **Java.lang.Runnable Interface**

Whenever JVM starts internally it will create one ThreadGroup whose name is 'main' under that ThreadGroup one more thread will be created with the name of 'main'.

Each thread in java will be assigned with specific task and after completion move to the dead states.

In java almost everything will be done on the basis of thread.

Example

```
public class Jtc181 {
    public static void main(String[] args) {
        Thread th1 = new Thread();
        System.out.println(th1);
        String name = th1.getName();
        System.out.println(name);
        int p = th1.getPriority();
        System.out.println(p);
        th1.setName("Jtc-th1");
        System.out.println(th1.getName());
        th1.setPriority(9);
        // th1.setPriority(11);
        // th1.setPriority(0);
        System.out.println(th1.getPriority());
        Thread th2 = new Thread();
        System.out.println(th2);
        Thread th3 = new Thread("Jtc");
```

```

        System.out.println(th3);
        for (int i = 0; i < 5; i++) {
            System.out.println(Thread.currentThread().getName() + "
under the thread group of : " + Thread.
currentThread().getThreadGroup().getName());
            try{
                Thread.sleep(500);
            }catch(Exception e){
                e.printStackTrace();
            }
        }
    }
}

```

Following are the task of main thread

- 1) Main thread internally will get the name of the main() method by using the reflection mechanism.
- 2) Internally main thread will create a default object of type java.lang.Class.
- 3) It creates one string object in order to pass that as an parameter to main() method.
- 4) Sets command line argument to the main() method if any of the command line is available.
- 5) It calls the main method by passing the string reference as an parameter after completing the specified task thread will move to dead state.
- 6) All the child threads will be created from main thread itself.

17.3 User define thread

A user can develop or write own thread also. The user defines thread can be created in mainly two ways:

1) By extending the java.lang.Thread class.

Steps to write the User Define Thread by extending the Thread class

- i. Write your User Define Thread by extending the Thread class.
- ii. Declare some instance variables inside the Thread class.
- iii. Declare constructor inside the Thread class.
- iv. In order to invoke the constructor from the thread class we need to use the super statement and immediately after that we need to start each thread by writing the start() method.

- v. In order to process the thread we need to write the run() method implementation inside your class but when you are creating the User Define Thread by extending the Thread class then it is not mandatory to override the run() method inside.

Example

```
class HelloThread extends Thread{
    public HelloThread(){
        super();
        start();
    }
    public HelloThread(String tname){
        super(tname);
        start();
    }
    HelloThread(ThreadGroup tg,String tname){
        super(tg,tname);
        start();
    }
    public void run(){
        for (int i = 0; i < 5; i++) {
            System.out.println(Thread.currentThread().getName() + " under
the thread group of : "
                + Thread.currentThread().getThreadGroup().getName());
            try{
                Thread.sleep(500);
            }catch(Exception e){
                e.printStackTrace();
            }
        }
    }
}

public class Jtc183 {
    public static void main(String[] args) {
        ThreadGroup tg=Thread.currentThread().getThreadGroup();
        HelloThread th1=new HelloThread();
        HelloThread th2=new HelloThread("Jtc-1");
        HelloThread th3=new HelloThread(tg, "Jtc-2");
    }
}
```



```

        for (int i = 0; i < 5; i++) {
            System.out.println(Thread.currentThread().getName() + " under
the thread group of : "
                                + Thread.currentThread().getThreadGroup().getName());
            try{
                Thread.sleep(500);
            }catch(Exception e){
                e.printStackTrace();
            }
        }
    }
}

```

Q1 : Why it is not necessary to override run () method while extending the thread class?

Ans : Because run() method is already been overridden in thread class with no implementation but if you want to process thread according to you run() method is override.

2) By implementing the java.lang.Runnable Interface.

Steps to write the User Define Thread by implementing Runnable interface:-

- i. Write your thread class by implementing the runnable interface.
- ii. Define some instance variables if required.
- iii. Write the constructor inside your thread class.
- iv. Try to invoke the corresponding constructor from java.lang.Thread class.
- v. Start each thread by writing the start().
- vi. It is mandatory to override run() method your class which is available in Runnable interface.

Example

```

class HelloThread implements Runnable{
    HelloThread() {
        Thread t1=new Thread(this);
        t1.start();
    }
    HelloThread(String tname){
        Thread t2=new Thread(this,tname);
        t2.start();
    }
}

```

```

HelloThread(String tname,ThreadGroup tg){
    Thread t3=new Thread(tg,this,tname);
    t3.start();
}
public void run(){
    for (int i = 0; i < 5; i++) {
        System.out.println(Thread.currentThread().getName() + " under
the thread group of : "+ Thread.currentThread().getThreadGroup().getName());
        try{
            Thread.sleep(500);
        }catch(Exception e){
            e.printStackTrace();
        }
    }
}
}
}
public class Jtc103 {
    public static void main(String[] args) {
        ThreadGroup tg1=new ThreadGroup("Jtc-pg");
        HelloThread t1=new HelloThread("Jtc-TG");
        HelloThread t2=new HelloThread("Jtc-1");
    }
}
class MyThread extends Thread implements Runnable{
    void m1(){
        System.out.println("M1 in MyThread");
        String tname="Jtc-ch";
        Thread t1=new Thread(this);
        Thread t2=new Thread(this,tname);
        t1.start();
        t2.start();
    }
    public void run(){
        for(int i=0;i<5;i++){
            System.out.println(Thread.currentThread().getName());
            try{
                sleep(1000);
            }catch(Exception e){

```

```

        e.printStackTrace();
    }
}
}
}
public class Jtc184 {
    public static void main(String[] args) {
        MyThread mt=new MyThread();
        mt.m1();
    }
}

```

17.4 Thread Priority

This is a number which will be assigned to each thread by the JVM. Mainly three Thread priorities has been define:-

- 1) **MIN_PRIORITY**
- 2) **MAX_PRIORITY**
- 3) **NORM_PRIORITY**

Minimum priority of any thread will be 1.

Normal priority of any thread will be 5.

Maximum priority of any thread will be 10.

This thread priority will be used internally by the JVM to allocate the CPU time when CPU scheduler is follows the priority based algorithm. If CPU scheduler is not following any priority based algorithm then there is no use thread priority.

final void setPriority(int): By using the setPriority() method we can change the priority of the thread but you can't provide the thread priority more than 10 or less than 1.

final int getPriority() : By using this we can get the priority of a thread. This getPriority() is a final method which can'tbe override inside your class.

Q2 : Can I call run()?

Ans : Yes, we can call the run explicitly but this run() is not behave as an proper run() which would be control by CPU scheduler.

Q3 : Can I overload the run()?

Ans : Yes, we can overload the run() but the overloaded run() is not behave as an proper run() and it will not called automatically by the thread scheduling algorithm or by the thread.

Example

```
interface I1{
    void run();
    void start();
}

class Hello extends Thread implements I1{
    public Hello(){
        super();
        run();
    }
    public void start(){
        System.out.println("Run in Hello");
        System.out.println(Thread.currentThread().getName());
    }
    void m1(){
        System.out.println("m1 in Hello");
        Thread t1=new Thread(this);
        t1.start();
    }
}

public class Jtc185 {
    public static void main(String[] args) {
        Hello h1=new Hello();
        h1.m1();
        //h1.run();
    }
}
```

Q4 : Can I start a thread more than once?

Ans: We can start() that but it will throw exception java.lang.
IllegalThreadStateException.

Q5 : Suppose you are getting a requirement to develop your custom thread by implementing Runnable interface or extending Thread class

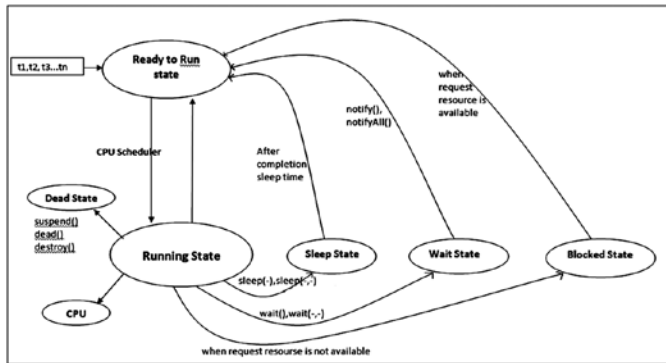
then which one you prefer and why?

Ans : It is always recommendable to write custom thread by implementing Runnable interface, so that you can achieve the multiple inheritance also where as if you are developing the custom thread by extending the Thread class then you will not be able to achieve the multiple inheritance.

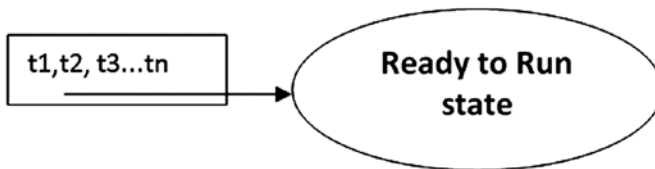
yield(): When you are using yield then it will give the preference to thread which is there in Ready to run state with the highest priority but this method will work only when if the CPU scheduler is following or processing the thread on the basis of priority based algorithm. But it is recommended not to use the yield() because there is no processing guaranty.

17.5 Thread Life Cycle

As we know that each thread will be process by life cycle mechanism, In order to complete a complete Thread processing has to be across the different, different state.



1) Start to Ready to Run state

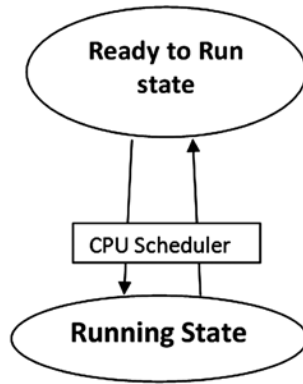


When we are starting the thread then it will just move to the ready to run state which means that thread is ready to move the Running state.

Note

When we are starting a thread it doesn't mean that directly thread is going to achieve the running state.

Once Thread is there in the ready to Run state with there some properties and specified task.



Now CPU scheduler is ready with the some scheduling algorithm which main task is to keep busy the Running state. That running state should not be idle. In CPU scheduler many different types of scheduling algorithm is being implemented, but which scheduling algorithm is currently running we are not sure about that , so in that scheduling algorithm which threads comes in will be processed for running states. Getting Running state means executing the specified task or got the CPU time.

Q6 : When exactly run() called?

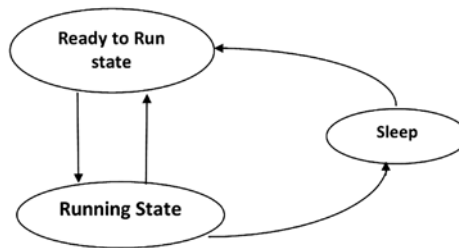
Ans : run() will be called just by starting the thread but it will automatically be called when ever CPU scheduler allocates the CPU time for that specific thread or whenever the thread gets the CPU time associated with that time the run() will be called.

Q7 : Can I call the run() explicitly?

Ans : Yes, we can call the run() explicitly but there will be not any proper use of that because according to JLS (Java language Specification) run() will be automatically called whenever threads get the CPU time.

Q8 : Suppose two threads is in Ready to Run state is having same priority and by chance CPU scheduler is processing the thread priority based algorithm then out of that two thread which one is process.

Ans : In that case, when more than one thread comes under the scheduling algorithm then will try to switch another scheduling algorithm. On the basis of that it can make some specific thread. We cannot guess that which scheduling algorithm will be adopted by the CPU scheduler



2)

If a thread is running in the Running state, when we are calling the sleep() by specifying some parameter then that running thread will move to sleep state for specified amount of time and in that sleeping time no one can interrupt the thread. After completing the sleep time automatically the thread will move to the Ready to Run State and will wait for CPU time.

It will throw an exception i.e. InterruptedException.

1) public static void sleep(long) throws InterruptedException

2) public static void sleep(long , int) throws InterruptedException

Sleep (long) it represents the time in millisecond.

This sleep() is available inside Thread class. When you are calling this method , then specified thread will move to the sleep state for the specified amount of time, between that time duration the thread cannot be disturb.

sleep(long , int)
 ↓ ↓
 Millisecond + Nanosecond

In the case of `sleep()`, first long parameter will be considered as millisecond and second int parameter considered as nanosecond and by adding millisecond & nanosecond the resultant amount of time is sleeping duration, after completing that duration it will move to the run state. In order to get CPU time again it has to follow the CPU scheduler.

4)



When a thread is moving to waiting state then some points you should remember:

- **Wait():** When a thread is available in Running state and `wait()` is being called on that to that thread will move to the wait state.
- **Wait():** When you are calling `wait` without parameter, then thread will move waiting state for the un- specified amount of time until the time you are not calling `notify()` or `notifyAll()`.
- **Wait(long):** Then that running thread will move to the wait state for the specified amount of time and once the specified time duration is completed then automatically thread will move to the ready to run state. But in case if you want to move the waiting thread to ready to run state before completing that time duration then you can notify that thread to ready to run state.
- **Wait(long, int):** When you are calling method `(long, int)` millisecond and nanosecond and by adding that duration the resultant amount of time it will wait in the waiting state and after completing the time duration automatically it will move to the ready to run state.

Q9 : Can I notify the waiting thread before completion of the given time duration?

Ans : Yes, we can do that by using `notify()` and `notifyAll()`. We can move thread to the ready to run state.

Q10 : Can I ask to specific thread to wait or sleep?

Ans : No, we cannot ask any thread to wait or sleep because there is no such type of specification which thread is currently in the process.

Q 11: Suppose there are following threads which is there in the wait state, let say t1, t2, t3, t4. Can I notify any specific thread out of that.

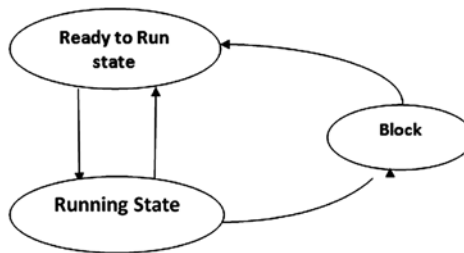
Ans: No, we cannot.

notify(): notify() is used to notify the thread which is there in the wait state from the longest time.

notifyAll(): When we are using notifyAll() then all the thread which is there in the waiting state to ready to run state in a specific order, which is waiting from a long time to move first.

5)

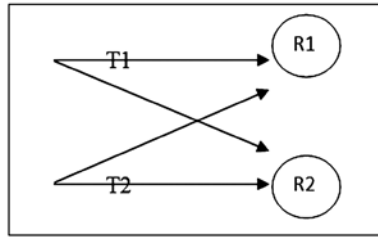
When a thread is moving to running state in order to process some task or when a thread is requesting for any specific resource and that resource is not available



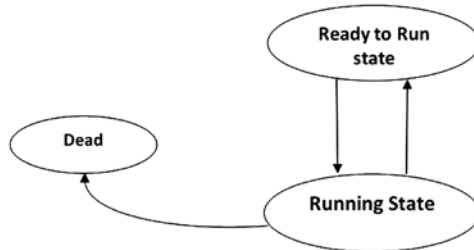
then the thread will move to the block state until the requested resource is not available. The thread which is there in the block state will move to the ready to run state only when that requested resource is available, in this cases it may cause the dead lock condition also.

Deadlock Condition

Lets take condition, there are two threads t1 and t2 are holding resource R1 and R2 respectively. Thread t1 is requesting for resource R2 and thread t2 is requesting for R1. Thread t1 will release resource R1 only after getting the resource R2 an thread t2 release R2 only after getting the resource R1.



6)



When we are calling `destroy()` or `stop()` then that thread will move to the dead state and once the thread is dead it cannot be restarted or Once the thread is move to the dead state again it can't go to the Ready to run state. When some task is assigned to the thread, if that thread is completed the task then automatically the thread will move the dead state.

Join(): When `join()` method , we are calling this method with some specific thread then currently running thread will move to the wait state and it will give the chance to run the thread with which we have called `join()`.

Join(long): When we are calling `join()` by providing some parameter or by specifying parameter then that thread will wait for specified amount of time to complete the running thread and immediately after that it will move the wait state.

Example

```

class JtcThread1 extends Thread {
    public void run() {
        for (int i = 0; i < 15; i++) {
            System.out.println("JT1\t" + i);
            try {
                Thread.sleep(100);
            } catch (Exception e) {
                e.printStackTrace();
            }
        }
    }
}

class JtcThread2 extends Thread {

```

```

Thread th = null;
void setThreadToJoin(Thread th) {
    this.th = th;
}
public void run() {
    for (int i = 0; i < 115; i++) {
        System.out.println("JT2\t" + i);
        try {
            if (i == 105) {
                th.join();
            }
            Thread.sleep(100);
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
}
}
public class Jtc189 {
    public static void main(String[] args) {
        JtcThread1 ath=new JtcThread1();
        JtcThread2 bth=new JtcThread2();
        bth.setThreadToJoin(ath);
        ath.start();
        bth.start();
    }
}
}

```

17.6 Synchronization

It is the process of blocking the current object. This can be done in two ways:

1) Method Level Synchronization: In order to achieve this synchronization we need to declare it.

Case 1: There are two threads t2, t2 is wait to access non synchronized method M1 with the object h1 concurrency will be there.

Example

```
class Account{
```

```

        synchronized void withdraw(){
            for(int i=0;i<5;i++){
                System.out.println("Withdraw \t:" + Thread.currentThread().
getName());
                try{
                    Thread.sleep(500);
                }catch(Exception e){
                    e.printStackTrace();
                }
            }
        }
        void deposit(){
            for(int i=0;i<5;i++){
                System.out.println("Deposit \t :"+Thread.currentThread().
getName());
            }
        }
    }
}

class AThread implements Runnable{
    Account acc=null;
    public AThread(Account acc,String tname){
        this.acc=acc;
        Thread t1=new Thread(this, tname);
        t1.start();
    }
    public void run(){
        acc.withdraw();
    }
}

class BThread implements Runnable{
    Account acc=null;
    public BThread(Account acc,String tname){
        this.acc=acc;
        Thread t2=new Thread(this,tname);
        t2.start();
    }
}

```

```

        public void run(){
            acc.deposite();
        }
    }
}

public class Jtc190 {
    public static void main(String[] args) {
        Account acc1=new Account();
        Account acc2=new Account();
        AThread ath1=new AThread(acc1, "A");
        BThread bth1=new BThread(acc2, "B");
    }
}

```

Case 2: There are two threads t1 and t2 are trying to access non-synchronized method m1 with object h1 and h2, concurrency will be there.

Case 3: There are two threads t1 and t2 are trying to access non-synchronized method m1 and m2 with the same object, concurrency will be there.

Case 4: There are two threads t1 and t2 are trying to access non-synchronized method m1 and m2 with the two different object h1 and h2 respectively, concurrency will be there.

Case 5: There are two threads t1 and t2 are trying to access synchronized method m1 with the same object h1 accessing will be one by one.

Case 6: There are two threads t1 and t2 are trying to access synchronized method m1 with the two different object h1 and h2 respectively, concurrency will be there.

Case 7: There are two threads t1 and t2 are trying to access synchronized method m1 and m2 with the same object one by one processing.

Case 8: There are two threads t1 and t2 are trying to access synchronized method m1 and m2 with the two different object h1 and h2 respectively, concurrency will be there.

Case 9: There are two threads t1 and t2 are trying to access synchronized method m1 and non-synchronized method m2 with the same object h1, concurrency will be there.

Case 10: There are two threads t1 and t2 are trying to access synchronized method m1 and non-synchronized method m2 with two different object h1 and

h2, concurrency will be there. In case of static method or static synchronized method default object will be lock of type `java.lang.Class`.

Case 11: There are two threads t1 and t2 are trying to access static method m1 with the same object then what will happen, concurrency will be there.

Case 12: There are two threads t1 and t2 are trying to access static method m1 with the two different objects respectively, concurrency will be there.

Case 13: There are two threads t1 and t2 are trying to access static method m1 and m2 with the same object, concurrency will be there.

Case 14: Two threads t1 and t2 static synchronized m1 with same object then what will happen, one by one.

Case 15: There are two threads t1 and t2 are trying to access static synchronized object with two different objects, one by one.

Object locking

When you are creating an object by default or internally lock will be available on that but that lock will not be visible to any of the thread, so accessing with the object does not affect the concurrency of any application.

That lock are the implicit lock available with the object will be visible only when that object is accessing any synchronized context.

Remember the following points

- 1) When you are accessing any synchronized method that only that lock will be visible to the corresponding object.
- 2) In the case of static synchronized method default object of type `java.lang.Class` will be lock.
- 3) In the case block level synchronization 3rd party object will be lock.

Daemon Thread

This is a service thread which is used to provide the service for the for main thread which is currently running. Demon thread will exist till the time main threads are executing. Once the main thread has completed the task, demon thread will automatically be destroyed.

Program 17.1

```

package com.p4;
class JtcThread1 extends Thread{
    public void run(){
        for (int i = 0; i < 15; i++) {
            System.out.println("JTH1\t"+i);
            try{
                sleep(100);
            }catch(Exception e){
                e.printStackTrace();
            }
        }
    }
}
class JtcThread2 extends Thread{
    Thread th=null;
    void setThreadToJoin(Thread th){
        this.th=th;
    }
    public void run(){
        for (int i = 0; i < 115; i++) {
            System.out.println("JTH2\t"+i);
            try{
                if(i==105){
                    th.join();
                }
            }catch(Exception e){
                e.printStackTrace();
            }
        }
    }
}
public class Jtc8 {
    public static void main(String[] args) {
        JtcThread1 ath=new JtcThread1();
        JtcThread2 bth=new JtcThread2();
        bth.setThreadToJoin(ath);
    }
}

```

```

        ath.start();
        bth.start();
    }
}

```

2) Block Level Synchronization: In the case of block level synchronization only third object will be lock.

```

Void m1(){
Statement-1;
Statement-2;
synchronized(object){
Statement-3;
Statement-4;
}
Statement-5;
Statement-6;
}

```

Program 17.2

Program: Example using Block Level Synchronization

```

package com.p2;
class HelloJtc{
    synchronized void m1(){
        for(int i=1;i<=5;i++){
            System.out.println("m1-"+i);
            try{
                Thread.sleep(1000);
            }catch(Exception e){

            }
        }
    }
    synchronized void m2(){
        for(int i=10;i<=15;i++){
            System.out.println("m2-"+i);
            try{
                Thread.sleep(1000);
            }catch(Exception e){

```



```

        }
    }
}

class A implements Runnable{
    HelloJtc h=null;
    A(HelloJtc h,String name){
        this.h=h;
        Thread t=new Thread(this,name);
        t.start();
        System.out.println(t.getThreadGroup().getName());
    }
    public void run(){
        h.m1();
    }
}

class B implements Runnable{
    HelloJtc h=null;
    B(HelloJtc h,String name){
        this.h=h;
        Thread t2=new Thread(this,name);
        t2.start();
        System.out.println(t2.getThreadGroup().getName());
    }
    public void run(){
        h.m2();
    }
}

public class Jtc5 {
    public static void main(String[] args) {
        HelloJtc h=new HelloJtc();
        new A(h,"A");
        new B(h,"B");
    }
}

```

Q12 : Can I declare run method as synchronize?

Ans : Yes, we can because run() method already work as synchronized manner.

Q 13: Can I call run() method explicitly?

Ans : Yes, but no use of that.

isDaemon(): It is used to check whether the thread is Daemon or not. If it is daemon thread then it returns true or else it will return false.

setDaemon(Boolean): If you want to set any thread as a Daemon the this method is used and pass the value as true.

From JDK1.5 stop(), suspend() and destroy() has been deprecated. Whenever thread is going into sleep state then it will not release the lock but in wait state it will release the lock state.

17.7 Inter Thread Communication**Difference between sleeping Thread & wait Thread**

There are two threads t1 and t2 both are accessing synchronized method consider when thread t1 is accessing method m1() and that thread t1 is being asked to sleep, then thread t1 is going to sleep state without releasing the lock where as t2 assess synchronize method m2() and being asked to go for the wait then that will release the lock.

Example

```
class Stack{
    int x;
    boolean flag=false;
    public synchronized void push(int x){
        if(flag){
            try{
                wait();
            }catch(Exception e){
                System.out.println(e);
            }
        }
        this.x=x;
        System.out.println(x+" is pushed...");
    }
}
```

```

        flag=true;
        notify();
    }
    synchronized public int pop(){
        if(!flag){
            try{
                wait();
            }catch(Exception e){
                System.out.println(e);
            }
        }
        System.out.println(x+" is popped");
        try{
            Thread.sleep(2000);
        }catch(Exception e){
            System.out.println(e);
        }
        flag=false;
        notify();
        return x;
    }
}

class A1 implements Runnable{
    Stack st=null;
    A1(Stack st,String name){
        this.st=st;
        Thread t1=new Thread(this,name);
        t1.start();
    }
    public void run(){
        int a=1;
        for(int i=0;i<7;i++){
            st.push(a++);
        }
    }
}

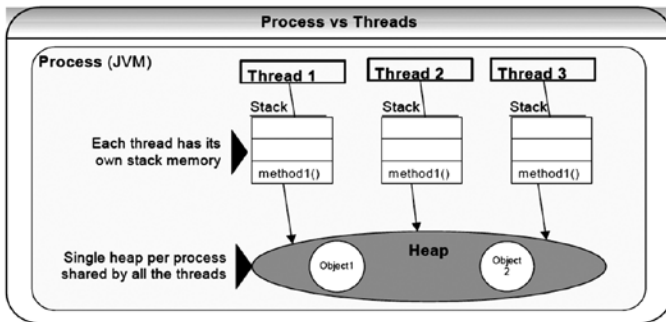
```

```

class B1 implements Runnable{
    Stack st=null;
    B1(Stack st,String name){
        this.st=st;
        Thread t2=new Thread(this,name);
        t2.start();
    }
    public void run(){
        for(int i=0;i<7;i++){
            st.pop();
        }
    }
}

public class Jtc194 {
    public static void main(String[] args) {
        Stack st=new Stack();
        A1 obj1=new A1(st, "A");
        B1 obj2=new B1(st, "B");
    }
}

```



Q14 : What is the difference between processes and threads?

Ans : A process is an execution of a program but a thread is a single execution sequence within the process. A process can contain multiple threads. A thread is sometimes called a lightweight process.

A JVM runs in a single process and threads in a JVM share the heap belonging to that process. That is why several threads may access the same object. Threads **share the heap and have their own stack space**. This is how one thread's invocation of

a method and its local variables are kept thread safe from other threads. But the heap is not thread-safe and must be synchronized for thread safety.

Q15 : What is the difference between yield and sleeping? What is the difference between the methods sleep() and wait()?

Ans15 : ?

Q16 : What does join() method do?

t1.join() allows the current thread to wait indefinitely until thread “t1” is finished.

t1.join (5000) allows the current thread to wait for thread “t1” to finish but does not wait longer than 5 seconds.

```
try {
```

```
t.join(5000); //current thread waits for thread “t” to complete but does not wait more than 5 sec
```

```
if(t.isAlive()){
```

```
//timeout occurred. Thread “t” has not finished
```

```
}else {
```

```
//thread “t” has finished
```

```
}}
```

17.8 Lazy thread

- If we are commenting line 1 then both main and child Threads are non daemon and hence both will be executed until they completion.
- If we are not commenting line 1 then main Thread is non daemon and child
- Thread is daemon and hence whenever main Thread terminates automatically child Thread will be terminated.

17.9 Deadlock vs Starvation

- A long waiting of a Thread which never ends is called deadlock.
- A long waiting of a Thread which ends at certain point is called starvation.
- A low priority Thread has to wait until completing all high priority Threads. This long waiting of Thread which ends at certain point is called starvation.

17.10 How to kill a Thread in the middle of the line?

- We can call stop() method to stop a Thread in the middle then it will be entered into dead state immediately.
- `public final void stop();`

- stop() method has been deprecated and hence not recommended to use.

17.11 suspend and resume methods

- A Thread can suspend another Thread by using suspend() method then that
- Thread will be paused temporarily.
- A Thread can resume a suspended Thread by using resume() method then suspended Thread will continue its execution.
 - o public final void suspend();
 - o public final void resume();
- Both methods are deprecated and not recommended to use.

17.12 RACE condition

- Executing multiple Threads simultaneously and causing data inconsistency problems is nothing but Race condition
- we can resolve race condition by using synchronized keyword.

17.13 ThreadLocal (1.2 v)

- We can use ThreadLocal to define local resources which are required for a particular Thread like DBConnections, counterVariables etc.
- We can use ThreadLocal to define Thread scope like Servlet Scopes(page, request, session, application).

17.14 GreenThread

- Java multiThreading concept is implementing by using the following 2 methods :
 - o GreenThread Model
 - o Native OS Model

GreenThread Model

- The threads which are managed completely by JVM without taking support for underlying OS, such type of threads are called Green Threads.

Native OS Model

- The Threads which are managed with the help of underlying OS are called
- Native Threads.
- Windows based OS provide support for Native OS Model
- Very few OS like SunSolaries provide support for GreenThread Model
- Anyway GreenThread model is deprecated and not recommended to use.

17.15 Thread Local

- ThreadLocal Provides ThreadLocal Variables.
- ThreadLocal Class Maintains Values for Thread Basis.
- Each ThreadLocal Object Maintains a Separate Value Like userID, transactionID etc for Each Thread that Accesses that Object.
- Thread can Access its Local Value, can Manipulates its Value and Even can Remove its Value.
- In Every Part of the Code which is executed by the Thread we can Access its Local Variables.

Example

- Consider a Servlet which Calls Some Business Methods. we have a Requirement to generate a Unique transactionID for Each and Every Request and we have to Pass this transactionID to the Business Methods for Logging Purpose.
- For this Requirement we can Use ThreadLocal to Maintain a Separate transactionID for Every Request and for Every Thread.

Note

- ThreadLocal Class introduced in 1.2 Version.
- ThreadLocal can be associated with Thread Scope.
- All the Code which is executed by the Thread has Access to Corresponding ThreadLocal Variables.
- A Thread can Access its Own Local Variables and can't Access Other Threads Local Variables.
- Once Thread Entered into Dead State All Local Variables are by Default Eligible for Garbage Collection.

17.16 Constructor

`ThreadLocal tl = new ThreadLocal();` Creates a ThreadLocal Variable.

Methods

1) Object get(); Returns the Value of ThreadLocal Variable associated with Current Thread.

2) Object initialValue();

Returns the initialValue of ThreadLocal Variable of Current Thread. The Default Implementation of initialValue() Returns null.

To Customize Our initialValue we have to Override initialValue().

3) void set(Object newValue);To Set a New Value.

4) void remove();

To Remove the Current Threads Local Variable Value.

After Remove if we are trying to Access it will be reinitialized Once Again by invoking its initialValue().

This Method Newly Added in 1.5 Version.

Example

```
class ThreadLocalDemo {
    public static void main(String[] args) {
        ThreadLocal tl = new ThreadLocal();
        System.out.println(tl.get()); //null
        tl.set("JTC");
        System.out.println(tl.get()); //JTC
        tl.remove();
        System.out.println(tl.get()); //null
    }
    //Overriding of initialValue()
    class ThreadLocalDemo {
        public static void main(String[] args) {
            ThreadLocal tl = new ThreadLocal() {
                protected Object initialValue() {
                    return "abc";
                }
            };
            System.out.println(tl.get()); //abc
            tl.set("JTC");
            System.out.println(tl.get()); //JTC
            tl.remove();
            System.out.println(tl.get()); //abc
        }
    }
}
```

17.17 Java.util.concurrent.locks package

Problems with Traditional synchronized Key Word

- If a Thread Releases the Lock then which waiting Thread will get that Lock we are
- Not having any Control on this.
- We can't Specify Maximum waiting Time for a Thread to get Lock so that it will

- Wait until getting Lock, which May Effect Performance of the System and Causes Dead Lock.
- We are Not having any Flexibility to Try for Lock without waiting. There is No API to List All Waiting Threads for a Lock.
- The synchronized Key Word Compulsory we have to Define within a Method and it is Not Possible to Declare Over Multiple Methods.
- To Overcome Above Problems SUN People introduced java.util.concurrent.locks
- Package in 1.5 Version.
- It Also Provides Several Enhancements to the Programmer to Provide More Control on Concurrency.

17.18 Lock(I)

- A Lock Object is Similar to Implicit Lock acquired by a Thread to Execute synchronized Method OR synchronized Block
- Lock Implementations Provide More Extensive Operations than Traditional Implicit Locks.

Important Methods of Lock Interface

1) void lock();

It Locks the Lock Object.

If Lock Object is Already Locked by Other Thread then it will wait until it is Unlocked.

2) boolean tryLock();

To Acquire the Lock if it is Available.

If the Lock is Available then Thread Acquires the Lock and Returns true. If the Lock Unavailable then this Method Returns false and Continue its

Execution.

In this Case Thread is Never Blocked.

```
if (l.tryLock()) {
    Perform Safe Operations
}
else {
    Perform Alternative Operations
}
```

3) `boolean tryLock(long time, TimeUnit unit);`

- o To Acquire the Lock if it is Available.
- o If the Lock is Unavailable then Thread can Wait until specified Amount of Time. Still if the Lock is Unavailable then Thread can Continue its Execution.

eg: `if (l.tryLock(1000, TimeUnit.SECONDS)) {}`

TimeUnit: `TimeUnit` is an enum Present in `java.util.concurrent` Package.

```
enum TimeUnit {
    DAYS, HOURS, MINUTES, SECONDS, MILLI SECONDS, MICRO SECONDS,
    NANO SECONDS;
}
```

4) `void lockInterruptibly();`

- Acquired the Lock Unless the Current Thread is Interrupted. Acquires the Lock if it is Available and Returns Immediately.
- If it is Unavailable then the Thread will wait while waiting if it is Interrupted then it won't get the Lock.

5) `void unlock();` To Release the Lock.**17.19 ReentrantLock**

- It implements Lock Interface and it is the Direct Child Class of an Object.
- Reentrant Means a Thread can acquires Same Lock Multiple Times without any Issue.
- Internally ReentrantLock Increments Threads Personal Count whenever we Call `lock()` and Decrements Counter whenever we Call `unlock()` and Lock will be Released whenever Count Reaches '0'.

1) `ReentrantLock rl = new ReentrantLock();`

Creates an Instance of ReentrantLock.

2) `ReentrantLock rl = new ReentrantLock(boolean fairness);`

- Creates an Instance of ReentrantLock with the Given Fairness Policy.
- If Fairness is true then Longest Waiting Thread can acquired Lock Once it is
- Available i.e. if follows First - In First – Out.
- If Fairness is false then we can't give any Guarantee which Thread will get the
- Lock Once it is Available.

Note: If we are Not specifying any Fairness Property then by Default it is Not Fair.

Which of the following 2 Lines are Equal?

```
ReentrantLockrl = new ReentrantLock();
```

```
ReentrantLockrl = new ReentrantLock(true);
```

```
ReentrantLockrl = new ReentrantLock(false); ✓
```

Important Methods of ReentrantLock

- 1) `void lock();`
- 2) `booleantryLock();`
- 3) `booleantryLock(long l, TimeUnit t);`
- 4) `void lockInterruptedly();`
- 5) **`void unlock();`**

To Release the Lock.

If the Current Thread is Not Owner of the Lock then we will get Runtime

Exception Saying `IllegalMonitorStateException`.

- 6) **`intgetHoldCount();`** Returns Number of Holds on this Lock by Current Thread.
- 7) **`booleanisHeldByCurrentThread();`** Returns true if and Only if Lock is Hold by Current Thread.
- 8) **`intgetQueueLength();`** Returns the Number of Threads waiting for the Lock.
- 9) **`Collection getQueuedThreads();`** Returns a Collection containing Thread Objects which are waiting to get the Lock.
- 10) **`booleanhasQueuedThreads();`** Returns true if any Thread waiting to get the Lock.
- 11) **`booleanisLocked();`** Returns true if the Lock acquired by any Thread.
- 12) **`booleanisFair();`** Returns true if the Lock's Fairness Set to true.
- 13) **`Thread getOwner();`** Returns the Thread which acquires the Lock.

Example

```
import java.util.concurrent.locks.ReentrantLock;
class Test {
    public static void main(String[] args) {
        ReentrantLock l = new ReentrantLock();
        l.lock();
        l.lock();
        System.out.println(l.isLocked()); //true
        System.out.println(l.isHeldByCurrentThread()); //true
        System.out.println(l.getQueueLength()); //0
        l.unlock();
```

```

System.out.println(l.getHoldCount()); //1
System.out.println(l.isLocked()); //true
l.unlock();
System.out.println(l.isLocked()); //false
System.out.println(l.isFair()); //false
}}

```

Program 17.3

```

import java.util.concurrent.locks.ReentrantLock;
class Display {
    ReentrantLock l = new ReentrantLock(true);
    public void wish(String name) {
        l.lock();
        for(int i=0; i<5; i++) {
            System.out.println("Good Morning");
            try {
                Thread.sleep(2000);
            }
            catch(InterruptedException e) {}
            System.out.println(name);
        }
        l.unlock();
    }
}
class MyThread extends Thread {
    Display d;
    String name;
    MyThread(Display d, String name) {
        this.d = d;
        this.name = name;
    }
    public void run() {
        d.wish(name);
    }
}
class ReentrantLockDemo {
    public static void main(String[] args) {
        Display d = new Display();
        MyThread t1 = new MyThread(d, "Dhoni");
        MyThread t2 = new MyThread(d, "Yuva Raj");
    }
}

```

```

MyThread t3 = new MyThread(d, "ViratKohli");
t1.start();
t2.start();
t3.start();
}}

```

- If we Comment Both Lines 1 and 2 then All Threads will be executed Simultaneously and Hence we will get Irregular Output.
- If we are Not Commenting then the Threads will be executed One by One and Hence we will get Regular Output

Example

Demo Program To Demonstrate tryLock();

```

import java.util.concurrent.locks.ReentrantLock;
class MyThread extends Thread {
    static ReentrantLock l = new ReentrantLock();
    MyThread(String name) {
        super(name);
    }
    public void run() {
        if(l.tryLock()) {
            System.out.println(Thread.currentThread().getName()+" Got Lock and Performing Safe Operations");
            try{
                Thread.sleep(2000);
            }
            catch(InterruptedException e) {}
            l.unlock();
        }
        else {
            System.out.println(Thread.currentThread().getName()+" Unable To Get Lock and Hence Performing Alternative Operations");
        }
    }
}
class ReentrantLockDemo {
    public static void main(String args[]) {
        MyThread t1 = new MyThread("First Thread");
        MyThread t2 = new MyThread("Second Thread");
        t1.start();

```

```

t2.start();
}}
import java.util.concurrent.TimeUnit;
import java.util.concurrent.locks.ReentrantLock;
class MyThread extends Thread {
    static ReentrantLock l = new ReentrantLock();
    MyThread(String name) {
        super(name);
    }
    public void run() {
        do {
            try {
                if(l.tryLock(1000, TimeUnit.MILLISECONDS)) {
                    SOP(Thread.currentThread().getName()+"----- Got Lock");
                    Thread.sleep(5000);
                    l.unlock();
                    SOP(Thread.currentThread().getName()+"----- Releases Lock");
                    break;
                }
            }
            else {
                SOP(Thread.currentThread().getName()+"----- Unable To Get Lock And Will Try Again");
            }
        }
        catch(InterruptedException e) {}
    }
    while(true);
}
class ReentrantLockDemo {
    public static void main(String args[]) {
        MyThread t1 = new MyThread("First Thread");
        MyThread t2 = new MyThread("Second Thread");
        t1.start();
        t2.start();
    }
}

```

17.20 Thread Pools

- Creating a New Thread for Every Job May Create Performance and Memory Problems.
- To Overcome this we should go for Thread Pool Concept.
- Thread Pool is a Pool of Already Created Threads Ready to do Our Job.
- Java 1.5 Version Introduces Thread Pool Framework to Implement Thread Pools. Thread Pool Framework is Also Known as Executor Framework.
- We can Create a Thread Pool as follows
 - o `ExecutorService service = Executors.newFixedThreadPool(3);`
 - o `//Our Choice We can Submit a Runnable Job by using submit().`
 - o `service.submit(job);`
- We can ShutdownExecutorService by using `shutdown(). service.shutdown();`

Example

```
import java.util.concurrent.ExecutorService;
import java.util.concurrent.Executors;
class PrintJob implements Runnable {
String name;
PrintJob(String name){
this.name = name;
}
public void run(){
SOP(name+"....Job Started By Thread:" +Thread.currentThread().getName());
try {
Thread.sleep(10000);
}
catch (InterruptedException e) {}
SOP(name+"....Job Completed By Thread:" +Thread.currentThread().getName());
}}
class ExecutorDemo {
public static void main(String[] args) {
PrintJob[] jobs = {
newPrintJob("Som"),
newPrintJob("Prakash"),
newPrintJob("Rai"),
newPrintJob("Java"),
newPrintJob("Training"),
newPrintJob("Center")
}
```

```
};
ExecutorService service = Executors.newFixedThreadPool(3);
for (PrintJob job : jobs) {
    service.submit(job);
}
service.shutdown();
}}
```

On the Above Program 17.3 Threads are Responsible to Execute 6 Jobs. So that a Single Thread can be reused for Multiple Jobs.

Note: Usually we can Use ThreadPool Concept to Implement Servers (Web Servers And Application Servers).

17.21 Callable and Future

- In the Case of Runnable Job Thread won't Return anything.
- If a Thread is required to Return Some Result after Execution then we should go for Callable.
- Callable Interface contains Only One Method public Object call() throws Exception. If we Submit a Callable Object to Executor then the Framework Returns an Object of Type java.util.concurrent.Future
- The Future Object can be Used to Retrieve the Result from Callable Job.

Example

```
import java.util.concurrent.Callable;
import java.util.concurrent.ExecutorService;
import java.util.concurrent.Executors;
import java.util.concurrent.Future;
class MyCallable implements Callable{
    int num;
    MyCallable(int num) {
        this.num = num;
    }
    public Object call() throws Exception {
        int sum = 0;
        for(int i=0; i<num; i++) {
            sum = sum+i;
        }
        return sum;
    }
}
```



```

}}
class CallableFutureDemo {
public static void main(String args[]) throws Exception {
MyCallable[] jobs = {
newMyCallable(10),
newMyCallable(20),
newMyCallable(30),
newMyCallable(40),
newMyCallable(50),
newMyCallable(60)
};
ExecutorService service = Executors.newFixedThreadPool(3);
for(MyCallable job : jobs){
Future f = service.submit(job);
System.out.println(f.get());
}
service.shutdown();
}

```

Program: Examples on Method level Synchronization

```

package com.p1;
class HelloJtc{
    synchronized void m1(){
        for (int i = 1; i <=5; i++) {
            System.out.println("m1-"+i);
            try{
                Thread.sleep(1000);
            }catch(Exception e){
            }
        }
    }
    synchronized void m2(){
        for (int i = 10; i <=15; i++) {
            System.out.println("m2-"+i);
            try{
                Thread.sleep(1000);
            }catch(Exception e){
            }
        }
    }
}

```

```

        }
    }
}

class A implements Runnable{
    HelloJtc h=null;
    A(HelloJtc h,String name){
        this.h=h;
        Thread t=new Thread(this,name);
        t.start();
        System.out.println(t.getThreadGroup().getName());
    }
    public void run(){
        h.m1();
    }
}

class B implements Runnable{
    HelloJtc h=null;
    B(HelloJtc h,String name){
        this.h=h;
        Thread t2=new Thread(this,name);
        t2.start();
        System.out.println(t2.getThreadGroup().getName());
    }
    public void run(){
        h.m2();
    }
}

public class Jtc4 {
    public static void main(String[] args) {
        HelloJtc h=new HelloJtc();
        new A(h,"A");
        new B(h,"B");
    }
}

```

Program: Example using simple sleep and run method

```
package com.p2;
```

```

class Account{
    int bal=970;
    public void withdraw(int amt){
        if(bal>=amt){
            System.out.println(Thread.currentThread().getName()+"
is going to withdraw..." +bal);
            try{
                Thread.sleep(1200);
            }catch(Exception e){
            }
            bal=amt;
            System.out.println(Thread.currentThread().
getName()+"is Complete withdraw..." +bal);
        }else{
            System.out.println("No Funds for "+Thread.
currentThread().getName());
        }
    }
    public int getBal(){
        return bal;
    }
}

class AccThread implements Runnable{
    Account acc=null;
    public AccThread(Account acc) {
        // TODO Auto-generated constructor stub
        this.acc=acc;
        Thread t1=new Thread(this,"A");
        Thread t2=new Thread(this,"B");
        t1.start();

```

```

        t2.start();
    }
    public void run(){
        for(int i=0;i<5;i++){
            acc.withdraw(225);
            if(acc.getBal()<0){
                System.out.println("Amount is Overdrawn....");
            }
        }
    }
}

public class Jtc6 {
    public static void main(String[] args) {
        Account acc=new Account();
        new AccThread(acc);
    }
}

```

Program: Example using Wait(), notify(), notifyAll()

```

package com.p3;
class Stack{
    int x;
    boolean flag=false;
    public synchronized void push(int x){
        if(flag){
            try{
                wait();
            }catch(Exception e){
                System.out.println(e);
            }
        }
        this.x=x;
        System.out.println(x+" is pushed");
    }
}

```

```

        flag=true;
        notify();
    }
    synchronized public int pop(){
        if(!flag){
            try{
                wait();
            }catch(Exception e){
                System.out.println(e);
            }
        }
        System.out.println(x+" is popped");
        try{
            Thread.sleep(2000);
        }catch(Exception e){
            System.out.println(e);
        }
        flag=false;
        notify();
        return x;
    }
}

class A1 implements Runnable{
    Stack st=null;
    public A1(Stack st,String name) {
        // TODO Auto-generated constructor stub
        this.st=st;
        Thread t1=new Thread(this,name);
        t1.start();
    }
    public void run() {
        // TODO Auto-generated method stub
        int a=1;
        for(int i=0;i<7;i++){
            st.push(a++);
        }
    }
}

```

```

class B1 implements Runnable{
    Stack st=null;
    public B1(Stack st,String name) {
        // TODO Auto-generated constructor stub
        this.st=st;
        Thread t2=new Thread(this,name);
        t2.start();
    }
    public void run() {
        // TODO Auto-generated method stub
        int a=1;
        for(int i=0;i<7;i++){
            st.pop();
        }
    }
}

public class Jtc7 {
    public static void main(String[] args) {
        Stack st=new Stack();
        B1 obj2=new B1(st,"B");
        A1 obj1=new A1(st,"A");
    }
}

```

Chapter 18

Collections

18.1 Java.util package or Collection Framework

In java when we have trying to store some elements inside the Array then we may get the following problems:

- Array is static in nature i.e. in order to store elements inside the Array first we need to declare the size.
- If we are storing some elements inside the array which might not fulfill the complete length of the Array so in this case memory wastage is being done.
- In case if you are exceeding the size of the Array then it will throw an exception ie java.lang.Array index out of bounds exception.
- Before declaring any Array first we need to declare the type of the Array i.e. which type of you wanted to declare and in that Array you can store only that same type of element. But in some cases we can store the different types of elements inside the Array.

For ex: `Object O[] =new Object [3];`
 `O[]=new Hail();`
 `O[1]=new Hello();`
 `O[2]=new String("AB");`

While accessing the array we need to initialize the computer perform the conditional check & increment or decrement the counter. These many task you need to perform while accessing the array. Suppose there is an array which is having length of 1000&in that only some hundred elements are available so even though you need to iterate the loop thousand no of times unnecessarily which is of no use.

- If you wanted to delete any specific index position element from the array then it is very difficult.
- In case of array we can't avoid the duplicacy of the elements.
- In array elements will be store in added order i.e. is no features have been provided to store the elements in sorted order.
- When you are creating any array of some specific size then it requires the contiguous memory location a disk space. If that memory location is not available then it may give some memory related problem i.e. heap. It will give the "java.lang.virtual machine Error".

- Suppose in the case of string array when we are trying to perform the reverse of that then first we need to split that &store that inside the string, array then we need to split that &store that inside the string, array&then we need to perform the accessing (by using the for loop) in which it will create numbers of unnecessary objects which is not recommendable.

In order to solve the problems with the array collection framework is been introduced. In this collection framework they have introduced five classes:

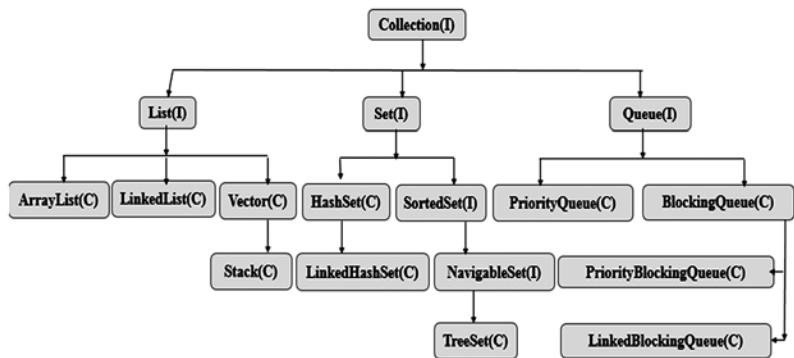
- 1) Dictionary
- 2) Properties
- 3) Hash table
- 4) Vector
- 5) Stack

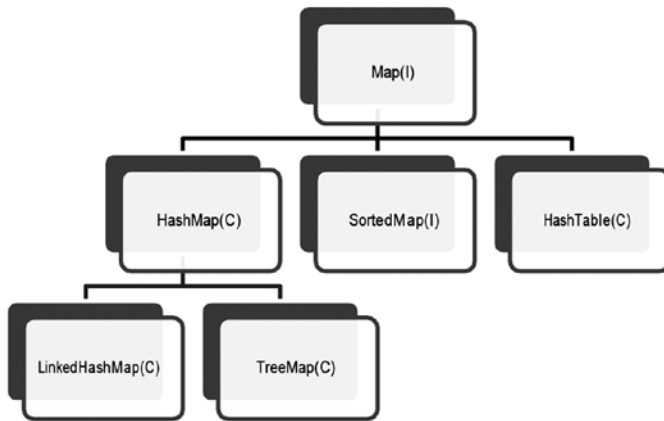
These five classes are further called as legacy classes. This five above classes almost solves all the problems with the array but still it was not able to solve the many major problem with the array.

- This five legacy classes can store the different types of element.
- There are not static in nature i.e. it's grow able in size. In order to solve the problem with the accessing SUN has introduced on Enumeration interface to access the elements from this legacy classes.
- All these legacy classes are completely synchronized in nature so the use of these legacy classes slow down the application speed.Still these legacy are not able to solve the problems with the array i.e.- Duplicity of the elements etc.

This collection framework is been reengineered in order to solve the problems with legacy classes out of legacy classes two legacy classes were re-engineering and added to the collection

- 1) HashTable
- 2) Vector





18.2 Collection

Collection is interface which is available inside java.util package .collection is further being extended by list set.

All the methods collections are available inside list& set both. This collection interface is further not being extended by map interface.

18.3 List Interface

List is a interface which is available inside java.util package and further this list is being implemented by following classes:

- 1) ArrayList
- 2) Linked List
- 3) vector

All this list classes stores the element with delicacy.

ArrayList

- It is a class which is available inside java.util package.
- It used to extends AbstractList class and implements List, RandomAccess, Cloneable & Serializable interface.
- ArrayList is implementing list interface.
- ArrayList stores the element in added order.
- ArrayList uses indexing representation to store the element.
- In ArrayList different types of element are allowed.
- ArrayList element can be accessed more than once by using ListIterator.
- None of the methods of ArrayList are synchronized. So the ArrayList class is also not synchronized. So the use of this doesn't affect the concurrency of the

application.

- ArrayList elements can be accessed by using the Iterator and ListIterator. By using enumeration we cannot access the elements of the ArrayList.
- ArrayList elements can be accessed only once in the forward by using the Iterator.

Example

```
import java.util.*;

class Hello{
void m1(){
System.out.println("m1 in Hello"); System.out.println(this.getClass().getName());
}

public String toString(){
return this.getClass().getName()+"@"+Integer.toHexString(hashCode());
}}

class Hai{
public String toString(){
return "xyz";
}}

public class Jtc187 {
public static void main(String[] args) {
ArrayList al1=new ArrayList();
System.out.println(al1);
System.out.println(al1.size());
al1.add(1234);
al1.add(new Integer(111));
al1.add("abc");
al1.add(new Hello());
al1.add(new Hai());
al1.add("abc");
al1.add(null);
System.out.println(al1);
System.out.println("Each ..... loop \n");
for(Object o:al1){
if(o instanceof Integer){
System.out.println(o.toString());
}else if(o instanceof Hello){
System.out.println(o);
}
```

```

Hello h1=(Hello)o;
h1.m1();
}else{
System.out.println(o);
}}
System.out.println("Iterator");
Iterator it=a1.iterator();
while(it.hasNext()){
Object o=it.next();
System.out.println(o);
if(o instanceof Hello){
Hello h11=(Hello)o;
}}
System.out.println("accessing again");
while(it.hasNext()){
System.out.println(it.next());
}
System.out.println("Using ListIterator"); ListIterator li=a1.listIterator();
while(li.hasNext()){
System.out.println(li.next());
}
System.out.println("*****Previous*****");
while(li.hasPrevious()){
System.out.println(li.previous());
}}}

```

addMethod()

It is used to add one element to the collection in case when you are trying to add a one collection to the another collection by using the add method then that all collection will be treated as an single unit element inside the another collection.

addAll()

This method is used to add collection by segregate their element as an single unit element.

Example

```

import java.util.*;
public class Jtc188 {

```

```

public static void main(String[] args) {
    ArrayList al1=new ArrayList();
    al1.add("abc");
    al1.add(new Integer(199));
    al1.add(new Double(111.11));
    System.out.println(al1);
    System.out.println("*****\n");
    ArrayList al2=new ArrayList();
    al2.add(9999); al2.add("Som");
    System.out.println(al2);
    al1.add(al2);
    System.out.println("*****\n");
    al1.addAll(al2);
    System.out.println(al1);
    System.out.println("*****\n");
    Iterator it1=al1.iterator();
    while(it1.hasNext()){
        System.out.println(it1.next());
    }
    System.out.println("*****\n");
    System.out.println(al2);
    ArrayList al3=new ArrayList();
    al3.add(9999);
    al3.add("som");
    al3.add("abc");
    al3.add(91819);
    System.out.println(al3);
    System.out.println("Reurn Method");
    System.out.println(al2);
    System.out.println(al3);
    //al2.returnAll(al3);
    //al3.returnAll(al3);
    System.out.println(al2);
    System.out.println(al3);
    al2.clear();
    al3.clear();
    System.out.println(al2);
    System.out.println(al3);
}

```

```
}}
```

LinkedList

- It is a class which extends AbstractSequentialList class and implements List, Deque, Cloneable & Serializable interface.
- LinkedList stores the element in added order.
- LinkedList can store the different types of element.
- LinkedList uses node representation to store the element.
- In the LinkedList we can store the null.
- LinkedList allows the duplicate elements.
- LinkedList element can be synchronized. So that LinkedList class is also not synchronized.
- LinkedList objects are eligible for serialization and cloning.
- LinkedList elements can be accessed by using the Iterator and ListIterator but by using Enumeration will not be able to access the LinkedList members.
- As the LinkedList is not synchronized so it doesn't affect the concurrency of an application.

Example

```
import java.util.*;
public class Hello01 {
    public static void main(String[] args) {
        LinkedList li = new LinkedList();
        System.out.println(li.size());
        System.out.println(li); li.add("first");
        li.add(123);// new Integer(123);
        li.add(new Hello());
        li.add(new Hello());
        li.add(27.333);
        li.add("last");
        System.out.println("*****HashCode*****");
        System.out.println(li.hashCode());
        System.out.println("\n");
        li.addFirst("Before First");
        li.addLast("After Last");
        System.out.println(li);
        System.out.println(li.hashCode());
        System.out.println("*****\n");
```

```

List lst = new LinkedList();
lst.add("abc"); lst.add(123);
System.out.println(lst.add("abc"));
System.out.println(lst);
// lst.addFirst("Before First");
// lst.addLast("After First");
boolean b1 = li.contains(123);
System.out.println(b1);
System.out.println(li);
LinkedList ll1 = new LinkedList();
ll1.add(123);
ll1.add("first");
ll1.add("last");
System.out.println(ll1);
System.out.println(li.contains(ll1));
System.out.println(ll1.contains(li));
li.addAll(ll1);
// li.add(ll1);
System.out.println(li);
System.out.println("using Iterator");
Iterator it = li.iterator();
while (it.hasNext()) {
    System.out.println(it.next());
}
while (it.hasNext()) {
    System.out.println(it.next());
}
System.out.println("\n*****List Iterator Forward*****");
ListIterator l = li.listIterator();
while (l.hasNext()) {
    System.out.println(l.next());
}
System.out.println("*****previous*****");
while (l.hasPrevious()) {
    System.out.println(l.previous());
// System.out.println(l.previous());
Object o = l.previous();
System.out.println(o);

```

```
}}}
```

Vector

- Vector is a legacy class
- Vector extends AbstractList and implements List, RandomAccess, Cloneable & Serializable interface.
- Vector has been reengineered and included in collection framework.
- Vector also used to store the different types of elements.
- Vector allows the duplicate elements.
- Vector uses indexing representation to store the element.
- Vector elements can access randomly.
- Null is allowed in vector.
- More than one null is allowed in Vector.
- It stores the elements in added order.
- Vector elements can be accessed, by using Iterator, ListIterator and Enumeration.
- When Vector was legacy class, Enumeration was there to access the elements. Same enumeration is being continued in collection also.
- By using Enumeration we can access the elements from the legacy classes only or the classes which have been reengineered and included in collection framework.
- When Vector was a legacy class then almost all the members of vector were synchronized. So it affects the concurrency of the application.
- Vector objects are eligible for serialization and cloning.

Example

```
import java.util.*;
public class Hello02 {
    public static void main(String[] args) {
        Vector v1=new Vector();
        ArrayList al=new ArrayList();
        System.out.println(al.size());
        System.out.println(al);
        al.add("aa");
        al.add("bb");
        al.add("cc");
        al.add("dd");
        al.add("ee");
```

```
System.out.println(al.size());
System.out.println(al);
System.out.println("*****\n");
v1.addAll(al);
System.out.println(v1);
v1.add("xyz");
v1.add(123);
v1.addElement("abc");
v1.addElement(9999);
System.out.println(v1);
System.out.println("*****\n");
Enumeration en=v1.elements();
while(en.hasMoreElements()){
System.out.println(en.nextElement());
}
ListIterator li=v1.listIterator();
while(li.hasNext()){
System.out.println(li.next());
}
System.out.println("*****\n");
while(li.hasPrevious()){
System.out.println(li.previous());
}}}
```

LIST	Duplicate Element	Different Or Same	Null	Index Or Node	Random Or Sequence	Synchronize	Enumeration	Iterator	ListIterator	Serialization & Cloning	Order
ArrayList	Yes	Differ	Yes	Index	Random	No	No	Yes	Yes	Yes	Added
LinkedList	Yes	Differ	Yes	Node	Sequential	No	No	Yes	Yes	Yes	Added
Vector	Yes	Differ	Yes	Index	Random	Yes	Yes	Yes	Yes	Yes	Added

Q1 : Can I synchronize the ArrayList & LinkedList?

Ans : Yes, we can synchronize explicitly by calling collections synchronize list (list).

18.4 SET

Set is an interface which is available inside java.util package and it is extending collection Interface. Further set is being implemented by hashSet and it been extended by sorted set. None of the Set classes stores the element with duplicacy.

Hash Set

- It is a class which is available inside java.util package.
- It used to extends AbstractSet class and implements Set, Cloneable & Serializable interface.
- Hash set stores the different type of element without any duplicacy i.e. Hash set is not going to store duplicate element.
- Hashset stores the element in unordered way.
- Hash set element can be accessed only in the forward direction by using the Iterator. We cannot access the element by using List Iterator.
- Hash set element can be accessed randomly.
- Null is allowed inside the hash set.
- Hash set member are eligible for serialization and cloning.
- Hash set none of the method is synchronized. So the hash set class is also not synchronized so it does not affect the concurrency of application.

Example

```
import java.util.*;
public class Jtc191 {
    public static void main(String[] args) {
        HashSet hs=new HashSet();
        hs.add("aa");
        hs.add("bb");
        hs.add("cc");
        hs.add("dd");
        System.out.println(hs);
        System.out.println(hs.size());
        System.out.println(hs.add("aa"));
        System.out.println(hs);
        hs.add(null);
        hs.add(null);
        System.out.println(hs);
    }
}
import java.util.*;
public class Jtc192 {
    public static void main(String[] args) {
        HashSet hs=new HashSet();
        hs.add("aa");
        hs.add("bb");
        hs.add("cc");
```

```

hs.add("dd");
System.out.println(hs);
Iterator it=hs.iterator();
while(it.hasNext()){
System.out.println(it.next());
}
System.out.println("*****\n");
List list=new ArrayList(hs);
ListIterator lt=list.listIterator();
while(lt.hasNext()){
System.out.println(lt.next());
}
System.out.println("*****Reverse*****");
while(lt.hasPrevious()){
System.out.println(lt.previous());
}
System.out.println("*****\n");
ArrayList al2=new ArrayList();
al2.add("aa");
al2.add("bb");
al2.add("cc");
System.out.println(al2);
}}

```

If you want to access the HashSet in reverse order first you have to convert to the type of list and by using that you can apply the listIterator.

LinkedHashSet

- LinkedHashSet used to extends HashSet class and implements Set, Cloneable & Serializable interface.
- LinkedHashSet also stores the element without duplicacy.
- LinkedHashSet elements will be stored in added order.
- LinkedashSet uses the node representation to store the element.
- Elements of linked HashSet can be accessed in sequential manner.

Linked HashSet none of the members are synchronized so the linked HashSet objects are also not synchronized and it does not affect the concurrency of the application.

Linked HashSet objects are eligible for serialization and cloning.

TreeSet

- TreeSet is a class which extends AbstractSet and implementing NavigableSet, Cloneable & Serializable interface.
- TreeSet stores the elements without duplicacy.
- TreeSet stores only one type of element.
- Inside TreeSet null is not allowed.
- TreeSet stores the elements in sorted order.
- None of members are synchronized so the TreeSet is also not synchronized.
- TreeSet objects are eligible for serialization and cloning.

Example

```
import java.util.*;
public class Jtc194 {
    public static void main(String[] args) {
        HashSet hs=new HashSet();
        hs.add(new Integer(99));
        System.out.println(hs.add("som"));
        hs.add("som@jtc");
        System.out.println(hs.add("som"));
        System.out.println(hs);
        TreeSet ts=new TreeSet();
        //ts.add(new Integer(99));
        System.out.println(ts.add("som"));
        ts.add("som@jtc");
        ts.add("som");
        ts.add("aaaa");
        ts.add("cccc");
        ts.add("bbbb");
        System.out.println(ts);
        LinkedHashSet lhs=new LinkedHashSet();
        lhs.add(new Integer(99));
        System.out.println(lhs.add("som"));
        lhs.add("som@jtc");
        System.out.println(lhs.add("som"));
        System.out.println(lhs);
    }
}
```

Question

1. Difference between ArrayList and Array?
2. Difference between ArrayList and LinkedList?
3. Difference between ArrayList and Vector?
4. Difference between ArrayList and HashSet?
5. Difference between List and Set?
6. Difference between Collection and Collections?
7. List out all the contract between hashCode() and equals().

Example using HashSet,TreeSet,LinkedHashSet

```
import java.util.*;

public class Jtc4 {

    public static void main(String as[])
    { HashSet hs=new HashSet();
      hs.add(new Integer(99));
      System.out.println(hs.add("som"));
      hs.add("som@jtc");
      System.out.println(hs.add("som"));
      System.out.println(hs);
      TreeSet ts=new TreeSet();
      //ts.add(new Integer(99));
      System.out.println(ts.add("som"));
      ts.add("som@jtc");
      System.out.println(ts.add("som"));
      ts.add("aaaa");
      ts.add("cccc");
      ts.add("bbbb");
      System.out.println(ts);
      LinkedHashSet lhs=new
      LinkedHashSet(); lhs.add(new
      Integer(99));
      System.out.println(lhs.add("som"));
      lhs.add("som@jtc");
```

```

System.out.println(lhs.add("som"));
System.out.println(lhs);
}
}

```

Example

```

import java.util.*;
public class Jtc6 {

    public static void main(String as[])
    {
        LinkedHashMap hm=new LinkedHashMap();
        hm.put("sid",new Integer(99));
        hm.put("sname", "Som");
        hm.put("fee",new Double(9000.99));
        System.out.println(hm);
        hm.put("x","10");
        hm.put(new Integer(99),"10");
        System.out.println(hm);
        hm.put("x","20");
        System.out.println(hm);
        hm.put(null,null);
        System.out.println(hm);
        Hashtable ht=new Hashtable();
        ht.put("sid",new Integer(99));
        ht.put("sname", "Som");
        ht.put("fee",new Double(9000.99));
        System.out.println(ht);
        ht.put("x","10");
        ht.put(new Integer(99),"10");
        System.out.println(ht);
        ht.put("x","20");
        System.out.println(ht);
    }
}

```

```

/*
ht.put("z",null);
System.out.println(ht);
*/
/*
ht.put(null,"s");
System.out.println(ht);
*/
TreeMap tm=new TreeMap();
tm.put("sid",new Integer(99));
tm.put("sname","Som");
tm.put("fee",new Double(9000.99));
System.out.println(tm);
tm.put("x","10");
//tm.put(new Integer(99),"10");
System.out.println(tm);
tm.put("x","20");
System.out.println(tm);
tm.put("z",null);
System.out.println(tm);
/*
tm.put(null,"s");
System.out.println(tm);
*/
}
}

```

Example

```

import java.util.*;
public class Jtc5 {

public static void main(String as[]) {
HashMap hm=new HashMap();

```

```

System.out.println(hm);
System.out.println(hm.size());
System.out.println(hm.isEmpty());
hm.put("sid",new Integer(99));
hm.put("sname","som");
hm.put("email","abc");
System.out.println(hm);
System.out.println(hm.size());
System.out.println(hm.isEmpty());
System.out.println(hm.containsKey("sid"));
System.out.println(hm.containsKey("sid1"));
System.out.println(hm.containsValue("som"));
System.out.println(hm.containsValue("som1"));
System.out.println(hm.get("sname"));
System.out.println(hm);
hm.put("sname","rai");
System.out.println(hm);
hm.put("sname1","rai");
System.out.println(hm);
hm.put(null,"rai");
hm.put(new Double(999.99),null);
System.out.println(hm);
//hm.remove("xx");
System.out.println(hm);
Collection col=hm.values();
System.out.println(col);
System.out.println("using keySet()");
Set s=hm.keySet();
System.out.println(s);
Iterator it=s.iterator();
while(it.hasNext()){
Object o1=it.next();
String key="";
if(o1!=null){
key=o1.toString();
}
else{
key=null;

```

```

}
Object o2=hm.get(key);
String val="";
if(o2!=null){
val=o2.toString();
}
else{
val=null;
}
System.out.println(key+"..." +val);
}
System.out.println("using entrySet()");
Set es=hm.entrySet();
Iterator it1=es.iterator();
while(it1.hasNext()){
Object o=it1.next();
Map.Entry me=(Map.Entry)o;
Object o1=me.getKey();
String key="";
if(o1!=null){
key=o1.toString();
}
else{
key=null;
}
Object o2=me.getValue();
String val="";
if(o2!=null){
val=o2.toString();
}
else{
val=null;
}
System.out.println(key+"..." +val);
}
}
}
}

```


18.5 Map

Map is a interface which is not extending collection Interface. Map is being further implemented by HashMap and Hashtable and is been extended by SortedMap. All the map classes stores the element on the basis of key and value pair. In the case of Map, its classes stores the value on the basis of key and value pair and it will be talking about the duplicacy of the values on the basis of key or value.

HashMap

- HashMap is the class which is available inside java.util package and also it stores value on the basis of key and value pair.
- HashMap used to extends AbstractMap class and impements Map, Cloneable & Serializable interface.
- HashMap stores the different types of element.
- In the case of HashMap we can't duplicate the key part or we cannot have the duplicate key.
- Different types of keys are allowed.
- Null is allowed as on the key.
- We can have the value of different types or same type.
- All the values can be of same type or different types i.e. value can be duplicated.
- HashMap uses the indexing representation to store the element i.e. element of HashMap can be accessed randomly.
- HashMap stores the element in unordered way on the basis of key.
- HashMap objects are eligible for serialization and cloning.

value() : It is available inside HashMap which returns to the type of collection and it gives all the values associated with that map. Even we can access the values of HashMap by using the entrySet().

Map.Entry: It is an interface which getKey() and getValue() method is available and on the basis of getKey() and getValue() we can get the elements correspondingly by using the Map.Entry reference.

LinkekedHashMap

1. It is a class which is available inside java.util package.
2. LinkedHashMap is extending the HashMap & implements Map interface.
3. LinkedHashMap uses node representation to store the elements.
4. It also stores value on the basis of key and value format.
5. In case of LinkedHashSet we can store the different types of key and value.

6. Key cannot be duplicated but value can be duplicated.
7. Null is allowed as an key and value.
8. LinkedHashMap element can be accessed in a sequential manner.
9. LinkedHashMap none of the members are synchronized. So this is also not synchronized.
10. It is eligible for serialization and cloning.
11. It stores the elements in added order.

Hashtable

1. It is a legacy class which has been re-engineered and included in collection framework.
2. It used to extends Dictionary class and implements Map, Serializable & Cloneable interface.
3. Hashtable also stores the element in key and value format.
4. Hashtable stores the element in unordered way.
5. It allows the different types of element for key and value.
6. Null is not allowed in an key and value format.
7. Hashtable almost all the members are synchronized. So it affect the concurrency of Application

TreeMap

- Stores the element in sorted order on the basis of key.
- Same type of key is allowed where as values can be of different types.
- Null is not allowed as an key and value both.
- TreeMap members are not synchronized so it cannot affect the concurrency of an application.

Collection	Duplicate Element	Different Or Same	Null	Index Or Node	Random Or Sequence	Synchronize	Enumeration	ListIterator & Iterator	Serialization & Cloning	Order
ArrayList	Yes	Differ	Yes	Index	Random	No	No	Both	Yes	Added
LinkedList	Yes	Differ	Yes	Node	Sequential	No	No	Both	Yes	Added
Vector	Yes	Differ	Yes	Index	Random	Yes	Yes	Both	Yes	Added
HashSet	No	Differ	Yes	Index	Random	No	No	Iterator	Yes	Unordered
LinkedHashSet	No	Differ	Yes	Node	Sequential	No	No	Iterator	Yes	Added
TreeSet	No	Same	No	Index	Random	No	No	Iterator	Yes	Sorted order

Map	Duplicate		Differ Or Same		Null		Index Or Node	Random Or Sequence	Synchronize	List Iterator & Iterator	Serialization & Cloning	Order
	Key	Val	Key	Val	Key	Val						
HashMap	No	Yes	Diff	Diff	Yes	Yes	Index	Random	No	No	Yes	unordered
Linked HashMap	No	Yes	Diff	Diff	Yes	Yes	Node	Sequential	No	No	Yes	Added
TreeMap	No	Yes	Same	Diff	No	No	Index	Random	No	No	Yes	Sorted
HashTable	No	Yes	Diff	Diff	No	No	Index	Random	Yes	No	Yes	unordered

18.6 Difference between comparable and Comparator.

When we want to sort objects in many different ways then we need to use the comparable and comparator.

Comparable	Comparator
- Comparable is an Interface which is available inside java.lang package. - If you wanted to sort any class object in one specific way then implement comparable interface provide the only one way of comparison. - When a class is implementing comparable interface then you need to override compareTo() inside the class. - Calls Collections.Sort of comparable type object i.e. Collection.Sort(Comparable).	- Comparator is also an Interface which is available inside java.util package. - Comparator interface provides many different way of comparison. - If a class is implementing comparator interface then that class has to override compare() inside the class. - Calls Collections.Sort (Collection_Type, Comparator). - Comparator Interface will be implemented by third party class without touching the main class whose object has to be sorted.

Example

```
import java.util.*;
class Student implements Comparable{
int sid;
String sname;
String email;
Student(int sid,String sname,String email){
this.sid=sid;
this.sname=sname;
```

```

this.email=email;
}
public String toString(){
return ""+sid+"\t"+sname+"\t"+email;
}
public boolean equals(Object o){
Student s=(Student)o;
if(this.sid==s.sid)
return true;
return false;
}
public int compareTo(Object o){
Student s=(Student)o;
return this.sid-s.sid;
}
}
class SnameComparator implements
Comparator{ public int compare(Object
o1,Object o2){ Student s1=(Student)o1;
Student s2=(Student)o2;
return s1.sname.compareTo(s2.sname);
}
}
class EmailComparator implements Comparator{
public int compare(Object o1,Object o2){ Student
s1=(Student)o1;
Student s2=(Student)o2;
return s1.email.compareTo(s2.email);
}
}
class Jtc7{

public static void main(String as[]){

```

```

ArrayList al=new ArrayList();
Student s1=new Student(22,"dd","cc@jtc");
Student s2=new Student(44,"aa","bb@jtc");
Student s3=new Student(11,"cc","dd@jtc");
Student s4=new Student(33,"bb","aa@jtc");
al.add(s1);  al.add(s2);  al.add(s3);  al.add(s4);
System.out.println("No Sorting");
Iterator it=al.iterator();
while(it.hasNext()){
Student s=(Student)it.next();
System.out.println(s);
}
System.out.println("Sorting by Sid");
Collections.sort(al);
it=al.iterator();
while(it.hasNext()){
Student s=(Student)it.next();
System.out.println(s);
}
System.out.println("Sorting by Sname");
Collections.sort(al,new SnameComparator());
it=al.iterator();
while(it.hasNext()){
Student s=(Student)it.next();
System.out.println(s);
}
System.out.println("Sorting by Email");
Collections.sort(al,new EmailComparator());
it=al.iterator();
while(it.hasNext()){
Student s=(Student)it.next();
System.out.println(s);
}
}
}
}

```

18.7 String Tokenizer

String Tokenizer is a class which is available inside java.util package.

when you are working with String if you wanted to reverse that String then you need to use the split method of split class and after specifying the criteria of string, you need to store that inside the string array. unnecessarily which will create string object which may cause heap memory related problem of java.lang.OutOfMemory error. After that when we are accessing the String array, initialize the counter and then increment or decrement the counter accordingly. But by using the String tokenizer we can tokenize the String in the form of tokens which will not create any unnecessary object and in enumeration manner easily we can access the tokens, which will not take any extra effort.

18.8 hasMoreTokens()

It checks whether any token is available or not, if it is available then it returns true or else false. While tokenizing the String if you are not specifying any condition then it will automatically tokenize on the basis of space.

Example

```
import java.util.*;
public class Jtc197 {
    public static void main(String[] args) {
        String str="welcome to Jtc";
        StringTokenizer token=new StringTokenizer(str);
        System.out.println(token.hasMoreTokens());
        System.out.println(token.countTokens());
        while(token.hasMoreTokens()){
            String str1=token.nextToken();
            System.out.println(str1);
        }
        System.out.println(token.hasMoreTokens());
        System.out.println(token.countTokens());
        StringTokenizer token1=new StringTokenizer(str,"ja"); System.out.println(token1.
        hasMoreTokens());
        while(token1.hasMoreElements());
        Object obj=token1.nextElement();
        System.out.println(obj);
    }
}
```

Example

Implementing of Constructing Custom ArrayList:

```

import java.util.*;
class MyArrayList {
private Object element[] = null;
private int Capacity;
private int Size;
public boolean add(int index, Object obj) {
if (index > Size || index < 0) {
throw new IndexOutOfBoundsException("Index:" + index + "Size:" + Size);
}
if (Size == Capacity) {
int newcap = Capacity * 2;
Object newelement[] = new Object[newcap];
System.arraycopy(element, 0, newelement, 0, Size);
element = newelement;
Capacity = newcap;
}
if (index == Size) {
element[index] = obj;
} else {
Object arr[] = new Object[Capacity];
if (index == 0) {
arr[index] = obj;
System.arraycopy(element, 0, arr, 1, Size);
} else {
System.arraycopy(element, 0, arr, 0, index); arr[index] = obj;
System.arraycopy(element, index, arr, index + 1, Size - index);
}
element = arr;
}
Size++;
return true;
}
public Object remove(int index) {
if (index > 0 && index < Size) {
Object obj = element[index];
int lastIndex = Size - (index + 1);
System.arraycopy(element, index, element[Size], index, lastIndex);
element[Size] = null;
}
}

```

```

Size--;
return obj;
} else {
throw new IndexOutOfBoundsException("Index:" + index + "Size" + Size);
}}
public Object get(int index) {
if (index >= 0 && index < Size)
return element[index];
else
throw new IndexOutOfBoundsException("Index:" + index + "size:" + Size);
}
public boolean remove(Object obj){
int index = indexOf(obj);
if (index > 0) {
remove(index);
return true;
}
return false;
}
public void clear() {
for (int i = 0; i < Size; i++) { element[i] = null;
}
Size = 0;
}
public boolean Contains(Object obj) {
int x = indexOf(obj);
if (x != -1) {
return true;
} else {
return false;
}}
public int indexOf(Object obj) {
for (int i = 0; i < Size; i++) { Object el = element[i];
if (el == null && obj == null) {
return i;
} else if (el != null && el.equals(obj)) {
return i;
}}
}

```



```

return -1;
}
public boolean add(Object obj) {
return add(Size, obj);
}
public MyArrayList(int initialCapacity) { element = new Object[initialCapacity];
Capacity = initialCapacity;
}
public int Capacity() {
return Capacity;
}
public int Size() {
return Size();
}
public MyArrayList() {
this(10);
}
public String toString() {
StringBuffer sb = new StringBuffer("[");
for (int i = 0; i < Size; i++) {
if (i == Size - 1) {
sb.append(element[i] + "");
} else {
sb.append(element[i] + ",");
}
}
sb.append("]");
return sb.toString();
}
class Itr implements Iterator{
private int CurrentPointer = -1;
public boolean hasNext() {
if (Size != 0 && CurrentPointer < Size - 1)
return true;
else
return false;
}
public Object next(){
if (CurrentPointer < Size - 1) {

```

```
CurrentPointer++;
return element[CurrentPointer];
} else {
throw new NoSuchElementException();
}
}

public void remove() {
if (CurrentPointer <= 0 && CurrentPointer < Size) {
MyArrayList.this.remove(CurrentPointer);
} else {
throw new IllegalStateException();
}}

public Iterator iterator() {
return new Itr();
}}

public class Jtc198 {
public static void main(String[] args) {
```

18.9 Difference between Enumeration and Iterator

Enumeration	Iterator
• Enumeration is an Interface which is available inside java.util package also called as legacy Inteface.	• It is an Interface which is also available inside java.util package and it is mainly introduced from collection frame work.
• By using enumeration we can access elements only from legacy classes.	• By using Iterator we can access the elements from legacy as well as collection classes also.
• By using enumeration we can only access the element in the forward direction order.	• By using Iterator we can access the elements in forward ordering i.e. only once.
• By using enumeration we cannot remove any element from the collection.	• By using Iterator we can remove the element from the collection.

18.10 Difference between Iterator and List Iterator

Iterator	ListIterator
• Iterator is an Interface which is available inside java.util package. • By using Iterator we can only access the elements with the forward direction. • By using Iterator we can only remove the elements from the collection. • By using Iterator we can access the element of List and Set both.	• ListIterator is also an Interface which is available inside java.util package and extending Iterator Interface. • By using ListIterator we can access the elements in the forward and reverse direction both. • By using ListIterator we can remove as well as you can add the element to the collection. • By using ListIterator we can access • only the element from the List classes.

18.11 addElements()

By using addElements() we can add elements to the legacy classes which has been re-engineered and included in collection framework. Because addElements() is used to add the elements to the legacy classes. There is no difference in order of processing between addElements() and add().

18.12 Timer and TimerTask

- TimerTask is an abstract class which is implementing Runnable Interface.
- TimerTask class is available inside java.util package.
- TimerTask class is not having its own abstract method.it is just having the abstract method of Runnable Interface which is not be overridden inside the TimerTask class.that has been left for the developer to write the implementation of run() accordingly.
- Timer class:-it is a class which is available inside java.util package.
- Timer class is having many methods to schedule the task accordingly.
- These both cases will be used to schedule the task accordingly.

18.13 Methods

addElements(): By using addElement() we can add element to the legacy classes which has been re-engineered and included in collection framework.Because

addElement is used to add the element to legacy classes.

There is no difference in order of processing between addElement() & add ().

Methods

- public void Schedule(TimerTask task, Date time);
- public void Schedule(TimerTask task, Date firstTime, long period);
- public void Schedule(TimerTask task, long delay);
- public void Schedule(TimerTask task, long delay, long period);
- public void ScheduleAtFixedRate(TimerTask task, long delay, long period);
- public void ScheduleAtFixedRate(TimerTask task, Date firstTime, long period);

Internal implements of schedule method

Example

```
import java.util.*;
class Reminder {
    Timer timer;
    public Reminder(int seconds) {
        timer = new Timer();
        timer.schedule(new RemindTask(), seconds *
            1000); }
    class RemindTask extends TimerTask {
        public void run() {
            System.out.println("Time's up!");
            timer.cancel();
        }
    }
}
public class Jtc13 {
    public static void main(String args[])
    { System.out.println("About to schedule task.");
        new Reminder(5);
```

```
System.out.println("Task scheduled.");
}
}
```

If the timer task execution thread terminates unexpectedly, for example because its stop method is invoked any further attempt to an `IllegalStateException` as if the timers cancel method had been invoked.

18.13.1 Stack

- It is the child class of `Vector`.
- Whenever last in first out(LIFO) order required then we should go for Stack.
`Stack s= new Stack();`

Method in Stack

Object push(Object o);

To insert an object into the stack.

Object pop();

To remove and return top of the stack.

Object peek();

To return top of the stack without removal.

boolean empty();

Returns true if Stack is empty.

Int search(Object o);

Returns offset if the element is available otherwise returns "-1"

18.13.2 Null Acceptance

For the empty `TreeSet` as the 1st element "null" insertion is possible but after inserting that null if we are trying to insert any other we will get `NullPointerException`.

For the non empty `TreeSet` if we are trying to insert null then we will get `NullPointerException`.

Example

```
import java.util.*;
class TreeSetDemo{
public static void main(String[] args){
```

```

TreeSet t=new TreeSet();
t.add(new StringBuffer("A"));
t.add(new StringBuffer("Z"));
t.add(new StringBuffer("L"));
t.add(new StringBuffer("B"));
System.out.println(t);
}

```

Note

- Exception in thread "main" java.lang.ClassCastException: java.lang.StringBuffer cannot be cast to java.lang.Comparable
- If we are depending on default natural sorting order compulsory the objects should be homogeneous and Comparable otherwise we will get ClassCastException.
- An object is said to be Comparable if and only if the corresponding class implements Comparable interface.
- String class and all wrapper classes implements Comparable interface but StringBuffer class doesn't implement Comparable interface hence in the above program we are getting ClassCastException.

18.14 Entry Interface

Each key-value pair is called one entry. Hence Map is considered as a group of entry Objects, without existing Map object there is no chance of existing entry object hence interface entry is define inside Map interface (inner interface).

18.15 IdentityHashMap

- It is exactly same as HashMap except the following differences:
- In the case of HashMap JVM will always use ".equals()" method to identify duplicate keys, which is meant for content comparison.
- But in the case of IdentityHashMap JVM will use == (double equal operator) to identify duplicate keys, which is meant for reference comparison.

Example

```

import java.util.*;
class HashMapDemo{
public static void main(String[] args){
HashMap m=new HashMap();
Integer i1=new Integer(10);

```

```
Integer i2=new Integer(10);
m.put(i1,"som");
m.put(i2,"prakash");
System.out.println(m);
}}
```

In the above program i1 and i2 are duplicate keys because i1.equals(i2) returns true.

In the above program if we replace HashMap with IdentityHashMap then i1 and i2 are not duplicate keys because i1==i2 is false hence in this case the output is {10=som, 10=prakash}.

18.16 WeakHashMap

- It is exactly same as HashMap except the following differences:
- In the case of normal HashMap, an object is not eligible for GC even though it doesn't have any references if it is associated with HashMap. That is HashMap dominates garbage collector.
- But in the case of WeakHashMap if an object does not have any references then it's always eligible for GC even though it is associated with WeakHashMap that is garbage collector dominates WeakHashMap.

Example

```
import java.util.*;
class WeakHashMapDemo{
public static void main(String[] args)throws Exception {
WeakHashMap m=new WeakHashMap();
Temp t=new Temp();
m.put(t,"som");
System.out.println(m);
t=null;
System.gc();
Thread.sleep(5000);
System.out.println(m);//{}
}}
class Temp{
public String toString(){
return "Temp";
}
```

```

}
public void finalize(){
System.out.println("finalize() method called");
}}

```

In the above program if we replace WeakHashMap with normal HashMap then object won't be destroyed by the garbage collector in this the output is

18.17 SortedMap

- It is the child interface of Map.
- If we want to represent a group of key-value pairs according to some sorting order of keys then we should go for SortedMap.
- Sorting is possible only based on the keys but not based on values. SortedMap interface defines the following 6 specific methods.
 - o Object firstKey();
 - o Object lastKey();
 - o SortedMap headMap(Object key);
 - o SortedMap tailMap(Object key);
 - o SortedMap subMap(Object key1, Object key2);
 - o Comparator comparator();

18.18 Properties

Properties class is the child class of Hashtable.

- In our program if anything which changes frequently like DBUserName, Password etc., such type of values not recommended to hardcode in java application because for every change we have to recompile, rebuild and redeployed the application and even server restart also required sometimes it creates a big business impact to the client.
- Such type of variable things we have to hardcode in property files and we have to read the values from the property files into java application.
- The main advantage in this approach is if there is any change in property files automatically those changes will be available to java application just redeployment is enough.
- By using Properties object we can read and hold properties from property files into java application.

Properties p=new Properties();

In properties both key and value "should be String type only".

Methods in Properties class

String getProperty(String propertyname) ;

Returns the value associated with specified property.

String setProperty(String propertyname,String propertyvalue);

To set a new property.

Enumeration propertyNames();

void load(InputStream is);

//Any InputStream we can pass.To load Properties from property files into java Properties object.

void store(OutputStream os,String comment);

//Any OutputStream we can pass. To store the properties from Properties object into properties file.

Example

```
import java.util.*;
import java.io.*;
class PropertiesDemo{
public static void main(String[] args)throws Exception{
Properties p=new Properties();
FileInputStream fis=new FileInputStream("abc.properties");
p.load(fis);
System.out.println(p);
String s=p.getProperty("som");
System.out.println(s);//8888
p.setProperty("abh","9999999");
Enumeration e=p.propertyNames();
while(e.hasMoreElements()){
String s1=(String)e.nextElement();
System.out.println(s1);
}
FileOutputStream fos=new FileOutputStream("abc.properties");
p.store(fos,"updated by som");
}}
```

Example

```
import java.util.*;
import java.io.*;
class PropertiesDemo{
```

```

public static void main(String[] args) throws Exception{
    Properties p=new Properties();
    FileInputStream fis=new FileInputStream("db.properties");
    p.load(fis);
    String url=p.getProperty("url");
    String user=p.getProperty("user");
    String pwd=p.getProperty("pwd");
    Connection con=DriverManager.getConnection(url, user, pwd);
    FileOutputStream fos=new FileOutputStream("db.properties");
    p.store(fos,"updated by som");
}
}

```

Example

```

import java.util.*;
import java.io.*;
class PropertiesDemo{
    public static void main(String[] args) throws Exception {
        Properties p=new Properties();
        FileInputStream fis=new FileInputStream("abc.properties");
        p.load(fis);
        System.out.println(p);
        String s=p.getProperty("som");
        System.out.println(s);//8888
        p.setProperty("som","9999999");
        Enumeration e=p.propertyNames();
        while(e.hasMoreElements()){
            String s1=(String)e.nextElement();
            System.out.println(s1);
        }
        FileOutputStream fos=new FileOutputStream("abc.properties");
        p.store(fos,"updated by som");
    }
}

```

18.19 Queue

- Queue is child interface of Collection.
- If we want to represent a group of individual objects prior (happening before something else) to processing then we should go for Queue interface.
- Usually Queue follows first in first out(FIFO) order but based on our

requirement we can implement our own order also.

- From 1.5v onwards LinkedList also implements Queue interface.
- LinkedList based implementation of Queue always follows first in first out order.

Assume we have to send sms for one lakh mobile numbers, before sending messages we have to store all mobile numbers into Queue so that for the first inserted number first message will be triggered(FIFO).

Queue interface methods

boolean offer(Object o);

To add an object to the Queue.

Object poll() ;

To remove and return head element of the Queue, if Queue is empty then we will get null.

Object remove();

To remove and return head element of the Queue. If Queue is empty then this method raises Runtime Exception saying NoSuchElementException.

Object peek();

To return head element of the Queue without removal, if Queue is empty this method returns null.

Object element();

It returns head element of the Queue and if Queue is empty then it will raise Runtime Exception saying NoSuchElementException.

18.20 PriorityQueue

- PriorityQueue is a data structure to represent a group of individual objects prior to processing according to some priority.
- The priority order can be either default natural sorting order (or) customized sorting order specified by Comparator object.
- If we are depending on default natural sorting order then the objects must be homogeneous and Comparable otherwise we will get ClassCastException.
- If we are defining our own customized sorting order by Comparator then the objects need not be homogeneous and Comparable.
- Duplicate objects are not allowed.
- Insertion order is not preserved but all objects will be inserted according to some priority.
- Null is not allowed even as the 1st element for empty PriorityQueue. Otherwise we will get the "NullPointerException".

```
PriorityQueue q=new PriorityQueue();
```

Creates an empty PriorityQueue with default initial capacity 11 and default natural sorting order.

```
PriorityQueue q=new PriorityQueue(int initialcapacity,Comparator c);
```

```
PriorityQueue q=new PriorityQueue(int initialcapacity);
```

```
PriorityQueue q=new PriorityQueue(Collection c);
```

```
PriorityQueue q=new PriorityQueue(SortedSet s);
```

Example

```
import java.util.Comparator;
```

```
import java.util.PriorityQueue;
```

```
import java.util.Scanner;
```

```
public class Jtc18 {
```

```
public static void main(String[] args) {
```

```
Scanner sc = new Scanner(System.in);
```

```
System.out.println("Enter values of int type[0 - Exit]");
```

```
PriorityQueue<Integer> pq = new PriorityQueue<Integer>();
```

```
while (true) {
```

```
int val = sc.nextInt();
```

```
if (val == 0)
```

```
break;
```

```
pq.offer(val);
```

```
}
```

```
System.out.printf("Number of values stored is %d.\n",
```

```
pq.size()); for (Integer in : pq) {
```

```
System.out.println(in);
```

```
}
```

```
PQSort pqs = new PQSort();
```

```
PriorityQueue<Integer> pq1 = new PriorityQueue<Integer>(8,
```

```
pqs); System.out.println("Enter values of int type[0 - Exit]");
```

```
while (true) {
```

```
int val = sc.nextInt();
```

```

if (val == 0)
break;
pq1.offer(val);
}
System.out.println("--");
System.out.println("Size\t:" + pq1.size());
System.out.println(pq1.peek());
System.out.println("Size\t:" + pq1.size());
System.out.println(pq1.poll());
System.out.println("Size\t:" + pq1.size());
System.out.println("Remaining Values in PQ1");
for (Integer in : pq1) {
System.out.println(in);
}}}
class PQSort implements Comparator<Integer> {
public int compare(Integer i1, Integer i2) {
return i1 - i2;
}
}

```

Note: Some platforms may not provide proper supports for PriorityQueue [windowsXP].

18.21 NavigableSet

- It is the child interface of SortedSet.
- It provides several methods for navigation purposes.

NavigableSet interface defines the following methods.

ceiling(e);

It returns the lowest element which is $\geq e$.

higher(e);

It returns the lowest element which is $> e$.

floor(e);

It returns highest element which is $\leq e$.

lower(e);

It returns height element which is $< e$.

pollFirst ();

Remove and return 1st element.

pollLast ();

Remove and return last element.

descendingSet ();

Returns SortedSet in reverse order

Example

```
import java.util.*;
class NavigableSetDemo{
public static void main(String[] args){
TreeSet<Integer> t=new TreeSet<Integer>();
t.add(1000);
t.add(2000);
t.add(3000);
t.add(4000);
t.add(5000);
System.out.println(t);//[1000, 2000, 3000, 4000,5000]
System.out.println(t.ceiling(2000));//2000
System.out.println(t.higher(2000));//3000
System.out.println(t.floor(3000));//3000
System.out.println(t.lower(3000));//2000
System.out.println(t.pollFirst());//1000
System.out.println(t.pollLast());//5000
System.out.println(t.descendingSet());//[4000, 3000, 2000]
System.out.println(t);//[2000, 3000, 4000] }
}
```

18.22 NavigableMap

It is the child interface of SortedMap and it defines several methods for navigation purpose.

NavigableMap interface defines the following methods.

```
ceilingKey(e);
higherKey(e);
floorKey(e);
lowerKey(e);
pollFirstEntry();
pollLastEntry();
descendingMap();
```

Example

```

import java.util.*;
class NavigableMapDemo{
public static void main(String[] args){
TreeMap<String,String> t=new TreeMap<String,String>();
    t.put("b","banana");
    t.put("b","banana");
    t.put("c","cat");
    t.put("a","apple");
    t.put("d","dog");
    t.put("g","gun");
System.out.println(t);
System.out.println(t.ceilingKey("c"));
System.out.println(t.higherKey("e"));
System.out.println(t.floorKey("e"));
System.out.println(t.lowerKey("e"));
System.out.println(t.pollFirstEntry());
System.out.println(t.pollLastEntry());
System.out.println(t.descendingMap());
System.out.println(t);
}}

```

Collections

- Collections class defines several utility methods for collection objects.
- Sorting the elements of a List:
- Collections class defines the following methods to perform sorting the elements of a List.

public static void sort(List l);

To sort the elements of List according to default natural sorting order in this case the elements should be homogeneous and comparable otherwise we will get `ClassCastException`.

The List should not contain null otherwise we will get `NullPointerException`.

public static void sort(List l,Comparator c);

To sort the elements of List according to customized sorting order.

Example

```

import java.util.*;

```

```

class Student implements Comparable{
    int sid;
    String sname;
    String email;
    Student(int sid,String sname,String email){
        this.sid=sid;
        this.sname=sname;
        this.email=email;
    }
    public String toString(){
        return ""+sid+"\t"+sname+"\t"+email;
    }
    public boolean equals(Object o){
        Student s=(Student)o;
        if(this.sid==s.sid)
            return true;
        return false;
    }
    public int compareTo(Object o){
        Student s=(Student)o;
        return this.sid-s.sid;
    }
}

class SnameComparator implements Comparator{
    public int compare(Object o1,Object o2){
        Student s1=(Student)o1;
        Student s2=(Student)o2;
        return s1.sname.compareTo(s2.sname);
    }
}

class EmailComparator implements Comparator{
    public int compare(Object o1,Object o2){
        Student s1=(Student)o1;
        Student s2=(Student)o2;
        return s1.email.compareTo(s2.email);
    }
}

```


Example

```

class Jtc7{

public static void main(String as[]){
ArrayList al=new ArrayList();
Student s1=new Student(22,"dd","cc@jtc");
Student s2=new Student(44,"aa","bb@jtc");
Student s3=new Student(11,"cc","dd@jtc");
Student s4=new Student(33,"bb","aa@jtc");
al.add(s1);   al.add(s2);   al.add(s3);
System.out.println("No Sorting");
Iterator it=al.iterator();
while(it.hasNext()){
Student s=(Student)it.next();
System.out.println(s);
}  al.add(s4);
System.out.println("Sorting by Sid");
Collections.sort(al);
it=al.iterator();
while(it.hasNext()){
Student s=(Student)it.next();
System.out.println(s);
}
System.out.println("Sorting by Sname");
Collections.sort(al,new SnameComparator());
it=al.iterator();
while(it.hasNext()){
Student s=(Student)it.next();
System.out.println(s);
}
System.out.println("Sorting by Email");
}
}

```

```

Collections.sort(al,new EmailComparator());
it=al.iterator();
while(it.hasNext()){
Student s=(Student)it.next();
System.out.println(s);
}}}

```

Searching the elements of a List

Collections class defines the following methods to search the elements of a List.

public static int binarySearch(List l,Object obj);

If the List is sorted according to default natural sorting order then we have to use this method.

public static int binarySearch(List l,Object obj,Comparator c);

If the List is sorted according to Comparator then we have to use this method.

Program 18.1

```

import java.util.*;
public class Jtc8 {

    public static void main(String[] args) {
        ArrayList al=new ArrayList();
        al.add("bb");
        al.add("cc");
        al.add("dd");
        al.add("aa");
        //al.add(new Integer(99));
        System.out.println(al);
        Collections.sort(al);
        System.out.println(al);
        Collections.reverse(al);
        System.out.println(al);
    }
}

```

```

Collections.shuffle(al);
System.out.println(al);
Collections.rotate(al,1);
System.out.println(al);
Collections.swap(al,1,3);
System.out.println(al);
System.out.println(Collections.max(al));
System.out.println(Collections.min(al));
Collections.sort(al);
System.out.println(Collections.binarySearch(al,"cc"));
Collections.fill(al,"jtc");
System.out.println(al);
Vector v=new Vector();
v.add("99");
v.add("som");
v.addElement("abc");
System.out.println(v);
Enumeration e=v.elements();
List al1=Collections.list(e);
System.out.println(v);
System.out.println(al1);
al1=Collections.unmodifiableList(al1);
al1.add("11");
}}

```

- Internally these search methods will use binary search algorithm.
- Successful search returns index unsuccessful search returns insertion point.
- Insertion point is the location where we can place the element in the sorted list.
- Before calling `binarySearch()` method compulsory the list should be sorted otherwise we will get unpredictable results.
- If the list is sorted according to `Comparator` then at the time of search operation also we should pass the same `Comparator` object otherwise we will get unpredictable results.

Note: For the list of n elements with respect to `binary Search()` method.

Successful search range is: 0 to $n-1$.

Unsuccessful search results range is: $-(n+1)$ to -1 .

Total result range is: $-(n+1)$ to $n-1$.

Reversing the elements of List

public static void reverse(List l);

reverse() vs reverseOrder() method

- We can use reverse() method to reverse the elements of List.
- Where as we can use reverseOrder() method to get reversed Comparator.

Program 18.2: To reverse elements of list.

```
import java.util.*;
class CollectionsReverseDemo{
public static void main(String[] args){
ArrayList l=new ArrayList();
l.add(15);
l.add(0);
l.add(20);
l.add(10);
l.add(5);
System.out.println(l);//[15, 0, 20, 10, 5] Collections.reverse(l);
System.out.println(l);//[5, 10, 20, 0, 15]
}}
```

Array

Arrays class defines several utility methods for arrays.

Sorting the elements of array:

public static void sort(primitive[] p);//any primitive data type we can give

To sort the elements of primitive array according to default natural sorting order.

public static void sort(object[] o);

To sort the elements of object[] array according to default natural sorting order. In this case objects should be homogeneous and Comparable.

public static void sort(object[] o,Comparator c);

To sort the elements of object[] array according to customized sorting order.

Note: We can sort object[] array either by default natural sorting order (or) customized sorting order but we can sort primitive arrays only by default natural

sorting order.

Program 18.3

```
import java.util.*;
public class Jtc9 {
    public static void main(String[] args) {
        int arr[] = { 10, 45, 25, 6, 78, 12, 21 };
        int arr1[] = { 10, 45, 25, 6, 78, 12, 21 };
        int arr2[] = { 10, 45, 85, 45, 12, 78, 896 };
        for (int i = 0; i < arr.length; i++)
        { System.out.print(arr[i] + "\t");
        }
        for (int i = 0; i < arr1.length; i++)
        { System.out.print(arr1[i] + "\t");
        }
        for (int i = 0; i < arr2.length; i++)
        { System.out.print(arr2[i] + "\t");
        }
        System.out.println("\n***** arr After sorting *****");
        Arrays.sort(arr);
        for (int i = 0; i < arr.length; i++)
        { System.out.print(arr[i] + "\t");
        }
        System.out.println();
        System.out.println(Arrays.binarySearch(arr, 6));
        System.out.println(Arrays.binarySearch(arr, 9));
        System.out.println(Arrays.equals(arr, arr1));
        System.out.println(Arrays.equals(arr, arr2));
        Arrays.fill(arr, 32);
        for (int i = 0; i < arr.length; i++)
        { System.out.print(arr[i] + "\t");
```

```

}
Object ob[] = { "jtc", "india", "som", "rai", "white", "red" };
for (int i = 0; i < ob.length; i++) {
    System.out.print(ob[i] + "\t");
}
System.out.println();
List list = Arrays.asList(ob);
Iterator it = list.iterator();
while (it.hasNext()) {
    System.out.print(it.next()+ "\t");
}
System.out.println();
Arrays.sort(ob);
for (int i = 0; i < ob.length; i++) {
    System.out.print(ob[i] + "\t");
}
System.out.println();
}
}

```

Searching the elements of array

Arrays class defines the following methods to search elements of array.

```

public static int binarySearch(primitive[] p,primitive key);
public static int binarySearch(Object[] p, object key);
public static int binarySearch(Object[] p,Object key,Comparator c);

```

All rules of Arrays class `binarySearch()` method are exactly same as Collections class `binarySearch()` method.

18.23 Converting array to List

Arrays class defines the following method to view array as List.

public static List asList (Object[] o);

Strictly speaking we are not creating an independent List object just we are viewing array in List form.

- By using List reference if we are performing any change automatically these changes will be reflected to array reference similarly by using array reference if we are performing any change automatically these changes will be reflected to the List reference.

- By using List reference if we are trying to perform any operation which varies the size then we will get runtime exceptions saying `UnsupportedOperationException`.
- By using List reference if we are trying to insert heterogeneous objects we will get runtime exception saying `ArrayStoreException`.

Example

```
import java.util.Calendar;
import java.util.Formatter;

public class Jtc15 {
    public static void main(String[] args) {
        Formatter fmt1 = new Formatter();
        String name = "SomPraksh";
        int age = 23;
        fmt1.format("My Name is %s and age is %d", name, age);
        System.out.println(fmt1);
        Formatter fmt2 = new Formatter();
        fmt2.format("%c - %f - %b %n%05d - %o - %x - %X", 'C', 99.99, true,
            3456, 3456, 3456, 3456);
        System.out.println(fmt2);
        Formatter fmt3 = new Formatter();
        Calendar cal = Calendar.getInstance();
        fmt3.format("%tr %n%tc %n%tl:%tM", cal, cal, cal, cal);
        System.out.println(fmt3);
    }
}
```

Chapter 19

Generics

19.1 Introduction

The main objective of Generics is to provide Type-Safety and to resolve Type-Casting problems.

Case 1: Type-Safety

- Arrays are always type safe that is we can give the guarantee for the type of elements present inside array.
- For example if our programming requirement is to hold String type of objects it is recommended to use String array. In the case of string array we can add only string type of objects by mistake if we are trying to add any other type we will get compile time error.
- That is we can always provide guarantee for the type of elements present inside array and hence arrays are safe to use with respect to type that is arrays are type safe.
- But collections are not type safe that is we can't provide any guarantee for the type of elements present inside collection.
- For example if our programming requirement is to hold only string type of objects it is never recommended to go for ArrayList.
- By mistake if we are trying to add any other type we won't get any compile time error but the program may fail at runtime.

Hence we can't provide guarantee for the type of elements present inside collections that is collections are not safe to use with respect to type.

Case 2: Type-Casting

In the case of array at the time of retrieval it is not required to perform any type casting.

But in the case of collection at the time of retrieval compulsory we should perform type casting otherwise we will get compile time error.

- That is in collections type casting is bigger headache.
- To overcome the above problems of collections(type-safety, type casting) sun people introduced generics concept in 1.5v hence the main objectives of generics are:

- To provide type safety to the collections.
- To resolve type casting problems.
- To hold only string type of objects we can create a generic version of ArrayList as follows.

For this ArrayList we can add only string type of objects by mistake if we are trying to add any other type we will get compile time error that is through generics we are getting type safety.

At the time of retrieval it is not required to perform any type casting we can assign elements directly to string type variables.

That is through generic syntax we can resolve type casting problems.

19.2 Generic Classes

Until 1.4v a non-generic version of ArrayList class is declared as follows.

Example

```
class ArrayList{
add(Object o);
Object get(int index);
}
```

add() method can take object as the argument and hence we can add any type of object to the ArrayList. Due to this we are not getting type safety.

The return type of get() method is object hence at the time of retrieval compulsory we should perform type casting.

But in 1.5v a generic version of ArrayList class is declared as follows.

Based on our requirement T will be replaced with our provided type.

For Example to hold only string type of objects we can create ArrayList object as follows.

Example

```
ArrayList<String> l=new ArrayList<String>();
```

For this requirement compiler considered ArrayList class is

Example

```
class ArrayList<String>{
add(String s);
```

```
String get(int index);
}
```

- add() method can take only string type as argument hence we can add only string type of objects to the List.
- By mistake if we are trying to add any other type we will get compile time error

Hence through generics we are getting type safety.

At the time of retrieval it is not required to perform any type casting we can assign its values directly to string variables.

In Generics we are associating a type-parameter to the class, such type of parameterised classes are nothing but Generic classes. Generic class: class with type-parameter

Based on our requirement we can create our own generic classes also.

Example

```
class Account<T>{}
Account<Gold> g1=new Account<Gold>();
Account<Silver> g2=new Account<Silver>();
```

Example

```
public class Jtc11 {

    public static void main(String[] args) {
        Student stud = new Student();
        stud.studId = "JTC-001";
        stud.studId = 1245; //Auto Boxing
        stud.studId = 1245L; //Auto Boxing
        Student stud2 = new Student();
        stud2.studId = "JTC-002";
        stud2.studId = 1245; //Auto Boxing
        stud2.studId = 1245L; //Auto Boxing
        Student<String> st3 = new
        Student<String>();
```

```

st3.studId = "WB-001";
// st3.studId=4512;
Student<Long> st4 = new Student<Long>();
// st4=new Student<Integer>();
// st4.studId="WB-001";
st4.studId = 4512L;
// st4.studId=4512;
Student<?> st5 = new Student<Long>();
st5 = new Student<String>();
st5 = new Student<Integer>();
Student<? extends Number> st6 = new Student<Long>();
// st6=new Student<String>();
st6 = new Student<Integer>();
Employee emp1 = new Employee();
emp1.empId = "JTC-001";
emp1.empId = 4512;
emp1.empName = "SomPraksh";
Employee<Integer, String> emp2 = new Employee<Integer, String>();
emp2.empId = 1234;
// emp2.empId="";
System.out.println();
System.out.println();
User<Integer, Long, Address<String, Integer>> ref1 = new User<Integer, Long,
Address<String, Integer>>();
}
}
class Student<T> {
T studId;
}
class Employee<T1, T2> implements Compare<Employee<T1, T2>> {
T1 empId;
T2 empName;
// static T1 val;
public boolean compare(Employee<T1, T2> ref) {
return false;
}
}
interface Compare<T> {

```

```

public boolean compare(T ref);
}
class User<T1, T2, T3> {
    T1 uid;
    T2 phone;
    T3 uad;
}
class Address<T1, T2> {
    T1 aid;
    T2 pin;
}

```

19.3 Bounded Types

We can bound the type parameter for a particular range by using extends keyword such types are called bounded types.

- If x is a class then as the type parameter we can pass either x or its child classes. If x is an interface then as the type parameter we can pass either x or its Implementation classes.

But implements keyword purpose we can replace with extends keyword.

As the type parameter we can use any valid java identifier but it convention to use T always.

We can pass any no of type parameters need not be one.

```
HashMap<Integer,String> h=new HashMap<Integer,String>();
```

We can define bounded types even in combination also.

As the type parameter we can pass any type which extends Number class and implements Runnable interface.

We can't extend more than one class at a time.

19.4 Generic method and wild-card character (?)

methodOne(ArrayList<String> l)

This method is applicable for ArrayList of only String type.

Example

```

l.add("A");
l.add(null);

```

`l.add(10);` //(invalid) Within the method we can add only String type of objects and null to the List.

`methodOne(ArrayList<?> l):`

We can use this method for ArrayList of any type but within the method we can't add anything to the List except null.

This method is useful whenever we are performing only read operation.

methodOne(ArrayList<? Extends x> l)

- If x is a class then this method is applicable for ArrayList of either x type or its child classes.
- If x is an interface then this method is applicable for ArrayList of either x type or its implementation classes.
- In this case also within the method we can't add anything to the List except null.

methodOne(ArrayList<? super x> l)

- If x is a class then this method is applicable for ArrayList of either x type or its super classes.
- If x is an interface then this method is applicable for ArrayList of either x type or super classes of implementation class of x.
- But within the method we can add x type objects and null to the List.

Which of the following declarations are allowed?

`ArrayList<String> l1=new ArrayList<String>();` //(valid)

`ArrayList<?> l2=new ArrayList<String>();` //(valid)

`ArrayList<?> l3=new ArrayList<Integer>();` //(valid)

`ArrayList<? extends Number> l4=new ArrayList<Integer>();` //(valid)

`ArrayList<? extends Number> l5=new ArrayList<String>();` (invalid)

We can declare the type parameter either at class level or method level.

Declaring type parameter at class level

```
class Test<T>{
```

We can use anywhere this 'T'.

```
}
```

Declaring type parameter at method level

We have to declare just before return type.

Example

```

public <T> void methodOne1(T t){} //valid
public <T extends Number> void methodOne2(T t){} //valid
public <T extends Number&Comparable> void methodOne3(T t){} //valid
public <T extends Number&Comparable&Runnable> void methodOne4(T t){} //
valid
public <T extends Number&Thread> void methodOne(T t){} //invalid
public <T extends Number&Thread> void methodOne(T t){} //valid
public <T extends Runnable&Number> void methodOne(T t){} //invalid

```

Output:

```

public <T extends Number&Runnable> void methodOne(T t){} //valid

```

19.5 Communication with non generic code

To provide compatibility with old version sun people compramized the concept of generics in very few area's the following is one such area.

Example

```

import java.util.*;
class Test{
public static void main(String[] args){
ArrayList<String> l=new ArrayList<String>();
l.add("A");
//l.add(10); //C.E:cannot find symbol,method add(int)
methodOne(l);
l.add(10.5); //C.E:cannot find symbol,method
add(double)
System.out.println(l); // [A, 10, 10.5, true]
}
public static void methodOne(ArrayList l){
l.add(10);
l.add(10.5);
l.add(true);
}}

```

Generics concept is applicable only at compile time, at runtime there is no such type of concept. Hence the following declarations are equal.

Example

```

import java.util.*;

public class Jtc12 {
    public static void main(String[] args)
    { System.out.println("-- Without Generics --");
      List list = new ArrayList();
      list.add("SomPraksh");
      list.add("Manish");
      list.add("Hello");
      list.add("Welcome");
      list.add(new Integer(12));
      Iterator it = list.iterator();
      while (it.hasNext()) {
          Object obj = it.next();
          if (obj instanceof Integer) {
              Integer in = (Integer) obj; System.out.println(in);
          } else if (obj instanceof String) {
              String str = (String) obj;
              System.out.println(str);
          }
      }
      System.out.println("\n-- Using Generics with
      List --"); List<String> list1 = new
      ArrayList<String>(); list1.add("SomPraksh");
      list1.add("Manish");
      list1.add("Hello");
      list1.add("Welcome");
      // list1.add(new Integer(12));
      Iterator<String> it1 = list1.iterator();
      while (it1.hasNext()) {
          String str = it1.next();
          System.out.println(str);
      }
    }
}

```

```

}
System.out.println("\n\nFrom Map Object ");
Map<Integer, String> map = new LinkedHashMap<Integer, String>();
map.put(1234, "Som");
map.put(8767, "Praksh");
map.put(5677, "Manish");
map.put(2343, "Rai");
map.put(9898, "Chandan");
Set<Map.Entry<Integer, String>> set = map.entrySet();
Iterator<Map.Entry<Integer, String>> it3 = set.iterator();
while (it3.hasNext()) {
    Map.Entry<Integer, String> entry = (Map.Entry<Integer, String>) it3.next();
    System.out.println(entry.getKey() + "\t" + entry.getValue());
}}

```

Example

```

import java.util.*;

public class Jtc4{
    public static void main(String JtcArgs[]){
        //Before Java 7
        List<String> list=new ArrayList<String>();
        Map<Integer,String> map=new HashMap<Integer,String>();
        Student<Integer,String,Long,String> st=new Student<Integer,String,Long,String>();
        //From Java 7
        List<String> list2=new ArrayList<>();
        Map<Integer,String> map2=new HashMap<>();
        Student<Integer,String,Long,String> st2=new Student<>();
    }
}

class Student<I,N,P,A>{
    I id;

```



```
N name;  
P phone;  
A add;  
}
```

Chapter 20

Internationalization (118N)

20.1 Introduction

- The process of designing a web application such that it supports various countries, various languages without performing any changes in the application is called Internationalization.
- If the request is coming from India then the response should be in India specific form, and if the request is from US then the response should be in US specific form.

We can implement Internationalization by using the following classes.

They are:

Locale
NumberFormat
DateFormat

20.2 Locale

- A Locale object can be used to represent a geographic (country) location (or) language.
- Locale class present in java.util package.
- It is a final class and direct child class of Object and , implements Cloneable, and Serializable Interfaces

How to create a Locale Object

We can create a Locale object by using the following constructors of Locale class.

```
Locale l=new Locale(String language);
```

```
Locale l=new Locale(String language,String country);
```

Locale class already defines some predefined Locale constants. We can use these constants directly.

Example

Locale. UK

Locale. US

Locale. ITALY

Locale. CHINA

Important methods of Locale class:

```
public static Locale getDefault()
public static void setDefault(Locale l)
public String getLanguage()
public String getDisplayLanguage(Locale l)
public String getCountry()
public String getDisplayCountry(Locale l)
public static String[] getISOLanguages()
public static String[] getISOCountries()
public static Locale[] getAvailableLocales()
```

Example

for Locale

```
import java.util.*;
public class Jtc11 {

    public static void main(String[] args)
    { System.out.println(Locale.getDefault());
      Locale locales[] = Locale.getAvailableLocales();
      for (int i = 0; i < locales.length; i++)
      { System.out.print(locales[i] + " , ");
        }
      String countries[] = Locale.getISOCountries();
      for (int i = 0; i < countries.length; i++)
      { System.out.print(countries[i] + " , ");
        }
      String languagess[] =
      Locale.getISOLanguages(); for (int i = 0; i <
      languagess.length; i++) {
```

```
System.out.print(languagess[i] + ", ");
}
System.out.println();
Locale loc = new Locale("EN");
System.out.println(loc.getCountry());
System.out.println(loc.getDisplayCountry());
System.out.println(loc.getDisplayLanguage());
System.out.println(loc.getDisplayName());
System.out.println(loc.getVariant());
System.out.println(loc.getDisplayVariant());
Locale loc1 = new Locale("EN", "US");
System.out.println( loc1.getCountry());
System.out.println(loc1.getDisplayCountry());
System.out.println( loc1.getDisplayLanguage());
System.out.println(loc1.getDisplayName());
}
}
```

20.3 NumberFormat

Various countries follow various styles to represent number.

Example	
double d=123456.789;	
1,23,456.789-----	INDIA
123,456.789-----	US
123.456,789-----	ITALY

- By using NumberFormat class we can format a number according to a particular Locale.
- NumberFormat class present in java.Text package and it is an abstract class.
- Hence we can't create an object by using constructor.

NumberFormat nf=new NumberFormat(); -----invalid

Getting NumberFormat object for the specific Locale

The methods are exactly same but we have to pass the corresponding Locale object as argument.

Example

public static NumberFormat getInstance(Locale l);

Once we got NumberFormat object we can call the following methods to format and parse numbers.

public String format(long l); public String format(double d);

To convert a number from java form to Locale specific form

public Number parse(String source)throws ParseException

To convert from Locale specific String form to java specific form.

Requirement: Write a program to display java number form into Italy specific form.

Example

```
import java.util.*;
import java.text.*;
class NumberFormatDemo{
public static void main(String args[]){
double d=123456.789;
NumberFormat nf=NumberFormat.getInstance(Locale.ITALY);
System.out.println("ITALY form is :"+nf.format(d));
}}
```

Q1 : Write a program to print a java number in INDIA, UK, US and ITALY currency formats.

Ans :

```
import java.util.*;
import java.text.*;
class NumberFormatDemo{
public static void main(String args[]){
double d=123456.789;
Locale INDIA=new Locale("pa","IN");
NumberFormat nf=NumberFormat.getCurrencyInstance(INDIA);
System.out.println("INDIA notation is :"+nf.format(d));
NumberFormat
nf1=NumberFormat.getCurrencyInstance(Locale.UK);
System.out.println("UK notation is :"+nf1.format(d));
NumberFormat
nf2=NumberFormat.getCurrencyInstance(Locale.US);
```

```

System.out.println("US notation is :"+nf2.format(d));
NumberFormat nf3=NumberFormat.getCurrencyInstance(Locale.ITALY);
System.out.println("ITALY notation is :"+nf3.format(d));
}}

```

20.4 Setting Maximum, Minimum, Fraction and Integer digits

NumberFormat class defines the following methods for this purpose.

```

public void setMaximumFractionDigits(int n);
public void setMinimumFractionDigits(int n);
public void setMaximumIntegerDigits(int n);
public void setMinimumIntegerDigits(int n);

```

Example

```

import java.text.*;
public class NumberFormatExample{
public static void main(String[] args){
NumberFormat nf=NumberFormat.getInstance();
nf.setMaximumFractionDigits(3);
System.out.println(nf.format(123.4));
System.out.println(nf.format(123.4567));
nf.setMinimumFractionDigits(3);
System.out.println(nf.format(123.4));
System.out.println(nf.format(123.4567));
nf.setMaximumIntegerDigits(3);
System.out.println(nf.format(1.234));
System.out.println(nf.format(123456.789));
nf.setMinimumIntegerDigits(3);
System.out.println(nf.format(1.234));
System.out.println(nf.format(123456.789));
}}

```

20.5 DateFormat

- Various countries follow various styles to represent Date. We can format the date according to a particular locale by using DateFormat class.
- DateFormat class present in java.text package and it is an abstract class.

Getting DateFormat object for default Locale

DateFormat class defines the following methods for this purpose.

The default style is Medium style

Getting DateFormat object for the specific Locale:

public static DateFormat getInstance(int style, Locale l);

Once we got DateFormat object we can format and parse Date by using the following methods.

public String format(Date date);

To convert the date from java form to locale specific string form.

public Date parse(String source)throws ParseException

To convert the date from locale specific form to java form.

Q2 : Write a program to represent current system date in all possible styles of us format.

Ans :

```
import java.text.*;
import java.util.*;
public class DateFormatDemo{
    public static void main(String args[]){
        System.out.println("full form is :"+DateFormat.getInstance(0).format(new
        Date()));
        System.out.println("long form is :"+DateFormat.getInstance(1).format(new
        Date())); System.out.println("medium form is :"+DateFormat.getInstance(2).
        format(new Date())); System.out.println("short form is :"+DateFormat.
        getInstance(3).format(new Date()))
    }
}
```

Note: The default style is medium style.

Q3 : Write a program to represent current system date in UK, US and ITALY styles.

Ans:

```
import java.text.*;
```

```
import java.util.*;
public class DateFormatDemo{
public static void main(String args[]){
DateFormat UK=DateFormat.getDateInstance(0,Locale.UK);
DateFormat US=DateFormat.getDateInstance(0,Locale.US);
DateFormat ITALY=DateFormat.getDateInstance(0,Locale.ITALY);
System.out.println("UK style is :"+UK.format(new Date()));
System.out.println("US style is :"+US.format(new Date()));
System.out.println("ITALY style is :"+ITALY.format(new
Date()));
}}
```

Getting DateFormat object to get both date and time:

DateFormat class defines the following methods for this.

Example

```
import java.text.*;
import java.util.*;
public class DateFormatDemo{
public static void main(String args[]){
DateFormat ITALY=DateFormat.getDateTimeInstance(0,0,Locale.ITALY);
System.out.println("ITALY style is:"+ITALY.format(new
Date()));
}}
```

20.6 System properties

- For every system some persistence information is available in the form of system properties. These may include name of the os, java version, vendor of jvm , userCountry etc.
- We can get system properties by using `getProperties()` method of system class.
- The following program displays all the system properties.

Example

```
import java.util.*;
class Test{
public static void main(String args[]){ //Properties is a class in util package.
//here getProperties() method returns the Properties object.
Properties p=System.getProperties();
```



```
p.list(System.out);  
}}
```

How to set system property from the command prompt:

We can set a system property explicitly from the command prompt by using `-D` option.

The main advantage of setting System Properties is we can customize the behaviour of java program.

Chapter 21

Regular Expression

21.1 Introduction

A Regular Expression is a expression which represents a group of Strings according to a particular pattern.

Example

We can write a Regular Expression to represent all valid mail ids.

We can write a Regular Expression to represent all valid mobile numbers.

The main important application areas of Regular Expression are

- To implement validation logic.
- To develop Pattern matching applications.
- To develop translators like compilers, interpreters etc. To develop digital circuits.
- To develop communication protocols like TCP/IP, UDP etc.

Example

```
import java.util.regex.*;
class RegularExpressionDemo{
public static void main(String[] args){
int count=0;
Pattern p=Pattern.compile("ab");
Matcher m=p.matcher("abbbabbaba");
while(m.find()){
count++;
System.out.println(m.start()+"-----"+m.end()+"-----"+m.group());
}
System.out.println("The no of occurences:"+count);
}}
```

21.2 Pattern Class

A Pattern object represents "compiled version of Regular Expression".

We can create a Pattern object by using compile() method of Pattern class.

```
public static Pattern compile(String regex);
```

Example

```
Pattern p=Pattern.compile("ab");
```

Note: if we refer API we will get more information about pattern class.

21.3 Matcher

A Matcher object can be used to match character sequences against a Regular Expression.

We can create a Matcher object by using matcher() method of Pattern class.

```
public Matcher matcher(String target);
Matcher m=p.matcher ("abbbabbaba");
```

Important methods of Matcher class:

boolean find();

It attempts to find next match and returns true if it is available otherwise returns false.

int start();

Returns the start index of the match.

int end();

Returns the offset(equalize) after the last character matched.(or) Returns the "end+1" index of the matched.

String group();

Returns the matched Pattern.

Note: Pattern and Matcher classes are available in java.util.regex package, and introduced in 1.4 version.

21.4 Character Classes

1.	[abc]-----	Either 'a' or 'b' or 'c'
2.	[^abc] -----	Except 'a' and 'b' and 'c'
3.	[a-z] -----	Any lower case alphabet symbol
4.	[A-Z] -----	Any upper case alphabet symbol
5.	[a-zA-Z] -----	Any alphabet symbol
6.	[0-9] -----	Any digit from 0 to 9
7.	[a-zA-Z0-9] -----	Any alphanumeric character
8.	[^a-zA-Z0-9] -----	Any special character

Example

```
import java.util.regex.*;
class RegularExpressionDemo{
public static void main(String[] args){
Pattern p=Pattern.compile("x");
Matcher m=p.matcher("a1b7@z#");
while(m.find()){
System.out.println(m.start()+"-----"+m.group());
}}}
```

21.5 Quantifiers

Quantifiers can be used to specify no of characters to match.

a-----Exactly one 'a'
a+-----At least one 'a'
a*-----Any no of a's including zero number
a? -----At most one 'a'

Example

```
import java.util.regex.*;
class RegularExpressionDemo{
public static void main(String[] args){
Pattern p=Pattern.compile("x");
Matcher m=p.matcher("abaabaaab");
while(m.find()){
System.out.println(m.start()+"-----"+m.group());
}}}
```

21.6 Pattern class split() method

Pattern class contains split() method to split the given string against a regular expression.

Example

```
import java.util.regex.*;
class RegularExpressionDemo{
public static void main(String[] args){
Pattern p=Pattern.compile("\\s");
```

```
String[] s=p.split("java training center");  
for(String s1:s){  
    System.out.println(s1);  
    //software  
    //solutions  
}}
```

Chapter 22

IO Package

22.1 Introduction

IO stands for input output. In java we can write the different types of programs but sun has provided the very large API to deal with the taking inputs and displaying corresponding output. Here are some basic points about I/O:

- Data in files on your system is called persistent data because it persists after the program runs.
- Files are created through streams in Java code.
- A stream is a linear, sequential flow of bytes of input or output data.
- Streams are written to the file system to create files.
- Streams can also be transferred over the Internet.
- Three streams are created for us automatically:

System.out - standard output stream

System.in - standard input stream

System.err - standard error

Table 1. Data Fields in the System Class

Variable	Type	Purpose
err	PrintStream	the "standard error" output stream. This stream is already open and ready to accept output data.
in	InputStream	the "standard input" stream. This stream is already open and ready to supply input data.
out	PrintStream	the "standard output" stream. This stream is already open and ready to accept output data.

22.2 Stream

Stream is the kind of pipe through which you can transfer the data from one place to another place. Depending on operation which you performing the operation are defining into two types: -

- 1) Reading the data
- 2) Writing the data

Now depending types of data these operation has been divided into two categories:

1) Byte Stream**2) Character Stream**

Byte Input Stream: The data which you are reading in the form of byte is called as byte input stream. For all the byte input stream `java.io.InputStream` is the base class.

Example

```
import java.io.*;
public class CopyFile {
    public static void main(String args[]) throws IOException {
        FileInputStream in = null;
        FileOutputStream out = null;
        try {
            in = new FileInputStream("input.txt");
            out = new FileOutputStream("output.txt");
            int c;
            while ((c = in.read()) != -1) {
                out.write(c);
            } finally {
                if (in != null) {
                    in.close();
                } if (out != null) {
                    out.close();
                } }
        }
    }
}
```

Byte Output Stream: The data which you are writing to some output device in the form of bytes is called Byte Output Stream. For all the byte output stream `java.io.OutputStream` is the base class.

Example

```
import java.io.*;
public class CopyFile {
    public static void main(String args[]) throws IOException {
        FileReader in = null;
        FileWriter out = null;
        try {
            in = new FileReader("input.txt");
            out = new FileWriter("output.txt");
```

```

int c;
while ((c = in.read()) != -1) {
    out.write(c);
}    }finally {
        if (in != null) {
            in.close();
        } if (out != null) {
            out.close();
        }    } }

```

The data which you are reading and writing in the form of character set is called character input/output stream.

Character Input Stream: The data which you are reading in the form of characters is called character input stream. For all the character Input Stream `java.io.Reader` is the base class.

Character Output stream: The data which you are writing to some output device in the form of characters is called Character Output Stream. For all the character output stream `java.io.writer` is the base class.

Example

```

import java.io.*;
public class Jtc207 {
    public static void main(String[] args) {
        InputStream is=new BufferedInputStream(System.in);
        PrintStream ps=System.out;
        System.out.println("Enter the char :");
        char ch=' ';
        do{
            try{
                int i=is.read();
                ch=(char)i;
                ps.write(ch);
            }catch(Exception e){
                e.printStackTrace();
            }
        }while(ch!='\n');
    }
}

```


22.3 What to use

There are a number of different questions to consider when dealing with the java.io package:

- What is your format: text or binary?
- Do you want random access capability?
- Are you dealing with objects or non-objects?
- What are your sources and sinks for data?
- Do you need to use filtering?

Text or binary

What's your format for storing or transmitting data? Will you be using text or binary data?

- If you use binary data, such as integers or doubles, then use the `InputStream` and `OutputStream` classes.
- If you are using text data, then the `Reader` and `Writer` classes are right.

Random access

- Do you want random access to records? Random access allows you to go anywhere within a file and be able to treat the file as if it were a collection of records.
- The `RandomAccessFile` class permits random access. The data is stored in binary format. Using random access files improves performance and efficiency.

Object or non-object

Are you inputting or outputting the attributes of an object? If the data itself is an object, then use the `ObjectInputStream` and `ObjectOutputStream` classes.

Sources and sinks for data

What is the source of your data? What will be consuming your output data, that is, acting as a sink?

You can input or output your data in a number of ways: sockets, files, strings, and arrays of characters. Any of these can be a source for an `InputStream` or `Reader` or a sink for an `OutputStream` or `Writer`.

22.4 Filtering

Do you need filtering for your data? There are a couple ways to filter data.

Buffering is one filtering method. Instead of going back to the operating system for each byte, you can use an object to provide a buffer.

Checksumming is another filtering method. As you are reading or writing a stream, you might want to compute a checksum on it. A checksum is a value you can use later on to make sure the stream was transmitted properly.

22.5 Storing data records

A data record is a collection of more than one element, such as names or addresses. There are three ways to store data records:

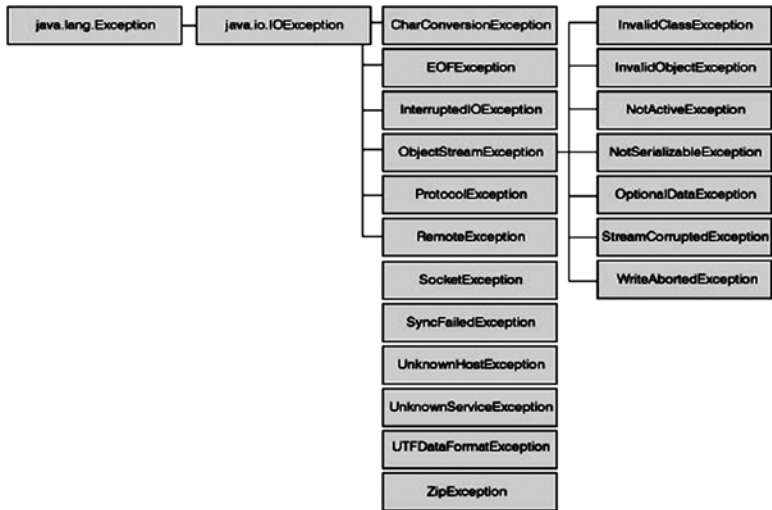
- Use delimited records, such as a mail-merge style record, to store values. On output, data values are converted to strings separated by delimiters such as the tab character, and ending with a new-line character. To read the data back into Java code, the entire line is read and then broken up using the `StringTokenizer` class.
- Use fixed size records to store data records. Use the `RandomAccessFile` class to store one or more records. Use the `seek()` method to find a particular record. If you choose this option to store data records, you must ensure strings are set to a fixed maximum size.
- Alternatively, you can use variable length records if you use an auxiliary file to store the lengths of each record. Use object streams to store data records. If object streams are used, no skipping around is permitted, and all objects are written to a file in a sequential manner.

22.6 Exceptions

Almost every input or output method throws an exception. Therefore, any time you do I/O, you need to catch exceptions. There is a large hierarchy of I/O exceptions derived from `IOException`. Typically you can just catch `IOException`, which catches all the derived exceptions. However, some exceptions thrown by I/O methods are not in the `IOException` hierarchy. One such exception is the `java.util.zip.DataFormatException`. This exception is thrown when invalid or corrupt data is found while data being read from a zip file is being uncompressed. `java.util.zip.DataFormatException` has to be caught explicitly because it is not in the `IOException` hierarchy.

Remember, exceptions in Java code are thrown when something unexpected happens.

List of exceptions



This figure shows a list of exceptions:

- `EOFException` signals when you reach an end-of-file unexpectedly.
- `UnknownHostException` signals that you are trying to connect to a remote host that does not exist.
- `ZipException` signals that an error has occurred in reading or writing a zip file.

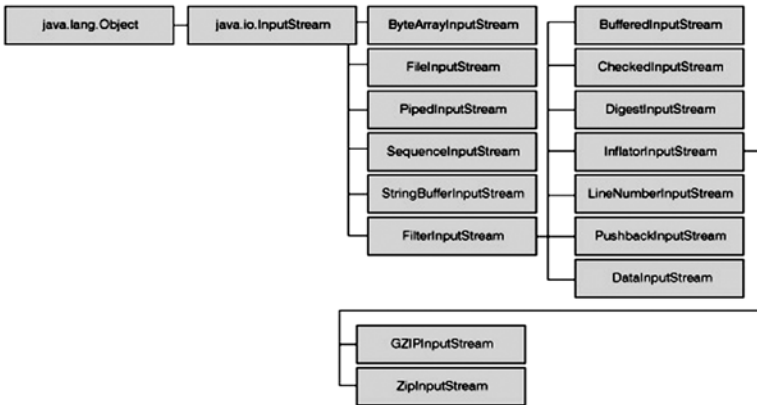
Typically, an `IOException` is caught in the try block, and the value of the `toString()` method of the `Object` class in the `IOException` is printed out as a minimum. You should handle the exception in a manner appropriate to the application

Input stream classes

In the `InputStream` class, bytes can be read from three different sources:

- An array of bytes
- A file
- A pipe

Sources, such as `ByteArrayInputStream` and `FilterInputStream`, subclass the `InputStream` class. The subclasses under `InflaterInputStream` and `FilterInputStream` are listed in the figure below.



InputStream methods

Various methods are included in the `InputStream` class.

- `abstract int read()` reads a single byte, an array, or a subarray of bytes. It returns the bytes read, the number of bytes read, or -1 if end-of-file has been reached.
- `read()`, which takes the byte array, reads an array or a subarray of bytes and returns a -1 if the end-of-file has been reached.
- `skip()`, which takes long, skips a specified number of bytes of input and returns the number of bytes actually skipped.
- `available()` returns the number of bytes that can be read without blocking. Both the input and output can block threads until the byte is read or written.
- `close()` closes the input stream to free up system resources.

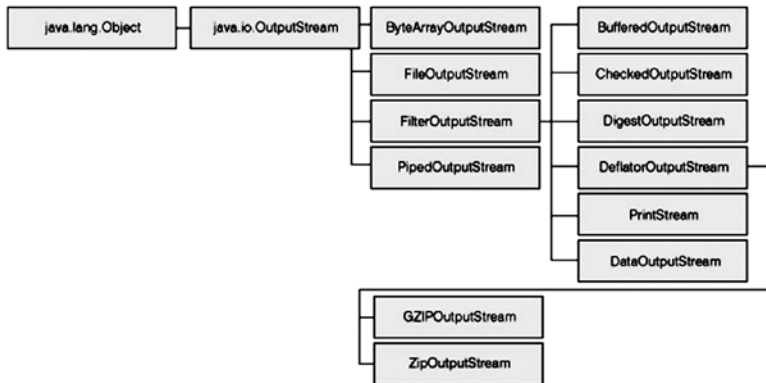
InputStream marking

- Some, but not all, `InputStreams` support marking. Marking allows you to go back to a marked place in the stream like a bookmark for future reference. Remember, not all `InputStreams` support marking. To test if the stream supports the `mark()` and `reset()` methods, use the boolean `markSupported()` method.
- The `mark()` method, which takes an integer `read_limit`, marks the current position in the input stream so that `reset()` can return to that position as long as no more than the specified number of bytes have been read between the `mark()` and `reset()`.

OutputStream classes

Bytes can be written to three different types of sinks:

- An array of bytes
- A file
- A pipe



The following figure shows OutputStream classes.

Let's look at some examples of OutputStream classes. Before sending an OutputStream to its ultimate destination, you can filter it. For example, the BufferedOutputStream is a subclass of the FilterOutputStream that stores values to be written in a buffer and writes them out only when the buffer fills up.

- CheckedOutputStream and DigestOutputStream are part of the FilterOutputStream class. They calculate checksums or message digests on the output.
- DeflaterOutputStream writes to an OutputStream and creates a zip file. This class does compression on the fly.
- The PrintStream class is also a subclass of FilterOutputStream, which implements a number of methods for displaying textual representation of Java primitive types. For
- Example
 - `println(long)`
 - `println(int)`
 - `println(float)`
 - `println(char)`

`DataOutputStream` implements the `DataOutput` interface. `DataOutput` defines the methods required for streams that can write Java primitive data types in a machine-independent binary format.

OutputStream methods

The `OutputStream` class provides several methods:

- The abstract void `write()` method takes an integer byte and writes a single byte.
- The void `write()` method takes a byte array and writes an array or subarray of bytes.
- The void `flush()` method forces any buffered output to be written.
- The void `close()` method closes the stream and frees up system resources.

It is important to close your output files because sometimes the buffers do not get completely flushed and, as a consequence, are not complete.

DataInput and DataOutput

The `DataInput` and `DataOutput` classes define the methods required for streams that can read Java primitive data types in a machine-independent binary format. They are implemented by `RandomAccessFile`.

The `ObjectInput` interface extends the `DataInput` interface and adds methods for deserializing objects and reading bytes and arrays of bytes.

The `ObjectOutputStream` class creates a stream of objects that can be deserialized by the `ObjectInputStream` class.

The `ObjectOutput` interface extends the `DataOutput` interface and adds methods for serializing objects and writing bytes and arrays of bytes.

`ObjectOutputStream` is used to serialize objects, arrays, and other values to a stream.

22.7 What are Readers?

Readers are character-based input streams that read Unicode characters.

- `read()` reads a single character and returns a character read as an integer in the range from 0 to 65535 or a -1 if the end of the stream is reached.
- abstract `read()` reads characters into a portion of an array (starting at offset up to length number of characters) and returns the number of characters read or -1 if the end of the stream is reached.

Read() API:

```
int read()
```

```
read (char[] buffer,  
int offset, int length)
```

Character input streams

Let's take a look at the different character input streams in the java.io package.

- Strings
- Character arrays
- Pipes

InputStreamReader is a character input stream that uses a byte input stream as its data source and converts it into Unicode characters.

LineNumberReader, a subclass of BufferedReader, is a character input stream that keeps track of the number of lines of text that have been read from it.

PushbackReader, a subclass of FilterReader, is a character input stream that uses another input stream as its input source and adds the ability to push characters back onto the stream.

22.8 What are Writers?

Writers are character-based output streams that write character bytes and turn Unicode into bytes. The base class includes these methods:

- The void write() method, which takes a character and writes single character in 16 low-order bits
- The abstract void write() method, which takes a character array and writes a portion of an array of characters

Write() API:

```
void write(int character)
```

```
void write(char[] buffer, int offset, int length)
```

Character output streams

Let's take a look at the different character output streams in the java.io package. There are several branches of this inheritance tree you can explore. Like Readers, any of the branches are available. Sinks for Writer output can be:

- Strings
- CharArray
- Pipes

`OutputStreamWriter` uses a byte output stream as the destination for its data.

`BufferedWriter` applies buffering to a character output stream, thus improving output efficiency by combining many small write requests into a single large request.

`FilterWriter` is an abstract class that acts as a superclass for character output streams. The streams filter the data written to them before writing it to some other character output stream.

`PrintWriter` is a character output stream that implements `print()` and `println()` methods that output textual representations of primitive values and objects.

Chapter 23

Sources and Sinks

23.1 Introduction

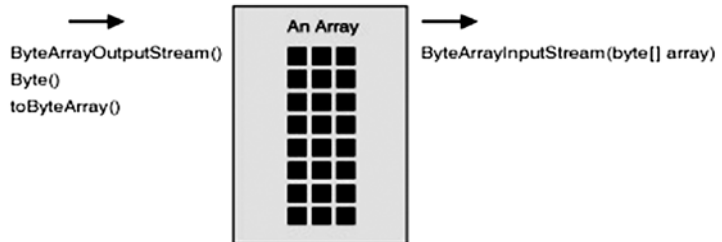
This section focuses on sources and sinks. Code samples help you visualize how some of these classes work:

- Byte arrays
- Pipes
- Sequences
- Char arrays

23.2 Byte arrays

Let's head toward byte array territory. Are bytes going to come from a byte array? When an object of a `ByteArrayInputStream` is created, it is passed an array. You can then read bytes from that array as an `InputStream`.

`ByteArrayOutputStream` writes to a byte array. Bytes that are written to the output stream are added to an internal array. You may be wondering how you get to the byte array. The `toByteArray()` method returns to you the byte array at any point in time. Also, a `toString()` method is available, which returns the byte array as a `String`. Therefore, the byte array can be either a source or a sink.



23.3 Pipes in Java code

What is a pipe? A pipe is a means of communication between two threads of a Java program. One thread writes to a piped output stream and another thread reads from a piped input stream, which is connected to the piped output stream.

A pipe between two threads works like a pipe works between two processes in UNIX and

MS-DOS. In those operating systems, the output from one process can be piped to the input of another process. For example, in MS-DOS, the command “dir | more” pipes the output of the “dir” command to the input of the “more” program.

Pipe: Example code

Although pipes are usually used with threads, this example simply writes data to a pipe and then reads it back later. First, a `PipedReader` is constructed, then a `PipedWriter` that writes to that `PipedReader`. The attributes are written to the pipe as strings with a vertical bar as a delimiter and a new-line character placed at the end. The entire contents of the `PipedReader` are read as a `String` and displayed.

We’ll show later, in Tokenizing, how to use a `StringTokenizer` to break up a string that is delimited into individual values.

Example

```
import java.io.*;
import java.util.*;

class PipedExample{
    static    BufferedReader    system_in    =    new    BufferedReader(new
    InputStreamReader(System.in));
    public static void main(String argv[]){
        PipedReader pr = new PipedReader();
        PipedWriter pw = null;
        try {
            pw = new PipedWriter(pr);
        }
        catch (IOException e)
        { System.err.println(e);
        }

        // Create it {
        // Read in three hotels
        for (int i = 0; i < 3; i++){
            Hotel a_hotel = new Hotel(); a_hotel.input(system_in);
            a_hotel.write_to_pw(pw);
        }
        }
        // Print it
```

```

{
char [] buffer = new char[1000];
int length = 0;
try
{
length = pr.read(buffer);
}
catch (IOException e)
{ System.err.println(e);
}
String output =new String(buffer, 0, length);
System.out.println("String is ");
System.out.println(output);
}}
class Hotel{
private String name;
private int rooms;
private String location;
boolean input(BufferedReader in)
{
try{
System.out.println("Name: ");
name=in.readLine();
System.out.println("Rooms: ");
String temp = in.readLine();
rooms = to_int(temp);
System.out.println("Location: ");
location = in.readLine();
}
catch(IOException e){
System.err.println(e);
return false;
}
return true;
}
boolean write_to_pw(PipedWriter pw){
try{
pw.write(name);

```

```

Integer i = new Integer(rooms);
pw.write(i.toString());
pw.write(location);
pw.write('backslash n');
// red font indicates that an actual backslash n (carriage return character)
// should be inserted in the code.
}
catch(IOException e)
{ System.err.println(e); return false;
}
return true;
}
void debug_print()
{
System.out.println("Name : " + name +
": Rooms : " + rooms + ": at : " + location + ":");
}
static int to_int(String value)
{
int i = 0;
try
{
i = Integer.parseInt(value);
}
catch(NumberFormatException e)
{}
return i;
}}

```

Sequences

`SequenceInputStream` combines input streams and reads each input stream until the end. You can use `SequenceInputStream` to read two or three streams as if they were one stream. You can concatenate streams from various sources, such as two or more files. To construct a `SequenceInputStream`, you can specify either two streams or an `Enumeration`, which represents a set of input streams. A program such as “cat” in UNIX would use something like this.

`SequenceInputStream` reads from the first underlying input stream that it is given.

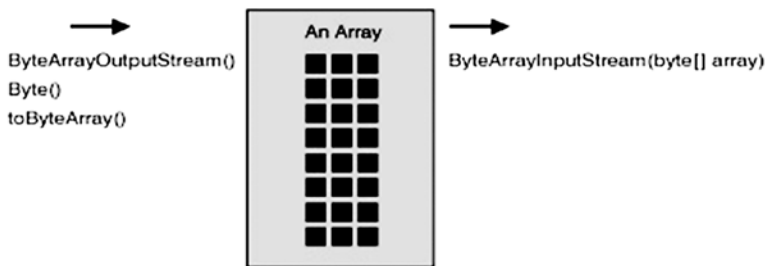
When the first one is exhausted, it opens the next input stream and reads that one. When that one is exhausted, it reads the next, and so on.

The user does not know when the input is transferred from one stream to the next stream; another byte is simply read. When all the input streams have been read, an EOF is received from the entire input stream.

23.4 Char arrays

Now let's explore Readers and Writers. The read and write methods input and output a character. Similar to `ByteArrayOutputStream`, a `CharArrayWriter` writes characters to a char array. As a character is written to a `CharArrayWriter` object, it's added to an array of characters whose size is automatically incremented.

At any point in time, we can get the character array that we have filled up. The `toCharArray()` method returns an array of characters. A `CharArrayReader` uses a character array as a source. Typically, the array is one that has been created with a `CharArrayWriter` object. With an alternative constructor, you can specify not only the array, but where to start in the array (an offset) and how many bytes to read (a length) before you return an EOF character.



String: Example code

`StringWriter` works like `CharArrayWriter`. An internal `StringBuffer` object is the destination of the characters written. Methods associated with the class are `getBuffer()`, which returns the `StringBuffer` itself, and `toString()`, which returns the current value of the string.

`StringReader` works like `CharArrayReader`. It uses a `String` object as the source of the characters it returns. When a `StringReader` is created, you must specify the `String` that it is read from.

In this example, instead of writing to a pipe, we are going to write to a `String`. A `StringWriter` object is constructed. After the output is complete, we obtain the

contents with `toString()` and print it out. This works like the previous example with `PipedReader` and `PipedWriter`, except a `String` is used to contain the data.

Example

```
import java.io.*;
import java.util.*;
class StringExample{
static BufferedReader system_in = new BufferedReader(new InputStreamReader
(System.in));
public static void main(String argv[]){
StringWriter sw = new StringWriter();
// Create it
{
// Read in three hotels
for (int i = 0; i < 3; i++){
Hotel a_hotel = new Hotel(); a_hotel.input(system_in); a_hotel.write_to_
string(sw);
}}
// Print it
{
String output = sw.toString();
System.out.println("String is ");
System.out.println(output);
}}
class Hotel{
private String name; private int rooms; private String location;
boolean input(BufferedReader in){
try {
System.out.println("Name: ");
name=in.readLine(); System.out.println("Rooms: ");
String temp = in.readLine();
rooms = to_int(temp);
System.out.println("Location: ");
location = in.readLine();
}catch(IOException e)
{ System.err.println(e); return false;
}
return true;
```

```

}
boolean write_to_string(StringWriter sw)
{
sw.write(name);
Integer i = new Integer(rooms); sw.write(i.toString()); sw.write(location);
sw.write('backslash n');
// red font indicates that an actual backslash n (carriage return character)
// should be inserted in the code. return true;
}
void debug_print()
{
System.out.println("Name : " + na
": Rooms : " + rooms + ": at : " + location+ ":");
}
static int to_int(String value)
{
int i = 0;
try {
i = Integer.parseInt(value);
}
catch(NumberFormatException e)
{}
return i;
}
}

```

InputStreamReader

InputStreamReader reads bytes from an InputStream and converts them to characters. An InputStreamReader uses the default character encoding for the bytes, which is usually ASCII. If the bytes that are being read are ASCII bytes, only a byte at a time is used to form a character.

If the bytes are not ASCII, such as Chinese or another language, you want conversion to Unicode as well. Specific encoding of the byte stream is necessary, and the InputStreamReader converts it to Unicode. With an alternate constructor, you can specify the encoding of the bytes on the InputStream.

OutputStreamReader

`OutputStreamWriter` is similar to `InputStreamReader`. The output characters, which are in Unicode, are converted to the underlying format of the machine using an `OutputStreamWriter`. The converted characters are written to an `OutputStream`. The underlying default format is typically ASCII. However, you can state a specific encoding scheme with an alternate constructor.

Chapter 24

Files

24.1 Introduction

This section examines the `File` class, an important non-stream class that represents a file or directory name in a system-independent way. The `File` class provides methods for:

- Listing directories
- Querying file attributes
- Renaming and deleting files

24.2 The File classes

The `File` class manipulates disk files and is used to construct `FileInputStreams` and `FileOutputStreams`. Some constructors for the `File` I/O classes take as a parameter an object of the `File` type. When we construct a `File` object, it represents that file on disk. When we call its methods, we manipulate the underlying disk file.

The methods for `File` objects are:

- Constructors
- Test methods
- Action methods
- List methods

24.3 Constructors

Constructors allow Java code to specify the initial values of an object. So when you're using constructors, initialization becomes part of the object creation step. Constructors for the `File` class are:

- `File(String filename)`
- `File(String pathname, String filename)`
- `File(File directory, String filename)`

24.4 Test Methods

Public methods in the `File` class perform tests on the specified file. For Example

- The `exists()` method asks if the file actually exists.
- The `canRead()` method asks if the file is readable.

- The `canWrite()` method asks if the file can be written to.
- The `isFile()` method asks if it is a file (as opposed to a directory).
- The `isDirectory()` method asks if it is a directory.

These methods are all of the boolean type, so they return a true or false.

24.5 Action methods

Public instance methods in the `File` class perform actions on the specified file. Let's take a look at them:

- The **`renameTo()`** method renames a file or directory.
- The **`delete()`** method deletes a file or directory.
- The **`mkdir()`** method creates a directory specified by a `File` object.
- The **`mkdirs()`** method creates all the directories and necessary parents in a `File` specification.

The return type of all these methods is boolean to indicate whether the action was successful.

24.6 List methods

The `list()` method returns the names of all entries in a directory that are not rejected by an optional `FilenameFilter`. The `list()` method returns null if the `File` is a normal file, or returns the names in the directory. The `list()` method can take a `FilenameFilter` filter and return names in a directory satisfying the filter.

FilenameFilter interface

The `FilenameFilter` interface is used to filter filenames. You simply create a class that implements the `FilenameFilter`. There are no standard `FilenameFilter` classes implemented by the Java language, but objects that implement this interface are used by the `FileDialog` class and the `list()` method in the `File` class.

The implemented `accept()` method determines if the filename in a directory should be included in a file list. It is passed the directory and a file name. The method returns true if the name should be included in the list.

File class: Example code

This example shows a file being tested for existence and a listing of the `C:\Windows` directory. The listing is performed for all files and then for files matching a filter.

Example

```

import java.io.*;
import java.util.*;
class FileClassExample
{
public static void main(String argv[])
{
File a_file = new File("test.txt");
if(a_file.canRead()) System.out.println("Can read file");
if(a_file.isFile()) System.out.println("Is a file");
File a_directory = new File("C:\\backslash, \\backslashWindows");
// red font indicates that an actual backslash n (carriage return character)
// should be inserted in the code.
if (a_directory.isDirectory())
{
System.out.println("Is a directory"); String names[] = a_directory.list(); for (int i =
0; i < names.length; i++)
{
System.out.println("Filename is " + names[i]);
}
}
System.out.println("Parent is " + a_directory.getParent());
if (a_directory.isDirectory())
{
String names[] = a_directory.list(new MyFilter());
for (int i = 0; i < names.length; i++)
{
System.out.println("Filename is " + names[i]);
}
}
}}
class MyFilter implements FilenameFilter
{
public boolean accept(File directory, String name)
{
if (name.charAt(0) == 'A' || name.charAt(0) == 'a')
return true;
}
}
}

```

FileInputStream and FileOutputStream

You can open a file for input or output.

`FileInputStream` reads from a disk file. You can pass that constructor either the name of a file or a `File` object that represents the file. The `FileInputStream` object is a source of data.

`FileOutputStream` writes to a disk file. You can pass it a `File` object or a name. The `FileOutputStream` object is a sink for data. For `FileInputStream` and `FileOutputStream`, you read and write bytes of data.

File: Example code

This example works like the previous examples, except the output is to a file. We open the file and write the data as bytes. After closing the file, we open it for reading, read all the bytes in the file, and print them as a string.

Example

```
import java.io.*;
import java.util.*;
class FileExample
{
    static BufferedReader system_in = new BufferedReader
(new InputStreamReader(System.in));
    public static void main(String argv[])
    {
        // Create it
        {
            try
            {
                FileOutputStream fos = new FileOutputStream("file.dat");
                // Read in three hotels
                for (int i = 0; i < 3; i++)
                {
                    Hotel a_hotel = new Hotel(); a_hotel.input(system_in); a_hotel.write_to_fos(fos);
                }
                fos.close();
            }
            catch(IOException e)
```

```

{
System.out.println(e);
}
}
// Now display it
{
byte [] buffer = null;
File a_file = new File("file.dat"); System.out.println("Length is " + a_file.length());
System.out.println(" Can read " + a_file.canRead()); try
{
FileInputStream fis = new FileInputStream(a_file);
int length = (int) a_file.length();
buffer = new byte[length]; fis.read(buffer); fis.close();
}
catch(IOException e)
{ System.out.println(e);
}
String s = new String(buffer); System.out.println("Buffer is " + s);
}
}
}
class Hotel
{
private String name; private int rooms; private String location;
boolean input(BufferedReader in)
{
try
{
System.out.println("Name: ");
name = in.readLine(); System.out.println("Rooms:"); String temp = in.readLine();
rooms = to_int(temp);
System.out.println("Location: ");
location = in.readLine();
}
catch(IOException e)
{ System.err.println(e);return false;
}
return true;
}

```

```

}
boolean write_to_fos(FileOutputStream fos)
{
try
{ fos.write(name.getBytes()); Integer i = new Integer(rooms);
fos.write(i.toString().getBytes()); fos.write(location.getBytes()); fos.write(' ');
}
catch (IOException e)
{
System.err.println(e);
return false;
}
return true;
}
void debug_print()
{
System.out.println("Name : " + name +
": Rooms : " + rooms + ": at : " + location + ":");
}
static int to_int(String value)
{
int i = 0;
try
{
i = Integer.parseInt(value);
}
catch(NumberFormatException e)
{}
return i;
}
}

```

FileReader and FileWriter

FileReader is a convenience subclass of InputStreamReader. FileReader is useful when you want to read characters from a file. The constructors of FileReader assume a default character encoding and buffer size. FileReader constructors use the functionality of InputStreamReader to convert bytes using the local encoding to the Unicode characters used by Java code.

If you want to read Unicode characters from a file that uses encoding other than the default, you must construct your own `InputStreamReader` on `FileInputStream`.

`FileWriter` is a convenience subclass of `OutputStreamWriter` for writing character files. Constructors for `FileWriter` also assume default encoding and buffer size. If you want to use encoding other than the default, you must create your own `OutputStreamWriter` on `FileOutputStream`.

Chapter 25

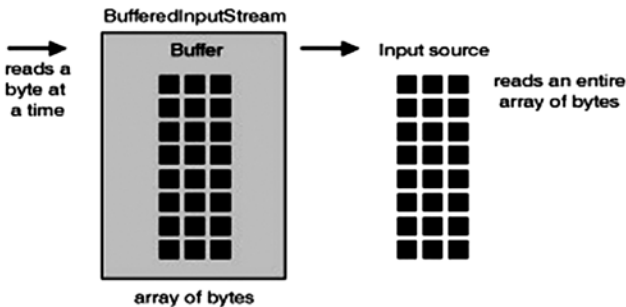
Buffering

25.1 What is buffering?

Reading and writing to a file, getting data one byte at a time, is slow and painstaking. One way to speed up the process is to put a wrapper around the file and put a buffer on it.

Our `BufferedInputStream` is going to put a buffer onto an `InputStream` that is specified in the constructor. The actual data source is what you pass it as an `InputStream`. The `BufferedInputStream` reads large chunks of data from the `InputStream`. Then you read individual bytes or small chunks of bytes from the `BufferedInputStream`. The default buffer size is 512 bytes, but there's a constructor that allows you to specify the buffer size if you want something different.

To improve your efficiency, you read from the object of `BufferedInputStream` instead of reading directly from the underlying `InputStream`. And you won't have to go back to the operating system to read individual bytes.



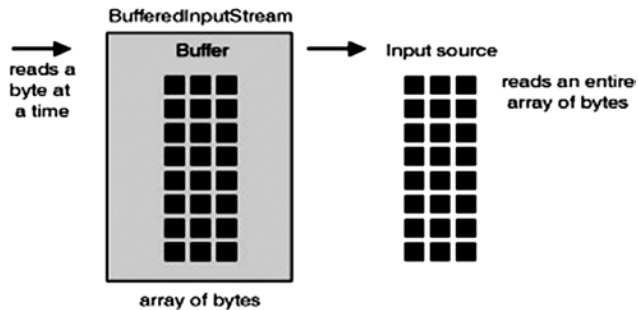
BufferedOutputStream

`BufferedOutputStream` extends `FilterOutputStream`. When you apply it to an `OutputStream`, you have a buffered output stream. Instead of going to the operating system for every byte you write, you have the intermediary that provides a buffer and you write to that.

When that buffer is full, it is written all at once to the operating system. And it is written automatically if the buffer gets full, if the stream is full, or if the `flush()` method is used. The `flush()` method forces any output buffered by the stream to be written to its destination. So for a `BufferedOutputStream`, you have to tell it:

- The output stream you are going to use

- The buffer size if you don't like the default



25.2 BufferedReader and BufferedWriter

A `BufferedReader` and a `BufferedWriter` act like `BufferedOutputStream` and `BufferedInputStream`, except they deal with reading and writing characters. For a `BufferedReader`, you specify an underlying `Reader` and optionally a buffer size. For a `BufferedWriter`, you specify an underlying `Writer` and optionally a buffer size.

`BufferedReader` has one additional method, called `readLine()`, which allows us to simply read an entire line of characters from the underlying `Reader`.

BufferedReader: Example code

If you've been using Java 1.0, the `readLine()` method was actually part of `DataInputStream`. Now you should be using a `BufferedReader` for `readLine()`, even though you can still do that with a `DataInputStream`.

A `DataInputStream` reads bytes but you are really reading characters when you read lines, so using `readLine()` and the `BufferedReader` is the preferred way.

Example

```
import java.io.*;
import java.util.*;
class TextReaderWriterExample
{
    static BufferedReader system_in = new BufferedReader
    (new InputStreamReader(System.in));
    public static void main(String argv[])
    {
        // Create it
        {
            try
```

```

{
    FileOutputStream fos = new FileOutputStream("text.dat");
    PrintWriter pw = new
    PrintWriter(fos);
    for (int i = 0; i < 3; i++)
    {
        Hotel a_hotel = new Hotel();
        a_hotel.input(system_in);
        a_hotel.write_to_pw(pw);
    }
    pw.close();
}
catch(IOException e)
{
    System.err.println(e);
}
}
// Now read it
{
    try
    {
        FileReader fr=new FileReader("text.dat");
        BufferedReader br = new BufferedReader(fr);
        Hotel a_hotel = new Hotel();
        while (a_hotel.read_from_br(br))
        {
            a_hotel.debug_print();
        }
        br.close();
    }
    catch(IOException e)
    { System.err.println(e);
    }
    }}}

class Hotel
{
    private String name;
    private int rooms;
    private String location;
    boolean input(BufferedReader in)
    {
        try
        {

```

```

System.out.println("Name: "); name = in.readLine(); System.out.println("Rooms:
"); String temp = in.readLine(); rooms = to_int(temp);
System.out.println("Location: ");
location = in.readLine();
}
catch(IOException e)
{ System.err.println(e); return false;
}
return true;
}
boolean write_to_pw(PrintWriter pw)
{ pw.print(name); pw.print(' '); pw.print(rooms); pw.print(' '); pw.print(location);
pw.println();
return true;
}
boolean read_from_br(BufferedReader br)
{
try
{
String line = br.readLine();
if (line == null)
return false;
StringTokenizer st = new StringTokenizer(line, "|");
int count = st.countTokens();
if (count < 3)
return false;
name = st.nextToken();
String temp = st.nextToken();
rooms = to_int(temp);
location = st.nextToken();
}
catch(IOException e)
{
System.err.println(e);
return false;
}
return true;
}

```

```

void debug_print()
{
    System.out.println("Name : " + name +
        ": Rooms : " + rooms + ": at : " + location + ":");
}

static int to_int(String value)
{
    int i = 0;
    try
    {
        i = Integer.parseInt(value);
    }
    catch(NumberFormatException e)
    {}
    return i;
}

```

BufferedWriter

As we mentioned, a `BufferedWriter` allows us to write to a buffer. It applies buffering to a character output stream, improving output efficiency by combining many small write requests into a single larger request.

This class has an interesting additional method that makes Java code portable. Most of the time, a new line is represented by a `\n`, but it may not be in all operating systems. Because of this, the Java language adds a method called `newLine()`.

Instead of outputting the character `\n`, you call `newLine()` and that outputs the appropriate new-line character for the particular operating system that you are running on. In most cases, this will be `\n`. The new-line character that is written is the one returned by passing `line.separator` to the `getProperty()` method in the `System` class.

Expressing a new line

So you may ask, how does the Java language know how to express a new line for a particular operating system?

The Java `newLine()` method asks the system, "What is your new-line character?" In the Java language, this is called a system property. There is a `System` class with properties. When you say, "What's your new-line character?" by passing `line.separator` to the `getProperty()` method in the `System` class, you get an answer

back. Depending on the platform, the new-line character can be a new-line character, a carriage-return character, or both.

The `newLine()` method, which is part of `BufferedWriter`, outputs the platform-dependent line separator to the stream by using such a call.

When to use `BufferedWriter`

You typically use a `BufferedWriter` for your output. The only time you don't want to have a buffered output is if you are writing out a prompt for the user. The prompt would not come up until the buffer was full, so for those sorts of things you do not want to use buffering.

`PrintWriter()` API:

```
PrintWriter out = new
```

```
PrintWriter (new BufferedWriter
```

```
(new FileWriter ("file.out"));
```

Buffering input and output streams: Example code

Here's an example of some code that shows the efficiency of buffering. Typically the `read()` method on `InputStreamReader` or `FileReader` performs a read from the underlying stream.

It might be more efficient to wrap a `BufferedReader` around these Readers. With buffering, the conversion from byte to character is done with fewer method invocations than if the `InputStreamReader read()` method is called directly.

Likewise, the `write()` method on an `OutputStreamWriter` or `FileWriter` performs a write to the underlying stream. It can be more efficient to wrap a `BufferedWriter` around these Writers.

Example

```
BufferedReader in=new BufferedReader(newInputStreamReader(System.in));
Writer out=new BufferedWriter(new FileWriter("file.out"));
```

Chapter 26

Filtering

26.1 What is filtering?

Use filtering to read or write a subset of data from a much larger input or output stream. The filtering can be independent of the data format (for example, you need to count the number of items in the list) or can be directly related to the data format (for example, you need to get all data in a certain row of a table). You attach a filter stream to an input or output stream to filter the data.

`FilterInputStream` and `FilterOutputStream` filter input and output bytes. When a `FilterInputStream` is created, an `InputStream` is specified for it to filter. Similarly, you specify an `OutputStream` to be filtered when you create a `FilterOutputStream`. They wrap around existing `InputStream` or `OutputStream` to provide buffering, checksumming, and so on.

For character I/O, we have the abstract classes `FilterWriter` and `FilterReader`. `FilterReader` acts as a superclass for character input streams that read data from some other character input stream, filter it, and then return the filtered data when its own `read()` methods are called. `FilterWriter` is for character output streams that filter data written to them before writing it to some other character output stream.

Pushback APIs

```
PushbackInputStream(InputStream is)
void unread(int byte)
void unread(byte [] array)
PushbackReader(Reader in) void unread(int character) void unread(char [] buffer)
```

26.2 Pushback

You use `PushbackInputStream` to implement a one-byte pushback buffer or a pushback buffer of a specified length. It can be used to parse data. When would you use this?

Sometimes when you read a byte from an `InputStream`, it signifies the start of a new block of data (for example, a new token). You might say, “I have read this byte, but I really cannot deal with it at this time. I want to push it back on the `InputStream`. When I come back later to read the `InputStream`, I want to get that

same byte again.”

You push the byte back onto the stream with the `unread()` method. When you read it again, you will read the data you unread before. You can `unread()` a single byte, an array of bytes, or an array of bytes with an offset and a length.

If you are reading characters, you can push back a character instead of a byte. You use `PushbackReader`, which works just like the `PushbackInputStream`.

Use the `unread()` methods to unread a single character, an array of characters, or an array of characters with an offset and a length.

`LineNumberReader` versus `LineNumberInputStream`.

A `LineNumberReader` is a character input stream that keeps track of the number of lines of text that have been read from it. A line is considered terminated by a new line (`\n`), a carriage return, or a carriage return followed by a linefeed. The `readLine()` method returns all the characters in the line without the terminator. The `getLineNumber()` method returns the number of lines that have been read.

If, for some reason, you want to reset the line number, you can do that with `setLineNumber()` and start the line count again from that point.

`LineNumberInputStream` also keeps track of the number of lines of data that have been read, but this class was deprecated in Java 1.1 because it reads bytes rather than characters. `LineNumberReader` was introduced, which reads characters.

LineNumberReader APIs

```
LineNumberReader(Reader r)
LineNumberReader(Reader r,int buffer_size)
String readLine() int getLineNumber()
void setLineNumber(int line_num)
```

PrintStream versus PrintWriter

`PrintStream` is used to output Java data as text. It implements a number of methods for displaying textual representation of Java primitive data types.

You’ve probably used a `PrintStream` object since your earliest Java programming days. `System.out` is a `PrintStream` object. Probably everyone has used that for debugging! That being said, `PrintStream` has been superseded by `PrintWriter` in Java 1.1. The constructors of this class have been deprecated, but the class itself has not because it is still used by the `System.out` and `System.err` standard output streams.

The methods in `PrintStream` and `PrintWriter` are very similar. `PrintWriter` does not

have methods for writing raw bytes, which `PrintStream` has.

Flushing

Both the `PrintStream` and `PrintWriter` classes employ flushing, which moves data from the internal buffers to the output stream. Flushing is employed in two ways:

- Automatic flushing, which says when you call `println` you get flushing
- Without automatic flushing, which says flushing occurs only when the `flush()` method is called.

With the `PrintStream` class, if a new-line character was written, the output is flushed. With the `PrintWriter` class, if you enable automatic flushing, the output is flushed only when a `println()` method is invoked. The `println()` methods of `PrintWriter` use the System property `line.separator` instead of the new-line (`\n`) character that `PrintStream` uses.

PrintWriter process

You can give the `PrintWriter` constructor a `Writer` or you can give it an `OutputStream`. With the latter, an `OutputStreamWriter` is transparently created, which performs the

conversion for characters to bytes using the default character encoding.

You can choose one of the two flushing options: with `autoflush` or without. You can print integers, strings, and characters without a new line with the `print()` methods and with a new line with the `println()` methods.

PrintWriter() APIs

```
PrintWriter (Writer w)
PrintWriter (Writer w, Boolean autoflush)
PrintWriter (OutputStream os)
PrintWriter (OutputStream os, Boolean autoflush)
```

26.3 Checksumming

This section explores checksumming. `Checksum` is an interface that several different classes implement. Let's take a look at what this section covers:

- What is a checksum and how does it work?
- What is a message digest and how does it work?
- Checksum versus message digest

What is checksumming?

What's a checksum? The Checksum interface defines the methods required to compute a checksum on a set of data. The computation of a checksum is such that if a single bit or two bits in the data are changed, the value for the checksum changes.

A checksum is computed based on the bytes of data supplied to the `update()` method. The `update()` method updates the current checksum with an array of bytes by adding it to an ultimate value.

Additionally, the current value of the checksum can be obtained at any time with the `getValue()` method. Use the `reset()` method to reset the default value to its initial value, usually 0.

There are two types of checksums. The most common is the CRC32, that's Cycle Redundancy Check. Another type of checksum is the Adler32, used for computing Adler32 checksum. The Adler32 has a faster computation. It's nearly as reliable as the CRC32.

Checksum methods

```
long getValue()
void reset()
void update(byte [] array, in offset, int length)
```

CheckedInputStream and CheckedExceptionStream

Let's create a Checksum object and use it to filter an input or output stream. When writing a byte to the CheckedExceptionStream, the checksum is automatically computed.

CheckedExceptionStream calls the `update()` method in the Checksum object. Then the byte goes out to its ultimate destination. At any point in time, you can ask the Checksum object for the current checksum. You can write all your bytes out and add that checksum to what you have written out.

- CheckedExceptionStream implements FilterInputStream and maintains a checksum on every read.
- CheckedExceptionStream implements FilterOutputStream and maintains a checksum on every write.

Adding checksum

Use a checksum to make sure that data was transmitted properly. Construct a CheckedExceptionStream using an OutputStream and an object of the Checksum

type. CRC32 and Adler32 are the two types of checksums you can choose from or you can create your own. At periodic times, say every 512 bytes, retrieve the current value of the checksum and send those four bytes over the stream.

On the receiving end, you construct a `CheckedInputStream` using an `InputStream` and an object of the same type you used for the `CheckedOutputStream`. After reading 512 bytes, retrieve the current value of the checksum and compare it to the next four bytes read from the stream.

CheckedOutputStream: Example code

Let's look at some sample code. We are going to open a `FileOutputStream`, `Temp1.tmp`. An object of the `CRC32` type is constructed, and that is used to create a `CheckedOutputStream`. When writing to the file output a byte at a time, the checksum is computed. At the end, the `Checksum` object is then asked the current value of the checksum.

Checksumming only gives you a 32-bit value. If a single bit changes, you will notice it, but if lots of bits change, you could possibly get the same checksum.

Here is the example code:

```
FileOutputStream os = new FileOutputStream("Temp1.tmp");
CRC32 crc32 = new CRC32(); CheckedOutputStream cos = new
CheckedOutputStream(os, crc32);
cos.write(1);
cos.write(2);
long crc = crc32.getValue();
```

26.4 Digesting

Let's create something larger than a Checksum. A larger set of bits that represents a sequence of bytes is a message digest. A message digest works like a checksum. It is a one-way hash that takes a variable amount of data and creates a fixed-length hash value. This is called a digital fingerprint.

If somebody makes changes in the original sequence of things and manipulates the contents, the message digest is not going to look the same. This is how we implement security. There is a possibility, but it is very small, that if the message were changed, you would get the same message digest.

A sequence of bytes is transformed into a digest. In the `MessageDigest` class, we have a static function called `getInstance`. `getInstance` is given the name of the `MessageDigest` that we want to be using. There are two strings that you can pass

it: SHA1 and MD5. Their details are in algorithmic books.

Additionally, the `update()` methods update the digest with the byte or array of bytes. The `digest()` method computes and returns the value of digest. Digest is then reset.

Digesting methods

```
static MessageDigest getInstance(String algorithm)
```

```
void update (byte [] array)
```

```
byte [] digest()
```

DigestInputStream and DigestOutputStream

We have two equivalent methods for digest: the `DigestInputStream` and `DigestOutputStream`. Give the `DigestInputStream` an input stream and the `MessageDigest` that will be doing its calculations. You can turn digesting on or off for a stream. You may want to input or output some data, but not calculate it as part of the `MessageDigest`.

The `read()` and `write()` methods update the `MessageDigest` in `DigestInputStream` and `DigestOutputStream` respectively.

MessageDigest: Example code

The first line creates a `FileOutputStream` going to the file `temp.tmp`. Create a `MessageDigest` object. As you can see in the code `md = MessageDigest.getInstance()`, you're passing it the name of the message digesting algorithm that you want to use. It returns back to you an object of a `MessageDigest` type. If the string that you passed it is not the name of a digest, it throws a `NoSuchAlgorithmException`.

After you have the `MessageDigest`, use that to construct a `DigestOutputStream`. Bytes are written out to the underlying `OutputStream` and added to the digest. You can get `MessageDigest` by calling `md.digest()`. It returns the digest or an array of bytes. Now you can store that away or send it along.

Example

```
FileOutputStream os = new FileOutputStream("Temp.tmp"); MessageDigest md
= null;
try
{
md = MessageDigest.getInstance("SHA");
```

```

}
catch(NoSuchAlgorithmException e)
{
    System.out.println(e);
}
DigestOutputStream dos = new DigestOutputStream(os, md);
dos.write(1);
dos.write(2);
byte [] digest = md.digest();

```

26.5 Checksum versus MessageDigest

Remember, a checksum is small. A checksum is typically used to verify the integrity of data over a noisy transmission line or to verify whether a file contains the same data. A message digest, on the other hand, is much larger. It is used more for security to insure that a message has not been tampered with.

26.6 Inflating and deflating

This section explores inflating and deflating in Java code. These terms have the same meaning as compressing and uncompressing. Some of the key points in this section are:

- DeflaterInputStream and DeflaterOutputStream
- ZipInputStream and ZipOutputStream
- ZipFile

Deflater

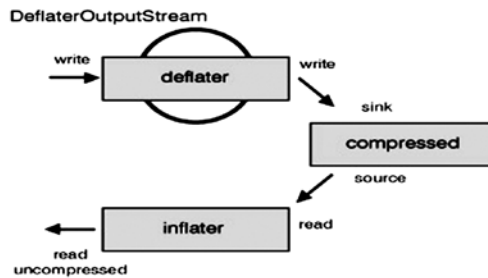
Let's move into deflating territory. Deflating writes a compressed representation of bytes to an output stream. Inflating takes those compressed bytes from an input stream, reads them in, and automatically uncompresses them. This process is similar to zipping and unzipping on the fly.

The algorithm of a Deflater is applied to the bytes output to a stream, and the corresponding algorithm of an Inflater is applied to the bytes input from a stream. The default Deflater and Inflater use the ZLIB compression library for compression and decompression. An alternate constructor allows for GZIP and PKZIP compression.

DeflaterOutputStream

Let's journey into the output stream. You can simply create a DeflaterOutputStream with an output sink. As we write to the DeflaterOutputStream, the bytes are compressed using the Deflater algorithm and only the compressed bytes are sent to the underlying output stream. The default constructor for DeflaterOutputStream uses ZLIB

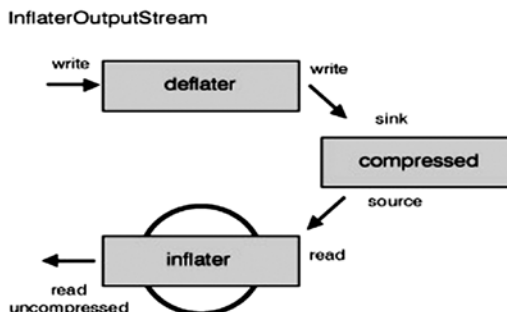
compression with a default-sized buffer. With the other constructors, you can specify a different Deflater and a different-sized buffer.



InflaterInputStream

On the receiving side, an InflaterInputStream is implemented. When we construct it, we specify an input source. It uncompresses the bytes from that source as you go along. Once again, you can specify your own Inflater if you use a different compression scheme.

If you decide to use your own Deflater or Inflater instead of the default, then it is up to you to make sure the Inflater and Deflater match and understand each other.



GZIP input and output stream

A `GZIPOutputStream` outputs the compressed bytes to the underlying stream in GZIP format. It is derived from the `DeflaterOutputStream`. The constructor is supplied an `OutputStream`. Correspondingly, a `GZIPInputStream` inputs from a source that is in GZIP format. It is derived from `InflaterInputStream`, and the constructor is supplied an `InputStream`.

Use `GZIPInputStream` and `GZIPOutputStream` on a single stream, which could be a file. However, it can be more efficient than a zip stream because it does not require creating a directory of zipped files.

ZipFile: Example code

Let's look into the `ZipFile` and `ZipEntry` classes. These are not actually in the I/O hierarchy but they are good to know about because lots of people have zip files.

The `ZipFile` class allows us to read a zip file. A zip file consists of two parts: a list of entries of what's been zipped and the data itself. You read the set of entries using `entries()` and pick out a particular one to unzip. A `ZipEntry` object represents the file you have selected

to uncompress. For that particular `ZipEntry`, you can obtain an `InputStream` using `getInputStream()` which, when read, returns the uncompressed data. The Java library is kept in a jar file, which is in zip file format. You can use a `ZipFile` to read a jar file if you want.

Example

```
import java.util.zip.*; import java.util.*; import java.io.*;
class ZipFileExample
{
    public static void main(String argv[]){
        try{
            ZipFile zf = new ZipFile("test.zip");
            // Enumerate the entries
            Enumeration zip_entries = zf.entries(); String last_name = null; while(zip_entries.hasMoreElements()){
                ZipEntry ze = (ZipEntry) zip_entries.nextElement(); System.out.println("Entry " + ze.getName()); last_name = ze.getName();
            }
            // Lets unpack the last one as an example
```

```
// Its name is in last_name
ZipEntry ze = zf.getEntry(last_name);
if (ze == null)
    System.out.println("Cannot find entry");
else{
    BufferedReader r =new    BufferedReader(new    InputStreamReader(zf.
    getInputStream(ze)));
    long size = ze.getSize();
    if (size > 0){
        System.out.println("File is " + size);
        String line = r.readLine();
        if (line != null)
            System.out.println("First line is " + line);
    }
    r.close();
}}catch(IOException e){
    System.out.println(e);
}}
```

26.7 ZipInputStream

ZipInputStream is our next stop. You can read a zip file in a stream using this class. You get the first entry in the directory of what has been zipped. You can then read the data for the first entry, which is automatically unzipped. Then you get the next entry and unzip it, and so forth. ZipInputStream extends InflaterInputStream. The methods include getting the next entry from the table and closing off the entry. Closing the entry allows for the next entry to be read without having to finish reading the data for the current entry. The read() method in ZipInputStream will return -1 when the last of the entries is read.

ZipInputStream methods

```
getNextEntry()
closeEntry()
read()
```

ZipInputStream: Example code

The code below is an example of unzipping a file. You are going to open test.zip, which is our zip file. That file is the source of a ZipInputStream. So zis is pointing to test.zip. The getNextEntry() method initially returns the first entry from the zip

file table. A reference to that entry is now available.

If the entry is null, it means that end-of-file has been reached. Reading from the zip file from

this point, data that is read corresponds to that entry and is automatically uncompressed. In this case, a `DataInputStream` has been wrapped and used to read it. The `dis` object is just going to read from the corresponding place in the zip file and close it. The next entry will open the file and start reading from the zip file again, reading the next entry.

Example

```
import java.util.zip.*;
import java.io.*;

class ZipInputStreamExample{
    public static void main(String argv[]){
        try{
            FileInputStream fis = new FileInputStream("test.zip");
            ZipInputStream zis = new ZipInputStream(fis);
            ZipEntry ze;
            // Get the first entry
            ze = zis.getNextEntry();
            if (ze == null)
                System.out.println("End entries");
            else{
                // Use it as stream or reader
                DataInputStream dis = new DataInputStream(zis);
                int i = dis.readInt();
                System.out.println("File contains " + i);
                zis.closeEntry();
            }
            zis.close();
        }catch(Exception e){
            System.out.println(e);
        }
    }
}
```

26.8 ZipOutputStream

A `ZipInputStream` reads a zip-formatted stream; a `ZipOutputStream` writes zip-formatted

streams. This class allows us to create a zip file. `ZipOutputStream` extends `DeflaterOutputStream`. One method in `ZipOutputStream` is `putNextEntry()`, which puts the next entry into the list of compressed files. You need to create a zip entry for each file you are compressing.

The `putNextEntry()` method writes an entry and positions the stream for data output. You can start writing the next set of compressed data. Similar to the `close()` methods in other classes, the `closeEntry()` method closes a finished entry; you are ready to create the next one.

ZipOutputStream methods

`putNextEntry()`

`closeEntry()`

`write()`

`ZipOutputStream`: Example code

In the code below, an output file is constructed called `test.zip`. A `ZipOutputStream` is then constructed using that file. We want to compress the file `test.txt` onto the zip file, so we construct a `ZipEntry` using that name. `test.txt` is going to show up in the directory for the zip file. At this point, anything written to the `ZipOutputStream` is now data for that entry and is written in compressed form.

A `DataOutputStream` is wrapped around `zos`. Then we write an integer, and the entry is closed. At this point, another `ZipEntry` could now be created and written out. You could use `WinZip` to decompress this file or use the preceding example to read the file.

Example

```
import java.util.zip.*;
import java.io.*;
class ZipOutputStreamExample{
public static void main(String argv[]){
try{
FileOutputStream fos = new FileOutputStream("test.zip");
ZipOutputStream zos = new ZipOutputStream(fos);
ZipEntry ze = new ZipEntry("test.txt");
zos.putNextEntry(ze);
// Use it as output
```

```
DataOutputStream dos = new DataOutputStream(zos);  
dos.writeInt (1);  
zos.closeEntry();  
// Put another entry or close it zos.close();  
}  
catch(Exception e){  
System.out.println(e);  
}}}
```

Chapter 27

Data I/O Introduction

27.1 What is Data I/O?

Several classes implement the `DataInput` and `DataOutput` interfaces. Let's talk about how they work.

The primitive data types, such as ints, may be represented differently on the various platforms that Java code runs on. For example, an int may be represented with the most significant byte first (called Big Endian), as on an IBM mainframe, or with the least significant byte first (called Little Endian), as on an IBM PC. If you wrote out an int in binary form with

a C program on one platform and tried to read it in on the other platform, the values would not agree.

That's where `DataInput` and `DataOutput` come in. Now you can take a primitive, like an int, and write it out. It is written in a platform-independent form. You can read that int back in on any other Java virtual machine and, regardless of where it was produced, it will be returned back into an int of the proper value.

All the primitives, such as doubles, ints, shorts, longs, and so forth, can be written in a system platform-independent manner using the methods in `DataOutput`. Likewise, they can be read using the methods in `DataInput`.

27.2 Unicode Text Format

A `String` object is written in Unicode Text Format or UTF for short. Strings are composed of two-byte Unicode characters. UTF is a method for compressing the most common Unicode values. If the `String` contains just ASCII characters, the values between 1 and 127 are written as a single byte. Values 128 - 2047 are written as two bytes. For some uncommon Unicode values, three bytes are used to represent their value. This expansion of some values is much rarer than the compression of ASCII values, so using UTF to represent a `String` is a net gain.

Unicode methods

```
void writeUTF(String s)
String readUTF()
```

`DataInputStream` and `DataOutputStream`: Example code

Two classes, `DataInputStream` and `DataOutputStream`, implement `DataInput` and `DataOutput`. Their methods include implementations for `readInt()`, `writeInt()`, and the other methods in the interfaces. You wrapper a `DataInputStream` around an `InputStream`. When `readInt()` is called, four bytes are read from the underlying stream and the resulting `int` is returned. Correspondingly, you wrapper a `DataOutputStream` around an `OutputStream`.

The code example below shows a `DataOutputStream` that uses a `FileOutputStream` as its underlying stream. The file is opened using a `FileInputStream`, which has a `DataInputStream` wrapped around it. The `data.dat` could be written on a PC.

Example

```
import java.io.*;

class DataInputOutputExample{
    static    BufferedReader    system_in    =    new    BufferedReader(new
    InputStreamReader(System.in));
    public static void main(String argv[]){
        // Create it
        {
            try{
                FileOutputStream fos = new FileOutputStream("data.dat");
                DataOutputStream dos = new DataOutputStream(fos);
                // Read in three hotels
                for (int i = 0; i < 3; i++){
                    Hotel a_hotel = new Hotel();
                    a_hotel.input(system_in);
                    a_hotel.write_to_dos(dos);
                }
                dos.close();
            }
            catch(IOException e){
                System.err.println(e);
            }
        }
        // Now read it
        {
            try{
                FileInputStream fis = new FileInputStream("data.dat");
                DataInputStream dis = new DataInputStream(fis);
```

```

Hotel a_hotel = new Hotel();
while (a_hotel.read_from_dis(dis)){
a_hotel.debug_print();
}
dis.close();
}
catch(IOException e){
System.err.println(e);
}}}}
class Hotel{
private String name;
private int rooms;
private String location;
boolean input(BufferedReader in){
try{
System.out.println("Name: ");
name = in.readLine();
System.out.println("Rooms: ");
String temp = in.readLine();
rooms = to_int(temp);
System.out.println("Location: ");
location = in.readLine();
}
catch(IOException e)
{
System.err.println(e);return false;
}
return true;
}
boolean write_to_dos(DataOutputStream dos){
try
{ dos.writeUTF(name);
dos.writeInt(rooms);
dos.writeUTF(location);
}
catch(IOException e){
System.err.println(e);
return false;
}
}
}

```

```

}
return true;
}

boolean read_from_dis(DataInputStream dis){
try{
name = dis.readUTF();
rooms = dis.readInt();
location = dis.readUTF();
}
catch(EOFException e){
return false;
}
catch(IOException e){
System.err.println(e);
return false;
}
return true;
}

void debug_print(){
System.out.println("Name :"+ name +": Rooms : " + rooms + ": at :"+ location+
":"");
}

static int to_int(String value)
{
int i = 0;
try
{
i = Integer.parseInt(value);
}
catch(NumberFormatException e){}
return i;
}
}

```

27.3 RandomAccessFile: Example code

RandomAccessFile implements both DataInput and DataOutput. You construct

RandomAccessFile with a file name. Then you can use both read and write methods, as readInt() and writeInt(), on a RandomAccessFile.

Random access means you can position the next read or write to occur at a particular byte position within the file. You can obtain the current position with getFilePointer(). Later on, you can go back to that same position by passing it to seek().

The example code below shows writing three ints to a file. The file position before the second write is stored in the pointer. After writing the next two ints, the position is restored to that pointer. The readInt() returns the value 15.

Example

```
import java.io.*;
class RandomAccessFileExample{
public static void main(String argv[]){
try{
RandomAccessFile raf =new RandomAccessFile("random.dat", "rw");
raf.writeInt(12);
long pointer = raf.getFilePointer();
raf.writeInt(15);
raf.writeInt(16);
// Now read back the 2nd one raf.seek(pointer);
int i = raf.readInt(); System.out.println("Read " + i);
}
catch (IOException e)
{ System.out.println(e);}
}
}
```

Chapter 28

Object Serialization

28.1 What is object serialization?

Object serialization lets you take all the data attributes, write them out as an object, and read them back in as an object. Object serialization is quite useful when you need to use object persistence. GUI builders of JavaBeans use serialization to store the attributes of a JavaBean so that it can be accessed or modified by others.

You could actually use `DataOutputStreams` and `DataInputStreams` to write each attribute out individually, and then you could read them back in on the other end. But you want to deal with the entire object, not its individual attributes. You want to be able to simply store away an object or send it over a stream of objects. Object serialization takes the data members of an object and stores them away or retrieves them, or sends them over a stream.

You have the `ObjectInput` interface, which extends the `DataInput` interface, and `ObjectOutput` interface, which extends `DataOutput`, so you are still going to have the methods `readInt()` and `writeInt()` and so forth. `ObjectInputStream`, which implements `ObjectInput`, is going to read what `ObjectOutputStream` produces.

28.2 How object serialization works

If you want to use the `ObjectInput` and `ObjectOutput` interface, the class must be serializable. How is a class serializable?

The serializable characteristic is assigned when a class is first defined. Your class must implement the `Serializable` interface. This marker interface says to the Java virtual machine that you want to allow this class to be serializable. You don't have to add any additional methods or anything.

There are a couple of other features of a serializable class. First, it has to have a zero parameter constructor. When you read the object, it needs to be able to construct and allocate memory for an object, and it is going to fill in that memory from what it has read from the serial stream.

The static fields, or class attributes, are not saved because they are not part of an object. If you do not want a data attribute to be serialized, you can make it `transient`. That would save on the amount of storage or transmission required to transmit an object.

Using ObjectOutputStream: Example code

A FileOutputStream is created using object.dat, which is going to be used to store away objects.

You create ObjectOutputStream using that file. Any other OutputStream, such as that returned by getOutputStream() for a socket, could have been used. And now you have YourClass. This is whatever class you have made up. You create an object of that class. If you want to store that away on the ObjectOutputStream, you simply call writeObject() and pass it the object. That's it!

The class could have 20 attributes in it or 50 attributes; it doesn't matter. All of those attributes are automatically stored away. The class information is stored automatically.

The Java code knows what's in a class and knows the attributes in a class. In fact, the first time the code stores such an object, it stores a couple of markers about what the class is. Calling writeObject() is all that is necessary to store an object.

Here is the example code:

```
FileOutputStream fos = new FileOutputStream("object.dat");
ObjectOutputStream oos = new ObjectOutputStream(fos);
YourClass yc = new YourClass();
oos.writeObject(yc);
```

28.3 Reading an object from a file: Example code

Now let's read that object from the file. Create a FileInputStream using the file object.dat that was written in the previous panel. That file is used as the source for an ObjectInputStream. You want to read the object, so we call readObject() and the object is read off the file. The attributes of the YourClass object are filled in with the data we have stored away.

The readObject() method returns a reference to an Object class object. You wrote out a YourClass object, so you have to cast the reference to YourClass. You have to keep track of the order in which you wrote things out. If you stored away a YourClass object, and then you stored away an AnotherClass object, you have to read them back in the same sequence.

Typically, you would write things out in a sequence, and then you would read them back in the same sequence. However, there are some alternative approaches, discussed in the next panel.

Example

```
FileInputStream fis = new FileInputStream ("object.dat"); ObjectInputStream ois
= new ObjectInputStream (fis);
YourClass yc;
yc = (YourClass) ois.readObject();
```

28.4 Alternative ways to read back: Example code

If you do not know the order in which the objects were stored, you can test for the class of the object that was returned by `readObject()` using `instanceof`, or you can use the `getClass()` method to determine the class of the object. Then you can assign the returned value to the appropriate object using a cast. You can do more elaborate reading and writing as you read back an object, test what it is, and then do an appropriate cast.

That's all you have to do to store away an object in a file. Now, you want to transmit it over a socket. All the values in that object are transmitted over a socket if you used the appropriate input and output streams from the socket object. Remote method invocation (RMI) passes objects using object serialization over a socket.

Example

```
FileInputStream fis = new FileInputStream("object.dat"); ObjectInputStream ois = new
ObjectInputStream(fis);
Object object;
object = ois.readObject();
if (object instanceof YourClass)
{
    YourClass yc = (YourClass) object;
    // Perform operations with yc;
}
else if (object instanceof AnotherClass)
{
    AnotherClass ac = (AnotherClass) object;
    // Perform operations with ac
}
```

28.5 Add serializable: Example code

We said before that if the base class implements `Serializable`, all you have to do is mark your derived class `Serializable`. But what if your base class doesn't

implement it? You can still implement `Serializable` in your derived class.

If the base class does not implement `Serializable` but you want to use serialization, you simply have to get all the attributes out of the base class and store them away. Your additional attributes are stored automatically, but the base class attributes are not. You have to use `get()` methods, and if the base class does not have gets and sets for its attributes, then you won't be able to serialize the base class. You need to define your own `readObject()` and `writeObject()` methods, which we describe next.

Example

```
class ABaseClass
{
    private int value;
    int getValue()
    {
        return value;
    }
    void setValue(int value)
    {
        this.value = value;
    }
}
```

```
class YourClass extends ABaseClass implements Serializable
{
    //..
}
```

28.6 Custom serialization: Example code

You may want to use custom serialization if the base class does not support serialization and for some reason you don't like the default customization. Define your own `readObject()` and `writeObject()` methods. The fields or attributes are no longer automatically serialized if you define these methods. You have to take care of saving and restoring each of your attributes.

To store away the non-transient and non-static attributes for each object, call `defaultWriteObject()`. The attributes can be loaded with `defaultReadObject()`. You can store and load additional values in your method, for instance, ones that you

have declared to be transient. If the class you derived from is serializable, then your data for the super class is already taken care of. All you have to do then is add your own custom code.

The example code below shows how `readObject()` and `writeObject()` are coded. Note that `ABaseClass` does not implement `Serializable`, but it does have `getValue()` and `setValue()`.

Example

```
class ABaseClass
{
    private int value;
    int getValue()
    {
        return value;
    }
    void setValue(int value)
    {
        this.value = value;
    }
}

class YourClass extends ABaseClass implements Serializable
{
    int another;
    private void readObject(ObjectInputStream ois)
        throws IOException, ClassNotFoundException
    {
        ois.defaultReadObject();
        setValue(ois.readInt());
    }
    private void writeObject(ObjectOutputStream oos)
        throws IOException
    {
        oos.defaultWriteObject(); oos.writeInt(getValue());
    }
}
```

28.7 Externalizable: Example code

As you become more experienced with Java programming, you may decide you would like to customize the way things are done. You want to implement your own

complete way of doing data persistence.

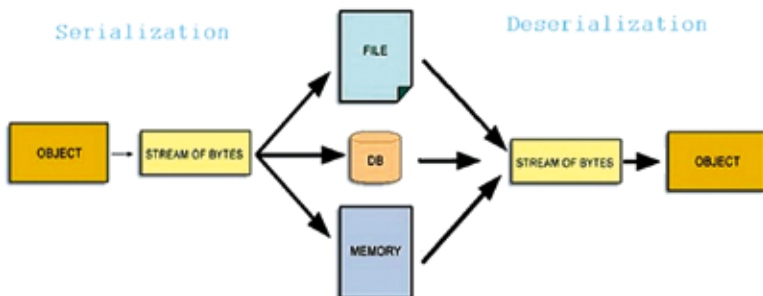
If you want to add your own custom code for storing away base class or superclass data, you must implement `Externalizable`. `Externalizable` has two methods: `readExternal()` and `writeExternal()`. In this case, you are fully responsible for everything.

Here is the example code:

```
private void readExternal
(ObjectInputStream ois) throws IOException, ClassNotFoundException
private void writeExternal(ObjectOutputStream os) throws IOException
```

Java provides `Serializable` API encapsulated under `java.io` package for serializing and deserializing objects which include,

- `java.io.Serializable`
- `java.io.Externalizable`
- `ObjectInputStream`
- and `ObjectOutputStream` etc.



Serialization

Serialization is the process of writing the state of object inside the file, text file or flag file. When you are working with any of distributed technology like Hibernate, RMI, JPA, EJB etc. Then you may get requirement to store a state of object in a file. The `String` class and all the wrapper classes implement `java.io.Serializable` interface by default. Serialization is mainly used to travel objects on the network. The serialized objects are JVM independent and can be re-serialized by any JVM. In this case the “in memory” java objects state are converted into a byte stream. This type of the file cannot be understood by the user. It is a special types of object i.e. reused by the JVM (Java Virtual Machine). This process of serializing an object is also called deflating or marshalling an object.

The given example shows the implementation of serialization to an object. This program takes a file name that is machine understandable and then serializes the object. The object to be serialized must implement `java.io.Serializable` class. Default serialization mechanism for an object writes the class of the object, the class signature, and the values of all non-transient and non-static fields. `class ObjectOutputStream` extends `java.io.OutputStream` implements `ObjectOutput`, `ObjectOutput` interface extends the `DataOutput` interface and adds methods for serializing objects and writing bytes to the file. The `ObjectOutputStream` extends `java.io.OutputStream` and implements `ObjectOutput` interface. It serializes objects, arrays, and other values to a stream. Thus the constructor of `ObjectOutputStream` is written as:

```
ObjectOutput ObjOut = new ObjectOutputStream(new FileOutputStream(f));
```

Above code has been used to create the instance of the `ObjectOutput` class with the `ObjectOutputStream()` constructor which takes the instance of the `FileOutputStream` as a parameter. The `ObjectOutput` interface is used by implementing the `ObjectOutputStream` class. The `ObjectOutputStream` is constructed to serialize the object.

`writeObject()`:

`Java.io.Serializable` is a marker interface. It is used to 'mark' java classes which support a certain eligibility or capability.

Example

```
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.ObjectOutputStream;
import java.io.Serializable;

class User{
    String name;
    long contact;
    public User(){}

    public User(String name, long contact) {
        this.name = name;
        this.contact = contact;
    }
}
```

```

class Account extends User implements Serializable{
    int acc;
    static double fee=880.998;
    transient String address;
    public Account(){ }
    public Account(String name,long contact,int acc,double fee, String address) {
        super(name,contact);
        this.acc = acc;
        //this.fee=fee;
        this.address = address;
    }
}

class Student extends Account{
    transient int pin;
    String quali;
    public Student(){ }
    Student(String name,long contact,int acc,double fee,String address,int
pin,String quali){
        super(name,contact,acc,fee,address);
        this.pin=pin;
        this.qualified=quali;
    }
    public String toString() {
        return "Name :"+name+" contact :"+contact+" Account No :"+acc+"
Fee : "+fee+" Address :"+address+" Pin :"+pin;
    }
}

public class Jtc208 {
    public static void main(String[] args) throws FileNotFoundException,
IOException {
        Student s1=new Student("Som",123,9999,2890.30,"Noida",201301,"B.E.");
        s1.fee=9999.9999;
        ObjectOutputStream oos=new ObjectOutputStream(new
FileOutputStream("abc.txt"));
        oos.writeObject(s1);
    }
}

```

De-Serialization

DeSerialization is a process of DeSerializing an object that means retrieving the object serialized in a binary format from a persistent storage and converting it to the actual object type is called as deserialization which is also called inflating or unmarshalling.. Class `ObjectInputStream` contains methods in java for deSerializing objects.

The given program shows how to read any data or contents from the serialized object or file. It takes a file name and then converts into java object. If any exception occurs during reading the serialized file, it is caught in the catch block.

`ObjectInputStream` extends `java.io.InputStream` and implements `ObjectInput` interface. It deserializes objects, arrays, and other values from an input stream. Thus the constructor of `ObjectInputStream` is written as:

```
ObjectInputStream obj = new ObjectInputStream(new FileInputStream(f));
```

Above code of the program creates the instance of the `ObjectInputStream` class to deserialize that file which had been serialized by the `ObjectInputStream` class. The above code creates the instance using the instance of the `FileInputStream` class which holds the specified file object which has to be deserialized because the `ObjectInputStream()` constructor needs the input stream.

```
readObject():
```

Method `readObject( )` reads the object and restore the state of the object. This is the method of the `ObjectInputStream` class that helps to traverse the object.

Example

```
import java.io.FileInputStream;
import java.io.ObjectInputStream;
public class Jtc209 {
    public static void main(String[] args) {
        try{
            ObjectInputStream osi=new ObjectInputStream(new FileInputStream("abc.
txt"));

            Object o=osi.readObject();
            System.out.println(o.getClass().getName());
            System.out.println(o);
        }catch(Exception e){
            e.printStackTrace();
        }
    }
}
```


Transient Modifier

- Transient is the modifier applicable only for variables but not for methods and classes.
- At the time of serialization if we don't want to serialize the value of a particular variable to meet the security constraints then we should declare that variable with transient modifier.
- At the time of serialization jvm ignores the original value of the transient variable and save default value that is transient means "not to serialize".
- Static variables are not part of object state hence serialization concept is not applicable for static variables due to this declaring a static variable as transient there is no use.
- Final variables will be participated into serialization directly by their values due to this declaring a final variable as transient as there is no impact.

Static Vs Transient

static variable is not part of object state hence they won't participate in serialization because of this declaring a static variable as transient there is no use.

Transient Vs Final

final variables will be participated into serialization directly by their values.

Hence declaring a final variable as transient there is no use.

//the compiler assign the value to final variable

28.8 Customized Serialization

During default Serialization there may be a chance of lose of information due to transient keyword. (Ex : mango ,money , box)

Example

```
import java.io.*;
class Account implements Serializable{
String userName="Som";
transient String pwd="Prakash";
}
class CustomizedSerializeDemo{
public static void main(String[] args)throws Exception{
Account a1=new Account();
System.out.println(a1.userName+"....."+a1.pwd);
```

```

FileOutputStream fos=new FileOutputStream("abc.ser");
ObjectOutputStream oos=new ObjectOutputStream(fos);
oos.writeObject(a1);
FileInputStream fis=new FileInputStream("abc.ser");
ObjectInputStream ois=new ObjectInputStream(fis);
Account a2=(Account)ois.readObject();
System.out.println(a2.userName+"....."+a2.pwd);
}}

```

In the above example before serialization Account object can provide proper username and password. But after Deserialization Account object can provide only username but not password. This is due to declaring password as transient. Hence doing default serialization there may be a chance of loss of information due to transient keyword.

We can recover this loss of information by using customized serialization.

We can implement customized serialization by using the following two methods.

`private void writeObject(ObjectOutputStream os) throws Exception.`

This method will be executed automatically by JVM at the time of serialization. It is a callback method. Hence at the time of serialization if we want to perform any extra work we have to define that in this method only.

(prepare encrypted password and write encrypted password separate to the file)

`private void readObject(ObjectInputStream is) throws Exception.`

- This method will be executed automatically by JVM at the time of Deserialization. Hence at the time of Deserialization if we want to perform any extra activity we have to define that in this method only.
- (read encrypted password , perform decryption and assign decrypted password to the current object password variable)

28.9 Serialization with respect to Inheritance

Case 1

If parent class implements Serializable then automatically every child class by default implements Serializable. That is Serializable nature is inheriting from parent to child.

Hence even though child class doesn't implement Serializable , we can serialize child class object if parent class implements serializable interface.

Example

```

import java.io.*;
class Animal implements Serializable{
int i=10;
}
class Dog extends Animal{
int j=20;
}
class SerializableWRTInheritance{
public static void main(String[] args)throws Exception{
Dog d1=new Dog();
System.out.println(d1.i+"....."+d1.j);
FileOutputStream fos=new FileOutputStream("abc.ser");
ObjectOutputStream oos=new ObjectOutputStream(fos);
oos.writeObject(d1);
FileInputStream fis=new FileInputStream("abc.ser");
ObjectInputStream ois=new ObjectInputStream(fis);
Dog d2=(Dog)ois.readObject();
System.out.println(d2.i+"....."+d2.j);
}}

```

Even though Dog class does not implements Serializable interface explicitly but we can Serialize Dog object because its parent class animal already implements Serializable interface.

Note: Object class doesn't implement Serializable interface.

Case 2

- Even though parent class does not implements Serializable we can serialize child object if child class implements Serializable interface.
- At the time of serialization JVM ignores the values of instance variables which are coming from non Serializable parent then instead of original value JVM saves default values for those variables to the file.
- At the time of Deserialization JVM checks whether any parent class is non Serializable or not. If any parent class is non Serializable JVM creates a separate object for every non Serializable parent and shares its instance variables to the current object.
- To create an object for non-serializable parent JVM always calls no arg constructor(default constructor) of that non Serializable parent hence

every non Serializable parent should compulsory contain no arg constructor otherwise we will get runtime exception “InvalidClassException” .

- If non-serializable parent is abstract class then just instance control flow will be performed and share it's instance variable to the current object.

Example

```
import java.io.*;

class Animal{
    int i=10;
    Animal(){
        System.out.println("Animal constructor called");
    }
}

class Dog extends Animal implements Serializable{
    int j=20;
    Dog(){
        System.out.println("Dog constructor called");
    }
}

class SerializableWRTInheritance{
    public static void main(String[] args)throws Exception{
        Dog d1=new Dog();
        d1.i=888;
        d1.j=999;
        FileOutputStream fos=new FileOutputStream("abc.ser");
        ObjectOutputStream oos=new ObjectOutputStream(fos);
        oos.writeObject(d1);
        System.out.println("Deserialization started");
        FileInputStream fis=new FileInputStream("abc.ser");
        ObjectInputStream ois=new ObjectInputStream(fis);
        Dog d2=(Dog)ois.readObject();
        System.out.println(d2.i+"....."+d2.j);
    }
}
```

28.10 Externalization

- In default serialization every thing takes care by JVM and programmer doesn't have any control.
- In serialization total object will be saved always and it is not possible to save part of the object which creates performance problems at certain point.
- To overcome these problems we should go for externalization where every

thing takes care by programmer and JVM doesn't have any control.

- The main advantage of externalization over serialization is we can save either total object or part of the object based on our requirement.
- To provide Externalizable ability for any object compulsory the corresponding class should implements externalizable interface.
- Externalizable interface is child interface of serializable interface.

Externalizable interface defines 2 method:

- writeExternal()
- readExternal()

public void writeExternal(ObjectOutput out) throws IOException

This method will be executed automatically at the time of Serialization with in this method, we have to write code to save required variables to the file.

public void readExternal(ObjectInput in) throws IOException ClassNotFoundException

- This method will be executed automatically at the time of deserialization with in this method, we have to write code to save read required variable from file and assign to the current object.
- At the time of deserialization Jvm will create a separate new object by executing public no-arg constructor on that object JVM will call readExternal() method.
- Every Externalizable class should compulsory contains public no-arg constructor otherwise we will get RuntimeException saying "InvalidClassException".

Example

```
import java.io.*;

class ExternalDemo implements Externalizable{
    String s;
    int i ;
    int j ;
    public ExternalDemo(){
        System.out.println("public no-arg constructor");
    }
    public ExternalDemo(String s , int i, int j){
        this.s=s ;
        this.i=i;
```

```
this.j=j;
}
public void writeExternal(ObjectOutput out) throws IOException{
out.writeObject(s);
out.writeInt(i);
}
public void readExternal(ObjectInput in) throws IOException
,ClassNotFoundException{
s=(String)in.readObject();
i= in.readInt();
}}
public class Externalizable1{
public static void main(String[] args)throws Exception{
ExternalDemo t1=new ExternalDemo("som", 10, 20);
FileOutputStream fos=new FileOutputStream("abc.ser");
ObjectOutputStream oos=new ObjectOutputStream(fos);
oos.writeObject(t1);
FileInputStream fis=new FileInputStream("abc.ser");
ObjectInputStream ois=new ObjectInputStream(fis);
ExternalDemo t2=(ExternalDemo)ois.readObject();
System.out.println(t2.s+"-----"+t2.i+"-----"+t2.j);
}}
```

In externalization transient keyword won’t play any role, hence transient keyword not required.

28.11 Difference between Serialization & Externalization

Serialization	Externalization
It is meant for default Serialization	It is meant for Customized Serialization
Here every thing takes care by JVM and programmer doesn’t have any control.	Here every thing takes care by programmer and JVM doesn’t have any control.
Here total object will be saved always and it is not possible to save part of the object.	Here based on our requirement we can save either total object or part of the object.
Serialization is the best choice if we want to save total object to the file.	Externalization is the best choice if we want to save part of the object.

relatively performance is low	relatively performance is high
Serializable interface doesn't contain any method, and it is marker interface.	Externalizable interface contains 2 methods : 1. writeExternal() 2. readExternal() It is not a marker interface.
Serializable class not required to contains public no-arg constructor.	Externalizable class should compulsory contains public no-arg constructor otherwise we will get RuntimeException saying "InvalidClassException"
transient keyword play role in serialization	transient keyword don't play any role in Externalization

28.12 SerialVersionUID

- To perform Serialization & Deserialization internally JVM will use a unique identifier, which is nothing but serialVersionUID.
- At the time of serialization JVM will save serialVersionUID with object.
- At the time of Deserialization JVM will compare serialVersionUID and if it is matched then only object will be Deserialized otherwise we will get RuntimeException saying "InvalidClassException".

The process in depending on default serialVersionUID are

- After Serializing object if we change the .class file then we can't perform deserialization because of mismatch in serialVersionUID of local class and serialized object in this case at the time of Deserialization we will get RuntimeException saying in "InvalidClassException".
- Both sender and receiver should use the same version of JVM if there any incompatibility in JVM versions then receive unable to deserializable because of different serialVersionUID , in this case receiver will get RuntimeException saying "InvalidClassException" .
- To generate serialVersionUID internally JVM will use complexAlgorithm which may create performance problems.

We can solve above problems by configuring our own serialVersionUID.

We can configure serialVersionUID as follows:

```
private static final long serialVersionUID = 1L;
```

Example

```

class Dog implements Serializable {
private static final long serialVersionUID=1L;
int i=10;
int j=20;
}

class Sender{
public static void main(String[] args) throws Exception{
Dog d1=new Dog();
FileOutputStream fos=new FileOutputStream("abc.ser");
ObjectOutputStream oos= new ObjectOutputStream(fos);
oos.writeObject(d1);
}}

class Receiver{
public static void main(String[] args)throws Exception{
FileInputStream fis=new FileInputStream("abc.ser");
ObjectInputStream ois=new ObjectInputStream(fis);
Dog d2=(Dog) ois.readObject(); System.out.println(d2.i+"-----"+d2.j);
}}

```

In the above program after serialization even though if we perform any change to Dog.class file , we can deserialize object.

We if configure our own serialVersionUID both sender and receiver not required to maintain the same JVM versions.

Note: some IDE's generate explicit serialVersionUID.

Chapter 29

Tokenizing

29.1 Introduction

This last stop on our journey introduces string tokenizing and covers the following topics:

- StringTokenizer
- StreamTokenizer

29.1.1 Using StringTokenizer

Remember earlier when you wrote lines that contain data items separated by delimiters? The delimiters, which separate the data items (or tokens, as they are called), might be commas or Semicolons or tab character

StringTokenizer APIs:

```
StringTokenizer(String string_to_tokenize,String delimiters)
```

```
boolean hasMoreTokens()
```

```
String nextToken()
```

```
String nextToken(String new_delimiter)
```

```
int countTokens()
```

When you read each line back in, you need to separate the tokens. Each line might represent a name and address, such as in a mail-merge file. We want to break that line up into its individual parts, such as name, street, city, state, and zip.

The way to do that is to use the StringTokenizer class. Although this is not part of the I/O hierarchy, we'll cover it here because it is useful in reading delimited streams. Here's how that class works.

Constructing a StringTokenizer object

You construct an object of StringTokenizer, giving it the string you want to break up and what the delimiters are. That is, you tell it what characters are going to be used to break up the tokens in the string. After we have constructed an object of that type, we have a couple of things we can ask it:

- How many tokens are in this string?
- Have I used up all my tokens in this string?
- Give me the next token in the string.

`StringTokenizer` will break up the string and return to you as a `String` the characters up to the next delimiter. Its methods will throw a `NoSuchElementException` if there are no more tokens. Because this exception is derived from `RuntimeException` so that it is not declared with a `throws` clause, you do not need to catch it.

One more thing: you can switch delimiters for each token. In other words, even though you have created a `StringTokenizer` object that is looking for particular delimiters, you can say, “For this next token, change the delimiter.”

StringTokenizer: Example code

Look at the code below. Notice our string has a vertical bar and a question mark. Those are our delimiters. We are going to create a new `StringTokenizer`, and we are going to pass it the string `abc (bar) def (question mark) ghi`. We are going to use as our delimiters a bar and a question mark.

The `StringTokenizer.hasMoreTokens()` returns `true` if there are still more tokens. If it is `true`, we are going to call `nextToken()`. This returns a `String`, and `s` will have the value `abc` in it the first time around the while loop. We come around the loop a second time and `hasMoreTokens()` is still `true`. Now when we get the next token, `def` is the value of the string that is returned. Go around the loop a third time and `hasMoreTokens()` is still `true`; `ghi` is returned.

Now go around the loop one more time and `hasMoreTokens()` is `false`. At this point, we drop out of the loop. So we have broken up the string without too much effort.

Example

```
import java.io.*;
import java.util.*;

public class StringTokenizerExample{
    public static void main(String args[]){
        String line = "\"abc|def?ghi\"";
        StringTokenizer st = new StringTokenizer(line, "\"|?\"");
        while (st.hasMoreTokens()){
            String s = st.nextToken(); System.out.println("\"Token is \" + s);
        }
    }
}
```

29.1.2 StreamTokenizer

You can parse an entire input stream. Unlike the `StringTokenizer`, which parses a `String`, the `StreamTokenizer` reads from a `Reader` and breaks the stream into tokens. The parsing process is controlled by syntax tables and flags.

Each token that is parsed is placed in a category. The categories include identifiers, numbers, quoted strings, and various comment styles. The parsing that is performed is suitable for breaking a Java, C, or C++ source file into its tokens. `StreamTokenizer` is not in the I/O hierarchy, but because it is used with streams, we cover it here.

The `StreamTokenizer` recognizes characters in the range from `u0000` through `u00FF`. Each character value can have up to five possible attributes. The attributes are white space, alphabetic, numeric, string quote, and comment character.

- A character that is white space is used to separate tokens.
- An alphabetic character is part of an identifier.
- A numeric character can be part of an identifier or a number.
- A string quote character surrounds a quoted string.
- A comment character precedes or surrounds a comment.
- A character that does not have any of these attributes is an ordinary character. When an ordinary character is encountered, it is treated as a single character token.

Flags can be set to alter the parsing. Line terminators can either be tokens or white space separating tokens. C-style and C++-style comments can be recognized and skipped. Identifiers are converted to lower case or left as is.

Use of `StreamTokenizer`

To use a `StreamTokenizer`, you construct it with the underlying `Reader`. Then you set up the syntax tables and flags. Next, you loop on the tokens, calling `nextToken()` until it returns `TT_EOF`.

The `nextToken()` method parses the next token. The type is both returned by the method and also placed in the type field. The value of the token is placed in the `sval` field (String value) if the token is a word or in the `nval` field (numeric value) if the token is a number.

The token type can be either a character or a value, which represents the type of the token. If a token consists of a single character, the value of the type is the character value. If the token is a quoted string, the value is the value of the quote character. Otherwise it is one of the following:

- `TT_EOF` means the end of the stream has been read.

- `TT_EOL` means the end of the line has been read (if end of line characters are treated as tokens).
- `TT_NUMBER` means that a number token has been read.
- `TT_WORD` means that a word token has been read.

Methods of StreamTokenizer

There are many methods in `StreamTokenizer` to set up the syntax tables. We'll only mention two here. The first is `resetSyntax()`, which sets all characters to ordinary. The second is `wordChars()`, which gives a set of characters the alphabetic attribute.

StreamTokenizer: Example code

This example is a simple one that shows the use of the `StreamTokenizer`. It breaks a file into words consisting of lower- or upper-case letters. When a word is found, `nextToken()` returns `TT_EOF`.

If an ordinary character is found (in this case, anything not set as alphabetic), `nextToken()` returns the value of that character.

Example

```
import java.io.*;
import java.util.*;

public class StreamTokenizerExample{
    public static void main(String args[]){
        try{
            FileReader fr = new FileReader(t.txt);
            BufferedReader br = new BufferedReader(fr);
            StreamTokenizer st = new StreamTokenizer(br);
            st.resetSyntax();
            st.wordChars('\A', '\Z');
            st.wordChars('\a', '\z');
            int type;
            while ((type = st.nextToken()) != StreamTokenizer.TT_EOF){
                if(type==StreamTokenizer.TT_WORD)
                    System.out.println(st.sval);
            }
        }
        catch (IOException e){
            System.out.println(e);
        }
    }
}
```

Chapter 30

Scanner Class

30.1 Introduction

The `java.util.Scanner` class is a simple text scanner which can parse primitive types and strings using regular expressions. Following are the important points about Scanner: A Scanner breaks its input into tokens using a delimiter pattern, which by default matches whitespace. A scanning operation may block waiting for input. A Scanner is not safe for multithreaded use without external synchronization.

30.2 Class declaration

Following is the declaration for `java.util.Scanner` class:

```
public final class Scanner
extends Object
implements Iterator<String>
```

Input/Output in Java can be done using the keyboard and screen, using files, or some combination of these methods. Input typed at the keyboard and output displayed on the screen are often referred to as console input/output.

Interactive input/output using the keyboard and screen is the default way of doing input/output in jGRASP. It is also possible to use a dialog box to get input from the user and to display a message. Java refers to these as an input dialog and message dialog respectively. An object is always needed to perform an input/output operation in Java.

Display Output [standard output]

Displaying screen output in Java is done using the `System.out` object. You do not need to create the `System.out` object. It is always available for you use in a Java program. There are 3 methods that can be used to display output: `print( )`, `printf( )`, and `println( )`.

- The `print( )` method is often used to display a prompt or a portion of a line. It does not move the cursor to the beginning of a new line automatically.
- The `println( )` method displays output exactly the same as the `print( )` method except that it will always move the cursor to the beginning of a new line as the last action it performs.
- The `printf( )` method is used to output formatted text and numbers. Like the

`print( )` method, It does not move the cursor to the beginning of a new line automatically.

Keyboard Input [standard input]

Accepting keyboard input in Java is done using a `Scanner` object. There is a `System.in` object in Java that can be used to get input in Java, but it is not very functional. Instead, Java programmers usually prefer to use a `Scanner` object that has been “mapped” to the `System.in` object. The `System.in` object (like the `System.out` object) already exists and is available for use. However, you must create a `Scanner` object.

Once you have created a `Scanner` object and “mapped” it to the `System.in` object you can use the following methods to test for and get input. Input always comes in to a Java program as a string of characters (`Java String` type). There are methods that can be used to convert the `String` to a character, an integer number, a floating-point number, etc. Here are commonly used `Scanner` methods:

`hasNext()` //is something available to read? Returns true or false.

`nextInt()` //get an int `nextDouble()` //get a double `next()` //get a `String` (delimited by whitespace) `nextLine()` //get the rest of the line as a `String`

Note: There is no method to read a character! Read a string instead.

30.3 Examples of Interactive input/output in Java

Use a `Scanner` object “mapped” to the `System.in` object for input and the standard output object `System.out` for output.

Input

1. Use the `Scanner` class to create an input object using the standard input object in the `System` class (`System.in`). The `Scanner` class is in the package `java.util`

```
Scanner input = new Scanner( System.in );
```

//input object is informally called “input”
2. Use `Scanner` class methods to get values the user enters using the keyboard.

```
int num1; double num2; String str;
num1 = input.nextInt();    //get an integer and assign it to num1
num2 = input.nextDouble(); //get a real number and assign it to num2
str = input.nextLine();    //get a String and assign it to str
```

Output:

1. Use the standard output object in the `System` class (`System.out`).

2. Use `PrintStream` class methods (`printf()`, `print()`, `println()`) to display the output to the screen using the `System.out` object. The `System.out` object is an object of type `PrintStream`.

```
System.out.print("What is your name? ");
String name = input.nextLine();
System.out.println("Hello there " + name + ", nice to meet you!");
double purchasePrice = 178.34;
System.out.printf("Your total is $%.2f\n", purchasePrice);
```

30.4 Class constructors

Scanner Class Constructor & Description

1 ScannerFilesource

This constructs a new `Scanner` that produces values scanned from the specified file.

2 ScannerFilesource,StringcharsetName

This constructs a new `Scanner` that produces values scanned from the specified file.

3 ScannerInputStreamsource

This constructs a new `Scanner` that produces values scanned from the specified input stream.

4 ScannerInputStreamsource,StringcharsetName

This constructs a new `Scanner` that produces values scanned from the specified input stream.

5 ScannerReadablesource

This constructs a new `Scanner` that produces values scanned from the specified source.

6 ScannerReadableByteChannelsource

This constructs a new `Scanner` that produces values scanned from the specified channel.

7 ScannerReadableByteChannelsource,StringcharsetName

This constructs a new Scanner that produces values scanned from the specified channel.

8 ScannerStringsource

This constructs a new Scanner that produces values scanned from the specified string.

30.5 Class methods

Scanner Class Method & Description

1 void close

This method closes this scanner.

2 Pattern delimiter

This method returns the Pattern this Scanner is currently using to match delimiters.

3 String findInLinePatternpattern

This method attempts to find the next occurrence of the specified pattern ignoring delimiters.

4 String findInLineStringpattern

This method attempts to find the next occurrence of a pattern constructed from the specified string, ignoring delimiters.

5 String findWithinHorizonPatternpattern,inthorizon

This method attempts to find the next occurrence of the specified pattern.

6 String findWithinHorizonStringpattern,inthorizon

This method attempts to find the next occurrence of a pattern constructed from the specified string, ignoring delimiters.

7 boolean hasNext

This method returns true if this scanner has another token in its input.

8 boolean hasNextPatternpattern

This method returns true if the next complete token matches the specified pattern.

9 boolean hasNextStringpattern

This method returns true if the next token matches the pattern constructed from the specified string.

10 boolean hasNextBigDecimal

This method returns true if the next token in this scanner's input can be interpreted as a BigDecimal using the nextBigDecimal method.

11 boolean hasNextBigInteger

This method returns true if the next token in this scanner's input can be interpreted as a BigInteger in the default radix using the nextBigInteger method.

12 boolean hasNextBigIntegerinradix

This method returns true if the next token in this scanner's input can be interpreted as a BigInteger in the specified radix using the nextBigInteger method.

13 boolean hasNextBoolean

This method returns true if the next token in this scanner's input can be interpreted as a boolean value using a case insensitive pattern created from the string "true|false".

14 boolean hasNextByte

This method returns true if the next token in this scanner's input can be interpreted as a byte value in the default radix using the nextByte method.

15 boolean hasNextByteinradix

This method returns true if the next token in this scanner's input can be interpreted as a byte value in the specified radix using the nextByte method.

16 boolean hasNextDouble

This method returns true if the next token in this scanner's input can be interpreted as a double value using the nextDouble method.

17 boolean hasNextFloat

This method Returns true if the next token in this scanner's input can be interpreted as a float value using the nextFloat method.

18 boolean hasNextInt

This method returns true if the next token in this scanner's input can be interpreted as an int value in the default radix using the nextInt method.

19 boolean hasNextIntinradix

This method returns true if the next token in this scanner's input can be interpreted as an int value in the specified radix using the nextInt method.

20 boolean hasNextLine

This method returns true if there is another line in the input of this scanner.

21 boolean hasNextLong

This method returns true if the next token in this scanner's input can be interpreted as a long value in the default radix using the nextLong method.

22 boolean hasNextLonginradix

This method returns true if the next token in this scanner's input can be interpreted as a long value in the specified radix using the nextLong method.

23 boolean hasNextShort

This method returns true if the next token in this scanner's input can be interpreted as a short value in the default radix using the nextShort method.

24 boolean hasNextShortinradix

This method returns true if the next token in this scanner's input can be interpreted as a short value in the specified radix using the nextShort method.

25 IOException ioException

This method returns the IOException last thrown by this Scanner's underlying Readable.

26 Locale locale

This method returns this scanner's locale.

27 MatchResult match

This method returns the match result of the last scanning operation performed

by this scanner.

28 String next

This method finds and returns the next complete token from this scanner.

29 String nextPatternpattern

This method returns the next token if it matches the specified pattern.

30 String nextStringpattern

This method returns the next token if it matches the pattern constructed from the specified string.

31 BigDecimal nextBigDecimal

This method scans the next token of the input as a BigDecimal.

32 BigInteger nextBigInteger

This method Scans the next token of the input as a BigInteger.

33 BigInteger nextBigIntegerintradix

This method scans the next token of the input as a BigInteger.

34 boolean nextBoolean

This method scans the next token of the input into a boolean value and returns that value.

35 byte nextByte

This method scans the next token of the input as a byte.

36 byte nextByteintradix

This method scans the next token of the input as a byte.

37 double nextDouble

This method scans the next token of the input as a double.

38 float nextFloat

This method scans the next token of the input as a float.

39 int nextInt

This method scans the next token of the input as an int.

40 int nextIntintradix

This method scans the next token of the input as an int.

41 String nextLine

This method advances this scanner past the current line and returns the input that was skipped.

42 long nextLong

This method scans the next token of the input as a long.

43 long nextLongintradix

This method scans the next token of the input as a long.

44 short nextShort

This method scans the next token of the input as a short.

45 short nextShortintradix

This method scans the next token of the input as a short.

46 int radix

This method returns this scanner's default radix.

47 void remove

The remove operation is not supported by this implementation of Iterator.

48 Scanner reset

This method resets this scanner.

49 Scanner skipPatternpattern

This method skips input that matches the specified pattern, ignoring delimiters.

50 Scanner skipStringpattern

This method skips input that matches a pattern constructed from the specified string.

51 String toString

This method returns the string representation of this Scanner.

52 Scanner useDelimiterPatternpattern

This method sets this scanner's delimiting pattern to the specified pattern.

53 Scanner useDelimiterStringpattern

This method sets this scanner's delimiting pattern to a pattern constructed from the specified String.

54 Scanner useLocaleLocalelocale

This method sets this scanner's locale to the specified locale.

55 Scanner useRadixinradix

This method Sets this scanner's default radix to the specified radix.

Example on Scanner

```
import java.util.Scanner;

public class Jtc16 {
    public static void main(String[] args)
    { Scanner sc = new Scanner(System.in);
    int count = 0;
    double sum = 0.0;
    System.out.println("Enter Numbers.");
    while (sc.hasNext())
    { System.out.println("Enter Numbers
    Again."); if (sc.hasNextDouble()) {
    sum += sc.nextDouble();
```

```

count++;
} else {
String str = sc.next();
if (str.equals("*"))
break;
}
}
System.out.printf("Sum of given %d numbers id %f", count, sum);
}
}

```

Example

```

import java.io.FileReader;
import java.io.FileWriter;
import java.util.Scanner;

public class Jtc17 {
public static void main(String[] args) {
Scanner sc = null;
int count = 0;
int sum = 0;
FileWriter wr = null;
try {
wr = new FileWriter("Numbers.txt");
wr.write("10 20 30 40 50 *");
wr.close();
FileReader rd = new
FileReader("Numbers.txt");
System.out.println("Numbers Stored in File.");
sc = new Scanner(rd);
} catch (Exception e) {
e.printStackTrace();
}
while (sc.hasNext()) {

```

```
if (sc.hasNextDouble()) {  
    sum += sc.nextDouble();  
    count++;  
} else {  
    String str = sc.next();  
    if (str.equals("*"))  
        break;  
}  
}  
System.out.printf("Sum of given %d numbers is %d", count, sum);  
}  
}
```

Chapter 31

Annotations

31.1 Introduction

Annotations provides the special way of specifying Metadata inside the java class instead of writing inside XML file.

When you are developing J2EE based applications using Struts, Hibernate, Spring, EJB etc, you need to write many XML documents which are specific to corresponding framework or technology.

With jdk1.5 you can avoid this XML by specifying the Meta-Data related to Struts, Hibernate, Spring inside the class itself.

There are mainly three tasks in Annotations implementation and uses.

- 1) Define the Annotations.
- 2) Provide the APT(Annotation Processing Tool).
- 3) Use the Annotation.

Normally technology or framework vendors are responsible for defining the Annotation and also for providing the Annoation Processing Tools(APT).

As a developer you are just responsible to use the Annotations.

If you want you can create custom annotation by using following syntax:-

- 1) Define the Annotaion

Syntax:

```
@interface Anno_Name{
    datatype member_name();
    datatype member_name();
}
```

Example

- 1) Define the Annotaion

```
@interface Author{
    String aname();
    int age();
}
```

- 2) Using the Annotations


```
@Author(aname="Som",age=106)
class Hello
{
...
}
```

Example

- 1) Define the Annotation

```
@interface Servlet{
String name();
String url();
int load();
}
```

- 2) Using the Annotations

```
@Servlet
(name="login",url="/login.jtc",load=1)
class LoginServlet{
}
```

//Here no need to extends HttpServlet

Annotation definition gives you new userdefined datatype.

Every Annotation you define is a subtype of `java.lang.annotation.Annotation` and it the default supertype.

All the Annotation members will be implemented by the JVM internally.If you want to process the Annotation then you can use the following class and their methods to get the information:-

java.lang.Class

```
public boolean isAnnotationPresent(Class cls);
public class getAnnotation(Class cls);
public class[] getAnnotations();
```

31.2 Types of Annotations

You can develop the following types of Annotation:

- 1) Marker Annotations
- 2) Single Valued Annotations
- 3) Multi Valued Annotations

Marker Annotations

Annotation which is defined without any members are called as Marker Annotation.

Example

```
@interface Serializable{
}
```

Usage:

```
@Serializable
class Hello{
}
```

Single Valued Annotations

Annotations which are defined with only one member are called as single valued Annotations.

Example

```
@interface Author{
String value();
}
```

usage:

```
@Author(value="som")
    or
@Author("som")
class Hello{
}
```

Multi Valued Annotations

Annotations which are defined with two or more members are called as Multi Valued Annotations.

Example

```
@interface Employee{
int eid();
String ename();
long phone();
}
```

usage:

```
@Employee(eid=101,ename="Som",phone=3333333)
class Hello{
...
}
```

If you are defining the custom Annotation then you may get the requirement to use some predefined Annotation.

31.3 Built-In Annotations

- 1) @Retention
- 2) @Target
- 3) @Inherited
- 4) @Deprecated
- 5) @Override
- 6) @SuppressWarnings
- etc.

@Retention

This Annotation is used to specify the Retention policy of the Annotation. Retention policy indicates that what will be the scope of the Annotation. You can define the following Retention

Policy for any Annotation

-->SOURCE

-->CLASS

-->RUNTIME

These three Retention Policies are defined as Constants in the enum called RetentionPolicy.

Usage

```
@Retention
(RetentionPolicy.CLASS/SOURCE/RUNTIME)
@interface Author{
}
```

SOURCE

When the RetentionPolicy is SOURCE then annotations will be available only in source code and will not be available in Byte Code and Native Code. Source is the

Default RetentionPolicy

CLASS

When the RetentionPolicy is CLASS then Annotations will be available only in source code and Byte Code and will not be available in and Native Code.

RUNTIME

When the RetentionPolicy is RUNTIME then Annotations will be available in source code, Byte Code and Native Code.

@Target

This annotation is used to specify the Target, i.e to specify whether the annotation should be used for class or method or field or constructor or all.

All the targets are defined as constants in the enum called ElementType.

E.g

```
@Target(ElementType.Type)
```

```
@interface Author{
}
```

```
@Target(ElementType.Method)
```

```
@interface Author{
}
```

@Inherited

If you are defining any custom annotation then by default the annotation will not be inherited in the subclass.

If you want to inherit the super class annotations to the subclass then annotation definition must be marked with @Inherited.

@Deprecated

This annotation will be used to define the custom method as deprecated.

If you are defining any method that implementation is not available or it may be changed in the future version then you can define that method as @Deprecated.

@Override

It will be used with the methods. It forces the compiler to match the signature of the method in the super class.

If the method signature will not be available in the super class then compiler will give compilation error.

`@SuppressWarnings`

By using this annotation you can suppress the warning message. While compiling the source file compiler won't display any warning message if the warning will be suppressed.

Example

```
package com.JTC.anno;
import java.lang.annotation.*;
@Inherited
@Target(ElementType.TYPE)
@Retention(RetentionPolicy.RUNTIME)
@interface Author{
String aname();
int age();
}
@Author(aname="Som",age=88)
class Hai{
@Deprecated
void m1(){
}}
class A extends Hai{
    void m1(){}
}
public class Test5{
public static void main(String args[]){
try{
Class cls=Class.forName("com.JTC.anno.A");
boolean b=cls.isAnnotationPresent
(Author.class);
System.out.println(b);
if(b){
Author a=(Author)cls.getAnnotation
(Author.class);
System.out.println(a.aname()+"\t"+a.age());
}}
```

```
catch(Exception e){
e.printStackTrace();}}}
```

Examples on Annotation

```
import java.util.Date;

public class Jtc19 {
public static void main(String[]
args) { Student st = new Student();
System.out.println(st);
st.m2Info();
st.m1StudentInfo();
Date dt = new Date();
System.out.println(dt.getDate());
System.out.println(dt.getMonth());
System.out.println(dt.getYear());
}}

class Student {
@Override
public boolean equals(Object obj) {
return super.equals(obj);
}

@Deprecated
void m1StudentInfo() {
}

void m2Info()
{ System.out.println("-- M2Info --");
}

public int hashCode() {
return 10;
}

/*
@Override
```

```

public String toString() {
    return "Student Obj";
}
*/
}
@Deprecated
class Hello{}
@Deprecated
interface Inter1{}

```

Example

```

import java.util.Date;

public class Jtc20 {
    @SuppressWarnings({ "deprecation", "static-access", "unused" })
    public static void main(String[] args) {
        int ab = 10;
        Date dt = new Date();
        System.out.println(dt.getDate());
        System.out.println(dt.getDate());
        Employee emp = null;
        emp.m2();
    }
    class Employee {
        void m1() {
            @SuppressWarnings("unused")
            int xy = 10;
            int mn = 90;
        }
        static void m2() {
            System.out.println("*** M2 Static Method **");
        }
    }
}

```

Example

```

import java.lang.annotation.*;

public class Jtc21 {
    public static void main(String[] args) {
        @Author(id = 101, name = "SomPraksh", phone = 8585745L)
        Book b1 = new Book("Core Java");
        @Author(name = "Vikas", phone = 9673563844L)
        Book b2 = new Book("Hibernate");
        System.out.println("---- APT -----");
        boolean pre = b2.getClass().isAnnotationPresent(BBAnno.class);
        if (pre) {
            BBAnno ref = (BBAnno) b2.getClass().getAnnotation(BBAnno.class);
            System.out.println("Value\t:" + ref.value());
        }
        Annotation ans[] = b2.getClass().getAnnotations();
        for (Annotation an : ans) {
            System.out.println(an);
        }
        System.out.println();
        JtcServlet serv = RegisterServlet.class.getAnnotation(JtcServlet.class);
        System.out.println(serv.url());
    }
    @Target(ElementType.TYPE)
    @interface JtcAnno { }
    @Retention(RetentionPolicy.RUNTIME)
    @interface BBAnno {
        String value();
    }
    @Retention(RetentionPolicy.RUNTIME)
    @interface JtcServlet {
        String url();
    }

```



```

}
@Target(ElementType.LOCAL_VARIABLE)
@interface Author {
int id() default 99;
String name();
long phone();
}
@JtcAnno
@BBAnno("1234")
class Book {
// @JtcAnno
String name;
Book(String name) {
this.name = name;
}}
@BBAnno(value = "1234")
class BBStudent { }
/*
* @JtcServlet("/login.Jtc") class LoginServlet{}
*/
@JtcServlet(url = "/regster.Jtc")
class RegisterServlet { }

```

Example

```

import java.lang.annotation.*;

class Jtc22 {
public static void main(String[] args) throws CloneNotSupportedException {
Emp emp = new Emp(99, "SomPraksh");
System.out.println(emp);
Emp emp2 = emp.getClonedObject();
System.out.println(emp2);
System.out.println(emp == emp2);
}
}

```

```

}}
@JtcCloneable
class Emp {
    int eid;
    String ename;
    Emp(int eid, String ename) {
        this.eid = eid;
        this.ename = ename;
    }
    @Override
    public String toString() {
        return eid + "\t" + ename;
    }
    public Emp getClonedObject() throws CloneNotSupportedException {
        boolean b1 = this.getClass().isAnnotationPresent(JtcCloneable.class);
        if (b1) {
            return new Emp(this.eid, this.ename);
        }
        throw new CloneNotSupportedException("Emp class not using JtcCloneable");
    }
}
@Retention(RetentionPolicy.RUNTIME)
@interface JtcCloneable { }

```

Example

```

import java.lang.annotation.*;
import java.lang.reflect.*;

public class Jtc23 {
    public static void main(String[] args) throws Exception {

        JtcEmployee emp = new JtcEmployee();
        emp.empld = 99;
    }
}

```

```
emp.empName = "SomPraksh";
emp.empPhone = 6526668;
JtcJdbcTemplate res = new JtcJdbcTemplate();
res.save(emp);
```

```
JtcStudent stud = new JtcStudent();
stud.studId = 3131;
stud.studName = "Manish";
stud.phone = 9590712983L;
stud.studFee = 27000.0F;
res.save(stud);
}}
```

```
@Table(name = "studTable")
class JtcStudent {
    @Column(name = "sid")
    int studId;
    @Column(name = "sfee")
    float studFee;
    long phone;
    @Column(name = "sname")
    String studName;
}
```

```
@Table(name = "empTable")
class JtcEmployee {
    @Column(name = "eid")
    int empld;
    @Column(name = "eage")
    int age;
    @Column(name = "ename")
    String empName;
    @Column(name = "ephone")
    long empPhone;
}
```

// IMPLEMENTED BY JTC VENDOR

```
@Retention(RetentionPolicy.RUNTIME)
```

```

@Target(ElementType.TYPE)
@interface Table {
    String name();
}

@Target(ElementType.FIELD)
@Retention(RetentionPolicy.RUNTIME)
@interface Column {
    String name();
}

class JtcJdbcTemplate {
    public void save(Object obj) throws Exception {
        boolean tabPresent = obj.getClass().isAnnotationPresent(Table.class);
        if (tabPresent) {
            Table tab = obj.getClass().getAnnotation(Table.class);
            String tableName = tab.name();
            String qry = "insert into " + tableName;
            StringBuffer cols = new StringBuffer("(");
            StringBuffer values = new StringBuffer(" values (");
            Field fs[] = obj.getClass().getDeclaredFields();
            for (int i = 0; i < fs.length; i++) {
                Field f1 = fs[i];
                boolean colPresent = f1.isAnnotationPresent(Column.class);
                if (colPresent) {
                    String colName = f1.getAnnotation(Column.class).name();
                    cols.append(colName + ",");
                    String type = f1.getType().getSimpleName();
                    if (type.equals("int")) {
                        int val = f1.getInt(obj);
                        values.append(val);
                        values.append(",");
                    } else if (type.equals("String")) {
                        String val = f1.get(obj).toString();
                        values.append("''");
                        values.append(val);
                        values.append(",");
                    } else if (type.equals("long")) {
                        long val = f1.getLong(obj);
                        values.append(val);

```

```

values.append(",");
} else if (type.equals("float")) {
float val = f1.getFloat(obj);
values.append(val);
values.append(",");
}}}
qry = qry + cols.substring(0, cols.length() - 1) + ")" + values.substring(0, values.
length() - 1) + " ";
System.out.println(qry);
// Code to Store in DB
} else {
String cName = obj.getClass().getName();
throw new RuntimeException(cName + " is not annotated with @Table");
}}}

```

Chapter 32

java.util.concurrent package

32.1 The java.util.concurrent

Package contains a number of concurrent collection classes that serve the purpose of removing memory inconsistency. Java does this by defining a happens-before relationship between an operation that adds items to a collection and any operations that attempt to read or remove that item from the collection.

A number of classes are added to the collections framework by java.util.concurrent package. The additional collection classes are:

- **BlockingQueue:** FIFO queue, blocks or times out when adding to a full queue or retrieving from an empty queue.
- **ConcurrentMap:** Subinterface of java.util.Map that includes a number of additional atomic operations.
- **ConcurrentNavigableMap:** Subinterface of ConcurrentMap that supports approximate matches.

BlockingQueue is safe to use with multiple threads and defines an initial capacity which cannot be changed. Items can be added and removed from the queue but once the queue has filled up its capacity it will block attempts to add any new elements. In addition the queue will block any attempt to remove items from a queue.

BlockingQueues are used to implement producer-consumer design pattern. This is a design pattern where single or multiple producer threads can add items to a queue to be consumed by a number of other threads. The use of the queue here allows for the producer to add items to a collection at any speed it wants without worrying about a consumer's ability to consume the items at the same rate.

ConcurrentMap defines methods to replace or remove a key-value pair if the key already exists or add a value if the key is not associated with a value.

ConcurrentHashMap is a concrete implementation of the ConcurrentMap interface and is analogous to a concurrent version of the HashMap class. When accessing a HashMap using multiple threads it is possible to synchronize access to it however this has a performance impact as the entire object will be locked meaning read and write operations will all be blocked until the current operation is complete.

The `ConcurrentHashMap` provides better performance by allowing multiple threads to read concurrently while limiting modification operations by multiple threads.

Because `ConcurrentHashMap` allows multiple concurrent reads iterators do not throw a `ConcurrentModificationException` as the collection does not need to be locked while iterating over the elements.

The question arises what happens to items added to a `ConcurrentHashMap` after iteration of the map is begun. It is possible that the map will iterate over the items that existed at the time the iterator was created but on some platforms the iterator may include new items that were added. This however is not guaranteed.

Because the collection is not locked while modifying the elements some of the methods might not return an accurate value when called by multiple threads. The `size()` method is one example, an accurate size of the map may not be returned.

The `ConcurrentMap` interface added a number of methods that wrapped previous operations into a single atomic operation. When using a `HashMap` the tendency is to check if a key exists in a map and if not then put the key value pair onto the map.

A `ConcurrentHashMap` wraps both of these operations into a single `replace` method. This prevents any possibility of a race condition between making the two method calls from multiple threads.

When executing a `synchronize` block an implicit lock is acquired on an object. Java offers a number of `Lock` objects that offer a greater level of locking control. `Lock` objects can define a time duration to wait to acquire a lock on an object and lock polling.

The `lock()` method attempts to acquire a lock on a `Lock` object or waits if the lock cannot be acquired. When a lock is no longer required a call to `unlock()` must be made to release this lock. This call should be made in a `finally` block to ensure it is executed in the event of an exception.

Implicit locks that have attempted to acquire a lock cannot quit, in contrast explicit locks can attempt to acquire a lock and the call will return immediately. Calling method `tryLock()` on a lock object immediately returns a boolean defining whether or not it is possible to lock an object.

Threads that acquire explicit locks will never deadlock which is in contrast to threads that acquire implicit locks which can at times deadlock.

When finished with a locked resource a call to the `unlock()` method releases the lock on that object.

If we don't want to wait any longer than a specified time we can call the `tryLock()` method with arguments to specify how long we

want to wait before giving up on trying to acquire a lock. We pass two arguments to this method, a long value representing the value of time and a `TimeUnit` object.

It is also possible to declare that a thread can be interrupted while waiting for a lock. The `lockInterruptibly()` method will attempt to acquire a lock on an object unless it is interrupted at which point an `InterruptedException` is thrown.

Implicit locks cannot span multiple blocks of code, locks are released at the end of the synchronized block. Explicit lock objects however can span multiple methods.

ReadWriteLock interface maintains a pair of locks, one for reading that can be acquired by multiple threads provided there are no writing operations ongoing and a write lock that can only be acquired by a single thread at a time. To acquire a read lock a call to `readLock()` has to be made. The `writeLock()` method must be called to acquire a write lock.

ReentrantReadWriteLock is a class that implements the `ReadWriteLock` interface.

A number of classes were added to java to support atomic operations of reading and comparing variable and also modifying and writing variables. These additions were made to remove possibility of thread interference of non atomic operations such as incrementing primitive variables.

`AtomicInteger` is one such class and has atomic operations `incrementAndGet()` and `decrementAndGet()`. Any concurrent execution of these methods will not result in thread interference. Methods of this class are in the form `getAndXxx` and `xxxAndGet` where `xxx` is the operation to be performed such as increment or decrement.

When calling the `getAndXxx` type methods these return the previous value whereas when calling the `xxxAndGet` methods these perform the operation and then return the updated value.

32.2 Other atomic classes are:

- `AtomicBoolean`
- `AtomicLong`
- `AtomicIntegerArray`
- `AtomicLongArray`

- AtomicReference<V>

Invalid atomic classes are:

- AtomicByte

- AtomicShort

- AtomicFloat

- AtomicDouble

Chapter 33

Executor Framework

33.1 Introduction

Executor framework in Java separate out submissions of tasks and execution of those tasks. Tasks can be created from the `Runnable` and `Callable` interfaces.

ExecutorService extends **Executor** and adds a number of methods to manage the lifecycle of tasks being executed and the executors executing those tasks.

`ScheduledExecutorService` extends `ExecutorService` and supports periodic execution or future execution of tasks. The `Future` class represents the state of asynchronous tasks and from this class there are a number of operations to query the state and cancel `Future` objects.

The `Executors` class defines a number of utility and factory methods for the above interfaces.

When implementing the `Executor` interface only one method - `execute()` needs to be implemented. With this method we can determine how we want to execute `Runnable` (interface `Runnable`) tasks including order of task execution, number of tasks that can be executed concurrently and the number of tasks that can be queued.

The `Callable` interface differs from the `Runnable` interface only in the signature of its method, firstly the sole method of this interface is called `call` and unlike the `Runnable` interface its method for the return of a value and the ability to throw a checked exception.

In addition the `Callable` interface is a parameterised interface and the type argument defines the type of values that can be returned from the `call()` method.

If the `call` method is not required to return a value we can define the type argument to be `Void`.

`Callable` objects cannot be passed to a thread for execution in the same way a `Runnable` object can so we must use the `ExecutorService` interface when executing `Callable` objects.

The `ExecutorService` defines methods to manage progress and termination of tasks. Single `Runnable` or `Callable` objects can be submitted to the `ExecutorService` with `Future` objects being returned. Multiple `Runnable` objects can be submitted

with Future objects being returned. There are also methods for shutting down the ExecutorService and allowing or disallowing submitted tasks to be completed.

Thread pools allow for the creation of a pool of reusable threads of a predefined size to execute tasks concurrently.

The fork/join framework was added to Java 7 to support hardware parallelism. Its purpose is to split larger tasks into smaller tasks that can be executed in parallel and once complete merge the results of the individual tasks.

The fork/join framework works by evaluating the size of the task to be executed and if it is not large enough to be divided it will be executed sequentially or if it is then tasks will be divided until they are small enough to be executed sequentially.

On completion of each tasks the result is merged back into the thread from which this thread was forked.

ForkJoinPool is a concrete implementation of the fork/join framework that implements the ExecutorService interface. The fork/join

framework maintains a queue of tasks that are submitted for execution but implements a work-stealing algorithm. Once a thread no longer has any tasks left in its queue it will steal tasks from the queues of other threads. This prevents the blocking of waiting threads.

When using the fork/join framework any user defined class that must do processing should extends the abstract class ForkJoinTask to be executed within a ForkJoinPool.

ForkJoinTask is extended by two classes - RecursiveAction for computations that do not return a result and RecursiveTask for computations that do return a result.

Example

CarHire.java

```
public class CarHire implements Runnable {
    String carReg;
    public CarHire(String carReg) {
        this.carReg = carReg;
    }
    public void run() {
        System.out.println("Completed hire of " + this.carReg);
    }
}
```

Example

CarHireCallable.java

```
import java.util.Map;
import java.util.HashMap;
import java.util.concurrent.Callable;

public class CarHireCallable implements Callable<Boolean> {
    String carReg;
    String driverName;
    private static Map<String, String> driverCarMap = new HashMap<>();
    static {
        driverCarMap.put("AA01 ABC", "SOM Pra");
        driverCarMap.put("BB02 DEF", "Rai Pra");
    }
    public CarHireCallable(String carReg, String driverName) {
        this.carReg = carReg;
        this.driverName = driverName;
    }
    public Boolean call() throws Exception {
        if(driverName.equals(driverCarMap.get(carReg))) {
            System.out.println(getDriverName()+" has hired car "+getCarReg());
            return true;
        }
        else {
            System.out.println("Incorrect driver car pairing: "+this.carReg+", "+this.driverName);
            return false;
        }
    }
    public String getDriverName() {
        return driverName;
    }
    public String getCarReg() {
        return carReg;
    }
}
```

Example

CarHireException.java

```
public class CarHireException extends Exception {
```

```

public CarHireException(String message) {
    super(message);
}
}
import java.util.Queue;
import java.util.ArrayDeque;
import java.util.concurrent.Executor;
public class HireDesk implements Executor {
    final Queue<Runnable> waitingCustomers = new ArrayDeque<>();
    public void execute(Runnable r) {
        waitingCustomers.offer(r);
        serveNextCustomer();
    }
    private void serveNextCustomer() {
        Runnable task = waitingCustomers.poll();
        new Thread(task).start();
    }
    public static void main(String[] args) {
        HireDesk desk = new HireDesk();
        desk.execute(new CarHire("AA01 ABC"));
        desk.execute(new CarHire("BB02 DEF"));
        desk.execute(new CarHire("CC03 GHJ"));
    }
}

```

Example

```

HireDeskExecutorService.java
import java.util.concurrent.*;
public class HireDeskExecutorService {
    ExecutorService service = Executors.newFixedThreadPool(5);
    public void serveNextCustomer(CarHireCallable chc) {
        // Submit this Callable instance to the ExecutorService. Will be queued until
        // thread from the pool becomes available
        service.submit(chc);
    }
    public void closeDesk() {
        service.shutdown();
    }
}

```

```

public static void main(String[] args) {
    HireDeskExecutorService desk = new HireDeskExecutorService();
    desk.serveNextCustomer(new CarHireCallable("AA01 ABC", "SOM Pra"));
    desk.serveNextCustomer(new CarHireCallable("BB02 DEF", "Rai Pra"));
    desk.serveNextCustomer(new CarHireCallable("CC03 GHJ", "Man"));
    desk.closeDesk();
    // This one should throw an exception as it cannot be executed after the
    // ExecutorService has been shutdown
    desk.serveNextCustomer(new CarHireCallable("AB12 CDE", "Tom"));
}
}

```

Example

ScheduledTask.java

```

public class ScheduledTask implements Runnable {
    public void run() {
        System.out.println("Scheduled task executed at "+System.currentTimeMillis());
    }
}

```

Example

ScheduledTaskExecutor.java

```

import java.util.concurrent.*;

public class ScheduledTaskExecutor {
    ScheduledExecutorService service = Executors.newScheduledThreadPool(1);
    ScheduledTask task = new ScheduledTask();
    public void executeScheduledTask() {
        service.scheduleAtFixedRate(task, 0, 1, TimeUnit.MINUTES);
    }

    public static void main(String[] args) {
        ScheduledTaskExecutor executor = new ScheduledTaskExecutor();
        executor.executeScheduledTask();
    }
}

```

Enum

- enum is a keyword added from jdk1.5.
- enum is a type of class whose fields consist of fixed set of constants of the enum type.

- If you want to define the constant of Class type it is not possible but same we can do with enum.

```
class Hello{
public static final int x=0;
public static final int y=1;
}
interface Inter{
int x=0;
int y=1;
}
enum MyEnum{
X,Y
}
```

for every enum constants defined, one Enum object will be created which is of Enum type.

Example

```
MyEnum obj1= new MyEnum();//Not Ok
MyEnum obj2=MyEnum.X;
MyEnum obj3=MyEnum.Y;
```

- You can't create instance for Enum, but you can declare the reference variable.

```
package com.jtc.jtc2;
enum Days{
MON , TUE,WED,THU,FRI,SAT,SUN
}
public class jtc2{
public static void main(String ar[]){
Days d1=Days.FRI;
System.out.println(d1+"\t"+d1.ordinal());
if(d1==Days.SAT){
System.out.println("OK");
}
else{
System.out.println("Not Ok");
}
Days dd[]=Days.values();
for(Days d:dd){
System.out.println(d+"\t"+d.ordinal()+"\t"+d.name());
```

```

}
show(Days.MON);
show(Days.SUN);
}
static void show(Days d){
switch(d){
case MON:case TUE:case WED:case THU:case FRI: System.out.println("Weekdays");
break;
case SAT: case SUN: System.out.println("Weekends");
break;
default:
System.out.println("Not Available");
}}

```

- For every enum object some name and ordinal(integer) value will be allocated by the JVM.
- You can access those values by using the following methods:
 - ✓ String name();
 - ✓ int ordinal();

Difference B/W Enum and Class

Class	Enum
1) For every class, java.lang.Object is the super class.	1) For every enum, java.lang.Enum is the super type
2) Class can be instantiated.	2) Enum can't be instantiated but you can declare the reference variable.
3) One class can extend another class.	3) One enum can't extend another enum.
4) Class can be marked as abstract and final.	4) Enum can't be marked as abstract and final.
5) Class can contain abstract methods.	5) Enum can't contain abstract methods.

Similarity

- 1) Both can have the following members:
 - a) instance variable

- b) static variable
 - c) instance block
 - d) static block
 - e) instance method
 - f) static method
 - g) constructors
- 2) Both can implements interfaces
- 3) Class can have inner classes and Enum can also have inner enum.

Course.java

```
package com.jtc.jtc3;
interface inter1{
void m1();
}
interface inter2{
void m2();
}
public enum Course implements inter1,inter2{
JAVA(101,5000,"vikas"),SERVLET(102,3000,"Manish"),
JSP(103,2000,"shalni"), STRUTS(104,4500,"somprakash"),
HIBERNATE(105,5500,"rai"),    EJB(106,6500,"webearn"),SPRING;
private int cid;
double cost;
String fname;
static String company="jtc";
Course(){System.out.println("Course D.C");}
private Course(int cid, double cost, String fname) {
this.cid = cid;
this.cost = cost;
this.fname = fname;
}
static{System.out.println("Course S.B");}
{System.out.println("Course I.B");}
public int getCid() {
return cid;
}
public void setCid(int cid) {
this.cid = cid;
```

```

}
public double getCost() {
return cost;
}
public void setCost(double cost) {
this.cost = cost;
}
public String getFname() {
return fname;
}
public void setFname(String fname) {
this.fname = fname;
}
public static String getCompany() {
return company;
}
public static void setCompany(String company) {
Course.company = company;
}
public String toString(){
return “ “+cid+”\t”+cost+”\t”+fname+”\t”+company+”\t”+ordinal());
}
public enum Hello{
A,B
}
//abstract void m1();
public void m2() {
System.out.println(“m1()”);
}
public void m1() {
System.out.println(“m2()”);
}
}

```

jtc3.java

```

package com.jtc.jtc3;
public class jtc3 {
public static void main(String[] args) {

```

```

Course c1=Course.EJB;
System.out.println(c1);
for(Course c:Course.values()){
System.out.println(c);
}
//Course c2 = new Course(111,4500,"som");
c1=Course.SPRING;
System.out.println(c1);
c1.setCid(107);
c1.setCost(99911);
c1.setFname("pra");
System.out.println(c1);
System.out.println(Course.Hello.A);
Course.Hello h= Course.Hello.B;
System.out.println(h);
}
}

```

Example

```

public class Jtc14 {

public static void main(String[] args) {
Color col = null;
// col=new Color();
Color.RED.m1Color();
Color.BLACK.m1Color();
Color.BLUE.m1Color();
Color.WHITE.m1Color();
// switch (col) {} System.out.println("***
-- ENUM -- **");
System.out.println(Days.VAL);
Days d1 = null;
// d1=new Days();

```

```

Days.SUN.m1Days();
Days.MON.m1Days();
Days.TUE.m1Days();
Object obj = Days.SUN;
Enum ref = Days.MON;
System.out.println(Days.SUN);
System.out.println("\n\n----- COURSE -----");
Course c1 = Course.JAVA;
Course c2 = Course.JDBC;
Course c3 = Course.EJB;
Course c4 = Course.JSP;
System.out.println(c1.name() + "\t" + c1.ordinal());
System.out.println(c2.name() + "\t" + c2.ordinal());
System.out.println(c3.name() + "\t" + c3.ordinal());
System.out.println(c4.name() + "\t" + c4.ordinal());
System.out.println(c1);
System.out.println(c2);
System.out.println(c3);
System.out.println(c4);
System.out.println();
switch (c1) {
case JAVA:
System.out.println("Selected course information");
System.out.println(Course.JAVA);
break;
case JDBC:
System.out.println("Selected course information");
System.out.println(Course.JDBC);
break;
case EJB:
System.out.println("Selected course information");
System.out.println(Course.EJB);
break;
case JSP:
System.out.println("Selected course information");
System.out.println(Course.JSP);
break;
}

```

```

System.out.println("\n-- Method from java.lang.Enum --");
Course courses[] = Course.values();
for (Course c : courses) {
    System.out.println(c);
}
System.out.println();
String st = "JAVA";
Course cou = Course.valueOf(st);
System.out.println(cou);
}
}
class Color {
    public static final Color RED = new Color("Col-001", "RED");
    public static final Color BLUE = new Color("Col-002", "Blue");
    public static final Color BLACK = new Color("Col-003", "Black");
    public static final Color WHITE = new Color("Col-004", "White");
    String colorId;
    String colorName;
    private Color() {
        super();
    }
    private Color(String colorId, String colorName) {
        this.colorId = colorId;
        this.colorName = colorName;
    }
    void m1Color() {
        System.out.println("\n-- m1() in Color --");
        System.out.println(colorId + "\t" + colorName);
        Color c1 = new Color();
    }
}
interface Inter1 {
    void m2();
}
enum Days implements Inter1 {
    SUN, MON, TUE;
    static int VAL = 90;
    static {

```

```

System.out.println("-- Static Block in Days --");
}
{
System.out.println("*** Instance Block in Days ***");
}
Days() {
// super();
System.out.println("__ Days() Cons __");
}
public void m1Days() {
System.out.println("\n** m1Days() in Days enum***");
System.out.println("Name\t:" + name());
// Days d=new Days();
}
public void m2() {
System.out.println("-- M2 in Enum Days --");
}
}
enum E1 {}
// enum E2 extends E1{}
// enum E3 extends java.lang.Object{}
// enum E4 extends java.lang.Enum{}
enum Course {
JAVA(1, "2 Months", "SP"), JDBC(2, "7 Days", "Som"), EJB(3, "15 Days"), JSP(
4, "3 Days");
int id;
String duration;
String faculty = "SomPraksh";
static double fullCourseFee = 17000.0;
Course(int id, String duration, String faculty) {
this.id = id;
this.duration = duration;
this.faculty = faculty;
}
Course(int id, String duration) {
this.id = id;
this.duration = duration;
}
}

```

```
static void m1StaticMethod() {  
    System.out.println("-- STATIC method in ENUM--");  
}  
void m1CourseDetails() {  
    System.out.println("\n-- INSTANCE in ENUM--\t:" + this);  
    System.out.println("Course ID\t:" + id);  
    System.out.println("Duration\t:" + duration);  
    System.out.println("Faculty\t:" + faculty);  
}  
public String toString() {  
    return id + "\t" + name() + "\t" + duration + "\t" + faculty;  
}  
}
```