

What is language?

- Language is to communicate.
- Communication is the process of transformation data between objects.
- For this communication, we will use general languages such as English, telugu, hindi

What is computer language?

- It is pre-defined application software.
- It is standalone application.

Need of computer language?

1. Communication is possible using a language.
2. We need to communicate with the machine to perform complex operations in an easy way
3. Directly we cannot pass machine understandable instructions to communicate.
4. Hence we need to write programs using any high level language(source code)
5. Machine can't understand source code.
6. Compiler is the pre-defined program that translates Source code into Machine code.

C++ vs Java

There are many differences and similarities between C++ programming language and Java. A list of top differences between C++ and Java are given below:

Comparison Index	C++	Java
Platform-independent	C++ is platform-dependent.	Java is platform-independent.
Mainly used for	C++ is mainly used for system programming.	Java is mainly used for application programming. It is widely used in window, web-based, enterprise and mobile applications.
Goto	C++ supports goto statement.	Java doesn't support goto statement.
Multiple inheritance	C++ supports multiple inheritance.	Java doesn't support multiple inheritance through class. It can be achieved by interfaces in java.
Operator Overloading	C++ supports operator overloading.	Java doesn't support operator overloading.

Pointers	C++ supports pointers. You can write pointer program in C++.	Java supports pointer internally. But you can't write the pointer program in java. It means java has restricted pointer support in java.
Compiler and Interpreter	C++ uses compiler only.	Java uses compiler and interpreter both.
Call by Value and Call by reference	C++ supports both call by value and call by reference.	Java supports call by value only. There is no call by reference in java.
Structure and Union	C++ supports structures and unions.	Java doesn't support structures and unions.
Thread Support	C++ doesn't have built-in support for threads. It relies on third-party libraries for thread support.	Java has built-in thread support.
Documentation comment	C++ doesn't support documentation comment.	Java supports documentation comment (<code>/** ... */</code>) to create documentation for java source code.
Virtual Keyword	C++ supports virtual keyword so that we can decide whether or not override a function.	Java has no virtual keyword. We can override all non-static methods by default. In other words, non-static methods are virtual by default.
unsigned right shift >>>	C++ doesn't support >>> operator.	Java supports unsigned right shift >>> operator that fills zero at the top for the negative numbers. For positive numbers, it works same like >> operator.
Inheritance Tree	C++ creates a new inheritance tree always.	Java uses single inheritance tree always because all classes are the child of Object class in java. Object class is the root of inheritance tree in java.

History of java

Developed by	:	James Gosling
Vendor	:	Sun Micro System
Project name	:	Green Project
Type	:	open source & free software
Initial Name	:	OAK language
Present Name	:	java
Extensions	:	.java & .class & .jar
Initial version	:	jdk 1.0 (java development kit)
Present version	:	java 7 2011
Operating System	:	multi Operating System
Implementation Lang	:	c, cpp.....
Symbol	:	coffee cup with saucer
Objective	:	To develop web applications
SUN	:	Stanford Universally Network
Motto	:	WORA(write once run anywhere)



James Gosling initiated the Java language project in **June 1990** for use in one of his many **set-top box projects**. The language, initially **called Oak** after an oak tree that stood outside Gosling's office, also went by the name **Green** and ended up later being renamed as **Java**, from a list of random words.

Sun released the first public implementation as Java 1.0 in 1995. It promised **Write Once, Run Anywhere (WORA)**, providing no-cost run-times on popular platforms.

On **13 November 2006**, Sun released much of Java as free and open source software under the terms of the **GNU General Public License (GPL)**.

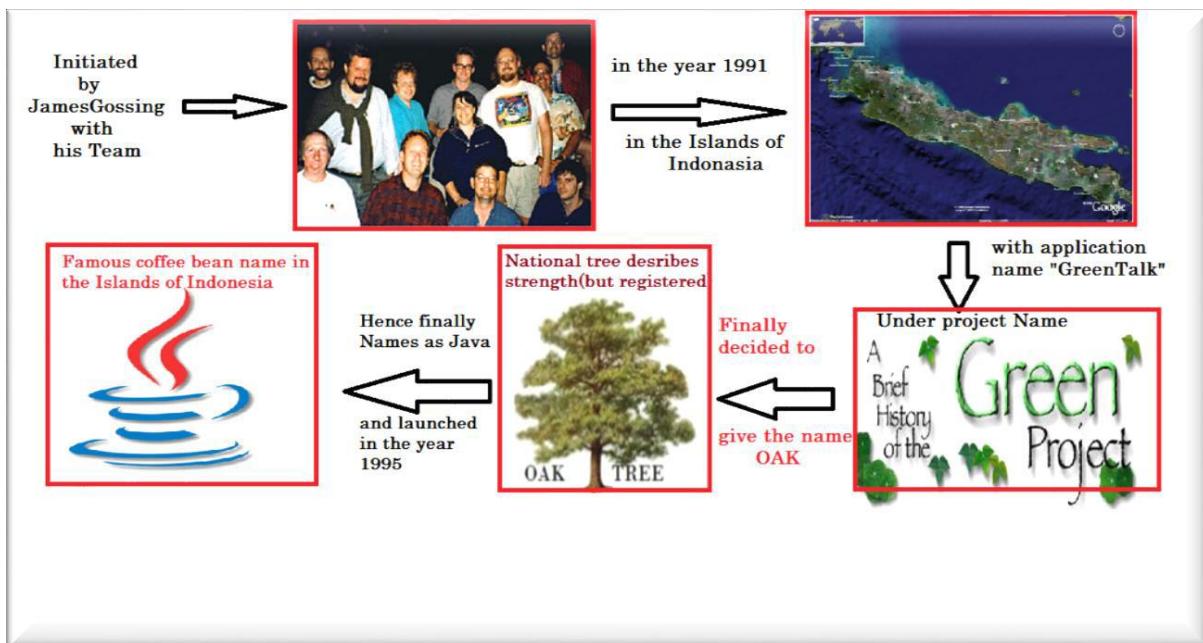
On **8 May 2007**, Sun finished the process, making all of Java's core code free and open-source, aside from a small portion of code to which Sun did not hold the copyright.

In 1990 --- Sun Micro systems INC, US → James Gauzing.

In 1993 --- Oak – It is completely system Independent.

Later on Oak name has changed. Its name is JAVA.

The Symbol of java is Tea cup.

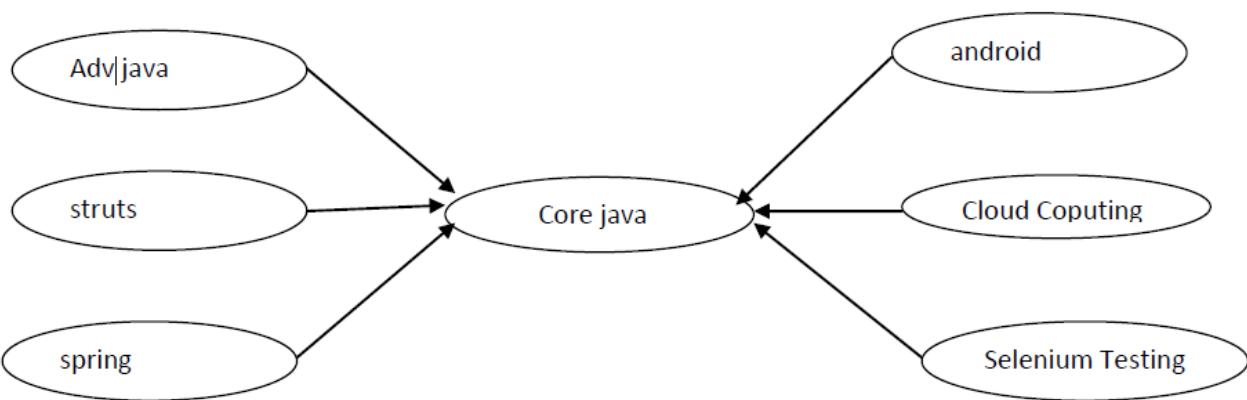


Java used at Where?

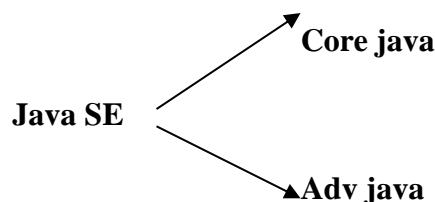
According to the SUN 3 billion devices run on the java language only.

- 1) Java is used to develop Desktop Applications such as MediaPlayer, Antivirus etc.
- 2) Java is Used to Develop Web Applications such as irstc.co.in etc.
- 3) Java is Used to Develop Enterprise Application such as Banking applications.
- 4) Java is Used to Develop Mobile Applications.
- 5) Java is Used to Develop Embedded System.
- 6) Java is Used to Develop SmartCards.
- 7) Java is Used to Develop Robotics.
- 8) Java is used to Develop Games etc.

Technologies Depends on Core java:-



Java SE: -Java Standard Edition



It deals with developing stand alone applications & also basic programs for the network.

Java EE: - Java Enterprise Edition

It Deals with developing business solutions on a network.

Java ME: - Java Micro Edition (or) Java Mobile Edition

It deals with developing embedded systems & wireless applications.

Types of Java Applications

There are mainly 4 type of applications that can be created using java programming:

1) Standalone Application

It is also known as desktop application or window-based application. An application that we need to install on every machine such as media player, antivirus etc. AWT and Swing are used in java for creating standalone applications.

2) Web Application

An application that runs on the server side and creates dynamic page, is called web application. Currently, servlet, jsp, struts, jsf etc. technologies are used for creating web applications in java.

3) Enterprise Application

An application that is distributed in nature, such as banking applications etc. It has the advantage of high level security, load balancing and clustering. In java, EJB is used for creating enterprise applications.

4) Mobile Application

An application that is created for mobile devices. Currently Android and Java ME are used for creating mobile applications

Java Version History

There are many java versions that has been released. Current stable release of Java is Java SE 8.

JDK VERSION	RELEASED IN	CODE NAME
1.0	January, 1996	-
1.1	February, 1997	Pumpkin
1.2(java 2)	December, 1998	Play ground
1.3(java 2)	May, 2000	Kestrel
1.4(java 2)	February, 2002	Merlin
Java 5	September, 2003	Tiger
Java 6	December, 2006	Mustang
Java 7	July 28, 2011	Dolphin
Java 8	18 march 2014	

Java Version SE 8

- Lambda Expressions
- New Collection Package `java.util.stream` to provide Stream API.
- Enhanced Security
- Nashorn Javascript Engine included
- Parallel Array Sorting
- The JDBC-ODBC Bridge has been removed etc.

Java Version SE 7

J2SE 1.7 is called as Dolphin and it is released on 28 July, 2011.

**Features**

- Strings in switch Statement
- Type Inference for Generic Instance Creation
- Multiple Exception Handling
- Support for Dynamic Languages
- Try with Resources
- Java nio Package
- Binary Literals, underscore in literals
- Diamond Syntax

- Automatic null Handling

Java Version SE 6

J2SE 1.2 is called as Mustang and it is released on 11 December, 2006.

**Features**

- Scripting Language Support
- JDBC 4.0 API
- Java Compiler API
- Pluggable Annotations
- Native PKI, Java GSS, Kerberos and LDAP support.
- Integrated Web Services.
- Lot more enhancements.

J2SE Version 5.0

J2SE 1.2 is called as Tiger and it is released on 30 September, 2004.



Features

- Generics
- Enhanced for Loop
- Autoboxing/Unboxing
- Typesafe Enums
- Varargs
- Static Import
- Metadata (Annotations)
- Instrumentation

J2SE Version 1.4

J2SE 1.2 is called as Merlin and it is released on 6 February, 2002.

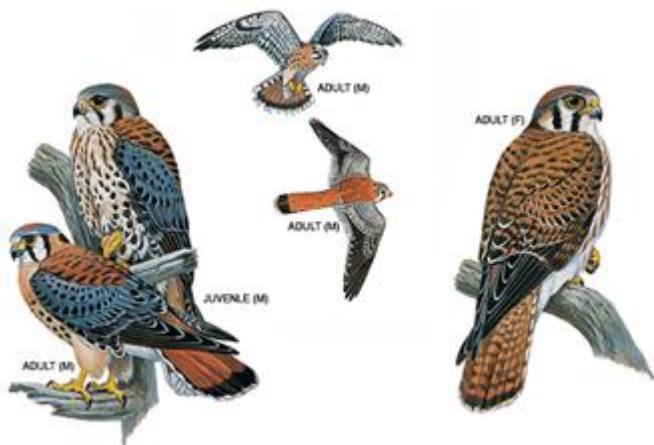


Features

- XML Processing
- Java Print Service
- Logging API
- Java Web Start
- JDBC 3.0 API
- Assertions
- Preferences API
- Chained Exception
- IPv6 Support
- Regular Expressions
- Image I/O API

J2SE Version 1.3

J2SE 1.2 is called as Kestrel and it is released on 8 May, 2000.

**Features**

- Java Sound
- Jar Indexing
- A huge list of enhancements in almost all the java area.

J2SE Version 1.2

J2SE 1.2 is called as playground and it is released on 8 December, 1998.

Features

- Collections framework.
- Java String memory map for constants.
- Just In Time (JIT) compiler.
- Jar Signer for signing Java ARchive (JAR) files.
- Policy Tool for granting access to system resources.
- Java Foundation Classes (JFC) which consists of Swing 1.0, Drag and Drop, and Java 2D class libraries.
- Java Plug-in
- Scrollable result sets, BLOB, CLOB, batch update, user-defined types in JDBC.
- Audio support in Applets.

JDK Version 1.1

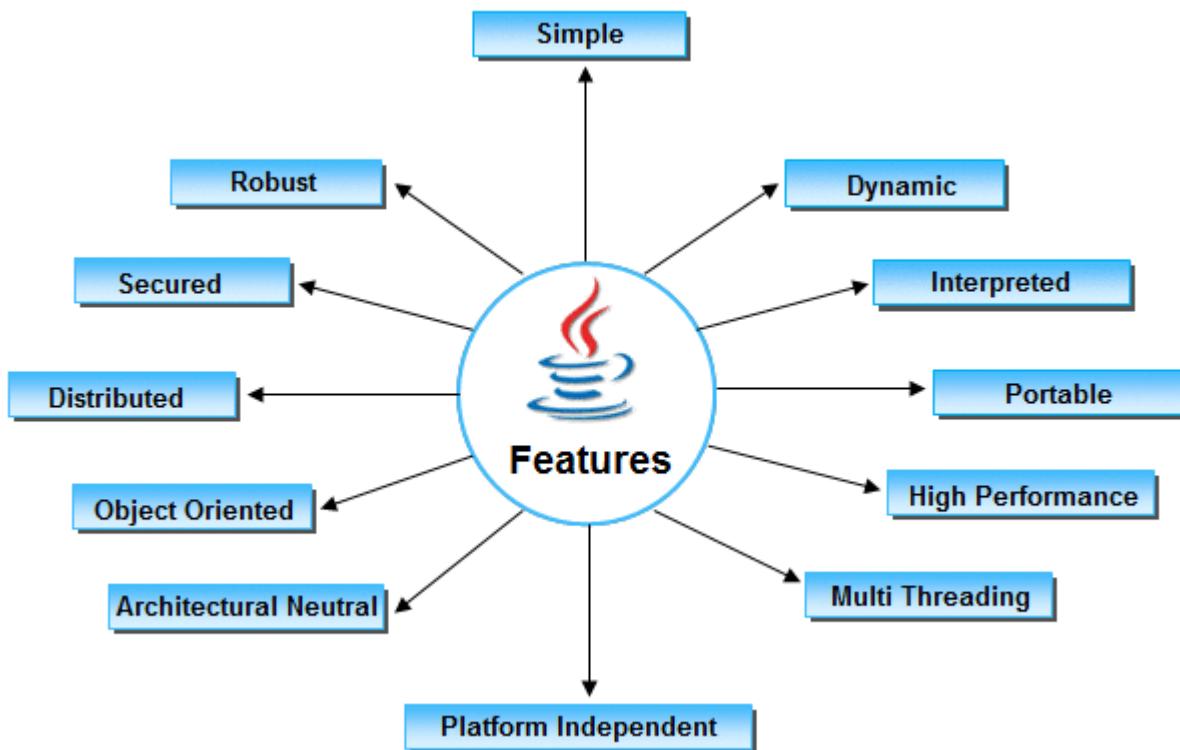
JDK 1.1 is released on 19 januray, 1997

Features

- JDBC (Java Database connectivity)
- Inner Classes
- Java Beans
- RMI (Remote Method Invocation)
- Reflection(introspection only)

JDK Version 1.0

JDK 1.0 is called as OAK, and it is released on 23 januray, 1996.

Features of java: -

- Simple:** - Java is a simple programming language. Learning & practicing java is easy because of its resemblance C and C++ pointers are not available on JAVA.

Why pointers are not available on java?

A) 1. Pointers lead to confusion for a programmer

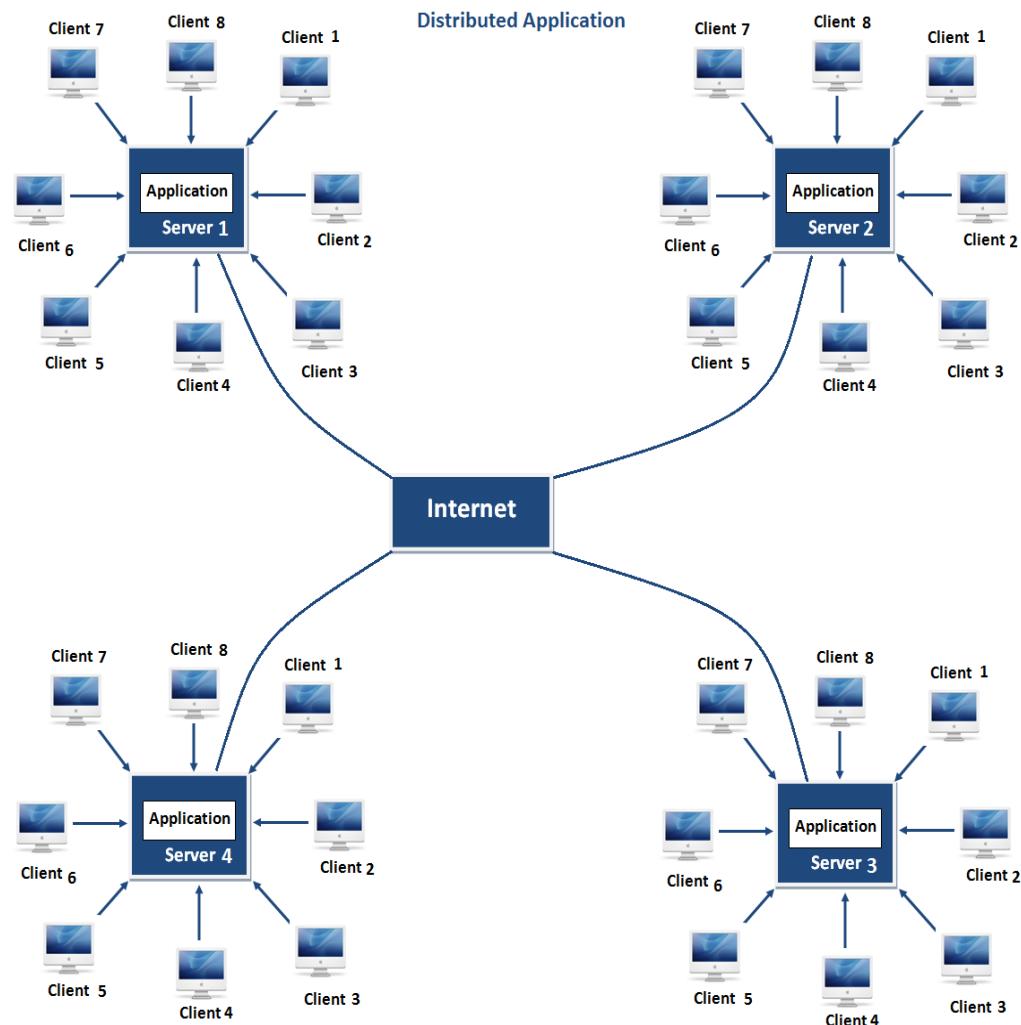
2. Pointers easily crash a program.
3. Using pointers virus & hacking programs can be written.

2. **Object Oriented:** - Java is a purely object-oriented programming language. Java programs use objects & classes.

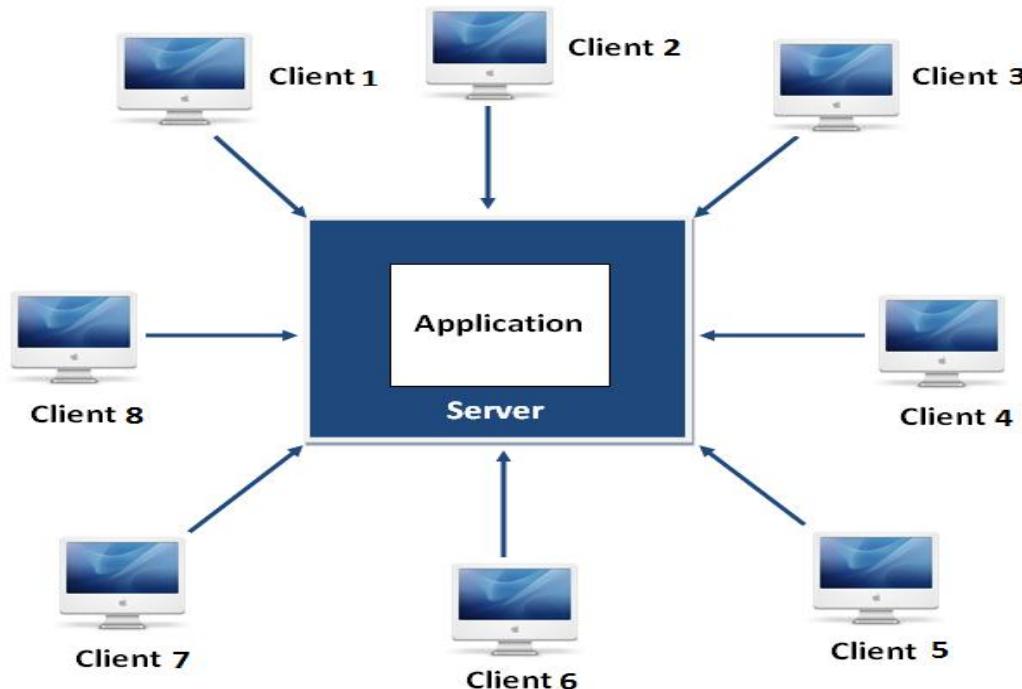
→Object: - Object is anything exists really in the world. An object contains properties and performs actions. Properties are represented by variables & actions are formed by functions. Functions are called methods in java.

→Class: - A class is a group of name that represents properties & actions of objects. Class does not exist physically. Where as object exist physically. A class is a plan or model to create objects, from the class to create the objects. In every java program we must contain at least one class.

3. Distributed: - Information is distributed on various on a network. Using java, we can write programs which capture information & distribute it to clients.



4.java is networked



4. Platform Independent

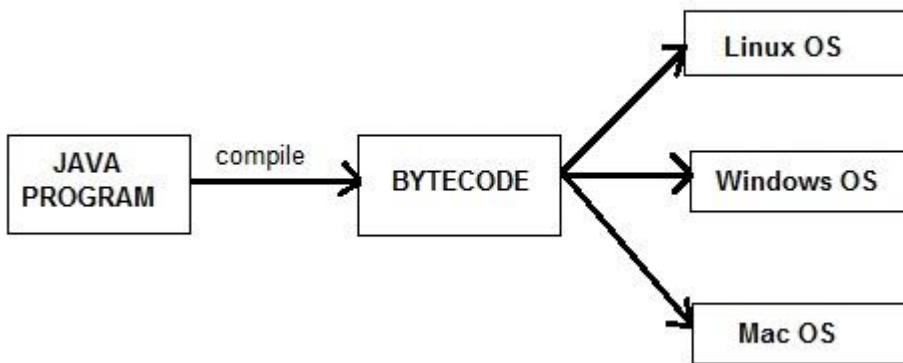
A platform is the hardware or software environment in which a program runs.

There are two types of platforms software-based and hardware-based. Java provides software-based platform.

The Java platform differs from most other platforms in the sense that it is a software-based platform that runs on the top of other hardware-based platforms. It has two components:

1. Runtime Environment
2. API(Application Programming Interface)

Java code can be run on multiple platforms e.g. Windows, Linux, Sun Solaris, Mac/OS etc. Java code is compiled by the compiler and converted into bytecode. This bytecode is a platform-independent code because it can be run on multiple platforms i.e. Write Once and Run Anywhere(WORA).



5.Robust: - Robust means strong. Java programs are strong. Java programs will not crash easily because of its execution handling and its memory management features. In C, C++ less or more than memory the programs are crash. In java this problem is not occurred, because JVM is allotted the memory.

→Memory allocation: - JVM'S class loader sub system

→Memory management: - JVM'S class garbage collection, this collector removes the unused memory. Exception means run time error. We can not handling the exception in 'C'. In case java we can handle smoothly

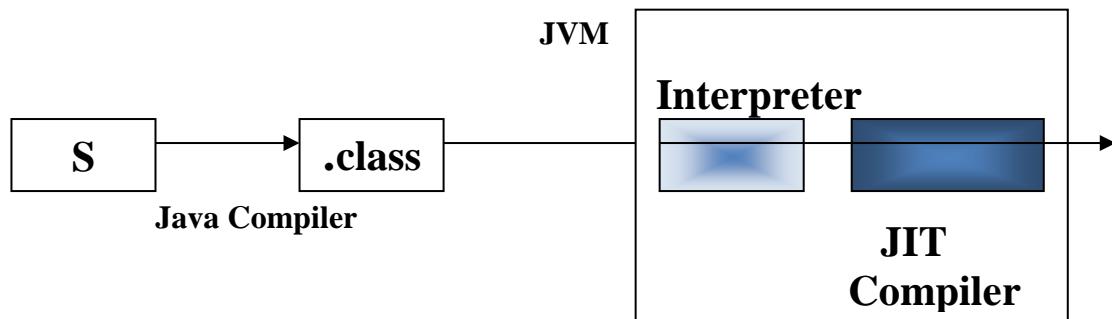
6.Secure: - Java enables the constructions of various free and tamper-free systems.

7.Architecture Neutral: - Java's byte code isn't machine dependent. It can be run any machine with any processor & with any operating system.

8.Portable: - Java programs give same results on all machines. Everything is clearly defined in java specification & nothing is left to o / s. If the Program is eliding give same results are called portability.

9.Interpreted: - Java programs are compiled to generate the byte code. This byte code can be downloaded & interpreted by the interpreter in JVM. In java we use interpreter. This interpreter is slow that's why problem is occurred

10.High performance: - Along with interpreter, there will be JIT (just in time) compiler which enhance the speed of execution.



11. Multi Threaded: - A thread is a process or execution. We can create different processing in java called 'threads'. This as an essential feature to design server side programs.

12. Dynamic: - We can develop programs in java which dynamically interact with the user on internet.

Ex: - Applets.

How to set path in Java

The path is required to be set for using tools such as javac, java etc.

If you are saving the java source file inside the jdk/bin directory, path is not required to be set because all the tools will be available in the current directory.

But If you are having your java file outside the jdk/bin folder, it is necessary to set path of JDK.

There are 2 ways to set java path:

1. temporary
2. permanent

1) How to set Temporary Path of JDK in Windows

To set the temporary path of JDK, you need to follow following steps:

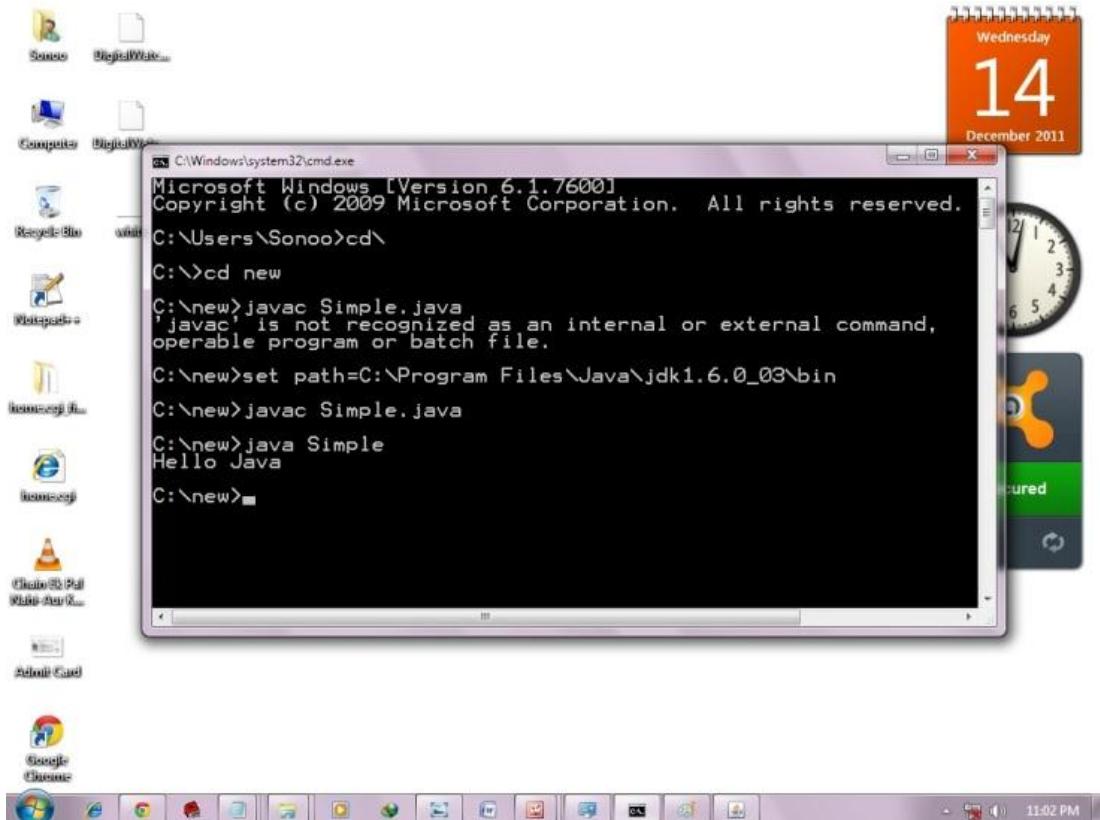
- Open command prompt
- copy the path of jdk/bin directory

- write in command prompt: set path=copied_path

For Example:

```
set path=C:\Program Files\Java\jdk1.8.0_25\bin
```

Let's see it in the figure given below:



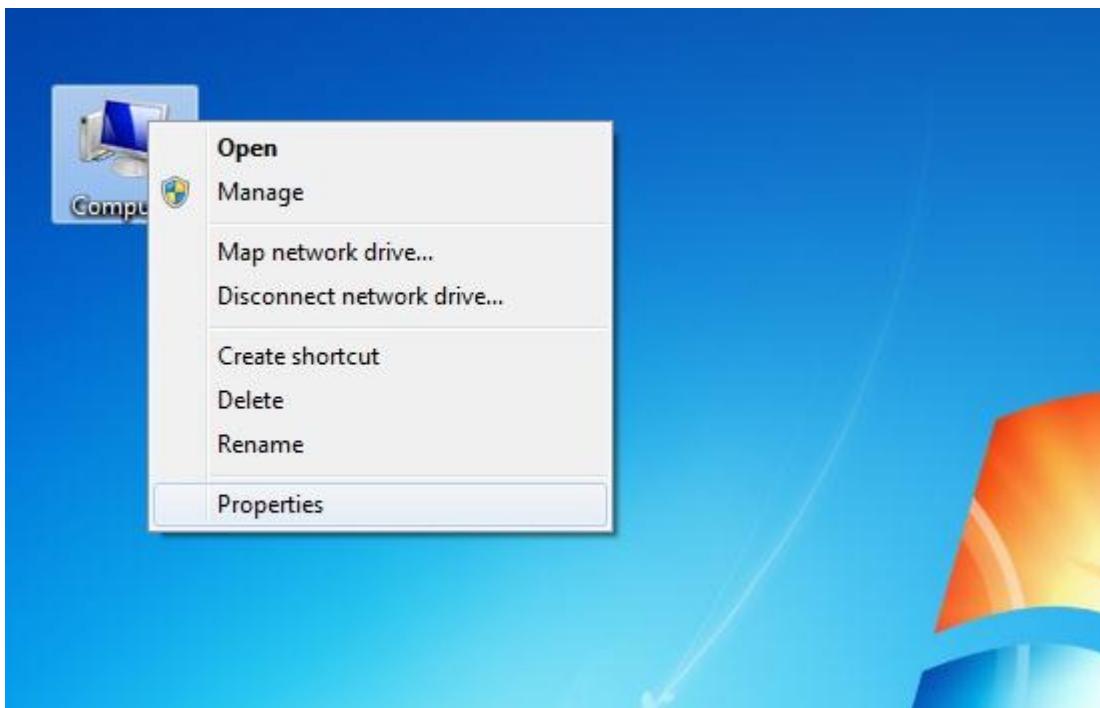
2) How to set Permanent Path of JDK in Windows

For setting the permanent path of JDK, you need to follow these steps:

- Go to MyComputer properties -> advanced tab -> environment variables -> new tab of user variable -> write path in variable name -> write path of bin folder in variable value -> ok -> ok -> ok

For Example:

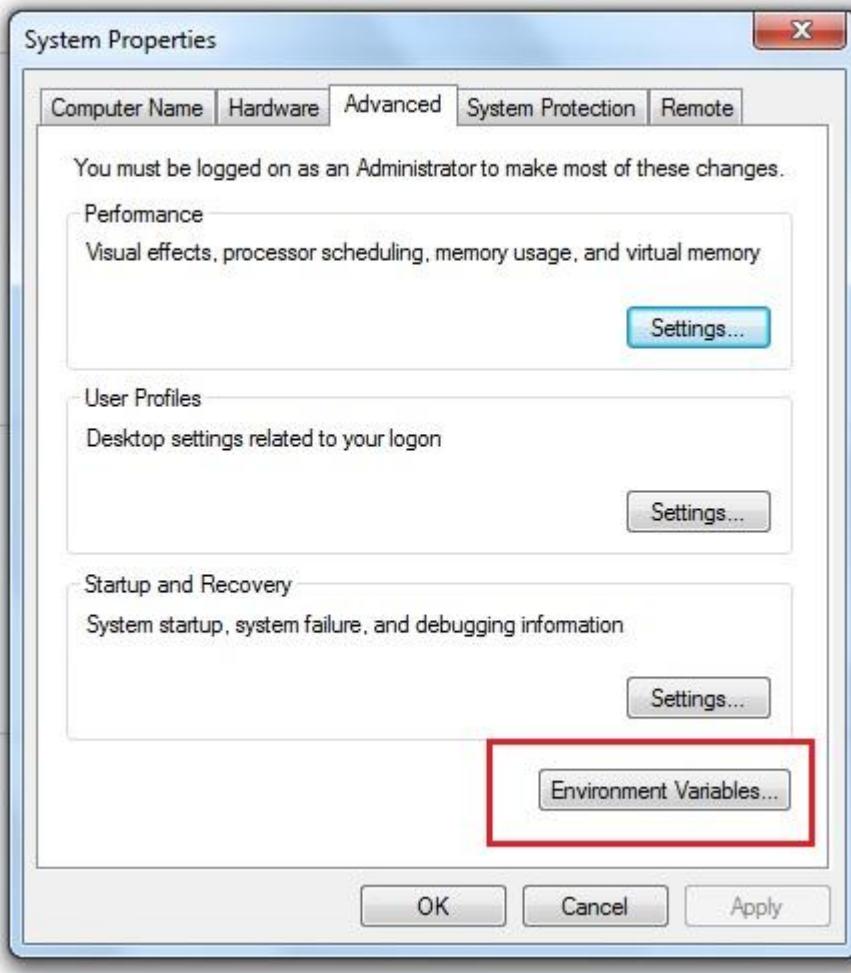
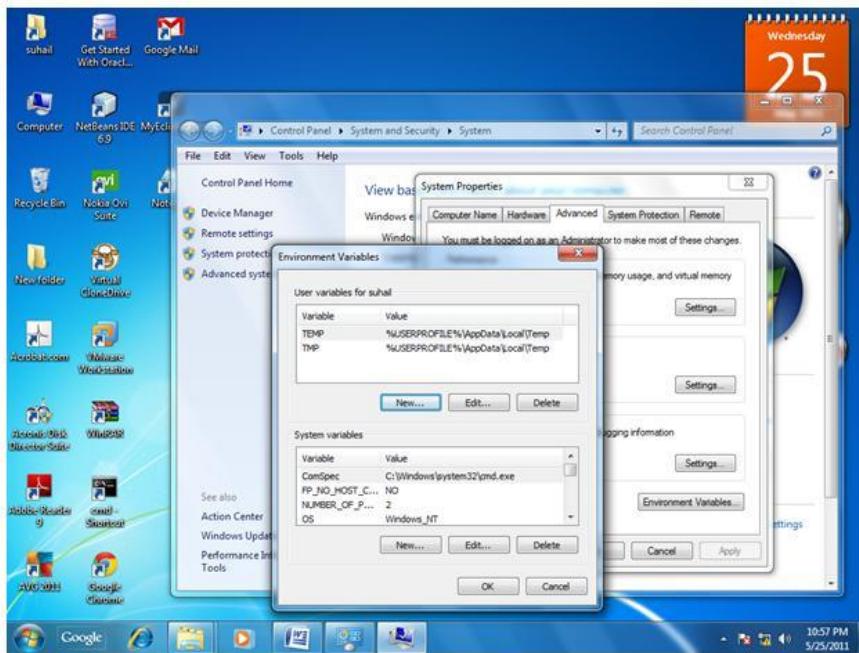
1) Go to MyComputer properties



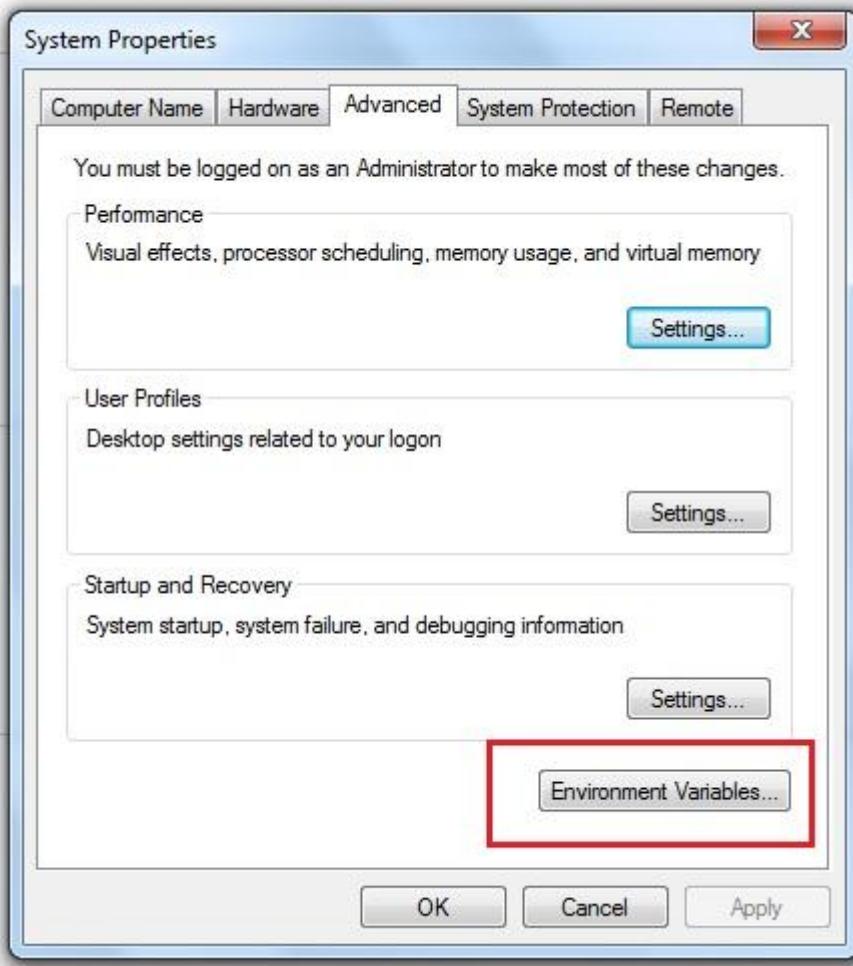
2)click on advanced tab



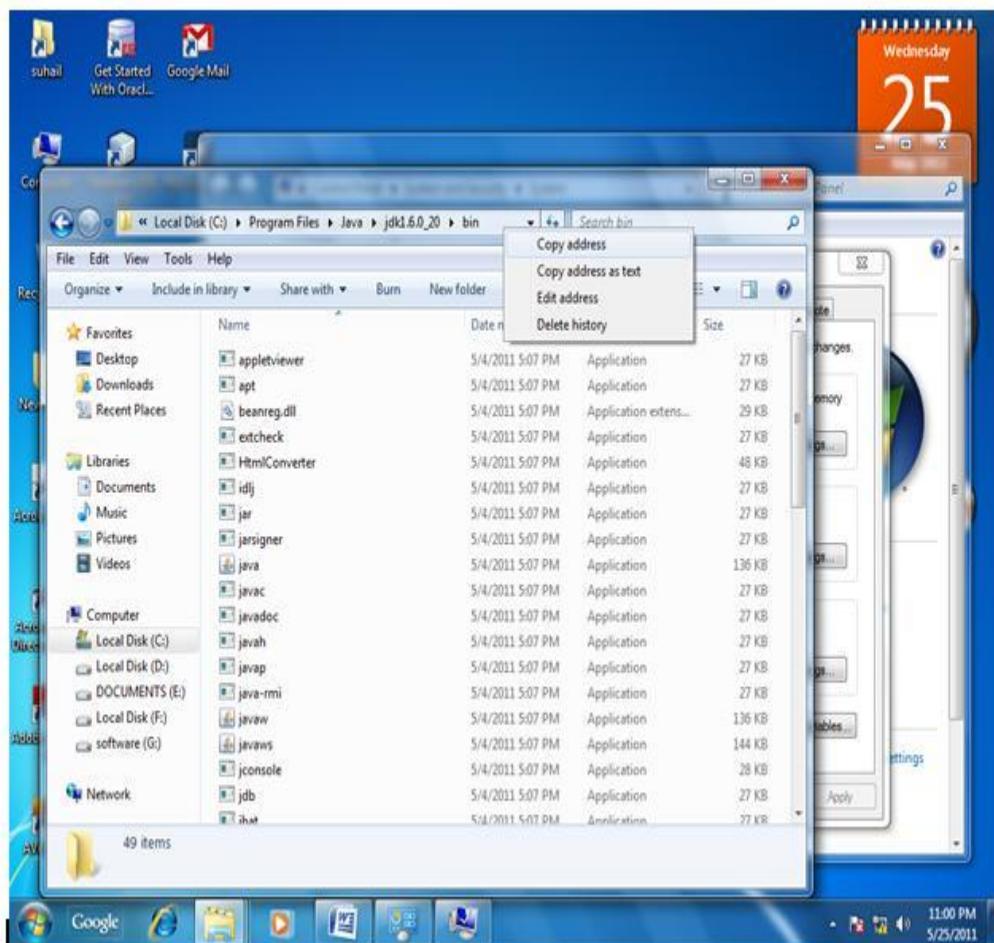
3)click on environment variables

**4)click on new tab of user variables**

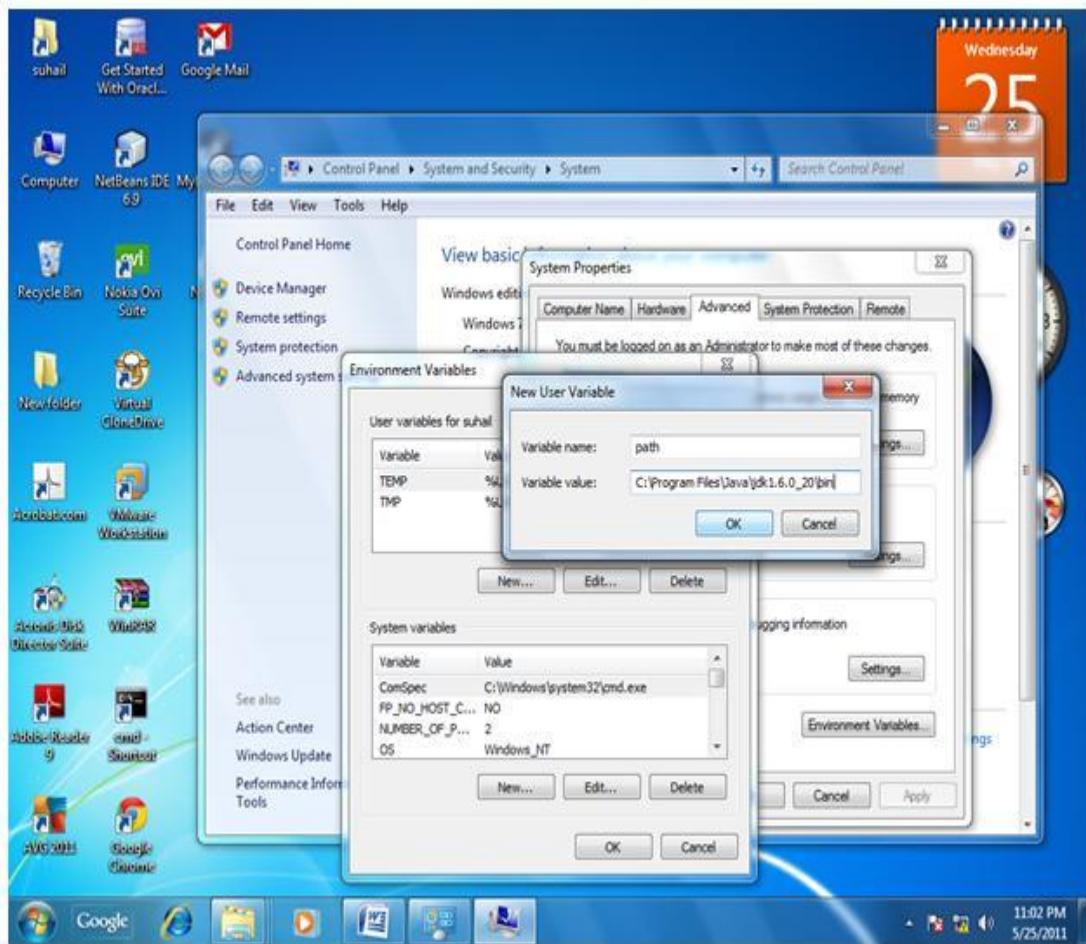
5)write path in variable name



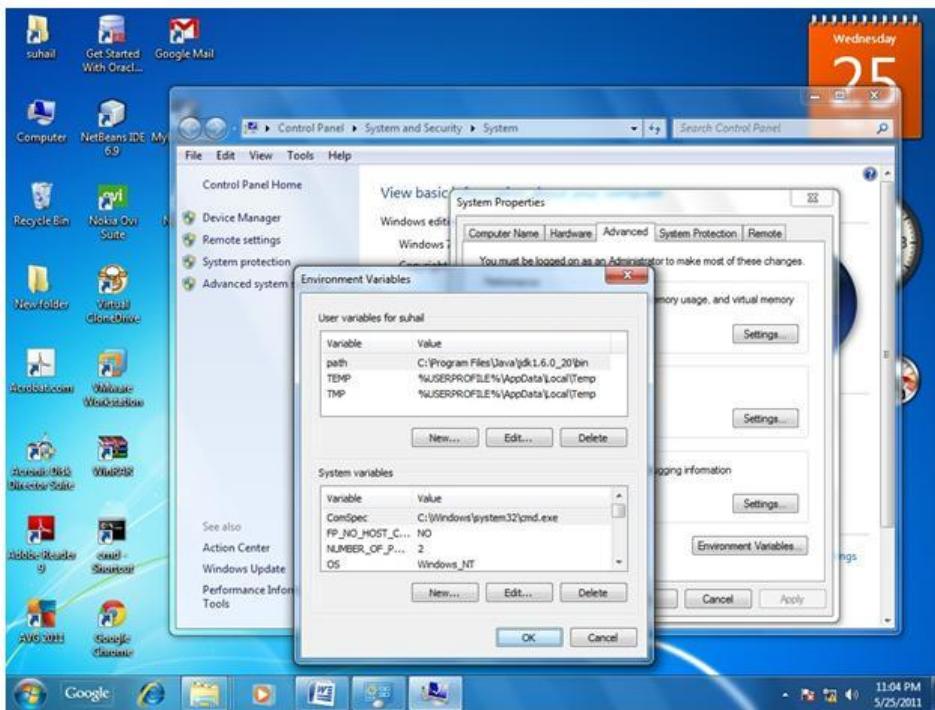
6)Copy the path of bin folder



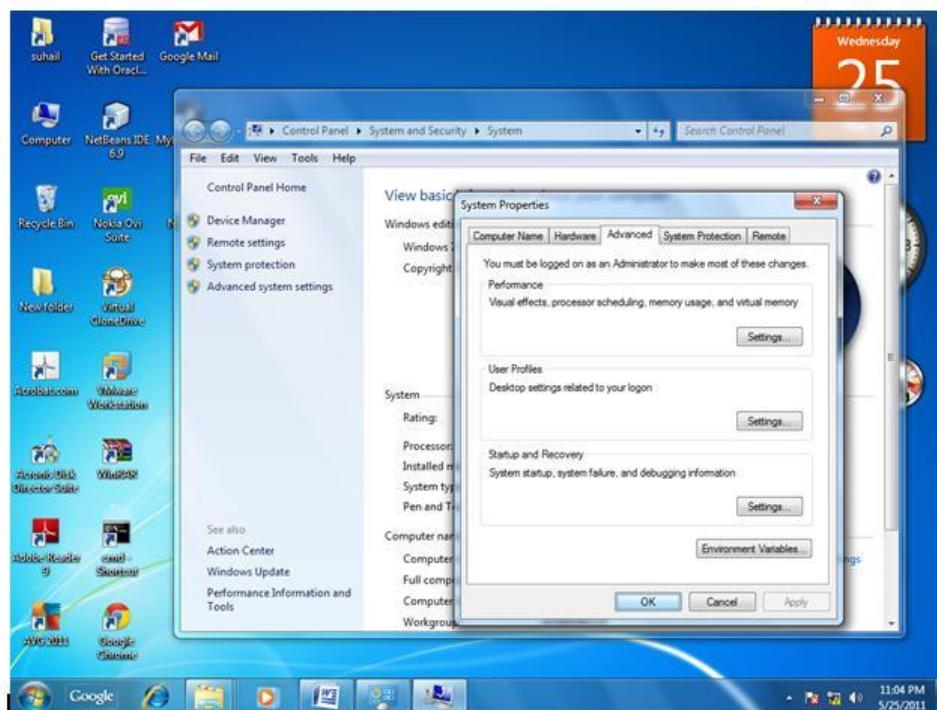
7) paste path of bin folder in variable value



8)click on ok button



9)click on ok button



Now your permanent path is set. You can now execute any program of java from any drive

What is the difference between a JDK and a JVM?

JDK is Java Development Kit which is for development purpose and it includes execution environment also. But JVM is purely a run time environment and hence you will not be able to compile your source files using a JVM.

What is a pointer and does Java support pointers?

Pointer is a reference handle to a memory location. Improper handling of pointers leads to memory leaks and reliability issues hence Java doesn't support the usage of pointers.

Q1. How to display and set the Class path in Unix ?

Ans. To display the current CLASSPATH variable, use these commands in UNIX (Bourne shell):

```
% echo $CLASSPATH
```

To delete the current contents of the CLASSPATH variable,

```
In UNIX: % unset CLASSPATH; export CLASSPATH
```

To set the CLASSPATH variable,

```
In UNIX: % CLASSPATH=/home/george/java/classes; export CLASSPATH
```

Difference between PATH and CLASSPATH ?

Ans. PATH is the variable that holds the directories for the OS to look for executables. CLASSPATH is the variable that holds the directories for JVM to look for .class files (Byte Code).

TOKENS:-

Smallest individual part in a java program is called Token. It is possible to provide any number of spaces in between two tokens.

Identifiers:-

Any name in the java program like variable name, class name, method name, interface name is called identifier

Class Naresh

```
{  
public static void main(String args[])  
{  
int regfee=100;  
}  
}
```

Rules to define java identifiers:

Rule 1:

The only allowed characters in java identifiers are:

- 1) a to z
- 2) A to Z
- 3) 0 to 9
- 4) _ (underscore)
- 5) \$

Rule 2:

If we are using any other character we will get compile time error.

Example:

- 1) stu_number-----valid
- 2) stu#-----invalid

Rule 3:

Identifiers are not allowed to start with digit.

Example:

- 1) ABC123-----valid
- 2) 123ABC-----invalid

Rule 4:

Java identifiers are case sensitive up course Java language itself treated as case sensitive language.

Example:

```
class Test{  
int number=10;  
int Number=20;  
int NUMBER=20;      we can differentiate with case.  
int NuMbEr=30;  
}
```

Rule 5:

There is no length limit for Java identifiers but it is not recommended to take more than 15 lengths.

Rule 6:

We can't use reserved words as identifiers.

Example:

int if=10; -----invalid

Rule 7:

All predefined Java class names and interface names we use as identifiers.

Comments:

Package:

Class

Interface

Enum;jdk 1.5 create set of named constsnts

Access specifier

Variables

Data types

Main method

Method

Constants

Keywords

Comments –

- Comments are description given by the programmer within the program. So that if another programmer looks at the programmer can easily understand the program by reading the comments.
- The characters or words or anything which are given in comments section won't be considered by java compiler for compilation process and will be ignored by java compiler during compilation.

Comments can be given in three ways inside a java program. They are,

1. Single line comment
2. Multi line comment
3. Documentation comments

1. SINGLE LINE COMMENT IN JAVA:

It starts with two forward slashes and continues to the end of the line.

Syntax:

```
// This comment extends to the end of the line.  
String welcome = "Hi"; //The text to print
```

2. MULTI LINE COMMENT IN JAVA:

Multi line comment starts with forward slash followed by an asterisk, and end with an asterisk followed by a forward slash.

Syntax:

```
/* This is a  
comment used for  
multiple lines */  
/* This comment can also be used for single line also */
```

3. DOCUMENTATION COMMENTS IN JAVA:

Documentation comments starts with a forward slash followed by two asterisks, and ends with an asterisk followed by a forward slash. This comment contains descriptions about the code and is used to create API document.

Syntax:

```
/** This  
documentation comment */
```

These comments are useful to create a HTML file called API (application programming Interface) document.

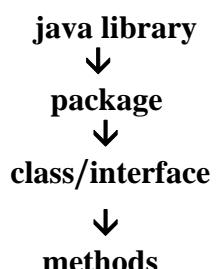
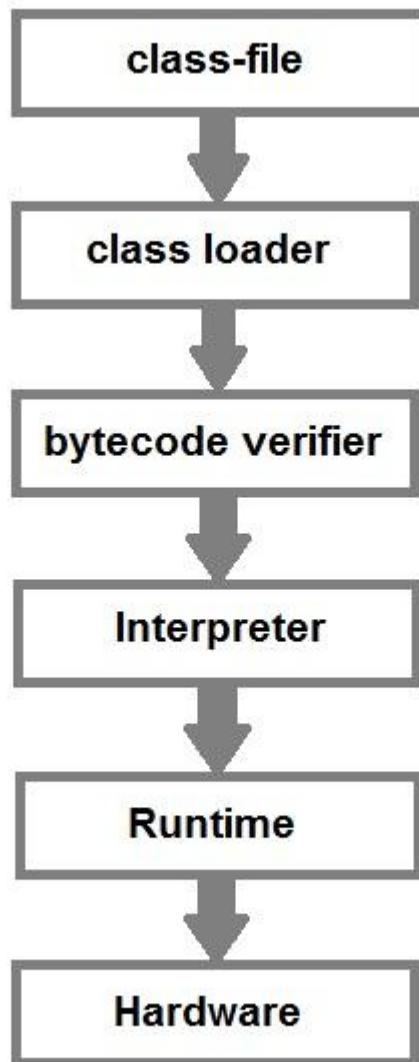
First program: -

```
→ /*this is my first java program  
To display a message  
Author: BALAKRISHNA REDDY  
Version: V1.0  
Project: WINGS */  
class Hello  
{  
    public static void main(String args[]){  
        {  
            // prints the string "Hello BALAKRISHNA REDDY" on screen  
            System.out.println("Hello BALAKRISHNA REDDY");  
        }  
    }  
}
```

D:\bkr\Core java>javac Hello.java

D:\bkr\Core java>java Hello

Hello BALAKRISHNA REDDY



Interface is similar to a class

```
java.lang(package)
  ↓
String.system(class)
```

```

import java.lang.String;
import java.lang.System;
Star (*) means all classes and interfaces.
→import java.lang.*;
```

→ **What is the difference between #include and import?**

A) #include makes a C or C++ compiler to go to standard library and copy the entire header file code in to a C/C++ programs. So the program size increases unnecessarily wasting memory & processor time.

Import statement makes the JVM to go to the java library, execute the code & substitute the result in to the java program. That's JVM doesn't copy any code. So import is more efficient than #include

→ **What is JRE?**

A) JRE means java run time environment.

JRE=JVM + java library

JRE is available in JDK1.5. It represents entire software.

public static void main(String args[])

JVM starts execution of a java program from main method. Main method is standing of the JVM. The main method is not, the execution will be stopped.

- A method can receive data from outside
- A method can also return some result.

[] → A group of elements. We can store several strings in to args x.

[] called array.

Void means no value. Main method does not return any value.

→ **What are command line arguments?**

A) The values passed to main method at the time of running a program or called command line arguments. Its stores in array.

Calling method in java:-

1. Create an object to the class
Classname obj=new Classname();
2. Using object name .method name called the method
Obj.method();

Static: - A static method is a method that can be called can without using any object.

Public: - Public members are available to out side programs

System.out → It print stream class.

System is a class. Out (field) is a variable.

Variable represent the memory location to store the data.

→ Print method displays the results and then keeps the cursor in the same line.

→ println method also display the result. After displaying the result it throws the cursor to the next line.

Comments are not executable.

Valid java main method signature

```
public static void main(String[] args)
public static void main(String []args)
public static void main(String args[])
public static void main(String... args)
static public void main(String[] args)
public static final void main(String[] args)
final public static void main(String[] args)
final strictfp public static void main(String[] args)
```

Invalid java main method signature

```
public void main(String[] args)
static void main(String[] args)
public void static main(String[] args)
abstract public static void main(String[] args)
```

Escape sequence

	<u>Meaning</u>
\n	next line
\t	<tab>
\v	<enter>
\b	<back space>
\\\	\
\\\"	"
\\'	'

Every java program every statement must end with a semicolon (;).

Java is a case sensitive language.

Naming conventions in java (Rules): -

- 1) Package names in java are written in all small letters

EX: - java.awt;

java.io;

javax.swing;

- 2) Each word of a class name and interface name start with a capital letter.

EX: - String, DataInputStream, ActionListener

- 3) Method names starts with a small letter, then each word starts with a capital letter.

EX: - readLine(); , getInstance();

- 4) Variable names also follow the above rules.

EX: - age, empName, employee_Net_Sal;

- 5) Constants should be written using all capital letters.

EX: - PI, MAX-VALUE, Font.BOLD;

Here Font is a class, BOLD is a constant.

- 6) All key words should be written in all small letters.

EX: - public, static, void, main etc...

→ java.lang package by default import in to every java program.

Important packages of core java: -

- 1) java.lang: - This package got primary classes and interfaces essential for java program. It consists of wrapper classes, strings, threads, etc. wrapper classes are useful to create an object. Thread means process.
- 2) java.util: - (utility) in this util data structures are available. This package contains useful classes and interfaces like stack, linked list, hash table, arrays. etc.
- 3) java.io: - This package handles files and input output related tasks. (io – input output).
- 4) java.awt : - abstract window tool kit.

This package helps to develop GUI. It consists of important sub packages.

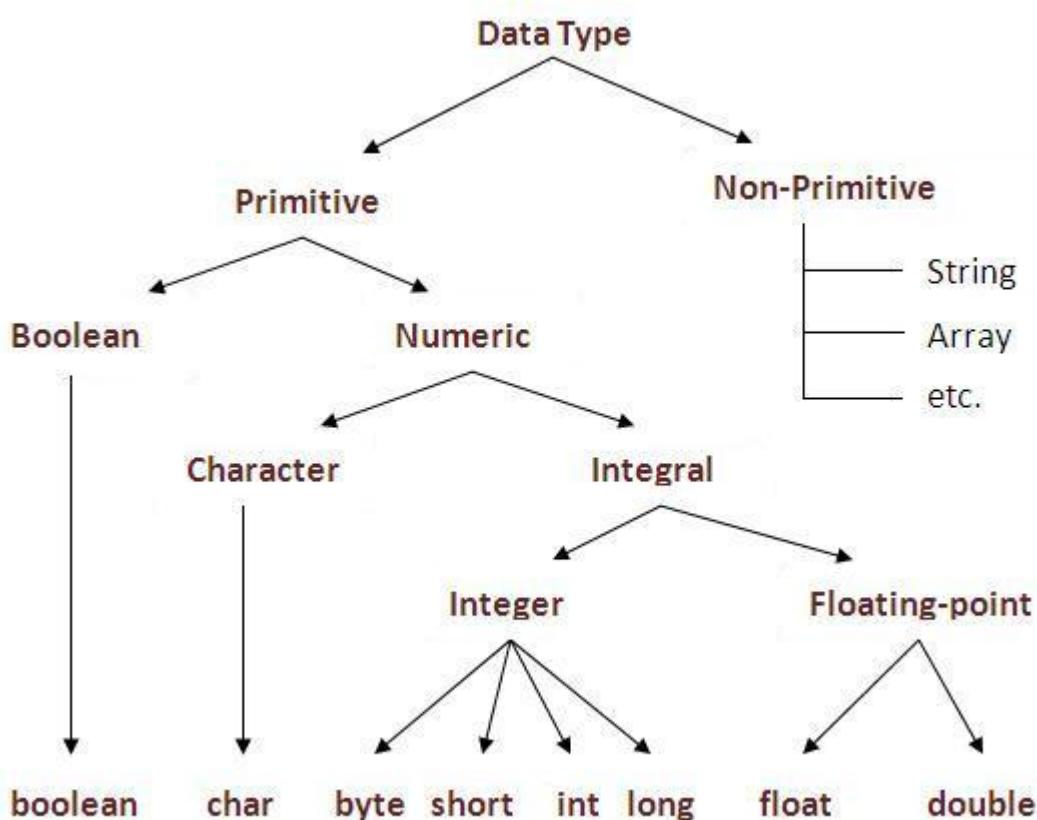
Java.awt.event → event is useful to providing actions.

GUI→ graphics user interface.

- 5) **javax.swing:** - This package helps to develop GUI like java.awt. X means extended. This package is developed from another package.
 - 6) **java.net:** - Client – server programming can be done using this package. Net stands for network.
 - 7) **java.applet:** - applets are programs which come from a server into a client and get executed on the client machine. An applet is a program that comes to the internet server.
 - 8) **java.sql:** - structured query language. This package helps to connect to database like oracle and utilize them in java. A data base is a storage data.
- A variable is memory location to store data. Int represent you can store integer numbers. Numbers are without decimal points. It is a data type.
- A data type represents the type of data stored into variable.

Data types and literals:-

A data type represents the type of data stored in variable.



Integer data types:

<u>Data type</u>	<u>Memory size</u>	<u>min and max value</u>
1. byte	1 Byte	- 128 to +127
2. short	2 Bytes	- 32768 to +32767
3. int	4 Bytes	- 21474836 to +2147483647
4. long	8 Bytes	-9223372036854775808 to + 9223372036854775807

These data types represent integer numbers

For example 0, 15000, 125, and -15 etc....

'0' is stored in +ve range. So 1 number is less.

'0' comes under the numbers to add.

Ex: - byte rno = 10; means literal.

long x = 150L;

A literal represents the value directly stored in to a variable.

This L is writing the JVM will allot of 8 bytes otherwise the JVM will allot 2 bytes.

Float data types: - Float means floating point numbers. Float data type represent numbers with decimal points.

Ex: - 1.5, 3.14159, -2.2, 100.0 etc....

Float means single precision floating point number. Double means double precision floating point number. Precision means accuracy.

<u>Data type</u>	<u>Memory size</u>	<u>min and max value</u>
1) float	4 bytes	1.4e-45 to +3.4e38.
2) double	8 bytes	4.1e-324 to +1.8e308.

e means into 10 powers. (x 10power).

Ex:-

$$1.4\text{e}-45 = 1.4 \times 10^{-45}$$

```
Float      pi      =      3.142f;  
(Data type) (Variable)      (Literal))  
double    distance  =  1.98e8;
```

Floating Point Data types:

Float	double
If we want to 5 to 6 decimal places of accuracy then we should go for float.	If we want to 14 to 15 decimal places of accuracy then we should go for double.
Size:4 bytes.	Size:8 bytes.
Range:-3.4e38 to 3.4e38.	-1.7e308 to 1.7e308.
float follows single precision.	double follows double precision.

→ What is the distance between a float and double?

A). Float represent up to 7 digits accurately after decimal point.

Double represent up to 15 digits accurately after decimal point.

Character data type: -

Data type	Memories size	min and max value
char	2 bytes	0 to 65535

This data type represents a single character. In C or CPP languages using only one byte in a char.

Ex: - char ch = 'x';

Single character literal you writing put them inside ''(single code).

You write string put the double codes ("").

ASCII: - American Standard Code for Information Interchange.

It is a standard. It should

followed by all keyboard manufactures.

114 keys are using in keyboard. 230 characters are there in our keyboard approximately (small & large).

Summary of java primitive data type:

data type	Size	Range	Corresponding Wrapper class	Default value
Byte	1 byte	-27 to 27-1(-128 to 127)	Byte	0
Short	2 bytes	-215 to 215-1 (-32768 to 32767)	Short	0
Int	4 bytes	-231 to 231-1 (-2147483648 to 2147483647)	Integer	0
Long	8 bytes	-263 to 263-1	Long	0
Float	4 bytes	-3.4e38 to 3.4e38	Float	0.0

double	8 bytes	-1.7e308 to 1.7e308	Double	0.0
boolean	Not applicable	Not applicable(but allowed values true false)	Boolean	false
Char	2 bytes	0 to 65535	Character	0(represents blank space)

→ **What is uni code?**

A) Uni code is a standard to include the alphabet from all human languages in java. Uni code system uses 2 bytes to represent a character.

String data types: - A string represent group of characters.

EX: -

```
String name="BALAKRISHNA REDDY";
String str=new String("BALAKRISHNA REDDY");
```

Boolean data types: - These data types represent only 2 values either a true or false.

EX: - boolean response=true;

Type Casting

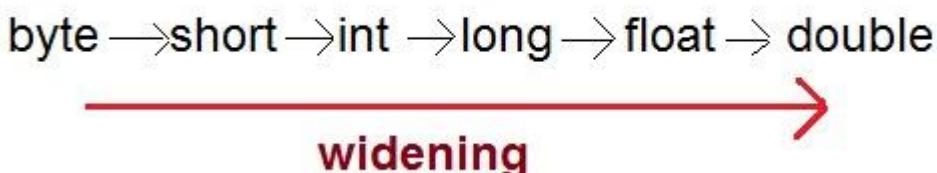
Assigning a value of one type to a variable of another type is known as **Type Casting**.

Example :

```
int x = 10;
byte y = (byte)x;
```

In Java, type casting is classified into two types,

- Widening Casting(Implicit)



- Narrowing Casting(Explicitly done)



Widening or Automatic type conversion

Automatic Type casting take place when,

- the two types are compatible
- the target type is larger than the source type

Example :

```
public class Test  
{  
    public static void main(String[] args)  
    {  
        int i = 100;  
  
        long l = i; //no explicit type casting required  
  
        float f = l; //no explicit type casting required  
  
        System.out.println("Int value "+i);  
  
        System.out.println("Long value "+l);  
  
        System.out.println("Float value "+f);  
    }  
}
```

Narrowing or Explicit type conversion

When you are assigning a larger type value to a variable of smaller type, then you need to perform explicit type casting.

```
public class Test
{
    public static void main(String[] args)
    {
        double d = 100.04;
        long l = (long)d; //explicit type casting required
        int i = (int)l;      //explicit type casting required

        System.out.println("Double value "+d);
        System.out.println("Long value "+l);
        System.out.println("Int value "+i);
    }
}
```

Reserved words:

These words already defined by java s/w developers.

Reserved words for data types: (8)

- 1) byte
- 2) short
- 3) int
- 4) long
- 5) float
- 6) double
- 7) char
- 8) boolean

Reserved words for flow control:(11)

- 1) if
- 2) else
- 3) switch
- 4) case
- 5) default
- 6) for
- 7) do
- 8) while
- 9) break
- 10) continue
- 11) return

Keywords for modifiers:(11)

- 1) public
- 2) private
- 3) protected
- 4) static
- 5) final
- 6) abstract
- 7) synchronized
- 8) native
- 9) strictfp(1.2 version)
- 10) transient
- 11) volatile

Keywords for exception handling:(6)

- 1) try
- 2) catch
- 3) finally
- 4) throw
- 5) throws
- 6) assert(1.4 version)

Class related keywords:(6)

- 1) class
- 2) package
- 3) import
- 4) extends
- 5) implements
- 6) interface

Object related keywords:(4)

- 1) new
- 2) instanceof
- 3) super
- 4) this

VARIABLES in java

Variable is a placeholder (reserved memory block) to store a value of any type. A variable is defined by its name (identifier), type and initialization is optional.

SYNTAX FOR VARIABLE DECLARATION:

Datatype variable_name [= value];

Two or more variables of same type can also be declared in a single line separated by commas.
Example:

```
Inta=5;
floata,b;
int i = 0, j =1;
```

TYPES OF VARIABLES IN JAVA:

Java has 3 kinds of variables. They are

Local variables

Instance variables (fields)

Static variables (class variables)

1. LOCAL VARIABLES:

A local variable is defined inside a method block. A block begins with an opening curly brace and ends with a closing curly brace.

The scope of the variable is limited within the block. In other words, local variables are visible only in the block (method) in which they are declared.

2. INSTANCE VARIABLES:

Instance variables are declared inside a class, but outside of any method / constructor / any block. They are also referred to as fields.

Objects store their individual states in these fields. Their values are unique to each object (instance of class) and hence they are called as instance variables.

Example – The ‘employeeId’ field of the Employee class will have a unique value for each of its object. (Consider Emp1, Emp2... as objects of the class Employee, then each object will have unique value for the property ‘employee-id’).

3. STATIC VARIABLES (CLASS VARIABLES):

Static variables belong to the class rather than objects in which they are declared. The keyword ‘static’ is prefixed before the variable to represent static variables. Only one copy of this variable is maintained for all the objects.

Example – static int width; would now indicate that the width of all the boxes (b1,b2... -instances of the Box class) would have the same value. One one copy of variable ‘width’ is maintained by all the objects of class Box.

Rules to declare a Variable

Every variable name should start with either alphabets or underscore (_) or dollar (\$) symbol. No space are allowed in the variable declarations.

Except underscore (_) no special symbol are allowed in the middle of variable declaration

Variable name always should exist in the left hand side of assignment operators.

Maximum length of variable is 64 characters.

No keywords should access variable name.

“A variable is an identifier whose value will be changed during execution of the program”.

Rules for writing variables:

- i. First letter must be an alphabet.
- ii. The length of the variable should not exceed more than 32 characters.
- iii. No special symbols are allowed except underscore.
- iv. No keywords should use as variable names.

Types of variables in JAVA:

- Whenever we develop any JAVA program that will be developed with respect to class only.
- In a class we can use ‘n’ number of data members and ‘n’ number of methods.
- Generally in JAVA, we can use two types of data members or variables. They are instance/non-static variables and static variables.

CONSTANTS in java

“Constant is an identifier whose value cannot be changed during execution of the program”.

In JAVA to make the identifiers are as constants, we use a keyword called final.

Final is a keyword which is playing an important role in three levels. They are at variable level, at method level and at class level.

When we don’t want to change the value of the variable, then that variable must be declared as final.

Literals:

Any constant value which can be assigned to the variable is called literal.

Example:

Integral Literals:

For the integral data types (byte, short, int and long) we can specify literal value in the following ways.

1)Decimal literals: Allowed digits are 0 to 9.

Example: int x=10;

2)Octal literals: Allowed digits are 0 to 7. Literal value should be prefixed with zero.

Example: int x=010;

3)Hexa Decimal literals:

The allowed digits are 0 to 9, A to Z.

For the extra digits we can use both upper case and lower case characters.

This is one of very few areas where java is not case sensitive.

Literal value should be prefixed with ox(or)oX.

Example: int x=0x10;

These are the only possible ways to specify integral literal.

Which of the following are valid declarations?

```
int x=0777; //(valid)
int x=0786; //C.E:integer number too large: 0786(invalid)
```

```
int x=0xFACE; (valid)
int x=0xbeef; (valid)
int x=0xBeer; //C.E:';' expected(invalid) //:int x=0xBeer; ^// ^
int x=0xabb2cd;(valid)
```

Example:

```
int x=10;
int y=010;
int z=0x10;
System.out.println(x+"----"+y+"----"+z); //10----8----16
```

By default every integral literal is int type but we can specify explicitly as long type by suffixing with small "L" (or) capital "L".

Example:

```
int x=10;(valid)
long l=10L;(valid)
long l=10;(valid)
int x=10l;//C.E:possible loss of precision(invalid)
           found : long
           required : int
```

There is no direct way to specify byte and short literals explicitly. But whenever we are assigning integral literal to the byte variables and its value within the range of byte compiler automatically treats as byte literal. Similarly short literal also.

Example:

```
byte b=127;(valid)
byte b=130;//C.E:possible loss of precision(invalid)
short s=32767;(valid)
short s=32768;//C.E:possible loss of precision(invalid)
```

Floating Point Literals:

Floating point literal is by default double type but we can specify explicitly as float type by suffixing with f or F.

Example:

```
float f=123.456;//C.E:possible loss of precision(invalid)
float f=123.456f;(valid)
double d=123.456;(valid)
```

We can specify explicitly floating point literal as double type by suffixing with d or D.

Example:

```
double d=123.456D;
```

We can specify floating point literal only in decimal form and we can't specify in octal and hexadecimal forms.

Example:

```
double d=123.456;(valid)
double d=0123.456;(valid) //it is treated as decimal value but not octal
double d=0x123.456;//C.E:malformed floating point literal(invalid)
```

Which of the following floating point declarations are valid?

```
float f=123.456; //C.E:possible loss of precision(invalid)
```

```
float f=123.456D; //C.E:possible loss of precision(invalid)
double d=0x123.456; //C.E:malformed floating point literal(invalid)
double d=0xFace; (valid)
double d=0xBeef; (valid)
```

We can assign integral literal directly to the floating point data types and that integral literal can be specified in decimal , octal and Hexa decimal form also.

Example:

```
double d=0xBeef;
System.out.println(d); //48879.0
```

But we can't assign floating point literal directly to the integral types.

Example:

```
int x=10.0;//C.E:possible loss of precision
```

We can specify floating point literal even in exponential form also(significant notation).

Example:

```
double d=10e2;//==>10*102(valid)
System.out.println(d); //1000.0
float f=10e2;//C.E:possible loss of precision(invalid)
float f=10e2F;(valid)
```

Boolean literals:

The only allowed values for the boolean type are true (or) false where case is important.
i.e., lower case

Example:

```
boolean b=true;(valid)
boolean b=0;//C.E:incompatible types(invalid)
boolean b=True;//C.E:cannot find symbol(invalid)
boolean b="true";//C.E:incompatible types(invalid)
```

Char literals:

1) A char literal can be represented as single character within single quotes.

Example:

```
char ch='a';(valid)
char ch=a;//C.E:cannot find symbol(invalid)
char ch="a";//C.E:incompatible types(invalid)
char ch='ab';//C.E:unclosed character literal(invalid)
```

2) We can specify a char literal as integral literal which represents Unicode of that character.
We can specify that integral literal either in decimal or octal or hexadecimal form but allowed values range is 0 to 65535.

Example:

```
char ch=97; (valid)
char ch=0xFace; (valid)
System.out.println(ch); //?
char ch=65536; //C.E: possible loss of precision(invalid)
```

3) We can represent a char literal by Unicode representation which is nothing but '\uxxxx' (4 digit hexa-decimal number) .

Example:

```
char ch='\\ubeef';
char ch1='\\u0061';
System.out.println(ch1); //a
char ch2=\\u0062; //C.E:cannot find symbol
char ch3='\\iface'; //C.E:illegal escape character
Every escape character in java acts as a char literal.
```

Syntax for FINAL VARIABLE INITIALIZATION:

Final data type v1=val1, v2=val2 ... vn=valn;

For example:

```
Final int a=10;
a=a+20; //invalid
a=30; //invalid
```

ii. When the final variable is initialized, no more modifications or assignments are possible.

Syntax for FINAL VARIABLE DECLARATION:

Final data type v1, v2.....vn;

For example:

```
Final int a;
a=a+1; //invalid
a=30+2; //invalid
a=400; //valid for 1st time
a=500; //invalid
```

Whenever a final variable is declared first time assignment is possible and no more modification and further assignments are not possible. Hence, final variables cannot be modified.

JVM Architecture: - (Black diagram of JVM)

Java virtual Machine(JVM) is a virtual Machine that provides runtime environment to execute java byte code. The JVM doesn't understand Java typo, that's why you compile your *.java files to obtain *.class files that contain the bytecodes understandable by the JVM.

JVM control execution of every Java program. It enables features such as automated exception handling, Garbage-collected heap.

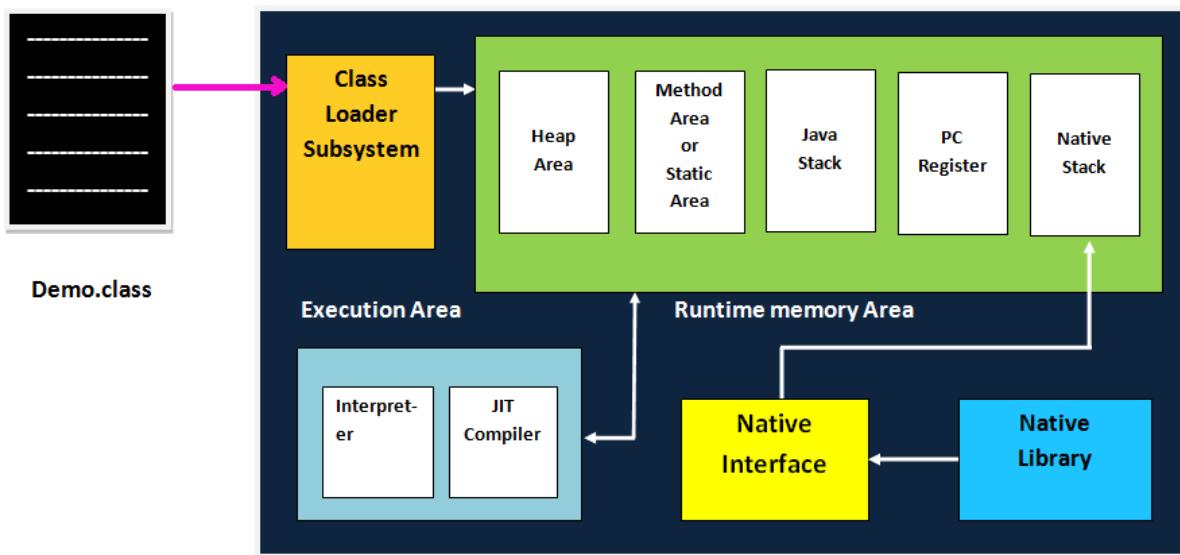
Class files: - Means .class files. It contains byte code.

Class loader sub system: - The class loader sub system does following tasks.

- 2) It loads .class files into memory.
- 3) It verifies the byte code instructions.
- 4) Then it will be a lot necessary memory for the java program.

This memory is divided into 5 parts or blocks are called run time data areas. They are

- 1) **Method area:** - Class code & method code is stored on method area.
- 2) **Heap:** - (Heap) Objects are created on heap memory. The separated memory is allotted in heap.
- 3) **Java stacks:** - These are the areas where java methods are executed. Each java stack is divided into several frames. On each frame separate method is executed.
- 4) **PC registers:** - PC means program counter. In PC registers store memory address.



- 5) **Native method stacks:** - These are the memory areas where native methods.

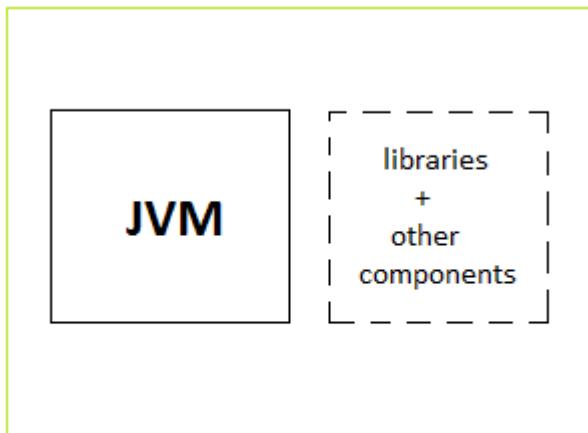
EX: - C and CPP functions are executed. JVM executes C and CPP programs are also executed.

Note: - To execute native methods libraries (C, CPP header files) are required. These headers files are linked with JVM by a program called native method interface.

Execution engine: - It contains interpreter & JIT compiler. This converts the byte code instructions into machine language instructions, so that the processor can execute them.

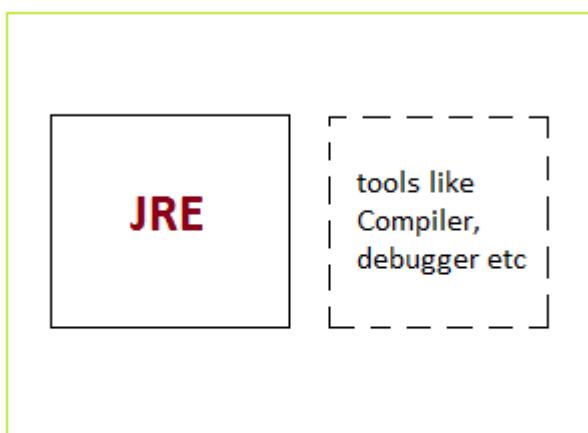
Difference between JDK and JRE

JRE : The Java Runtime Environment (JRE) provides the libraries, the Java Virtual Machine, and other components to run applets and applications written in the Java programming language. JRE does not contain tools and utilities such as compilers or debuggers for developing applets and applications.



JRE - Java Runtime Environment

JDK : The JDK also called Java Development Kit is a superset of the JRE, and contains everything that is in the JRE, plus tools such as the compilers and debuggers necessary for developing applets and applications.



JDK - Java Development Kit

Operators: - An operator is a symbol that performs on an operation.

EX: -

If an operator acts on a single variable, it is called unary operator.

If an operator acts on a 2 variables, it is called binary operator.

If an operator acts on a 3 variables, it is called ternary operator.

Java provides a rich set of operators environment. Java operators can be divided into following categories

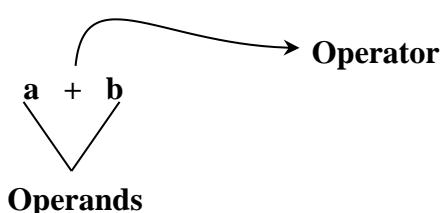
- Arithmetic operators
- Relation operators
- Logical operators
- Bitwise operators
- Assignment operators
- Conditional operators
- Misc operators

Operators in java: -

1) **Arithmetic operators:** - + , - , * , / , %

These operators perform basic arithmetic calculations.

EX: - If a=13, b=5;



Operator	Meaning	Example	Result
+	Addition	a + b	18
-	Subtraction	a - b	8
*	Multiplication	a * b	65
/	Division	a / b	2.6
%	Modules	a % b	3

2) **Unary operators:** - - , ++ , --

These operators act upon a single operand.

→ **Unary minus (-):-** This operator negates the value of a variable.

EX:

```
int x=5;
System.out.println(-x);      o/p:- -5
System.out.println(-(-x));   o/p:- 5
```

Increment (++): - This operator increases the value of a variable by '1'.

EX: - int x=1;
 ++x; O/P:- 2
 x++; O/P:- 3

Writing ++ before a variable is called pre – incrementation

Writing ++ after a variable is called post – incrementation

In pre incrementation , incrementation is done and other task is done next. In post incrementation any other task is performed first, incrementation is done at next.

EX: - Pre incrementation

```
x=1;
System.out.print(x); o/p: - 1
System.out.print(++x);    o/p: - 2
System.out.print(x++);    o/p: - 2
a=1; b=2;
a=++b;
System.out.print(a); o/p: 3
System.out.print(b); o/p: 3
```

Post incrementation

```
x=1;
System.out.print(x); o/p: 1
System.out.print(x++);    o/p: 1
System.out.print(x); o/p: 2
a=1; b=2;
a=b++;
System.out.print(a); o/p: 2
System.out.print(b); o/p: 3
```

→What is the value of the following expression? Given a=7? ++a * a++?

- A) A) 49 B) 64 C) 72 D) none of these

Decrement (--): - This operator decreases the value of a variable by ‘1’.

EX: - int x=1;
 --x; O/P: 0
 x--; O/P: -1

Writing -- before a variable is called pre – decrementation

Writing -- after a variable is called post – decrementation

In pre decrementation , decrementation is done and other task is done next. In post decrementation any other task is performed first, decrementation is done at next.

EX: - Pre decrementation

```
x=1;
System.out.print(x); o/p: 1
System.out.print(--x);    o/p: 0
System.out.print(x--);    o/p: 0
a=1; b=2;
a=--b;
System.out.print(a);      o/p: 1
System.out.print(b);      o/p: 1
```

Post decrementation

```

x=1;
System.out.print(x);      o/p: 1
System.out.print(x--);   o/p: 1
System.out.print(x);      o/p: 0
a=1; b=2;
a=b--;
System.out.print(a); o/p: 2
System.out.print(b); o/p: 1

```

3) Assignment operators: - = , += , -= , *= , /= , %=

a) This operator is used to store a value in to a variable.

EX: - int x=5;

b) It is used to store the value of a variable in to another variable.

EX: - int x=y;

c) It is used to store the value of an expression in to a variable.

EX: - int x = y + z - 4;

Expressions mean combination of a variables & values.

Note: - We can not write more than 1 variable at the left side of assignment.

EX: - x + y = 15; // invalid

Compact notation: -

EX: - num +=10;

sal *=0.5;

p /=k;

i -=34;

y %=5;

These are the compact notations. These are shortcuts of java programs.

4) Relational operator: - < , > , <= , >= , == , !=

These operators are useful to compare to quantities. They are used in construction of condition.

EX: -

if(a>b)-----;

if(x!=1)-----;

if(sal>=5000.00)----;

if(a==b)-----;

if(a<=10)----;

5) Logical operators: - && , || , !

These operators are useful to create compound condition. The compound condition is condition of several simple conditions.

EX: -

If(a>b||b>c||c>d)-----;

Any one thing is true this will be executed;

If(x==10&&y!=10)----;

Both are true this will be executed.

```
If(!(x==!))-----;
If(!(x==1 && y==2))----;
```

6) Boolean operators: - & , | , !

Boolean operators act upon Boolean variables and they return Boolean result

```
boolean a,b;
```

```
a=true;
```

```
b=false;
```

EX: -

a & b → false	}	boolean type results
a b → true		
a & a → true		
b b → false		
! a → false		
! b → true		

7) Bitwise operators: - ~ , & , | , ^ , << , >> , >>>

These operators act up on individual bits binary digits, (0&1) of numbers.

Deci means 10. (0-9).

Ex: - Convert 45 into binary?

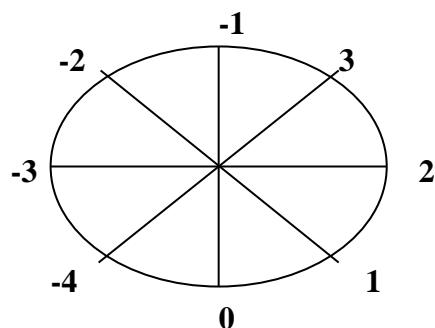
Rule: -Divide the decimal number successively by 2 and take the remainders from bottom to top.

$$\begin{array}{r}
 2 \overline{)45} \\
 2 \overline{)22} - 1 \\
 2 \overline{)11} - 0 \\
 2 \overline{)5} - 1 \\
 2 \overline{)2} - 1 \\
 \end{array}
 \quad = 101101 \rightarrow 8 \text{ bits} \\
 \quad = 00101101$$

→ Convert 00101101 into decimal.

Rule: -Multiply each digit by the powers of 2 and take the sum of the products.

$$\begin{array}{r}
 0 \ 0 \ 1 \ 0 \ 1 \ 1 \ 0 \ 1 \\
 2^7 \ 2^6 \ 2^5 \ 2^4 \ 2^3 \ 2^2 \ 2^1 \ 2^0 \\
 \hline
 0 + 0 + 32 + 0 + 8 + 4 + 0 + 1 = 45
 \end{array}$$



Bitwise complement (~):-

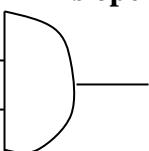
This operator returns the complement from of a given number.

EX: - int x=10;

System.out.print(~x); o/p: - -11

Truth table: - Truth table is a table giving relationship between input bits & output bits. Anding means multiplication.

Bitwise and (&): - This operator performs anding operation on individual bits of numbers.



AND Gate

And gate is a electronic circuit. It performs anding operations.

X	Y	X & Y
0	0	0
0	1	0
1	0	0
1	1	1

EX: -

x=10, y=11, x & y = ? ;

Convert binary

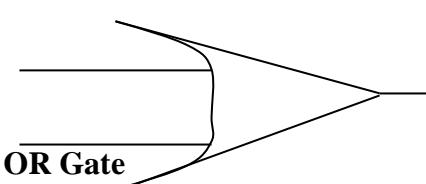
X = 0 0 0 0 1 0 1 0

$$\begin{array}{r} Y = 0 0 0 0 1 0 1 1 \\ \hline 0 0 0 0 1 0 1 0 = 10 \end{array}$$

. . . x & y = 10.

Bitwise or (|):

This operator performs oring operations on individual bits of numbers. Oring means both adding x & y.



EX: -

x=10, y=11, x | y = ? ;

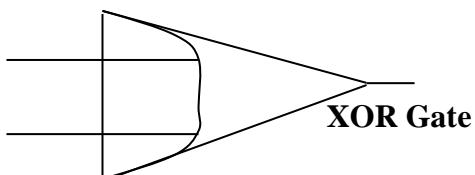
Convert binary

X	Y	X Y
0	0	0
0	1	1
1	0	1
1	1	1

$$\begin{array}{r} X = 0 \ 0 \ 0 \ 0 \ 1 \ 0 \ 1 \ 0 \\ Y = 0 \ 0 \ 0 \ 0 \ 1 \ 0 \ 1 \ 1 \\ \hline 0 \ 0 \ 0 \ 0 \ 1 \ 0 \ 1 \ 1 = 11 \end{array}$$

. . . $x | y = 11$.

Bitwise xor (^) : - Here x means exclusive. ^ Means cap, carpet, circumflex. This operator performs exclusive orring operation on individual bits of numbers. In the input bits if odd number of '1's is there in the output we will get '1'.



X	Y	$X \wedge Y$
0	0	0
0	1	1
1	0	1
1	1	0

Ex: -

$$x=10, y=11, x \wedge y = ? ;$$

Convert binary

$$\begin{array}{r} X = 0 \ 0 \ 0 \ 0 \ 1 \ 0 \ 1 \ 0 \\ Y = 0 \ 0 \ 0 \ 0 \ 1 \ 0 \ 1 \ 1 \\ \hline 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 1 = 1 \end{array}$$

. . . $x \wedge y = 1$.

Bitwise left shift (<<) :

This operator shifts the bits of a number two words left .a specified number of times.

Ex:- $x = 10$;
 $x << 2 = ?$

$$X=10 \rightarrow$$

0	0	0	0	1	0	1	0
---	---	---	---	---	---	---	---

$x \ll 2 \rightarrow$	<table border="1"> <tr> <td>0</td><td>0</td><td>1</td><td>0</td><td>1</td><td>0</td><td>0</td><td>0</td> </tr> <tr> <td>2^7</td><td>2^6</td><td>2^5</td><td>2^4</td><td>2^3</td><td>2^2</td><td>2^1</td><td>2^0</td> </tr> </table>	0	0	1	0	1	0	0	0	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
0	0	1	0	1	0	0	0										
2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0										
.	$0 + 0 + 32 + 0 + 8 + 0 + 0 + 0 = 40$																

$\therefore x \ll 2 = 40.$

Bitwise Right Shift ($>>$): -

This operator shifts the bits of a number two words right a specified number of times.

Ex: - $x=10;$
 $x>>2 = ?$

$x=10 \rightarrow$	<table border="1"> <tr> <td>0</td><td>0</td><td>0</td><td>0</td><td>1</td><td>0</td><td>1</td><td>0</td> </tr> </table>	0	0	0	0	1	0	1	0								
0	0	0	0	1	0	1	0										
$x>>2 \rightarrow$	<table border="1"> <tr> <td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>1</td><td>0</td> </tr> <tr> <td>2^7</td><td>2^6</td><td>2^5</td><td>2^4</td><td>2^3</td><td>2^2</td><td>2^1</td><td>2^0</td> </tr> </table>	0	0	0	0	0	0	1	0	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
0	0	0	0	0	0	1	0										
2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0										

$0 + 0 + 0 + 0 + 0 + 0 + 2 + 0 = 2$

$\therefore x>>2 = 2.$

Bitwise zero-fill right shift ($>>>$): - It is the same as $>>$.

→ What is the difference between $>>$ & $>>>$?

A) $>>$ shift the bits 2 words right & properties the sign bit (0 represent +ve 1 represent -ve sign)
 $>>>$ also shift the bits 2 words right bit. It does not protect sign bit. It always fills the sign bit with 0. That reason it is called zero-fill right shift.

→ // bitwise operators

```
class Bits
{
    public static void main(String[] args)
    {
        byte x,y;
        x=10;
        y=11;
        System.out.println(~x+"+"+(~x));
        System.out.println('x' & y+"+"+(x&y));
        System.out.println('x' | y+"+"+(x|y));
        System.out.println('x' ^ y+"+"+(x^y));
```

```

        System.out.println("x<<2="+(x<<2));
        System.out.println("x>>2="+(x>>2));
        System.out.println("x>>>2="+(x>>>2));
    }
}

```

```

D:\bkr\Core java>javac Bits.java
D:\bkr\Core java>java Bits
~x=-11
x & y=10
x | y=11
x ^ y=1
x<<2=40
x>>2=2
x>>>2=2

```

Syntax: - Syntax means correct format of writing a statement.

8) Ternary operator or conditional operator (?, :):-

This operator called ternary operator because it acts on three variables. Ternary means 3. This operator is also called conditional operator because it represents a conditional statement.

Syntax: -

```
var=exp1?exp2:exp3;
```

→ // conditional operators

```

class Conditional
{
    public static void main(String[] args)
    {
        int x=10,y=11;
        int max=(x>y)?x:y;
        System.out.println("Maximum value is =" +max);
    }
}

```

```
D:\bkr\Core java>javac Conditinal.java
```

```
D:\bkr\Core java>java Conditinal
```

Maximum value is=11

9) Dot operator (.) :- (Membership operator)

- a) To refer to a class in a package
java.util.Date, java.io.BufferedReader
- b) To refer to a method in a class
math.pow(), emp.getName(), br.read()
- c) To refer to a variable in a class
Employee.name(), emp.eno(), System.out

10) **Instanceof operator:** - To test whether an object belongs to a class

EX: -

boolean x=emp instanceof Employee

Instants means object. It is a single word, so no space.

Executing the statements one by one is called sequential execution.

A program can write simple programs only using sequential execution. If we want to develop bigger & complex programs random execution we needed.

Executing the statements randomly & repeatedly is called random execution. It is useful to write a complex program. It is achieved using control statements.

```
class Test{  
  
    public static void main(String args[]){  
  
        Test t=new Test();  
  
        System.out.println(t instanceof Test);//true  
  
    }  
  
}
```

Precedence of Java Operators:

Operator precedence determines the grouping of terms in an expression. This affects how an expression is evaluated. Certain operators have higher precedence than others; for example, the multiplication operator has higher precedence than the addition operator:

For example, $x = 7 + 3 * 2$; here x is assigned 13, not 20 because operator $*$ has higher precedence than $+$, so it first gets multiplied with $3*2$ and then adds into 7.

Here, operators with the highest precedence appear at the top of the table, those with the lowest appear at the bottom. Within an expression, higher precedence operators will be evaluated first.

Category	Operator	Associativity
Postfix	() [] . (dot operator)	Left to right
Unary	++ -- ! ~	Right to left
Multiplicative	* / %	Left to right
Additive	+ -	Left to right
Shift	>> >>> <<	Left to right
Relational	> >= < <=	Left to right
Equality	== !=	Left to right
Bitwise AND	&	Left to right
Bitwise XOR	^	Left to right
Bitwise OR		Left to right
Logical AND	&&	Left to right
Logical OR		Left to right
Conditional	?:	Right to left
Assignment	= += -= *= /= %= >>= <<= &= ^= =	Right to left
Comma	,	Left to right

Accepting data from keyboard: -

If you want to data read input from keyboard.

Stream:- A stream represent flow of data from one place to another place. Stream carries data.

There are 2 types of streams.

- 1) **In put streams:** - in put streams read data or accept data.
- 2) **Out put streams:** - out put streams send or write data .All streams are represented by classes in java.io package.

System.in: - InputStream obj – keyboard. Here in means field.

System.out: - PrintStream obj – monitor

System.err: - PrintStream obj – monitor

→ What is difference between System.out & System.err?

A) System.out is useful to display general messages.

System.err is used to display error messages. Stream is one of the io device.

Input/output devices are represented by streams. Streams are useful to achieve hardware independence in a java program.

InputStreamReader: - It is read data from keyboard.

BufferedReader: - We read the data at BufferedReader.

- 1) Connect keyboard to InputStreamReader
InputStreamReader isr=new InputStreamReader(System.in)
- 2) Connect InputStreamReader to BufferedReader
BufferedReader br=new BufferedReader(isr)
- 3) Read data from br using read() or readLine() methods.

→ // accepting a char from keyboard

```
import java.io.*;
class CharTest
{
    public static void main(String[] args) throws IOException
    {
        // create BufferedReader obj
```

```

BufferedReader br=new BufferedReader(new
InputStreamReader(System.in));

System.out.print("Enter a char : ");
char ch=(char)br.read();
System.out.println("you entered : "+ch);
}
}

```

D:\bkr\Core java>javac CharTest.java

D:\bkr\Core java>java CharTest

Enter a char : 4

you entered : 4

Type casting: - Converting one data type into another data type is called type casting or casting.

Writing the data type with in the simple braces, it is called caste operator. Cast means converting.

Exception represents run time errors. Throws Exception, throw out exception without handling it.

→// accepting a string from keyboard

```

import java.io.*;
class StringTest
{
    public static void main(String[] args) throws IOException
    {
        BufferedReader br=new BufferedReader(new
InputStreamReader(System.in));
        System.out.print("Enter a char : ");
        String ch=br.readLine();
        System.out.println("you entered : "+ch);
    }
}

```

D:\bkr\Core java>javac StringTest.java

D:\bkr\Core java>java StringTest

Enter a char : Usha

you entered : Usha

→// accepting integers from keyboard

```
import java.io.*;
class IntTest
{
    public static void main(String[] args) throws IOException
    {
        // create BufferedReader obj
        BufferedReader br=new BufferedReader(new
                                         InputStreamReader(System.in));
        System.out.print("Enter a number : ");
        int n=Integer.parseInt(br.readLine());
        System.out.println("you entered : "+n);
    }
}
```

D:\bkr\Core java>javac IntTest.java

D:\bkr\Core java>java IntTest

Enter a number : 67

you entered : 67

→// accepting float numbers from keyboard

```
import java.io.*;
class FloatTest
{
    public static void main(String[] args) throws IOException
    {
        // create BufferedReader obj
        BufferedReader br=new BufferedReader(new
                                         InputStreamReader(System.in));
        System.out.print("Enter a number : ");
        float n=Float.parseFloat(br.readLine());
        System.out.println("you entered : "+n);
    }
}
```

D:\bkr\Core java>javac FloatTest.java

D:\bkr\Core java>java FloatTest

Enter a number : 23

you entered : 23.0

→// accepting and displaying emp data

```
import java.io.*;
class Empdata
{
    public static void main(String[] args) throws IOException
    {
        // create BufferedReader obj
        BufferedReader br=new BufferedReader(new
                                         InputStreamReader(System.in));
        // accept employee data
        System.out.print("Enter id : ");
        int id=Integer.parseInt(br.readLine());
        System.out.print("Enter sex : ");
        char sex=(char)br.read();
        br.skip(2);
        System.out.print("Enter name : ");
        String name=br.readLine();
        System.out.println("Id =" +id);
        System.out.println("Sex =" +sex);
        System.out.println("Name =" +name);
    }
}
```

D:\bkr\Core java>javac Empdata.java

D:\bkr\Core java>java Empdata

Enter id : 130

Enter sex : m

Enter name : Sathish

Id =130 Sex =m Name =Sathish

→ /* accepting two no's from key board & find the result + / - *

```
import java.io.*;
class Arithmetic
{
    public static void main(String args[])throws IOException
    {
        BufferedReader br=new BufferedReader(new
                                         InputStreamReader(System.in));
        System.out.print("enter m value: ");
        int m=Integer.parseInt(br.readLine());
        System.out.print("enter n value: ");
        int n=Integer.parseInt(br.readLine());
        System.out.println("Addition is:="+(m+n));
        System.out.println("Subtraction is:="+(m-n));
        System.out.println("multiplication is:="+(m*n));
        System.out.println("Division is:="+(m/n));
    }
}
```

*Scanner(1.5) This class is introduce java 1.5 version and it can be used to Read the contents from a resource.

```
import java.io.*;
import java.util.*;
class Read{
public static void main(String[] args) throws IOException{
Scanner s = new Scanner(System.in);
System.out.println("Enter your name:");
String name = s.nextLine();
System.out.println("name:" + name);
System.out.println("Enter your age:");
Int age = s.nextInt();
System.out.println("Age:" + age);

    }
}
```

***Console(class)(1.6):** This class is introduce in java 1.6 version. This class is used to read the contents from the keyboard. Console is available in “io” package. We can not object of console class directly. It can be created by using console method available in System class.

```
import java.io.*;
class Read1{
public static void main(String[] args) throws IOException{
Console c = System.console();
String name = c.readLine("Enter your name:");
System.out.println(name);
Char[] pwd = c.readPassword("Enter your password");
System.out.println(pwd);
}}
```

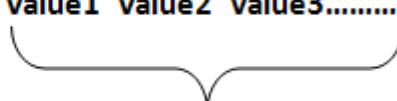
Java Command Line Arguments

The java command-line argument is an argument i.e. passed at the time of running the java program.

The arguments passed from the console can be received in the java program and it can be used as an input.

So, it provides a convenient way to check the behavior of the program for the different values. You can pass N (1,2,3 and so on) numbers of arguments from the command prompt.

```
javac Mainclass.java
java Mainclass value1 value2 value3.....
```



Command Line Arguments

Simple example of command-line argument in java

In this example, we are receiving only one argument and printing it. To run this java program, you must pass at least one argument from the command prompt.

```
class CommandLineExample
{
public static void main(String args[]){
System.out.println("Your first argument is: "+args[0]);
}
}
```

Example of command-line argument that prints all the values

In this example, we are printing all the arguments passed from the command-line. For this purpose, we have traversed the array using for loop.

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

Control statements: - Control statements are the statements which alter or changed the flow of execution. They are given below

- 1) if-else statement
- 2) do-while loop
- 3) while loop
- 4) for loop
- 5) for each loop
- 6) switch statement
- 7) break statement
- 8) continue statement
- 9) return statement

Statements can execute only one time but loop can execute several times repeatedly.

1) if-else statement: - This statement is useful to execute a task. Task means 1 or more statements depending upon whether a condition is true or not.

Syntax: - **if(condition)**

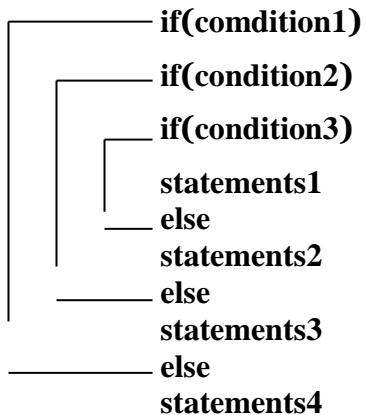
```
{
    Statements 1;
}
else
{
    Statements 2;
}
```

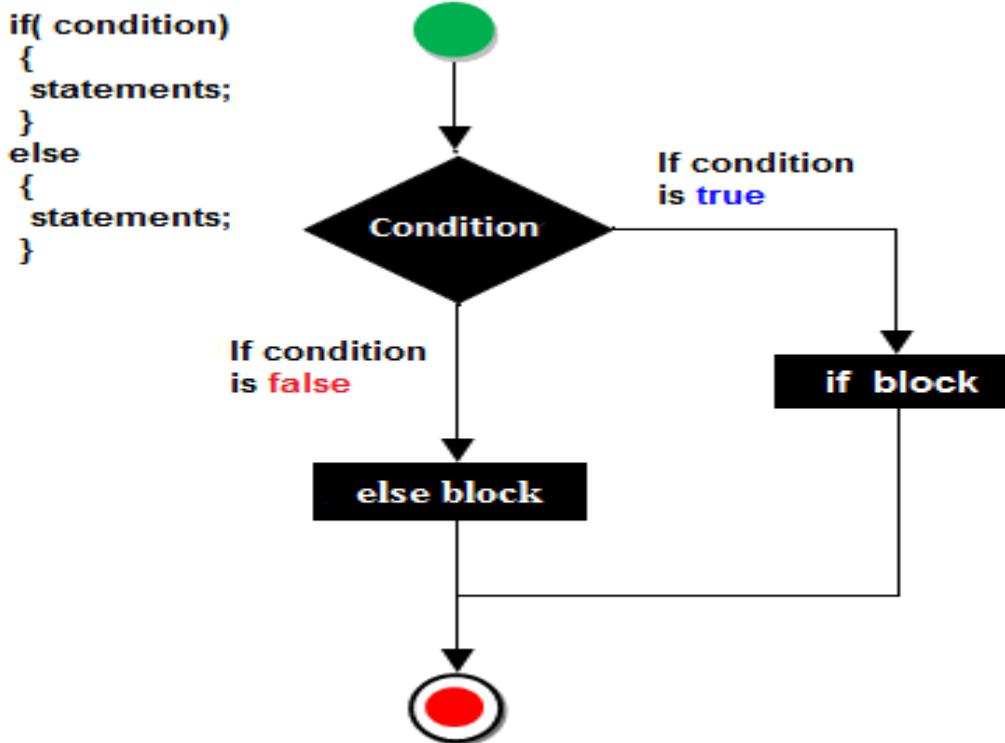
If the condition is true statements 1 is executed. Otherwise statements 2 are executed.

(or)

```
if(condition1)
{
    Statements 1;
}
```

```
else  
if(condition2)  
{  
    Statements 2;  
}  
else  
if(condition3)  
{  
    Statements 3;  
}  
else  
{  
    Statements 4;  
}  
(or)
```





→// test a number is +ve or -ve

```

class IfElseTest
{
    public static void main(String[] args)
    {
        int num=5;
        if(num==0)
            System.out.println("It is Zero");
        else if(num>0)
            System.out.println("It is +ve");
        else
            System.out.println("It is -ve");
    }
}
  
```

D:\bkr\Core java>javac IfElseTest.java

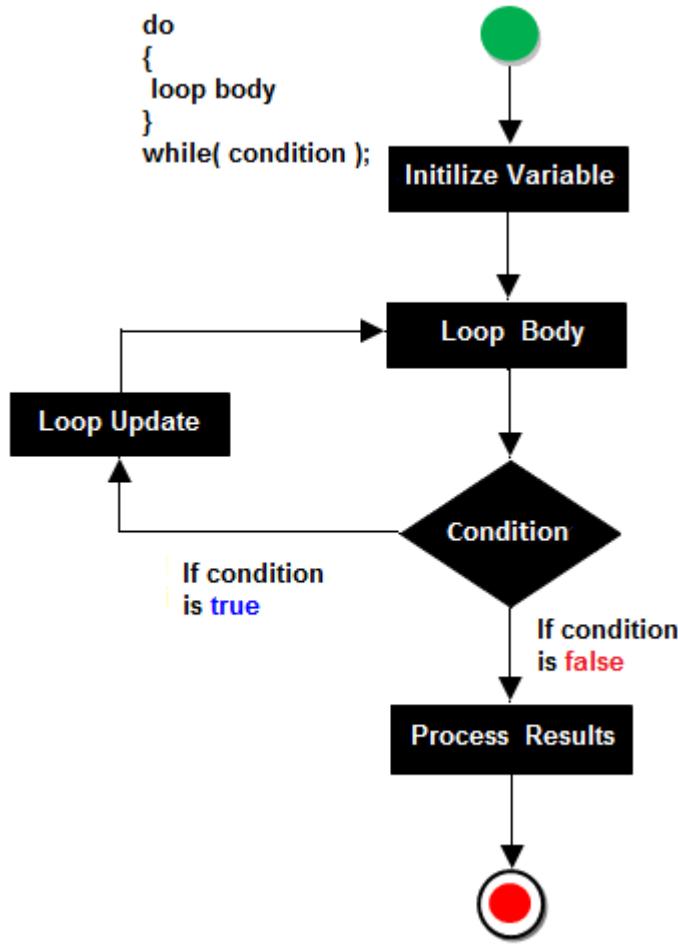
D:\bkr\Core java>java IfElseTest

It is +ve

2) do-while loop: - This loop repeatedly execute a group of statements as long as a condition is true.

Syntax: -

```
do
{
    Statements;
}
while (condition);
```



→// generates no's from 1 to 10

```
class DoWhileTest
{
    public static void main(String[] args)
    {
        int x=1;
        do
        {
            System.out.print(x+"\t");
            x++;
        }
    }
}
```

```

        }
        while (x<=10);
    }
}

```

D:\bkr\Core java>javac DoWhileTest.java

D:\bkr\Core java>java DoWhileTest

1 2 3 4 5 6 7 8 9 10

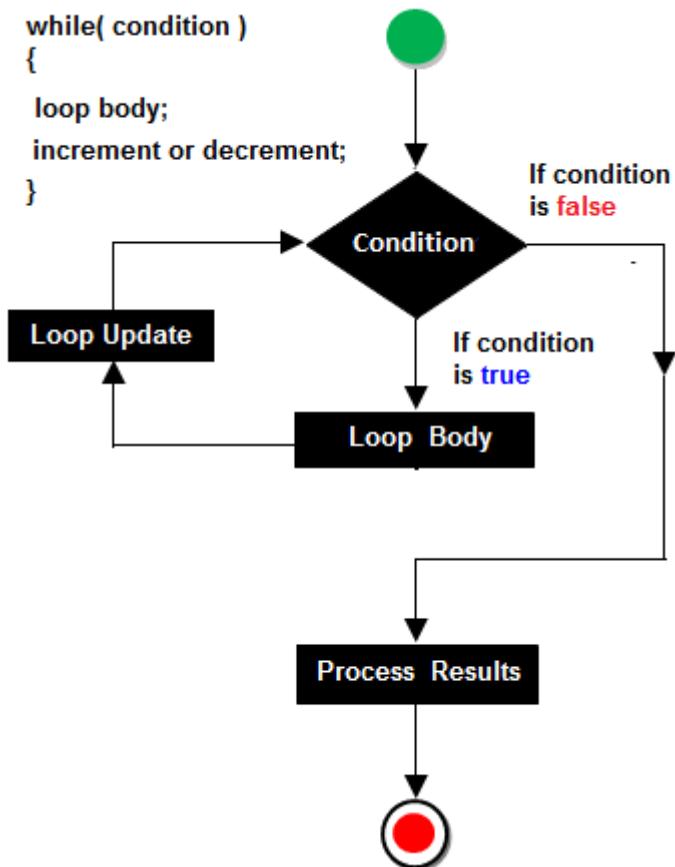
3) while loop: -This loop is useful to repeatedly execute a group of statements as long as a given condition is true.

Syntax: -

```
while(condition)
```

```
{
    Statements;
}
```

```
while( condition )
{
    loop body;
    increment or decrement;
}
```



→// even no's up to 10

```
class WhileTest
{
    public static void main(String[] args)
    {
        int x=2;
        while (x<=10)
        {
            System.out.println(x+"\t");
            x+=2;
        }
    }
}
```

D:\bkr\Core java>javac WhileTest.java

D:\bkr\Core java>java WhileTest

2 4 6 8 10

While loop is more efficient from do-while loop. Because will have better control right from beginning.

```
import java.util.*;
class Whildemo
{
public static void main(String args[])
{
Scanner sc=new Scanner(System.in);
System.out.println("enter any no below 100");
int n=sc.nextInt();
while(n<=100)
{
System.out.println(n);
n++;
}
}
}
```

4) for loop: -

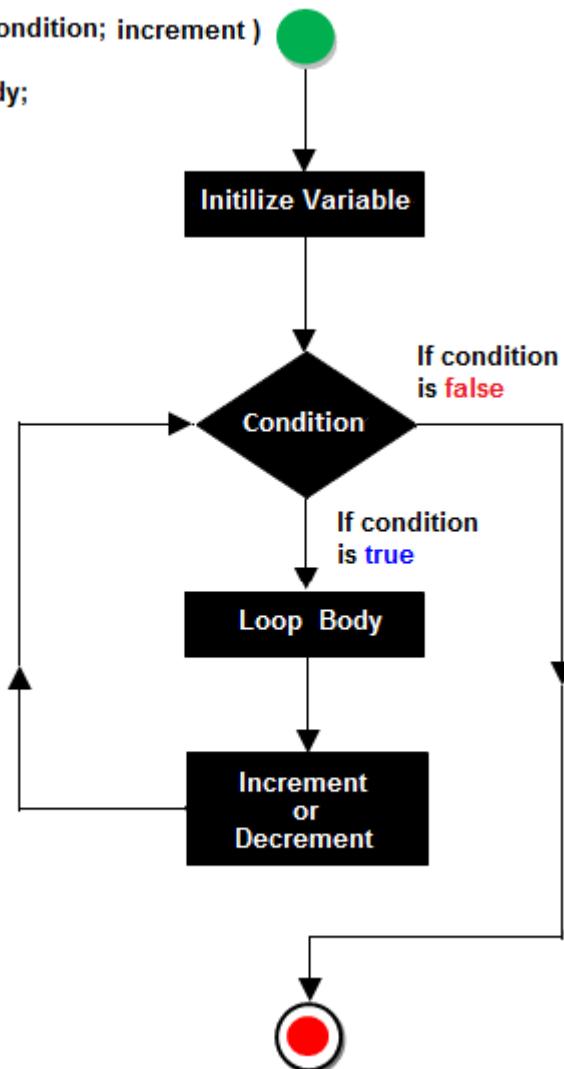
This is used to repeatedly execute a group of statements as long as a condition is true. For loop is suitable for when the user knows the exact number of repetitions.

Syntax: -

```
for(exp1;exp2;exp3)
```

```
{  
Statements;  
}
```

```
for( init; condition; increment )  
{  
loop body;  
}
```



Exp1 is called initializing expression.

Exp2 is called conditional expression.

Exp3 is called modifying expression.

Note: - We can write for loop by eliminating exp1 or exp2 or exp3 or any 2 exp or all the 3 expressions.

→ // generates no's from 1 to 100

```
class ForTest
{
    public static void main(String[] args)
    {
        for(int i=0;i<=100;i++)
        {
            System.out.println(i+"\t");
        }
    }
}
```

D:\bkr\Core java>javac ForTest.java

D:\bkr\Core java>java ForTest

1	2	3	4	5	6	7	8	9	10
11	12	13	14	15	16	17	18	19	20
21	22	23	24	25	26	27	28	29	30
31	32	33	34	35	36	37	38	39	40
41	42	43	44	45	46	47	48	49	50
51	52	53	54	55	56	57	58	59	60
61	62	63	64	65	66	67	68	69	70
71	72	73	74	75	76	77	78	79	80
81	82	83	84	85	86	87	88	89	90
91	92	93	94	95	96	97	98	99	100

Infinite loop is executed then press CTRL + C. That is executes for ever. Infinite loops are draw backs in a program.

(or)

→// generates no's from 1 to 50

```
class ForTest1
{
    public static void main(String[] args)
    {
        int i=0;
        for(;;)
        {
            System.out.println(i);
            i++;
            if(i>50)
                break;
        }
    }
}
```

D:\bkr\Core java>javac ForTest1.java

D:\bkr\Core java>java ForTest1

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	
36	37	38	39	40	41	42	43	44	45	46	47	48	49	50			

Break is statement to come out from the loop.

Note: - It is possible to write a loop inside other loops, they are called nested loops.

Syntax: -

```
for(exp1;exp2;exp3)
{
    for(exp1;exp2;exp3)
    {
        statements;
    }
}
```

5) for each loop: - This loop is used to repeatedly executes a group of statements for each element of a collection. Collection means group of elements. Array means a group of elements & also java.util classes.

Syntax: -

```
for(var:collection)
{
    statements;
}
```

→// for each loop

```
class ForEachTest
{
    public static void main(String[] args)
    {
        int arr[]={10,33,56,20,90};
        for(int x:arr)
        {
            System.out.println(x+"\t");
        }
    }
}
```

D:\bkr\Core java>javac ForEachTest.java

D:\bkr\Core java>java ForEachTest

10 33 56 20 90

6) switch statement: - switch statement is useful to execute a particular task from several available tasks depending upon the value of a variable

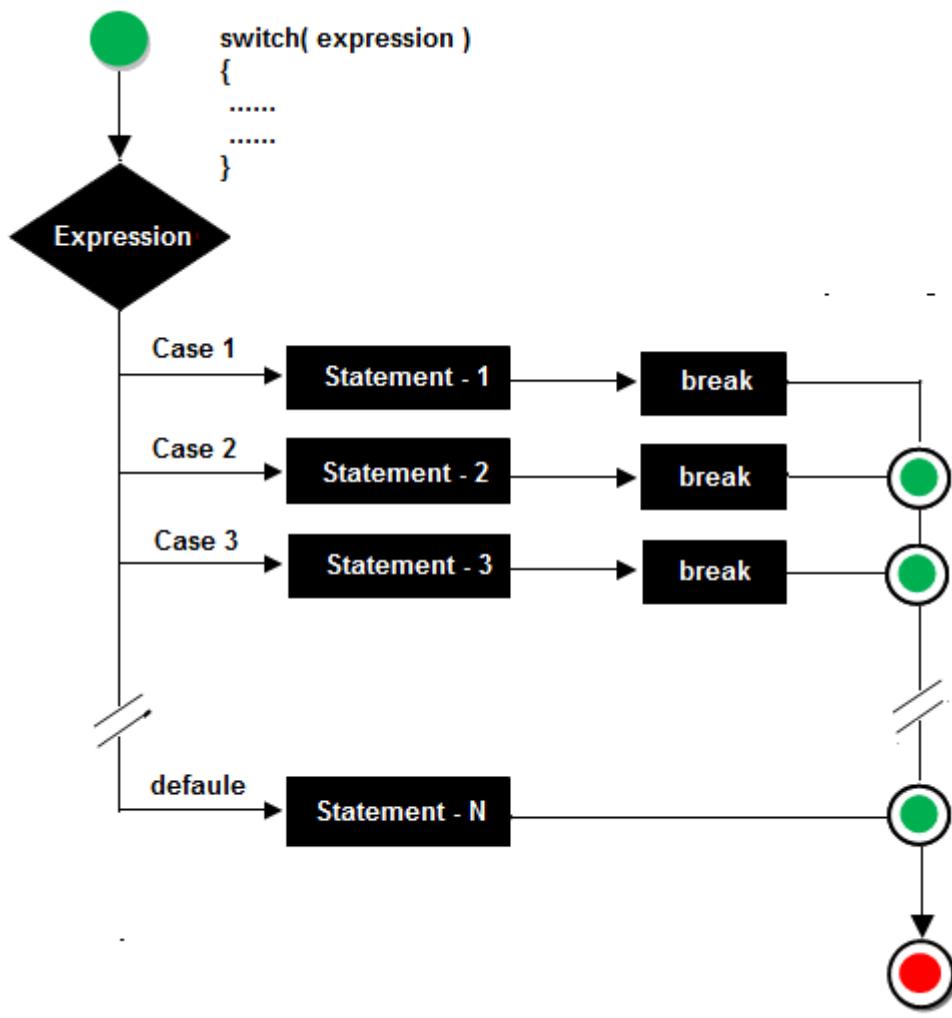
Syntax: -

```
switch(var)
{
```

```

case value1:
    Statements 1;
    break;
case value2:
    Statements 2;
    break;
    ||
case value n:
    Statements n;
    break;
default :
    Statements;
}

```



Switch with no

```
import java.util.*;  
class SwitchNoDemo  
{  
public static void main(String []args)  
{  
Scanner sc=new Scanner(System.in);  
System.out.print("enter p value:");  
float p=sc.nextFloat();  
System.out.print("enter t value:");  
int t=sc.nextInt();  
float i;  
switch(t)  
{  
case 3:  
{  
i=p*t*8/100;  
System.out.print("intrest for 3 yr=" +i);  
break;  
}  
case 5:  
{  
i=p*t*6/100;  
System.out.print("intrest for 5 yr=" +i);  
break;  
}  
case 8:  
{  
i=p*t*4/100;  
System.out.print("intrest for 8 yr=" +i);  
break;  
}  
case 10:  
{  
i=p*t*1/100;  
System.out.print("intrest for 10 yr=" +i);  
}
```

```
break;  
}  
default:  
  
System.out.print("enter 3or 5or 8or 10 only");  
}  
}  
}
```

→// switch statement

```
class SwitchTest  
{  
    public static void main(String[] args)  
    {  
        char color='g';  
        switch(color)  
        {  
            case 'r':  
            case 'R':  
                System.out.println("Red");  
                break;  
            case 'g':  
            case 'G':  
                System.out.println("Green");  
                break;  
            case 'b':  
            case 'B':  
                System.out.println("Blue");  
                break;  
            case 'w':  
            case 'W':  
                System.out.println("White");  
                break;  
            default:  
                System.out.println("No color");  
        }  
    }  
}
```

D:\bkr\Core java>javac SwitchTest.java

D:\bkr\Core java>java SwitchTest

Green

Switch with String(jdk 1.7)

```
import java.util.*;  
class SwitchStringDemo  
{  
    public static void main(String []args)  
    {  
        Scanner balu=new Scanner(System.in);  
        System.out.print("enter course");  
        String course=balu.next();  
        int REGFEE=100;  
        int COREJAVA=1000;  
        int ADVJAVA=2000;  
        int TOTJAVA=4500;  
        int fee;  
        switch(course)  
        {  
            case "COREJAVA":  
            case "corejava":  
            {  
                fee=REGFEE+COREJAVA;  
                System.out.println("the tot fee= "+fee);  
                break;  
            }  
            case "AdvJAVA":  
            case "advjava":  
            {  
                fee=REGFEE+ADVJAVA;  
                System.out.println("the tot fee= "+fee);  
            }  
        }  
    }  
}
```

```
break;
}
case "TOTJAVA":
case "totjava":
{
fee=REGFEE+TOTJAVA;
System.out.println("the tot fee=" +fee);
break;
}
default:
System.out.println("communicate in office");
}
}
}
```

SwitchEnumDemo

```
class SwitchEnumDemo
{
enum Day
{
SUNDAY,MONDAY,TUESDAY,WEDDAY,THUSDAY,FRIDAY,SATDAY
}
public static void main(String args[])
{
Day d=Day.FRIDAY;
switch(d)
{
case SUNDAY:
{
System.out.println("the give day SUNDAY");
break;
}
case MONDAY:
{
System.out.println("the give day MONDAY");
break;
```

```
}

case TUESDAY:
{
System.out.println("the give day TUESDAY");

break;
}

case FRIDAY:
{
System.out.println("the give day FRIDAY");

break;
}

case SATDAY:
{
System.out.println("the give day SATDAy");

break;
}

case WEDDAY:
{
System.out.println("the give day Wedday");

break;
}

default:
System.out.println("no case is matching");
}

}
```

Break is statement to come out from a switch statement.

Switch statement is useful to handle menu driven programs. Menu means a list of items or options for the user select.

7) break statement: -

- a) it is useful to come out of a loop.
- b) it is useful to come out of a switch statement.
- c) it is used in a nested blocks to go to end of a block

Syntax: -

```
break blockname; // go to end of block
```

go to statement is used in to jump one statement to another statement.

→ Why goto statement are not available in java?

- A) 1) goto statements lead to confusion for a programmer.
- 2) goto statement from infinite loops which are draw backs in a program.
- 3) The documentation of the program will become difficult with more goto's are used. Documentation means preserving a copy of the program. Algorithm means a step of logic.
- 4) goto statements are not part of structured programming. Because of above reasons the goto statements are eliminates of java.

→// break test

```
class BreakTest
{
    public static void main(String args[])
    {
        for(int i=0;i<=10;i++)
        {
            if(i==4)
                break;
            System.out.println(i);
        }
        System.out.println("break executed");
    }
}
```

D:\bkr\Core java>javac BreakTest.java

D:\bkr\Core java>java BreakTest

```
0
1
2
3
break executed
```

Labeled break statement: - It can be used as a form of goto statement in java.

→// break to go to end of a block

```
class BreakTest1
{
    public static void main(String[] args)
    {
        boolean x=true;
        bl1:{

            bl2:{

                bl3:{

                    System.out.println("Block 3");
                    if(x=true)
                        break bl2;
                }
                System.out.println("Block 2");
            }
            System.out.println("Block 1");
        }
        System.out.println("Out of all");
    }
}
```

D:\bkr\Core java>javac BreakTest1.java

D:\bkr\Core java>java BreakTest1

Block 3

Block 1

Out of all

8) continue statement: -

This statement is useful to continue the next repetition of a loop. When the statement will subsequent statements in the loop are not executed.

→// continue statement

```
class ContinueStatementTest
{
    public static void main(String[] args)
    {
        for(int i=10;i>=1;i--)
        {
            if(i>5)
                continue;
            System.out.print(i+"\t");
        }
    }
}
```

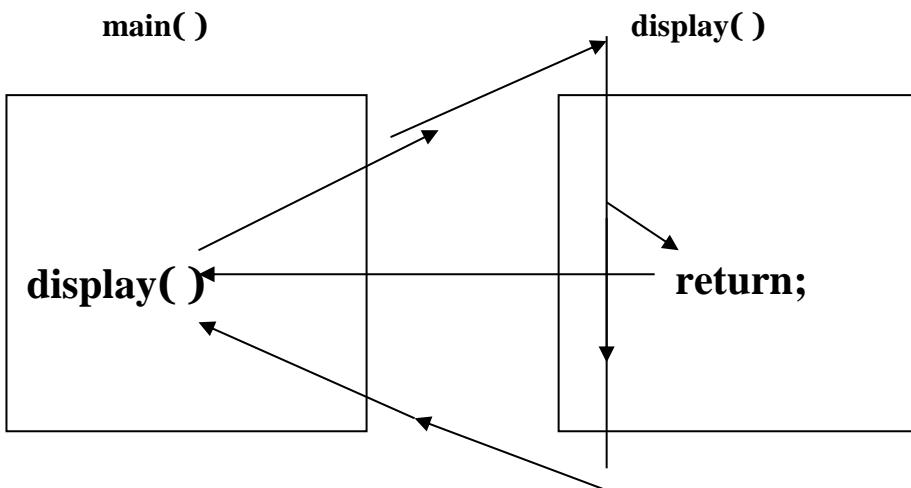
D:\bkr\Core java>java ContinueStatementTest

D:\bkr\Core java>java ContinueStatementTest

5 4 3 2 1

9) return statements: - it is used in 2 ways.

- 1) It is used to come out of a method to a calling method.



Note: - If return is used in main the entire program is terminated.

- 2) return statement can be used to return some value to the calling method.

EX: - return x;

→// return in main()

```
class ReturnTest
{
    public static void main(String[] args)
    {
        int num=1;
        System.out.println("Before return");
        if(num==1)
            return; // System.exit(0);
        System.out.println("After return");
    }
}
```

D:\bkr\Core java>javac ReturnTest.java

D:\bkr\Core java>java ReturnTest

Before return

System.exit(0) will terminate the program from any method.

→What is the difference between System.exit(0) and System.exit(1)?

A) System.exit(0) represents normal termination.

System.exit(1) represents termination due to an error.

Assertion:

Assertion is a statement in java. It can be used to test your assumptions about the program.

While executing assertion, it is believed to be true. If it fails, JVM will throw an error named AssertionException. It is mainly used for testing purpose.

Advantage of Assertion:

It provides an effective way to detect and correct programming errors.

Syntax of using Assertion:

There are two ways to use assertion. First way is:

assert expression;

and second way is:

```
assert expression1 : expression2;
```

Simple Example of Assertion in java:

```
import java.util.Scanner;  
class AssertionExample  
{  
    public static void main( String args[] ){  
  
        Scanner scanner = new Scanner( System.in );  
        System.out.print("Enter ur age ");  
  
        int value = scanner.nextInt();  
        assert value>=18:" Not valid";  
  
        System.out.println("value is "+value);  
    }  
}
```

If you use assertion, It will not run simply because assertion is disabled by default. To enable the assertion, -ea or -enableassertions switch of java must be used.

Compile it by: **javac AssertionExample.java**

Run it by: **java -ea AssertionExample**

Where not to use Assertion:

There are some situations where assertion should be avoid to use. They are:

1. According to Sun Specification, assertion should not be used to check arguments in the public methods because it should result in appropriate runtime exception e.g. `IllegalArgumentException`, `NullPointerException` etc.
2. Do not use assertion, if you don't want any error in any situation.

→ // print odd no's up to 50

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

```

{
    int n=1;
    System.out.println("odd no's are:");
    while(n<100)
    {
        System.out.print("\t"+n);
        n+=2;
    }
}

```

D:\bkr\Core java>javac Odd.java

D:\bkr\Core java>java Odd

odd no's are:

1	3	5	7	9	11	13	15	17	
19	21	23	25	27	29	31	33	35	37
39	41	43	45	47	49	51	53	55	57
59	61	63	65	67	69	71	73	75	77
79	81	83	85	87	89	91	93	95	97
99									

→ // Test whether a number is even or odd

```

class EvenOROdd
{
    public static void main(String[] args)
    {
        int a=20;
        if(a%2==0)
            System.out.println("It is even number");
        else
            System.out.println("It is odd number");
    }
}

```

D:\bkr\Core java>javac EvenOROdd.java

D:\bkr\Core java>java EvenOROdd

It is even number

→ // given no is prime or not

```

class PrimeOrNot
{

```

```

public static void main(String args[])
{
    int i,j,k,l;
    i=9;
    j=2;
    l=1;
    while(j<9)
    {
        k=i%j;
        if(k==0)
            l=k;
        j++;
    }
    if(l==0)
    {
        System.out.println("not prime");
    }
    else
    {
        System.out.println("prime");
    }
}
}

```

D:\bkr\Core java>javac PrimeOrNot.java

D:\bkr\Core java>java PrimeOrNot

not prime

→//display a multiplication table

```

class MultificationTable
{
    public static void main(String args[])
    {
        int i=2,j;
        System.out.println("Multiplication table is=:");
        for(j=1;j<=10;j++)
        {
            System.out.println(i+"*"+j+"="+i*j);
        }
    }
}

```

```
    }  
}
```

D:\bkr\Core java>javac MultificationTable.java

D:\bkr\Core java>java MultificationTable

Multiplication table is=:

2*1=2

2*2=4

2*3=6

2*4=8

2*5=10

2*6=12

2*7=14

2*8=16

2*9=18

2*10=20

→ /* display the stars the following

```
*  
* *  
* * *  
* * * *  
* * * * *  
*/  
class StarsTest  
{  
    public static void main(String args[])  
    {  
        int stars=1,lines=5,spaces=5;  
        int i=1,j,k;  
        for(i=0;i<lines;i++)  
        {  
            // print spaces first  
            for(k=0;k<spaces;k++)  
                System.out.print(" ");
```

```

        for(j=0;j<(stars);j++)
            System.out.print("* ");
        spaces--;
        stars +=1;
        System.out.println( );
    }
}
}

```

D:\bkr\Core java>javac StarsTest.java

D:\bkr\Core java>java StarsTest

```

*
 *
 *
 *
 *
 *
 *
 *
 *

```

→ // print the Fibonacci series up to 10

```

class Fibonacci
{
    public static void main(String args[])
    {
        int i=0;
        int j=1,c;
        System.out.print(i);
        System.out.print("\t"+j);
        for(int k=3;k<=10;k++)
        {
            c=i+j;
            System.out.print("\t"+c);
            i=j;
            j=c;
        }
    }
}

```

D:\bkr\Core java>javac Fibonacci.java

D:\bkr\Core java>java Fibonacci

0 1 1 2 3 5 8 13 21 34

→ // print prime no's

```
class PrimeSeries
{
    public static void main(String args[])
    {
        int k,m,j;
        for(j=2;j<100;j++)
        {
            k=0;
            for(int i=1;i<=j;i++)
            {
                m=j%i;
                if(m==0)
                    k=k+1;
            }
            if(k==2)
                System.out.print("\t"+j);
        }
    }
}
```

D:\bkr\Core java>javac PrimeSeries.java

D:\bkr\Core java>java PrimeSeries

2 3 5 7 11 13 17 19 23 29 31 37 41 43 47 53 59 61
67 71 73 79 83 89 97

Arrays: - An array represents a group of elements of same data type. They are 2 types.

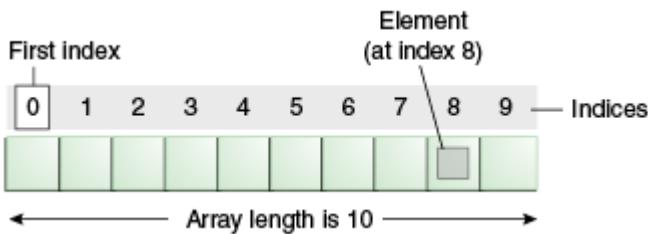
- 1) Single dimensional array. 2) Multi dimensional arrays (2d,3d,.....)

1) Single dimensional arrays (1d): -

A Single dimensional array represents is a single row or a single column of an elements.

Row - Horizontal arrangement, Column - Vertical arrangement.

For example marks obtained by a student in 5 subjects are Single dimensional array.



Creating a Single dimensional array: - There are 2 ways

- 1) We can declare a Single dimensional array & directly store the elements in to it.

EX: - int marks[]={50,55,99,86,72};

50	55	99	86	72
marks [0]	[1]	[2]	[3]	[4]

marks[i] → i is called index. Index represents the position number of the element in the array.

The Single dimensional array is only one index. That is i.

float sal[]={9000f,1200.50f,5678.34f};

char ch[]={'a','b','c','d'};

String name[]={"raju","sudheer","sunil","latha"};

- 2) We can allot for Single dimensional array using new operator and later store the elements.

```
int marks[ ]=new int[5];
marks[0]=54; marks[1]=65; ----- marks[4]=98;
double x[ ]=new double[100];
String name[ ]=new String[50];
```

Note: - Array name .length (is a property of any array) gives the size of the array.

array.length

→// total marks & percentage

```
import java.io.*;
class SingleDimensionalArray
{
    public static void main(String[] args) throws IOException
    {
        // to accept data from keyboard
        BufferedReader br=new BufferedReader(new
                                         InputStreamReader(System.in));
        System.out.print("How many subjects ? =");
```

```
int n=Integer.parseInt(br.readLine( ));
// create an array with size n
int marks[]={};
// store marks into marks[]
for(int i=0;i<n;i++)
{
    System.out.println("Enter marks :");
    marks[i]=Integer.parseInt(br.readLine( ));
}
// display the marks & percentage
System.out.print("The marks are :");
    int tot=0;
for(int i=0;i<n;i++)
{
    System.out.println(marks[i]);
    tot+=marks[i];
}
System.out.println("Total marks = "+tot);
// calculate percentage
float percent=(float)tot/n;
System.out.println("Percentage = "+percent);
System.out.println("Size of marks = "+marks.length);
}
```

D:\bkr\Core java>javac SingleDimensionalArray.java

D:\bkr\Core java>java SingleDimensionalArray

How many subjects ? =3

Enter marks :

36

Enter marks :

75

Enter marks :

97

The marks are : 36 75 97

Total marks = 208

Percentage = 69.333336

Size of marks = 3

2) **Two dimensional array (2d)** - A two dimensional array represents several rows & columns of data.

For example obtained by a group of students in 5 different subjects.

Creating Two dimensional array: - There are 2 ways

- 1) We can declare initialize a two dimensional array with elements.

```
int marks[ ][ ]={{50,51,52,53,54},
                 {60,61,62,63,64},
                 {70,71,72,73,74}};
```

r0	50	51	52	53	54
r1	60	61	62	63	64
r2	70	72	73	73	74
	c1	c2	c3	c4	c5

marks[i][j] → indexes of marks.

A Two dimensional array has 2 indexes. They represent row position & column position numbers.

A Two dimensional array is a combination of single dimensional arrays.

A three dimensional array has 3 indexes. A three dimensional array is a combination of several Two dimensional array.

- 2) We can created two dimensional arrays using new operator & stored the elements later

```
int marks[ ][ ]=new int[3][5];
float sal[ ][ ]=new float[5][10];
String str[ ][ ]=new String[3][10];
```

→// Two dimensional array

```
import java.io.*;
class TwoDimensionalArray
{
    public static void main(String[] args)
    {
        // take Two dimensional array
        float x[][]={{1.1f,1.2f,1.3f,1.4f},
                     {2.1f,2.2f,2.3f,2.4f},
                     {3.1f,3.2f,3.3f,3.4f}};
        // read from the Two dimensional array & display
        for(int i=0;i<3;i++)
        {
            for(int j=0;j<4;j++)
            {
                System.out.print(x[i][j]+"\t");
            }
            System.out.println();
        }
    }
}
```

```

    }
}

```

D:\bkr\Core java>javac TwoDimensionalArray.java

D:\bkr\Core java>java TwoDimensionalArray

```

1.1  1.2  1.3  1.4
2.1  2.2  2.3  2.4
3.1  3.2  3.3  3.4

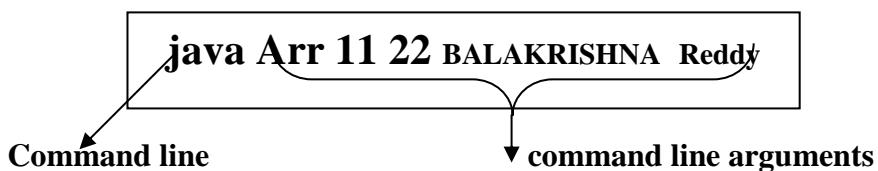
```

A two dimensional array represent a matrix.

Method parameter: - Method parameter is a variable that receive data from out side into the method.

Command line arguments: -

Command line arguments are values passed to main method from system prompt.



JVM allots memory in run time.

args.length → Size of arguments

→// argument length example

```

class Argument
{
    public static void main(String args[])
    {
        int n=args.length;
        System.out.println("no. of arguments:="+n);
        for(int i=0;i<n;i++)
        {
            System.out.println(args[i]);
        }
    }
}

```

D:\bkr\Core java>javac Argument.java

D:\bkr\Core java>java Argument 12 34 54 Sunil Kumar Reddy

no. of arguments:=6

12

34

54

Sunil

Kumar

Reddy

```
→ // to find sum of two matrices
import java.io.*;
class SumMatrix
{
    public static void main(String[] args) throws IOException
    {
        BufferedReader br=new BufferedReader(new
                                         InputStreamReader(System.in));
        System.out.print("enter r value : ");
        int row=Integer.parseInt(br.readLine( ));
        System.out.print("enter c value : ");
        int col=Integer.parseInt(br.readLine( ));
        int a[][]=new int[row][col];
        System.out.println("Enter first matrix elements :\n");
        for(int i=0;i<row;i++)
        {
            for(int j=0;j<col;j++)
            {
                a[i][j]=Integer.parseInt(br.readLine( ));
            }
        }
        System.out.println("First matrix elements are:\n");
        for(int i=0;i<row;i++)
        {
            for(int j=0;j<col;j++)
            {
                System.out.print(a[i][j]+"\t");
            }
            System.out.println();
        }
        int b[][]=new int[row][col];
        System.out.println("Enter second matrix elements :\n");
        for(int i=0;i<row;i++)
        {
            for(int j=0;j<col;j++)
            {
                b[i][j]=Integer.parseInt(br.readLine( ));
            }
        }
        System.out.println("Second matrix elements are:\n");
        for(int i=0;i<row;i++)
        {
            for(int j=0;j<col;j++)
            {
                System.out.print(b[i][j]+"\t");
            }
            System.out.println();
        }
    }
}
```

```
System.out.println("Sum matrix is : ");
int c[][]=new int[row][col];
for(int i=0;i<row;i++)
{
    for(int j=0;j<col;j++)
    {
        c[i][j]=a[i][j]+b[i][j];
        System.out.print(c[i][j]+"\t");
    }
    System.out.println();
}
}
```

D:\bkr\Core java>javac SumMatrix.java
D:\bkr\Core java>java SumMatrix

enter r value : 2
enter c value : 3
Enter first matrix elements :

1
2
3
4
5
6

First matrix elements are:

1 2 3
4 5 6

Enter second matrix elements :

7
8
9
10
11
12

Second matrix elements are:

7 8 9
10 11 12

Sum matrix is :
8 10 12
14 16 18

→ // to find product of two matrices

```
import java.io.*;
class Prodmatrix
{
    public static void main(String[] args) throws IOException
    {
        BufferedReader br=new BufferedReader(new
                                         InputStreamReader(System.in));
        System.out.print("enter r value : ");
        int row1=Integer.parseInt(br.readLine( ));
        System.out.print("enter c value : ");
        int col1=Integer.parseInt(br.readLine( ));
        int a[][]=new int[row1][col1];
        System.out.println("Enter first matrix elements :\n");
        for(int i=0;i<row1;i++)
        {
            for(int j=0;j<col1;j++)
            {
                a[i][j]=Integer.parseInt(br.readLine( ));
            }
        }
        System.out.println("First matrix elements are:\n");
        for(int i=0;i<row1;i++)
        {
            for(int j=0;j<col1;j++)
            {
                System.out.print(a[i][j]+"\t");
            }
            System.out.println();
        }
        System.out.println("enter r value :");
        int row2=Integer.parseInt(br.readLine( ));
        System.out.println("enter c value : ");
        int col2=Integer.parseInt(br.readLine( ));
        int b[][]=new int[row2][col2];
        System.out.println("Enter second matrix elements :\n");
        for(int i=0;i<row2;i++)
        {
            for(int j=0;j<col2;j++)
            {
                b[i][j]=Integer.parseInt(br.readLine( ));
            }
        }
        System.out.println("Second matrix elements are:\n");
        for(int i=0;i<row2;i++)
        {
            for(int j=0;j<col2;j++)
            {
```

```

        System.out.print(b[i][j]+"\t");
    }
    System.out.println();
}
if(row2==col1)
{
    System.out.println("Prod matrix is : ");
    int c[][]=new int[row1][col2];
    for(int i=0;i<row1;i++)
    {
        for(int j=0;j<col2;j++)
        {
            for(int k=0;k<row2;k++)
            {
                c[i][j]+=a[i][k]*b[k][j];
            }
        }
    }
    for(int i=0;i<row1;i++)
    {
        for(int j=0;j<col2;j++)
        {
            System.out.print(c[i][j]+"\t");
        }
    }
    System.out.println();
}
else
{
    System.out.println("can not product the matrix");
}
}
}

```

D:\bkr\Core java>javac Prodmatrix.java

D:\bkr\Core java>java Prodmatrix

enter r value : 3

enter c value : 2

Enter first matrix elements :

1
2
3
4
5
6

First matrix elements are:

1 2
3 4
5 6

```
enter r value : 2
enter c value : 3
Enter second matrix elements:
```

```
1
2
3
4
5
6
```

Second matrix elements are:

```
1 2 3
4 5 6
```

Prod matrix is :

```
9 12 15
19 26 33
29 40 51
```

→ // to transpose a matrices

```
import java.io.*;
class TransposeMatrix
{
    public static void main(String[] args) throws IOException
    {
        BufferedReader br=new BufferedReader(new
                                         InputStreamReader(System.in));
        System.out.print("enter r value :");
        int row=Integer.parseInt(br.readLine());
        System.out.print("enter c value : ");
        int col=Integer.parseInt(br.readLine());
        int a[][]=new int[row][col];
        System.out.println("Enter matrix elements :\n");
        for(int i=0;i<row;i++)
        {
            for(int j=0;j<col;j++)
            {
                a[i][j]=Integer.parseInt(br.readLine());
            }
        }
        System.out.println("matrix elements are:\n");
        for(int i=0;i<row;i++)
        {
            for(int j=0;j<col;j++)
            {
                System.out.print(a[i][j]+"\t");
            }
            System.out.println();
        }
        System.out.println("Transpose matrix is :");
```

```

for(int i=0;i<col;i++)
{
    for(int j=0;j<row;j++)
    {
        System.out.print(a[j][i]+"\t");
    }
    System.out.println();
}
}
}

```

D:\bkr\Core java>javac TransposeMatrix.java

D:\bkr\Core java>java TransposeMatrix

enter r value : 2

enter c value : 3

Enter matrix elements :

1
2
3
4
5
6

matrix elements are:

1 2 3
4 5 6

Transpose matrix is :

1 4
2 5
3 6

Anonymous Arrays:

- Sometimes we can create an array without name such type of nameless arrays are called anonymous arrays.
- The main objective of anonymous arrays is "just for instant use".
- We can create anonymous array as follows.
- new int[]{10,20,30,40};(valid)
- new int[][]{{10,20},{30,40}};(valid)
- At the time of anonymous array creation we can't specify the size otherwise we will get compile time error.

Example:

```

new int[3]{10,20,30,40}; //C.E: ';' expected (invalid)
new int[] {10,20,30,40}; (valid)

```

Based on our programming requirement we can give the name for anonymous array then it is no longer anonymous.

Example:

```
int[] a=new int[]{10,20,30,40}; (valid)
```

```
class Test
{
    public static void main(String[] args)
    {
        System.out.println(sum(new int[]{10,20,30,40}));//100
    }
    public static int sum(int[] x)
    {
        int total=0;
        for(int x1:x)
        {
            total=total+x1;
        }
        return total;
    }
}
```

Jagged array

It is a new feature supported by Java. In **Jagged arrays**, each row, in a two-dimensional array, may contain different lengths. Let us design a two-dimensional array with 4 rows where the first row contains 4 elements, the second row with 1 element, the third row with 2 elements and the fourth row with 3 elements.

Following is the schematic representation of the array.

Columns	0	1	2	3
0	44	55	66	77
1	36			
2	87	97		
3	68	78	88	

Figure : Varying columns 2D array - matrix form

```
class JaggedArrays
{
    public static void main(String args[])
    {
        int student[][] = new int[4][];
        student[0] = new int[4];
        student[1] = new int[1];
        student[2] = new int[2];
        student[3] = new int[3];
        student[0][0] = 4;
        student[0][1] = 6;
        student[0][2] = 8;
        student[0][3] = 77;
        student[1][0] = 25;
        student[2][0] = 21;
        student[2][1] = 92;
        student[3][0] = 6;
        student[3][1] = 78;
        student[3][2] = 88;
        System.out.println("student[3][1] marks: " + student[3][1]);
    }
}
```

PALLA BALAKRISHNA REDDY

101

```
System.out.println("\nMatrix Form");

for(int i = 0; i < student.length; i++)
{
    for(int j = 0; j < student[i].length; j++)
    {
        System.out.print(student[i][j] + "\t");
    }
    System.out.println();
}
}
```

Copying a java array

We can copy an array to another by the **arraycopy** method of System class.

Syntax of arraycopy method

```
public static void arraycopy( Object src, int srcPos, Object dest, int destPos, int length )
```

Example of arraycopy method

```
class TestArrayCopyDemo {
    public static void main(String[] args) {
        char[] copyFrom = { 'd', 'e', 'c', 'a', 'f', 'f', 'e',
                           'i', 'n', 'a', 't', 'e', 'd' };
        char[] copyTo = new char[7];

        System.arraycopy(copyFrom, 2, copyTo, 0, 7);
        System.out.println(new String(copyTo));
    }
}
```

OOPS

OOPS is a programming approach which provides solution to problems with the help of algorithms based on real world. It uses real world approach to solve a problem. So object oriented technique offers better and easy way to write program than procedural programming model such as C, ALGOL, PASCAL.

→ What is software requirement specification (SRS)?

A) SRS is a document that contains client requirements and strategies to fulfill those requirements. SRS is prepared by project leader.

→ What is software design specification(SDS)?

A) SDS is a document that contains I/O screens, PSUDO code, DFD's dataflow diagrams, Black end data bases etc.., it is prepared by module leader.

C, PASCAL, FORTRAN, COBAL → these languages are called procedural languages. Because they follow procedure oriented approach

The languages c++ & java are called object oriented languages. Because they follow object oriented approach.

→1) In procedure oriented approach a programmer thinks about the problem and its solution. We need another approach where the behavior of entire system is studied before developing the software.

(So we can create full flexed procedure systems).

→2) In procedure oriented approach when the code size exceeds 10,000 lines & before reaching 1,00,000 lines, at a particular level the programs are suddenly loosing control on the code. So we need another approach where we can develop bigger & complex projects.

→3) Programming in procedure oriented approach is completely unnatural. We need another approach where programming reflex human being life.

Because of the above approaches they new approach called object orient approach is developed by computer scientists. It is also called OOPS. (Object oriented programming language).

Features of OOPS: -

1) Object/class: - An object is anything that exists in the real world. An object contains properties & actions. Properties are represented by variables, actions are represented by methods.

A class is a group name which specifies properties & actions of an object. A class also contains variables & methods.

2) Encapsulation: - Taking the data & methods as a single unit is called encapsulation. Class is example of encapsulation. It is a protective mechanism. This main advantage is one class method will mixed in another class.

3) Abstraction: -Hiding unnecessary data from the user called abstraction.

Ex: - Car

4) Inheritance: - Producing new class from existing class is called inheritance. Re using the code without re writing it is the advantage of inheritance.

5) Polymorphism: - Poly – many, morphs – forms

The ability to exist in different form is called polymorphism, if same method is called performing several tasks.

6) Message passing: - Calling a method & passing data to it is called message passing.

→ What is the difference between object oriented programming language & object based programming language?

A) Object oriented programming languages follow all the features of OOP's.

Ex: - C++, Java, Small Talk, Simula 67

Object based programming language follow all the features of OOP's, expect inheritance.

Ex: - Java script, VB script.

Class is a model for creating the object based on the class objects is created.

Instance of a class: - Instance means physically exists.

Objects are created by JVM on heap at run time dynamically. In java executing is created only in dynamic memory, no static memory in java.

Class in Java

A class is a group of objects that has common properties. It is a template or blueprint from which objects are created.

A class in java can contain:

- **data member**
- **method**
- **constructor**
- **block**
- **class and interface**

Syntax to declare a class:

```
class <class_name>{
    data member;
    method;
}
```

Rules for Java Class

- A class can have only public or default(no modifier) access specifier.
- It can be either abstract, final or concrete (normal class).
- It must have the class keyword, and class must be followed by a legal identifier.
- It may optionally extend one parent class. By default, it will extend java.lang.Object.
- It may optionally implement any number of comma-separated interfaces.
- The class's variables and methods are declared within a set of curly braces {}.
- Each .java source file may contain only one public class. A source file may contain any number of default visible classes.
- Finally, the source file name must match the public class name and it must have a .java suffix.

→// a class & object

```
class Person
{
    // properties instance vars
    String name;
    int age;
```

```

// action methods
public void talk( )
{
    System.out.println("Hello I am "+name);
    System.out.println("My age is "+age);
}
class InstanceTest
{
    public static void main(String[] args)
    {
        // create an obj to person
        Person p1;
        p1=new Person( );
        System.out.println(p1.hashCode( ));
        p1.name=" BALAKRISHNA REDDY ";
        p1.age=23;
        p1.talk( );
    }
}

```

D:\bkr\Core java>javac InstanceTest.java

D:\bkr\Core java>java InstanceTest

8567361

Hello I am BALAKRISHNA REDDY

My age is 23

Instance variables are variables whose copy is available in the object.

When the instance variables are not initialized by the programmer, java compiler adds some code to the class where it initializes. The instance variables with default values are shown below.

Class is also called a data type. It is called the user data type.

<u>Data Type</u>	<u>default values</u>
int	0
float	0.0
double	0.0
char	a space(single space store)

String	null
Any class	null
boolean	false

Initializing the instance variables: - Initializing means storing starting value. Initializing the instance variables are various types.

Type1: - We can initialize one class instance variables in another class.

Private: - Private is called access specifiers. Access specifier is a key word that specifies how to access or read the member of a class or the class it self.

A class can contain any of the following variable types.

- **Local variables:** Variables defined inside methods, constructors or blocks are called local variables. The variable will be declared and initialized within the method and the variable will be destroyed when the method has completed.
- **Instance variables:** Instance variables are variables within a class but outside any method. These variables are instantiated when the class is loaded. Instance variables can be accessed from inside any method, constructor or blocks of that particular class.
- **Class variables:** Class variables are variables declared within a class, outside any method, with the static keyword.

There are 4 access specifiers.

- 1) **Private:** - private members are not available out side the class.
- 2) **Public:** - public members are available any where out side the class.
- 3) **Default:** - Default members are available out side the class.

Note: - If the user does not use any access specifier then java compiler uses default specifier.

- 4) **Protected:** - protected members are available out side the class.

→ Private can not be used before the class name. In type1 initialization, if the instance variables are declared as private, they are not available to other classes.

"The difference between class and object..?"

Class	Object
1) A <i>class</i> is a way of binding the data and associated methods in a single unit.	1) <i>Class variable</i> is known as an <i>object</i> .
2) Whenever we start executing a JAVA program, the <i>class</i> will be loaded into main memory with the help of class loader subsystem (a part of JVM) only once.	2) After loading the <i>class</i> into main memory, <i>objects</i> can be created in n number.
3) When the <i>class</i> is defined there is no memory space for <i>data members</i> of a <i>class</i> .	3) When an <i>object</i> is created we get the memory space for <i>data members</i> of the <i>class</i> .

Difference between Class and Object

	Class	Object
1	Class is a container which collection of variables and methods.	object is a instance of class
2	No memory is allocated at the time of declaration	Sufficient memory space will be allocated for all the variables of class at the time of declaration.
3	One class definition should exist only once in the program.	For one class multiple objects can be created.
4		

→ /* this example demonstrate how to pass an objects to a function & how to return object */

```
import java.io.*;
class Distance
{
    private int feet;
    private float inches;
    void showDistance( )
    {
        System.out.println(feet+"-"+inches);
    }
    void setDistance(int ft,float in)
    {
        feet=ft;
```

```
    inches=in;
}
void getDistance( ) throws IOException
{
    BufferedReader br=new BufferedReader(new
                                    InputStreamReader(System.in));
    System.out.print("enter feet : ");
    feet=Integer.parseInt(br.readLine( ));
    System.out.print("enter inches : ");
    inches=Float.parseFloat(br.readLine( ));
}
Distance addDistance(Distance d2)
{
    Distance temp=new Distance( );
    temp.feet=0;
    temp.inches=inches+d2.inches;
    while(temp.inches>=12)
    {
        temp.inches=temp.inches-12;
        temp.feet++;
    }
    temp.feet=temp.feet+feet+d2.feet;
    return temp;
}
}
class DistanceAddition
{
    public static void main(String[] args) throws IOException
    {
        Distance d1=new Distance( );
        Distance d2=new Distance( );
        Distance d3=new Distance( );
        d1.getDistance( );
        d1.showDistance( );
        d2.getDistance( );
        d2.showDistance( );
```

```
        d3=d1.addDistance(d2);
        System.out.println("result Distance =");
        d3.showDistance( );
    }
}
```

D:\bkr\Core java>javac DistanceAddition.java

D:\bkr\Core java>java DistanceAddition

enter feet : 10

enter inches : 20

10-20.0

enter feet : 20

enter inches : 10

20-10.0

result Distance =

32-6.0

Creating multiple objects by one type only

```
class Rectangle{
    int length;
    int width;

    void insert(int l,int w){
        length=l;
        width=w;
    }
    void calculateArea(){System.out.println(length*width);}

}
public static void main(String args[]){
    Rectangle r1=new Rectangle(),r2=new Rectangle(); //creating two objects

    r1.insert(11,5);
    r2.insert(3,15);
    r1.calculateArea();
    r2.calculateArea();
}
```

Constructors: - A constructor is similar to a method. Which is used to initialize the instance variables?

- 1) Constructors name & class name must be same.

EX: - Person()

- 2) A constructor without parameters is called default constructors. A constructor with one or more parameters is called parameterized.

ADVANTAGES of constructors:

1. A constructor eliminates placing the default values.
2. A constructor eliminates calling the normal method implicitly.

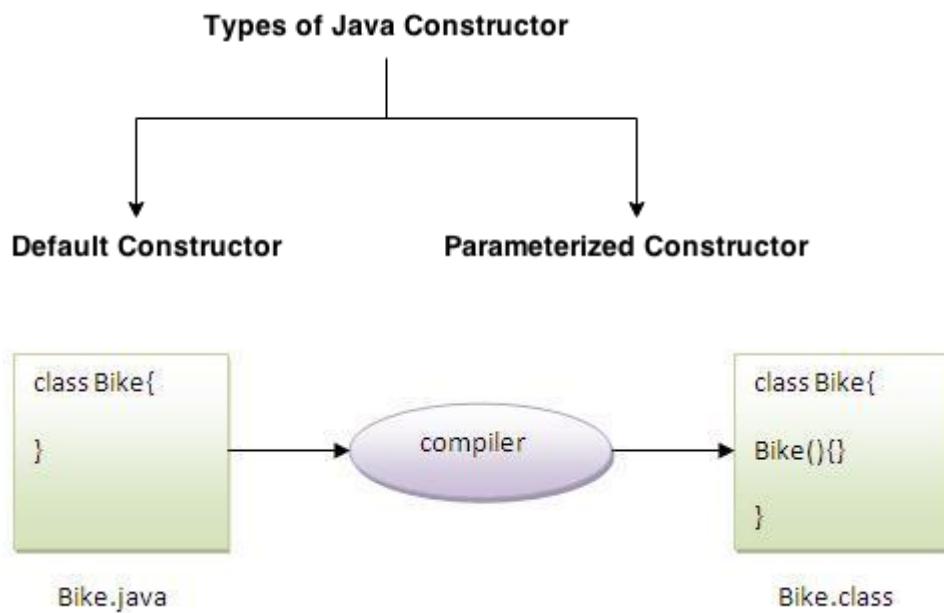
RULES/PROPERTIES/CHARACTERISTICS of a constructor:

1. Constructor name must be similar to name of the class.
2. Constructor should not return any value even void also (if we write the return type for the constructor then that constructor will be treated as ordinary method).
3. Constructors should not be static since constructors will be called each and every time whenever an object is creating.
4. Constructor should not be private provided an object of one class is created in another class (constructor can be private provided an object of one class created in the same class).
5. Constructors will not be inherited at all.
6. Constructors are called automatically whenever an object is creating.

Types of java constructors

There are two types of constructors:

1. **Default constructor (no-arg constructor)**
2. **Parameterized constructor**



For example:

```

class Test
{
int a, b;
Test ()
{
System.out.println ("I AM FROM DEFAULT CONSTRUCTOR...");
a=10;
b=20;
System.out.println ("VALUE OF a = "+a);
System.out.println ("VALUE OF b = "+b);
}
};

class TestDemo
{
public static void main (String [] args)
{
Test t1=new Test ();
}
}

EX: -
→// default constructor
Person()
{
  
```

```
name="Sudheer";
age=23;
}
```

A constructor does not return any value. A constructor is called & executed at the time of creating the object automatically. A constructor is only once for object.

EX: -

```
→// parametrized constructor
Person(String s,int i)
{
    name=s;
    age=i;
}
Person p2=new Person("Sudheer",23);
Person p3=new Person("Sunil",24);
```

→ //use of constructor

```
class Counter
{
    int count;
    Counter( )
    {
        count=0;
        System.out.println("Constructor called");
    }
    void showCount()
    {
        System.out.println("count : "+count);
    }
    void incrementCount( )
    {
        count++;
    }
    void decrementCount()
    {
        count--;
    }
}
class ConstructorDemo
```

```
{  
public static void main(String[] args)  
{  
    Counter c1=new Counter();  
    Counter c2=new Counter();  
    c1.incrementCount();  
    c2.incrementCount();  
    c2.incrementCount();  
    c1.decrementCount();  
    c2.decrementCount();  
    c1.showCount();  
    c2.showCount();  
}  
}
```

D:\bkr\Core java>javac ConstructorDemo.java

D:\bkr\Core java>java ConstructorDemo

Constructor called

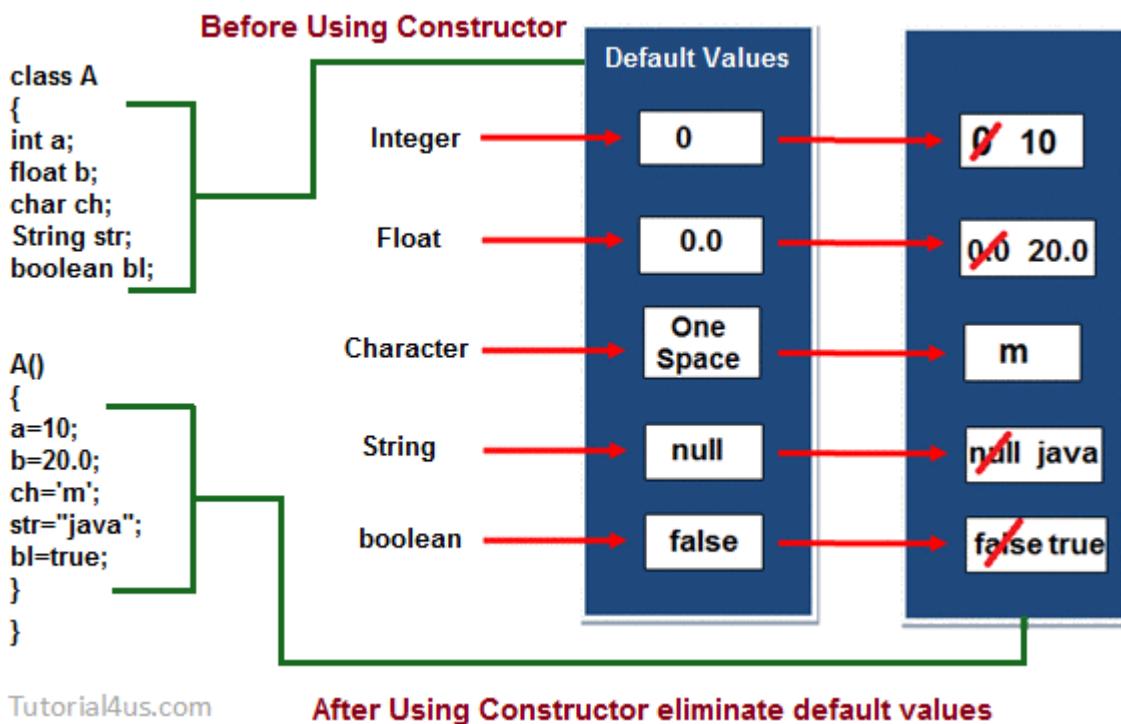
Constructor called

count : 0

count : 1

How Constructor eliminate default values ?

Constructor are mainly used for eliminate default values by user defined values, whenever we create an object of any class then its allocate memory for all the data members and initialize there default values. To eliminate these default values by user defined values we use constructor.



Difference between constructor and method in java

There are many differences between constructors and methods. They are given below.

Java Constructor	Java Method
Constructor is used to initialize the state of an object.	Method is used to expose behaviour of an object.
Constructor must not have return type.	Method must have return type.
Constructor is invoked implicitly.	Method is invoked explicitly.
The java compiler provides a default constructor if you don't have any constructor.	Method is not provided by compiler in any case.
Constructor name must be same as the class name.	Method name may or may not be same as class name.

→ What is the default constructor & parameterized constructor?

A) Default constructor is used to initialize every object with same data.

Parameterized constructor is used to initialize each object with different data.

→ What is constructor overloading?

A) Writing 2 or more constructors with same name with difference in the parameter is called constructor overloading.

```
class Student5{  
    int id;  
    String name;  
    int age;  
    Student5(int i,String n){  
        id = i;  
        name = n;  
    }  
    Student5(int i,String n,int a){  
        id = i;  
        name = n;  
        age=a;  
    }  
    void display()  
{  
    System.out.println(id+" "+name+" "+age);  
}  
    public static void main(String args[]){  
        Student5 s1 = new Student5(111,"Karan");  
        Student5 s2 = new Student5(222,"Aryan",25);  
        s1.display();  
        s2.display();  
    }  
}
```

Can we execute a program without main() method?

Ans) Yes, one of the way is static block but in previous version of JDK not in JDK 1.7.

```
class A3{
    static{
        System.out.println("static block is invoked");
        System.exit(0);
    }
}
```

Methods: -

A method represent a group of stats with performs a task. A method will have 2 parts.

```
public String getName(String st)
```

modifier return-type method-name parameter

1) Method header (or) prototype: - It contain name of the method & method parameters (p1,p2,p3,...) & methods return type.

returntype methodname(para1,para2,.....);

method parameters are variables which receive data from outside.

EX: -

return type → data type

void sum()

double sum(double a,double b)

double sqrt(double x)

double power()

long fact(int num)

```
void calculateTsx(float sal)
```

2) **Method body:** - It represents a group of statements which perform the task.

EX: -method body

```
{  
    Statements;  
}
```

Note: - If a method returns some value we should write return statement in the body.

EX: -

```
return x;  
return 1;  
return x+y;  
return obj;  
return arr;
```

Note: - return statement can return only one entity or 1 value it returns.

A method can not return more than one value.

Parameter Vs. Argument

While talking about method, it is important to know the difference between two terms parameter and argument.

Parameter is variable defined by a method that receives value when the method is called. Parameter are always local to the method they dont have scope outside the method. While argument is a value that is passed to a method when it is called.

```

public void sum( int x, int y )
{
    System.out.println(x+y);
}
public static void main( String[ ] args )
{
    Test b=new Test( );
    b.sum( 10, 20 );
}

```

parameter
argument

→// understanding the methods

```

class Sample
{
    // instance vars
    double d1,d2;
    // Para constructor
    Sample(double a, double b)
    {
        d1=a;
        d2=b;
    }
    // a method
    void sum()
    {
        double d3=d1+d2;
        System.out.println("Sum = "+d3);
    }
}
class MethodTest
{
    public static void main(String[] args)
    {
        // sample object
    }
}

```

```
Sample s=new Sample(20,30.5);
// call the method
s.sum();
}
}
```

D:\bkr\Core java>javac MethodTest.java

D:\bkr\Core java>java MethodTest

Sum = 50.5

Local variable: - It is a variable that is declared locally within a method or constructor. The scope of that variable is limited with in method only or only to that constructor where it is declared.

Methods in java: -

- 1) Static methods
- 2) Instance methods
 - a) Accessor methods
 - b) Mutator methods
- 3) Factory methods

1) Static methods: -

These are the methods which don't act upon the instance variables.

These are declared as static.

```
static double sum(double d1,double d2)
```

Static methods are called using object name.method name

Static methods they can not read instance variables why because the reason is JVM first execute static methods & after that it creates a object next). So at the time of calling static method instance variables are not available. Static methods can acts as static variables. It should be declared as static.

→ What is different between instance variables and static variables?

Difference between non-static and static variable

	Non-static variable	Static variable
1	<p>These variable should not be preceded by any static keyword Example:</p> <pre>class A { int a; }</pre>	<p>These variables are preceded by static keyword.</p> <p>Example</p> <pre>class A { static int b; }</pre>
2	Memory is allocated for these variable whenever an object is created	Memory is allocated for these variable at the time of loading of the class.
3	Memory is allocated multiple time whenever a new object is created.	Memory is allocated for these variable only once in the program.
4	Non-static variable also known as instance variable while because memory is allocated whenever instance is created.	Memory is allocated at the time of loading of class so that these are also known as class variable.
5	Non-static variable are specific to an object	Static variable are common for every object that means there memory location can be sharable by every object reference or same class.
6	<p>Non-static variable can access with object reference.</p> <p>Syntax</p> <pre>obj_ref.variable_name</pre>	<p>Static variable can access with class reference.</p> <p>Syntax</p> <pre>class_name.variable_name</pre>
7.	Instance variables are created on heap.	Static variables are stored on method area.
8	Instance variable are also known as object level data members since they are dependent on objects.	Static variable are also known as class level data members since they are dependent on classes.

Understand static and non-static variable using counter

Program of counter without static variable

In this example, we have created an instance variable named count which is incremented in the constructor. Since instance variable gets the memory at the time of object creation, each object will have the copy of the instance variable, if it is incremented, it won't reflect to other objects. So each objects will have the value 1 in the count variable.

Example

```
class Counter
{
    int count=0;//will get memory when instance is created
    Counter()
    {
        count++;
        System.out.println(count);
    }
    public static void main(String args[])
    {
        Counter c1=new Counter();
        Counter c2=new Counter();
        Counter c3=new Counter();
    }
}
```

Output

```
1
1
1
```

Program of counter by static variable

As we have mentioned above, static variable will get the memory only once, if any object changes the value of the static variable, it will retain its value.

Example

```
class Counter
{
    static int count=0; //will get memory only once
    Counter()
    {
        count++;
        System.out.println(count);
    }
    public static void main(String args[])
    {
        Counter c1=new Counter();
        Counter c2=new Counter();
        Counter c3=new Counter();
    }
}
```

Difference between non-static and static Method

	Non-Static method	Static method
1	<p>These method never be preceded by static keyword Example:</p> <pre>void fun1() { }</pre>	<p>These method always preceded by static keyword Example:</p> <pre>static void fun2() { }</pre>
2	Memory is allocated multiple time whenever method is calling.	Memory is allocated only once at the time of class loading.

3	It is specific to an object so that these are also known as instance method.	These are common to every object so that it is also known as member method or class method.
4	These methods always access with object reference Syntax: Objref.methodname();	These property always access with class reference Syntax: className.methodname();
5	If any method wants to be execute multiple time that can be declare as non static.	If any method wants to be execute only once in the program that can be declare as static .

Note: -

1) Static blocks, static methods & static variables are stored in method area & only one copy is shared by all objects.

2) A static block represents a block of statements declared as static.

3) JVM execute in the following order

- a) static block
- b) static method
- c) instance method.

a) static block: -**→// static test**

```
class StaticTest
{
    static
    {
        System.out.println("static block");
        System.exit(0);
    }
}
```

(or)

→// static test

```
class StaticTest1
{
    static
    {
        System.out.println("static block");
        // System.exit(0);
    }
    public static void main(String[] args)
    {
    }
```

```
        System.out.println("static method");
    }
}
```

D:\bkr\Core java>javac StaticTest1.java

D:\bkr\Core java>java StaticTest1

static block

static method

→ // static block

class StaticBlock

{

static int a=3;

static int b;

static void show(int x)

```
    {      System.out.println("x = "+x);
```

```
          System.out.println("a = "+a);
```

```
          System.out.println("b = "+b);
```

}

static

{

```
    System.out.println("static block initialized");
```

```
    b=a*4;
```

}

}

class Static1Demo

{

```
    public static void main(String[] args)
```

{

```
        Static1 ob=new Static1();
```

```
        ob.show(100);
```

}

}

D:\bkr\Core java>javac Static1Demo.java

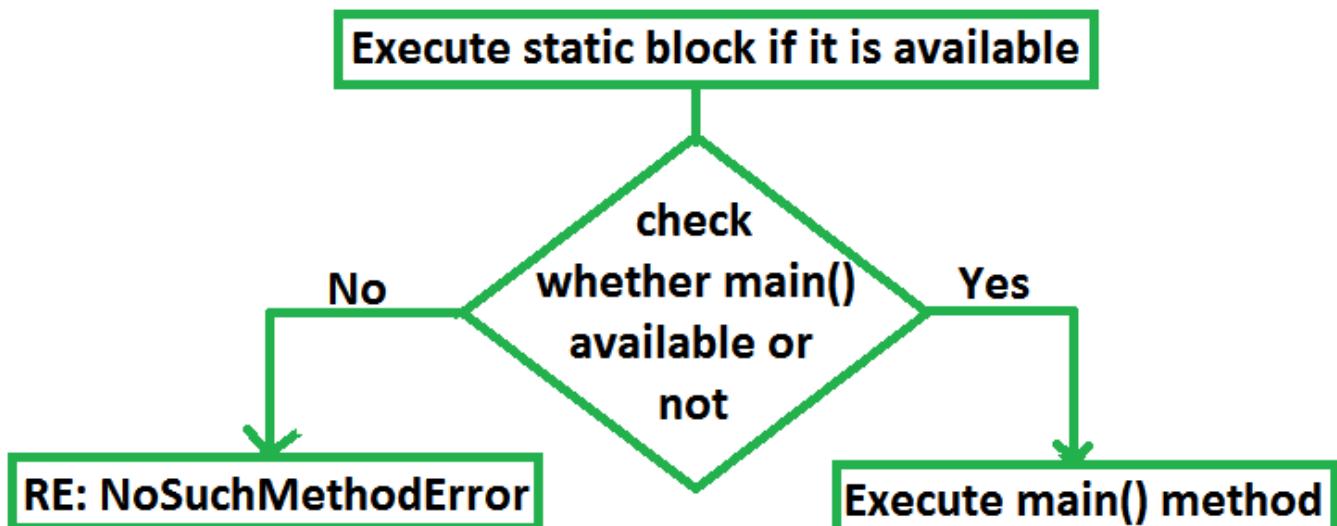
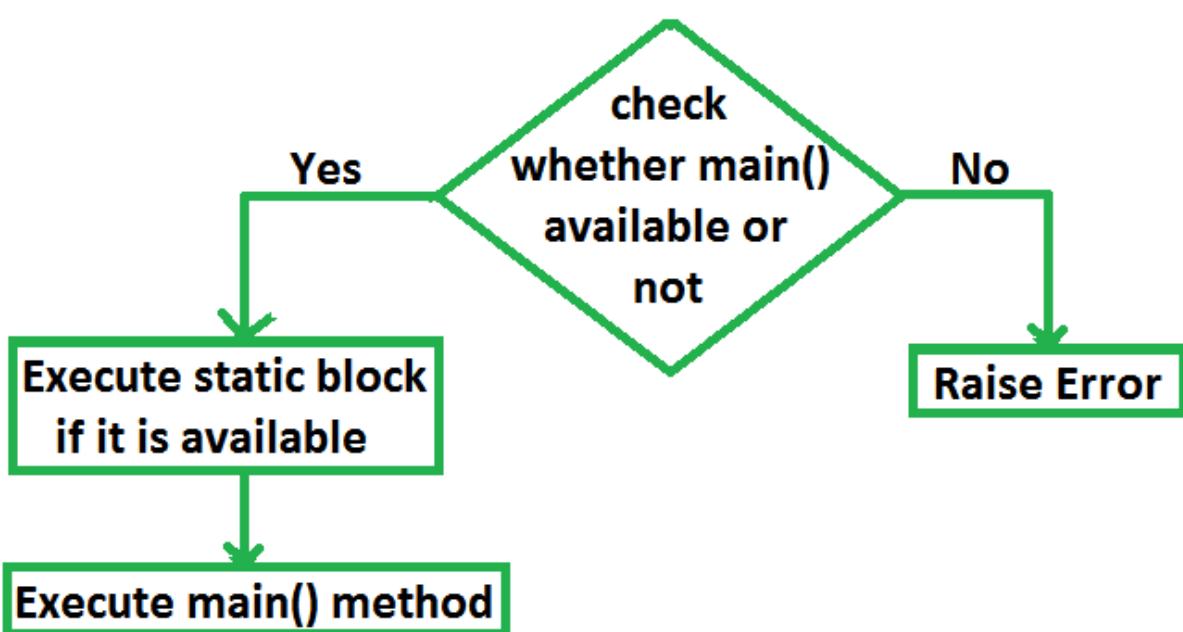
D:\bkr\Core java>java Static1Demo

static block initialized

x = 100

a = 3

b = 12

1.6 version :**1.7 version :**

b) static method: -

→ // static method

```

class Static
{
    int x;
    static int y;
    static void showy( )
    {
        System.out.println(y);
    }
    void showx( )
    {
        System.out.println(x);
    }
}
class StaticMethod
{
    public static void main(String[] args)
    {
        Static ob=new Static();
        ob.y=200;
        ob.x=100;
        ob.showx();
        ob.showy();
    }
}

```

D:\bkr\Core java>javac StaticMethod.java

D:\bkr\Core java>java StaticMethod

100
200

c) Instance methods: - These are those methods which act upon instance variables of a class.

Instance methods are called using object name.

.methodName()

EX: - s.sum()

A method will return the result from where it is called.

EX: -double sum()

```
{
    double d3=d1+d2;
    return d3;
}
calling
double res=s.sum();
System.out.println("result = "+res);
```

These Instance methods are two types

- a) **Accessor methods:** - Accessor methods read the instance variables. They don't modify the instance variables.
- b) **Mutator methods:** - These methods not only read the instance variables but also modify them.

→// accessor & mutator methods

```
class Person
{
    // instance variables
    String name;
    int age;
    char sex;
    // constructor to initialize
    Person(String name,int age,char sex)
    {
        this.name=name;
        this.age=age;
        this.sex=sex;
    }
    // accessor method
    void display()
    {
        System.out.println("name= "+name);
        System.out.println("age ="+age);
        System.out.println("sex =" +sex);
    }
}
```

```
// mutator method
Person modify(Person obj)
{
    obj.name="Latha";
    --obj.age;
    obj.sex='f';
    return obj;
}
class Mutate
{
    public static void main(String args[])
    {
        Person p1=new Person("sudheer",23,'m');
        p1.display();
        Person p2=p1.modify(p1);
        p2.display();
        Person p3=p1.modify(new Person("sunil",30,'m'));
        p3.display();
    }
}
```

D:\bkr\Core java>javac Mutate.java

D:\bkr\Core java>java Mutate

name= sudheer

age =23

sex =m

name= Latha

age =22

sex =f

name= Latha

age =29

sex =f

This: - this is a key word that refers to present class object.

It refers to → present class instance variables

- present class methods.
- present class constructor.

→//this demo

```
class Sample
{
    int n;
    Sample( )
    {
        this(55); // present class constructor
        this.display( ); // present class method
    }
    Sample(int n)
    {
        this.n=n; // present class variable
    }
    void display( )
    {
        System.out.println("n="+n);
    }
}
class ThisDemo
{
    public static void main(String args[])
    {
        Sample s=new Sample( );
        s.display( );
    }
}
```

D:\bkr\Core java>javac ThisDemo.java

D:\bkr\Core java>java ThisDemo

n=55

n=55

3) Factory methods: - A factory method is a method that returns an object of the class to which to belong.

EX: -

```
NumberFormat obj=new NumberFormat.getInstance( );
```

All factory methods are static methods. But all static methods are not factory methods.

→ In how many ways can you create an object in java?

A) Ways of creating an object: -

1) Using new operator

```
Employee obj=new Employee( );
```

2) using factory method

```
NumberFormat obj=new NumberFormat.getInstance( );
```

3) using newInstance() method

```
Class c=Class.forName("Employee");
```

For name method will store employee that class name in an object

This is represented by c.

4) Using cloning()

Cloning means creating bitwise exact copy of an existing object.

Polymorphism: - It represents many forms. If a method performs several tasks, it is called polymorphism. They are 2 types

1) Dynamic polymorphism 2) Static polymorphism

1) Dynamic polymorphism: -

The polymorphism exhibited at run time is called dynamic polymorphism. In this dynamic polymorphism a method call is linked with method body at the time of running the program by JVM. This is also known as dynamic binding or run time polymorphism.

EX: -

Method over loading: - Writing 2 or more methods with the same name, but with a difference in the method signatures is called method over loading.

In method over loading JVM understand in the method is called depending upon the difference in the method signature. The difference may be due to the following.

1) There is a difference in the no. of parameters.

```
void add(int a,int b)
```

```
void add(int a,int b,int c)
```

- 2) There is a difference in the data types of parameters.

```
void add(int a,float b)
```

```
void add(double a,double b)
```

- 3) There is a difference in the sequence of parameters.

```
void swap(int a,char b)
```

```
void swap(char a,int b)
```

Any one of the above differences will make method signature to be different, hence JVM can identifies methods uniquely.

→Method over loading is an example for dynamic polymorphism.

```
import java.io.*;
import java.util.*;
class Bank
{
int acno;
int balance;
String Ifscocode;
void read()
{
Scanner sc = new Scanner(System.in);
balance = sc.nextInt();
}
void transfor(int acno,String Ifscocode)
{
System.out.println("pls tranfor the amount to="+acno+Ifscocode);
}
void transfor(int acno)
{
balance=this.balance;
System.out.println("ls tranfor the amount to="+acno+"ac balance="+balance);
}
}
class BankDemo
{
public static void main(String args[])
{
```

```
Bank b=new Bank();
b.read();
b.transfor(1234,"Hdfc123");
b.transfor(12345677);
}
}
```

→// sum of 2 numbers & sum of 3 numbers

```
class Sample
{
    void add(int a,int b)
    {
        System.out.println("sum of two="+(a+b));
    }
    void add(int a,int b,int c)
    {
        System.out.println("sum of three="+(a+b+c));
    }
}

class Poly
{
    public static void main(String[] args)
    {
        Sample s=new Sample();
        s.add(20,25);
        s.add(20,25,30);
    }
}
```

D:\bkr\Core java>javac Poly.java

D:\bkr\Core java>java Poly

sum of two=45

sum of three=75

Can we overload main() method?

Yes, by method overloading. You can have any number of main methods in a class by method overloading. Let's see the simple example:

```
class Overloading1{
    public static void main(int a){
        System.out.println(a);
    }

    public static void main(String args[]){
        System.out.println("main() method invoked");
        main(10);
    }
}
```

→ What is method signature?

A) Method name & its parameters is called method signature

Method overriding: - writing 2 or more methods in super & sub classes with same name & same signatures is called method overriding. In method overriding JVM executes a method depending on the data type of the object.

→ Method overriding is an example for dynamic polymorphism

→ //dynamic polymorphism

```
class One
{
    void calculate(double x)
    {
        System.out.println("Square value ="+(x*x));
    }
}

class Two extends One
{
    void calculate(double x)
    {
        System.out.println("Square root =" +Math.sqrt(x));
    }
}
```

```

class Poly1
{
    public static void main(String args[])
    {
        Two t=new Two();
        t.calculate(16);
    }
}

```

D:\bkr\Core java>javac Poly1.java
D:\bkr\Core java>java Poly1
Square root =4.0

→ **What is the difference between method over loading & method overriding?**

A) Method over loading: - Writing 2 or more methods with the same name, but with a difference in the method signatures is called method over loading. In method over loading JVM understand which method is called depending upon the difference in the method signature.

Method overriding: - writing 2 or more methods in super & sub classes with same name & same signatures is called method overriding. In method overriding JVM executes a method depending on the data type of the object.

Achieving method overloading & method overriding using instance methods is an example of dynamic polymorphism.

Achieving method overloading & method overriding using instance methods is an example of dynamic polymorphism.

2) Static polymorphism: - The polymorphism exhibited at compile time is called Static polymorphism. Here the compiler knows which method is called at the compilation. This is also called compile time polymorphism or static binding.

Achieving method overloading & method overriding using

1) private 2) static 3) final methods, is an example of Static polymorphism.

Note: - A final method is a method written in a final class. A class is declared as a final is called final class.

Ex: - final class A

Note: - final key word before class name prevents inheritance. We can create sub classes to a final class.

Ex: - final class A

class B extends A // invalid

→ We can not override final methods, only final methods overloading.

Converting 1 data type into another data type is called casting.

→ **What is final?**

A) 1) final is used to declare Constance

Ex: - final double PI 3.14159

2) final key word is used to prevent inheritance.

Ex: - final class A

class B extends A // invalid

Difference between Overloading and Overriding

	Overloading	Overriding
1	Whenever same method or Constructor is existing multiple times within a class either with different number of parameter or with different type of parameter or with different order of parameter is known as Overloading.	Whenever same method name is existing multiple time in both base and derived class with same number of parameter or same type of parameter or same order of parameters is known as Overriding.
2	Arguments of method must be different at least arguments.	Argument of method must be same including order.
3	Method signature must be different.	Method signature must be same.
4	Private, static and final methods can be overloaded.	Private, static and final methods can not be override.
5	Access modifiers point of view no restriction.	Access modifiers point of view not reduced scope of Access modifiers but increased.
6	Also known as compile time polymorphism or static polymorphism or early binding.	Also known as run time polymorphism or dynamic polymorphism or late binding.
7	Overloading can be exhibited both are method and constructor level.	Overriding can be exhibited only at method label.
8	The scope of overloading is within the class.	The scope of Overriding is base class and derived class.
9	Overloading can be done at both static and non-static methods.	Overriding can be done only at non-static method.
10	For overloading methods return type may or may not be same.	For overriding method return type should be same.

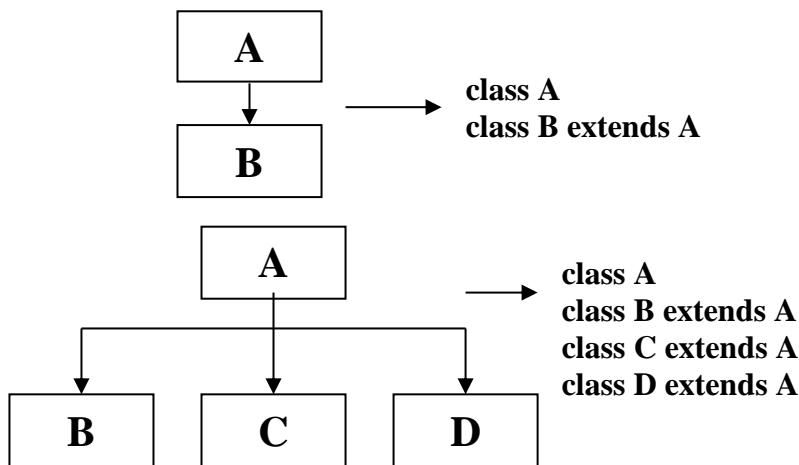
Inheritance: - Creating new class from existing classes such that all the features of existing classes are available to the new classes is called inheritance. Already existing class is called super class & produced class is called sub class.

Syntax: -

class subclass extends super class

Types of inheritance: -

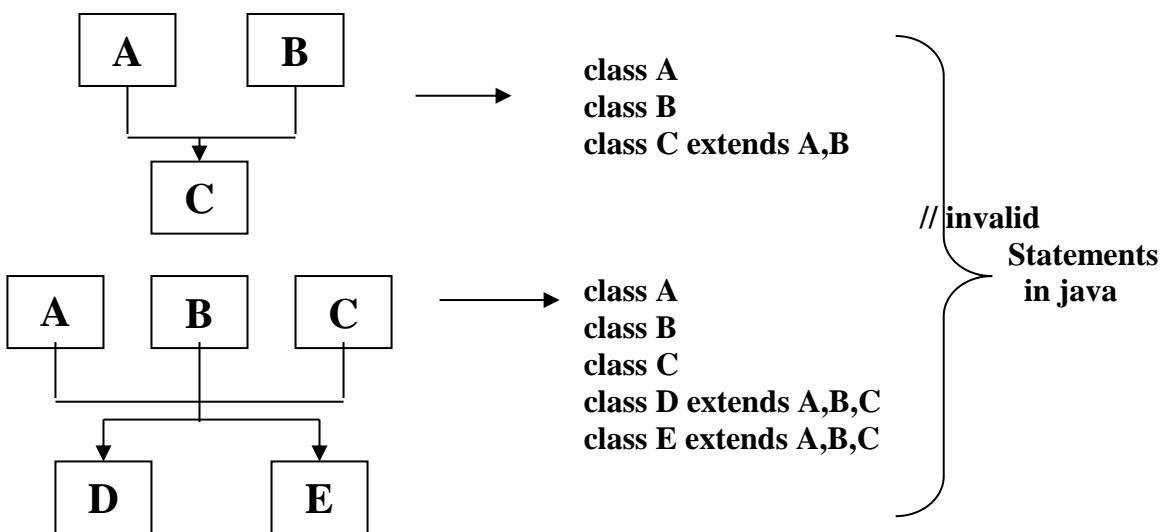
1) **Single inheritance:** - A producing sub class from a single super class is called single inheritance.



2) **Multiple inheritances:** - Producing sub class from multiple super classes is called multiple inheritances. Multiple means more than one.

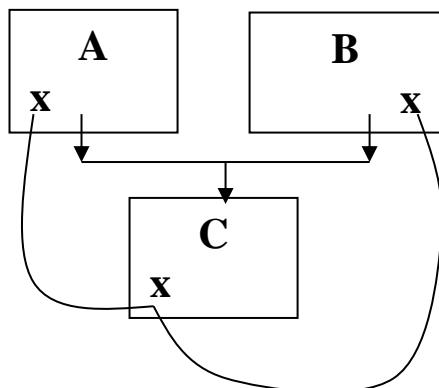
Multiple inheritances are not supported in java.

EX: -



→ **Why multiple inheritances are not supported in java?**

- A) 1) Multiple inheritance leads to confusion for a programmer. Operator over loading, multiple inheritances are not available in java.



2) If the programmer wants we can achieve multiple inheritance using interfaces.

class D extends A,B,C // invalid

class D implements A,B,C// valid

Here A,B,C are not classes only interfaces. So it is not multiple inheritances. This is way of multiple inheritances indirectly.

3) A programmer can achieve multiple inheritances by using single inheritance repeatedly. Because the above reasons the direct multiple inheritances is not available in java.

→// inheritance example of Teacher & student class

```

class Teacher
{
    int id;
    String name;
    String address;
    float sal;
    void setId(int id)
    {
        this.id=id;
    }
    int getId()
    {
        return id;
    }
    void setName(String name)
    {
        this.name=name;
    }
    String getName()
    {
        return name;
    }
    void setAddress(String address)
    {
        this.address=address;
    }
}
  
```

```

        }
        String getAddress( )
        {
            return address;
        }
        void setSal(float sal)
        {
            this.sal=sal;
        }
        float getSal( )
        {
            return sal;
        }
    }
}

```

D:\bkr\Core java>javac Teacher.java

→// inheritance example of Teacher & student class

```

class Student extends Teacher
{
    int marks;
    void setMarks(int marks)
    {
        this.marks=marks;
    }
    int getMarks( )
    {
        return marks;
    }
}

```

D:\bkr\Core java>javac Student.java

→// inheritance example of Teacher & student class

```

class StudentDemo
{
    public static void main(String args[])
    {
        // create student class object
        Student s=new Student( );
        // store data into s
        s.setId(100);
        s.setName("Sudheer");
        s.setAddress("Naghireddy Palli, AP, ATP");
        s.setMarks(662);
        // get data forms
        System.out.println("ID = "+s.getId());
        System.out.println("Name = "+s.getName());
    }
}

```

```

        System.out.println("Address = "+s.getAddress());
        System.out.println("Marks = "+s.getMarks());
    }
}

```

```

D:\bkr\Core java>javac StudentDemo.java
D:\bkr\Core java>java StudentDemo
ID = 100
Name = Sudheer
Address = Naghireddy Palli, AP, ATP
Marks = 662

```

→ Redundancy of the code is a draw back of programming.

Extends is key word used in inheritance. The subclass object contains object contain a copy of super class object in it. We can create one object to only sub class.

→ What is the advantage of inheritance?

A) In inheritance a programmer can reuse already developed code. Because of these developing new programs will become easy. This increases productivity of programmer & hence the overall productivity of the company or organization will also increase.

→ // if super has a default constructor

```

class One
{
    One()
    {
        System.out.println("super");
    }
}
class Two extends One
{
    Two()
    {
        System.out.println("sub");
    }
}
class Const
{
    public static void main(String args[])
    {
        Two t=new Two();
    }
}

```

```

D:\bkr\Core java>javac Const.java
D:\bkr\Core java>java Const
super
sub

```

Super class default constructor is available to sub class by default.

Super class parameterized constructor is not available to sub class by default. Super is a key word that refers to super class. For example

```
→// super used super vars, methods & constructors
class One
{
    int x;
    One(int x)
    {
        this.x=x;
    }
    void show()
    {
        System.out.println("x = "+x);
    }
}
class Two extends One
{
    int y;
    Two(int a,int b)
    {
        super(a); // (or) x=a;
        y=b;
    }
    void show()
    {
        System.out.println("x = "+super.x);
        System.out.println("y = "+y);
    }
}
class Const1
{
    public static void main(String args[])
    {
        Two t=new Two(10,24);
        t.show();
    }
}
```

D:\bkr\Core java>javac Const1.java

D:\bkr\Core java>java Const1

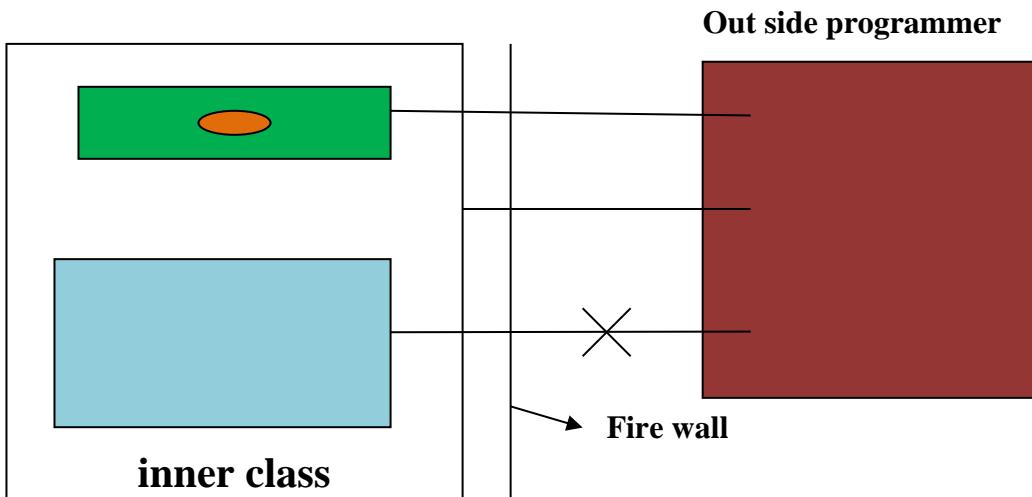
x = 10

y = 24

Super key word is used in sub class only. Calling constructor first declare the super.

Inner class: -

Inner class is a class written in another class. It is a security mechanism. We can use private in inner class. Private keyword can be used before inner class only. Fire wall means optical that checks the authorization of user.



→// inner class demo

```

class BankAcct
{
    private double bal;
    BankAcct(double b)
    {
        bal=b;
    }
    void start(double r)
    {
        Interest in=new Interest(r);
        in.calculateInterest();
    }
    private class Interest
    {
        private double rate;
        Interest(double r)
        {
            rate=r;
        }
        void calculateInterest()
        {
            System.out.println("Balance = "+bal);
            double interest=bal*rate/100;
            System.out.println("interest = "+interest);
            bal+=interest;
            System.out.println("New Balance = "+bal);
        }
    }
}

```

```
class InnerDemo
{
    public static void main(String args[])
    {
        BankAcct account=new BankAcct(20000);
        account.start(7.5);
    }
}
```

D:\bkr\Core java>javac InnerDemo.java

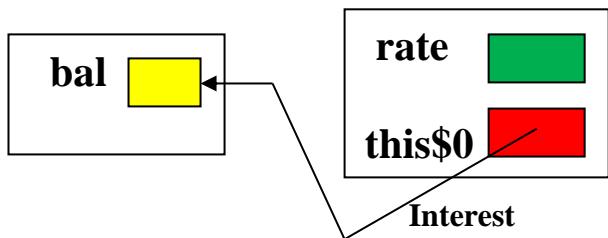
D:\bkr\Core java>java InnerDemo

Balance = 20000.0

interest = 1500.0

New Balance = 21500.0

- 1) Inner class is a written within another class.
- 2) It is a security mechanism.
- 3) Inner class is hidden in outer class from other classes.
- 4) Only inner class can be 'private'
- 5) An object to inner class can not be created in any other class.
- 6) An object to inner class can be created only in its outer class.
- 7) Outer class object and inner class objects are created separately in memory.
- 8) Outer class members are available to inner class.
- 9) Inner class object contains an additional invisible field 'this \$0' that contains outer class reference.



- 10) if same names are used for members ,then outer class members can be referenced in inner class as: outer class.this.member;

Ex: -bankAcct.this.bal.

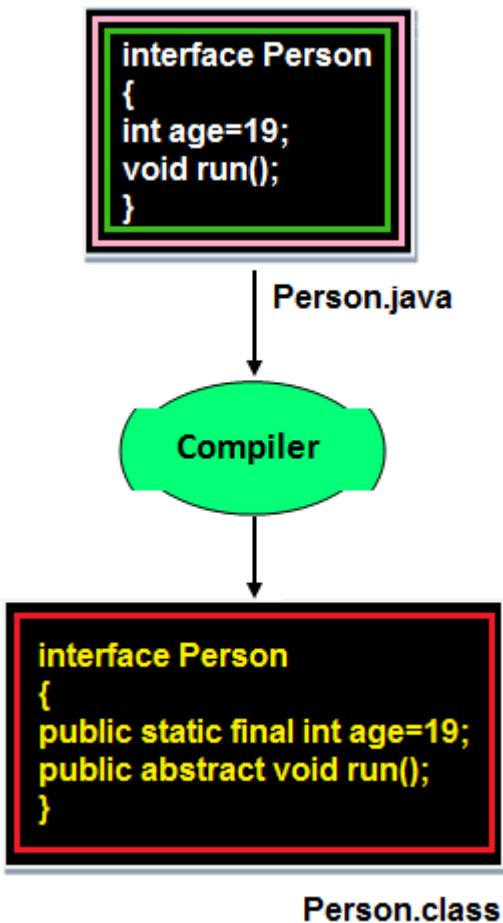
Inner class members are referenced as: this.member

Ex: - this.bal

Interfaces

An interface in java is a blueprint of a class

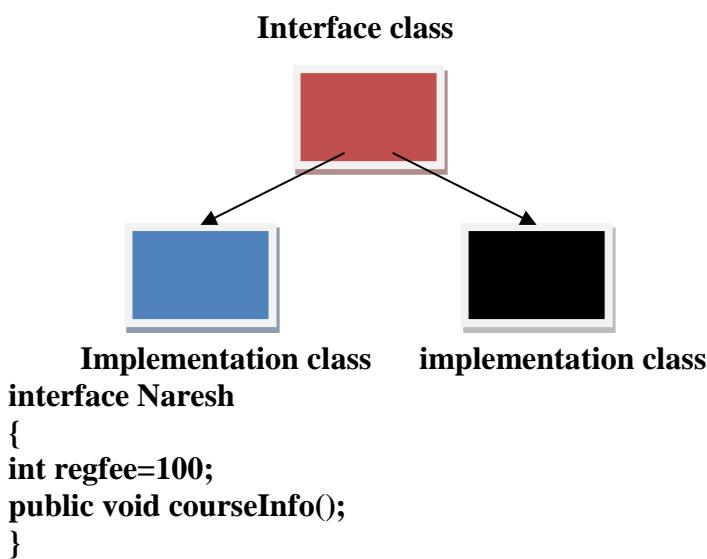
An interface is a specification of method properties. We can not create an object in interface. We create reference variable for interface.



- 1) An interface is a specification of method prototype.
- 2) An interface contains 0 or more abstract methods.
- 3) All the methods in the interface are public and abstract by default.
- 4) An interface can also contain variables which are public, static and final by default.
- 5) We can not create an object to an interface.
- 6) But we can create a reference variable of interface type.

- 7) All the methods of the interface should be implemented in its implementation classes.
- 8) If any method is not implemented, then that class should be declared as ‘abstract’.
- 9) Interface reference can be used to refer to all the objects of its implementation classes.
- 10) Once an interface is written, any third party vendor can provide implementation classes.
- 11) An interface can not implement another interface.
- 12) An interface can extend another interface.
- 13) We can create classes in an interface.
- 14) A class can implement multiple interfaces.

Class A implements B,C...



```

class Java implements Naresh
{
    public void courseInfo()
    {
        System.out.println("this is java course");
        System.out.println("it consists of 3 parts");
        System.out.println("1.core java");
        System.out.println("2.adv java");
        System.out.println("3.j2ee");
    }
}

```

```

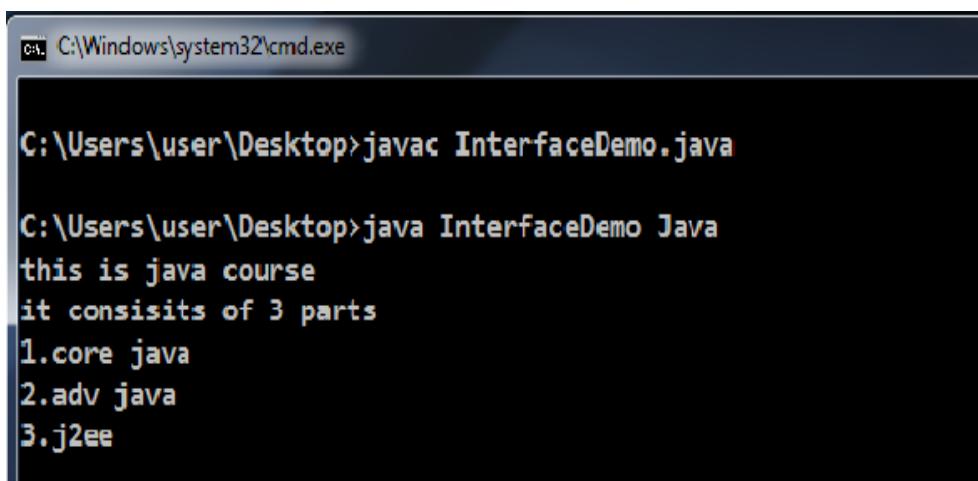
class Php implements Naresh
{
    public void courseInfo()
    {

```

```
System.out.println("this is php course");
System.out.println("it consists of 3 parts");
System.out.println("1.html");
System.out.println("2.java script");
System.out.println("3.php");
}

class Msnet implements Naresh
{
public void courseInfo()
{
System.out.println("this is .net course");
System.out.println("it consists of 3 parts");
System.out.println("1.c#");
System.out.println("2.asp");
System.out.println("3.Ado");
}
}

class InterfaceDemo
{
public static void main(String args[])throws Exception
{
Class c=Class.forName(args[0]);
Naresh n=(Naresh)c.newInstance();
//Naresh n=new Naresh();wrong
n.courseInfo();
}
}
```



The screenshot shows a Windows Command Prompt window titled 'cmd' with the path 'C:\Windows\system32\cmd.exe'. The command 'javac InterfaceDemo.java' is entered and executed. The output shows the program's execution, printing 'this is java course', 'it consists of 3 parts', '1.core java', '2.adv java', and '3.j2ee' to the console.

```
C:\Users\user\Desktop>javac InterfaceDemo.java

C:\Users\user\Desktop>java InterfaceDemo Java
this is java course
it consists of 3 parts
1.core java
2.adv java
3.j2ee
```

```
interface FinAdv
{
double amount=20000000;
```

```
void fd();
void nd();
}

class Hdfc implements FinAdv
{
public void fd()
{
System.out.println("intrest amount for 1 yr fd="+(amount*9.2/100));
System.out.println("intrest amount for 2 yr fd="+(amount*9.6/100));
}
public void nd()
{
System.out.println("intrest amount for 1 yr nd="+(amount*0.99/100));
System.out.println("intrest amount for 2 yr nd="+(amount*1.2/100));
}
}

class Icici implements FinAdv
{
public void fd()
{
System.out.println("intrest amount for 1 yr fd="+(amount*9.6/100));
System.out.println("intrest amount for 2 yr fd="+(amount*9.8/100));
}
public void nd()
{
System.out.println("intrest amount for 1 yr nd="+(amount*0.7/100));
System.out.println("intrest amount for 2 yr nd="+(amount*1.1/100));
}
}

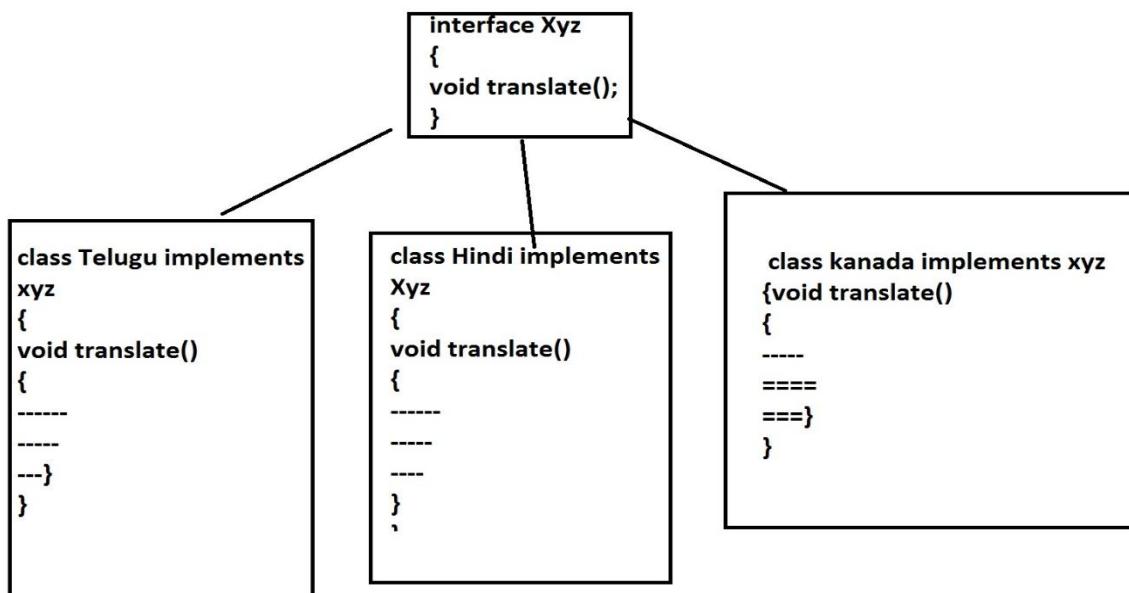
class Sbi implements FinAdv
{
public void fd()
{
System.out.println("intrest amount for 1 yr fd="+(amount*8.6/100));
System.out.println("intrest amount for 2 yr fd="+(amount*9.2/100));
}
public void nd()
{
System.out.println("intrest amount for 1 yr nd="+(amount*1.2/100));
System.out.println("intrest amount for 2 yr nd="+(amount*1.6/100));
}
}

class IntrestDemo
{
```

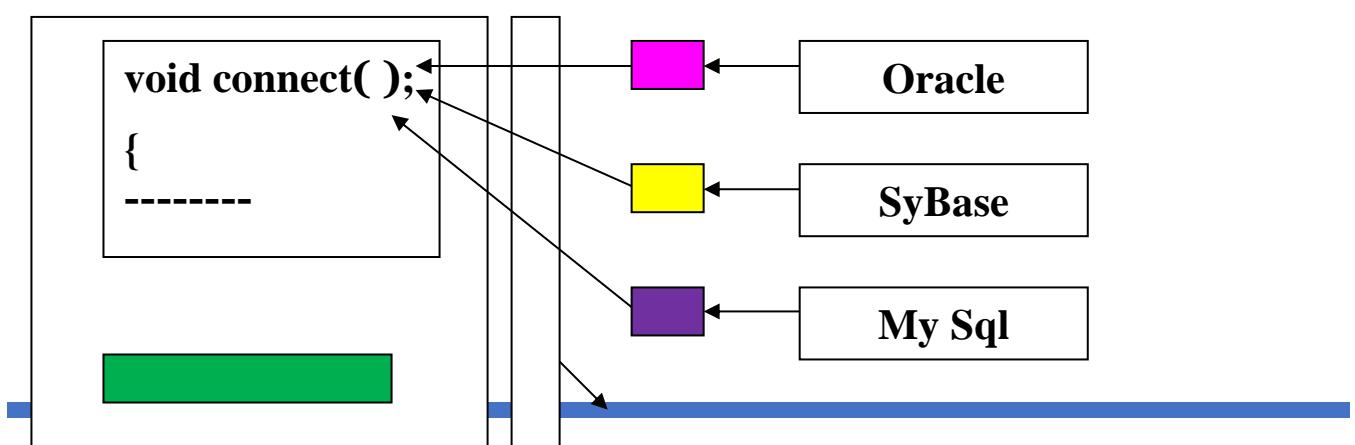
```

public static void main(String args[])
{
FinAdv f=new Hdfc();
System.out.println("\t\t\t Hdfc bank result");
f.fd();
f.nd();
FinAdv f1=new Sbi();
System.out.println("\t\t\t Sbi bank result");
f1.fd();
f1.nd();
FinAdv f2=new Icici();
System.out.println("\t\t\t Icici bank result");
f2.fd();
f2.nd();
}
}

```



Driver: - A driver represents implementation classes of an interface.



API

→ // interface example

```
interface MyInter
{
    void connect( );
}

class OracleDB implements MyInter
{
    public void connect( )
    {
        System.out.println("Connecting to oracle data base .....");
    }
}

class SybaseDB implements MyInter
{
    public void connect( )
    {
        System.out.println("Connecting to Sybase data base .....");
    }
}

class DataBase
{
    public static void main(String args[]) throws Exception
    {
        // accept user data base name through command line arguments store in an object c
        Class c=Class.forName(args[0]);
        // create another object to the class whose name is in c
        MyInter mi=(MyInter)c.newInstance();
        // call convert method of the object
        mi.connect();
    }
}
```

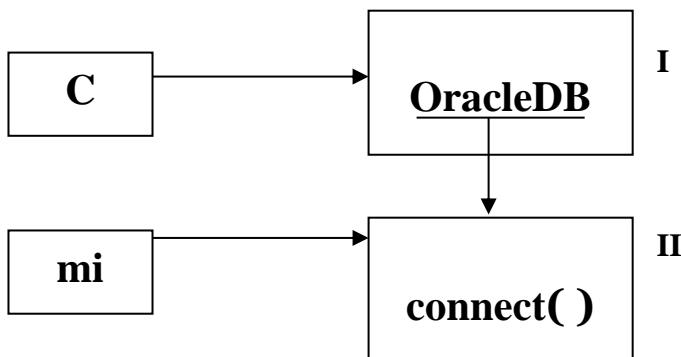
D:\bkr\Core java>javac DataBase.java

D:\bkr\Core java>java DataBase SybaseDB

Connecting to sybase data base.....

D:\bkr\Core java>java DataBase OracleDB

Connecting to oracle data base.....



→// multiple inheritances Using interfaces

EX:1

```

interface Hdfc
{
int hacbal=50000;
}

interface Sbi
{
int sbibal=45000;
}
class Balance implements Hdfc,Sbi
{
void totAmount()
{
int totbal=hacbal+sbibal;
System.out.println("tot amount="+totbal);
}
}
class MulInherDemo
{
public static void main(String args[])
{
Balance b=new Balance();
b.totAmount();
}
}
  
```

Ex:2

```
interface Father
{
    int prop1=500000;
    float ht1=6.2f;
}

interface Mother
{
    int prop2=400000;
    float ht2=5.2f;
}

class Child implements Father,Mother
{
    void property( )
    {
        System.out.println("child property="+ (prop1+prop2));
    }
    void height( )
    {
        System.out.println("child height="+(ht1+ht2)/2);
    }
}
class Multi
{
    public static void main(String args[])
    {
        Child ch=new Child();
        ch.property();
        ch.height();
    }
}
D:\bkr\Core java>javac Multi.java
D:\bkr\Core java>java Multi
child property=900000
child height=5.7
```

Difference between Abstract class and Interface

	Abstract class	Interface
1	It is collection of abstract method and concrete methods.	It is collection of abstract method.
2	There properties can be reused commonly in a specific application.	There properties commonly usable in any application of java environment.
3	It does not support multiple inheritance.	It supports multiple inheritance.
4	Abstract class is preceded by abstract keyword.	It is preceded by Interface keyword.
5	Which may contain either variable or constants.	Which should contain only constants.
6	The default access specifier of abstract class methods are default.	The default access specifier of interface methods are public.
7	These class properties can be reused in other class using extend keyword.	These properties can be reused in any other class using implements keyword.
8	Inside abstract class we can take constructor.	Inside interface we can not take any constructor.
9	For the abstract class there is no restriction like initialization of variable at the time of variable declaration.	For the interface it should be compulsory to initialize of variable at the time of variable declaration.
10	There are no any restriction for abstract class variable.	For the interface variable can not declare variable as private, protected, transient, volatile.
11	There are no any restriction for abstract class method modifier that means we can use any modifiers.	For the interface method can not declare method as strictfp, protected, static, native, private, final, synchronized.

Why interface have no constructor ?

Because, constructor are used for eliminate the default values by user defined values, but in case of interface all the data members are public static final that means all are constant so no need to eliminate these values.

Other reason because constructor is like a method and it is concrete method and interface does not have concrete method it have only abstract methods that's why interface have no constructor.

→ How much memory is taken by class two's object?

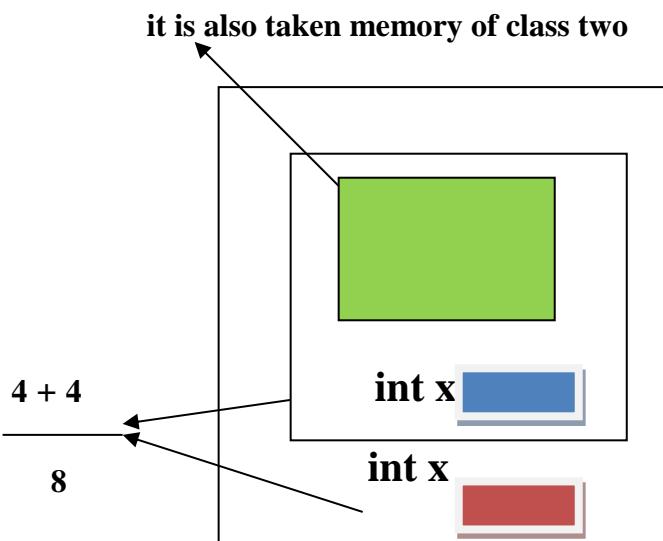
1) 4 b 2) 8 b 3) >8 b 4) <8 b

A) Ans is >8 b because

Ex: -

```
class One
{
    int x;
}

class Two extends One
{
    int x;
}
```



Abstract methods: - When different objects need different implementation of same method then we can not write body for the method. Such a method is called abstract method. It is a method without body. Abstract method should be declared with abstract key word. A class that contains abstract methods is called abstract class. Abstract class should also be declared using the abstract key word. We can not create object to abstract class.

Difference Between Abstract class and Concrete class

Concrete class	Abstract class
Concrete class are used for specific requirement	Abstract class are used for fulfill common requirement.
Object of concrete class can be create directly.	Object of abstract class can not be create directly (can create indirectly).
Concrete class containing fully defined methods or implemented method.	Abstract class have both undefined method and defined method.

→// abstract class

```
import java.util.*;

abstract class Cust
{
    int custid;
    String name;
    float balance;
    Scanner sc=new Scanner(System.in);
```

```
Cust(int custid,String name)
{
this.custid=custid;
this.name=name;
}
void balance()
{
System.out.println("enter balance");
balance=sc.nextFloat();
System.out.println("cust id value=" +custid);
System.out.println("cust name"+name);
System.out.println("cust balance=" +balance);
}
abstract void fdCal();
abstract void inCal();
}
→ // this is a concrete sub class of Hdfc class
class Hdfc extends Cust
{
Hdfc(int custid,String name)
{
super(custid,name);
}
void fdCal()
{
System.out.println("hdfc gives 9.5% fd for 1 yr cal fd");
System.out.println("hdfc gives 9.9% fd for 2 yr cal fd");
}
void inCal()
{
System.out.println("hdfc gives 1% int for 1 yr cal in");
System.out.println("hdfc gives 1.3% int for 2 yr cal in");
}
```

→ // this is a concrete sub class of Icici class

```
class Icici extends Cust
{
Icici(int custid,String name)
{
super(custid,name);
}
void fdCal()
{
System.out.println("icici gives 9.2% fd for 1 yr cal fd");
System.out.println("Sbi gives 9.6% fd for 2 yr cal fd");
}
void inCal()
{
System.out.println("Icici gives 0.9% int for 1 yr cal in");
System.out.println("hdfc gives 1.3% int for 2 yr cal in");
}
}
```

→ // this is a concrete sub class of Sbi class

```
class Sbi extends Cust
{
Sbi(int custid,String name)
{
super(custid,name);
}
void fdCal()
{
System.out.println("Sbi gives 9.8% fd for 1 yr cal fd");
System.out.println("Sbi gives 10.4% fd for 2 yr cal fd");
}
void inCal()
{
System.out.println("sbi gives 1.2% int for 1 yr cal in");
System.out.println("hdfc gives 1.6% int for 2 yr cal in");
}}
```

Main class

```
class AbstactDemo
{
    public static void main(String args[])
    {
        //Cust c=new Cust(123,"balu");is not allowed
        Hdfc h=new Hdfc(1234,"balu");
        Sbi s=new Sbi(254789,"balu");
        Icici i=new Icici(9899854,"balu");

        Cust ref;
        ref=h;

        h.balance();
        h.fdCal();
        h.inCal();
        ref=s;

        s.balance();
        s.fdCal();
        s.inCal();
        ref=i;

        i.balance();
        i.fdCal();
        i.inCal();

    }
}
```

→ // abstract class

```
abstract class Car
{
    // every car will have a regno
    int regno;
    // to store regno
    Car(int r)
    {
        regno=r;
    }
    // every car will have fuel tank
    // mechanism to fill the fuel is same for all cars
    void fillTank( )
    {
        System.out.println("Use car Key");
        System.out.println("And fill the tank");
    }
    /* every car will have steering. But different cars will have different steering mechanisms
    */
    abstract void steering(int direction);
    /* every cars will have breaks. But different cars will have different breaking mechanisms
    */
    abstract void braking(int force);
}
```

D:\bkr\Core java>javac Car.java

→ // this is a concrete sub class of car class

```
class Maruthi extends Car
{
    Maruthi(int regno)
    {
        super(regno);
    }
    void steering(int direction)
    {
        System.out.println("Maruthi uses manual steering");
        System.out.println("please drive the car");
```

```

    }
    void braking(int force)
    {
        System.out.println("Maruthi uses hydraulic breaks");
        System.out.println("apply breaks & stop the car");
    }
}
D:\bkr\Core java>javac Maruthi.java

```

→ // this is a concrete sub class of car class

```

class Santro extends Car
{
    Santro(int regno)
    {
        super(regno);
    }
    void steering(int direction)
    {
        System.out.println("Santro uses power steering");
        System.out.println("Start it");
    }
    void braking(int force)
    {
        System.out.println("Santro uses gas breaks");
        System.out.println(" stop it");
    }
}
D:\bkr\Core java>javac Santro.java

```

→ // using the car

```

class UseCar
{
    public static void main(String[] args)
    {
        // create a maruthi & Santro obj
        Maruthi m=new Maruthi(6666);
        Santro s=new Santro(2222);
        // create a reference of car class

```

```
Car ref;
// use ref to refer to maruthi
ref=m;
ref.fillTank();
ref.steering(2);
ref.braking(200);
// use ref to refer to Santro
ref=s;
ref.fillTank();
ref.steering(4);
ref.braking(400);
}
}
```

D:\bkr\Core java>javac UseCar.java

D:\bkr\Core java>java UseCar

Use car Key
And fill the tank
Maruthi uses manual steering
please drive the car
Maruthi uses hydraulic breaks
apply breaks & stop the car
Use car Key
And fill the tank
Santro uses power steering
Start it
Santro uses gas breaks
stop it

Reference statement is used to element the representation of statements.

- 1) An abstract class is class with zero or more abstract methods
- 2) An abstract class contains instance variables & concrete methods in addition to abstract methods.
- 3) It is not possible to create objects to abstract class.
- 4) But we can create a reference of abstract class type.
- 5) All the abstract methods of the abstract class should be implemented in its sub classes.
- 6) If any method is not implemented, then that sub class should be declared as ‘abstract’.
- 7) Abstract class reference can be used to refer to the objects of its sub classes.
- 8) Abstract class references cannot refer to the individual methods of sub classes.

9) A class can not be both ‘abstract’ & ‘final’.

Ex: - final abstract class A // invalid

→ How can enforce discipline in the programmers to implement only your class features?

A) By writing an abstract class or an interface I can enforce.

Typecasting

Data type: - A data type represents types of data stored into a variable. These are 2 types.

1) Primitive data type: - (fundamental data types)

These are represents single values; methods are not available to handle them.

Ex: - char, byte, short, int, long, float, double, Boolean

2) Advanced data types: - (Referenced data types)

These are represents several value methods also available to handle them.

Ex: - int array, any classes (ex: employee, string.....)

→ What is difference between primitive & advanced data types?

A) 1) Casting can be used to convert one primitive data type in to another primitive data type.

2) Casting can be used to convert one advanced data type into another advanced data type.

3) Casting can not be used to convert a primitive data type into an advanced data type and vice versa (means reverse also).

For this purpose we can use wrapper classes.

Casting primitive data types: -

a) Widening: - Casting a lower data type into a higher data type is called widening.

Ex: - char, byte, short, int, long, float, double

Lower ←-----→higher

- i) char ch='A';
int n=(int)ch;
- ii) int num=15;
float f=(float)num;

In widening no digits or precision are lost.

b) Narrowing: - Converting a higher data type into lower type is called narrowing.

Ex: - i) int n=66;

- Char ch=(char)n;
- ii) double d=12.1234;
int n=(int)d;

In narrowing precision or digits are lost. This is called explicit casting.

Casting advanced data types:

We can cast advanced data types provide there is relationship between the classes by the way of inheritance.

→//Casting advanced data types:

```

class One
{
    void show1()
    {
        System.out.println("Super class");
    }
}
class Two extends One
{
    void show2()
    {
        System.out.println("Sub class");
    }
}
class Cast
{
    public static void main(String args[])
    {
        // super class reference to refer to super class object
        One o;
        o=new One();
        o.show1();
    }
}

```

D:\bkr\Core java>javac Cast.java

D:\bkr\Core java>java Cast

Super class

In super reference is used to refer to super class object. The programmer can refer to only super class members but not the sub class members

→

```

class Cast
{
    public static void main(String args[])
    {
        // sub class reference to refer to sub class object
        One o=new Two();
        Two t=(Two)o;
        t.show1();
        t.show2();
    }
}

```

D:\bkr\Core java>javac Cast.java

D:\bkr\Core java>java Cast

Super class

Sub class

In sub class reference is used subclass object; the programmer can access all the members of both the super class& sub class.

→

```
class Cast
{
    public static void main(String args[])
    {
        // super class reference to refer to sub class object
        One o;
        o=new Two();
        o.show1();
    }
}
```

D:\bkr\Core java>javac Cast.java

D:\bkr\Core java>java Cast

Super class

What is generalization & specialization?

A) Moving back from sub class to super class is called generalization. Casting a sub class type into a super class is called a up casting or widening.

Coming down from super class to sub class is called specialization. Casting a super class type into sub class type is called narrowing or down casting.

In widening we can access super class methods. In widening we can access sub class methods. In widening, we can not access sub class methods unless the override super class methods.

→

```
class Cast
{
    public static void main(String args[])
    {
        // sub class reference to refer to super class object
        One o=new Two();
        Two t=(Two)o;
        t.show1();
        t.show2();
    }
}
```

D:\bkr\Core java>javac Cast.java

D:\bkr\Core java>java Cast

Super class

Sub class

Narrowing using super class object can not access either super class methods or sub class methods.

Narrowing using sub class object will make the programmer to access are the members of super class as well as sub class.

→ **What is widening & narrowing?**

A) Widening: - Casting a lower data type into a higher data type is called widening. In widening no digits or precision are lost. Casting a sub class type into a super class is called an up casting or widening. In widening we can access super class methods. In widening we can access sub class methods. In widening us can not access sub class methods unless the override super class methods.

Narrowing: - Converting a higher data type into lower type is called narrowing. In narrowing precision or digits are lost. This is called explicit casting. Casting a super class type into sub class type is called narrowing or down casting. Narrowing using super class object can not access either super class methods or sub class methods. Narrowing using sub class object will make the programmer to access are the members of super class as well as sub class.

Final keyword in java

It is used to make a variable as a constant, Restrict method overriding, Restrict inheritance. It is used at variable level, method level and class level. In java language final keyword can be used in following way.

- Final at variable level
- Final at method level
- Final at class level



Final Keyword

- * **Restrict Changing value of variable**
- * **Restrict method Overriding**
- * **Restrict Inheritance**

Tutorial4us.com

Final at variable level

Final keyword is used to make a variable as a constant. This is similar to const in other language. A variable declared with the final keyword cannot be modified by the program after initialization. This is useful to universal constants, such as "PI".

Final Keyword in java Example

```
public class Circle
{
    public static final double PI=3.14159;

    public static void main(String[] args)
    {
        System.out.println(PI);
    }
}
```

Final at method level

It makes a method final, meaning that sub classes can not override this method. The compiler checks and gives an error if you try to override the method.

When we want to restrict overriding, then make a method as a final.

Example

```
public class A
{
    public void fun1()
    {
        .....
    }

    public final void fun2()
    {
        .....
    }
}
```

```
class B extends A
{
    public void fun1()
    {
        .....
    }
    public void fun2()
    {
        // it gives an error because we can not override final method
    }
}
```

Example of final keyword at method level

Example

```
class Employee
{
    final void disp()
    {
        System.out.println("Hello Good Morning");
    }
}
class Developer extends Employee
{
    void disp()
    {
        System.out.println("How are you ?");
    }
}
class FinalDemo
{
    public static void main(String args[])
    {
        Developer obj=new Developer();
        obj.disp();
    }
}
```

```
}
```

Output

It gives an error

Final at class level

It makes a class final, meaning that the class can not be inheriting by other classes. When we want to restrict inheritance then make class as a final.

Example

```
public final class A
{
    .....
    .....
}

public class B extends A
{
    // it gives an error, because we can not inherit final class
}
```

Example of final keyword at class level

Example

```
final class Employee
{
    int salary=10000;
}

class Developer extends Employee
{
    void show()
    {
        System.out.println("Hello Good Morning");
    }
}
```

```
}
```

```
class FinalDemo
```

```
{
```

```
public static void main(String args[])
```

```
{
```

```
Developer obj=new Developer();
```

```
Developer obj=new Developer();
```

```
obj.show();
```

```
}
```

```
}
```

Output

Output:

It gives an error

This keyword in java

this is a reference variable that refers to the current object. It is a keyword in java language represents current class object

There can be a lot of usage of java this keyword. In java, this is a reference variable that refers to the current object

Usage of this keyword

It can be used to refer current class instance variable.

this() can be used to invoke current class constructor.

It can be used to invoke current class method ()implicitly()

It can be passed as an argument in the method call.

It can be passed as argument in the constructor call.

It can also be used to return the current class instance.

this keyword can be used to refer current class instance variable.

this() can be used to invoke current class constructor.

this keyword can be used to invoke current class method (implicitly)

this can be passed as an argument in the method call.

this can be passed as argument in the constructor call.

this keyword can also be used to return the current class instance.

Why use this keyword in java ?

The main purpose of using this keyword is to differentiate the formal parameter and data members of class, whenever the formal parameter and data members of the class are similar then jvm get ambiguity)no clarity between formal parameter and member of the class(To differentiate between formal parameter and data member of the class, the data member of the class must be preceded by "this".

"this" keyword can be used in two ways.

this .)this dot(

this)()this off(

this .)this dot(

which can be used to differentiate variable of class and formal parameters of method or constructor.

"this" keyword are used for two purpose, they are

It always points to current class object.

Whenever the formal parameter and data member of the class are similar and JVM gets an ambiguity)no clarity between formal parameter and data members of the class(.

To differentiate between formal parameter and data member of the class, the data members of the class must be preceded by "this".

Syntax

this.data member of current class.

Note: If any variable is preceded by "this" JVM treated that variable as class variable.

Example without using this keyword

```
class Employee
{
    int id;
    String name;

    Employee)int id,String name(
    {
        id = id;
        name = name;
    }
    void show)(
    {
        System.out.println)id+" "+name();
    }
    public static void main)String args][(
    {
        Employee e1 = new Employee)111,"Harry"();
        Employee e2 = new Employee)112,"Jacy"();
```

```
e1.show)();
e2.show)();
}
}
Output
Output:
0 null
0 null
```

In the above example, parameter)formal arguments(and instance variables are same that is why we are using "this" keyword to distinguish between local variable and instance variable.

Example of this keyword in java

```
class Employee
{
    int id;
    String name;

    Employee)int id,String name(
    {
        this.id = id;
        this.name = name;
    }
    void show)(
    {
        System.out.println)id+" "+name();
    }
    public static void main)String args][[
    {
        Employee e1 = new Employee)111,"Harry"();
        Employee e2 = new Employee)112,"Jacy"();
        e1.show)();
        e2.show)();
    }
}
Output
111 Harry
112 Jacy
```

Note 1: The scope of "this" keyword is within the class.

Note 2: The main purpose of using "this" keyword in real life application is to differentiate variable of class or formal parameters of methods or constructor)it is highly recommended

to use the same variable name either in a class or method and constructor while working with similar objects(.

Difference between this and super keyword

Super keyword is always pointing to base class)scope outside the class(features and "this"keyword is always pointing to current class)scope is within the class(features.

Example when no need of this keyword

```
class Employee
{
    int id;
    String name;

    Employee)int i,String n(
    {
        id = i;
        name = n;
    }
    void show)(
    {
        System.out.println)id+" "+name();
    }
    public static void main)String args][(
    {
        Employee e1 = new Employee)111,"Harry"();
        Employee e2 = new Employee)112,"Jacy"();
        e1.show();
        e2.show();
    }
}
```

Output

111 Harry

112 Jacy

In the above example, no need of use this keyword because parameter)formal arguments(and instance variables are different. This keyword is only use when parameter)formal arguments(and instance variables are same.

this)(

which can be used to call one constructor within the another constructor without creation of objects multiple time for the same class.

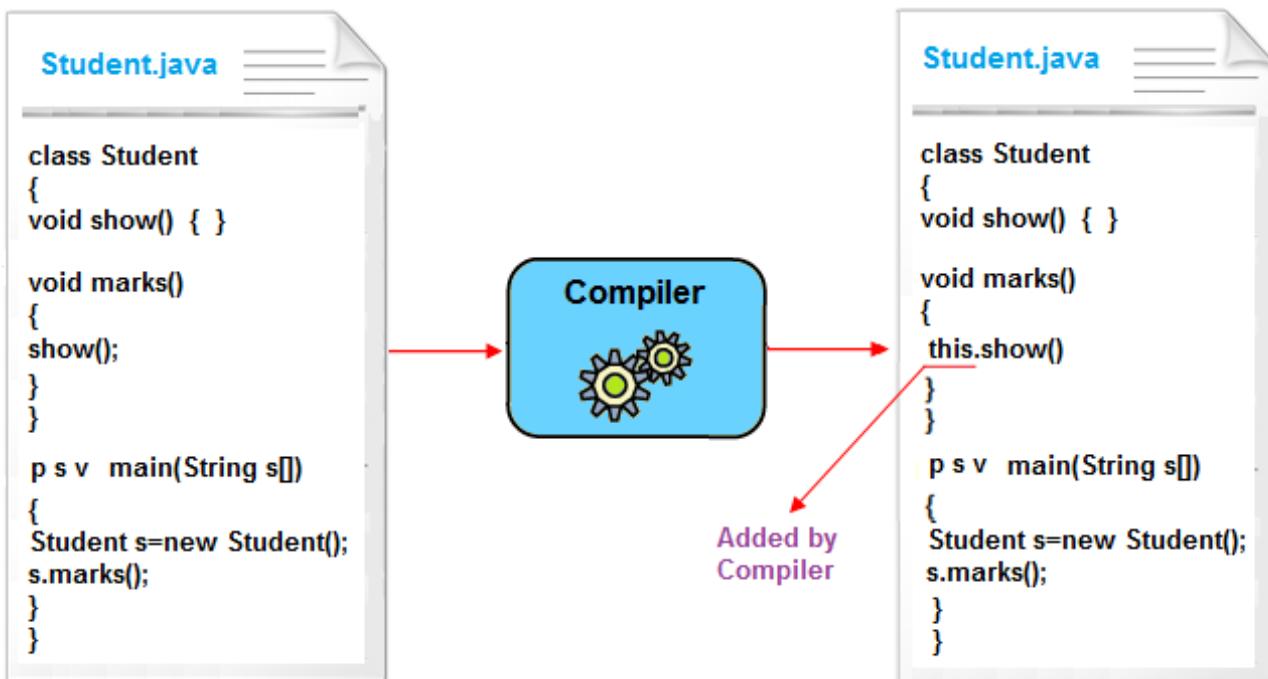
Syntax

this)(); // call no parametrized or default constructor

this)value1,value2,.....(//call parametrize constructor

this keyword used to invoke current class method)implicitly(

By using this keyword you can invoke the method of the current class. If you do not use the this keyword, compiler automatically adds this keyword at time of invoking of the method.



Example of this keyword

```

class Student
{
void show)(
{
System.out.println)"You got A+"(;
}
void marks)(
{
this.show()); //no need to use this here because compiler does it.
}
void display)(
{
show()); //compiler act marks)( as this.marks)(
}
public static void main)String args][(
{
Student s = new Student)();
s.display());
}
}
  
```

Syntax

You got A+

Rules to use this)(

this)(always should be the first statement of the constructor. One constructor can call only other single constructor at a time by using this)(.

```

class Sum
{
    → Sum()
    {
        System.out.println("No parametrized Constructor");
    }

    → Sum( int a)
    {
        this();
        System.out.println("One parametrized Constructor");
    }

    Sum(int a, int b)
    {
        this(10);
        System.out.println("Two parametrized Constructor");
    }
}

class thisDemo
{
    public static void main(String args[])
    {
        Sum obj=new Sum(10, 20);
    }
}

```

Tutorial4us.com

```

graph TD
    A[Call no parametrized Constructor] --> B[Sum()]
    C[Call one parametrized Constructor] --> D[Sum(int a)]

```

Super keyword in java

Super keyword in java is a reference variable that is used to refer parent class object. Super is an implicit keyword create by JVM and supply each and every java program for performing important role in three places.

At variable level

At method level

At constructor level

Need of super keyword:

Whenever the derived class is inherits the base class features, there is a possibility that base class features are similar to derived class features and JVM gets an ambiguity. In order to

differentiate between base class features and derived class features must be preceded by super keyword.

Syntax

super.baseclass features.

Super at variable level:

Whenever the derived class inherit base class data members there is a possibility that base class data member are similar to derived class data member and JVM gets an ambiguity. In order to differentiate between the data member of base class and derived class, in the context of derived class the base class data members must be preceded by super keyword.

Syntax

super.baseclass datamember name

if we are not writing super keyword before the base class data member name than it will be referred as current class data member name and base class data member are hidden in the context of derived class.

Program without using super keyword

Example

```
class Employee
{
float salary=10000;
}
class HR extends Employee
{
float salary=20000;
void display()
{
System.out.println("Salary: "+salary);//print current class salary
}
}
```

class Supervariable

```
{
public static void main(String[] args)
{
HR obj=new HR();
obj.display();
}}
```

Output

Salary: 20000.0

In the above program in Employee and HR class salary is common properties of both class the instance of current or derived class is referred by instance by default but here we want to refer base class instance variable that is why we use super keyword to distinguish between parent or base class instance variable and current or derived class instance variable.

Program using super keyword at variable level

Example

class Employee

{

float salary=10000;

}

class HR extends Employee

{

float salary=20000;

void display()

{

System.out.println("Salary: "+super.salary());//print base class salary

}

}

class Supervariable

{

public static void main(String[] args)

{

HR obj=new HR();

obj.display();

}

}

Output

Salary: 10000.0

Super at method level

The super keyword can also be used to invoke or call parent class method. It should be used in case of method overriding. In other word super keyword use when base class method name and derived class method name have same name.

Example of super keyword at method level

Example

class Student

{

void message()

{

System.out.println("Good Morning Sir");

```
}

}

class Faculty extends Student
{
void message)(
{
System.out.println)"Good Morning Students"(;
}

void display)(
{
message)(;//will invoke or call current class message)( method
super.message)(;//will invoke or call parent class message)( method
}

public static void main)String args][(
{
Student s=new Student)();
s.display)();
}
}

Output
```

Good Morning Students

Good Morning Sir

In the above example Student and Faculty both classes have message)(method if we call message)(method from Student class, it will call the message)(method of Student class not of Person class because priority of local is high.

In case there is no method in subclass as parent, there is no need to use super. In the example given below message)(method is invoked from Student class but Student class does not have message)(method, so you can directly call message)(method.

Program where super is not required

Example

```
class Student
{
void message)(
{
System.out.println)"Good Morning Sir"(;
}
}
```

```
class Faculty extends Student
{
void display()
{
message); //will invoke or call parent class message) method
}

public static void main(String args[])
{
Student s=new Student();
s.display();
}
}
Output
```

Good Morning Sir

1) The this keyword can be used to refer current class instance variable.

If there is ambiguity between the instance variable and parameter, this keyword resolves the problem of ambiguity.

Understanding the problem without this keyword

Let's understand the problem if we don't use this keyword by the example given below:

```
class Student10{
int id;
String name;

Student10(int id,String name){
id = id;
name = name;
}
void display(){System.out.println(id+" "+name);}

public static void main(String args[]){
Student10 s1 = new Student10(111,"Karan");
Student10 s2 = new Student10(321,"Aryan");
s1.display();
s2.display();
}
```

```
}
```

Test it Now

Output:0 null

0 null

In the above example, parameter (formal arguments) and instance variables are same that is why we are using this keyword to distinguish between local variable and instance variable.

Solution of the above problem by this keyword

```
//example of this keyword
```

```
class Student11{
```

```
    int id;
```

```
    String name;
```

```
    Student11(int id,String name){
```

```
        this.id = id;
```

```
        this.name = name;
```

```
}
```

```
    void display(){System.out.println(id+" "+name);}
```

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

```
        Student11 s1 = new Student11(111,"Karan");
```

```
        Student11 s2 = new Student11(222,"Aryan");
```

```
        s1.display();
```

```
        s2.display();
```

```
}
```

```
}
```

Test it Now

Output111 Karan

222 Aryan

If local variables(formal arguments) and instance variables are different, there is no need to use this keyword like in the following program:

Program where this keyword is not required

```
class Student12{
```

```
    int id;
```

```
    String name;
```

```
    Student12(int i,String n){
```

```
        id = i;
```

```
        name = n;
```

```
}
```

```

void display(){System.out.println(id+" "+name);}
public static void main(String args[]){
Student12 e1 = new Student12(111,"karan");
Student12 e2 = new Student12(222,"Aryan");
e1.display();
e2.display();
}
}

```

Test it Now

Output:111 Karan

222 Aryan

2) this() can be used to invoked current class constructor.

The this() constructor call can be used to invoke the current class constructor (constructor chaining). This approach is better if you have many constructors in the class and want to reuse that constructor.

//Program of this() constructor call (constructor chaining)

```

class Student13{
int id;
String name;
Student13(){System.out.println("default constructor is invoked");}

```

```

Student13(int id,String name){
this(); //it is used to invoked current class constructor.
this.id = id;
this.name = name;
}
void display(){System.out.println(id+" "+name);}

```

```

public static void main(String args[]){
Student13 e1 = new Student13(111,"karan");
Student13 e2 = new Student13(222,"Aryan");
e1.display();
e2.display();
}
}

```

Test it Now

Output:

default constructor is invoked
default constructor is invoked

111 Karan
222 Aryan

Where to use this() constructor call?

The this() constructor call should be used to reuse the constructor in the constructor. It maintains the chain between the constructors i.e. it is used for constructor chaining. Let's see the example given below that displays the actual use of this keyword.

```
class Student14{
    int id;
    String name;
    String city;

    Student14(int id, String name){
        this.id = id;
        this.name = name;
    }

    Student14(int id, String name, String city){
        this(id, name); //now no need to initialize id and name
        this.city = city;
    }

    void display(){System.out.println(id + " " + name + " " + city);}

    public static void main(String args[]){
        Student14 e1 = new Student14(111, "karan");
        Student14 e2 = new Student14(222, "Aryan", "delhi");
        e1.display();
        e2.display();
    }
}
```

Rule: Call to this() must be the first statement in constructor.

```
class Student15{
    int id;
    String name;
    Student15(){System.out.println("default constructor is invoked");}

    Student15(int id, String name){
        id = id;
        name = name;
        this(); //must be the first statement
    }
}
```

```

void display()
{
System.out.println(id+" "+name);
}
public static void main(String args[])
{
Student15 e1 = new Student15(111,"karan");
Student15 e2 = new Student15(222,"Aryan");
e1.display();
e2.display();
}
}

```

3) The this keyword can be used to invoke current class method (implicitly).

You may invoke the method of the current class by using the this keyword. If you don't use the this keyword, compiler automatically adds this keyword while invoking the method. Let's see the example

1. **class S{**
void m(){
System.out.println("method is invoked");
}
void n(){
this.m();//no need because compiler does it for you.
}
void p(){
n();//complier will add this to invoke n() method as this.n()
}
public static void main(String args[]){
S s1 = new S();
s1.p();
}
}

4) this keyword can be passed as an argument in the method.

The this keyword can also be passed as an argument in the method. It is mainly used in the event handling. Let's see the example:

```
class S2{
```

```
void m(S2 obj){  
    System.out.println("method is invoked");  
}  
void p(){  
    m(this);  
}  
  
public static void main(String args[]){  
    S2 s1 = new S2();  
    s1.p();  
}
```

Application of this that can be passed as an argument:

In event handling (or) in a situation where we have to provide reference of a class to another one.

5) The this keyword can be passed as argument in the constructor call.

We can pass the this keyword in the constructor also. It is useful if we have to use one object in multiple classes. Let's see the example:

```
class B{  
    A4 obj;  
    B(A4 obj){  
        this.obj=obj;  
    }  
    void display(){  
        System.out.println(obj.data);//using data member of A4 class  
    }  
}  
  
class A4{  
    int data=10;  
    A4(){  
        B b=new B(this);  
        b.display();  
    }  
    public static void main(String args[]){  
        A4 a=new A4();  
    }  
}
```

6) The this keyword can be used to return current class instance.

We can return the this keyword as an statement from the method. In such case, return type of the method must be the class type (non-primitive). Let's see the example:

Syntax of this that can be returned as a statement

1. `return_type method_name(){`
2. `return this;`
3. `}`

Example of this keyword that you return as a statement from the method

```
class A{  
A getA(){  
    return this;  
}  
void msg(){System.out.println("Hello java");}  
}
```

```
class Test1{  
public static void main(String args[]){  
    new A().getA().msg();  
}  
}  
Output:Hello java
```

Proving this keyword

Let's prove that this keyword refers to the current class instance variable. In this program, we are printing the reference variable and this, output of both variables are same.

```
class A5{  
void m(){  
    System.out.println(this); //prints same reference ID  
}  
  
public static void main(String args[]){  
    A5 obj=new A5();  
    System.out.println(obj); //prints the reference ID
```

```
obj.m();  
}  
}  
Output:A5@22b3ea59  
A5@22b3ea59
```

Super at constructor level

The super keyword can also be used to invoke or call the parent class constructor. Constructor are calling from bottom to top and executing from top to bottom.

To establish the connection between base class constructor and derived class constructors JVM provides two implicit methods they are:

- Super()
- Super(...)

Super()

Super() It is used for calling super class default constructor from the context of derived class constructors.

Super keyword used to call base class constructor

Syntax

```
class Employee  
{  
Employee()  
{  
System.out.println("Employee class Constructor");  
}  
}  
class HR extends Employee  
{  
HR()  
{
```

```

super(); //will invoke or call parent class constructor
System.out.println("HR class Constructor");

}
}

class Supercons
{
public static void main(String[] args)
{
HR obj=new HR();
}
}

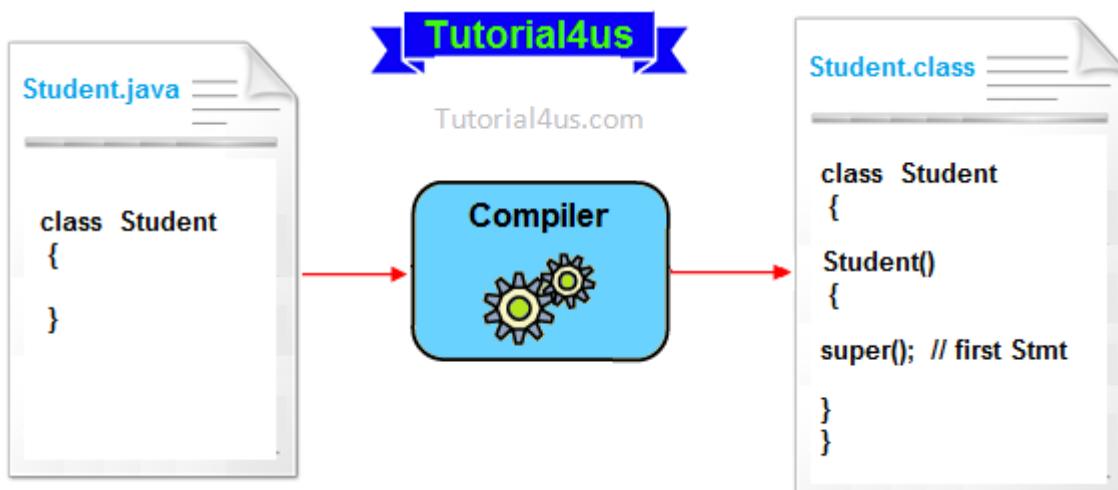
```

Output

Employee class Constructor

HR class Constructor

Note: super() is added in each class constructor automatically by compiler.



In constructor, default constructor is provided by compiler automatically but it also adds **super()** before the first statement of constructor. If you are creating your own constructor and you do not have either this() or super() as the first statement, compiler will provide super() as the first statement of the constructor.

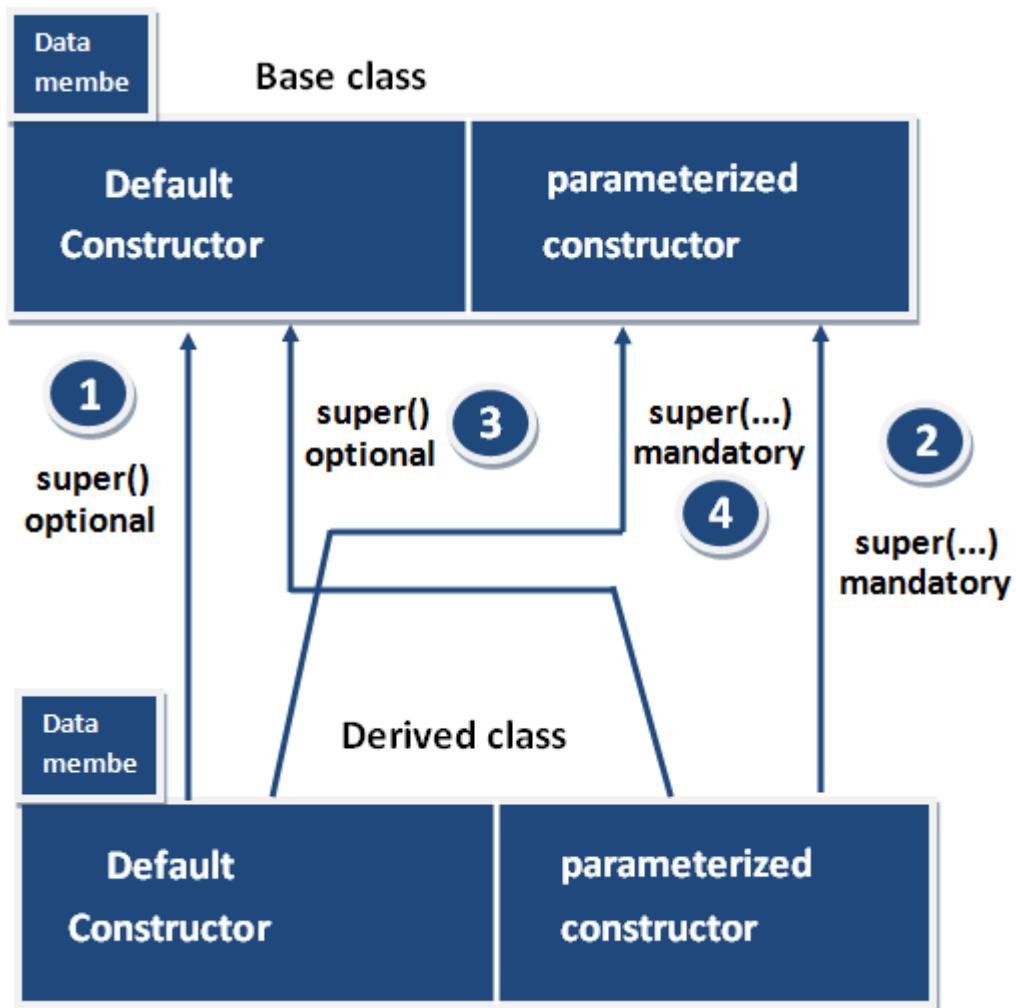
Super(...)

Super(...) It is used for calling super class parameterize constructor from the context of derived class constructor.

Important rules

Whenever we are using either super() or super(...) in the derived class constructors the **super** always must be as a first executable statement in the body of derived class constructor otherwise we get a compile time error.

The following diagram use possibilities of using super() and super(.....)

**Rule 1 and Rule 3**

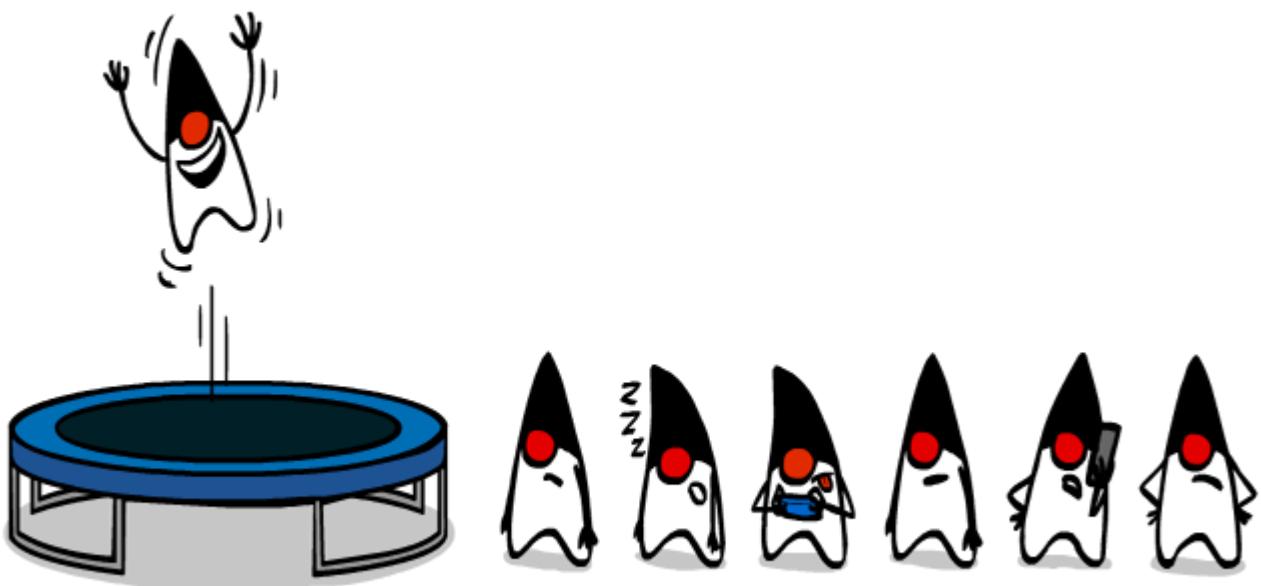
Whenever the derived class constructor want to call default constructor of base class, in the context of derived class constructors we write super(). Which is optional to write because every base class constructor contains single form of default constructor?

Rule 2 and Rule 4

Whenever the derived class constructor wants to call parameterized constructor of base class in the context of derived class constructor we must write super(...). which is mandatory to write because a base class may contain multiple forms of parameterized constructors.

Synchronized Keyword in Java

Synchronized Keyword is used for when we want to allowed only one thread at a time then use Synchronized modifier. If a method or block declared as a Synchronized then at a time only one thread is allowed to operate on the given object.



Synchronized is a Modifier which is applicable for the method or block, we can not declare class or variable with this modifier.

Advantage of Synchronized

The main advantage of Synchronized keyword is we can resolve data inconsistency problem.

Dis-Advantage of Synchronized

The main dis-advantage of Synchronized keyword is it increased the waiting time of thread and effect performance of the system, Hence if there is no specific requirement it is never recommended to use synchronized keyword.

Volatile Keyword in Java

If the variable keep on changing such type of variables we have to declare with volatile modifier. Volatile is a modifier applicable only for variables but not for method and class.

If a variable declared as volatile then for every thread a separate local copy will be created. Every intermediate modification performed by that thread will takes place in local copy instead of master copy.

Once the value got finalized just before terminating the thread the master copy value will be updated with local stable value.

Advantage of Volatile

The main advantage of Volatile keyword is we can resolve data inconsistency problems.

Dis-Advantage of Volatile

The main dis-advantage of Volatile keyword is, creating and maintaining a separate copy for every thread, increases complexity of the programming and effects performance of the system. Hence if there is no specific requirement it is never recommended to use volatile keyword, and it is almost outdated keyword.

Note: Volatile variable means its value keep on changing where as final variable means its value never changes. Hence final-Volatile combination is illegal combination for variables.

Packages: - A package represents a sub directory that contains a group of elements.

EX: - `import.java.io.*;`

Compile: - `javac -d . class name.java`

Rules to create user defined package

- package statement should be the first statement of any package program.
- Choose an appropriate class name or interface name and whose modifier must be public.
- Any package program can contain only one public class or only one public interface but it can contain any number of normal classes.
- Package program should not contain any main class (that means it should not contain any `main()`)
- modifier of constructor of the class which is present in the package must be public. (This is not applicable in case of interface because interface have no constructor.)
- The modifier of method of class or interface which is present in the package must be public (This rule is optional in case of interface because interface methods by default public)
- Every package program should be save either with public class name or public Interface name

Advantages of packages: -

- 1) Packages hide classes & interfaces. Thus they provide protection for them.
- 2) The classes of one Package are isolated from the classes of another Package. So it is possible to use same names for the classes into different packages.
- 3) Using package concept we can create our own Packages & also we can extend already available Packages.
- 4) Packages provide re usability of code.
- 5) Package is used to categorize the classes and interfaces so that they can be easily maintained
- 6) Application development time is less, because reuse the code
- 7) Application memory space is less (main memory)
- 8) Application execution time is less

- 9) Application performance is enhance (improve)
- 10) Redundancy (repetition) of code is minimized
- 11) Package provides access protection.
- 12) Package removes naming collision.

→ // creating our own Package: pack

```
package pack;
public class Addition
{
    private double d1,d2;
    public Addition(double a,double b)
    {
        d1=a;
        d2=b;
    }
    public void sum( )
    {
        System.out.println("Sum="+ (d1+d2));
    }
}
```

D:\bkr\Core java>javac -d . Addition.java

→ // using package: pack (Addition)

```
class AdditionUse
{
    public static void main(String args[])
    {
        pack.Addition obj=new pack.Addition(10,15.5);
        obj.sum( );
    }
}
```

D:\bkr\Core java>javac -d . AdditionUse.java

D:\bkr\Core java>java AdditionUse

Sum=25.5

→ // using package: pack

```
package pack;
public class Subtraction
{
    public static double sub(double a,double b)
    {
        return a-b;
    }
}
```

D:\bkr\Core java>javac -d . Subtraction.java

→ // using package: pack (Subtraction)

```
class SubtractionUse
{
    public static void main(String args[])
    {
        pack.Addition obj=new pack.Addition(13,43.5);
        obj.sum();
        double res=pack.Subtraction.sub(13,43.5);
        System.out.println("result = "+res);
    }
}
```

D:\bkr\Core java>javac -d . SubtractionUse.java

D:\bkr\Core java>java SubtractionUse

Sum=56.5

result = -30.5

(or)

```
import pack.Addition;          }
import pack.Subtraction;       } import pack.*
class SubtractionUse1
{
    public static void main(String args[])
    {
        Addition obj=new Addition(13,43.5);
        obj.sum();
```

```

        double res=Subtraction.sub(13,43.5);
        System.out.println("result = "+res);
    }
}

```

D:\bkr\Core java>set CLASSPATH=pskr;.;%CLASSPATH%

D:\bkr\Core java>javac -d . SubtractionUse1.java

D:\bkr\Core java>java SubtractionUse1

Sum=56.5

result = -30.5

Import the package name eliminates the before adding class names.

Error: - bad class file : .\Addition.java

We can handle these errors must follow some steps

Step 1: - Java compile searches from the package in .jar(java archair) file format in the following directory

C:\program files\java\jre1.5.0\lib\ext

→ What is java archair file?

A) Jar file is a compressed version of several. Class or. Java files. A jar file created using jar utility provided by sun micro systems.

Step 2:- Java compiler tapes the present directory as the package and searches for .class files there.

Step 3:- Java compiler searches for the package in current directory.

Compile:- pack directory cut & paste in to pskr folder

D:\ bkr> cd pskr.

D:\ bkr\ pskr>javac SubtractionUse.java

Step 4:- Finally java compiler searches for the package in the class path.

Class path: -

Class path is an operating system variable. That stores active directory path.

→ To sea the class path

D:\ bkr> echo %classpath%

Display content of class path

.\;C:\Program Files\Java\jdk1.5.0_07\jre\lib\rt.jar;d:\bkr;pskr;

→ To set the class path to the package directory.

```
D:\bkr> set classpath=d:\bkr\pskr;.;%classpath%
```

Compile the above program & run it.

→ // adding interface to the package: pack

```
package pack;
public interface MyDate // just display date & time
{
    void showDate(); // public abstract
}
D:\bkr\Core java>javac -d . MyDate.java
```

→ // this is implementation class of MyDate

```
package pack;
import pack.MyDate;
import java.util.Date;
public class MyDateImpl implements MyDate
{
    public void showDate()
    {
        Date d=new Date();
        System.out.println(d);
    }
}
D:\bkr\Core java>javac -d . MyDateImpl.java
```

→ // using the implementation class object

```
import pack.MyDateImpl;
class MyDateImplUse
{
    public static void main(String[] args)
    {
        MyDateImpl obj=new MyDateImpl();
        obj.showDate();
    }
}
```

```
D:\bkr\Core java>javac -d . MyDateImplUse.java
```

```
D:\bkr\Core java>java MyDateImplUse
```

Mon Sep 03 13:23:55 IST 2007

→ // creating sub package

```
package inet;
public class Sample
{
    public void show( )
    {
        System.out.println("Hai how are u");
    }
}
```

D:\bkr\Core java>javac -d . Sample.java

→ // using the sub package

```
import inet.Sample;
public class SampleUse
{
    public static void main(String args[])
    {
        Sample s=new Sample();
        s.show();
    }
}
```

D:\bkr\Core java>javac SampleUse.java

D:\bkr\Core java>java SampleUse

Hai how are u

Suppose inet directory cut & paste the directory in the drive pskr directory and set the path
set classpath d:\pskr;.;%classpath%

Static import

static import is a feature that expands the capabilities of **import** keyword. It is used to import **static** member of a class. We all know that static member are referred in association with its class name outside the class. Using **static import**, it is possible to refer to the static member directly without its class name. There are two general form of static import statement.

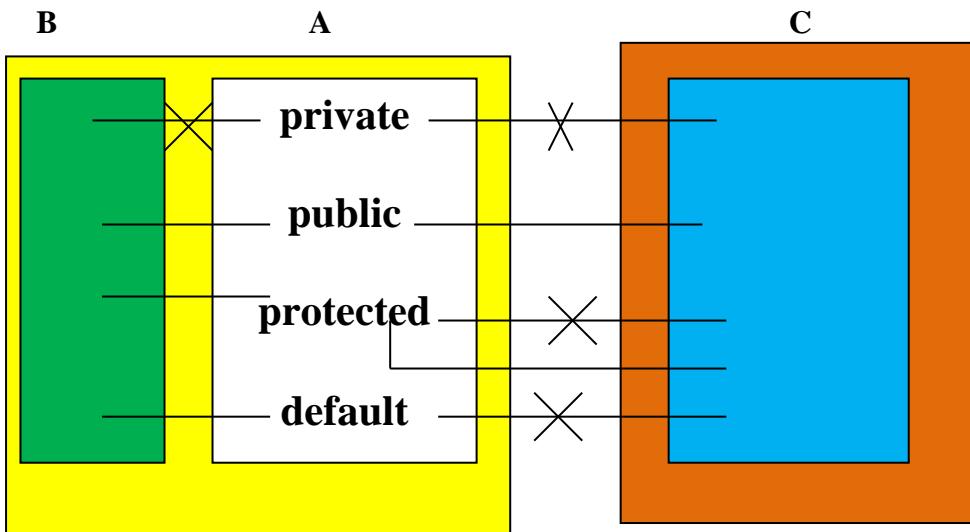
- The first form of **static import** statement, import only a single static member of a class

Syntax

```
import static package.class-name.static-member-name;
```

Access specifiers (or) access modifiers: - An access specifier is a key word that specifies how to read or access the members of a class or the class itself. There are 4 types of access specifiers. They are
1) private 2) public 3) protected 4) default

- 1) private members of a class are not accessible in other classes of same package or another package.
- 2) public members of a class are available to other classes of same package or another package. Its scope is global scope.
- 3) protected members are accessible to the other classes of the same package but not in another package.



- 4) default members are accessible in other classes of the same package. They are not accessible in another package. Its scope is package scope.

Note: - protected members of a class are always available to its sub classes in the same package or another package also. The scope of protected of some times global scope & some time package scope.
→ // access specifiers

```
package same;
public class A
{
    private int a=1;
    public int b=2;
    protected int c=3;
    int d=4;
}
```

D:\bkr\Core java>javac -d . A.java

→ // access specifier with same package

```
package same;
import same.A;
public class B
{
    public static void main(String args[])
    {
        A obj=new A();
        System.out.println(obj.a);
        System.out.println(obj.b);
        System.out.println(obj.c);
        System.out.println(obj.d);
    }
}
```

D:\bkr\Core java>javac -d . B.java

B.java:9: a has private access in same.A
 System.out.println(obj.a);
 ^

1 error

→ // access specifier with another package

```
package another;
import same.A;
public class C
{
    public static void main(String args[])
    {
        A obj=new A();
        System.out.println(obj.a);
        System.out.println(obj.b);
        System.out.println(obj.c);
        System.out.println(obj.d);
    }
}
```

D:\bkr\Core java>javac -d . C.java

C.java:9: a has private access in same.A
 System.out.println(obj.a);
 ^ C.java:11: c has protected access in same.A
 System.out.println(obj.c);
 ^

C.java:12: d is not public in same.A; cannot be accessed from outside package

System.out.println(obj.d);
 ^

3 errors

→ // access specifier with another package

```
package another;
import same.A;
public class C extends A
{
    public static void main(String args[])
    {
        C obj=new C();
        System.out.println(obj.a);
        System.out.println(obj.b);
        System.out.println(obj.c);
        System.out.println(obj.d);
    }
}
```

D:\bkr\Core java>javac -d . C.java

C.java:9: a has private access in same.A

```
    System.out.println(obj.a);  
          ^
```

C.java:12: d is not public in same.A; cannot be accessed from outside package

```
    System.out.println(obj.d);  
          ^
```

2 errors

API document: - Application programming interface.

Create API document: - D:\bkr>javadoc *.java (press enter)

Open internet explorer & select one choose (dos) file & press enter.

Strings: -

String class is encapsulated under java.lang package. In java, every string that you create is actually an object of type String.

strings represent a group of characters. In C or C++ strings are character arrays. \ 0 is null character. It is end of string.

In java strings are not character arrays. Strings are string class object in object. A string is an object of string class. Any string is an object of string class in java.

Creating strings: -

- 1) We can declare a string type variable & store directly in to it.

```
String str="hello";
```

- 2) We can create a string class object using new operator and pass the string to it.

```
String s1 =new String ("Hai");
```

- 3) We can create a string by converting a character array in to a string object.

```
Char arr[ ] = {'a','b','c','d','e','f'};
```

```
String s2 = new String (arr);
```

```
String s2=new String (arr,1,3);
```

String methods: -**Java.lang.string:** -

- 1) **String concat (String str):** - Concat means method. A method can for form a task .Concatenates the caning string with str.

Note: - +will also do the same

Concat is a method it joins two strings. When you call concat pass a string object to it.

Ex: - String s1 = "Hydera";

```
String s2 = "bad";
```

```
String x = s1.concat (s2);
```

Display Hyderabad.

+ is called concatenation operator.

Concat: - joining string at the end another string.

- 2) **int length ():**- returns the length of a string.

Ex:- String s1 = "Hydera";

```
Int n = s1.length();
```

- 3) **char charAt(int index):**

Returns the character at the specified location (from 0).

- 4) **int compareTo (string Str):** - Returns a negative value, if the calling Sting comes before str in dictionary order, a +ve value, if the String comes after str, or 0, if the String are equal.

EX: -

```
int n=s1.compareTo(s2);
```

```
if s1==s2,      n=0
```

```
if s1>s2,      n>0
```

```
if s1<s2,      n<0
```

```
s1="boy", s2="box";
```

It is a +ve number. X is first, first is less value.

5) **boolean equals(String str):** - Returns true if the calling string equals str. Equals is case sensitive method.

6) **boolean equalsIgnoreCase(String str):** - Same as above. This is case insensitive method.

7) **boolean startsWith(String prefix):** - Returns true if the calling string starts with prefix. Prefix means the string starting with beginning.

8) **boolean endsWith(String suffix):** - Returns true if the invoking string ends with suffix. Suffix means the string ending with beginning.

Note: - Above 2 methods use case sensitive comparison.

9) **int indexOf(String str):** - Returns the first occurrence of str in the string.

10) **int lastIndexOf(String str):** - Returns the last occurrence of str in the string.

Note: - Both the above methods return -ve value, if str not found in the calling string. Counting starts from 0.

Ex: - String str="This is a book";

2 5

int n=str.indexOf("is"); // It gives 2 only because first position.

int n=str.lastIndexOf("is"); // it gives 5 only.

11) **String replace(char oldChar,char newChar):** - Returns a new string. That is obtained by replacing characters old char in the string with new char.

12) **String substring(int beginIndex):** - Return a new string consisting of all characters from begin index until the end of the string.

13) **String substring(int beginIndex,int endIndex):** - Returns a new string consisting of all characters from begin index until end index(exclusive). Last character is exclusive.

14) **String toLowerCase():** - Converts all characters into lower case.

15) **String toUpperCase():** - Converts all characters into upper case.

16) **String trim():** - Eliminates all leading & trailing spaces.

Trim means deleting or cutting. Trim method doesn't remove in the middle spaces. It's only remove first spaces & last spaces. Last space is removing, it is trail method.

→// understanding the strings

```
class StringsTest
{
    public static void main(String[] args)
    {
        // create 3 strings
        String s1="This is java";
        String s2=new String(" I like it");
        char ch[]={‘P’,‘o’,‘t’,‘h’,‘u’,‘r’,‘a’,‘i’};
        String s3= new String(ch);
        // display the strings
        System.out.println("S1 = "+s1);
        System.out.println("S2 = "+s2);
        System.out.println("S3 = "+s3);
        // find no . of characters in s1
```

```

System.out.println("length of S1 = "+s1.length());
// join 2 strings
System.out.println("S1 joined with s2 = "+s1.concat(s2));
// join 3 strings
System.out.println(s1+" at "+s3);
// check the starting of s1
boolean x=s1.startsWith ("This");
if(x==true)
    System.out.println("s1 starts with This ");
else
    System.out.println("S1 does not start with This ");
/* extract substring from s1 & s3
String p=s2.subString(0,6);
String q=s3.subString(0);
System.out.println(p+q); */
//change the case of s3
System.out.println("Upper case of s3 = "+s3.toUpperCase( ));
System.out.println("Lower case of S3 = "+s3.toLowerCase( ));
}
}

```

D:\bkr\Core java>javac StringsTest.java
D:\bkr\Core java>java StringsTest
S1 = This is java
S2 = I like it
S3 = WINGS
length of S1 = 12
S1 joined with s2 = This is java I like it
This is java at WINGS
s1 starts with This
Upper case of s3 = WINGS
Lower case of S3 = WINGS

→// equality of strings

```

class StringEqual
{
    public static void main(String args[])
    {
        String a="Hello";
        String b=new String("Hello");
        if(a==b)
            System.out.println("same");
        else
            System.out.println("Not same");
    }
}

```

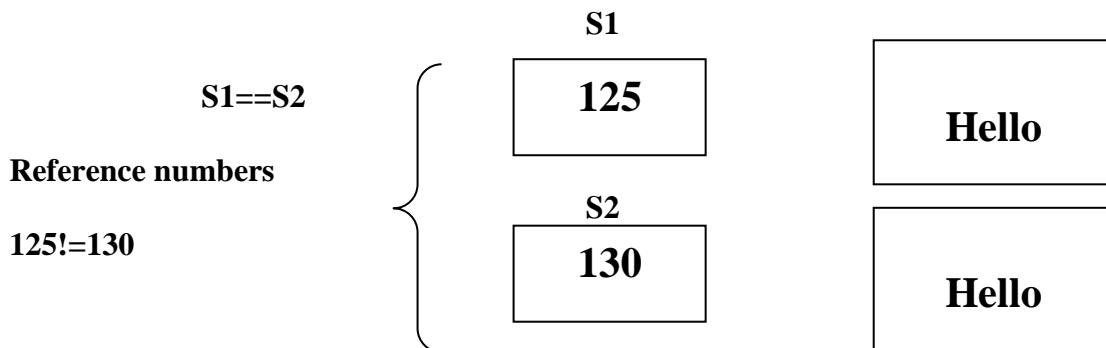
D:\bkr\Core java>javac StringEqual.java

D:\bkr\Core java>java StringEqual

Not same

→ What is hash code?

- A) It is a unique identification number that is allotted by JVM to the objects. This number is also called reference number.



So the output is not same. == is only compare reference numbers

In the above program correct format is if(s1.equals(s2))

22→ What is the difference between == to and equals method while comparing the strings?

- A) == operator compares only the reference of string objects.

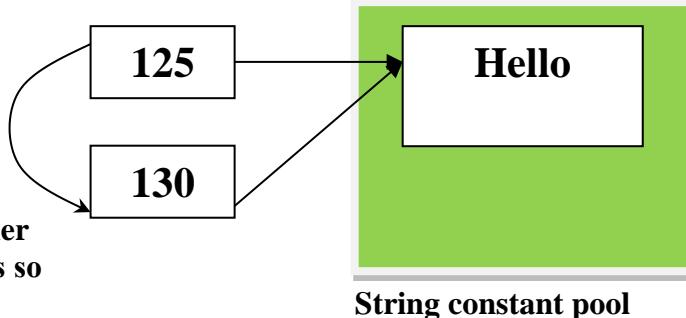
Equals method compares the contents of the string objects. Hence the equals' method gives reliable results.

23→ What is string constant pool?

- A) It is separate block of memory where the string objects are stored.

String s1="hello"
String s2="hello"

125 is copy to another
Same reference no's so
125==125



If(s1==s2)

o/p: - equal, because JVM does not waste memory.

24→ What is the difference between the following statements?

- a) String s="Hello";
- b) String s=new String("Hello");

A) When the first statement is executed JVM searches for the same object in the string constant pool. If the same object is found there then JVM will create another reference to the same object, if the same object not found JVM creates another object & stores it into string constant pool.

When the second statement is executed JVM always create a new object without searching in the string constant pool.

Types of objects: - 1) Mutable 2) Immutable.

1) Mutable: -

Mutable objects are the objects whose contents can be modified.

2) Immutable: -

Immutable objects are those objects whose contents can not be modified.

→// String class object are immutable

```
class Immutable
{
    public static void main(String[] args)
    {
        String s1="hello";
        String s2="hai";
        s1=s1+s2;
        System.out.println(s1);
    }
}
```

D:\bkr\Core java>javac Immutable.java

D:\bkr\Core java>java Immutable

Hellohai

String buffer: - String buffer is a mutable, where as string is immutable.

Creating a string buffer: -

- 2) StringBuffer sb=new StringBuffer("Hello");
- 3) StringBuffer sb=new StringBuffer

java.lang.StringBuffer: -

- 1) StringBuffer append(x): - x may be int, float, double, char, string or StringBuffer. It will be append to the calling StringBuffer.
- 2) StringBuffer insert(int offset,x): - x may be int, float, double, char, string or StringBuffer. It will be inserted into the StringBuffer at offset. Insert method is insert a value.
- 3) StringBuffer delete(int start,int end): - removes the characters from start to end.
- 4) StringBuffer reverse(): - Reverse the character sequence in the StringBuffer
- 5) String toString(): -Converting StringBuffer into a string
- 6) int length(): - Returns the length of the StringBuffer

→ What is the difference between a string & StringBuffer?

A) String objects are immutable. StringBuffer objects are mutable.

The methods which directly manipulate the data or not available in string class. Such methods are available in StringBuffer class.

→// display the full name

```
import java.io.*;
class Mutable
{
    public static void main(String[] args) throws IOException
    {
```

```

// to accept data from keyboard
BufferedReader br=new BufferedReader(new
                                InputStreamReader(System.in));
System.out.print("Enter sur name : ");
String sur=br.readLine();
System.out.print("Enter mid name : ");
String mid=br.readLine();
System.out.print("Enter last name : ");
String last=br.readLine();
// create String Buffer object
StringBuffer sb=new StringBuffer();
// append sur,last to sb
sb.append(sur);
sb.append(last);
// insert mid after sur
int n=sur.length();
sb.insert(n,mid);
// display full name
System.out.println("Full name = "+sb);
System.out.println("In reverse =" +sb.reverse());
}
}

```

D:\bkr\Core java>java Mutable

Enter sur name : Sunil
 Enter mid name : Kumar
 Enter last name : Reddy
 Full name = SunilKumarReddy

In reverse =yddeRramuKlinuS

```

import java.util.*;
public class SwapDemo
{
  public static void main(String[] args)
  {

```

```

    Scanner sc = new Scanner(System.in);
    System.out.println("Enter First String :");
    String s1 = sc.next();
    System.out.println("Enter Second String :");
    String s2 = sc.next();
    System.out.println("Before Swapping :");
    System.out.println("s1 : "+s1);
    System.out.println("s2 : "+s2);
    s1 = s1 + s2;
    s2 = s1.substring(0, s1.length()-s2.length());
    s1 = s1.substring(s2.length());
    System.out.println("After Swapping :");
    System.out.println("s1 : "+s1);
    System.out.println("s2 : "+s2);
  }
}
```

```
        }  
    }  
  
import java.util.*;  
class StringTokenizerDemo  
{  
public static void main(String args[])  
{  
String s1="palla,balakrishna,reddy,hyd,tg ";  
System.out.println(s1);  
StringTokenizer st=new StringTokenizer(s1,"");  
System.out.println(st.countTokens());  
    while(st.hasMoreElements())  
    {  
        System.out.println(st.nextToken());  
    }  
}
```

Exception handling: - An exception is a runtime error.

An error in software is called a bug. Removing errors from software is called debugging.

- 1) **Compile time errors:** - These are errors in the syntax or the grammar of the language. These errors are detected by the compiler at run time of compilation. (Java compiler can display up to 100 errors). Desk checking is the solution of compile time errors.
- 2) **Run time errors:** - These errors are the errors which occur because of insufficient compiler system. These errors are detected by JVM at the time of running the program. Run time errors are serious errors. Only the programmers are responsible for it.
- 3) **Logical errors:** - These errors represent bad logic (or) these errors are not detected by the compiler or JVM. Logical errors are detected by comparing the program output with manually calculated result.

→ **What is checked exception?**

A) The exception will occur during compilation & are executed by the java compiler are called checked exception. The exception which are detected at runtime by the JVM are called unchecked exceptions or run time errors.

Exception: - Any abnormal event in the program is called an exception. All exception are represented by classes in java

→ // Exception example

```
class ExceptionDemo
{
    public static void main(String args[])
    {
        System.out.println("open files");
        int n=args.length;
        System.out.println("n="+n);
        int a=45/n;
        System.out.println("a="+a);
        System.out.println("close files");
    }
}
```

D:\bkr\Core java>javac ExceptionDemo.java

```
D:\bkr\Core java>java ExceptionDemo Sunil Kumar Reddy
open files
n=3
a=15
close files
```

→ All exceptions are subclasses of which class?

A) Exception class.

→ What is throwable?

A) Throwable is a class that represents errors & exceptions in java

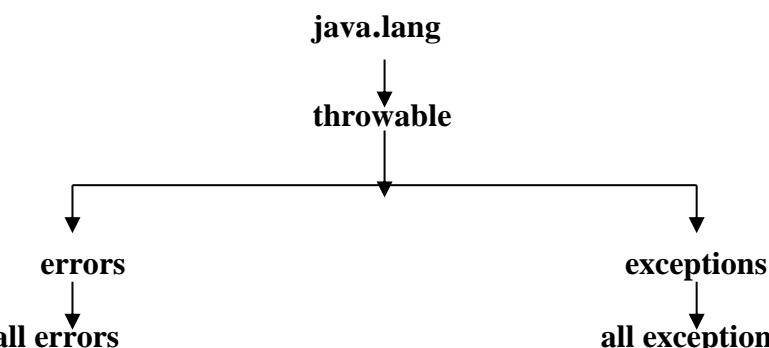
→ What is Exception?

A) Any abnormal event in the program is called an exception.

→ What is the difference between an error & exception? :-

An exception is an error that can be handled. An error can not be handled

When there is an exception JVM displays exception name & its description & then abnormally terminates the program. Because of these the files are data bases are not closed & hence user's entire data will be lost. This is effect of exception.



→ In case of an exception the programmer to do the following tasks.

1) The programmer should write all the statements, where there an exception, inside a try blocks.

```
try {
  Statements;
}
```

When there is an exception in try block JVM will not terminate the program. It will store exception details, in an exception stack & then jumps into catch block.

```
try
{
  Statements;
}
```

Exception
details

```
Catch(ExceptionClass obj)
{
    Statements;
}
                                         Exception block
```

2) In catch block the programmer has to display exception details & any other messages to the user.

3) The programmer should close the files & data bases In finally block.

Note: - finally blocks is always executed whether there is exception or not.

```
finally {
statements;
}
```

→ // Exception example

```
class ExceptionExample
{
    public static void main(String args[])
    {
        try
        {
            System.out.println("open files");
            int n=args.length;
            System.out.println("n="+n);
            int a=45/n;
            System.out.println("a="+a);
            int b[]={10,19,12,13};
            b[50]=100;
        }
        catch(ArithmaticException ae)
        {
            System.out.println("ae");
            System.out.println("plz type data while running the data");
        }
        catch(ArrayIndexOutOfBoundsException aie)
        {
            System.out.println("aie");
            System.out.println("please see that array index is within the range");
        }
    finally
```

```
        {
            System.out.println("close files");
        }
    }
}
```

D:\bkr\Core java>javac ExceptionExample.java

D:\bkr\Core java>java ExceptionExample

open files

n=0

ae

plz type data while running the data

close files

D:\bkr\Core java>java ExceptionExample Swarna Latha Reddy

open files

n=3

a=15

aie

plese see that array index is within the range

close files

- Even though multi exceptions are found in the program, only one exception is raised at a time.
- We can handle multiple exceptions by writing multiple catch blocks.
- A single try block can be followed by several catch blocks.
- Catch block does not always exit without a try, but a try block exit without a catch block.
- Finally block is always executed whether there is exception or not.

→ // not handling the exception

```
import java.io.*;
class Sample
{
    void accept( )throws IOException
    {
        BufferedReader br=new BufferedReader(new
                                         InputStreamReader(System.in));
        System.out.println("enter ur name : ");
        String name=br.readLine();
    }
}
```

```

        System.out.println("Hai "+name);
    }
}
class ExceptionNotHandle
{
    public static void main(String args[])throws IOException
    {
        Sample s=new Sample();
        s.accept();
    }
}

```

D:\bkr\Core java>javac ExceptionNotHandle.java

D:\bkr\Core java>java ExceptionNotHandle

enter ur name :

Rangha Reddy
Hai Rangha Reddy

throws: - throws statement is useful to throw out an exception without handling it.

Note: - if the programmer does not handle any type of exception we should write throws Exception.

throw:-it is useful to create an exception object and throw it out of try block

→ // Throw Example

```

class ThrowDemo
{
    static void Demo()
    {
        try
        {
            System.out.println("inside method");
            throw new NullPointerException("my data");
        }
        catch(NullPointerException ne)
        {
            System.out.println("ne");
        }
    }
    public static void main(String args[])
    {
        ThrowDemo.Demo();
    }
}

```

```

    }
}

```

D:\bkr\Core java>javac ThrowDemo.java

D:\bkr\Core java>java ThrowDemo

inside method

ne

→ **What is the difference between throws & throw?**

A) throws is used to throw out of an exception without handling it.

throw is used to create an exception & throwing it & handle it.

Uses of throw: - 1) throw is used in software testing to test whether a program is handling all the exceptions have climbed by programmer.

2) throw is used to create & throw user defined exceptions

Types of exceptions: -

1) **Built-in exceptions:** - These are the exceptions which are already available in java.

EX: -

- a) ArithmeticException
- b) ArrayIndexOutOfBoundsException
- c) StringIndexOutOfBoundsException
- d) NullPointerException
- e) NoSuchElementException
- f) ClassNotFoundException
- g) FileNotFoundException
- h) NumberFormatException
- i) RuntimeException
- j) InterruptedException

2) **User-defined exceptions:** - These are the exceptions created by user of the language.

Creating user defined exceptions: -

- a) Write user exception class as a sub class to exception class
Ex: - class MyException extends Exception

- b) Write a default constructor in user exception class
Ex – MyException() { }

- c) Write a string parameterized constructor & from their call the super class constructor.
Ex: - MyException(String str)

```

    {
        Super(str);
    }

```

- d) To raise the exception creates user exception class object & throw it using throw statement.

Ex: - throw me;

→ // user defined exception

```

class MyException extends Exception
{
    private static int accno[]={12,13,11,14,15};
    private static String name[]{"sun","Sud","sree","siri","latha"};
    private static double bal[]={345.53,5345.45,534.534,5435.56,324.643};

    MyException( )
    {
    }

    MyException(String str)
    {
        super(str);
    }

    public static void main(String args[])
    {
        try
        {
            System.out.println("accno "+'\t'+ "name" + "balance ");
            for(int i=0;i<5;i++)
            {
                System.out.println(accno[i]+'\t'+name[i]+'\t'+bal[i]);
                if(bal[i]<340.00)
                {
                    MyException me=new MyException("balance ammount is less");
                    throw me;
                }
            }
        catch(MyException me)
        {
            me.printStackTrace();
        }
    }
}

```

D:\bkr\Core java>javac MyException.java

D:\bkr\Core java>java MyException

accno namebalance

12 sun 345.53

13 Sud 5345.45

11 sree 534.534

14 siri 5435.56

15 latha 324.643

MyException: balance amount is less

at MyException.main(MyException.java:24)

Difference between throw and throws

	throw	throws
1	throw is a keyword used for hitting and generating the exception which are occurring as a part of method body	throws is a keyword which gives an indication to the specific method to place the common exception methods as a part of try and catch block for generating user friendly error messages
2	The place of using throw keyword is always as a part of method body.	The place of using throws is a keyword is always as a part of method heading
3	When we use throw keyword as a part of method body, it is mandatory to the java programmer to write throws keyword as a part of method heading	When we write throws keyword as a part of method heading, it is optional to the java programmer to write throw keyword as a part of method body.

Wrapper class:- All wrapper classes are defined in java.lang package.

A wrapper class wraps contains a primitive data type in its object.

Character is a wrapper class in to a char data type.

Primitive data type

char
byte
int
float
double
long

Wrapper class:-

Character
Byte
Integer
Float
Double
Long.

1) Character class:-

The character class wraps a value of the primitive type ‘char’ an object. An object of type character contains a single field whose type is char.

Constructors:-

→ 1) Character (char ch)

Ex: - char ch = ‘A’

Character obj=new Character (ch);

Methods:-

→ 1) char charValue ()

Returns the char value of the invoking object.

Ex: - char x =obj.charValue();

→ 2) static boolean isDigit(char ch)

Returns true if ch is a digit (0 to 9) otherwise returns false.

Ex:- char = ‘9’; boolean x = character.isDigit(ch);

→ 3) static boolean isLetter(char ch).

returns true if ch is a letter (A to Z or a to z)

→ 4) static boolean isUpperCase(char ch).

returns true if ch is an upper case letter (A to Z).

→ 5) static boolean isLowerCase(char ch)

returns true if ch is an Lower case letter (a to z).

→ 6) static boolean isSpaceChar(char ch).

returns true if ch is coming from Space Bar.

→ 7) static boolean isWhitespace(char ch)

returns true if ch is coming from Tab, Enter, Backspace.

→ 8) static char toUpperCase (char ch).

converts ch into upper case.

→ 9) static char toLowerCase (char ch)

Converts ch into lowercase.

→ Which of the wrapper classes does not a constructor with string parameter? (or) Which of the wrapper classes contains only one constructor?

A) Character

→ // test a character

```
import java.io.*;
class CharacterTest
{
    public static void main(String[] args) throws IOException
    {
        while(true)
        {
            // to accept data from key board
            BufferedReader br=new BufferedReader(new
                InputStreamReader(System.in));
            System.out.println("Enter a char");
            char ch=(char)br.read();
            // test the ch
            if(ch=='@')
                System.exit(0);
            else if(Character.isDigit(ch))
                System.out.println("It is a digit");
            else if(Character.isUpperCase(ch))
                System.out.println("It is a capital letter");
            else if(Character.isLowerCase(ch))
                System.out.println("It is a small Letter");
            else if(Character.isSpaceChar(ch))
                System.out.println("It is coming from space bar");
            else if(Character.isWhitespace(ch))
                System.out.println("It is a white space");
            else
                System.out.println("Sorry, i don't know that character");
        }
    }
}
```

D:\bkr\Core java>javac CharacterTest.java

D:\bkr\Core java>java CharacterTest

Enter a char

p

It is a small Letter

Enter a char

```

4
It is a digit
Enter a char
~
Sorry, i don't know that character
Enter a char
@

```

2) Byte class: - The byte class wraps a value of primitive type ‘byte’ in an object. An object of type byte contains a single field whose type is byte.

Constructor: -

1) `Byte(byte num)`

2) `Byte(String str)`

Ex: - 1) `byte b=99;`

```
Byte obj=new Byte(b);
```

2) `String str="120"`

```
Byte obj=new Byte(str);
```

Methods: -

→1) `byte byteValue()`

Returns the value of invoking object as a byte.

→2) `int compareTo(Byte b)`

Compares the numerical value of invoking object with that of ‘b’. returns 0, if the values all equal. Returns a –ve value, if the invoking object has a lower value. Returns a +ve value, if the invoking object has a grater value.

Ex: -`int n=b1.compareTo(b2)`

If `b1==b2;` `n==0;`

If `b1<b2;` `n== -ve;`

If `b1>b2;` `n== +ve;`

→3) `static byte parseBytes(String str)` throws `NumberFormatException`

Returns the byte equivalent ot the number contained in the string specified by ‘str’.

→4) `String toString()`

Returns a string that contains the decimal equivalent of the invoking object.

→5) `static ByteValueOf(String str)` throws `NumberFormatException`

Returns a byte object that contains the value specified by string ‘str’.

```
public class Test{
public static void main(String args[]){
Integer x =5;
// Returns byte primitive data type
System.out.println( x.byteValue());
// Returns double primitive data type
System.out.println(x.doubleValue());
// Returns long primitive data type
System.out.println( x.longValue());
}
}

→ // creating & comparing byte objects

import java.io.*;
class Bytes
{
    public static void main(String args[])throws IOException
    {
        BufferedReader br=new BufferedReader(new
                                         InputStreamReader(System.in));
        System.out.print("enter a byte no : ");
        String str=br.readLine();
        Byte b1=new Byte(str);
        System.out.print("enter another byte no : ");
        str=br.readLine();
        Byte b2=Byte.valueOf(str);
        int n=b1.compareTo(b2);
        if(n==0)
            System.out.println("both same ");
        else
        if(n>0)
            System.out.println(b1+"is bigger than"+b2);
        else
            System.out.println(b1+"is lesser than"+b2);
    }
}

D:\bkr\Core java>javac Bytes.java
D:\bkr\Core java>java Bytes
```

enter a byte no : 34

enter another byte no : 56

34 is lesser than 56

3) Integer Class: - The integer class wraps a value of the primitive type ‘int’ in a object. An object of type integer contains a single field whose type is int.

Constructors: -

- 1) Integer(int num)
- 2) Integer(String str)

Methods: -

→ 1) int intValue()

Returns the value of invoking object as an ‘int’.

→ 2) int compareTo(Integer obj)

Compare the numerical value of the invoking object of ‘obj’. Returns 0,-ve or +ve value.

→ 3) static int parseInt(String str) throws NumberFormatException

Returns int equivalent of the string str.

Ex: - String str="Hyderabad";

Int n=Integer.parseInt(str);

→ 4) String toString()

Returns a string from of the invoking object.

→ 5) static Integer valueOf(String str) throws NumberFormatException

Returns an integer object that contain the value shown by str.

→ 6) static String toBinaryString(int i)

Returns a string representation of the integer argument in base 2.

→ 7) static String toHexString(int i)

Returns a string representation of the integer argument in base 6.

→ 8) static String toOctalString(int i)

Returns a string representation of the integer argument in base 8.

→ // converting into other number system

```
import java.io.*;
class ConvertIntegers
{
    public static void main(String args[])throws IOException
```

```

{
    BufferedReader br=new BufferedReader(new
                                         InputStreamReader(System.in));
    System.out.println("Enter int :");
    String str=br.readLine();
    int n=Integer.parseInt(str);
    System.out.println("in Decimal =" +n);
    str=Integer.toBinaryString(n);
    System.out.println("In Binary =" +str);
    str=Integer.toHexString(n);
    System.out.println("In HexaDecimal =" +str);
    str=Integer.toOctalString(n);
    System.out.println("In Octal =" +str);
}
}

```

D:\bkr\Core java>javac ConvertIntegers.java

D:\bkr\Core java>java ConvertIntegers

Enter int :

123

in Decimal =123

In Binary =1111011

In HexaDecimal =7b

In Octal =173

4) Float class: - The float class wraps a value of primitive type ‘float’ in an object. An object of type float contains a single field whose type is float.

Constructors:-

- 1)Float (float num)
- 2)Float (String str)

Methods:-

→1) float floatValue()

returns the value of the involving object as a float.

→2) double doubleValue()

returns the value of the involving object as a double.

→3) int compareTo(Float f)

compares the numeric value of the involving object with that of ‘f’ returns .,-ve.+ve value.

→4) static float parseFloat (string str) throwsNumberFormatException

Returns the float equivalent the string str .

→5) String toString ()

returns the string equivalent of the involving object.

→6) static float valueOf (String str) throwsNumberFormatException

.Returns the float object with the value specified by string str.

5) double class: - The double class wraps a value of primitive type ‘double’ in an object. An object of type float contains a single field whose type is double.

Constructors:- →1) Double (double num)

→2) Double (String str)

Methods:-

→1) double floatValue()

returns the value of the involving object as a double.

→2) double floatValue()

returns the value of the involving object as a float.

→3) int compareTo(Double d)

compares the numeric value of the involving object with that of ‘d’ returns .,-ve.+ve value.

→4) static double parseDouble (string str)

throws NumberFormatException

Returns the double equivalent the string str .

→5) String toString ()

returns the string equivalent of the involving object.

→6) static double valueOf (String str) throws NumberFormatException

Returns the double object with the value specified by string str.

6) Math class: - The class math contains methods for performing numerical operations.

Methods: -

→1) static double sin(double arg)

Returns the sine value of arg. Arg is in radians.

→2) static double cos(double arg)

Returns the cosine value of arg. Arg is in radians.

→3) static double tan(double arg)

Returns the tangent value of arg. Arg is in radians.

→4) static double log(double arg)

Returns the natural logarithm value of arg.

→5) static double pow(double x,double y)

Returns x to the power of y value.

→6) static double sqrt(double arg)

Returns the square root of arg.

→7) static double abs(double arg)

Returns the absolute value of arg.

→8) static double ceil(double arg)

Returns the smallest integers which is greater or equal to arg.

Ex:- Math.ceil(4.5) → is 5.0

Ceil means ceiling. It means up.

→9) static double floor(double arg)

Returns the greater integer which is lower or equal to arg.

Ex: - Math.floor(4.5) → is 4.0

→10) static double min(arg1,arg2)

Returns the minimum value of arg1 & arg2.

→11) static double max(arg1,arg2)

Returns the maximum value of arg1 & arg2.

→12) static long round(arg)

Returns the rounded value.

Ex: - Math.round(4.6) → is 5

→13) static double random()

Returns the random numbers between 0&1.

→14) static double toRadians(double angle)

Converts angle in degrees in to radians.

→15) static double toDegrees(double angle)

Converts angle in radians in to degrees.

→ // random numbers between 1 & 10

```
class Rand
{
    public static void main(String args[]) throws InterruptedException
    {
        while(true)
        {
            double d=10*Math.random();
            int n=(int)d;
            System.out.print(n+"\t");
            if(n==0)
                System.exit(0);
            Thread.sleep(3000);
        }
    }
}
```

D:\bkr\Core java>javac Rand.java

D:\bkr\Core java>java Rand

7 7 8 5 6 2 7 6 1 0

****IO STREAMS****

***IO Stream**: Stream are used to perform sequential flow of data. Sequential of data one location to and other location. The streams are classified into two types. They are:

1. Byte Streams
2. Character Streams

1. **Byte Streams**: (Byte by Byte) The Streams that will perform the operations Byte by Byte are called as 'Byte Streams'. These Streams can handle any kind of data like Text, Audio, Video, Images etc. The Byte Streams are further classified into two categories.

1.1.**Input Stream**: These Streams are used to perform Reading operation from any resource using these Streams we can read data into an application.

Ex: FileInputStream

DataInputStream etc.

1.2.**OutputStream**: These Streams are used to perform writing operation to any resource. These Streams can be used to send data out of the applications.

Ex: FileOutputStream

BufferedOutputStream etc.

2. **Character Streams**: (char by char) The Streams that perform the operation character by character are called as character Streams. These Streams can handle only text. They are also called as Text Streams. The character Streams are faster than Byte Streams. The character Streams are further classified into two categories.

2.1. **Reader**: These Streams are similar to Input Streams performing Reading operations from various resources.

Ex: FileReader

BufferedReader

2.2. **Writer**: These Streams are similar to output Streams performing writing operations on to various resources.

Ex: `FileWriter`

`PrintWriter`

All the Stream related classes are available in “`java.io.package`”.

***DataInputStream**: This Stream is used to perform Reading operation from any resource.

Creation of DataInputStream

Syntax: `DataInputStream dis = new DataInputStream(resource);`

//pro from Read to keyboard.

```
import java.io.*;
```

```
class Readdemo{
```

```
public static void main(String[] args) throws IOException{
```

```
//Creating the object of DataInputStream and Connecting it to the resource.
```

```
DataInputStream dis = new DataInputStream(System.in);
```

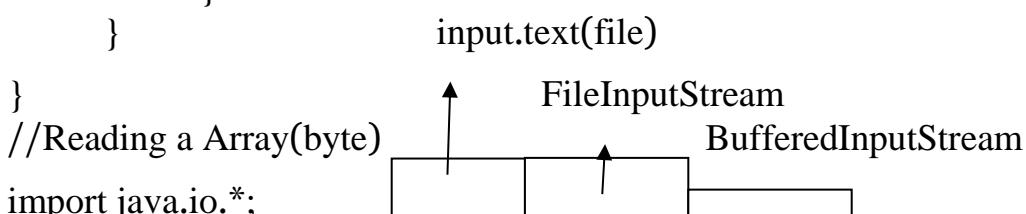
```
//reading multiple characters and displaying them
```

```
int ch;
```

```
while((ch = dis.read())!='$'){

System.out.println((char) ch);

    }
```



```
//Reading a Array(byte)
```

```
import java.io.*;
```

***FileInputStream**: This Stream is used to Read the contents from a file.

Syntax: FileInputStream fis = new FileInputStream(fileName);

If the fileName that is specified is not available then we get a runtime Exception called file not found Exception.

```
import java.util.*;
class FileRead{
public static void main(String[] args){
//connecting FileInputStream to a file
FileInputStream fis = new FileInputStream("input.txt");
//connection BufferedInputStream to FileInputStream
BufferedInputStream bis = new BufferedInputStream(fis);
//reading data from a file and displaying
int ch;
while((ch = bis.read())!= -1){
System.out.print((char)ch);
}
//releasing the resources
bis.close();
}
}
```



***FileOutputStream**: This class is used to write the contents into a file.

Syntax:

```
FileOutputStream fos = new FileOutputStream(fileName);
//This Syntax will overwrite the contents.

FileOutputStream fos = new FileOutputStream(fileName, boolean);
//This Syntax will append the contents.
```

```
import java.util.*;  
  
class Filecopy{  
public static void main(String[] args) throws Exception{  
FileInputStream fis = new FileInputStream("input.txt");  
FileOutputStream fos = new FileOutputStream("output.txt",true);  
int ch;  
while((ch = fis.read())!= -1){  
fos.write(ch);  
}  
fis.close();  
fos.close();  
}  
}
```

Note: If the output file i.e. specified is not available then we do not get file not found Exception nested a file with the specified name will be created automatically.

***FileReader:** This class is used to Read the content form a file 'Char by Char'. Cretion of file Reader.

```
Syntax: FileReader fr = new FileReader(fileName);  
  
import java.util.*;  
  
class FileReaderDemo{  
public static void main(String[] args) throws IOException{  
FileReader fr = new FileReader("input.txt");  
int ch;  
while((ch = fr.read())!= -1){  
System.out.print((char)ch);  
}  
fr.close();  
}  
}
```

*FileWriter: This file is used to write the contents into a file character by character.

Creation of filewriter.

Syntax:

```
FileWriter fw = new FileWriter(fileName); //over writing  
FileWriter fw = new FileWriter(fileName, boolean); //append
```

```
import java.util.*;  
  
class FileWriterdemo{  
public static void main(Stirng[] args) throws IOException{  
FileWriter fw = new FileWriter("abc.txt");  
String str = "abcdefghijklmnopqrstuvwxyz";  
fw.writer(str);  
char[] c = str.toCharArray();  
fw.write(c);  
for(int i =0; i<str.length(); i++){  
fw.write(str.charAt(i));  
}  
fw.close();  
}  
}  
***//program write using read a without IO package. ***  
  
import java.lang.*;  
  
class demo{  
public static void main(String[] args) throws Exception{  
char ch = (char) System.in.read(); //read  
System.out.print(ch); //write  
}  
}
```

```
***//using with do...while loop ***
***//program can read multiple character from keyboard.***
```

```
import java.util.*;
class demo2{
public static void main(String[] args) throws Exception{
char ch;
do{
ch =(char) System.in.read();
System.out.print(ch);
}
while(ch! = '@');
}
}
```

```
***//A program to read to line of data.***
import java.util.*;
class demo{
public static void main(String[] args) throws IOException{
BufferedReader br = new BufferedReader
(new InputStreamReader(System.in));
String str = br.readLine();
System.out.println(str);
}
}
```

Note:

InputStreamReader is a class that can be used for connecting a ByteStream and CharacterStream.

```
System.in InputStream //in = object, inputstream = class
System.out PrintStream //out = object, printstream = class
System.out.println(); //system = java.lang package, out = reference variable object
of PSC
```

***System.out.println**: out is a reference variable of print Stream class. Pointing to the object of printStream class declared as static inside System class.

A static reference variable presenting any class can be access directly by using class name. System.out will give us the object of printStream class, with that object we can call methods of printStream class.

Println method belong to printStream class.

//program to direct the connects into a file by using system.out.println.

```
import java.util.*;
class demo{
public static void main(String[] args) throws IOException{
FileoutputStream fos = new FileOutputStream("divert.txt");
printStream ps = new printStream(fos);
System.SetOut(ps);
System.out.println("hi");
System.out.println("h r u");
System.out.println("r u listening?");
}
}
```

***DeflaterStreams**:(java.util.Zip.*;) package:

DeflaterStreams these streams are used meant for compressing the data.

We can compress the data by reading the or writing the data.

DeflaterInputStream & DeflaterOutputStream both are meant for compressing the data.

Syntax:

```
DeflaterInputStream dis = new DeflaterInputStream(InputStream);
```

```
DeflaterOutputStream dos = new
```

```
DeflaterOutputStream(OutputStream);
```

* These Streams are used for uncompressing the data.

* We can uncompress the data while reading or while writing.

Syntax:

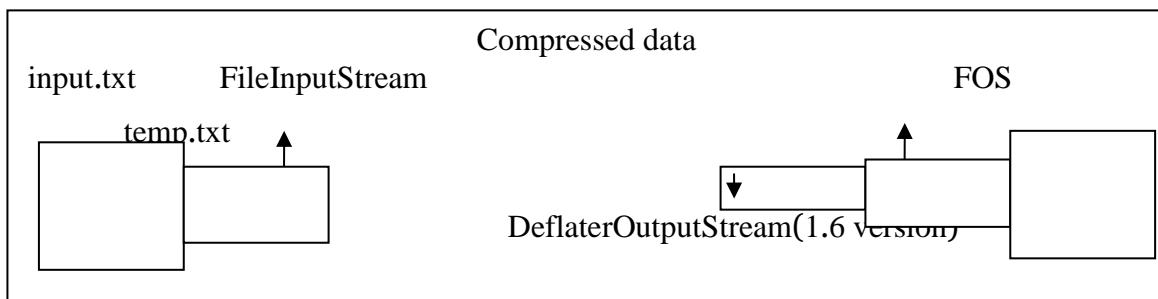
```
InflaterInputStream iis = new InflaterInputStream(InputStream);
InflaterOutputStream ios = new InflaterOutputStream(OutputStream);
```

All the InflaterStreams & DeflaterStreams are available in “import java.util.zip.*; package.

//program to compress the data

```
import java.io.*;
import java.util.zip.*;

class Compressdata{
public static void main(String[] args) throws IOException{
FileInputStream fis = new FileInputStream("input.txt");
FileOutputStream fos = new FileOutputStream("temp.txt");
DeflaterOutputStream dos = new DeflaterOutputStream(fos);
int ch;
while((ch = fis.read())!= -1){
dos.write(ch);
}
fis.close();
dos.close();
}
}
```



```
import java.util.*;
import java.util.*;
class Uncompressdata{
public static void main(String[] args) throws IOException{
```

```
FileInputStream fis = new FileInputStream("temp.txt");
InflaterInputStream iis = new InflaterInputStream(fis);
FileOutputStream fos = new FileOutputStream("output.txt");
int ch;
while((ch = iis.read())!= -1){
    fos.write(ch);
}
iis.close();
fos.close();
}
```

****Serialization:** It is a process of converting an object into stream of bytes.

***Deserialization:** It is a process of converting stream of into objects.

- An object is set to be serialized when it's corresponding class implementing serializable interface.
- We can prefer serialization and Deserialization only when the object is Serialized, otherwise a runtime exception and otherwise not serializable exception.
- The Serializable Interface is available in "java.io.*;" package and it contains 'zero' method or no methods.
- An interface does not contains methods are called as Tagged Interface or marked interface.
- Search kind of interface can be used to give instructions to jvm to perform a special operation.

***ObjectOutputStream:** This stream is used to write an object any resource.

***Creation of ObjectOutputStream:**

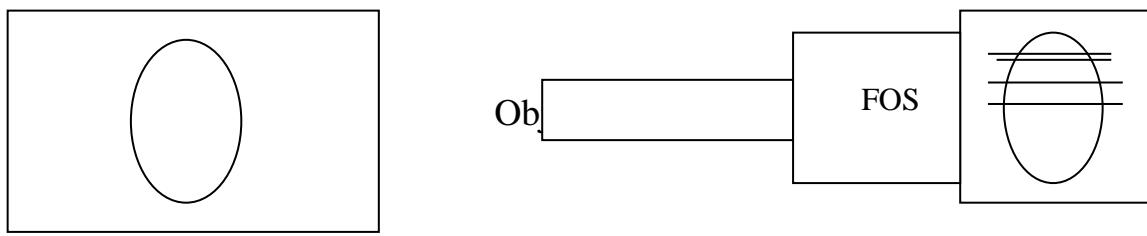
```
} ObjectOutputStream oos = new ObjectOutputStream(outputstream);
```

***Creation of InputStream:** This stream used to read an object.

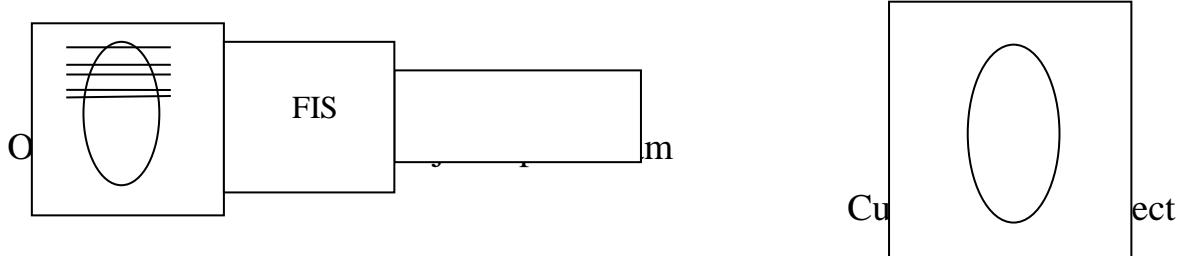
***Creation of ObjectInputStream:**

```
ObjectInputStream oos = new ObjectInputStream(ioutputstream);
```

```
//program for customer class
import java.util.*;
class Customer implements Serializable{
int custId;
String custName; // Instence Variables
Customer(int custId, String custName){
this.custId = custId;
this.custName = custName; // local variables
}
Public void showDetails(){
System.out.println(custId + “ “ + custName);
}
}
//program store object (or) writing object
import java.util.*;
class Storeobject{
public static void main(String[] args) throws IOException{
Customer c = new Customer(111, “inetsolv”);
FileOutputStream fos = new FileOutputStream(“object.txt”);
ObjectOutputStream oos = new ObjectOutputStream(fos);
oos.writeObject(c);
oos.close();
}
}
```



```
//program is a Reading the objects
import java.util.*;
class Readobject{
public static void main(String[] args) throws IOException{
FileInputStream fis = new FileInputStream("object.txt");
ObjectInputStream ois = new ObjectInputStream(fis);
Object o = ois.readObject();
Customer c = (Customer)o;
(OR)
Customer c = (Customer) readObject();
c.showDetails();
ois.close();
}
}
```



```
//A program read the contents by using the System.in
import java.io.*;
class Read{
public static void main(String[] args) throws IOException{
FileInputStream fis = new FileInputStream("Read.java");
System.setIn(fis);
int ch;
while((ch = System.in.read())!= -1){
System.out.print((char) ch);
```

```
        }
    }
}

import java.io.*;
class Xcopy
{
public static void main(String[] args)
{
FileInputStream fis=null;
FileOutputStream fos=null;
if (args.length!=2)
{
System.out.println ("INVALID ARGUMENTS..!");
}
else
{
try
{
fis=new FileInputStream (args [0]);
fos=new FileOutputStream (args [1], true);
int i;
do
{
i=fis.read ();
char ch=(char)i;// type casting int value into char value
fos.write (ch);
}
while (i!=-1);
}
catch (FileNotFoundException fnfe)
{
System.out.println (args [0]+"DOES NOT EXIST..!");
}
catch (IOException ioe)
{
```

```
System.out.println (ioe);
}
catch (Exception e)
{
e.printStackTrace ();
}
finally
{
try
{
if (fis!=null)// file is closed when it is opened
{
fis.close ();
}
if (fos!=null)
{
fos.close ();
}
}
catch (IOException ioe)
{
ioe.printStackTrace ();
}
catch (Exception e)
{
e.printStackTrace ();
}
}// finally
}// else
}//main
}
```

Fill class:

```
import java.io.*;
class FileDemo
{
public static void main(String args[])throws IOException
{
File f=new File("corejava.txt");
boolean b=true;
b=f.createNewFile();
System.out.println("file created");
//f.delete();
//System.out.println("file deleted");
boolean c=f.canRead();
if(c==true)
{
System.out.println("file haveing read per");
}
else
{
System.out.println("file do not haveing read per");
}
boolean d=f.canWrite();
if(d==true)
{
System.out.println("file haveing write per");
}
else
{
System.out.println("file do not haveing write per");
}
boolean e=f.canExecute();
if(e==true)
{
System.out.println("file haveing exe per");
}
else
{
System.out.println("file do not haveing exe per");
}
}}
```

```
import java.io.*;
class FilePerDemo
{
public static void main(String args[])
{
FilePermission fp=new FilePermission("corejava","read");
FilePermission fp1=new FilePermission("corejava","write");
FilePermission fp2=new FilePermission("corejava","execute");
String s=fp.getActions();
System.out.println("actions="+s);
String s1=fp1.getActions();
System.out.println("actions="+s1);
String s2=fp2.getActions();
System.out.println("actions="+s2);
}
}
```

Collections:*Array:**

An array is collection of elements which are stored in continuous memory locations.

Limitations of array (or) disadvantages of array:

1. The size of an array is fixed; it can't be increased or decreased. Some times. The memory may be wasted or memory may not be sufficient.
2. To perform the operations like Insertion, Deletion, Searching, Sorting etc. the program has to write their own logic.
3. Because of the problems in an array the java people or java s/w people have come up with collection framework introduction java 1.2 versions.

Differences between Arrays and Collections ?

Arrays	Collections
1) Arrays are fixed in size.	1) Collections are growable in nature.
2) Memory point of view arrays are not recommended to use.	2) Memory point of view collections are highly recommended to use.
3) Performance point of view arrays are recommended to use.	3) Performance point of view collections are not recommended to use.
4) Arrays can hold only homogeneous data type elements.	4) Collections can hold both homogeneous and heterogeneous elements.

5) There is no underlying data structure for arrays and hence there is no readymade method support.

6) Arrays can hold both primitives and object types.

5) Every collection class is implemented based on some standard data structure and hence readymade method support is available.

6) Collections can hold only objects but not primitives.

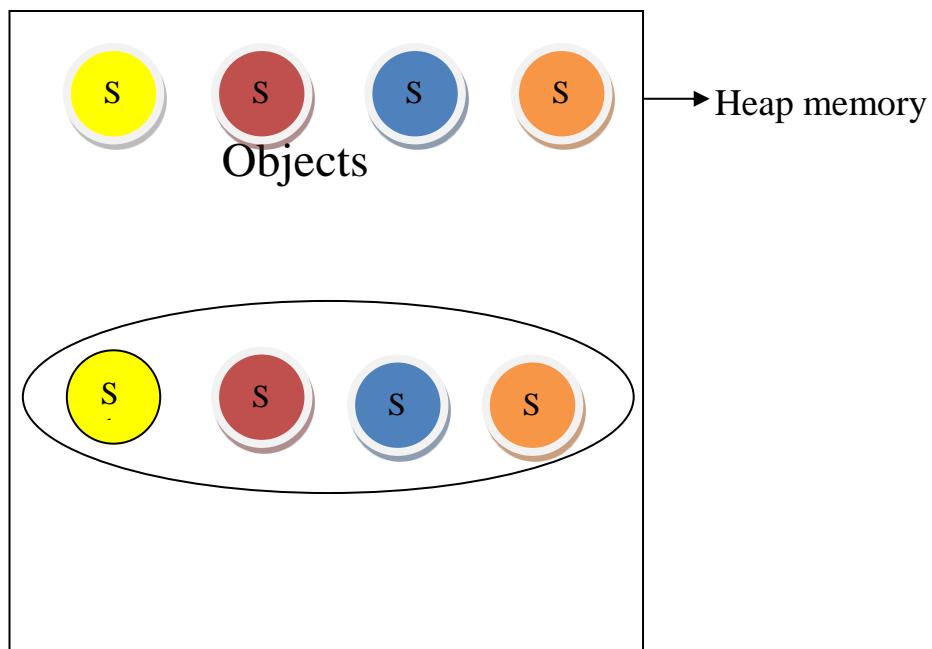
Collection object:

An object is said to be collection object if it holds or stores a group of other objects.

Collection class:

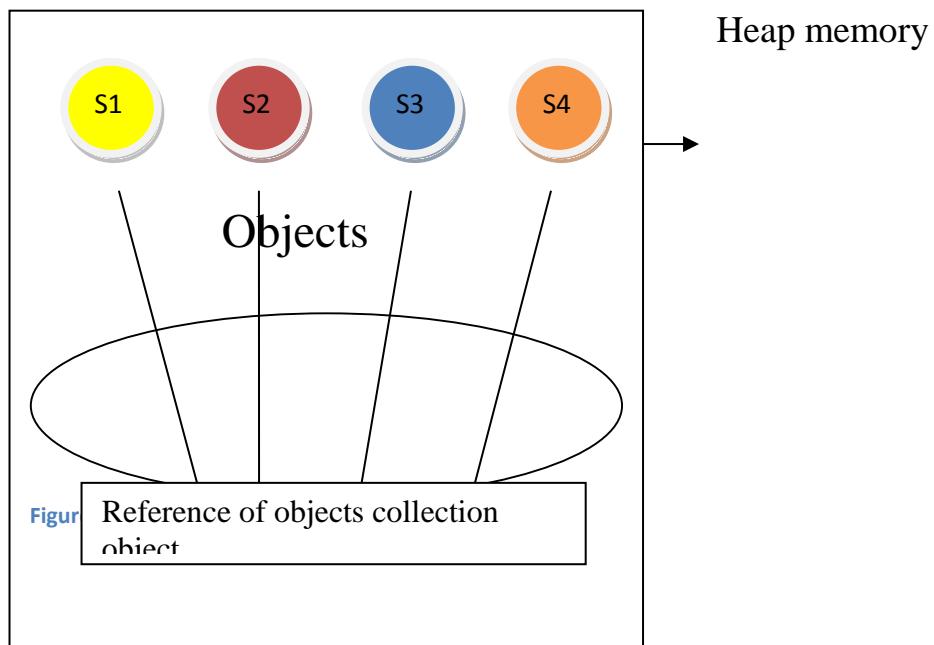
A collection class is a class whose object can store group of other objects.

Ex:



The objects performed by s1,s2,s3 and s4 are stored multiple times there by wasting the memory with in the JVM.

To save the memory with in the jvm when the objects are stored in the collection object, the jvm stores the references of the objects with in the collection objects inside of storing the objects directly.



What is framework in java

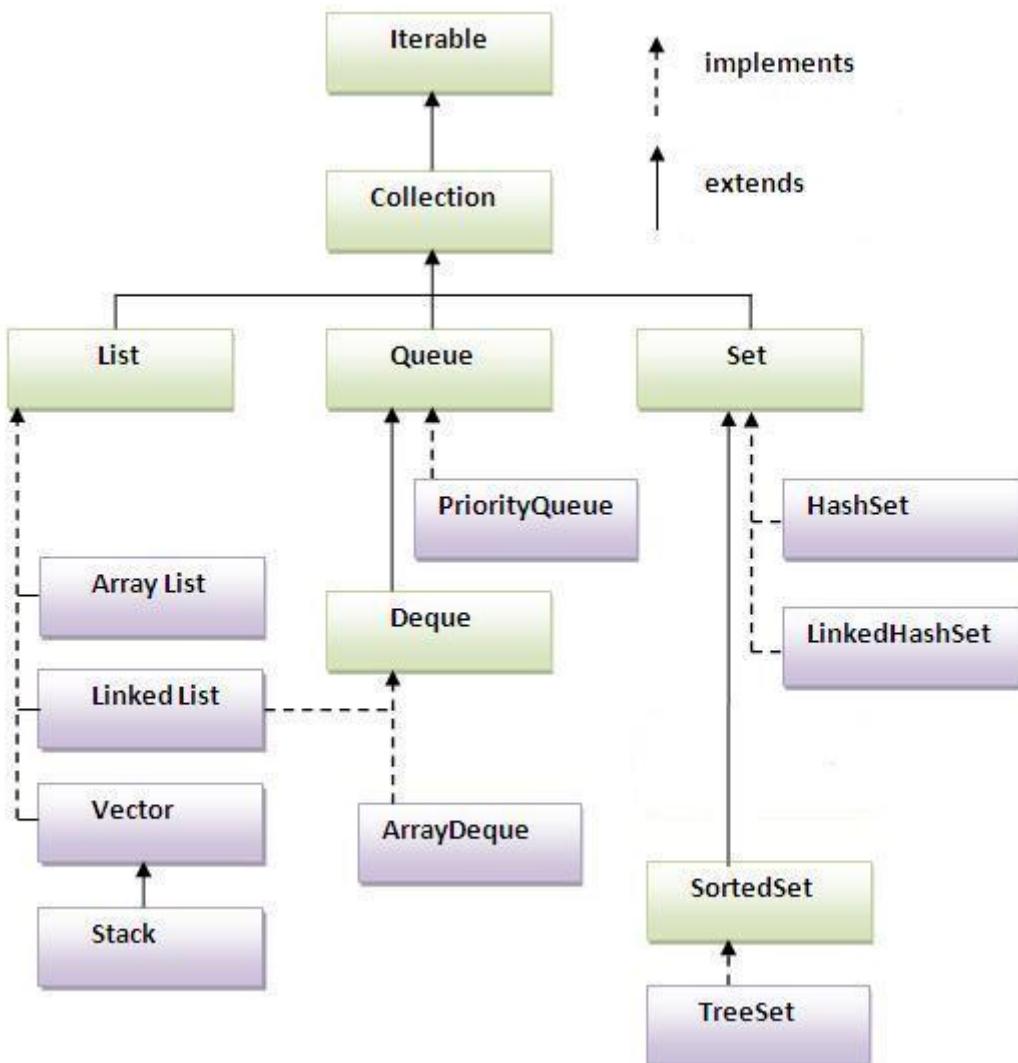
- provides readymade architecture.
- represents set of classes and interface.
- is optional.

The collections are designed to store only objects that is the collections can not store primitive type values.

All the collection classes are available in “java.util” (utility) package.

All the collection interfaces and collection class and together as collection frame work.

All the collection classes are categorized into three groups' i.e.



List: this category is used to store group of individual elements where the elements can be duplicated. List is an Interface. “ArrayList, Linked list and vector” are the implementations class of list interfaces.

Set: this category is used to store a group of individual elements. But they elements can't be duplicated. Set is an interface.

Hash set, Linked Hash set and Tree set implementations of set interface.

Map: this category is used to store the element in the form key value pairs where the keys can't be duplicated, values can be duplicated.

Map is an interfaces “HashMap, Linked Hash Map, Tree Map and Hash table are the implementation class of Map interface.

Java.utility: -

Stack: - A stack represents arrangement of elements (objects) in LIFO (last in first out) order.

Inserting the elements & deleting the elements will take place only from one side of the stack, called top of the stack. The other side is closed & it called bottom of the stack.

Inserting elements into stack is called push operation. Deleting element from top of the stack is called pop operation. Searching for an element in the element is called peep operation.

Ex: - A file of plates in a hotel, linking of coaches to a railway engine, CD holder, bangles to hand.

How to create a stack:

→1) To create a stack

```
Stack st=new Stack();
```

→2) To know whether a stack is empty or not, use empty()

```
boolean b=st.empty();
```

→3) To push an element in to the stack

```
st.push(element);
```

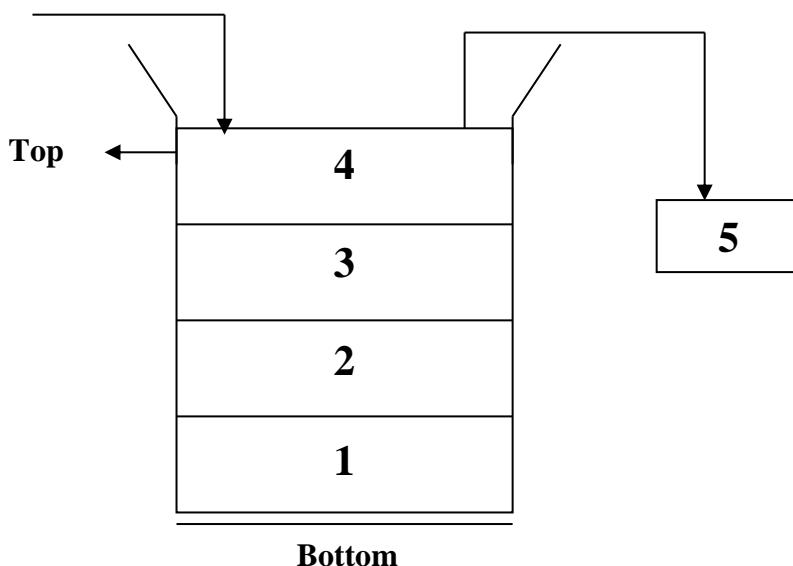
→4) To pop an element from a stack

```
Object element=st.pop();
```

→5) to search for an element in the stack

```
int i=st.search(element);
```

This method returns -1, if the element is not found.



→ // A stack with int values

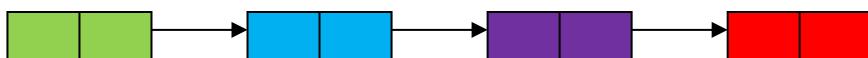
```
import java.io.*;
import java.util.*;
class StackDemo
{
    static void stpush(Stack st,int a)
    {
        st.push(new Integer(a));
    }
    static void stpop(Stack st)
    {
        Integer a=(Integer)st.pop();
        System.out.println("popped = "+a);
    }
    static int stSearch(Stack st,int a)
    {
        int i=st.search(new Integer(a));
        return i;
    }
    public static void main(String args[])throws Exception
    {
        Stack st=new Stack();
        int element,pos,choice=0;
        BufferedReader br=new BufferedReader(new
                                         InputStreamReader(System.in));
        // menu
        while(choice<4)
        {
            System.out.println("stack operations ");
            System.out.println("1 push element ");
            System.out.println("2 pop element ");
            System.out.println("3 search for element ");
            System.out.println("4 exit ");
            System.out.print(" your choice ");
            choice=Integer.parseInt(br.readLine());
        }
    }
}
```

```
// perform a task depending on choice
switch(choice)
{
    case 1:
        element=Integer.parseInt(br.readLine( ));
        stpush(st,element);
        break;
    case 2:
        stpop(st);
        break;
    case 3:
        System.out.print("enter element :");
        element=Integer.parseInt(br.readLine( ));
        pos = stSearch(st,element);
        if(pos== -1)
            System.out.println("element not found ");
        else
            System.out.println("element found at position :" +pos);
        break;
    default:
        return;
}
System.out.println("stack =" +st);
}
```

Linked list: -It represents a set of nodes such that each node contains two fields.



- 1) The data field stores data.
 - 2) The link field stores reference of the next node



Create a linked list: -

→1) To create a linked list:

```
LinkedList ll=newLinkedList( );
```

→2) To add elements to a linked list

```
ll.add(element);
```

To add element in second position

```
ll.add(2,element);
```

→3) To remove first element from the linked list

```
ll.remove( );
```

To remove last element from the linked list.

```
ll.removeLast( );
```

To remove second element

```
ll.remove(2);
```

→4) To change the second element with a new element

```
ll.set(2>New element);
```

→ to retrieve element by element from collection objects. We can use the following one of the following interfaces

- 1) Iterator
- 2) ListIterator
- 3) Enumeration

→ // a linked list with string

```
import java.io.*;  
import java.util.*;  
class LLDemo  
{  
    public static void main(String args[])throws IOException  
    {  
        // create an empty linked list  
        LinkedList ll=new LinkedList( );  
        // add elements to ll  
        ll.add("America");  
        ll.add("India");  
        ll.add("China");  
        ll.add("Japan");  
        ll.add("France");  
        ll.add("Sri lanka");  
    }  
}
```

```
// display the linked list
System.out.println("Linked List = "+ll);
// vars
BufferedReader br=new BufferedReader(new
InputStreamReader(System.in));
String element;
int pos,choice=0;
// menu
while(choice<4)
{
    System.out.println("Linked List operations ");
    System.out.println("1 . insert element : ");
    System.out.println("2 . remove element : ");
    System.out.println("3 . change element : ");
    System.out.println("4 . exit ");
    System.out.println(" your choice :");
    choice=Integer.parseInt(br.readLine());
    // depending on choice, perform a task
    switch(choice)
    {
        case 1:
            System.out.print("Enter element : ");
            element=br.readLine();
            System.out.print("Enter Position No : ");
            pos=Integer.parseInt(br.readLine());
            ll.add(pos,element);
            break;
        case 2:
            System.out.print("Enter Position No : ");
            pos=Integer.parseInt(br.readLine());
            ll.remove(pos);
            break;
        case 3:
            System.out.print("Enter element : ");
```

```

        element=br.readLine( );
        System.out.print("Enter Position No : ");
        pos=Integer.parseInt(br.readLine( ));
        ll.set(pos,element);
        break;
        default :
        return;
    }
    System.out.println("List =" +ll);
}
}
}

```

Arrays: - Arrays class contains methods to handle any array.

→1) To sort the elements of an array in ascending order.

Arrays.sort(arr);

→2) To sort only a range of elements of the array

Arrays.sort(arr,start,end);

These sort elements of ‘arr’ starting from ‘start’ till ‘end’ -1 element.

→3) To search for an element in an array

Arrays.binarySearch(arr,element);

This searches for ‘element’ in the array ‘arr’ & returns its position. If the ‘elements’ is not found, it returns a –ve value.

Note: - Binary search() will not act only a sorted array.

→4) To compare two arrays, to know if they are same or not.

Arrays.equals(arr1,arr2);

This returns true, if arr1 & arr2 are same, else it returns false.

→ // sorting & searching in an array

```

import java.io.*;
import java.util.*;
class ArraysDemo
{
    public static void main(String args[])throws IOException
    {
        // to accept data from key board
        Scanner sc=new Scanner(System.in);
        System.out.print("How many elements you want : ");

```

```
int n=br.nextInt();
// create int type array
int arr[] = new int[n];
// store elements into arr
for(int i=0;i<n;i++)
{
    System.out.print("Enter element : ");
    arr[i]=Integer.parseInt(br.readLine());
}
// display the elements
System.out.print("Your Array is : "+ "\t");
display(arr);
// sort the elements in to ascending order
Arrays.sort(arr);
// display the sorted elements
System.out.print("Sorted array is : "+ "\t");
display(arr);
// searches for an element in an array
System.out.println("Enter element to Search : ");
int x=Integer.parseInt(br.readLine());
int pos=Arrays.binarySearch(arr,x);
if(pos<0)
    System.out.println("Element not Found");
else
    System.out.println("it position is :" +(pos+1));
}
static void display(int arr[])
{
    for(int i:arr)
        System.out.println(i);
}
D:\bkr\Core java>javac ArraysDemo.java
D:\bkr\Core java>java ArraysDemo
How many elements you want : 4
```

```
Enter element : 7
Enter element : 9
Enter element : 3
Enter element : 6
Your Array is : 7    9    3    6
Sorted array is : 3    6    7    9
```

Enter element to Search :

9
it position is :4

ArrayList: this class is an implementation class of list interface. This class is similar to an Array but its size can be increased or decreased. This class is used to store individual elements which can be duplicated. This class is “not synchronized”.

If an object is synchronized then only one thread can use the object at the same time, if an object is not synchronized then multiple threads can access the object at the same time, which may lead to data inconsistency problems.

Creation of ArrayList:

Syntax: `ArrayList<E> al = new ArrayList<E>();`

`ArrayList<E> al = new ArrayList <E>(int initialcapacity);`

Here, E represents element data type

Methods of ArrayList:

Boolean add(Element obj): this method is used to place the specified element to the end of List.

Void add(int position, Element obj): this method is used to insert an element at the specified position.

Boolean remove(Element obj): this method is used to remove the 1st occurrence of the specified element.

Element remove(int position): this method is used to remove an element from the specified position.

Void clear(): this method removes all the elements available in the ArrayList.

Int Size(): this method will return the count of the no.of elements available in the Array List.

Boolean contains(element obj): this method returns true if the specified element is available in the Array List.

Element get(int position): this method is used to access an Element that is available in the specified position.

Element set(int position, Element obj): this method is used replace an element in the specified position with the specified element.

```
Import java.util.*;  
class ArrayList demo{  
Public static void main(string[] args){  
//creation of ArrayList  
ArrayList<String> al = new ArrayList<String>();  
//adding elements to the ArrayList  
al.add("Nokia");  
al.add("Samsung");  
al.add("Sony");  
al.add("Celkon");  
al.add("HTC");  
//insert an element into ArrayList  
al.add(3,"Motorla");  
//displaying the elements  
System.out.println("List:"+al);  
//deleting the elements  
al.remove("HTC");  
al.remove(1);  
System.out.println("List:"+al);
```

```
//displaying the size  
System.out.println("size:"+al.size());  
//displaying the elements using iterator  
Iterator it = al.iterator();  
while(it.hasNext()){  
    System.out.println(it.next());  
}//end of while  
}//end of main  
}//end of class
```

Note: Iterator is used for displaying the elements one by one. (iterator is an interface)

Difference between ArrayList and LinkedList:

ArrayList is an implementation class which follows array structure. ArrayList is faster in Accessing the elements and slower in insertion and Deletion.

LinkedList is an implementation class of List interface which follows tree structure. LinkedList is slower in Accessing the elements and faster in insertions and deletion.

Difference between ArrayList and Vector:

The vector class is exactly similar to ArrayList. But it is synchronized.

Vector class:

This class is similar to ArrayList which can store group of individual elements. It allows storing duplicate values. Vector is a synchronized class.

Creation of vector:

```
Vector<E> v = new Vector<E>();
```

```
Vector<E> v = new Vector<E>(int initial capacity);
```

Methods of vector:

```
boolean add(Element obj)  
void add(int position, Element obj)  
boolean remove(Element obj)  
Element remove(int position)
```

```
void clear()
int size()
boolean contains(Element obj)
Element get(int position)
Element get(int position, Element obj)

//Vector program
Import java.util.*;
class vectordemo{
public static void main(String[] xyz){
//creation of vector
Vector<Integer> v = new Vector<Integer>();
//adding elements to vector
v.add(new Integer(11));
v.add(new Integer(22));
v.add(new Integer(33));
v.add(44); //auto boxing
v.add(1,99);
v.add(100);
//displaying the elements
System.out.println("List:" + v);
//deleting the elements
v.remove(new Integer(22));
v.remove(1);
//displaying the elements using general for loop
System.out.print("List using for loop:");
for(int i=0;i<v.size();i++){
System.out.print(v.get(i) + " ");
}//displaying the elements using for each loop
```

```
System.out.print("\n List using for each loop:");
for(int i:v)//unboxing{
    System.out.print(i+"");
}
ListIterator lit = v.listIterator();
//displaying the elements in forward direction
System.out.print("\n forward direction:");
while(lit.hasNext()){
    System.out.print(lit.next()+" ");
}
//displaying the element in backward direction
System.out.print("\n backward direction:");
while(lit.hasPrevious()){
    System.out.println(lit.previous()+" ");
}
//end of while
}
//end of while
}
//end of class
```

Boxing: It is the process of converting a primitive type to object type and this process will be down automatically.

Unboxing: It is the process of converting an object type to primitive type and this process will be down automatically.

Note: The for each loop is design to work with Arrays and collections only, It is not a general purpose loop.

***Difference between Iterator and List Iterator:** Iterator and List Iterator both are Interfaces use for accessing the elements. The Iterator can be used to access the elements in forward direction only. Where as List Iterator accessing the elements in both forward and reserve direction.

***Hashset class:** This class is the implementation class of set interface. It does not Synchronized. It does not guarantee the order of insertion.

***Creation of HashSet:**

```
HashSet<E> hs = new HashSet<E>();
```

```
HashSet<E> hs = new HashSet<E>(int capacity);
```

***Methods of HashSet:**

1. **Boolean add(Element obj):** This method is used to place the specified element into the set.

2. **Boolean remove(Element obj):** This method is used to delete the specified element from the set. If it is available.

3. **Boolean contains(Element obj):** This method return true if the specified element is available in the set.

4. **Boolean isEmpty():** This method return true if the set isEmpty.

5. **int size():** This method returns the count of the no.of elements available in the set.

6. **void clear():** This method is used to delete all the elements from the set.

```
Import java.util.*;
```

```
class Hashsetdemo{  
public static void main(String[] args){  
    HashSet<Integer> hs = new HashSet<Integer>();  
    hs.add(12);  
    hs.add(23);  
    hs.add(34);  
    hs.add(45);  
    hs.add(56);  
    hs.add(67);  
    System.out.println("set:" + hs);  
    Iterator it = hs.iterator();  
    while(it.hasNext()){  
        System.out.print(it.next() + " ");  
    }  
}}
```

```
}
```

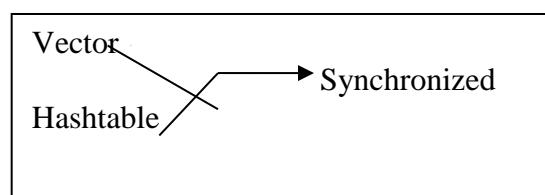
***Linked HashSet**: This class is similar to HashSet class. But it will guarantee order of insertion.

***TreeSet**: This class is used to store group of individual elements, it does not allow duplicate values, it is not synchronized. It stores the elements following the natural order. (ascending/sorting order).

Note: The HashSet and LinkedHashSet will internally follow hashing technique which will reduce the searching time.

```
Import java.util.*;  
  
class LinkedHashSetdemo{  
public static void main(String[] args){  
  
LinkedHashSet<Integer>lhs = new LinkedHashSet<Integer>();  
  
lhs.add(12);  
lhs.add(23);  
lhs.add(34);  
lhs.add(45);  
lhs.add(56);  
lhs.add(67);  
  
System.out.println("Set:" +lhs);  
Iterator it = lhs.iterator();  
While(it.hasNext()){  
System.out.println(it.next() + " ");  
}  
}  
}  
//Treeset  
  
import java.util.*;  
  
class TreeSetdemo{  
public static void main(String[] args){  
TreeSet<Integer> ts = new TreeSet<Integer>();
```

```
ts.add(12);
ts.add(23);
ts.add(34);
ts.add(45);
ts.add(56);
ts.add(67);
System.out.println("set:" + ts);
Iterator it = ts.iterator();
while(it.hasNext()){
    System.out.println(it.next() + "");
```



***Map:**

1. **HashMap**: It is an implementation class of Map Interface. This class is used to store the elements in the form of key value pairs. The keys must be unique and the values can be duplicated.

HashMap class is not synchronized. This class does not guarantee the order of Insertion.

*creation of HashMap:

```
HashMap<k,v> hm = new HashMap<k,v>();
```

```
HashMap<k,v> hm = new HashMap<k,v>(int capacity);
```

Here **K** represents the type of the key and **V** represents the type of the value.

*Methods of HashMap:

1. **value put(object key, object value)**: This method is used to place a key value pair into the HashMap.

2. **value remove(object key)**: This method is used remove the specified key and its corresponding value.

3. value get(object key): This method will return the value of the key that is specified.

- 4.**Set keyset()**: This method returns all the keys available in the hashmap in the form of set.
- 5.**collection values()**: This method returns all the values that are available in the hashmap in the form of a collection.
- 6.**void clear()**: This method is used to remove all the keyvalue pairs.
- 7.**int size()**: This metod use the count of the no.of keyvalue pairs available in the hashmap.
- 8.**boolean isEmpty()**: This method returns true if the map isEmpty.
- 9.**boolean containskey(object key)**: This method returns true if the specified key is available in the hashmap.
- 10.**boolean containsvalue(object value)**: This method return true if the specified value is available in the hashmap.

```
//HashMap  
import java.util.*;  
  
class Hashmapdemo{  
public static void main(String[] args){  
  
HashMap<String,Integer> hm = new HashMap<String,Integer>();  
hm.put("amit",90);  
hm.put("Salman",80);  
hm.put("Khan",70);  
hm.put("hari",60);  
hm.put("amit",50);  
hm.put("Ravi",99);  
  
System.out.println(hm); //System.out.println("Elements:"+hm);  
hm.remove("hari");  
set<String> s = hm.keySet();  
System.out.println("keys:"+s);  
Iterator it = s.iterator();  
while(it.hasNext()){
```

```

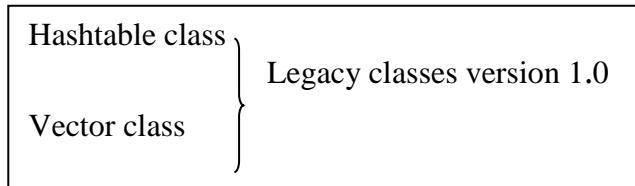
Object o = it.next();
String str = (String) o;
System.out.println(str + " " + hm.get(str));
}
collection<Integer> c = hm.values();
System.out.println("values:" + c);
}

```

***LinkedHashMap**: This class is used to store the elements in the form key value pairs. Keys can not be duplicated where as the values can be duplicated. This class is a not synchronized. This class will guaranty the order of Insertion.

***TreeMap**: This class is used to store the elements in the form key value pairs. The keys cannot be duplicated where as the values can be duplicated. This class is a not synchronized. This class will store the element in the sorted (natural) order sorted based on the keys.

***Hashtable**: This class is exactly similar to HashMap. But it is synchronized. The Hashtable class does not allow null into the keys or into the values. (1.2 version before called legacy classes)



```

import java.util.*;
class Hashtable demo{
public static void main(String... args){
Hashtable<String,Integer> ht = new Hashtable<String,Integer>();
ht.put("xyz ",67);
ht.put(" mno",23);
ht.put(" pqr",89);

```

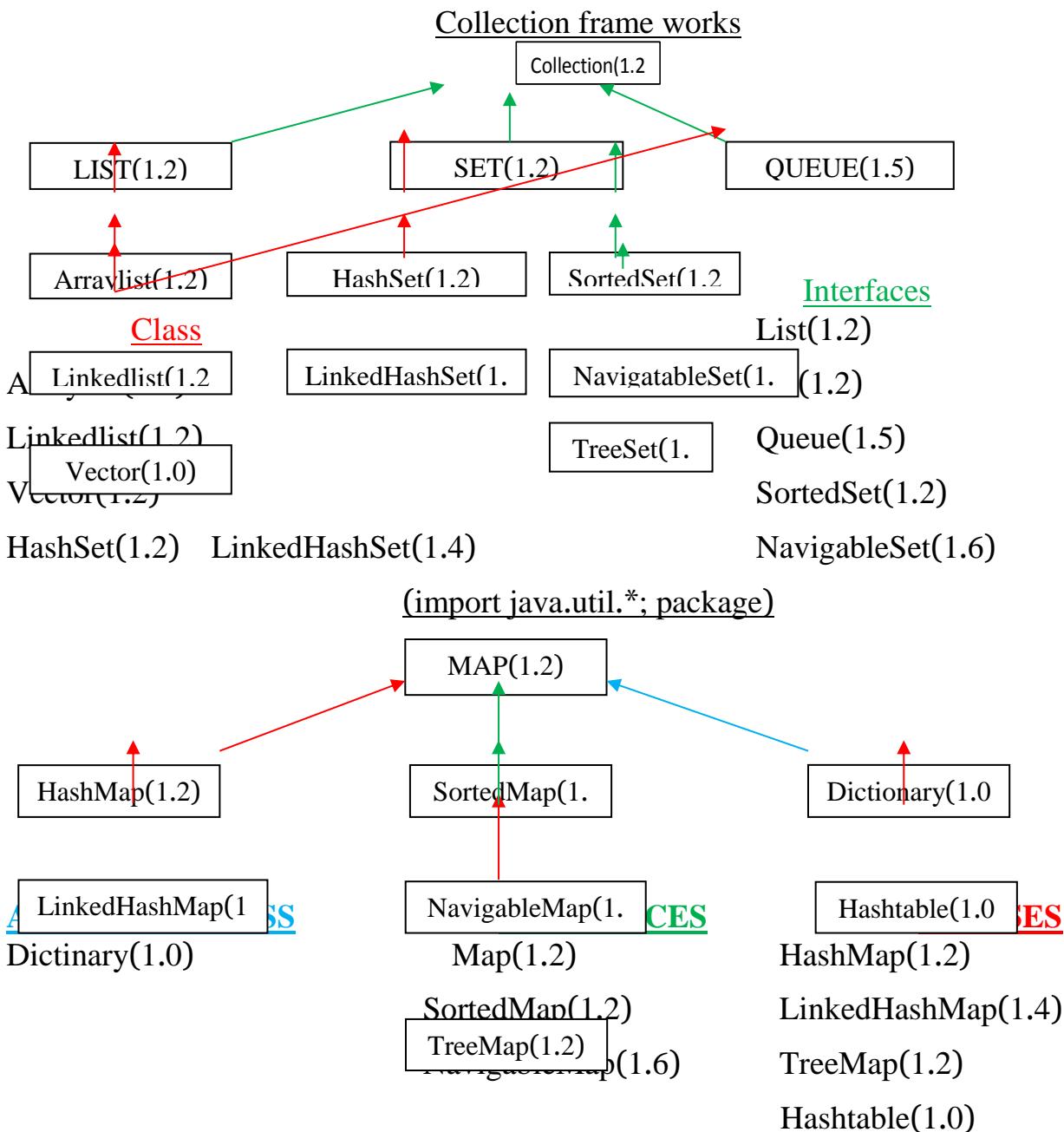
```
ht.put("ijk",39);
ht.put("abc",89);
ht.put("def",99);
System.out.println(ht);
Enumeration e = ht.keys();
while(e.hasMoreElements()){
System.out.println(e.nextElement());
}
}
}
//LinkedHashMap
import java.util.*;
class LinkedHashMapdemo{
public static void main(String[] args){
LinkedHashMap<String,Integer> lhm = new LinkedHashMap<String,Integer>();
lhm.put = ("Ravi",11);
lhm.put = ("Salman",22);
lhm.put = ("hari",33);
System.out.println(lhm);
Set<String> s = lhm.keySet();
System.out.println("key:" +s);
Iterator it = S.iterator();
while(it.hasNext()){
Object o = it.next();
String str = (String)o;
System.out.println(str + " " +lhm.get(str));
}
Collection<Integer> c = lhm.values();
System.out.println("Values: " +c);
}
```

```
//TreeMap  
import java.util.*;  
class TreeMapdemo{  
public static void main(String[] args){  
TreeMap<String,Integer> tm = new TreeMap<String,Integer>();  
tm.put("amit",68);  
tm.put("Khan",89);  
tm.put("hari",66);  
System.out.println™;  
tm.remove("Khan");  
Set<String> s = tm.keySet();  
System.out.println("keys:" + s);  
Iterator it = s.iterator();  
while(it.hasNext()){  
Object o = it.next();  
String str = (String)o;  
System.out.println(str + " " + tm.get(str));  
}  
Collection<Integer> c = tm.values();  
System.out.println("Values: " + c);  
}  
}
```

***cursors of collection frame work**: This cursors are used to Access the elements one by one and perform some other operations. They are 3 cursors and they are:

1. **Iterator**: This cursor can be applied to all the collection classes and it can be used to accesses the elements in forward direction only.
2. **ListIterator**: This cursor can be applied to all the List implementation classes and it can be used to accesses the element and both forward and backward direction.
(ArrayList,LinkedList,Vector);

3. **Enumeration**: This cursor can be applied to only the legacy classes and it can be used to access the elements in forward direction only. It can be used only for legacy interfaces.



* **StringTokenizer**: This class is used to break a String into multiple tokens (pieces).

Syntax: StringTokenizer st = new StringTokenizer(String, delimiter);

```
import java.util.*;
class sdemo{
```

```
public static void main(String[] args){  
    String str = "one a two a three a four a five a six a seven a eight a nine a ten";  
    StringTokenizer st = new StringTokenizer(str,",");  
    System.out.println(st.CountTokens());  
    while (st.hasMoreTokens()) { //has MoreElements  
        System.out.println(st.nextToken());  
    }  
}
```

***Calender & Date classes:** These classes are used to retrieve the date and time. In an application and use them accordingly.

```
import java.util.*;  
  
class Datedemo{  
    public static void main(String[] args){ //getInstance() is a factory Method  
        Calender c = Calender.getInstance();  
        int date = c.get(Calender.DATE);  
        int month = c.get(Calender.MONTH);  
        int year = c.get(Calender.YEAR);  
        System.out.println("Date:" +date+ "/" +(++month)+ "/" +year);  
        int hour = c.get(Calender.Hour);  
        int minute = c.get(Calender.MINUTE);  
        int second = c.get(Calender.SECOND);  
        System.out.println("Time =" +hour+ ":" +minute+ ":" +second);  
        Date d = new Date();  
        System.out.println("Date:" + d);  
    }  
}
```

Java Comparable interface

Java Comparable interface is used to order the objects of user-defined class. This interface is found in `java.lang` package and contains only one method named `compareTo(Object)`. It provides single sorting sequence only i.e. you can sort the elements on based on single data member only. For example it may be rollno, name, age or anything else.

compareTo(Object obj) method

public int compareTo(Object obj): is used to compare the current object with the specified object.

We can sort the elements of:

1. String objects
2. Wrapper class objects
3. User-defined class objects

Collections class

Collections class provides static methods for sorting the elements of collections. If collection elements are of Set or Map, we can use TreeSet or TreeMap. But We cannot sort the elements of List. Collections class provides methods for sorting the elements of List type elements.

Method of Collections class for sorting List elements

public void sort(List list): is used to sort the elements of List. List elements must be of Comparable type.

Note: String class and Wrapper classes implements Comparable interface by default. So if you store the objects of string or wrapper classes in list, set or map, it will be Comparable by default.

Java Comparable Example

Let's see the example of Comparable interface that sorts the list elements on the basis of age.

File: Student.java

```
class Student implements Comparable<Student>{
    int rollno;
    String name;
    int age;
```

```

Student(int rollno,String name,int age){
    this.rollno=rollno;
    this.name=name;
    this.age=age;
}

public int compareTo(Student st){
    if(age==st.age)
        return 0;
    else if(age>st.age)
        return 1;
    else
        return -1;
}
}

```

File: TestSort3.java

```

import java.util.*;
import java.io.*;
public class TestSort3{
    public static void main(String args[]){
        ArrayList<Student> al=new ArrayList<Student>();
        al.add(new Student(101,"Vijay",23));
        al.add(new Student(106,"Ajay",27));
        al.add(new Student(105,"Jai",21));
        Collections.sort(al);
        for(Student st:al){
            System.out.println(st.rollno+" "+st.name+" "+st.age);
        }
    }
}

```

Test it Now

```

Output:105 Jai 21
      101 Vijay 23
      106 Ajay 27

```

Java Comparator interface

Java Comparator interface is used to order the objects of user-defined class.

This interface is found in `java.util` package and contains 2 methods `compare(Object obj1, Object obj2)` and `equals(Object element)`.

It provides multiple sorting sequence i.e. you can sort the elements on the basis of any data member, for example `rollno`, `name`, `age` or anything else.

compare() method

public int compare(Object obj1, Object obj2): compares the first object with second object.

Collections class

Collections class provides static methods for sorting the elements of collection. If collection elements are of `Set` or `Map`, we can use `TreeSet` or `TreeMap`. But we cannot sort the elements of `List`. Collections class provides methods for sorting the elements of `List` type elements also.

Method of Collections class for sorting List elements

public void sort(List list, Comparator c): is used to sort the elements of `List` by the given Comparator.

Java Comparator Example (Non-generic Old Style)

Let's see the example of sorting the elements of `List` on the basis of `age` and `name`. In this example, we have created 4 java classes:

1. `Student.java`
2. `AgeComparator.java`
3. `NameComparator.java`
4. `Simple.java`

Student.java

This class contains three fields `rollno`, `name` and `age` and a parameterized constructor.

1. `class Student{`
2. `int rollno;`
3. `String name;`
4. `int age;`
5. `Student(int rollno, String name, int age){`

```
6. this.rollno=rollno;
7. this.name=name;
8. this.age=age;
9. }
10. }
```

AgeComparator.java

This class defines comparison logic based on the age. If age of first object is greater than the second, we are returning positive value, it can be any one such as 1, 2 , 10 etc. If age of first object is less than the second object, we are returning negative value, it can be any negative value and if age of both objects are equal, we are returning 0.

```
1. import java.util.*;
2. class AgeComparator implements Comparator{
3. public int compare(Object o1, Object o2){
4. Student s1=(Student)o1;
5. Student s2=(Student)o2;
6.
7. if(s1.age==s2.age)
8. return 0;
9. else if(s1.age>s2.age)
10. return 1;
11. else
12. return -1;
13. }
14. }
```

NameComparator.java

This class provides comparison logic based on the name. In such case, we are using the compareTo() method of String class, which internally provides the comparison logic.

```
1. import java.util.*;
2. class NameComparator implements Comparator{
3. public int compare(Object o1, Object o2){
4. Student s1=(Student)o1;
5. Student s2=(Student)o2;
6.
7. return s1.name.compareTo(s2.name);
8. }
9. }
```

Simple.java

In this class, we are printing the objects values by sorting on the basis of name and age.

```
import java.util.*;
import java.io.*;
class Simple{
public static void main(String args[])
{
ArrayList al=new ArrayList();
al.add(new Student(101,"Vijay",23));
al.add(new Student(106,"Ajay",27));
al.add(new Student(105,"Jai",21));
System.out.println("Sorting by Name...");
Collections.sort(al,new NameComparator());
Iterator itr=al.iterator();
while(itr.hasNext()){
Student st=(Student)itr.next();
System.out.println(st.rollno+" "+st.name+" "+st.age);
}

System.out.println("sorting by age...");

Collections.sort(al,new AgeComparator());
Iterator itr2=al.iterator();
while(itr2.hasNext()){
Student st=(Student)itr2.next();
System.out.println(st.rollno+" "+st.name+" "+st.age);
}
}
}
```

*****NETWORKING*****

***Networking:** A Network is an inter connection of computers and it is generally a combination of client and server machines.

A client machine is that machine which Receives information from the network.

A server machine is that machine which sends information on to the network.

To make communication between two different machines on a network they follow a protocol.

A protocol is a standard or a set of Rules that must be followed to make communication possible between the difference machines of a networks.

Ex: http, ftp, tcp/ip, smtp, nntp, pop and etc.

Very machine on a network will be Identify by a u-nick number called as ‘IP Address’.

Port number is a u-nick identification given to very program running on a system.

Ex: 1 to 65536 (Range of port numbers)

Shoket is an end point of communication.

Both the client and server machine Requires of shoket for communication.

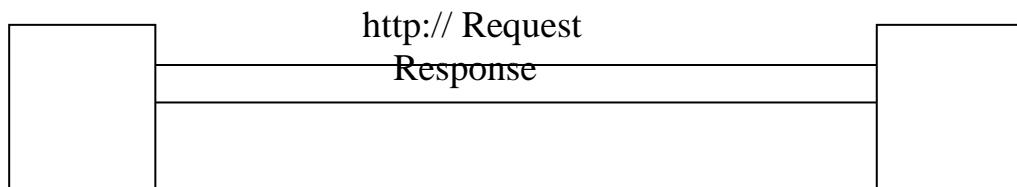
To create a shoket in a client machine to use a shoket class.

To create a shoket in a server machine we use a server shoket class.

Both the shoket and server shoket class are available in “java.net package” (network package).

Shoket

Server Shoket



Client IP Address port NO DNS->Domain Naming Service ex: gmail.com

How to get my machine or local host IP address in Java?

```
import java.net.InetAddress;
```

```
import java.net.UnknownHostException;
```

```
public class MyIpAddress {
```

```
    public static void main(String a[]){
```

```
        try {
```

```
            InetAddress ipAddr =
```

```
            InetAddress.getLocalHost();
```

```
            System.out.println(ipAddr.getHostAddress());
```

```
        } catch (UnknownHostException ex) {
```

```
        ex.printStackTrace();
    }
}
}
```

How to get Host name by IP address in Java?

```
import java.net.InetAddress;
import java.net.UnknownHostException;
```

```
public class MyHostName {
```

```
    public static void main(String a[]){
```

```
        try {
            InetAddress host = InetAddress.getByName("72.167.232.155");
            System.out.println(host.getHostName());
        } catch (UnknownHostException ex) {
            ex.printStackTrace();
        }
    }
}
```

```
import java.net.*;
class Test
{
    public static void main(String[] args)
    {
        InetAddress address = InetAddress.getLocalHost();
        System.out.println(address);
```

```
address = InetAddress.getByName("www.facebook.com");
System.out.println(address);
InetAddress sw[] = InetAddress.getAllByName("www.google.com");
for(int i=0; i< sw.length; i++)
{
    System.out.println(sw[i]);
}
```

Program using URL class

```
import java.net.*;
class Test
{
    public static void main(String[] arg) throws MalformedURLException
    {
        URL hp = New URL("http://www.java.com/index");
        system.out.println(hp.getProtocol[]);
        System.out.println(hp.getFile[]);
    }
}
//program Server
import java.net.*;
import java.io.*;
class Server{
    public static void main(String[] args) throws IOException{
        //creating the server side socket with 222 as the server program port no.
        ServerSocket ss = new ServerSocket(222);
        //waiting for the client and accepting the client side socket
        Socket s = ss.accept();
        //getting the client socketoutputstream
        OutputStream out = s.getOutputStream();
        //connecting the client outputstream to the printstream
        printStream ps = new printStream(out);
        //sending data to the client
        ps.println("hai");
        ps.println("hello h r u?");
        ps.close(); //closing the connection
    }
}
```

```
}

//program is a client side
import java.net.*;
import java.io.*;
class Client{
public static void main(String[] args) throws Exception{
//creating a client side socket by specifying the server machine
//ip address and server program port no.
Socket s = new Socket("localhost",222);
//getting the inputStream associated clientSocket
InputStream in = s.getInputStream();
//connecting theinputstream to the br to read the data line by line
BufferedReader br = new BufferedReader(new InputStreamReader(in));
//reading the data from server and display the data in the client
while(str!=null){
System.out.println(str);
str = br.readLine();
}
br.close(); //closing the connection
}
}
```

To make networking(communication) we need to write tow programs one for the server and the other for the client machine.

This networking program is called also socket programming.

To execute the network programs we required two commands prompts one for the server and other client.

Always the server program is execute the first and wait for the client request.

Otherwise we get a runtime exception is called "connect exception".

Default IP address = "127.0.0.1" (standard machine)

```
//Two way communication between client and server
import java.net.*;
import java.io.*;
class ServerApp{
public static void main(String[] args) throws IOException{
//creating the server side socket with 9999 as the server program port number
ServerSocket ss = new ServerSocket(9999);
//waiting for the client and accessing the client side socket
Socket s = ss.accept();
//stream for reading data from client
InputStream in = s.getInputStream();
```

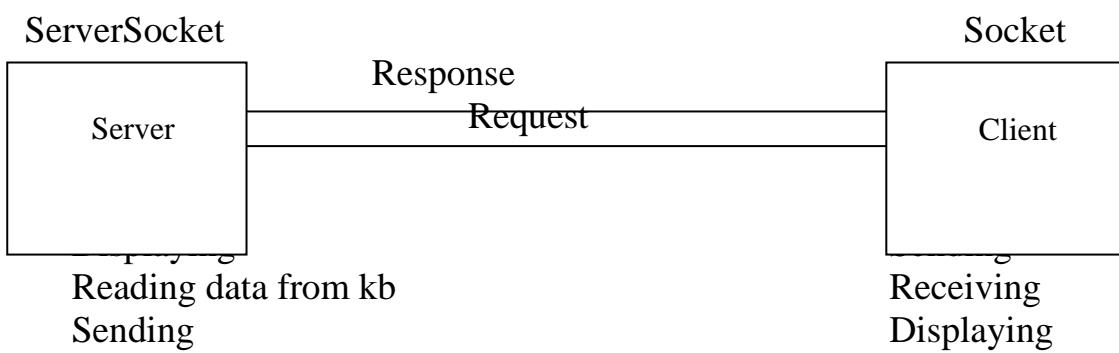
```
BufferedReader br = new BufferedReader(new InputStreamReader(in));
//Stream for reading data from keyboard
BufferedReader br = new BufferedReader
                    (new InputStreamReader(System.in));
//stream for sending data to the client
OutputStream out = s.getOutputStream();
printStream ps = new printStream(out);
//sending and receiving data from client
while(true){
String str, msg;
while((str = br.readLine())!= null){
System.out.println(str);
msg = kb.readLine();
ps.println(msg);
}
//releasing the resources
br.close();
kb.close();
ps.close();
System.exit(0);
} //end of while
} //end of main()
} //end of class
```

```
// Client program
import java.net.*;
import java.io.*;
class ClientApp{
public static void main(String[] args) throws IOException{
//creating a socket in client machine by specifying the server machine
//IP address andserver program port no
Socket s = new Socket("localhost", 9999);
//Stream for reading data from keyboard
BufferedReader kb = new BufferReadered
                    (new InputStream(System.in));
OutputStream out = s.getOutputStream();
printStream ps = new printStream(out);
//Stream for reading data from the server
InputStream in = s.getInputStream();
```

```
BufferedReader br = new BufferedReader  
                    (new InputStreamReader(in));  
//reading and sending data into a server  
String str, msg;  
while(!(str = kb.readLine()).equals("bye")){  
    ps.println(str);  
    msg = br.readLine();  
    System.out.println(msg);  
}  
//reading the resources  
kb.close();  
ps.close();  
br.close();  
}  
}  
}
```

Note: Instead of using local hosts we can use 127.0.0.1, which is default IP address of a standalone system.

Understand diagram



*****THREADS*****

***Single tasking**: Executing a single task at a time is called as single tasking. Here much of the processor time is wasted.

Ex: DOS

***Multi tasking**: Executing multiple tasks at the same time is called as multitasking. Here the processor time is utilized in an optimum way. This multitasking will improve the performance by reducing the response times.

Executing several tasks simultaneously is the concept of multitasking. There are two types of multitasking's.

Ex: windows



***Time Slice**: It is a small amount of processor time given to a process for execution. Multitasking is of two types. They are given below

1. Process based multitasking
2. Thread based multitasking

1. **Process based multitasking**: Executing different processes simultaneously at the same time which are independent of each other and every process contains its own memory. This multitasking is an operating system approach.

Ex: writing java program, down loading s/w.

Listen to music, copying s/w etc.

2. **Thread based multitasking**: Executing different parts of the same process simultaneously at the same time. Where those different parts have common memory, which may be dependent or independent. This multitasking is a programmatic approach.

Ex: Games, web applications

***Multithreading**: Executing multiple threads at the same time is called as multi threading or thread based multitasking.

***Thread**: A separate piece of code which is executed separately is called as thread. Program to get currently executing thread information.

Ex: class ThreadInfo{

```

Public static void main(String[] args){
    Thread t = thread.currentThread();
    System.out.println("Thread:" +t);
}
  
```

T.priority



Output: Thread Info: Thread[main, 5, main]

T.name Tgroupname

Current thread method will provide the information of the currently executing thread and it provides information like thread name, thread priority and the thread group name.

Every java program by default contains one thread called as main thread.

The user thread can be created in two ways.

1. By extending Thread class
2. By implementing Runnable interface

1. By extending Thread class:

*Procedure:

1. Create a class as sub class thread class.

Syntax: class Myclass extends Thread

2. Write the functionality of user thread with in the run method.

Syntax: public void run(){ }

3. Create the object of the class that is extending Thread class

Syntax: Myclass mc = new Myclass();

4. Attach the above created object to the thread class

Syntax: Thread t = new Thread(mc);

5. Execute the user thread by invoking start();

Syntax: t.start();

```
//User Thread(extending)
class ThreadDemo extends Thread{
//functionality of user Thread
Public void run(){
for(int i= 1; i<=10; i++){
System.out.println("user Thread:" +i);
}
}
public static void main(String[] args){
//creating the object
ThreadDemo td = new ThreadDemo();
//attaching the user thread
Thread t = new Thread(td);
//executing the user thread
t.start();
}
}
```

Difference between t.start() and t.run() methods.

- In the case of t.start() a new Thread will be created which is responsible for the execution of run() method.
- But in the case of t.run() no new Thread will be created and run() method will be executed just like a normal method by the main Thread.

2.By implementing Run able interface:***Procedure:**

1.Create a class implementing Run able interface.

Syntax: class Myclass implements Run able

2.Write the functionality of user thread with in the run method.

Syntax: public void run(){ }

3.Create the object of class implements Run able interface.

Syntax: MyClass mc = new MyClass();

 MyClass mc = new Myclass();

4.Attach the above created object to the thread class

Syntax: Thread t = new Thread(mc);

5.Execute the user thread by invoking start().

Syntax: t.start();

```
//user implementing run able interface
class RunnableDemo implements Runnable{
//functionality of user Thread
public void run(){
for(int i = 1; i <= 15; i++){
System.out.println("user Thread:" +i);
}
}
Public static void main(String[] args){
//creating the object
RunnableDemo rd = new RunnableDemo();
//attaching the user thread
Thread t = new Thread(rd);
//executing the user thread
t.start();
}
}
```

*****Different b/w extends Thread & implements Runnable:**

When we create thread by extending thread class we do not have a chance to extend from another class.

When we create thread by implementing Runnable we have a chance to extends from another class.

Note: It is recommended to use implements Runnable to create a ‘thread’.

// creating a user multiple Threads acting on multiple objects.

```
class MultiThread implements Runnable{
```

```
String name;
```

```
MultiThread(String name){
```

```
this.name = name;
```

```
}
```

```
Public void run(){
```

```
for(int i = 1; i <= 10; i++){
```

```
System.out.println("name value:" + i );
```

```
}
```

```
}
```

```
}
```

```
// un other program
```

```
class MultiThreadDemo{
```

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

```
MultiThread mt1 = new MultiThread("Thread1");
```

```
MultiThread mt2 = new MultiThread("Thread2");
```

```
Thread t1 = new Thread(mt1);
```

```
Thread t2 = new Thread(mt2);
```

```
t1.start();
```

```
t2.strat();
```

```
for(int i = 1; i <= 10; i++){
```

```
System.out.println("main value:" + i);
```

```
}
```

```
}
```

```
}
```

```
// Program creating multiple Threads
```

```
class College implements Runnable{
```

```
int seats'
```

```
college(int seats){
```

```
this.seats = seats;
```

```
}
```

```
public void run(){
```

```
Thread t = Thread.currentThread();
```

```
String name = t.getName();
```

```

System.out.println(name + "No.of seats before allotment" + seats);
if(seats > 0){
try{
Thread.sleep(2000);
System.out.println("seat allotted to :" + name);
Seats = seats -1;
}
catch(InterruptedException ie){
ie.printStackTrace();
}
}
else{
System.out.println("seat not allotted to: " + name); }
System.out.println(name + "No.of seats after allotment:" + seats); }
}

class Allotment{
public static void main(String[] args){
College c = new College(60);
Thread t1 = new Thread(c);
Thread t2 = new Thread(c);
t1.setName("student1");
t2.setName("student2");
t1.start();
t2.start();
}
}

```

When multiple threads are acting on the same object there is a chance of data inconsistency problem occurring in the application.

Data inconsistency problem occurring when one of the thread is updating the value when other thread is trying to read the value at same time.

To avoid the data inconsistency problem we have to synchronize the threads that are the acting on the same object.

****Thread Synchronization:** When multiple threads wants to access the object at the same time avoiding multiple threads to access the same and giving access to one of the thread is called as thread synchronization. Thread synchronization can be done into two ways.

1. Synchronized Block
2. synchronized Method

1. Synchronized Block: Synchronizing a group of statements or part of a code is called as Synchronized Block.

Syntax: Synchronized(object){
 Statements; }

2. **Synchronized Method:** when we want to Synchronized all the statements in a method we go for synchronized method.

Syntax: Synchronized returnType methodName(){
 Statements;
}

Note: In the previous program multiple threads acting on the same object leading to data inconsistency, to avoid the data inconsistency problem. We have to synchronize the threads acting on the same object.

Ex: public Synchronized void run(){
 Same code previous programs;
}

When multiple threads are acting on synchronized objects then there is chance of other problems like ‘Deadlock’ occurring in the application.

***Deadlock:** When a thread holds a resource and waits for another resource to be realized by second thread, the second thread holding a resource and waiting for realized by first thread. Both the threads will be waiting indefinitely and they never execute this switching is called as “Deadlock”.

In java there is no mechanism to avoid deadlock situation, it is the responsibility of the programmer to write proper logic to avoid deadlock situation.

***Creation of a Thread:**

Syntax: Thread t = new Thread();

The above syntax will create a thread having default names. The default names will be Thread – 0, Thread – 1, Thread – 2,

Syntax: Thread t = new Thread(String name);

The above syntax will create a thread with the specified name.

Syntax: Thread t = new Thread(Object obj);

The above syntax will create a thread which is attached to the specified object.

Syntax: Thread t = new Thread(Object obj, String name);

The above syntax will create a thread with the specified name and attached to the specified object.

***Methods of Thread class:**

1. **Current_Thread():** This method is used to provide the information of currently executing Thread.

2. **Start():** This method is used to execute the user thread, that is used to execute the logic of Run method.

3. **Sleep(milli seconds):** This method is used to suspend the execution of a thread for amount of time specified. This method throws an exception called interrupted exception which must be handled.

4. **getName():** This method returns the name of the thread.

5. **SetName(String name):** This method used to assigned a name to a thread.
6. **getpriority():** This method returns the priority of a thread.
7. **Set priority():** This method is used to change the priority of a thread. When we want to change the thread priority it is always recommended to take the support of the constant declared in the thread class.

* MIN_PRIORITY
 *NORM_PRIORITY
 *MAX_PRIORITY

Ex: t.SetPriority(8);

t.SetPriority(Thread.MAX_PRIORITY-2); //recommended)

8. **iSAlive():** This method returns true if the thread is true. Otherwise is false. A thread is said to be Alive as long as Thread is executing “run()”.
9. **join():** This method is used to make a thread wait until another thread dice.

*****Methods of object class related to Threads:**

- 1.**wait():** This method is used to suspend the execution of a thread until it receives a notification.
- 2.**notify():** This method is used to send a notification to one of the waiting threads.
- 3.**notifyAll():** This method is used to send a notification to All the waiting threads.

Note: The above three methods are used for making communication among the threads.

***Types of Threads:**

1. **Orphan Thread:** A thread which is executed without the help of a parent is called as Orphan Threads. Orphan threads can be created ‘join()’.
2. **Helper Threads:** when multiple threads having a same priority are competing for executing, allowing one of those threads to execute. Depending upon the requirement and the remain threads are called as helper threads. Helper threads give chance for other threads to execute.
3. **Selfish Thread:** A thread which takes lot of resources are execute for longer time periods or until completion are called as Selfish Threads.
4. **Starving Thread:** A thread that is waiting for longer time periods are called as Starving Threads.
5. **Green Thread:**(JVM level treads) These threads are also called as JVM level threads. These threads are used for allocating resource to the user thread. Here the allocation of the resources may not be efficient.
6. **Native threads:** These threads are also called as ‘operating System level threads. These threads are responsible for allocating resource to user threads. Here the allocating of resources of resource is efficient.
7. **Deamon Thread:** These threads are also called as background threads. These threads will execute where no other threads are under execution.

//program for inter thread communication

```
class Showroom{  
int value;  
boolean flag = true;  
public Synchronized void produce(int i){  
if(flag == true){  
value = i;  
System.out.println("produce value:" +i);  
notify();  
flag = flase;  
}  
try{  
wait();  
}  
catch(InterruptedException ie){  
}  
}  
Public Synchronized int consume(){  
if(flag == true){  
try{  
wait();  
}  
catch(InterruptedException ie){  
ie.printStackTrace();  
}  
}  
notify();  
flag = true;  
return value;  
}  
}//show Room class  
class producer extends Thread{  
ShowRoom s;  
Producer(ShowRoom s){  
this.s = s;  
}  
Public void run(){  
int i = 1;  
while(true){  
s.produce(i);  
i = i + 1;  
try{  
}
```

```
Thread.Sleep(2000);
    }
catch(InterruptedException ie){
System.out.println(ie);
    }
} //while
} //run
} //producer
class Consumer extends Thread{
ShowRoom s;
Consumer(ShowRoom s){
this.s = s;
}
Public void run(){
while(true){
int x = s.consume();
System.out.println("Consumed value:" + x);
try{
Thread.sleep(2000);
}
catch(InterruptedException ie){
System.out.println(ie);
    }
} //while
} //run
} // consumer
class producerConsumer{
public static void main(String[] args){
ShowRoom s = new ShowRoom();
Producer p = new Producer();
Consumer c = new Consumer(s);
Thread t1 = new Thread(p);
Thread t2 = new Thread(c);
t1.start();
t2.start();
}
}
```

Note: The wait(), the notify() and notifyAll() must be called with in a Synchronized block otherwise we get a run time error called Illegal monitor State Exception.

// Another Example

```
class MyThread extends Thread{
```

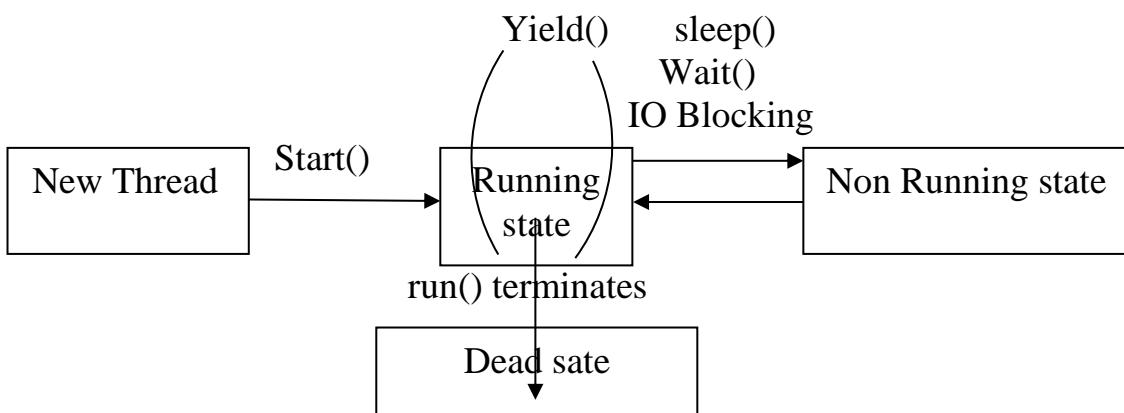
```

static int total = 0;
public Synchronized void run(){
System.out.println("user thread started calculation");
for(int i = 1; i <= 10; i++){
total = total +i;
}
System.out.println("user thread sending notification");
notifyAll();
System.out.println("user total = " + total);
}
Public static void main(String[] args) throws InterruptedException{
MyThread mt = new MyThread();
Thread t = new Thread(mt);
System.out.println("main thread calling user thread");
t.start();
Synchronized(mt){
mt.wait();
}
System.out.println("main thread got notification");
System.out.println("main Total = " + mt.total);
}
}

```

The `wait()`, `notify()` and `notifyAll()` are available in `Object` class. So that we can use those methods directly in our logic to make communication without the reference of thread class.

*****THREAD LIFE CYCLE*****



Localization:

Internationalization is also abbreviated as I18N because there are total 18 characters between the first letter 'I' and the last letter 'N'.

Internationalization is a mechanism to create such an application that can be adapted to different languages and regions.

Internationalization is one of the powerful concept of java if you are developing an application and want to display messages, currencies, date, time etc. according to the specific region or language.

Localization is also abbreviated as I10N because there are total 10 characters between the first letter 'L' and last letter 'N'. Localization is the mechanism to create such an application that can be adapted to a specific language and region by adding locale-specific text and component.

```
import java.util.Locale;
```

```
public class Test {  
    public static void main(String[] args) {  
        Locale locale=Locale.getDefault();  
        //System.out.println(locale.getLanguage()+"\t"+locale.getCountry());  
        Locale locales[]=Locale.getAvailableLocales();  
        for (int i = 0; i < locales.length; i++) {  
            System.out.println(locales[i].getLanguage()+"\t"+locales[i].getCountry());  
        }  
    }  
}
```

program:

MessageBundle_en_US:

```
# To change this license header, choose License Headers in Project Properties.  
# To change this template file, choose Tools | Templates  
# and open the template in the editor.  
name= Enter UserName  
password= Enter Password
```

MessageBundle_fr_FR:

```
# To change this license header, choose License Headers in Project Properties.  
# To change this template file, choose Tools | Templates  
# and open the template in the editor.  
name=entrez le nom d'utilisateur  
password=Entrer le mot de passe
```

```
import java.util.Locale;  
import java.util.ResourceBundle;  
import java.util.Scanner;  
  
public class RegionTest {  
    public static void main(String[] args) {  
        Locale englocale=new Locale("en","US");  
        Locale frlocale=new Locale("fr","FR");  
        ResourceBundle rd=ResourceBundle.getBundle("MessageBundle",frlocale);  
        String name=rd.getString("name");  
        String password=rd.getString("password");  
        Scanner sc=new Scanner(System.in);  
        System.out.println(name);  
        String username=sc.next();  
        System.out.println(password);  
        String userpassword=sc.next();  
        if(username.equals("NIT")&&userpassword.equals("NIT"))  
        {  
            System.out.println("success");  
        }  
        else  
        {  
            System.out.println("Failure");  
        }  
    }  
}
```

******AWT*****

***AWT**: (Abstract Windowing Toolkit)

When a user want to interact with an application, the user has to provide some information to the application and it can be done in two ways.

1. **Character user Interface(CUI)**: This Interface is use to interact with the application by typing some characters.

Ex: DOS (character user Interface)

This interface is not user friendly because the user has to type all the commands and the user has to remember all the commands.

2. **Graphical user Interface(GUI)**: This Interface will interact with the application in the help of some graphics like menu's, icons and images etc.

Ex: window xp, windows 7(os)

This Interface is user friendly because it prompts the user by providing the options (or) menus.

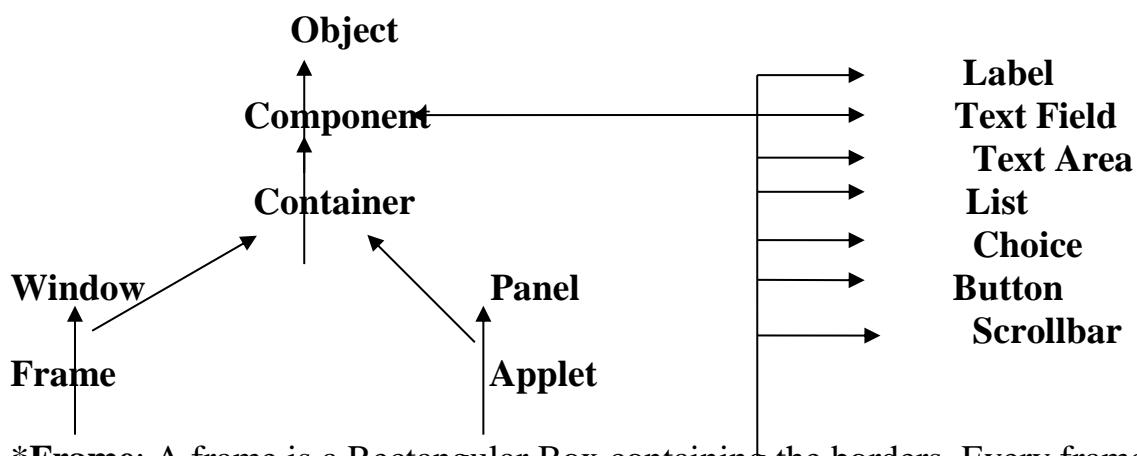
To develop the GUI to use AWT. It's Transform abstract windowing toolkit. The set of classes & Interfaces which are required to develop. GUI components together are called as 'Toolkit'. The GUI components will be used to design GUI programs.

Writing a program to display the created GUI components on the windows is called as 'windowing'. To display the components on the windows we need to support of graphics available in the operating system for a developer there is no direct interaction with the graphics and hence graphics is 'Abstract' to the developer.

Every GUI component will have a corresponding 'PEER' class which is responsible to interact with the graphics of the 'operating system'.



Collection of the all peer class is called as a 'PEER SERVICE'.

AWT HIERARCHY:

***Frame**: A frame is a Rectangular Box containing the borders. Every frame will by default contain a title, minimize, maximize and close Buttons by default the frame is invisible (false). By default the size of frame OXO (zero by zero) pixels. The frame can be created in two ways.

Create the object of Frame class directly.

Ex: **Frame f = new Frame();**

Create the object of any class that extends Frame class.

Ex: **class MyFrame extends Frame**

MyFrame mf = new MyFrame();

//Program to create a frame using the first technique

import java.awt.*;

public static void main(String[] args){

//Creation of frame object directly

Frame f = new Frame();

//making the frame visible

f.setVisible(true);

//setting the size of the frame

f.setSize(300,300);

}

}

//program to create a frame using the second technique

import java.awt.*;

class FrameDemo extends Frame{

public static void main(String[] args){

//creation of frame object directly

```

FrameDemo fd = new FrameDemo();
//making the frame visible
fd.setVisible(true);
//setting the size of the frame
fd.setSize(500,200);
}
}

```

***Event:** An event is used for making communication or interacting with the ‘GUI’. Events will be generated or triggered automatically based on the user operation.

Ex: Left click, Right click and Double click, typing some text, selecting some values etc. will generate the events automatically.

An event is an object which contains the information of the event that is generated and also contains the Information of the component that has generated the Event.

To perform any operation the component has to listen to the event that is generated. But the component can not listen to the events that are generated. So we take the help of the listeners which are interfaces which can listen to the events that are generated. After the listener listens to the generated event, it delegates (passes) the event information to one of the method available in that listener. This process is called as “Event Delegation Model”. They are different listeners which can listen to their corresponding events.

*procedure to use a listener in an application:

1. Choose a listener (interface) appropriate to the application requirement.
2. Once the listener is selected we have to provide the implementation to all the methods in the selected listener (null).

Ex:

```

import java.awt.*;
import java.awt.event.*;
class FrameDemo extends Frame Implements WindowListener{
FrameDemo(){
//making the frame visible
setVisible(true);
//setting the size of the frame
setSize(300,300);
//setting the title of the frame
setTitle("Window Listener");
//adding the listener to the frame
addWindowListener(this);  }
public static void main(String[] args){
//creation of frame object
FrameDemo fd = new FrameDemo();
}
Public void windowOpened(WindowEvent we){ ↗}

```

```
Public void windowClosing(WindowEvent we){  
System.exit(0);  
}
```

```
Public void windowClosed(WindowEvent we){ }
```

```
Public void windowActivated(WindowEvent we){ }
```

```
Public void windowDeactivated(WindowEvent we){ }
```

```
Public void windowIconified(WindowEvent we){ }
```

```
Public void windowDeiconified(WindowEvent we){ }
```

All the null implementation method

```
//end of the class
```

***WindowAdapter:** This calss is an implementation class of WindowListener it contains All the 7 mehods of WindowListener but available as null implementations.

WindowAdaper can we use to over ride the methods that we require and the remaining methods need not the implemented.

```
import java.awt.*;  
import java.awt.event.*;  
class FrameDemo extends Frame{  
FrameDemo(){  
this.setVisible(true);  
this.setSize(300,500);  
this.setTitle("windowAdapter");  
addWindowListener(new MyFrame());  
}
```

```
Public static void main(String[] args){  
FrameDemo fd = new FrameDemo();  
}  
}  
class MyFrame extends WindowAdapter{  
public void windowClosing(WindowEvent we){  
System.exit(0);  
}}
```

***Inner class:** If a class declared inside another class then it is called Inner class.

***Anonymous inner class:** An inner class which does not contains any name is called as Anonymous inner class.

```
import java.awt.*;  
import java.awt.event.*;  
class FrameDemo extends Frame{  
FrameDemo(){  
this.setVisible(true);  
this.setSize(300,300);
```

Creating the object of source calss which has no name and extending windowAdapter

```
this.setTitle("Anonymous inner class");
addWindowListener(new WindowAdapter(){
public void windowClosing(WindowEvent ae){
System.exit(0);
}
});
}

Public static void main(String[] args){
FrameDemo fd = new FrameDemo();
}
}
```



***DrawString Method**: This method used to display a message on the frame.

Syntax: drawString(String,x,y);

The string that is specified will be displayed (x,y) location of the frame.

The drawstring method belong to graphics class and we can get the reference of graphics class by using paintmethod.

Syntax: public void paint(Graphics g)

The paint() will be invoke automatically when the frame is created and loaded.

//program will be display on

```
import java.awt.*;
import java.awt.event.*;
class TextFrame extends Frame{
TextFrame(){
SetVisible(true);
SetSize(300,300);
SetTitle("message frame");
addWindowListener(new WindowAdapter(){
public void windowClosing(WindowEvent we){
System.exit(0);
}
});
}

public static void main(String[] args){
new TextFrame();
}

public void paint(Graphics g){
//creation a color
Color c = new Color(200,0,0);
//setting the color
g.setColor(c);
//creating a font
```

```
Font f = new Font("arial", Font.BOLD,34);
//setting the font
g.setFont(f);
//displaying a message on the frame
g.drawString("Hello students",100,100);
    }
}
```

***Button**: This component can be used to perform some operation when the user clicks on a button.

***Creation of button**: Button b = new Button(String label);

```
import java.awt.*;
import java.awt.event.*;
class ButtonDemo extends Frame{
//declaring the components
Button b1,b2;
ButtonDemo(){
//creating the components
b1 = new Button("OK");
b2 = new Button("CANCEL");
//setting the layout to null layout
this.setLayout(null);
//setting the boundaries for the components
b1.setBounds(100,100,80,40); //(x, y, w, h)
b2.setBounds(200,100,80,40); //(x, y, w, h)
//adding the component to the frame
this.add(b1);
this.add(b2);
this.setVisible(true);
this.setSize(300,300);
this.setTitle("button");
this.addWindowListener(new WindowAdapter(){
public void windowClosing(WindowEvent we){
System.exit(0);
}
});
}
Public static void main(String[] args){
New ButtonDemo();
}
}
```

***Label**: This component is used to display a message on the frame. This component will be generally used along with other components.

Creation of Label: Label l = new Label(String);

***TextField**: This component will allow the user to enter some text.

Creation of TextField: TextField tf = new TextField(size);

```
import java.awt.*;
import java.awt.event.*;
class LoginDemo extends Frame{
Label userl, pwdl;
TextField usertf, pwdtf;
LoginDemo(){
userl = new Label("user_Name");
Pwdl = new Label("password");
Usertf = new TextField(20);
Pwdtf = new TextField(20);
setLayout(null);
userl.setBounds(100,100,80,40);
usertf.setBounds(200,100,80,30);
pwdl.setBounds(100,200,80,30);
pwdtf.setBounds(200,200,80,30);
this.add(userl);
this.add(usertf);
this.add(pwdl);
this.add(pwdl);
this.setVisible(true);
this.setSize(400,500);
this.setTitle("TextField Label");
this.addWindowListener(new WindowAdapter(){
public void windowClosing(WindowEvent we){
System.exit(0);
}
});
}
public static void main(String[] args){
new LoginDemo();
}
```

***Checkbox**: This component allows the user select any number of options from an even group of options.

Creation of checkbox: Checkbox cb = new Checkbox(Label);

***RadioButton**: This component will allow the user to select any one option from group of options. To place the options under single group we are support to use class called “CheckboxGroup”.

Creation of RadioButton: CheckboxGroup cbg = new CheckboxGroup();
Checkbox rb = new Checkbox(Label, cbg, boolean);

Note: To create the RadioButton we use the same Checkbox class.

***TextArea**: This component will allow the user to write the text in multiple lines.

Creation of TextArea: TextArea ta = new TextArea(rows,cols);

```
import java.awt.*;  
import java.awt.event.*;  
class SelectionFrame extends Frame{  
    Checkbox cb1, cb2, cb3, rb1, rb2, rb3;  
    CheckboxGroup cbg;  
    TextArea ta;  
    SelectionFrame(){  
        cb1 = new Checkbox("programming");  
        cb2 = new Checkbox("Reading");  
        cb3 = new Checkbox("Browsing");  
        cbg = new CheckboxGroup();  
        rb1 = new Checkbox("Btech",cbg,false);  
        rb2 = new Checkbox("BE",cbg,false);  
        rb3 = new Checkbox("MCA",cbg,false);  
        ta = new TextArea(6,20);  
        SetLayout(new FlowLayout());  
        add(cb1);  
        add(cb2);  
        add(cb3);  
        add(rb1);  
        add(rb2);  
        add(rb3);  
        add(ta);  
        SetVisible(true);  
        SetSize(400,500);  
        SetTitle("SelectionFrame");  
        addWindowListener(new WindowAdapter(){  
            public void windowClosing(WindowEvent we){  
                System.exit(0);  
            }  
        });
```

```
}
```

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

```
new SelectionFrame();
```

```
}
```

```
}
```

***Choice**: This component will display group of times as a drop down menu from which a user can select only one item.

Creation of choice: Choice ch = new Choice();

To add the items to the choice we use add()

```
ch.add(item);
```

***List**: This component will display a group of items as a scrolling menu from which the user can select any no.of items.

Creation of List: List l = new List(int, boolean);

We can add the items to the List by using add()

```
l.add(item);
```

```
import java.awt.*;
```

```
import java.awt.event.*;
```

```
class SelectionFrame extends Frame{
```

```
Choice ch;
```

```
List l;
```

```
SelectionFrame(){
```

```
ch = new Choice();
```

```
ch.add("Hyderabad");
```

```
ch.add("pune");
```

```
ch.add("Chennai");
```

```
ch.add("Noida");
```

```
ch.add("Mysore");
```

```
ch.add("Bangalore");
```

```
l=new List(5,true);
```

```
l.add("Hyderabad");
```

```
l.add("Pune");
```

```
l.add("Chennai");
```

```
l.add("Noida");
```

```
l.add("Mysore");
```

```
l.add("Bangalore");
```

```
setLayout(new FlowLayout());
```

```
add(ch);
```

```
add(l);
```

```
setVisible(true);
```

```
setSize(500,500);
```

```
setTitle("ChoiceList");
```

```

addWindowListener(new WindowAdapter(){
public void windowClosing(WindowEvent we){
System.exit(0);
}
});
}

public static void main(String[] args){
new SelectionFrame();
}
}

```

***Listeners**: Listeners are the interfaces which can listen to the Events that are generated.

<u>Listener</u>	<u>Event</u>	<u>Methods</u>
-----------------	--------------	----------------

- 1.ActionListener - ActionEvent - public void actionPerformed(ActionEvent)
- 2.ItemListener - ItemEvent - public void itemStateChanged(ItemEvent)
- 3.FocusListener - FocusEvent – public void focusGained/Lost(FocusEvent)
- 4.MouseMotionListener-MouseEvent-p.v mouseDragged/Moved(MouseEvent)
- 5.AdjustmentListener - AdjustmentEvent - p.v adjustmentValueChange(AdjEve)

To register the Listeners with the Components we have to use addxxxListener()

Ex: addWindowListener();
 addActionListener();

To unregister a Listener with a Components we use removexxxListener()

Ex: removeWindowListener();
 removeActionListener();

//program to use Action Listener an a textFields public.

```

Public class ListenerDemo Extends Frame implements ActionListener{
Label Userl, Pwdl;
TextField usertf, pwdf;
ListenerDemo(){
Userl = new Label("username");
Pwdl = new Label("password");
Usertf = new TextField(20);
Pwdf = new TextField(30);
SetLayout(null);
Userl.setBounds(100,100,80,30);
Usertf.setBounds(200,100,80,30);
Pwdl.setBounds(100,200,80,30);
Pwdf.setBounds(200,200,80,30);
add(Pwdf);
add(Userl);
add(pwdl);

```

```
add(Usertf);
Usertf.addActionListener(this);
Pwdtf.addActionListener(this);
setVisible(true);
setSize(500,500);
setTitle("MyFrame");
addWindowListener(new WindowAdapter(){
public void windowClosing(WindowEvent we){
System.exit(0);
}
});
}

Public static void main(String[] ar){
new ListenerDemo();
}

Public void actionPerformed(ActionEvent ae){
String name = Usertf.getText();
String pass = Pwdtf.getText();
Graphics g = this.getGraphics();
}

}

//program Login Applictaion
*ActionListener on Button Component:
import java.awt.*;
import java.awt.event.*;
class LoginApp extends Frame implements ActionListener{
Label ul, pl;
TextField utf, ptf;
Button logb;
LoginApp(){
ul = new Label("username");
pl = new Label("password");
utf = new TextField(30);
ptf = new TextField(30);
logb = new Button("Login");
ptf.setEchoChar('*');
this.setLayout(null);
ul.setBounds(100,100,90,30);
utf.setBounds(200,100,90,30);
pl.setBounds(100,150,90,30);
ptf.setBounds(200,150,90,30);
```

```
logb.setBounds(150,200,90,30);
this.add(ul);
this.add(utf);
this.add(pl);
this.add(ptf);
this.add(logb);
logb.addActionListener(this);
this.setVisible(true);
this.setSize(300,300);

this.setSize("Listener");

this.addWindowListener(new WindowAdapter(){
    public void windowClosing(WindowEvent we){
        System.exit(0);
    }
});

}

Public void actionPerformed(ActionEvent ae){
    if(ae.getSource().equals(logb)){ //cancel Button = canb
        String name = utf.getText();
        String pwd = ptf.getText();
        if(name.equals("i netsolv") & pwd.equals("students")){
            new MessageFrame("valid user");
        }
        else{
            new MessageFrame("Invalid user");
        }
    }
}
```

```
}

Public static void main(String[] args){

new LoginApp();

}

}

class MessageFrame extends Frame{

String str;

MessageFrame(String str){

this.str = str;

SetVisible(true);

SetSize(200,200);

SetTitle("Message");

SetBackground(color.yellow);

SetForeground(color.red);

}

Public void paint(Graphics g){

g.drawString(str,100,100);

}

}

//program to implement ItemListener on checkbox & RadioButton

import java.awt.*;

import java.awt.event.*;

class SelectionFrame extends Frame implements ItemListener{

Checkbox jcb,scb,ocb,mrb,frb;
```

```
CheckboxGroup cbg;  
SelectionFrame;  
jcb = new Checkbox("Java");  
scb = new Checkbox("Scjp");  
ocb = new Checkbox("Oracle");  
cbg = new CheckboxGroup();  
mrb = new Checkbox("Male",cbg,true);  
frb = new Checkbox("Female",cbg,false);  
this.setLayout(new FlowLayout());  
this.add(jcb);  
this.add(scb);  
this.add(ocb);  
this.add(mrb);  
this.add(frb);  
jcb.addItemListener(this);  
scb.addItemListener(this);  
ocb.addItemListener(this);  
mrb.addItemListener(this);  
frb.addItemListener(this);  
this.setVisible(true);  
this.setSize(400,400);  
this.setTitle("SelectionFrame");  
this.addWindowListener(new WindowAdapter(){  
public void windowClosing(WindowEvent we){
```

```
System.exit(0);  
    }  
});  
  
}  
  
public void itemStateChanged(ItemEvent ie){  
repaint();  
}  
  
Public void paint(Graphics g){  
String str = “Course Selected:”;  
if(jcb.getState()){  
str += “scjp”;  
}  
if(ocb.getState()){  
str += “Oracle”;  
}  
g.drawString(str,100,200);  
String msg = “Gender:”;  
msg += cbg.getSelectedCheckbox().getLabel();  
g.drawString(msg,100,250);  
}  
  
Public static void main(String[] args){  
new SelectionFrame();  
}  
}
```

```
//program to implement ItemListener on choice  
import java.awt.*;  
import java.awt.event.*;  
class SelectionFrame extends Frame implements ItemListener{  
    Choice ch;  
    TextArea ta;  
    Label l;  
    SelectionFrame(){  
        ch = new Choice();  
        ch.add("BE");  
        ch.add("BTech");  
        ch.add("MCA");  
        ch.add("MBA");  
        ta = new TextArea(5,30);  
        l = new Label("Qualification");  
        SetLayout(new FlowLayout());  
        add(l);  
        add(ch);  
        add(ta);  
        ch.addItemListener(this);  
        this.setVisible(true);  
        this.setSize(500,500);  
        this.setTitle("SelectionFrame");  
        this.addWindowListener(new WindowAdapter(){  
            public void windowClosing(WindowEvent we){
```

```

        System.exit(0);
    }
}
}

public void itemStateChanged(ItemEvent ie){
String option = ch.getSelectedItem();
ta.setText("Qualification Selected:" + option);
}

public static void main(String[] args){
new SelectionFrame();
}
}

```

***Listeners and Listener Methods:**

<u>Componet</u>	<u>Listener</u>	<u>Listener methods</u>
1.Button	ActionListener	public void actionPerformed(ActionEvent e)
2.Checkbox	ItemListener	public void itemStateChanged(ItemEvent e)
3.CheckboxGroup	ItemListener	public void itemStateChanged(ItemEvent e)
4.TextField	ActionListener	public void actionPerformed(ActionEvent ae)
	FocusListener	public void focusGained(FocusEvent fe) Public void focusLost(FocusEvent fe)
5.TextArea	ActionListener	public void actionPerformed(ActionEvent ae)
	FocusListener	public void focusGained(FocusEvent fe) Public void focusLost(FocusEvent fe)
6.Choice	ActionListener	public void actionPerformed(ActionEvent ae)
	ItemListener	public void itemStateChanged(ItemEvent ie)
7.List	ActionListener	public void actionPerformed(ActionEvent ae)
	ItemListener	public void itemStateChanged(ItemEvent ie)
8.Scrollbar	AdjustmentListener	p.v adjustmentValueChange(AdjEvent ae)
	MouseMotionListener	p.v mouseDragged(MouseEvent me) Public void mouseMoved(MouseEvent me)
9.Frame	WindowListener	Public void windowActivated(WindowEvent we) Public void windowClosed(WindowEvent we) Public void windowClosing(WindowEvent we) Public void windowDeactivated(WindowEvent we) Public void windowDeiconified(WindowEvent we) Public void windowIconified(WindowEvent we) Public void windowOpened(WindowEvent we)
10.Keyboard	KeyListener	p.v keyPressed/Released/Type(KeyEvent ke)
11.Label	Nolistener	is needed

*****JFC-SWING*****

***Swing:** Swing is use to develop a better efficient GUI.

The swing components are part of JFC (java foundation classes) which are developed in java. The swing components are called as light weight components which will improve the performance of the application because the amount of resources required is very minimum. They are no peer classes for the swing components to interact with the operating system. Swing component supports pluggable look and feel using which the component can be presented in various flat forms having same behavior.

Note: swing is not a replacement to AWT. But it is an extension.

The empty area that is available in a container is which is used for displaying the components is called as window pane.

The window pane internally is divided into multiple panes.

***Glass pane:** This is the first pane closer to the window (screen) and it is used for displaying foreground components.

***Content pane:** This pane is available behind Glass pane and it is used for displaying individual components.

***Layered pane:** This pane is available behind the content pane and it is used to display group of components.

***Root pane:** This pane is available behind the Layered pane and it is used to display background components.

Note: All the four panes are placed on top of one another and they are transparent. All the swing components are available in javax.swing package.

*Creation of JFrame

```
import javax.swing.*;
class JFrameDemo{
public static void main(String[] args){
JFrame jf = new JFrame();
jf.setVisible(true);
jf.setSize(400,500);
jf.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
}
//JLabel
import javax.swing.*;
import java.awt.*;
class JFrameDemo extends JFrame{
JLabel jl;
JFrameDemo(){
jl = new Label("Good Morning");
Container c = this.getContentPane();
c.setLayout(new FlowLayout());
c.setBackground(color.black);
Font f = new Font("arial",Font.BOLD,34);
```

```
jl.setFont(f);
jl.setForeground(Color.white);
c.add(jl);
this.setVisible(true);
this.setSize(400,400);
this.setTitle("Label");
this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
}
Public Static void main(String[] args){
new JFrameDemo();
}
}
public static void main(String[] args){
new JFrameDemo();
}
}
```

***JRadioButton**: This component allows the user to select only one item from a group items.

Creation of JRadioButton:

```
JRadioButton jrb = new JRadioButton();
JRadioButton jrb = new JRadioButton(Label);
JRadioButton jrb = new JRadioButton(Label,boolean);
```

***ButtonGroup**: This class is used to place multiple RadioButton into a single group. So that the user can select only one value from that group.

Creation of ButtonGroup:

```
ButtonGroup bg = new ButtonGroup();
We can add RadioButtons to the ButtonGroup by using add method.
bg. Add(jrb);
```

***CheckBox**: This component allows the user to select multiple item from a group of items.

Creation of JCheckBox:

```
JCbeckBox jcb = new JCheckBox();
JCbeckBox jcb = new JCheckBox(Label);
JCbeckBox jcb = new JCheckBox(Label,boolean);
```

***JTextField**: This component allows the user to type some text in a single line.

Creation of JTextField:

```
JTextField jtf = new JTextField(size);
```

***JTextArea**: This component allows the user to type the text in multiple lines.

Creation of JTextArea:

```
JTextArea jta = new JTextArea(rows,cols);

import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
class Example extends JFrame implements ActionListener{
JRadioButton eng,doc;
ButtonGroup bg;
JTextField jtf;
JCheckBox bcb,ccb,acb;
JTextArea jta;
Example(){
eng = new JRadioButton("Engineer");
doc = new JRadioButton("Doctor");
bg = new ButtonGroup();
bg.add(eng);
bg.add(doc);
jtf = new JTextField(20);
bcb = new JCheckBox("Bike");
ccb = new JCheckBox("car");
acb = new Jcheckbox("aeroplane");
jta = new JTextArea(3,20);
Container c = this.getContentPane();
c.setLayout(new FlowLayout());
//Registering the listeners with the components
eng.addActionListener(this);
doc.addActionListener(this);
bcd.addActionListener(this);
ccb.addActionListener(this);
acb.addActionListener(this);
c.add(eng);
c.add(doc);
c.add(jft);
c.add(bcd);
c.add(ccb);
c.add(acb);
c.add(jta);
this.setVisible(true);
this.setSize(500,500);
this.setTitle("Selection example");
this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
```

```

        }
Public void actionPerformed(ActionEvent xyz){
if(xyz.getSource() == eng){
jtf.setText("you are an Engineer");
}
if(xyz.getSource() == doc){
jtf.setText("you are an Doctor");
}
String str = " ";
if(bcb.isSelected()){
str += "Bike\n";
}
if(bcb.isSelected()){
str += "car\n";
}
if(acb.isSelected()){
str += "Aeroplane";
}
Jta.setText(str);
}
public static void main(String[] args){
new Example();
}
}

```

***JTable**: This component is used to display the data in the form of rows and columns.

***Creation of JTable**: JTable jt = new JTable(rowData, ColumnNames);

The row data represents a two dimensional array and the columnNames represents a single dimensional array.

***Methods of JTable**:

1. **getRowCount()**: This method returns the count of the number of rows available in the table.
2. **getColumnCount()**: This method returns the count of the number of columns available in the table.
3. **getSelectedRow**: This method returns the index of the row that is selected. It returns -1 when no row is selected.
4. **getSelectedRows**: This method returns the indexes of the rows that are selected.
5. **getSelectedRowCount()**: This method returns the count number of rows that are selected.
6. **getSelectedColumn()**: Method returns the index of the column that is selected. It returns -1 if no column is selected.
7. **getSelectedColumn()**: Returns the indexes of the columns that are selected.

8. **getSelectedColumnCount()**: Returns the number of columns that are the Selected.
9. **getValueAt(row,column)**: This method returns a value. That is available in the specified location.
10. **getJTableHeader()**: This method returns the heading of the table.

```

import javax.swing.*;
import javax.swing.table.*;
import java.awt.*;
class JTableDemo extends JFrame{
JTable jt;
JTableDemo(){
String[][] data = { {"abcd","java","70"}, {"defg", "orcle", "80"}, {"xyz",
".net","90"} };
String[] names = {"Name", "course", "Marks"};
jt = new JTable(data,names);
JTableHeader head = jt.getTableHeader();
Container c = this.getContentPane();
c.setLayout(new BorderLayout());
c.add("North", head);
c.add("Center",jt);
this.setVisible(true);
this.setSize(300,400);
this.setTitle("JTableDemo");
this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);      }
public static void main(String[] args){
new JTableDemo();      }
}

```

***JprogressBar**: This component will display the progress of a task visually.

Creation of JprogressBar:

```

JProgressBar pbar = new JProgressBar();
JProgressBar pbar = new JProgressBar(int orientation);

```

Orientation will specify whether the progressBar should be displayed either horizontally or vertically.

Methods of ProgressBar:

1. **SetMinimum(int)**: This method will set minimum value of the progressbar.
2. **SetMaximum(int)**: Set the maximum value of the progressbar.
3. **SetValue(int)**: This method will set maximum value the progressbar. The value to this method must be with in the range of min and max value.
4. **getMinimum()**: This method returns the minimum value set to the progressbar.
5. **getMaximum()**: The maximum value set to the progressbar.
6. **getValue()**: The current status of the progressbar.

7. **SetOrientation(int)**: This method returns is use to specify the Orientation of the progressbar.

8. **getOrientation()**: This method will return the orientation of the progressbar.

9. **SetStringPainted(boolean)**: This method will display the percentage of the task that is executed.

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
class JProgressBarDemo extends JFrame implements ActionListener{
JProgressBar pbar;
JButton b;
JProgressBarDemo(){
pbar = new JProgressBar();
pbar = setMinimum(0);
pbar = setMaximum(100);
pbar = setValue(0);
pbar = setForeground(color.red);
pbar = setStringpainted(true);
b = new JButton("Click Here");
Container c = this.getContentPane();
c.setLayout(new FlowLayout());
c.add(pbar);
c.add(b);
b.addActionListener(this);
this.setVisible(true);
this.setSize(400,400);
this.setTitle("JProgressBar");
this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
}

public void actionPerformed(ActionEvent ae){
if(pbar.getValue() == pbar.getMaximum())
s.exit(0);
pbar.setValue(pbar.getValue(7+5));
}
public static void main(String[] args){
new JProgressBarDemo();
}
}
```

***JButton**: This component can be used to perform some operation when the user click on it.

Creation of JButton: JButton jb = new JButton(label);

***JComboBox**: This component will display a group of items as drop down menu from which one of the items can be selected.

Creation of JComboBox: JComboBox jcb = new JComboBox();

We can add the items to the ComboBox by using add item method.

jcb.addItem(item);

***Pane**: pane is an area in which we can display the components.

JTabbedPane: It is a pane which can contain tabs and each tab can display any component in the same pane. To add the tabs to the JTabbedPane we can use the following methods.

jtp.addTab(TabName,Components)

jtp.addTab(TabName,Component)

Border: Border is an interface using which we can apply a border to every component. To create the borders we have to use the methods available in BorderFactory class.

We can apply the created border to any component by using SetBorder method.

Component.setBorder(Border);

```
import.java.swing.*;
import.javax.swing.border.*;
import.java.awt.*;
class JTabbedPaneDemo extends JFrame{
JTabbedPane jtp;
JTabbedPaneDemo(){
jtp = new JTabbedPane();
jtp = addTab("Button", new ButtonPanel());
jtp = addTab("ComboBox", new ComboPanel());
Container c = this.getContentPane();
c.setLayout(new FlowLayout());
c.add(jtp);
setVisible(true);
setSize(400,400);
SetDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
}
public static void main(String args[]){
new JTabbedPaneDemo();
}
Class Button_Panel extends JPanel{
JButton b1,b2,b3,b4;
ButtonPanel(){

```

```

b1 = new JButton("Button1");
b2 = new JButton("Button2");
b3 = new JButton("Button3");
b4 = new JButton("Button4");
Border b = BorderFactroy.createBevelBorder
        (BevelBorder.LOWERED, color.red, color.green);
        b1.setBorder(b);
b = BorderFactory.create BevelBorder
        (BevelBorder.LOWERED,color.red,color.green);
        b2.setBorder(b);
b = BorderFactroy.createLineBorder(color.blue,10);
        b3.setBorder(b);
b = BorderFactroy.createMatteBorder(5,10,15,20,color.red);
        b4.setBorder(b);
add(b1);
add(b2);
add(b3);
add(b4);
}
}

class ComboPanel extends JPanel{
JComboxBox jcb;
ComboPanel(){
jcb = new JComboxBox();
jcb.addItem("Hyderabad");
jcb.addItem("Chennai");
jcb.addItem("Delhi");
jcb.addItem("Nellore");
add(jcb);
}
}

```

***JMenuBar**: This component is used to create a menu bar which can contain some menus.

***Creation of menu bar**: JMenuBar mbar = new JMenuBar();

***JMenu**: This component is used to create a menu which can contain some menu item.

***Creation of JMenu**: JMenu FileMenu = new JMenu("File");

We can add the menu to the menu bar by using add Method.

mbar.add(FileMenu);

***JMenuItem**: This component is used to create menu's items which can be placed on to the menu.

***Creation of JMenuItem**:

JMenuItem newItem = new JMenuItem("New");

We can add the menu item to the menu by using add method.

```
    FileMenu.add(newItem);
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
class MenuDemo extends JFrame implements ActionListener{
JMenuBar mbar;
JMenu FileMenu, EditMenu;
JMenuItem newItem, openItem, saveItem, exitItem, cutItem, copyItem, pasteItem;
JCheckBoxMenuItem cbox;
MenuDemo(){
mbar = new JMenuBar();
FileMenu = new JMenu("File");
EditMenu = new JMenu("Edit");
mbar.add(FileMenu);
mbar.add(EditMenu);
newItem = new JMenuItem("new");
openItem = new JMenuItem("open");
saveItem = new JMenuItem("save");
exitItem = new JMenuItem("exit");
cutItem = new JMenuItem("cut");
copyItem = new JMenuItem("copy");
pasteItem = new JMenuItem("paste");
cbox = new JCheckBoxMenuItem("choice");
FileMenu.add(newItem);
FileMenu.add(openItem);
FileMenu.add(saveItem);
FileMenu.add(exitItem);
FileMenu.addSeparator();
EditMenu.add(cutItem);
EditMenu.add(copyItem);
EditMenu.add(pasteItem);
EditMenu.add(cbox);
newItem.addActionListener(this);
openItem.addActionListener(this);
saveItem.addActionListener(this);
exitItem.addActionListener(this);
cutItem.addActionListener(this);
copyItem.addActionListener(this);
pasteItem.addActionListener(this);
Container c = this.getContentPane();
```

```
c.setLayout(new BorderLayout());
c.add("North",mbar);
setVisible(true);
setSize(400,600);
setTitle("menu bar");
setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
}

Public void actionPerformed(ActionEvent ae){
if(newItem.isArmed())
System.out.println("new item clicked");
if(openItem.isArmed())
System.out.println("open item clicked");
if(exitItem.isArmed())
System.out.println("exit item clicked");
if(cutItem.isArmed())
System.out.println("cut item clicked");
}

public static void main(String[] args){
new MenuDemo();
}
```

***Layouts**: Layout will specify the format or the order which the components has to be placed on the container.

Layout manager is a class /component that it responsible for arrange the components on the container according to the specified layout.

Different types of layout that can be used are

1. FlowLayout
2. BorderLayout
3. CardLayout
4. GridLayout
5. GridBagLayout

***FlowLayout**: This Layout will display the components in sequence from left to right, from top to bottom. The components will always be displayed in firstline and in the firsts line is fill these components displayed next line automatically.

***Creation of FlowLayout**:

```
FlowLayout fl = new FlowLayout();
FlowLayout fl = new FlowLayout(int align);
FlowLayout fl = new FlowLayout(int align, int hgap, int vgap);
```

```
import javax.swing.*;
import java.awt.*;
class LayoutDemo extends JFrame{
JButton b1, b2, b3, b4, b5;
LayoutDemo(){
b1 = new JButton("Button1");
b2 = new JButton("Button2");
b3 = new JButton("Button3");
b4 = new JButton("Button4");
b5 = new JButton("Button5");
Container c = this.getContentPane();
FlowLayout fl = new FlowLayout(FlowLayout.LEFT,20,30);
c.setLayout(fl);
c.add(b1);
c.add(b2);
c.add(b3);
c.add(b4);
c.add(b5);
setVisible(true);
setSize(400,600);
setTitle("LayoutDemo");
setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
}
Public static void main(String[] args){
new LayoutDemo();
}
```

***BorderLayout**: This Layout will display the components along the border of the container. This Layout contains five locations where the component can be displayed. Locations are North, South, East, West and Center(N,S,E,W & C).

***Creation of BorderLayout**:

```
BorderLayout bl = new BorderLayout();
BorderLayout bl = new BorderLayout(int vgap, int hgap);
import javax.swing.*;
import java.awt.*;
class LayoutDemo extends JFrame{
JButton b1, b2, b3, b4, b5;
Layoutdemo(){
b1 = new JButton("Button1");
b2 = new JButton("Button2");
```

```

b3 = new JButton("Button3");
b4 = new JButton("Button4");
b5 = new JButton("Button5");
Container c = thisContentPane();
BorderLayout bl = new BorderLayout(10,20);
c.setLayout(bl);
c.add("North",b1);
c.add("South",b2);
c.add("East",b3);
c.add("West",b4);
c.add("Center",b5);
setVisible(true);
setSize(400,400);
setTitle("BorderDemo");
setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
}
public static void main(String[] args){
new LayoutDemo();
}
}

```

To add the components in Border Layout we use add method.

```

add("North",Component);
add(Component, BorderLayout, NORTH);

```

***CardLayout:** A cardLayout represent a stack of cards displayed on a container. At time only one card can be visible and each can contain only one component.

***Creation of CardLayout:**

```

CardLayout cl = new CardLayout();
CardLayout cl = new CardLayout(int hgap, int vgap);
First(conis);

```

To add the components in CardLayout we use add method.

```

add("Cardname",Component);

```

methods of cardLayout to access athercards:

```

first(Container);
last(Container);
next(Container);
previous(Container);
show(Container,cardname);

```

```

import javax.swing.*;
import java.awt.*;

```

```
import java.awt.event.*;
class LayoutDemo extends JFrame implements ActionListener{
JButton b1, b2, b3, b4, b5;
CardLayout cl;
Container c;
LayoutDemo(){
b1 = new JButton("Button1");
b2 = new JButton("Button2");
b3 = new JButton("Button3");
b4 = new JButton("Button4");
b5 = new JButton("Button5");
c = this.getContentPane();
cl = new CardLayout(10,20);
c.setLayout(cl);
c.add("card1",b1);
c.add("card2",b2);
c.add("card3",b3);
b1.addActionListener(this);
b2.addActionListener(this);
b3.addActionListener(this);
setVisible(true);
setSize(400,400);
setTitle("CardLayout");
setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
}
public void actionPerformed(ActionEvent ae){
cl.next(c);
}
public static void main(String args[]){
new LayoutDemo();
}
}
```

***GridLayout**: Layout will display the components in the format of rows and columns. The container will be divided into table of rows and column. The intersection of a row and column cell and every cell contain only one component and all the cells are equal size.

***Creation of GridLayout:**

```
GridLayout gl = new GridLayout(int rows, int cols);
GridLayout gl = new GridLayout(int rows, int cols, int vgap, int hgap);
import javax.swing.*;
import java.awt.*;
class LayoutDemo extends JFrame{
```

```
 JButton b1, b2, b3;
 GridLayout gl;
 Container c;
 LayoutDemo(){
    b1 = new JButton("Button1");
    b2 = new JButton("Button2");
    b3 = new JButton("Button3");
    c = this.getContentPane();
    gl = new GridLayout(2,3,10,20);
    c.setLayout(gl);
    c.add(b1);
    c.add(b2);
    c.add(b3);
    setVisible(true);
    setSize(400,600);
    setTitle("GridLayout");
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
}
public static void main(String[] args){
new LayoutDemo();
}
```

***GridBagLayout**: This layout is the most efficient layout that can be used for displaying components. In this layout we can specify the location specify the size and etc.

***Creation of GridBagLayout**:

```
GridBagLayout gbl = new GridBagLayout();
```

We can specify the location (or) the size with the help of GridBagConstraints.

***Creation of GridBagConstraint**:

```
GridBagConstraints cons = new GridBagConstraints();
```

1. **gridx, gridy**: These constraints will specify the location of the cell where the component has to be placed.
2. **Gridwidth, gridheight**: How many cells can be used to display. The component either horizontally or vertically. The default value of gridwidth and gridheight is '1'.
3. **ipadx, ipady**: These constraints are used to add extra pixels to the component either horizontally or vertically.

4. weightx, weighty: These constraints will specify how much the component must be resized. Either horizontally or vertically, when the component size is smaller than the container (resized).

Fill: This component used to stretch the component either horizontally or vertically, when the component size is smaller than the container.

```
import javax.swing.*;
import java.awt.*;
class LayoutDemo extends JFrame{
JButton b1, b2, b3, b4, b5;
Container c;
GridLayout gbl;
GridBagConstraints cons;
LayoutDemo(){
b1 = new JButton("Button1");
b2 = new JButton("Button2");
b3 = new JButton("Button3");
b4 = new JButton("Button4");
b5 = new JButton("Button5");
c = this.getContentPane();
gbl = new GridLayout();
c.setLayout(gbl);
cons.Fill = GridBagConstraints();
cons.Weightx = 0.8;
cons.gridx = 0;
cons.gridy = 0;
gbl.setConstraints(b1,cons);
c.add(b1);
cons.gridx = 1;
cons.gridy = 0;
gbl.setConstraints(b2,cons);
c.add(b2);
cons.gridx = 2;
cons.gridy = 0;
gbl.setConstraints(b3,cons);
c.add(b3);
cons.gridx = 0;
cons.gridy = 1;
cons.gridwidth = 3;
cons.ipady = 100;
```

```
gbl.setConstraints(b4,cons);
c.add(b4);
cons.gridx = 1;
cons.gridy = 2;
cons.gridwidth = 2;
cons.ipady = 50;
gbl.setConstraints(b5,cons);
c.add(b5);
setVisible(true);
setSize(400,400);
setTitle("Layout");
setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
}
public static void main(String[] args){
LayoutDemo ld = new LayoutDemo();
}
}
```

APPLET

In java we can have two types of applications.

1. **Standalone applications:** The applications that are executed in the context of a local machine are called as standalone applications. These applications use most of system resources and the resources are not sharable. These kind of applications contain main() method.
2. **Distributed applications:** The applications that are executed in the control of a browser are called as distributed applications. The amount of resources required is very minimum and the resources are sharable. These applications will not contain main() method. To develop a distributed GUI we use Applet.

***Applet:** An applet is a java program which registers on the server and will be executed in the client.

***Life cycle methods:**

1. **init()**: This is the 1st method to be executed this method contains the code for initialization. This method will be executed only one time.
2. **start()**: This method will be executed after init() method. This method contains business logic like connecting to files data bases processing the data, generating report etc. this method will be executed multiple times as long as the applet has the control.
3. **stop()**: This method will be executed when the applet loses the control. This method contains a logic for the performing “cleanup activities”. This method will be executed multiple times when ever the applet loses the control.
4. **destroy()**: This is the last method to be executed before terminating the applet. This method is used to destroy the applet. This method executed only one time.

Note: when even destroy() method is called before invoking it will call stop() method.

The above methods can be called as life cycle methods or call back methods.

All the life cycle methods are available in class called as applet and they are available as null implementations.

All the four methods are optional methods.

Note: The applet class is available in “java.applet” package.

```
import java.applet.*;  
import java.awt.*;  
  
public class AppletDemo extends Applet{  
    public void init(){  
        SetForeground(color.white);  
        SetBackground(color.black);  
        SetFont(new Font("arial",Font.ITALIC,34));  
        System.out.println("inside init()");  
    }  
}
```

```
public void start(){  
    System.out.println("inside start()");  
}  
  
public void stop(){  
    System.out.println("inside stop()");  
}  
  
public void destroy(){  
    System.out.println("inside destroy()");  
}  
  
public void paint(Graphics g){  
    g.drawString("hello",50,30);  
}
```

Compile the program AppletDemo.java, then write

HTML DOCUMENT

```
<html>  
<applet code = "AppletDemo" width = "300" height = "300">  
</applet>  
</html>
```

Save as AppletDemo.html

The execution of above applet can be done in 2 ways.

Way1: open AppletDemo.html in a browser

Way2: execute the AppletDemo.html file by using appletviewer command.

***appletviewer AppletDemo.html:**

We can write the <applet> in an html document or it can be specified in the java file it self.

```
/*
<applet code = "AppletDemo" width = "300" height = "300">
</applet>
*/
```

Compile: javac appletDemo.java

Execute: appletviewer AppletDemo.java (But it not display in browser)

Note: for every applet there should be one <applet> tag.

```
import java.applet.*;
import java.awt.*;
import java.awt.event.*;

public class FirstApplet extends Applet implements ActionListener{
    TextField tf;
    Button b;
    AppletContext ac;
    public void init(){
        tf = new TextField(25);
        b = new Button("send");
        //getting the AppletContext
        ac = this.getAppletContext();
        add(tf);
        add(b);
        b.addActionListener(this);
    }
}
```

```
public void actionPerformed(ActionEvent ae){  
    String str = tf.getText();  
    Applet a = ac.getApplet("secont");  
    SecondApplet sa = (SecondApplet)a;  
    sa.setText(str);  
}  
}  
  
//save: FirstApplet.java  
  
import java.applet.*;  
import java.awt.*;  
public class SecondApplet extends Applet{  
    String str;  
    public void init(){  
        setForeground(color.white);  
        setBackground(color.green);  
       setFont(new Font("arial",Font.BOLD,23));  
        str = "Applet to Applet Communication";  
    }  
    public void setText(String str){  
        this.str = str;  
        repaint();  
    }  
    public void paint(Graphics g){  
        g.drawString(str,50,50);    }  
}
```

```
}

//save: SecondApplet.java

//HTML Document

<html>

<applet code = "FirstApplet" name = "first" width = "200" height = "200">

</applet>

<applet code = "SecondApplet" name = "second" width = "200" height = "200">

</applet>

</html>

import javax.swing.*;

import java.awt.*;

/*
<applet code = "Running" width = "800" height = "400">

</applet>

*/
public class Running extends JApplet{

public void paint(Graphics g){

Image i = getImage(getCodeBase(),"man.gif");

for(int x = 0; x <=800; x++){

g.drawImage(i,x,0,null);

try{

Thread.Sleep(50);

}

Catch(Interrupted Exception ie){
```

```
ie.printStackTrace();  
    }  
}  
}  
}  
  
<html>  
  <applet code = "Running" width = "800" height = "400">  
  </applet>  
</html>
```

Generic Data Types

*Generics:

Generics is a concepts introduced in the java 1.5 version. Generics is called as parameterized types.

Generics are design to provide type safely, which will reduce the need for type_casting.

Generics are set to be type erasures, which mean the generics information will be available up to compilation once compile it does not contain generic information it will be erasure.

*Procedure to create a generic method:

Specify the generic type parameter before the return type of the method.

Syn: <E> returnType methodName(){

```
}
```

E represents the Generic type parameter.

```
Class GenericDemo{  
    Public <T> void display(T[] x){  
        for(T i : x){  
            System.out.println(i);  
        }  
    }  
    public static void main(String[] args){  
        Integer[] iarr = {1,2,3,4,5};  
        Double[] darr = {1.2,2.3,3.4,4.5,5.6};  
        String[] sarr = {"abc","def","ghi"};  
        GenericDemo gd = new GenericDemo();  
        gd.display(iarr);  
        gd.display(darr);  
        gd.display(sarr);  
    }  
}
```

***Procedure to create Generic class:** Declare the Generic type parameter after the class declaration.

Syn: class className<E>{

```
}
```

```
Class Myclass<T>{
```

```
    T obj;
```

```
Myclass(T obj){  
    This.obj = obj;  
}  
  
T getValue(){  
    return obj;  
}  
  
Public void showType(){  
    System.out.println("Type :" +obj.getClass().getName());  
}  
}  
  
class GenericDemo{  
    public staic void main(String[] args){  
        Integer iobj = new Integer(123);  
  
        MyClass<Integer> mc1 = new MyClass<Integer>(iobj);  
  
        System.out.println("value:" +mc1.getValue());  
  
        mc1.showType();  
  
        String sobj = new String("java");  
  
        MyClass<String> mc2 = new MyClass<String>(sobj);  
  
        System.out.println("value:" +mc2.getValue());  
  
        mc2.showType();  
    }  
}
```

New Features in Java

J2SE 5 Features

Java 1.5 was called **J2SE 5**, it added following major new features :

- Generics
- Annotations
- Autoboxing and autounboxing
- Enumerations
- For-each Loop
- Varargs
- Static Import
- Formatted I/O
- Concurrency utilities

Next major release was **Java SE 7** which included many new changes, like :

- Now **String** can be used to control Switch statement.
- Multi Catch Exception
- *try-with-resource* statement
- Binary Integer Literals
- *Underscore* in numeric literals, etc.

And the latest addition to the lot is, **Java SE 8**, it was released on March 18, 2014. Some of the major new features introduced in JAVA 8 are,

- Lambda Expressions
- New Collection Package `java.util.stream` to provide Stream API.
- Enhanced Security
- Nashorn Javascript Engine included
- Parallel Array Sorting
- The JDBC-ODBC Bridge has been removed et

For-each loop (Advanced or Enhanced For loop):

The for-each loop introduced in Java5. It is mainly used to traverse array or collection elements. The advantage of for-each loop is that it eliminates the possibility of bugs and makes the code more readable.

Advantage of for-each loop:

- It makes the code more readable.
- It eliminates the possibility of programming errors.

Syntax of for-each loop:

```
for(data_type variable : array | collection){}
```

Simple Example of for-each loop for traversing the array elements:

```
class ForEachExample1{
    public static void main(String args[]){
        int arr[]={12,13,14,44};

        for(int i:arr){
            System.out.println(i);
        }
    }
}
```

Variable Argument (Varargs):

The varargs allows the method to accept zero or multiple arguments. Before varargs either we use overloaded method or take an array as the method parameter but it was not considered good because it leads to the maintenance problem. If we don't know how many argument we will have to pass in the method, varargs is the better approach.

Advantage of Varargs:

We don't have to provide overloaded methods so less code.

Syntax of varargs:

The varargs uses ellipsis i.e. three dots after the data type. Syntax is as follows:

```
return_type method_name(data_type... variableName){}
```

Simple Example of Varargs in java:

```

class VarargsExample1{

static void display(String... values){
    System.out.println("display method invoked ");
}

public static void main(String args[]){

    display();//zero argument
    display("my","name","is","varargs");//four arguments
}
}

1. class VarargsExample2{
2.
3. static void display(String... values){
4.     System.out.println("display method invoked ");
5.     for(String s:values){
6.         System.out.println(s);
7.     }
8. }
9.
10. public static void main(String args[]){
11.
12. display();//zero argument
13. display("hello");//one argument
14. display("my","name","is","varargs");//four arguments
15. }
16. }
17.
```

Static Import:

The static import feature of Java 5 facilitate the java programmer to access any static member of a class directly. There is no need to qualify it by the class name.

Advantage of static import:

- Less coding is required if you have access any static member of a class oftenly.

Disadvantage of static import:

- If you overuse the static import feature, it makes the program unreadable and unmaintainable.

Simple Example of static import

```
import static java.lang.System.*;
class StaticImportExample{
    public static void main(String args[]){
        out.println("Hello");//Now no need of System.out
        out.println("Java");
    }
}
```

What is the difference between import and static import?

The import allows the java programmer to access classes of a package without package qualification whereas the static import feature allows to access the static members of a class without the class qualification. The import provides accessibility to classes and interface whereas static import provides accessibility to static members of the class.

Autoboxing and Unboxing:

The automatic conversion of primitive data types into its equivalent Wrapper type is known as boxing and opposite operation is known as unboxing. This is the new feature of Java5. So java programmer doesn't need to write the conversion code.

Advantage of Autoboxing and Unboxing:

No need of conversion between primitives and Wrappers manually so less coding is required.

Simple Example of Autoboxing in java:

```
class BoxingExample1{
    public static void main(String args[]){
        int a=50;
        Integer a2=new Integer(a);//Boxing
        Integer a3=5;//Boxing
        System.out.println(a2+" "+a3);
    }
}
```

Simple Example of Unboxing in java:

The automatic conversion of wrapper class type into corresponding primitive type, is known as Unboxing. Let's see the example of unboxing:

```
class UnboxingExample1{
    public static void main(String args[]){
        Integer i=new Integer(50);
        int a=i;

        System.out.println(a);
    }
}
```

Autoboxing and Unboxing with comparison operators

Autoboxing can be performed with comparison operators. Let's see the example of boxing with comparison operator:

```
class UnboxingExample2{
    public static void main(String args[]){
        Integer i=new Integer(50);

        if(i<100){           //unboxing internally
            System.out.println(i);
        }
    }
}
```

Autoboxing and Unboxing with method overloading

In method overloading, boxing and unboxing can be performed. There are some rules for method overloading with boxing:

- **Widening beats boxing**
- **Widening beats varargs**
- **Boxing beats varargs**

1) Example of Autoboxing where widening beats boxing

If there is possibility of widening and boxing, widening beats boxing.

```
class Boxing1{
    static void m(int i){System.out.println("int");}
    static void m(Integer i){System.out.println("Integer");}

    public static void main(String args[]){
        short s=30;
        m(s);
    }
}
```

```
}
```

2) Example of Autoboxing where widening beats varargs

If there is possibility of widening and varargs, widening beats var-args.

```
class Boxing2{
    static void m(int i, int i2){System.out.println("int int");}
    static void m(Integer... i){System.out.println("Integer...");}
}

public static void main(String args[]){
    short s1=30,s2=40;
    m(s1,s2);
}
```

3) Example of Autoboxing where boxing beats varargs

Let's see the program where boxing beats variable argument:

```
class Boxing3{
    static void m(Integer i){System.out.println("Integer");}
    static void m(Integer... i){System.out.println("Integer...");}
}

public static void main(String args[]){
    int a=30;
    m(a);
}
```

Method overloading with Widening and Boxing

Widening and Boxing can't be performed as given below:

```
class Boxing4{
    static void m(Long l){System.out.println("Long");}
}

public static void main(String args[]){
    int a=30;
    m(a);
}
```

Java Enum

Enum in java is a data type that contains fixed set of constants.

It can be used for days of the week (SUNDAY, MONDAY, TUESDAY, WEDNESDAY, THURSDAY, FRIDAY and SATURDAY) , directions (NORTH, SOUTH, EAST and WEST) etc. The java enum constants are static and final implicitly. It is available from JDK 1.5.

Java Enums can be thought of as classes that have fixed set of constants.

Points to remember for Java Enum

- o enum improves type safety
- o enum can be easily used in switch
- o enum can be traversed
- o enum can have fields, constructors and methods
- o enum may implement many interfaces but cannot extend any class because it internally extends Enum class

Simple example of java enum

```
class EnumExample1{
public enum Season { WINTER, SPRING, SUMMER, FALL }

public static void main(String[] args) {
for (Season s : Season.values())
System.out.println(s);

}}
```

What is the purpose of values() method in enum?

The java compiler internally adds the values() method when it creates an enum. The values() method returns an array containing all the values of the enum.

Internal code generated by the compiler for the above example of enum type

The java compiler internally creates a static and final class that extends the Enum class as shown in the below example:

```
public static final class EnumExample1$Season extends Enum
{
private EnumExample1$Season(String s, int i)
{
    super(s, i);
}

public static EnumExample1$Season[] values()
{
    return (EnumExample1$Season[][])VALUES.clone();
}
```

```

    }

public static EnumExample1$Season valueOf(String s)
{
    return (EnumExample1$Season)Enum.valueOf(EnumExample1$Season, s);
}
public static final EnumExample1$Season WINTER;
public static final EnumExample1$Season SPRING;
public static final EnumExample1$Season SUMMER;
public static final EnumExample1$Season FALL;
private static final EnumExample1$Season $VALUES[];

static
{
    WINTER = new EnumExample1$Season("WINTER", 0);
    SPRING = new EnumExample1$Season("SPRING", 1);
    SUMMER = new EnumExample1$Season("SUMMER", 2);
    FALL = new EnumExample1$Season("FALL", 3);
    $VALUES = (new EnumExample1$Season[] {
        WINTER, SPRING, SUMMER, FALL
    });
}

}

```

Defining Java enum

The enum can be defined within or outside the class because it is similar to a class.

Java enum example: defined outside class

1. **enum** Season { WINTER, SPRING, SUMMER, FALL }
2. **class** EnumExample2{
3. **public static void** main(String[] args) {
4. Season s=Season.WINTER;
5. System.out.println(s);
6. }}

Java enum example: defined inside class

1. **class** EnumExample3{
2. **enum** Season { WINTER, SPRING, SUMMER, FALL; }*//semicolon(;) is optional here*
3. **public static void** main(String[] args) {
4. Season s=Season.WINTER;*//enum type is required to access WINTER*
5. System.out.println(s);
6. }}

Initializing specific values to the enum constants

The enum constants have initial value that starts from 0, 1, 2, 3 and so on. But we can initialize the specific value to the enum constants by defining fields and constructors. As specified earlier, Enum can have fields, constructors and methods.

Example of specifying initial value to the enum constants

```

1. class EnumExample4{
2. enum Season{
3. WINTER(5), SPRING(10), SUMMER(15), FALL(20);
4.
5. private int value;
6. Season(int value){
7. this.value=value;
8. }
9. }
10. public static void main(String args[]){
11. for (Season s : Season.values())
12. System.out.println(s+" "+s.value);
13.
14. }}}

```

Constructor of enum type is private. If you don't declare private compiler internally creates private constructor.

```

enum Season{
WINTER(10),SUMMER(20);
private int value;
Season(int value){
this.value=value;
}
}

```

Internal code generated by the compiler for the above example of enum type

```

1. final class Season extends Enum
2. {
3.   public static Season[] values()
4.   {
5.     return (Season[])$VALUES.clone();
6.   }
7.   public static Season valueOf(String s)
8.   {
9.     return (Season)Enum.valueOf(Season, s);
10. }
11. private Season(String s, int i, int j)
12. {
13.   super(s, i);
14.   value = j;

```

```

15. }
16. public static final Season WINTER;
17. public static final Season SUMMER;
18. private int value;
19. private static final Season $VALUES[];
20. static
21. {
22.     WINTER = new Season("WINTER", 0, 10);
23.     SUMMER = new Season("SUMMER", 1, 20);
24.     $VALUES = (new Season[] {
25.         WINTER, SUMMER
26.     });
27. }
28. }
```

Can we create the instance of enum by new keyword?

No, because it contains private constructors only.

Can we have abstract method in enum?

Yes, ofcourse! we can have abstract methods and can provide the implementation of these methods.

Java enum in switch statement

We can apply enum on switch statement as in the given example:

Example of applying enum on switch statement

```

class EnumExample5{
enum Day{ SUNDAY, MONDAY, TUESDAY, WEDNESDAY, THURSDAY, FRIDAY, SATURDAY}
public static void main(String args[]){
    Day day=Day.MONDAY;

    switch(day){
        case SUNDAY:
            System.out.println("sunday");
            break;
        case MONDAY:
            System.out.println("monday");
            break;
        default:
            System.out.println("other day");
    }
}}
```

Java Annotations

Java **Annotation** is a tag that represents the *metadata* i.e. attached with class, interface, methods or fields to indicate some additional information which can be used by java compiler and JVM.

Annotations in java are used to provide additional information, so it is an alternative option for XML and java marker interfaces.

First, we will learn some built-in annotations then we will move on creating and using custom annotations.

Built-In Java Annotations

There are several built-in annotations in java. Some annotations are applied to java code and some to other annotations.

Built-In Java Annotations used in java code

- @Override
- @SuppressWarnings
- @Deprecated

Built-In Java Annotations used in other annotations

- @Target
- @Retention
- @Inherited
- @Documented

Understanding Built-In Annotations in java

Let's understand the built-in annotations first.

@Override

@Override annotation assures that the subclass method is overriding the parent class method. If it is not so, compile time error occurs.

Sometimes, we does the silly mistake such as spelling mistakes etc. So, it is better to mark @Override annotation that provides assurity that method is overridden.

```
1. class Animal{  
2. void eatSomething(){System.out.println("eating something");}  
3. }  
4.  
5. class Dog extends Animal{  
6. @Override  
7. void eatsomething(){System.out.println("eating foods");}//should be eatSomething  
8. }  
9.  
10. class TestAnnotation1{  
11. public static void main(String args[]){  
12. Animal a=new Dog();  
13. a.eatSomething();  
14. }}}
```

@SuppressWarnings

@SuppressWarnings annotation: is used to suppress warnings issued by the compiler.

```
1. import java.util.*;  
2. class TestAnnotation2{  
3. @SuppressWarnings("unchecked")  
4. public static void main(String args[]){  
5. ArrayList list=new ArrayList();  
6. list.add("sonoo");  
7. list.add("vimal");  
8. list.add("ratan");  
9.  
10. for(Object obj:list)  
11. System.out.println(obj);  
12.  
13. }}
```

If you remove the @SuppressWarnings("unchecked") annotation, it will show warning at compile time because we are using non-generic collection.

@Deprecated

@Deprecated annotation marks that this method is deprecated so compiler prints warning. It informs user that it may be removed in the future versions. So, it is better not to use such methods.

```
1. class A{  
2. void m(){System.out.println("hello m");}  
3.  
4. @Deprecated  
5. void n(){System.out.println("hello n");}  
6. }  
7.  
8. class TestAnnotation3{  
9. public static void main(String args[]){  
10.  
11.A a=new A();  
12.a.n();  
13.}}
```

At Compile Time:

Note: Test.java uses or overrides a deprecated API.

Note: Recompile with -Xlint:deprecation for details.

Generics in Java

The **Java Generics** programming is introduced in J2SE 5 to deal with type-safe objects.

Before generics, we can store any type of objects in collection i.e. non-generic. Now generics, forces the java programmer to store specific type of objects.

Advantage of Java Generics

There are mainly 3 advantages of generics. They are as follows:

1) Type-safety : We can hold only a single type of objects in generics. It doesn't allow to store other objects.

2) Type casting is not required: There is no need to typecast the object.

Before Generics, we need to type cast.

```
List list = new ArrayList();
```

```
list.add("hello");
String s = (String) list.get(0); //typecasting
```

After Generics, we don't need to typecast the object.

```
List<String> list = new ArrayList<String>();
list.add("hello");
String s = list.get(0);
```

3) Compile-Time Checking: It is checked at compile time so problem will not occur at runtime. The good programming strategy says it is far better to handle the problem at compile time than runtime.

```
List<String> list = new ArrayList<String>();
list.add("hello");
list.add(32); //Compile Time Error
```

Syntax to use generic collection

1. ClassOrInterface<Type>

Example to use Generics in java

1. ArrayList<String>

Full Example of Generics in Java

Here, we are using the ArrayList class, but you can use any collection class such as ArrayList, LinkedList, HashSet, TreeSet, HashMap, Comparator etc.

```
import java.util.*;
class TestGenerics1{
public static void main(String args[]){
ArrayList<String> list=new ArrayList<String>();
list.add("rahul");
list.add("jai");
//list.add(32); //compile time error

String s=list.get(1); //type casting is not required
System.out.println("element is: "+s);
```

```
Iterator<String> itr=list.iterator();
while(itr.hasNext()){
System.out.println(itr.next());
}
```

Example of Java Generics using Map

Now we are going to use map elements using generics. Here, we need to pass key and value. Let us understand it by a simple example:

```

1. import java.util.*;
2. class TestGenerics2{
3. public static void main(String args[]){
4. Map<Integer,String> map=new HashMap<Integer,String>();
5. map.put(1,"vijay");
6. map.put(4,"umesh");
7. map.put(2,"ankit");
8.
9. //Now use Map.Entry for Set and Iterator
10. Set<Map.Entry<Integer,String>> set=map.entrySet();
11.
12. Iterator<Map.Entry<Integer,String>> itr=set.iterator();
13. while(itr.hasNext()){
14. Map.Entry e=itr.next(); //no need to typecast
15. System.out.println(e.getKey()+" "+e.getValue());
16. }
17.
18. }}
```

Generic class

A class that can refer to any type is known as generic class. Here, we are using **T** type parameter to create the generic class of specific type.

Let's see the simple example to create and use the generic class.

Creating generic class:

```

1. class MyGen<T>{
2. T obj;
3. void add(T obj){this.obj=obj;}
4. T get(){return obj;}
5. }
```

The **T** type indicates that it can refer to any type (like String, Integer, Employee etc.). The type you specify for the class, will be used to store and retrieve the data.

Using generic class:

Let's see the code to use the generic class.

```
1. class TestGenerics3{
```

```

2. public static void main(String args[]){
3. MyGen<Integer> m=new MyGen<Integer>();
4. m.add(2);
5. //m.add("vivek");//Compile time error
6. System.out.println(m.get());
7. }

```

Output:2

Type Parameters

The type parameters naming conventions are important to learn generics thoroughly. The commonly type parameters are as follows:

1. T - Type
2. E - Element
3. K - Key
4. N - Number
5. V - Value

Generic Method

Like generic class, we can create generic method that can accept any type of argument.

Let's see a simple example of java generic method to print array elements. We are using here **E** to denote the element.

```

public class TestGenerics4{
    public static < E > void printArray(E[] elements) {
        for ( E element : elements){
            System.out.println(element );
        }
        System.out.println();
    }
    public static void main( String args[] ) {
        Integer[] intArray = { 10, 20, 30, 40, 50 };
        Character[] charArray = { 'J', 'A', 'V', 'A', 'T', 'P', 'O', 'I', 'N', 'T' };

        System.out.println( "Printing Integer Array" );
        printArray( intArray );

        System.out.println( "Printing Character Array" );
        printArray( charArray );
    }
}

```

Wildcard in Java Generics

The ? (question mark) symbol represents wildcard element. It means any type. If we write <? extends Number>, it means any child class of Number e.g. Integer, Float, double etc. Now we can call the method of Number class through any child class object.

Let's understand it by the example given below:

```

import java.util.*;
abstract class Shape{
abstract void draw();
}
class Rectangle extends Shape{
void draw(){System.out.println("drawing rectangle");}
}
class Circle extends Shape{
void draw(){System.out.println("drawing circle");}
}
class GenericTest{
//creating a method that accepts only child class of Shape
public static void drawShapes(List<? extends Shape> lists){
for(Shape s:lists){
s.draw(); //calling method of Shape class by child class instance
}
}
public static void main(String args[]){
List<Rectangle> list1=new ArrayList<Rectangle>();
list1.add(new Rectangle());

List<Circle> list2=new ArrayList<Circle>();
list2.add(new Circle());
list2.add(new Circle());

drawShapes(list1);
drawShapes(list2);
}}}
```

JavaSE 6 Features

Collections Framework Enhancements: Java SE 6 API provides bi-directional collection access. New collection interfaces includes Deque, NavigableSet, NavigableMap.

New Collection classes include ArrayDeque.

ArrayDeque : Efficient resizable-array implementation of the Deque interface. It implements a double-ended queue, which allows efficient insertion and deletion at both ends. This an excellent choice for stacks and queues.

Example

```
ArrayDeque<String> a = new ArrayDeque<String>();  
  
for (Iterator<String> iter = a.iterator(); iter.hasNext(); ) {  
  
    System.out.println(iter.next());  
  
}
```

LinkedList : implements the Deque, interface. These operations allow linked lists to be used as a stack, queue, or double-ended queue.

TreeSet : implements the NavigableSet interface.

TreeMap : implements the NavigableMap interface.

java.io Enhancements: New class "Console" is added and it contains methods to access a character-based console device. The readPassword() methods disable echoing thus they are suitable for retrieval of sensitive data such as passwords. The method System.console () returns the unique console associated with the Java Virtual Machine.

Internationalization Enhancements: To enable plug-in of locale sensitive data (e.g., date format strings) and services (e.g., date formatters), some Service Provider Interfaces (SPIs) for locale-sensitive classes in the java.text and java.util packages have been added. These SPIs make it much easier for developers to provide support of more locales in addition to the currently available locales in Java SE - Unicode Normalizer API, Internationalized Domain Names Support, Japanese Imperial Calendar Support, ResourceBundle Enhancements.

JavaSE 7 Features

The important features of JavaSE 7 are try with resource, catching multiple exceptions etc.

- String in switch statement (Java 7)
- Binary Literals (Java 7)
- The try-with-resources (Java 7)
- Caching Multiple Exceptions by single catch (Java 7)
- Underscores in Numeric Literals (Java 7)

Given all the fun developers have with *hexadecimal notation*, the visionaries at Sun and Oracle decided to expand the party by introducing the equally cryptic *binary notation* to the Java language. With the addition of *binary notation*, there are now three different ways to initialize a numeric variable, first using the standard

base ten number system, secondly using hex notation, and finally using binary. Here's an example that assigns the number twelve to three different variables of type byte.

```

1 public class BinaryLiterals {
2     public static void main(String args[]) {
3         byte twelve = 12;
4         byte sixPlusSix = 0xC;
5         byte fourTimesThree = 0b1100;
6
7         System.out.println(twelve);
8         System.out.println(sixPlusSix);
9         System.out.println(fourTimesThree);
10    }
11 }
12

```

When the code above runs, it simply prints out the number 12 on three separate lines, demonstrating the veracity of all three notations:

```

12
12
12

```

Passing the Java 7 Professional Upgrade Exam

As part of the [Oracle Certified Profession, Java 7 certification upgrade exam](#), currently in beta as of this writing, one of the exam objectives is to "Use binary literals." So, in this article, we're going to take a look at some of the more interesting aspects of binary notation, focussing on what you'll need to know in order to whiz by the related questions on the exam. Even if you're not planning on writing a Java cert, you'll likely find the new news on binary notation quite interesting, so you are implored to keep reading.

Taking a byte out of binary initializations

In the above code snippet, you see an example of a byte being initialized to the value of 12 using binary notation: `byte fourTimesThree = 0b1100;`

Now, everyone knows that a byte represents eight bits of data, so the following attempted initialization, where you assign a ten bit binary number to a byte, completely blows up with the following error: **Type mismatch: cannot convert from int to byte**

```

1 byte data = 0b1100110011; // Type mismatch: cannot convert from int to
  byte

```

Interestingly, though not necessarily surprisingly, if the leading binary digits are zeroes, the JVM recognizes that these numbers are just placeholders, and initialization using ten bits is successful:

```

1 byte data = 0b0000110011; //successful

```

So, as was mentioned, a byte represents eight bits of binary data, so you would expect the following to compile, seeing that it initializes eight bits to a byte:

```

1 byte data = 0b10101010;

```

Sadly, this fails, because the byte data type only uses seven of its eight bits for holding data, while reserving the extra bit to indicate whether a number is positive or negative. That's why the range of the eight bit byte ($2^8 = 256$) is -128 to 127, and not 0 to 255.

By the way, sometimes make the mistake of thinking the leading zero before the 'b' in byte notation (`0b1100`) indicates whether the number is positive or negative. That's not correct. You can toggle the positivity or negativity of a binary notation number the same way as you would with a decimal number: with a minus sign:

```

1 //byte data = 1b1010101; not valid
2     byte data = -0b1010101; //valid

```

Now speaking of the problem of type mismatches and stuffing too many ones and zeroes into a data type, the **short** has the same issue as the **byte**, although with the short, you can initialize it to fifteen binary digits instead of eight:

```

1     short number = 0b1111111111111111; //valid
2 //short number = 0b1111111111111111; not valid

```

Inconsistent treatment of ints vs. shorts and bytes

So, the eight bit byte can take seven bits, the sixteen bit short can take fifteen bits, but for some reason, you can specify a full 32 binary bits for a 32-bit int:

```
1 int overflow = 0b10101010101010101010101010101011;
2 System.out.println(overflow);
```

Unfortunately, the assignment here does not work as nicely as planned. This number prints out as -1431655765, which is the negative value of the first 31 binary digits. The last digit of the binary number, which is a 1, is used to indicate that the number is negative. If the row of bits was literally translated into a number, without overflowing it into an int, it would actually have the value of 2863311531, not -1431655765. Oh, and just so you know, there is a way to force the Java compiler to treat a binary number as a long - all you have to do is throw the letter 'L' at the end of the bits. So, take a look at what happens when you print out these two numbers that use binary notation:

```
1 System.out.println(0b101010101010101010101010101011); // -1431655765 using 32 signed bits
2 System.out.println(0b101010101010101010101010101011L); // 2863311531 using 32 unsigned bits
```

The latter example can also be useful when assigning long binary numbers to a long; its utility goes far beyond printing out data to the console.

Isn't this int-eresting?

By default, the Java compiler likes to treat numbers as ints, but that can be a problem when you're defining numbers that fall into the exclusive range of a long. Take a look at the following code, which tries to initialize a long to a 33 bit value, a number that is well within the 64 bit range of a long, but far too large to be stuffed into a variable of type int:

```
1 long bow = 0b1010101010101010101010101010111; // causes a compile error
```

This line of code triggers the following compile error: The literal 0b101010101010101010101010111 of type int is out of range.

Of course, we know that this *should* work, so we can force the compiler to treat our binary number as a 64 bit long, again, by appending the letter 'L'.

```
1 //long bow = 0b1010101010101010101010101010111; fails to compile
2 long bow = 0b1010101010101010101010101010111L; //compiles
```

The appended 'L' can be both upper or lowercase, but since the lower case letter can be easily confused with a one, it's recommended to stick with the upper case value.

Now, since it's fine and dandy to place the letter L after a variable of type long is being initialized, you'd probably expect that the same type of notation could be used for float and double types. Well, you'd be wrong. So, while the following four lines of code are all valid:

```
1 float f1 = 12f;
2 double d1 = 12d;
3
4 float f2 = 0b111;
5 double d2 = 0b111;
```

The following lines of code will actually blow up:

```
1 float f3 = 0b111f;
2 double d3 = 0b111d;
```

Using binary notation with an 'f' or a 'd' to denote that a number is a float or a double generates the following compile time error: **Syntax error on token "f", delete this token Syntax error on token "d", delete this token**

And that's about it. That's everything you need to know about binary notation in order to tackle the corresponding objective in the Oracle Certified Professional for Java 7 exam. Best of luck!

Resource Management With Try-Catch-Finally, Old School Style

Managing resources that need to be explicitly closed is somewhat tedious before Java 7.

Look at the following method which reads a file and prints it to the `System.out`:

```
private static void printFile() throws IOException {
    InputStream input = null;

    try {
        input = new FileInputStream("file.txt");

        int data = input.read();
        while(data != -1){
            System.out.print((char) data);
            data = input.read();
        }
    } finally {
        if(input != null){
            input.close();
        }
    }
}
```

The code marked in bold is where the code can throw an `Exception`. As you can see, that can happen in 3 places inside the `try`-block, and 1 place inside the `finally`-block.

The `finally` block is always executed no matter if an exception is thrown from the `try` block or not. That means, that the `InputStream` is closed no matter what happens in the `try` block. Or, attempted closed that is.

The `InputStream`'s `close()` method may throw an exception too, if closing it fails.

Imagine that an exception is thrown from inside the `try` block. Then the `finally` block is executed. Imagine then, that an exception is also thrown from the `finally` block. Which exception do you think is propagated up the call stack?

The exception thrown from the `finally` block would be propagated up the call stack, even if the exception thrown from the `try` block would probably be more relevant to propagate.

Try-with-resources

In Java 7 you can write the code from the example above using the `try-with-resource` construct like this:

```
private static void printFileJava7() throws IOException {
```

```
try(FileInputStream input = new FileInputStream("file.txt")) {  
    int data = input.read();  
    while(data != -1){  
        System.out.print((char) data);  
        data = input.read();  
    }  
}
```

Notice the first line inside the method:

```
try(FileInputStream input = new FileInputStream("file.txt")) {
```

This is the `try-with-resources` construct. The `FileInputStream` variable is declared inside the parentheses after the `try` keyword. Additionally, a `FileInputStream` is instantiated and assigned to the variable.

When the `try` block finishes the `FileInputStream` will be closed automatically. This is possible because `FileInputStream` implements the Java interface `java.lang.AutoCloseable`. All classes implementing this interface can be used inside the `try-with-resources` construct.

If an exception is thrown both from inside the `try-with-resources` block, and when the `FileInputStream` is closed (when `close()` is called), the exception thrown inside the `try` block is thrown to the outside world. The exception thrown when the `FileInputStream` was closed is suppressed. This is opposite of what happens in the example first in this text, using the old style exception handling (closing the resources in the `finally` block).

Using Multiple Resources

You can use multiple resources inside a `try-with-resources` block and have them all automatically closed. Here is an example:

```
private static void printFileJava7() throws IOException {  
    try( FileInputStream      input      = new FileInputStream("file.txt");  
         BufferedInputStream bufferedInput = new BufferedInputStream(input)  
    ) {  
  
        int data = bufferedInput.read();  
        while(data != -1){  
            System.out.print((char) data);  
            data = bufferedInput.read();  
        }  
    }  
}
```

```
    }  
}
```

This example creates two resources inside the parentheses after the `try` keyword. An `FileInputStream` and a `BufferedInputStream`. Both of these resources will be closed automatically when execution leaves the `try` block.

The resources will be closed in reverse order of the order in which they are created / listed inside the parentheses. First the `BufferedInputStream` will be closed, then the `FileInputStream`.

Custom AutoClosable Implementations

The `try-with-resources` construct does not just work with Java's built-in classes. You can also implement the `java.lang.AutoCloseable` interface in your own classes, and use them with the `try-with-resources` construct.

The `AutoCloseable` interface only has a single method called `close()`. Here is how the interface looks:

```
public interface AutoCloseable {  
    public void close() throws Exception;  
}
```

Any class that implements this interface can be used with the `try-with-resources` construct. Here is a simple example implementation:

```
public class MyAutoCloseable implements AutoCloseable {  
  
    public void doIt() {  
        System.out.println("MyAutoCloseable doing it!");  
    }  
  
    @Override  
    public void close() throws Exception {  
        System.out.println("MyAutoCloseable closed!");  
    }  
}
```

The `doIt()` method is not part of the `AutoCloseable` interface. It is there because we want to be able to do something more than just closing the object.

Here is an example of how the `MyAutoClosable` is used with the `try-with-resources` construct:

```
private static void myAutoClosable() throws Exception {  
    try (MyAutoClosable myAutoClosable = new MyAutoClosable()) {  
        myAutoClosable.doIt();  
    }  
}
```

Here is the output printed to `System.out` when the method `myAutoClosable()` is called:

```
MyAutoClosable doing it!  
MyAutoClosable closed!
```

As you can see, `try-with-resources` is a quite powerful way of making sure that resources used inside a `try-catch` block are closed correctly, no matter if these resources are your own creation, or Java's built-in components.

Catching Multiple Exceptions in Java 7

In Java 7 it was made possible to catch multiple different exceptions in the same `catch` block. This is also known as multi catch.

Before Java 7 you would write something like this:

```
try {  
    // execute code that may throw 1 of the 3 exceptions below.  
}  
    catch(SQLException e) {  
        logger.log(e);  
    }  
    catch(IOException e) {  
        logger.log(e);  
    }  
    catch(Exception e) {  
        logger.severe(e);  
    }
```

As you can see, the two exceptions `SQLException` and `IOException` are handled in the same way, but you still have to write two individual `catch` blocks for them.

In Java 7 you can catch multiple exceptions using the multi catch syntax:

```
try {  
    // execute code that may throw 1 of the 3 exceptions below.  
}  
catch(SQLException | IOException e) {  
    logger.log(e);  
}  
catch(Exception e) {  
    logger.severe(e);  
}
```

Notice how the two exception class names in the first `catch` block are separated by the pipe character `|`. The pipe character between exception class names is how you declare multiple exceptions to be caught by the same `catch` clause.

LAMBDA EXPRESSIONS

Lambda expressions are introduced in Java 8 and are touted to be the biggest feature of Java 8. Lambda expression facilitates functional programming, and simplifies the development a lot.

Syntax

A lambda expression is characterized by the following syntax –
param eter -> expression body

Following are the important characteristics of a lambda expression –

Optional type declaration – No need to declare the type of a parameter. The compiler can inference the same from the value of the parameter.

Optional parenthesis around parameter – No need to declare a single parameter in parenthesis. For multiple parameters, parentheses are required.

Optional curly braces – No need to use curly braces in expression body if the body contains a single statement.

Optional return keyword – The compiler automatically returns the value if the body has a single expression to return the value. Curly braces are required to indicate that expression returns a value.

METHOD REFERENCES

Method references help to point to methods by their names. A method reference is described using `:: doublecolon` symbol. A method reference can be used to point the following types of methods –

Static methods

Instance methods

Constructors using new operator `TreeSet:: new`

Method Reference Example

Let's look into an example of method referencing to get a more clear picture. Write the following program in an code editor and match the results

```
import java.util.List;
import java.util.ArrayList;
public class Java8Tester {
    public static void main(String args[]){
        List names = new ArrayList();
        names.add("Mahesh");
        names.add("Suresh");
        names.add("Ram esh");
        names.add("Naresh");
        names.add("Kalpesh");
        names.forEach(System.out::println);
    }
}
```