UNIT-I: Introduction to JAVA

Introduction:

- JAVA is general purpose OOP (Pure OOP, since supports almost all OOP concepts) language developed by "Sun Microsystems" of USA in1991
- 'James Gosling' was the inventor (Creator) of JAVA language.
- Originally or firstly JAVA was named as "Oak" (Oak is name of tree which was found in front of Goslings Office)
- Basically, JAVA was designed for the development of software's for electronic devices like TV's, VCR's, set-top box, Toasters etc.
- Java runs on a variety of platforms, such as Windows, Mac OS, and the various versions of UNIX.
- The 'C' and 'C++' languages had limitations in terms of reliability and portability therefore they modeled their new language JAVA to overcome the drawbacks of 'C' and 'C++'. Thus, JAVA made really simple, reliable, portable and powerful language.

History OR Evolution of JAVA:

Following table shows some milestones happen in developing of JAVA language:

Year	Development
1990	Sun Microsystem decided to develop special software that could be used to manipulate with electronic devices. A team of Sun Microsystems programmer headed by James Gosling was formed to undertake this task.
1991	After exploring the possibility of such idea, the team announced a new programming language called 'Oak' (Oak was the first name for JAVA)
1992	In this year, team of Sun Microsystems actual implements there language in home appliances like Microwave Oven etc. with tiny touch-sensitive screen.
1993	In this year, team of Sun Microsystems came up with new idea to develop web based application that could run on all types of computers connected to Internet. For that, they creates 'applet' (tiny program run on Internet by the browser)
1994	In this year, team of Sun Microsystems developed web browser called "HotJava" to locate and run applet on Internet.
1995	"Oak" was renamed as "JAVA" due to some legal snags (problems). Also, many popular companies like Netscape and Microsoft announced to support for JAVA
1996	Sun Microsystem releases Java Development Kit 1.0 (JDK 1.0) to develop different kinds of software.
1997	Sun Microsystem releases Java Development Kit 1.1 (JDK 1.1)
1998	Sun Microsystem releases JAVA 2 with JDK 1.2 of Software Development Kit (SDK 1.2).
1999	Sun Microsystem releases standard Edition of Java which was called J2SE(Java 2 Standard Edition) and J2EE (Java 2 Enterprise Edition)
2000	J2SE with SDK 1.3 was released
2002	J2SE with SDK 1.4 was released
2004	J2SE with JDK 5 (JDK 1.5) was released

➢ The latest version of Java is Java 17 or JDK 17 released on September, 14th 2021 and Java is now under administration of Oracle organization.

Features or Characteristics or Advantages of Java:

- **Compiled & Interpreted:**
  - ➢ Usually, programming language is either compiled or interpreted. But Java combines both approaches that make Java 'two-stage system'.
  - ➢ In case of Java, First Java compiler translates or converts Java source program into 'byte code' (Byte code is not machine instruction code & byte code file having extension '.class')
  - ➢ After compilation, Java Interpreter executes this byte code & thus we got our desired output.

    Thus, we can say that Java is Compiled & Interpreted language.
- **Object Oriented:**
  - ➢ Java is pure object oriented language that supports for all OOP's concepts.
  - ➢ Almost, In Java, everything is an Object. All data and methods are resided (exist in) within an object and classes.
  - ➢ The object model in Java is easy to extend because it supports for Inheritance concept.
- **Platform independent and Portable:**
  - ➢ Portable: We know that, after compilation of Java source program it produce ".class" file i.e. byte code which is not machine dependent that's why such file is easily moved or transferred from one computer to another computer and hence Java is Portable.
  - ➢ Platform independent: After generation of byte code (.class file), this byte code is easily interpreted or executed on different kinds of computers having different platforms (Computers having different Operating system like windows, Linux, Mac OS etc and different processors etc).
- **Simple:**
  - ➢ Java is designed in such a way that it would be easy to learn since, most of syntax of java is same as C and C++.
  - ➢ If you understand the basic concepts of OOP then it is easy to implement in Java language.
- **Secure:**
  - ➢ We know that, most of viruses are attacked on files having extension '.exe', '.doc', '.gif', '.mpg' etc. but after compilation of Java source program it produce ".class" file i.e. byte code and which is virus free. And hence, Java enables us to develop virus-free, tamper -free systems.
- **Architectural-neutral:**
  - ➢ Java compiler generates an architecture-neutral class file format which makes the compiled code to be executable on many processors, with the presence of Java runtime system.
- **Robust:**
  - ➢ Java is strict type checking language which checks an error at both time i.e. at compile time and also at run time of program.
  - ➢ Due to this ability of checking errors at run time (exception Handling), we can eliminates any risk of crashing the system using Java.
- **Multithreaded:**
  - ➢ Multithreaded means handling multiple tasks (jobs) simultaneously (at one time).
  - ➢ Java supports for multithreaded programs that means we need not wait for the application to finish its task before beginning another.
  - ➢ That is using Java, we can run multiple java applications without waiting to finish another.
- **Distributed:**
  - ➢ Java enables us to make such applications that can open and access remotely over the internet or network.
  - ➢ That is, multiple programmers at multiple remote locations are capable to work together on single project. That's why Java is distributed.
- **Dynamic and Extensible:**
  - ➢ Dynamic: Java is dynamic language which is capable to link new class libraries, methods and objects dynamically.
  - ➢ Extensible: Java supports to write functions in C or C++ language such functions are called "native methods" and then we can add or link these methods with Java such that they can be used in many applications.

- **Ability to Deal with Database:**
  - Java supports for JDBC (Java Database Connectivity) to send & retrieve data in tabular format with the database thus with the help of Java we are able to deal with database.
- **Automatic Memory Management:**
  - We know that ‘memory’ is very important issue while dealing with computer and we have to manage it very efficient manner.
  - Java language supports for ‘Garbage Collector’ that automatically manages all the memory in efficient manner.

Limitations or Disadvantages of Java:

- **Slow language:**
  As compared to C and C++ languages, Java language compiler took much more time to compile the program & also Java interpreter took much more time to interpret the program that’s why Java is slow language.
- **Strict type checking language:**
  Due to strict type checking, Java language checks much run time errors & that’s why Java application took much time to execute.
- **Case sensitive language:**
  Due to case sensitive language, we must have to write correct spelling of inbuilt methods, classes, interfaces etc. while doing programming.
- Java does not support for Multiple Inheritance but we can implement it by using ‘interface’.

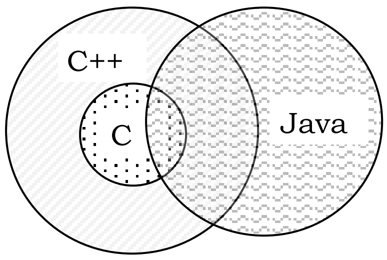
Difference Between C and Java:

‘C’ Language	**Java Language**
1) It is not OOP language.	1) It is pure OOP language.
2) It has preprocessor directive statements like #define, #include etc	2) It has no preprocessor directive statements like #define, #include etc.
3) It does not support for data type ‘class’	3) It does supports for data type ‘class’
4) It has type modifier keywords like auto, extern, register , signed and unsigned.	4) It does not have type modifier keywords like auto, extern, register, signed and unsigned.
5) It supports for ‘pointer’	5) It does not supports for ‘pointer’
6) It supports for data type ‘struct’ and ‘union’	6) It does not supports for data type ‘struct’ and ‘union’
7) It has ‘sizeof’ and ‘typedef’ keywords	7) It has not ‘sizeof’ and ‘typedef’ keywords
8) Automatic memory management is not supported.	8) Automatic memory management is supported by ‘Garbage Collector’.

Difference Between C++ and Java:

‘C++’ Language	**Java Language**
1) It is not pure OOP language.	1) It is pure OOP language.
2) It supports for template classes.	2) It does not support for template classes.
3) It supports for ‘Multiple inheritance’	3) It does not supports for Multiple Inheritance but we implement it using ‘interface’
4) It supports for global variable.	4) It does not have global variable
5) It supports for ‘pointer’	5) It does not supports for ‘pointer’
6) It supports for “destructor”	6) It does not supports for “destructor” but it is replaced by finalize( ) method.
7) It has ‘goto’ statement.	7) It has not ‘goto’ statement.
8) It has preprocessor directive statements like #define, #include etc.	8) It has no preprocessor directive statements like #define, #include etc.
9) It has three access specifiers viz: public, private and protected	9) It has four access specifiers viz: public, Private, protected and default.
10) It supports for operator overloading.	10) It does not supports for operator overloading
11) Automatic memory management is not supported.	11) Automatic memory management is supported by ‘Garbage Collector’.

Following fig. shows overlapping of C, C++ and Java:



From above fig.

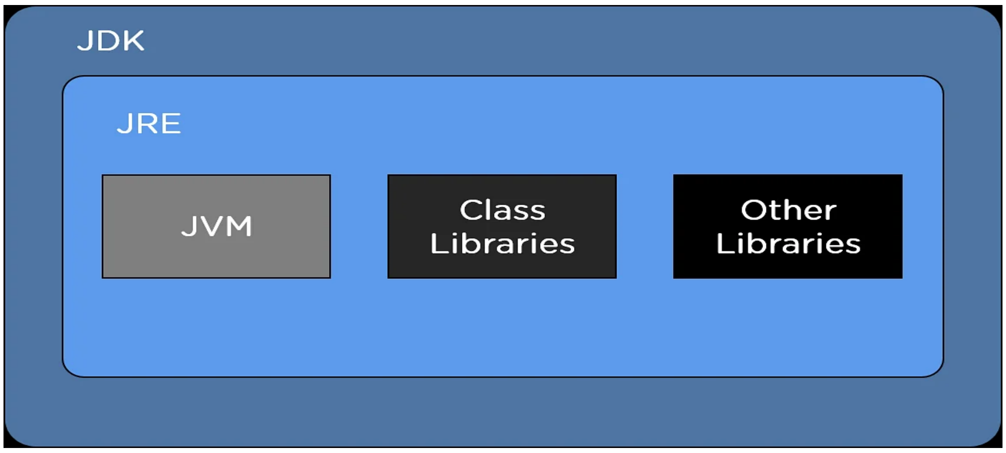
- We know that, ‘C++’ is superset of ‘C’ language therefore every ‘C’ program is easily executed by ‘C++’ compiler.
- But, Java language is partly combination of ‘C’ and ‘C++’ language and it having its own extra features therefore Java can be considered as first cousin of ‘C++’ and second cousin of ‘C’

Java Development Kit (JDK):

- JDK in Java is an essential component necessary to develop programs or software’s using JAVA language.
- It is technically an implementation of either Java Standard Edition or Java Enterprise Edition.
- JDK in Java is an abbreviation for Java Development Kit. It is a bundle of software development tools and supporting libraries combined with the Java Runtime Environment (JRE) and Java Virtual Machine (JVM).
- JSL (Java Standard Library) also called as Java API (Application Programming Interface) is the main part of JDK that contains thousands of Packages.
- Further, Packages contains thousands of classes, methods, interfaces etc.

The Architecture of JDK in Java:

- The architecture of JDK in Java includes the following modules as described in the image below.

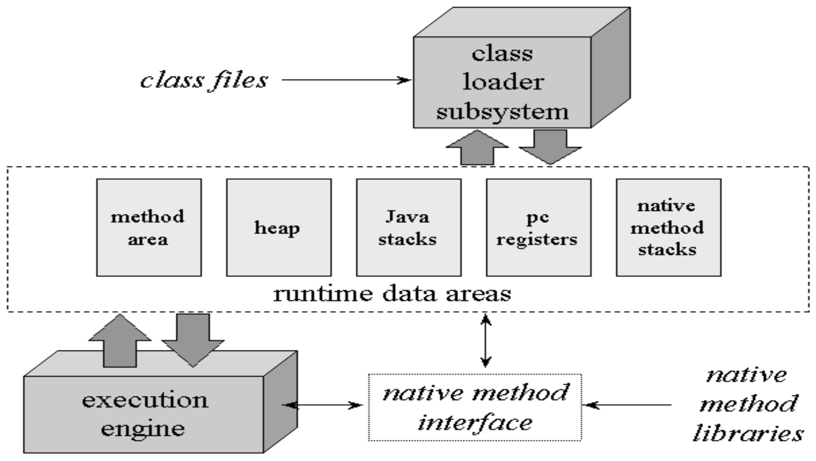


- The two vital software modules of JDK are:

1) JVM (Java Virtual Machine):

- Java Virtual Machine is a software tool responsible for creating a run-time environment for the Java source code to run. The very powerful feature of Java, "Write once and run anywhere," is made possible by JVM.
- The JVM stays right on top of the host operating system and process the Byte Code (machine language), such that it would easily executed by microprocessor.
- Java Virtual Machine plays vital or important role in execution of java program therefore it is heart of java.

Following Figure shows Architecture of JVM:



Note:

- *Java*
  Compiler generates or creates *byte code which is machine independent or platform independent* *therefore it is easily interpreted by any JVM that's why it is called as "write once run anywhere".*
- *But,* *JVM is platform dependent* *i.e. windows, Linux, Mac OS, Unix etc. operating system having their different-different JVM's.*

Working of JVM:

- First of all, java source file (.java file) is converted into byte code (.class file) by the java compiler and this byte code file is given to the JVM.
- In JVM, there is one module or program called 'Class loader sub system' which performs following functions:
  - First, 'Class loader sub system' loads the '.class' file into memory.
  - Then it verifies whether all byte code instructions are proper or not.
  - If it finds some problem in byte code then it immediately terminates the execution.
  - If byte code is proper then it allocates necessary memory to execute the program.

Also, this memory is divided into 5 parts called 'Runtime data area' & these parts as follows:

1) Method area:
   In this memory area, all class code, variables codes, methods codes etc. are stored.
2) Heap:
   In this memory area, all objects are created and stored.
3) Java Stacks:
   Actually, java methods are stored in 'Method area' but actual execution of such java methods are happen under 'Java stacks' area.
4) PC registers:
   This area contains the memory addresses of instructions of the methods.
5) Native method stacks:
   All native methods (C, C++ functions) are executed under native methods stacks.
   And all native methods are connected with JVM by '*native method interfaces'*

After, allocation of memory into corresponding parts then it comes towards 'Execution Engine'.

- Execution Engine can consists of two things VIZ:
  1) Interpreter 2) JIT (Just In Time) compiler.
- This interpreter and JIT compiler are responsible for converting byte code into machine instruction such that it easily executed by microprocessor.
- After, loading the ".class" file into memory, JVM first identifies which code is to be left to interpreter and which one to JIT compiler so that the performance is better. The blocks of code allocated for JIT compiler are also called 'hotspots'. Thus, the interpreter and JIT compiler will work simultaneously to translate the byte code into machine instructions.

Note that: JIT compiler is a part of JVM which increases execution speed of program.

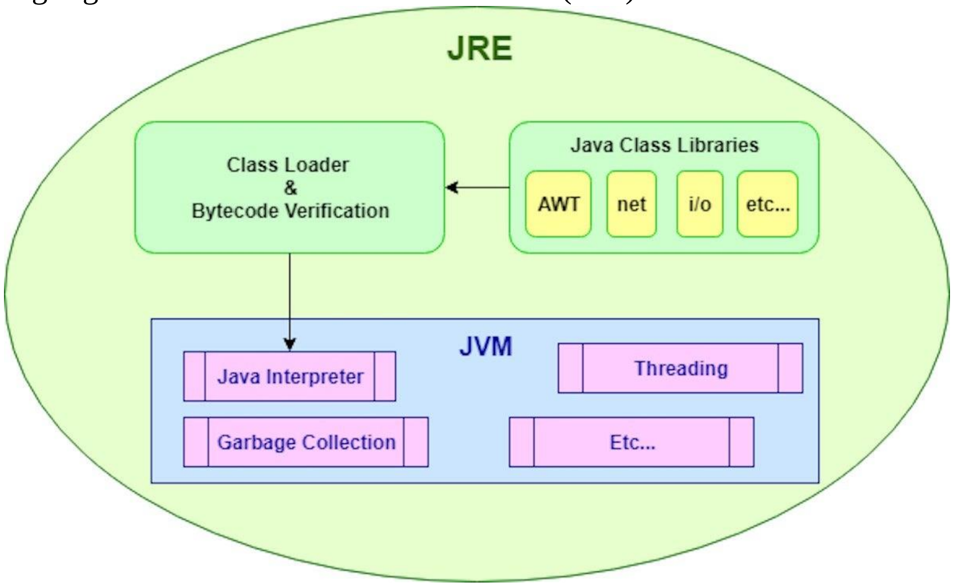
2) JRE (Java Run-time Environment)

- Java Run-time Environment is a software platform where all the Java Source codes are executed.
- JRE is responsible for integrating the software plugins, jar files, and support libraries necessary for the source code to run.
- The Java Runtime Environment, or JRE, is a software layer that runs on top of a computer's operating system software and provides the class libraries and other resources that a specific Java program needs to run.
- A Java™ runtime environment (JRE) is a set of components to create and run a Java application.
- A JRE is part of a Java development kit (JDK).
- A JRE is made up of a Java virtual machine (JVM), Java class libraries, and the Java class loader.
- In short JDKs are used to develop Java software whereas JREs provide programming tools and deployment technologies.

Why use a Java runtime environment?

- In order for software to execute a program, it needs an environment to run in—usually an operating system (OS) like Linux, Unix, Microsoft Windows, or MacOS.
- Because of JRE, java programs are constrained the capabilities of the OS and its resources (such as memory and program files).
- A JRE acts as facilitator or interface between the Java program and the OS that demands resources towards the OS.

Following Fig. shows Java Runtime Environment (JRE):



JDK Components (JDK tools):

Following is the list of tools or components of JDK which are used to develop and run the java programs:

Sr.NO	Tool or Component	Description or Use
1	javac (Java Compiler)	It translates or converts java source program into byte code file & that file understood by java interpreter
2	java (java Interpreter)	It runs java applications by reading corresponding byte code file & gives result.
3	Appletviewer	It runs or views java applets onto the web browser.
4	javap (Java disassembler)	It converts byte code file into program description
5	Javadoc	It creates or produces HTML format documentation of java source file. But it needs public class for documentation.
6	Javah	It creates or produces header files for use of native methods.
7	jdb (Java Debugger)	It helps us to checks errors in java program.

Why Java does not support for pointer?

→

1) We know that 'pointers' are used to hold memory address. And most of viruses are trying to attack on memory that's why Java does not support for pointer and hence Java is secured.

2) Also, pointers are helpful for dynamic memory allocation i.e. it is used for run time memory management, but in Java all memory management is automatically done by 'Garbage collector' that's why not need of pointer.

Structure of JAVA Program:

- We know that single Java program may contains multiple classes but out of these classes, one class should be public and that class contains main( ) method from which JVM interprets the byte code.
- Note that: Java is pure OOP language i.e. all programs must have classes and objects.
- A typical Java Program is divided into several Sections which are shown in following figure:

Documentation Section
Package statement Section
Import statement Section
Interface statement Section
Class definition section

main( ) method Section
{
// main() definition
}

1) Documentation Section:

- This section contains set of comments lines showing details of java source program such as program name, programmer name, date of program, version etc. this help program readability.

In Java, we can give comments by three ways VIZ:-

1) Single line comment:

- If we have to specify general information of program within single line then single line comment is used. Single line comment is given by // notation.
- Also, we can specify this comment anywhere in program.

E.g. // Program Name= Addition of two numbers.

2) Multiline comment:

- If we have to specify general information of program within multiple lines then multi line comment is used. Multiline comment is given by following notation.

```
/* ----------
   ----------
---------- */
```

E.g.

```
/*      Program Name:   Multiplication
          Programmer: James Gosling */
```

3) Third Style comment: (Java documentation Comment)

- This type of comment is specially used for documentation purpose.
- If we specify description using 'Third style comment' then it is shown in HTML files created by using 'Javadoc'
- This comment is used to provide description for every feature in Java source program.

Third Style comment is given by-

```
                                        /**   ----------------------
                                              ----------------------   */
```

E.g.

```
/**  This class is used for addition   */
public     class       add
{
       /**  This method is used for addition   */
         public    void  addition( )
         {
                 // statements
         }
}
```

- In above example, two times documentation comment is used that will show description of class 'add' and description of method 'addition( )' in HTML file. Note that: For generation of HTML documentation of java source program using 'javadoc' component, class and method should be public or protected.

2) <u>Package statement Section:</u>

- This section is used to declare our own package. When we declare own package then it informs to the java compiler to link all classes of our package with java source program.
- Syntax to specify package statement:

```
package       package_name;
```

E.g.

```
package      student;
```

More about package will be discussed in next chapter.

<u>3) import statement Section:</u>

- In this section we can import existing package in our java source program.
- We know that, in case of 'C' language if we have to use printf( ) method then we include 'stdio.h' header file using preprocessor directive '#include'.
- Similarly, if we have to use existing classes or exiting methods of JSL (Java Standard Library) then we have to import that package in our source program using 'import' statement.
- Syntax to import package in program:

```
import           package_name;
```

E.g.

```
import           java.lang.* ;
```

<u>Difference between #include & import:</u>
→
- ❖ *When we include header file in program then C/C++ compiler goes to the standard library (it is available at c:\tc\lib) and searches for included header file there. When it finds the header file, <u>it copies entire header file</u> into the program where the #include statement is written. Thus, if our C/C++ program has only 10 lines still C/C++ compiler shows hundreds of line compiled this is due to copy of included header file at #include statement. Therefore our program size increases & hence it causes memory wastage.*
- ❖ *When we import package in Java program then JVM checks whether imported package is present in JSL or not. If JVM finds imported package then it executes corresponding method code there and only returns its result to source program therefore size of source program in not increased as happened in C/C++. And hence, memory wastage is solved.*

<u>4) interface statement Section:</u>

- In this section we can define interfaces.
- Interfaces are similar to the classes but all methods of interface are by default 'abstract.
- This is optional section, used while implementing multiple inheritance in java.

5) class definition Section:

- We know that single Java program may contain multiple classes and every class has its own attributes (data members) and methods. Such type of classes can be defined under class definition section.

  Note that:

  - *We know that single Java program may contains multiple classes but out of these classes, one class should be* *public* *and that class contains main( ) method from which JVM interprets the byte code.*

6) main( ) method Section:

- We know that in case of C/C++, main( ) function is compulsory from which execution of program starts. Like that java program also have main( ) method from which JVM starts program interpretation. This is compulsory section. Also, main( ) method in Java must be public. If we made main( ) as private or protected then it is not assessable for JVM also.
- main( ) method should be defined under any class of program but that class should be public.

Simple Java Program:

Let's consider following simple java program;

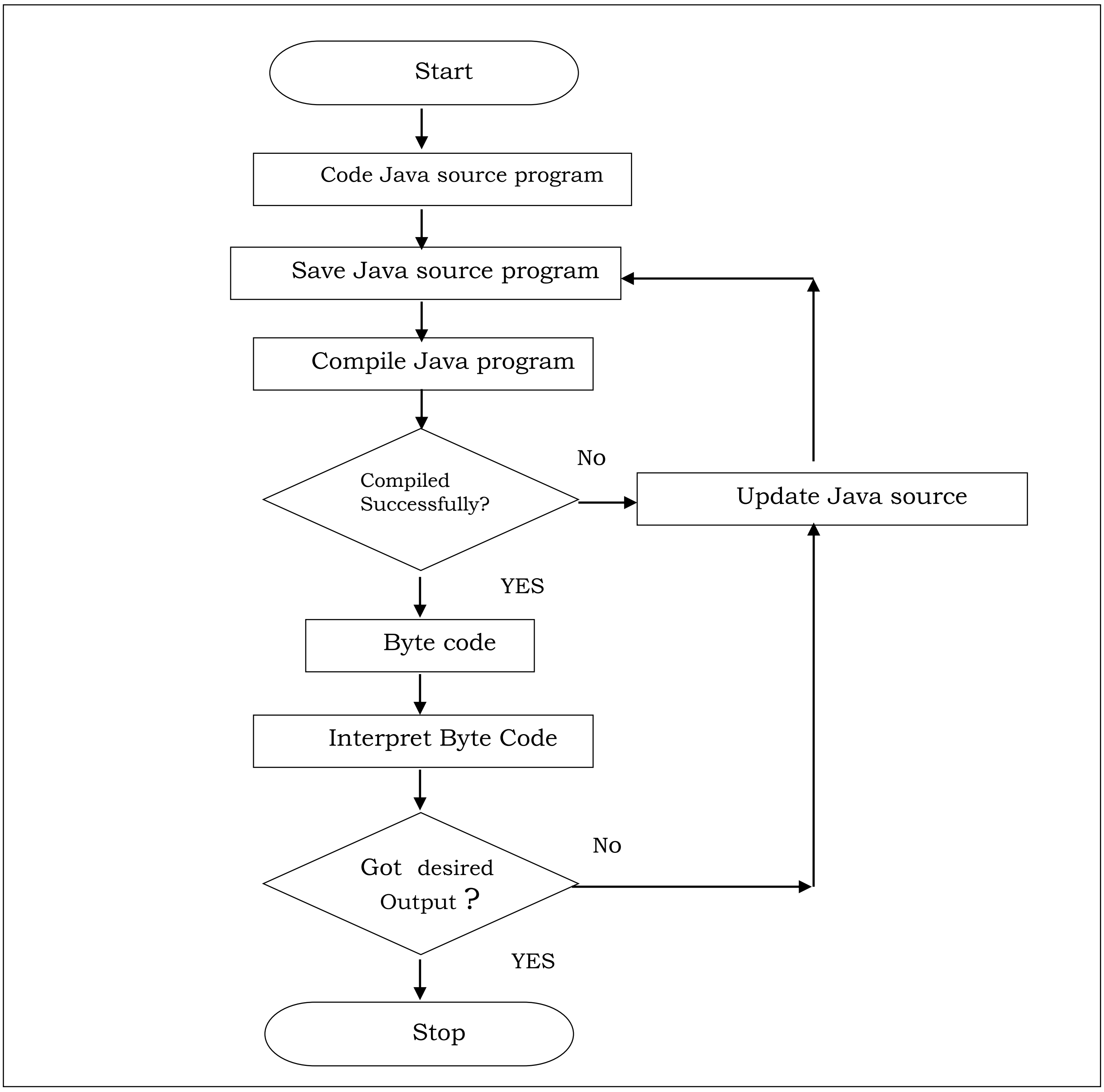
```
import    java.lang.*;
class              first
{
    public   static void   main(String   args[ ])
    {
            System.out.println("Welcome in JAVA programming");
     }
}
```

In above program,

- "java.lang" is a package which is imported using 'import' keyword. This package contains lots of inbuilt classes such as System, String, Integer, Float etc. This is default package i.e. there is no necessary to import it.
- Here, main( ) method is compulsory which is declared as public, static and void
  - It is public because it made available for JVM for interpretation of java program.
  - It is static because it should be called without any object; it is invoked by JVM with class name.
  - It is void because it does not return any value.
- Also, main( ) method accepts array of string as argument which is called as command line argument. The passed values are stored in args[ ] array at individual indices.
- System.out.println( ) statement:
  - "System" is inbuilt wrapper class which was found under 'java.lang' package.
  - "out" is object of 'System' class which is 'static' & hence it is accessed by 'System' class name.
  - "println( )" is a method was found in "System" class used to display output and called by using "out" object.

Steps to execute Java Program:

Following flowchart shows compiling and interpreting Java program;



Syntax to Compile Java Source program:

Java program is compiled with 'java source program' name along with 'javac' component which is given as fallow:

```
javac        JavaSourcePgmName.java ↵
```

E.g. Consider, we have *'good.java'* source program then we can compile it as follow:

```
javac        good.java ↵
```

If *'good.java'* program have one class named 'good' then *'good. Class'* byte code is generated.

Syntax to interpret or Run or Execute the Byte code:

Java program is interpreted or run or execute using byte code (.class file) along with 'java' interpreter which is given as fallow:

```
java    ByteodeFile ↵
```

E.g. Consider, we have *'good.class'* byte code then it is interpreted as follow:

```
java    good ↵
```

Note that: *After compilation of java source program, byte code (.class file) is generated. And then JVM interpret that byte code and we got our result.*

Syntax to pass arguments to main( ) method while interpretation of bytecode :

We can also pass some string type arguments to main ( ) method called 'command line arguments' using following syntax.

```
java    ByteCodeFile    arg1    arg2  - - - -   argN ↵
```

In above syntax;
arg1, arg2, - - - - ,argN are the command line arguments passed to main() method while interpreting.
Note that: All passed arguments are stored in formal parameter (String type array) of main( ) method at individual indices.
E.g. Consider following Program:

```
import    java.lang.*;
class            first
{
    public   static void   main(String   args[ ])
    {

            System.out.println("FirstName= "+ args[0]);
            System.out.println("MiddleName= "+ args[1]);
            System.out.println("LastName= "+ args[2]);

    }
}
```

OUTPUT:

```
            javac     first.java
                                 ↵
            java     first          SACHIN     RAMESH     SHINDE
```

In above example; three command line arguments are passed to main( ) method. They are SACHIN RAMESH SHINDE.
All these arguments are stored in 'args' String type array in main( ) method at individual indices as fallow;

args[0]=>	SACHIN
args[1]=>	RAMESH
args[2]=>	SHINDE

Also, we use '+' operator to concatenates two strings with each other.

Naming Conventions in Java:

- *Naming Conventions* specify the rules to be followed by java programmer while writing or coding java source program.
- We know that java program contains the package, classes, interfaces, methods, variables etc. and all these have separate naming conventions they are as follow:

Naming Conventions for Package:

- We know that, Package is one kind of directory that contains the classes and interfaces.
- Package name in java should write in small letters only.

Example:

java.lang
java.awt
javax.swing

Naming Conventions for class or interface:

- We know that, class is model for creating object.
- Class specifies the properties and action for objects.
- An interface is similar to class but it has abstract methods only.
- Class and interface name in java should start with capital letter.

Example:

System
String
Integer
Float etc.

Naming Conventions for methods:

- We know that, methods contain the executable statements or instructions after execution it produce desired result.
- The first word of a method name is in small letters, then from second word onwards, each new word starts with capital letter as:
  Example:

  println();
  readLine();
  getNumberInstance();

Naming Conventions for variables:

- Naming conventions for variable is same as that of methods i.e. *The first word of a variable name is in small letters, then from second word onwards, each new word starts with capital letter as:*
  Example: age
  empName
  empNetSal

Java Tokens:

- "Token is nothing but smallest individual unit of java source program."
- We know that Java is pure OOP language i.e. every program has classes and every classes has some methods and methods contains executable statement and every executable statement contains the tokens i.e. statements are made up of several tokens.
- Following are the several tokens in Java program:
  1) Keywords 2) Data type 3) Identifier 4) Variable
  5) Constant or Literals 6) Operators 7) Special symbols.

Let us see all tokens in details:

1) Keywords (Reserve words):

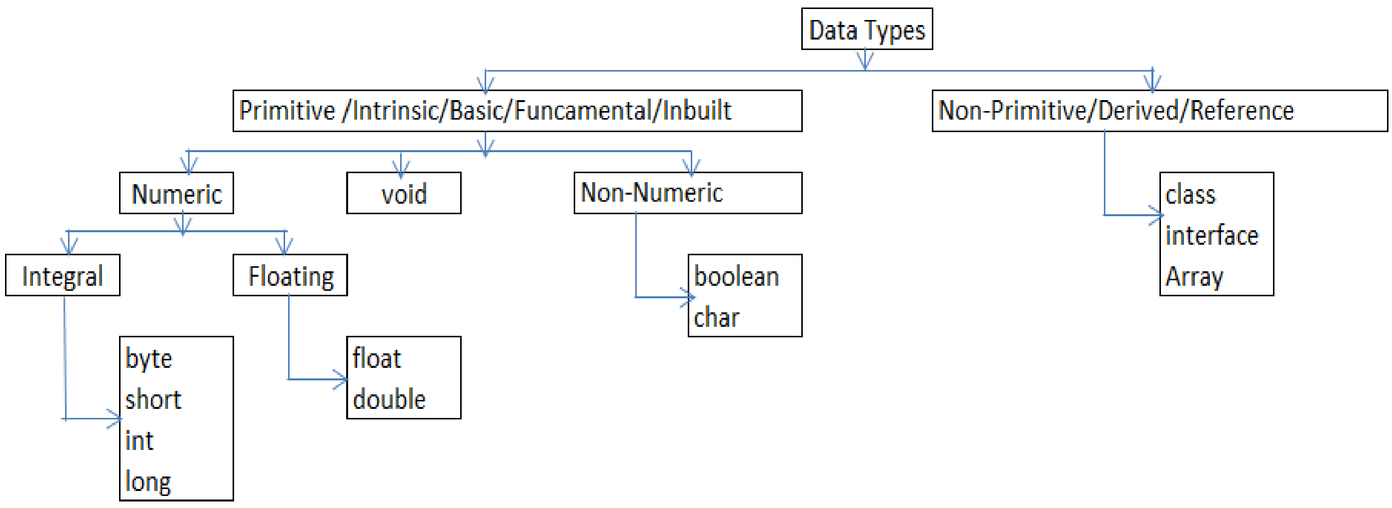
- The words whose meaning is already known by java compiler are called as 'Keywords'.
- These words having fix meaning and we are not able to change that meaning therefore they are also called as 'Reserve Word'.
- Java language contains more than 50 keywords and they are listed as fallow:

Abstract	Continue	For	new	switch
Assert	Default		package	synchronized
Boolean	Do	If	private	this
Break	Double	implements	protected	throw
Byte	Else	import	public	throws
Case	Enum	instanceof	return	transient
Catch	extends	Int	short	try
Char	Final	interface	static	void
Class	finally	long	strictfp	volatile
	Float	native	Super	while

2) Data Type:

- Data: "Data is nothing but collection of raw information or unprocessed information that we provide for the computer for processing"
  e.g. numbers, string, alphanumeric etc.
- Data Type:
- Concept: When we give data to the computer for processing at that time compiler does not know which type of input data is.

Generally, Data types are used to tell the compiler which type of input data is.



- *Definition*: "Type of Data is called as Data Type"

Following tree diagram shows data types in Java language:
Let us see all these data types in details:

Primitive Data Types:

There are nine primitive data types supported by Java. Primitive data types are predefined by the language and named by a keyword.

1) byte:

- byte data type is an 8-bit(1 byte) integral data type.
- Its Minimum range value is -128 (i.e. -2^7)
- Its Maximum range value is 127 (inclusive)(i.e. 2^7 -1)
- Its Default value is 0
- byte data type is used to save space in large arrays, mainly in place of integers, since a byte is four times smaller than an int.
- Example: byte a = 100; byte b = -50;

2) short:

- short data type is a 16-bit (2 bytes) integral.
- Its Minimum range value is -32,768 (i.e. -2^15)
- Its Maximum value is 32,767 (inclusive) (i.e. 2^15 -1)

- Short data type can also be used to save memory as int data type. A short is 2 times smaller than an int
- Its Default value is 0.
- Example: short   s = 10000, r = -20000;

3) int:

- int data type is a 32-bit (4 bytes) signed integral data type.
- Its Minimum range value is - 2,147,483,648.(i.e. -2^31)
- Its Maximum range value is 2,147,483,647(inclusive).(i.e. 2^31 -1)
- int is generally used as the default data type for integral values unless there is a concern about memory.
- Its default value is 0.
- Example: int a = 100000, b = -200000;

4) long:

- long data type is a 64-bit (8 bytes)signed integral data type.
- Its Minimum range value is -9,223,372,036,854,775,808.(i.e. -2^63)
- Its Maximum range value is 9,223,372,036,854,775,807 (inclusive). (i.e. 2^63 -1)
- This type is used when a wider range than *int* is needed.
- Its Default value is 0L.
- Example: long a = 100000L,   int b = -200000L;

5) float:

- float data type is a single-precision 32-bit (4 bytes) floating data type.
- Its Minimum range value is - $3.4e^{38}$ to $-1.4e^{-45}$ for negative value.
- Its Maximum range value is $3.4e^{38}$ to $1.4e^{-45}$ for positive value.
- Float is mainly used to save memory in large arrays of floating point numbers.
- Its default value is 0.0f.
- Float data type is never used for precise values such as currency.
- Example: float   f1 = 234.5f;

6) double:

- double data type is a double-precision 64-bit (8 bytes)floating data type.
- Its Minimum range value is $-1.8e^{308}$ to $-4.9e^{324}$ for negative value.
- Its Maximum range value is $1.8e^{308}$ to $4.9e^{324}$ for positive value.
- This data type is generally used as the default data type for decimal values, generally the default choice.
- Double data type should never be used for precise values such as currency.
- Its Default value is 0.0d.
- Example: double   d1 = 123.4;

7) boolean:

- boolean data type represents *one bit* of information.
- There are only two possible values: *true and false*.
- This data type is used for simple flags that track true/false conditions.
- Its default value is *false*.
- Example: boolean   one = true;

8) char:

- char data type is a single 16-bit (2 bytes) non-numeric character data type.
- Its Minimum value is '\u0000' (or 0).
- Its Maximum value is '\uffff' (or 65,535 inclusive).
- Char data type is used to store any character.
- Example: char letter ='A';

9) void:

- void means no value
- This data type is generally used to specify return type of method.
- If return type of method is void then that method does not return any value.

Non-Primitive or Derived or Reference Data Types:

- The data types derived or created with the help of inbuilt of data types is called 'Non-primitive or derived or reference data types"
- Java language has three Non-primitive data types viz. array, class and interface.
- Note- In next units we will discuss above mentioned non-primitive data types.

3) Identifier:

- "Identifier is the name given by the programmer for any variable, package, class, interface, array, object etc."
- There are several rules to declare or define the identifier:
    1) Identifier should not be keyword.
    2) Identifier should not start with digit.
    3) Identifier can be combination of alphabets, digits or **underscore or dollar sign($)**.
    4) Identifier should not contain special symbol except underscore and **dollar sign($)** .
    5) Identifier should not contain any white space character like horizontal tab, vertical tab, new line etc.
    6) Identifier should be meaningful.
    7) Identifier can be of any length.

4) Variable:

- "Variable is the name given to the memory location where the data is stored such quantity is called as Variable"

OR

- "The quantity that changes during program execution is called as Variable"
- Concept: The main concept behind variable is that <u>every variable has an ability to store the data.</u>

<u>Syntax to declare variable:</u>

```
DataType       variableName ;
```

Here;

DataType is any valid data type in 'Java' language.
variableName is an identifier.

Example:
```
int     rollno;
char   x;
```

- There are several rules to declare the variable:

1) Variable should not be keyword.
2) Variable should not start with digit.
3) Variable can be combination of alphabets, digits or underscore **or dollar sign($)**.
4) Variable should not contain special symbol **except underscore and dollar sign**.
5) Variable should not contain any white space character like horizontal tab, vertical tab, new line etc.
6) Variable should be meaningful.
7) Variable can be of any length.
8) Declared *local variable* <u>must</u> be initialized anywhere in block.

<u>Types of Variables in java:</u>

- **Local variables:**
    - ➢ The variables which are declared inside methods, constructors or blocks are called <u>local variables</u>.
    - ➢ These variables are declared and initialized within the method and they will be destroyed automatically when the method has completed its execution.
- **Instance variables:**
    - ➢ Instance variables are variables which are declared 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 but not accessed within static method *directly*.
    - ➢ These variables are in the scope of object.
- **Class variables:**
    - ➢ Class variables are variables which are declared within a class but outside any method and declared with the static keyword.
    - ➢ These types of variables are common to all objects of class i.e. all static data are shared among all objects of class commonly.
    - ➢ Note: Such class variables are not in the scope of object.

Following program shows variables in Java-

```
public    class     first
{
                  int        a;              //  instance variable
                  static  float     b;   // class variable
     public   static void main(String     []arg)
     {
              boolean    flag;     //local variable
      }
}
```

5) Constant (Literals):

- A *literal* represent a fixed value that is stored into variable directly in the program. They are represented directly in the code without any computation.
- Literals can be assigned to any primitive type variable.
  For example:
    - byte   p = 68;
    - char a = 'A';

Java has different types of literals VIZ:
1) Integer Literals
2) String Literals
3) Character Literals
4) Float Literals
5) Boolean Literals

Let us see all literals in details:

1) Integer Literals:

- Integer literals represent the fixed integer values like 23, 78, 658, -745 etc.
- The data type byte, int, long, short belongs to decimal number system that uses 10 digits (from 0 to 9) or octal number system that uses 8 digits (from 0 to 7) or hexadecimal number system that uses 16 digits (from 0 to F) to represent any number.

Note that:
Prefix 0 is used to indicate octal and prefix 0x indicates hexadecimal when using these number systems for literals.

For example:
- int   decimal = 100;
- int   octal = 0144;
- int   hexa = 0x64;

2) String Literals:

- String literals are collection of characters which are representing in between a pair of double quotes.

Example:
```
String     x="Hello World";
```

3) Character Literals:

- Character literals are characters which are representing in between a pair of single quotes.
- Character literals are like ‘A’ to ‘Z’, ‘a’ to ‘z’, ‘0’ to ‘9’or Unicode character like ‘\u0042’ or

escape sequence like '\n', '\b' etc.

Example:

```
char    x= ' Z';
```

4) Float Literals:

- Float literals represents fractional values like 2.3, 86.58, 0.0, -74.5 etc.
- These types of literals are used with float and double data types.
- While writing such literals, we can use E or e for scientific notation, F or f for float literal and D or d for double literals (this is default and generally omitted)

Example:

```
float      p = 9.26;
double   q = 1.56e3;
float      m =986.8f;
```

5) Boolean Literals:

- Float literals represents only two values – true or false. It means we can store either 'true' or 'false' into a Boolean type variable

Example:

```
boolean      p =true;
```

6) Operators:

An '*operator*' is a symbol that tells computer to perform specific task.

OR

An 'Operator' is a symbol that operates onto the operand to perform specific task.

Following are the several operators present in Java:

1) Arithmetic operators
2) Relational operators
3) Logical operators
4) Increment and decrement operator
5) Assignment operator
6) conditional operator
7) Bitwise operators
8) 'new' operator
9) 'instanceof' operator
10) cast operator

(Note that: All the operators listed above from 1 to 6 are same as C/C++ language therefore refer notes of C/C++ language)

Let's see some operators of Java language as follows:

7) Bitwise operators:

- Bitwise operators are used to manipulate data at bit (0 or 1) level.
- These operators act on individual bits of the operands.
- Bitwise operators only act on integral data types such as byte, int, short, long. That is they are not worked on float and double data type.
- When these operators work on data then internally (automatically) data is converted into binary format & then they start their working.
- There are 7 different bitwise operators present in Java as follows:

Operator	Meaning
&	Bitwise AND
|	Bitwise OR
^	Bitwise exclusive OR (i.e. XOR)
~	Bitwise Complement
<<	Bitwise left Shift
>>	Bitwise Right Shift
>>>	Bitwise Zero fill Right shift

The truth table or working of bitwise operator &, | and ^ is shown in following table:

Op1	**Op2**	**Op1 & Op2**	**Op1 | Op2**	**Op1 ^ Op2**
1	1	1	1	0
1	0	0	1	1
0	1	0	1	1
0	0	0	0	0

<u>Bitwise AND operator (&):</u>

This operator performs 'AND' operation on individual bits of the numbers. To understand the working of *'&'* operator see following example.

E.g.:

1) 22&5

→

		128	**64**	**32**	**16**	**8**	**4**	**2**	**1**
22	→	0	0	0	1	0	1	1	0
5	→	0	0	0	0	0	1	0	1
22&5	→	**0**	**0**	**0**	**0**	**0**	**1**	**0**	**0**

In above table, '1' bit is found in '4' column only therefore '22&5' gives result **'4'**

<u>Bitwise OR operator (|):</u>

This operator performs 'OR' operation on individual bits of the numbers. To understand the working of '|' operator see following example.

E.g.:

1) 35|7

→

		128	**64**	**32**	**16**	**8**	**4**	**2**	**1**
35	→	0	0	1	0	0	0	1	1
7	→	0	0	0	0	0	1	1	1
35|7	→	**0**	**0**	**1**	**0**	**0**	**1**	**1**	**1**

In above table, '1' bit is found in '1', '2', '4' and '32' columns therefore '35|7' gives result 1+2+4+32= **39**

<u>Bitwise XOR operator (^):</u>

This operator performs 'exclusive OR' operation on individual bits of the numbers. Its symbol is denoted by '^' which is called *cap, carat or circumflex* symbol. To understand the working of '^' operator see following example.

E.g.:1) 47^4

→

		128	**64**	**32**	**16**	**8**	**4**	**2**	**1**
47	→	0	0	1	0	1	1	1	1
4	→	0	0	0	0	0	1	0	0
47^4	→	**0**	**0**	**1**	**0**	**1**	**0**	**1**	**1**

In above table, '1' bit is found in '1', '2', '8' and '32' columns therefore '47^4' gives result 1+2+8+32=**43**

<u>Bitwise complement operator (~):</u>

This operator gives complement form of the given number. Its symbol is denoted by '~' which is called '*tiled'* symbol. To understand the working of *'~'* operator see following example.

E.g.:

1) ~47

→ It gives result= -48

2) ~(-26)

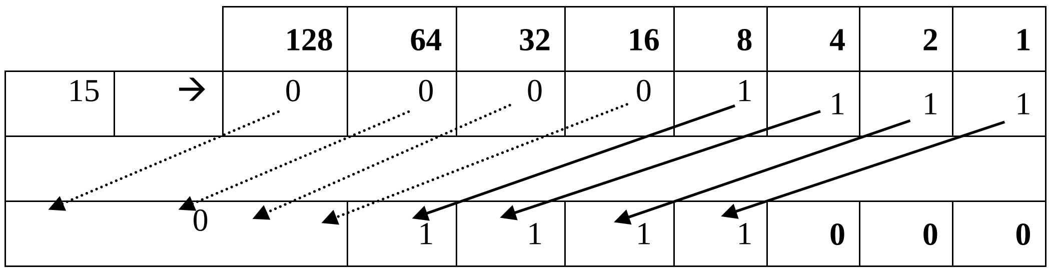
→ It gives result= 25

<u>Bitwise left shift operator (<<):</u>

This operator shifts the bits towards *<u>left side</u>* by a specified number of positions. Its symbol is denoted by '<<' which is called *double less than* symbol. To understand the working of '<<' operator see following example.

E.g.:1) 15<<3

→



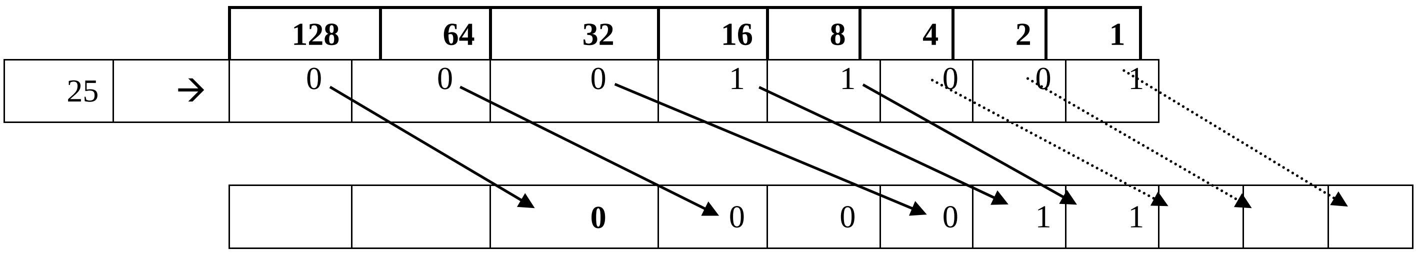
In above table, '1' bit is found in '8', '16', '32' and '64' columns therefore '15<<3' gives result 8+16+32+64=**120**

<u>Bitwise Right shift operator (>>):</u>

This operator shifts the bits towards *<u>right side</u>* by a specified number of positions. Its symbol is denoted by '>>' which is called *double greater than* symbol. To understand the working of '>>' operator see following example.

E.g.:1) 25>>3

→



In above table, '1' bit is found in '1', and '2' columns therefore '25>>3' gives result 1+2=**3**

<u>Bitwise Zero Fill Rightshift operator (>>>):</u>

- This operator also shifts the bits towards *<u>right side</u>* by a specified number of positions. But, it stores '0' in the sign bit. Its symbol is denoted by '>>>' which is called trip*le greater than* symbol. Since, it always fills '0' in the sign bit therefore it is called *<u>zero fill right shift operator</u>*.
- In case of negative numbers, its output will be positive because sign bit is filled with '0'

<u>8) 'new' operator:</u>

- 'new' operator is used to *<u>create object of class</u>*.
- We know that, objects are created on 'heap' memory by JVM dynamically.

<u>Syntax to create object:</u>

```
className    obj=new   className( );
```

Here,

'className' is name of the class.

'obj' is name of created object which is an identifier.

Example: Consider, there is class named 'Employee' then we create its object as follow,

```
Employee      emp = new   Employee( );
```

Here, 'emp' is an object of class 'Employee'

9) ‘instanceof’ operator:

- ‘instanceof’ operator is used *to check created object belongs to particular class or not*.
- Also, this operator used *to check created **reference** belongs to particular interface or not*.

Syntax:

```
boolean     var = obj     instanceof   className;
                               OR
boolean     var=   ref   instanceof     interfaceName;
```

Here,

‘var’ is variable of boolean data type.
‘obj’ is object of class.
‘ref’ is reference of interface.
‘className’ is name of class.
‘interfaceName’ is name of interface.

Example:

```
boolean       x = emp      instanceof    Employee;
```

Here, ‘instanceof’ operator checks an object ‘emp’ is an object of class ‘Employee’ or not.
If ‘emp’ is an object is class ‘Employee’ then ‘instanceof’ operator return *‘true’* otherwise it returns *‘false’*

```
// Program that demonstrate use of ‘instanceof’ operator
class worker
{
}
class   sangola
{
    public static void main(String[] args)
    {
            worker   wk=new   worker();
            boolean  x;
            x=wk   instanceof    worker;
            if(x==true)
                    System.out.println("It is instance of Worker class");
            else
                    System.out.println("It is not instance of Worker class");
    }
}
OUTPUT:          It is instance of Worker class
```

10) ‘cast’ operator:

- cast operator is used to convert one data type into another data type.
- To convert data type of any variable or an expression, just we have to specify conversion data type before variable or expression within simple bracket (braces).

Example:

1) double   x=15.26;
int   y=x;   *//Error- because data type of ‘x’ and ‘y’ are different.*

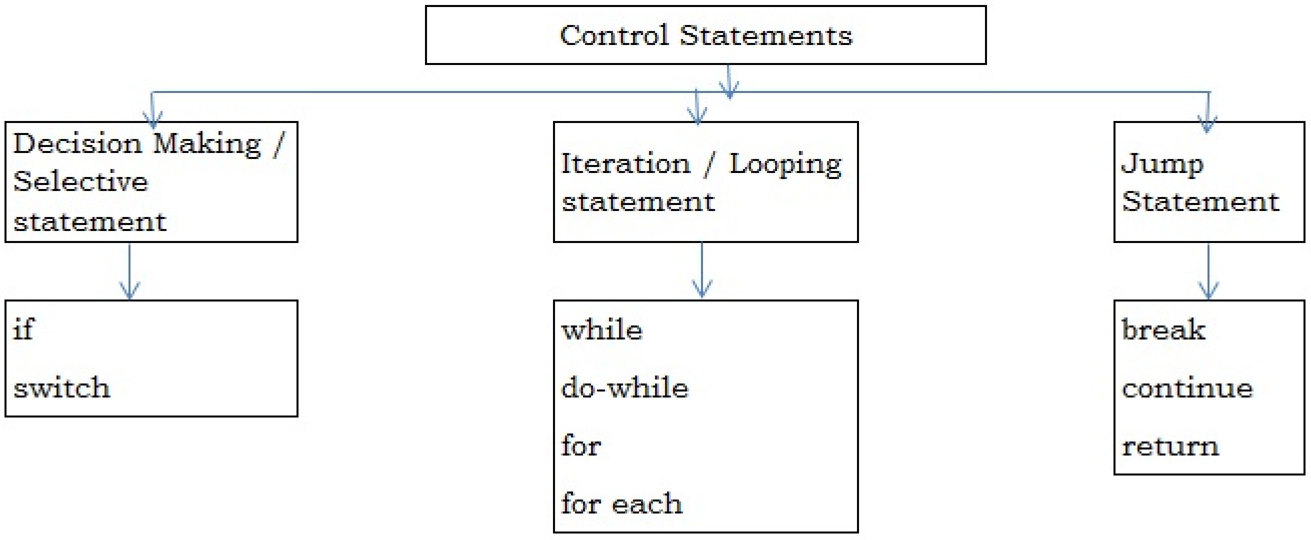
To store value of ‘x’ into ‘y’, we have to convert data type of ‘x’ into data type of ‘y’ as follow,
int y= (int) x;   *//here, the data type of ‘x’ is converted into data type of ‘y’ using **(int) cast operator***

2) int   x=65;
char   y = (char) x;*//here, the data type of ‘x’ is converted into data type of ‘y’ using (char) cast operator*

Control Statement in Java:

The statement that controls the flow of execution of program is called as "Control statement" or "Control Structure".

The following tree diagram shows control statements in Java language:



(Note that: All the <u>Control Statement in Java is same as that of C/C++ language</u> therefore refer notes of C/C++ language)

<u>for-each loop:</u>

- This loop is specially designed to handle elements of 'collection'.
- *Collection* represents a group or set of elements or objects.

For example: We can take an 'array' as collection because 'array is set or group of elements

- Also, any class in '<u>java.util</u>' package can be considered as 'collection' <u>because any class in 'java.util' package handles group of objects such as 'stack', 'vector', 'LinkedList' etc</u>.
- The *for-each* loop repeatedly executes a group of statements for each element of the collection.
- The execution of for-each loop depends upon <u>total number of elements or objects present in the collection</u>.

<u>Syntax:</u>

```
for (datatype   var : collection )
{
    Statements;
}
```

Here,

'var' is an identifier which represents each element of collection one by one. Suppose, the collection has 5 elements then this loop will be executed 5 times and 'var' will *store each element of collection one by one*.

'datatype' is any valid data type in Java which is same as collection.

'collection' is any collection such as array, stack, linked list, vector etc.

```
// Program that demonstrate use of for-each loop
class      myloop
{
    public  static void main(String  [ ] args)
    {
            int     arr[ ]={5,6,7,8,9};

            for (int  i : arr )          //  'i' represents each element of 'arr'
            {
              System.out.println(i);
            }
    }
}
```

‘continue’ statement:

- ‘continue’ statement is specially used in looping statement.
- When ‘continue’ statement is executed then *control transferred back to check the condition in loop* and rest of statements are ignored.

```
// Syntax or execution of ‘continue’  statement

While ( condition1 )   ◄··········
{                                ·
        if ( condition2 )        ·
        {                        ·
          continue;    ···········
        }
            -----------
            -----------
}
```

```
// Program that demonstrate use of ‘continue’ statement
class     myloop
{
    public  static void main(String   [  ]st)
    {
        int     i=10;
        while(i>=1)
        {
                if(i>5)
                     {
                            System.out.print(“\t”+i);
                            i--;
                            continue;
                     }
                else
                     {
                             i--;
                     }
        }
    }
}
       OUTPUT:    10     9   8       7       6
```

Reading Input using ‘*Scanner’* class of *‘java.util’* package:

- We can read varieties of inputs from keyboard or from text file using methods of ‘*Scanner’* class.
- *Scanner* class belongs to ‘java.util’ package.
- When *Scanner* class receives input, it breaks the input into several pieces, called ‘tokens’ and these tokens can be retrieved using object of *Scanner* class.
- Note that: Following methods of *Scanner* class are *non-static* therefore they are called or accessed with the help of *object* of *Scanner* class.

We can create object of Scanner class as follows:

```
Scanner  obj=new    Scanner(System.in);
```

Here, ‘obj’ is object of *Scanner* class.
‘System.in’ represents *InputStream*, which is by default represents standard input device i.e. Keyboard.
There are several methods of *Scanner* class used to take different inputs as follows:

Method	Working
next( )	It is used to read single string
nextByte( )	It is used to read single byte type value
nextInt( )	It is used to read single integer type value
nextFloat( )	It is used to read single Float type value
nextLong( )	It is used to read single Long type value
nextDouble( )	It is used to read single Double type value
nextShort( )	It is used to read single short type value

Following program demonstrate the use of different methods of *Scanner* class.

```
import      java.util.Scanner;
class    CriClass
{
            byte     no;
            String   name;
            long     contact;
            int         t_sc;
            short    t_wk;
            float    ball_avg;
            double   bat_avg;
    public static void main(String []args)
    {
            Scanner   sc=new   Scanner(System.in);
            CriClass  obj=new  CriClass();
            System.out.print("Enter Cricketer No= ");
            obj.no=sc.nextByte();
            System.out.print("Enter Cricketer Name= ");
            obj.name=sc.next();
            System.out.print("Enter Cricketer Contact No= ");
            obj.contact=sc.nextLong();
            System.out.print("Enter Cricketer Total Score= ");
            obj.t_sc=sc.nextInt();
            System.out.print("Enter Cricketer Wickets= ");
            obj.t_wk=sc.nextShort();
            System.out.print("Enter Ball AVG= ");
            obj.ball_avg=sc.nextFloat();
            System.out.print("Enter Batting AVG= ");
            obj.bat_avg=sc.nextDouble();
            System.out.println("----------------------------------------");
            System.out.println("CricketerNO="+obj.no);
            System.out.println("CricketerName="+obj.name);
            System.out.println("ContactNo="+obj.contact);
            System.out.println("Total Score="+obj.t_sc);
            System.out.println("Total Wickets="+obj.t_wk);
            System.out.println("Balling AVG="+obj.ball_avg);
            System.out.println("Batting AVG="+obj.bat_avg);
      }
}
```

User defined methods in JAVA:

- Like C or C++ language, JAVA language also have 4 types of methods depending upon parameter acceptance or not and value return or not.
- Following program shows defining 4 types methods in Java language.

```
import     java.util.Scanner;
public    class    FunctionDemo
{
            int     x,y,z;
            Scanner   sc=new Scanner(System.in);
    void      add(int   a,int   b)       //with arg. without return value
    {
            int c;
            c=a+b;
            System.out.println("Addition="+c);
    }
    int     sub(int   a,int   b)      //with arg. with return value
    {
            int c;
            c=a-b;
            return(c);
    }
    int     multi()       //without arg. with return value
    {
            System.out.println("Enter Two Numbers=");
            x=sc.nextInt();
            y=sc.nextInt();
            z=x*y;
            return(z);
    }
    void   division()      //without arg. without return value
    {
            System.out.println("Enter Two Numbers=");
            x=sc.nextInt();
            y=sc.nextInt();
            z=x/y;
            System.out.println("Division="+z);
    }
    public static void main(String []args)
    {
            FunctionDemo    obj=new    FunctionDemo();
            System.out.println("Enter two number");
            obj.x=obj.sc.nextInt();
            obj.y=obj.sc.nextInt();
            obj.add(obj.x,obj.y);
            System.out.println("Enter two number");
            obj.x=obj.sc.nextInt();
            obj.y=obj.sc.nextInt();
            obj.z=obj.sub(obj.x,obj.y);
            System.out.println("Subtraction="+obj.z);
            obj.z=obj.multi();
            System.out.println("Multiplication="+obj.z);
            obj.division();
    }
}
```

Type casting: (Type conversion)

- Converting one data type into another data type is called “Type casting” or “Type-conversion”.
- We can convert the values from one type to another explicitly using the cast operator as follows.
- Syntax for type casting:

  (type_name) expression;
- Here, type_name is any valid data type into which we can convert value of expression.
-

Following program shows type casting that convert char data type into int data type.

```
import    java.util.Scanner;
public   class    typeCast
 {
    public   static   void   main(String []args )
   {
      char       ch;
      int         p;
      Scanner   sc=new  Scanner(System.in);
      System.out.println("Enter any Character:");
      ch=sc.next().charAt(0);  //reading single character
      p = (int) ch;          //type casting
      System.out.println("ASCII value="+p);
     }
}
```

Following program shows type casting that convert int data type into float data type.

```
import    java.util.Scanner;
public   class   typeCast
 {
   public   static   void   main(String []args )
  {
        int        a,b;
        float      p;
        Scanner   sc=new  Scanner(System.in);
  System.out.println("Enter any Two numbers=");
        a=sc.nextInt();
        b=sc.nextInt();
        p = (float) a/b;          //type casting
        System.out.println("Division="+p);
   }
}
```

Theory Assignment No: 01

1) What is Java? Explain its various features.
2) What is Java? Write its evolution. And list out its drawbacks.
3) Write difference between C and Java.
4) Write difference between C++ and Java.
5) Explain different components of JDK with their use.
6) Explain JVM architecture. **OR** Explain working of JVM **OR** How JVM works?
7) Why Java does not supports for Pointers?
8) What is Java Tokens? List out its different tokens.
9) What are the different naming conventions used in Java?
10) Explain for-each loop in Java.
11) What is type casting? How type casting is done in Java?

Practical Assignment No: 1

Note: Implement following programs by using command line Argument in JAVA

1) Write a program to print First name, Middle name and Last name of employee.
2) Write a program which find sum of even numbers and odd numbers from 1 to 20.
3) Write a program which prints first ‘n’ numbers.
4) Write a program which find sum of first ‘n’ numbers.
5) Write a program which prints factors of entered number.
6) Write a program which check entered number is Perfect or not.
7) Write a program which find sum of digits (digit sum) of entered number
8) Write a program which check entered number is Armstrong or not.
9) Write a program which reverses the entered number.
10) Write a program which check entered number is Palindrome or not.
11) Write a program which finds face value of entered number.
12) Write a program which check entered number is Prime or not.
13) Write a program which finds factorial of an entered number.
14) Write a program which prints Fibonacci series up to ‘n’ numbers.
15) Write a program to check entered number is Strong or not.

    *(**Hint:** Strong number is a special number whose sum of the factorial of digits is equal to the original number For Example: 145 is strong number. Since, 1! + 4! + 5! = 145)*
16) Write a program to check entered number is Magic or not.

    *(**Hint:** For example, 325 is a magic number because the sum of its digits (3+2+5) is 10, and again sum up the resultant (1+0), we get a single digit (1) as the result. Hence, the number 325 is a magic number. Some other magic numbers are 1234, 226, 10, 1, 37, 46, 55, 73, etc.)*

Practical Assignment: 02

Note: To accept inputs form keyboard use methods of ‘Scanner’ class of java.util

1) Write a program to find addition, subtraction, multiplication, division of two numbers.
2) Write a program to find average of five numbers.
3) Write a program to find area of circle.
4) Write a program to find circumference (perimeter) of circle.
5) Write a program to find area of triangle.
6) Write a program which accepts six subject marks and calculates total marks and percentage of student.
7) Write a program to calculate simple interest.
8) Write a program to calculate compound interest.
9) Write a program to swap two integers.
10) Write a program to find distance between two points.
11) Write a program to check entered number is positive or negative.
12) Write a program to check entered number is even or odd.
13) Write a program to check entered year is leap or not.
14) Write a program to find maximum number between three numbers.
15) Write a program to find minimum number between three numbers.

16) Write a program that demonstrate the use of 'instanceof' operator
17) Write a program that demonstrates the use of 'cast' operator.
18) Write a program which calculates total marks and percentage obtained in six subjects and also display grade of student according to following table:

Percentage	Grade
0 to 39.99	Fail
40 to 49.99	Third
50 to 59.99	Second
60 to 69.99	First
70 to 100	Distinction

19) Write a program which calculates income tax corresponding to Following table:

Income	Tax
0 to 150000	0%
150001 to 300000	10%
300001 to 500000	20%
500001 and above	30%

20) Write a program which calculates telephone bill corresponding to following table:

Unit Consumed	Rate/unit in RS.
0 to 200	1.00
201 to 350	1.20
351 to 500	1.50
501 and above	1.75

21) Write a program which take single digit number as input and print corresponding number into word.
22) Write a program menu driven program to find out area of circle, circumference (perimeter) of circle, area of triangle and area of square
23) Write a menu driven program for:
    1: Addition of two numbers.
    2: Subtraction of two numbers.
    3: Multiplication of two numbers.
    4: Division of two numbers.
    5: Modulation of Two numbers.
24) Write a program to print First name, Middle name and Last name of employee.
25) Write a program which find sum of even numbers and odd numbers from 1 to 20.
26) Write a program which prints first 'n' numbers.
27) Write a program which find sum of first 'n' numbers.
28) Write a program which prints factors of entered number.
29) Write a program which check entered number is Perfect or not.
30) Write a program which find sum of digits (digit sum) of entered number
31) Write a program which check entered number is Armstrong or not.
32) Write a program which reverses the entered number.
33) Write a program which check entered number is Palindrome or not.
34) Write a program which finds face value of entered number.
35) Write a program which check entered number is Prime or not.
36) Write a program which finds factorial of an entered number.
37) Write a program which prints Fibonacci series up to 'n' numbers.
38) Write a program to check entered number is Strong or not.

39) Write a program to check entered number is Magic or not.

40) Write a program to find all Armstrong numbers from 1 to 1000

41) Write a program to find all Prime numbers from 1 to 1000

42) Write a program to find all palindrome numbers from 500 to 700

43) Write a program which prints multiplication table

Practical Assignment: 03

Que. Write the program that prints following pattern:

1)

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

2)

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

3)

```
A B C D C B A
A B C   C B A
A B       B A
A           A
```

4)

```
                A
            A   B
        A   B   C
    A   B   C   D
A   B   C   D   E
```

5)

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

6)

```
1
2   3
4   5   6
7   8   9   10
```

7)

```
1
0   1
0   1   0
1   0   1   0
```

8)

```
@
@@
@@@
@@@@
@@@@@
@@@@
@@@
@@
@
```

9)

```
     @
    @@
   @@@
  @@@@
 @@@@@
@@@@@@
```

10)

```
     @
    @@
   @@@
  @@@@
 @@@@@
@@@@@@
 @@@@@
  @@@@
   @@@
    @@
     @
```

11)

```
$ $ $ $ $ $ $ $ $ $
$ $ $ $ $ $ $ $ $
$ $ $ $ $ $ $ $
$ $ $ $ $ $ $
$ $ $ $ $ $
$ $ $ $ $
$ $ $ $
$ $ $
$ $
$
```

UNIT-II: Array, String and I/O

INTRODUCTION:

- We know that, the concept of variable is introduced **to store the data**.
- But single variable can store only one value at a time, this is the drawback of variable. To overcome the drawback of single variable the concept of *Array* is introduced.
- That is *array* is also single variable but it has an ability to store multiple (more than one) value at a time.

Definition:

- "An Array is collection of elements or items having **same data type** referred by common name"
- **OR**
- "An Array is collection of homogeneous (having same type) elements or items referred by common name"

- In an array the individual element is accessed with the help of integer value and is called **subscript or index.** Also all elements of array are stored in **continuous memory** allocation.
- *Note that:*
  - ➢ *We know that, In C/C++ language, memory for array is get allocated at compile time (i.e. static memory allocation)*
  - ➢ *But, in JAVA everything is dynamic i.e. for variable, array, objects etc. memory is get allocated at run time (Dynamic memory allocation) by JVM*

Types of Array:

Depending on number of subscripts used in array, array having three types:

1) One Dimensional array: (1 D array)

The array having **only one subscript** is called as "*One dimensional array*"

Declaration Syntax:

```
datatype    array_name[ ]=new datatype [Size];
                      OR
datatype    [ ]array_name=new datatype [Size];
```

Here;
datatype is any valid *data type* in JAVA language
array_name is name of array which is an identifier.
'*size*' is integer value that denotes total number of elements stored in array
'*new*' is an operator.
e.g.

1) int x[ ]=new int[5];*//declares array 'x' & allocates memory for 5 integers*
   OR
   int [ ]x =new int[5];

   here ;
   'x' is array which can holds(stores) **five** integers at a time.

2) int marks[ ]; *//declares array 'marks'*
   marks=new int[5]; *//allocates memory for 5 integers*

Initialization of One dimensional array:

One dimensional array can be initialized as fallow;
e.g. int p[ ] = { 10 , 20 , 30 , 40 , 50 } ;

Here;
'p' is integer type array which stores five integers at a time. The storage of all elements in array 'p' is shown in following figure:

Index →	*0*	*1*	*2*	*3*	*4*
Elements of 'P'	**10**	**20**	**30**	**40**	**50**
Address →	110	114	118	122	126

In above figure the elements 10, 20, 30, 40, 50 are stored in array 'p' at individual index. That is the element 10 is stored at index 0, 20 is stored at index 1 like that, 50 is stored at index 4. Also all elements are stored in continuous memory location.
Note:
Index is integer value which is nothing but position of the element in an array.

2) Two Dimensional array: (2D array)

The array having **two subscripts** is called as "Two dimensional array or matrix"
The two dimensional array is used to perform matrix operations.
Declaration Syntax:

```
datatype        array_name[ ][ ]=new  datatype [Size1][Size2];
                            OR
datatype        [ ][ ]array_name=new  datatype [Size1][Size2];
```

Here;
datatype is any valid *data type* in Java language
array_name is name of array which is an identifier.
'*Size1*' is integer value that denotes total number of rows in array
'*Size2'* is integer value that denotes total number of columns in array
E.g.

1) int x[ ][ ]=new int[3][2];
   OR
   int [ ][ ] x=new int[3][2];

here ;
'*x*' is array which can holds total 6 integers at a time.
3 is *rowsize* i.e. there are 3 rows in matrix 'x'
2 is *columnsize* i.e. there are 2 columns in matrix 'x'

Initialization of Two dimensional array:
Two dimensional array can be initialized as follow;
e.g. 1) int p[ ][ ]={{3,4},{2,9},{7,6}};

```
2)      int           z[ ][ ] ={  {  5, 3 , 6 } ,
                                  {  1, 7 , 8 } ,
                                  {  9, 4 , 2 }  };
```

Here;
'z' is integer type two dimensional array which stores nine integers at a time. The storage of all elements in array 'z' is shown in following figure:

row index ↓	**0**	**1**	**2** ← column index
0	5	3	6
1	1	7	8
2	9	4	2

In above figure the individual element of array ‘z’ is also accessed by following way:

The element 8 can be accessed as z[1][2]
The element 7 can be accessed as z[1][1]
The element 9 can be accessed as z[2][0]   etc.

like that we can access all individual elements of matrix ‘z’

<u>3) Multi-Dimensional array:</u>

“The array having **more than two subscripts** is called as *multi-dimensional* array”

<u>Declaration Syntax:</u>

```
datatype array_name[ ][ ]……[ ]=new datatype [Size1][Size2]……[SizeN];
                              OR
datatype [ ][ ]……[ ]array_name=new datatype [Size1][Size2]……[SizeN];
```

here;

datatype is any valid *datatype* in Java language.
array_name is an identifier (name given by programmer)
size1, size2,………sizeN are **integer constants** and that denotes total number of elements stored in an array.

e.g. 1) int z[][][]=new int[2]**[4][3]**;

Above array is the multidimensional array having three subscripts and which stores 12 integer values at a time.

*That is above multi-dimensional array stores **2 sets of 4*3 matrices**.*

2) float x[ ][][][]=new float[2][3][2][2];

The above array ‘x’ is also multidimensional array having four subscripts and which stores 24 float values at a time.

<u>Initialization of Multi-dimensional array:</u>

Multi-dimensional array can be initialized as follows:

Example:

```
1)  int  x[][][]={{{1,2},{4,5},{7,8},{3,6}},
            {{11,12},{14,15},{17,18},{13,16}} };
```

In above multi-dimensional array, elements are stored in following manner;

X[0] ⇨

	0	*1*
0	**1**	**2**
1	**4**	**5**
2	**7**	**8**
3	**3**	**6**

x[1] ⇨

	0	*1*
0	**11**	**12**
1	**14**	**15**
2	**17**	**18**
3	**13**	**16**

From above, multi-dimensional array we can access individual element as follow:

```
X[0][1][0]   access  4
X[0][2][1]   access  8
X[0][3][0]   access  3
X[1][1][1]   access  15
X[1][2][0]   access  17
X[1][3][1]   access  16
```

'length' property of array:

- 'length' is property of an array *returns one integer value which is nothing but* ***<u>size of array</u>***.
- We use this property as follows:

```
int     var = arrayname.length;
```

Here, 'var' is an *int* type variable which stores size of array return by *arrayname.length* poperty.

Exampl:

```
1) int   arr[ ]=new  int[10];    //declares array with 10 size
          arr[0]=45;
          arr[1]=90;
   int    sz=arr.length;       //returns array size
   System.out.print("Array Size="+sz);

        OUTPUT:
                  Array Size=10
```

<u>Note that:</u>

- In above example 'arr' is array declared with 10 size. And arr[0] and arr[1] are initialized with values 45 and 90 respectively. But '*arr.length*' statement returns value 10 which is size of array (Since, *<u>'length' property returns</u>* ***<u>size of array</u>****<u>. It</u>* ***<u>not</u>*** *<u>returns total number of array elements</u>*)
- *Also, in case of 2D or Multi-dimensional array, 'length' property returns* ***<u>number of rows of the array</u>***.

Following program shows use of 'length' property of array.

```
public   class   Myarr
{
   int    arr[]=new   int[5];
   int    arr1[][]=new  int[2][3];
   int    arr2[][][]=new   int[4][5][6];
   void get()
   {
    int   i=arr.length;
    int   j=arr1.length;
    int   k=arr2.length;
    System.out.println("Size of 1D="+i);
    System.out.println("Size of 2D="+j);
    System.out.println("Size of MD="+k);
   }
  public static void main(String []ar)
  {
    Myarr  p=new  Myarr();
    p.get();
  }

}
OUT PUT:
           Size of 1D=5
           Size of 2D=2
           Size of MD=4
```

STRING

Introduction:

- String is nothing but collection or group of characters.
- We know that, in case of C/C++ language string is nothing but *'array of characters'* & that terminates with '\0' character i.e. NULL character. But this is not true in Java language.
- In JAVA, *"String is nothing but Object of String class*" and which is not array of character. Also, in JAVA we can create array of character but it is not treated as string.
- While dealing with string, JAVA language has special class called "String" class which was found under "java.lang" package.

Creating Strings in Java:

We can create Sting in Java by three ways:

- We can create a String by assigning group of characters to a String type object:

```
String    str;   //declare string
str= "Hello";      //assign a group of characters to it
```

  Above two statements can be combined as follow:

```
String    str= "Hello";
```

  In this case, JVM creates an string reference & stores the string "Hello"

- We can create an object of String class by using 'new' operator:

```
String    str = new    String("Hello");
```

  In this case, First we create object '*str'* using *new* operator and then we store "Hello" string in it.

- Third way of creating String is by converting array of character into String:

```
char    arr[]={'H','e','l','l','o'};//create character array.
String    str=new    String(arr);//creating string 'str' by passing 'arr' to it.
```

Difference between object & Reference:

Object	Reference
1) The object is created using 'new' operator. e.g. String str=new String( );	1) Reference is not created using 'new'operator. e.g. String str;
2) Such type of object is stored in 'Heap' section by 'class loader sub system' of JVM.	2) Such type of referece is stored in 'Method area' section by 'class loader sub system' of JVM.
3) When such object is created then, Constructor is implicitly invoked.	3) When such reference is created then, Constructor is NOT implicitly invoked.
4) When such object is created then JVM allocated separate memory location to each object.	4) When such reference is created then JVM insert it into "*String constant pool*" (Special memory block where String references are stored)

String Class Methods:

While dealing with Strings, there are several methods belong to String class:

Note that: Following methods of String class are non-static therefore they are called with object of String class.

1) length ( ) :

- This method is used to find length of string.
- This method returns one integer value which is length (total characters) of string.

Syntax:

```
int    len = s.length( );
```

Here, *'s'* is an object or reference of String class and that contains the string.
'len' is integer variable used to store length of string.

E.g.

```
class        stringLen
{
     public static void main(String args[ ])
     {
               String    str=new    String("Hello");
               intlen=str.length( );
               System.out.println("Length="+len);
     }
}
OUTPUT: Length=5
```

2) concat ( ) :

- This method is used to concatenate two strings together.
- That is, it concatenates second string at the end of first string and returns concatenated string as a result.

Syntax:

```
String    str= s1.concat(s2);
```

Here, 's1' is an object or reference of String class and that contains the first string.
's2' is also object or reference of String class and that contains the second string.
'str' is also string that stores concatenated string.

E.g.

```
class    stringCat
{
      public static void main(String   args[ ])
      {
               String    s1= "Delhi";
               String    s2= "Mumbai";
                      String    str=s1.concat(s2);
               System.out.println(str);
      }
}
OUTPUT:             DelhiMumbai
```

3) charAt ( ) :

- This method accepts one integer value and returns one character from string corresponding to passed integer value
- That is,it accepts index value (position of element) and returns corresponding character from string.

Syntax:

```
char     ch= s.charAt(pos);
```

Here, 's' is an object or reference of String class and that contains the string.
'pos' is an integer value.
'ch' is char variable which stores returned character from string at position '*pos'*

E.g.

```
class       stringChar
{
      public static void main(String   args[ ])
      {
               String    s= "Delhi";
               char  ch= s.charAt(3);
               System.out.println(ch);
      }
}
OUTPUT:
               h
```

4) equals( ) :

- This method is also used to compare two strings with each other for equality.
- This method returns boolean value depending upon Strings content.
- It returns boolean value 'true' if both strings are equal otherwise it returns 'false'.

Syntax:

```
boolean     m = s1.equals(s2);
```

Here, 's1' is an object or reference of String class and that contains first string.
's2' is also an object or reference of String class and that contains second string.
'm' is boolean variable to store returned value.

E.g.

```
class          stringEqual
{
    public static void main(String   args[ ])
    {
            String    s1= “box”;
            String    s2= “BOX”;
            boolean  m= s1.equals(s2);
            if (m = = true)
                    System.out.print(“Both strings are equal”);
            else
                    System.out.print(“Strings are not equal”);
    }
}
OUTPUT:    Strings are not equal
```

5) equalsIgnoreCase( ) :

- This method is also used to compare two strings with each other for equality. But it ignores the case of characters present in strings. i.e. it treated “box” string same as “BOX”
- This method returns *boolean* value depending upon Strings contain.
  - ❖ It returns boolean value ‘true’ if both strings are equal otherwise it returns ‘false’.

Syntax:

```
boolean     m = s1.equalsIgnoreCase(s2);
```

Here, ‘s1’ is an object of String class and that contains first string.
‘s2’ is also an object of String class and that contains second string.
‘m’ is boolean variable to store returned value.

E.g.

```
class       stringEqual
{
    public static void main(String   args[ ])
    {
            String    s1= “box”;
            String    s2= “BOX”;
            boolean  m= s1.equalsIgnoreCase(s2);
            if (m = = true)
                    System.out.print(“Both strings are equal”);
            else
                    System.out.print(“Strings are not equal”);
    }
}
OUTPUT:   Both Strings are equal
```

6) compareTo ( ) :

- This method is used to compare two strings with each other for equality.
- This method returns one integer value depending upon Strings contain.
  - ❖ It returned value is Zero then both strings are equal.
  - ❖ If returned value is positive then first string is greater than second string.
  - ❖ If returned value is negative then Second string is greater than first string.

Syntax:

```
int     m = s1.compareTo(s2);
```

Here, ‘s1’ is an object of String class and that contains first string.

‘s2’ is also an object of String class and that contains second string.

‘m’ is integer variable to store returned value.

E.g.

```
class            stringCmp
{
    public static void main(String   args[ ])
    {
            String    s1= "abc";
            String    s2= "abd";
            int  m= s1.compareTo(s2);
            if (m = = 0)
                    System.out.print("Both strings are equal");
            else  if( m > 0)
                    System.out.print("First string is greater");
            else
                    System.out.print("Second string is greater");
    }
}
OUTPUT:        Second string is greater
```

7) compareToIgnoreCase( ) :

- This method is also used to compare two strings with each other for equality. But it ignores the case of characters present in strings. i.e. it treated "box" string same as "BOX"
- This method returns one integer value depending upon Strings contain.
  - ❖ If returned value is Zero then both strings are equal.
  - ❖ If returned value is positive then first string is greater than second string.
  - ❖ If returned value is negative then Second string is greater than first string.

Syntax:

```
int     m = s1.compareToIgnoreCase(s2);
```

Here, 's1' is an object of String class and that contains first string.
's2' is also an object of String class and that contains second string.
'm' is integer variable to store returned value.

E.g.

```
class        stringCmp
{
    public static void main(String   args[ ])
    {
            String    s1= "box";
            String    s2= "BOX";
            int  m= s1.compareToIgnoreCase(s2);
            if (m = = 0)
                    System.out.print("Both strings are equal");
            else  if( m > 0)
                    System.out.print("First string is greater");
            else
                    System.out.print("Second string is greater");
    }
}
OUTPUT:
              Both strings are equal
```

8) startsWith( ) :

- This method returns *boolean* value 'true' if first string starts with second string otherwise it returns 'false'.

Syntax:

```
boolean     m = s1.startsWith(s2);
```

Here, 's1' is an object of String class and that contains first string.
's2' is also an object of String class and that contains second string.
'm' is boolean variable to store returned value.

E.g.

```
class             stringStart
{
    public static void main(String   args[ ])
    {
            String    s1= “Box is heavy”;
            String    s2= “Box”;
            boolean  m= s1.startsWith(s2);
            if (m = = true)
                    System.out.print(“First string start with second string”);
            else
                    System.out.print(“First string NOT start with second string”);
    }
}
OUTPUT: First string start with second string
```

9) endsWith( ) :

- This method returns boolean value ‘true’ if <u>first string ends with second string</u> otherwise it returns ‘false’.

   Syntax:

```
boolean     m = s1.endsWith(s2);
```

Here, ‘s1’ is an object of String class and that contains first string.
‘s2’ is also an object of String class and that contains second string.
‘m’ is boolean variable to store returned value.

E.g.

```
class           stringEnd
{
    public static void main(String   args[ ])
    {
            String    s1= “Box is heavy”;
            String    s2= “heavy”;
            boolean  m= s1.startsWith(s2);
            if (m = = true)
                    System.out.print(“First string ends with second string”);
            else
                    System.out.print(“First string NOT ends with second string”);
    }
}
OUTPUT: First string ends with second string
```

10) indexOf( ) :

- This method returns integer value which is nothing but <u>first position</u> of substring into main string.
- If substring is not found in main string then it returns <u>**-1**</u> <u>negative</u> value.

   Syntax:

```
int    m = s1.indexOf(s2);
```

Here, ‘s1’ is an object of String class and that contains first string.
‘s2’ is also an object of String class and that contains second string.
‘m’ is integer variable to store returned value.

E.g.

```
class         stringFirstIndex
{
     public static void main(String   args[ ])
     {
               String    s1= "Box is heavy and it is dirty";
               String    s2= "is";
               int m= s1.indexOf(s2);
               if (m < 0 )
                         System.out.print("Substring Not found");
               else
                         System.out.print("Substring found at= "+m);
     }
}
OUTPUT: Substring found at= 4
```

11) lastIndexOf( ) :

- This method returns integer value which is nothing but last position of substring into main string.
- If substring is not found in main string then it returns **-1** negative value.
- Syntax:

```
int    m = s1.lastIndexOf(s2);
```

Here, 's1' is an object of String class and that contains first string.
's2' is also an object of String class and that contains second string.
'm' is integer variable to store returned value.

E.g.

```
class            stringLastIndex
{
     public static void main(String   args[ ])
     {
               String    s1= "Box is heavy and it is dirty";
               String    s2= "is";
               int m= s1.lastIndexOf(s2);
               if (m < 0 )
                         System.out.print("Substring Not found");
               else
                         System.out.print("Substring found at= "+m);
     }
}
OUTPUT: Substring found at= 20
```

12) replace( ) :

- This method replaces characters of existing string with new given character.

Syntax:

```
String    str = s.replace(old_char,new_char);
```

Here, 'old_char' is character to be replaced by 'new_char' of string.
'str' is String variable to store returned value.

E.g.

```
class         stringReplace
{
     public static void main(String   args[ ])
     {
               String    s= "Box is heavy";
               String    str= s.replace('B','D');
               System.out.print(str);
     }
}
OUTPUT: Dox is heavy
```

13) substring( ) :

- This method has two forms:-

I) substring(int):

- This method returns a new string consisting of <u>all characters from *given position* to the *end of string*</u>.

Syntax:

```
String    str = s.substring(pos);
```

Here, 's' is an object of String class and that contains string.
'pos' is an integer value from which new string is to be started.
'str' is string object to store resultant string.

E.g.

```
class           stringSubstr
{
     public static void main(String   args[ ])
     {
               String    s= "Box is heavy";
               String    str= s.substring(4);
               System.out.print(str);
     }
}
OUTPUT:   is heavy
```

II) substring(int,int):

- This method returns a new string consisting of <u>all characters from *given **first position*** to *the **last position -1***</u>.

Syntax:

```
String    str = s.substring(pos1,pos2);
```

Here, 's' is an object of String class and that contains string.
'pos1' is an integer value i.e. *first position*from which new string is to be started.
'pos2' is an integer value i.e. *last position*and new string ends at <u>*last postion-1*</u>.
'str' is string object to store resultant string.

E.g.

```
class    stringSubstr1
{
     public static void main(String   args[ ])
     {
               String    s= "Box is heavy";
               String    str= s.substring(7,11);
               System.out.print(str);
     }
}
OUTPUT:   heav
```

14) toLowerCase( ) :

- This method converts all characters of string into *lower case* and returns that lower-cased string as result.

Syntax:

```
String    str = s.toLowerCase();
```

Here, 's' is an object of String class and that contains string.
'str' is also an object of String that stores returned lower-cased string.

E.g.

```
class              stringLower
{
     public static void main(String   args[ ])
     {
               String    s= "BOX IS HEAVY";
               String    str= s.toLowerCase( );
               System.out.print(str);
     }
}
OUTPUT:   box is heavy
```

15) toUpperCase( ) :

- This method converts all characters of string into *upper case* and returns that upper-cased string as result.

Syntax:
```
String    str = s.toUpperCase();
```

Here, ‘s’ is an object of String class and that contains string.
‘str’ is also an object of String that stores returned upper-cased string.

E.g.
```
class              stringUpper
{
    public static void main(String   args[ ])
    {
            String    s= “box is heavy”;
            String    str= s.toUpperCase( );
            System.out.print(str);
    }
}
OUTPUT:   BOX IS HEAVY
```

16) getChars( ) :

- This method copies characters from *string into character array*.
- Characters copied into character array from starting position ‘*pos1’* up to last position ‘*pos2-1*’ to location starting from ‘pos3’ in a character array.

Syntax:
```
s.getChars(pos1,pos2,arr,pos3);
```

Here, ‘s’ is an object of String class and that contains string.
‘pos1’ is an integer value represents *starting position* from which string copied.
‘pos2’ is an integer value represents *ending position*& string copied ends at <u>*pos2-1*</u> index.
‘arr’ is character array that stores copied characters from string ‘s’.
‘pos3’ is an integer value from which character array ‘arr’ stores the copied character.

E.g.
```
class           stringGetchar
{
    public static void main(String   args[ ])
    {
            String    s= "Box is heavy";
            char [ ]arr=new char[20];
            s.getChars(7,11,arr,0);
            for(char i:arr)
            {
               System.out.print(i);
            }
    }
}
OUTPUT:   heav
```

17) trim( ) :

- This method removes unwanted i.e. extra spaces which were found ***before and after*** the string.
- *Note that:*This method does <u>*not remove extra spaces between two words*</u> of string.

Syntax:
```
String    str = s.trim( );
```

Here, ‘s’ is an object of String class and that contains string.
‘str’ is also object of String class used to store resultant string.

E.g.

```
class    stringTrim
{
      public static void main(String   args[ ])
      {
                  String    s= "              Box is heavy           ";
                  String    str = s.trim( );
                  System.out.print(str);
      }
}

OUTPUT:   Box is heavy
```

18) split( ) :

- This method splits or cuts the given string into number of pieces (sub strings) corresponding to delimiter (specified character) and store it into array of string.

Syntax:
```
String  [  ] str = s.split(delimiter);
```

Here, ‘s’ is an object of String class and that contains string.
‘delimiter’ is string at which we specify splitting character
‘str’ is array of String that stores splitted strings at individual indices.

E.g.
```
class            stringSplit
{
      public static void main(String   args[ ])
      {
                  String    s= "Box is heavy,dirty,red and useless";
                  String [ ]str=s.split(",");
                  for(String i: str)
                  {
                              System.out.println(i);
                  }
      }
}
OUTPUT:
            Box is heavy
            dirty
            red and useless
```

String Comparison:

- We know that for comparison of two or more quantities with each other, we use relational operators like <, >, <=, >=, !=, ==.
- But for string comparison, relational operators are *not* suitable because these operators *compare reference (address) of object with each other*. They ***not** compare strings contents*.
- For that purpose, compareTo() or equals() methods are used to compare two strings. And *these functions compares strings contents for equality*.
  Consider following example;

```
class          stringCmp
{
      public static void main(String   args[ ])
      {
                  String    s1=new    String(“Hello”);
                  String    s2=new    String(“Hello”);
                  if(s1= = s2)
                              System.out.println(“Strings are same”);
                  else
                              System.out.println(“Strings are NOT same”);
      }
}
OUTPUT:   Strings are NOT same
```

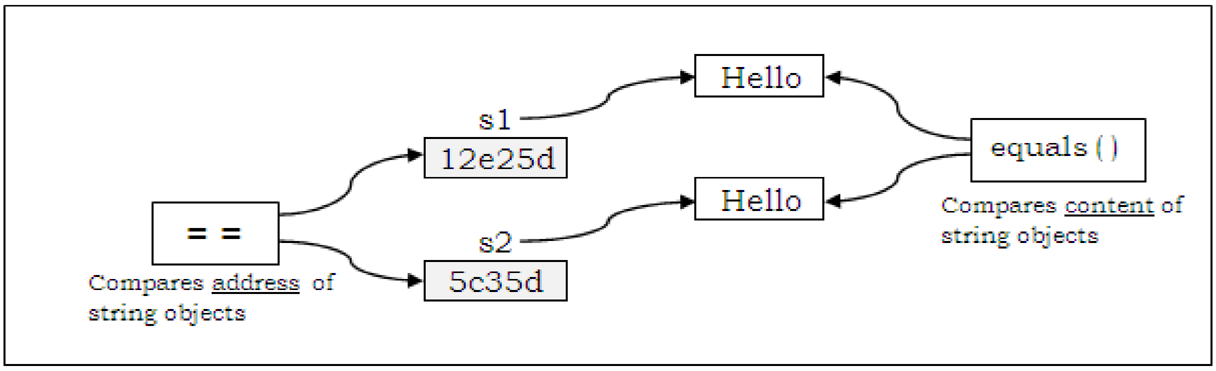
- In above example, we got output "Strings are NOT same" still content of strings are same. This is due to relational operator (= =) because it compares address (*address is in Hexadecimal number*) of both strings, it not compares strings content.
- When an object is created by JVM, it returns the memory address of the object as a hexadecimal number which is called "*object reference*" and this *object reference is separate for every object* i.e. whenever an abject is created, a new address number is allotted to it by JVM.

  Now, consider another example:

```
class      stringCmp
{
    public static void main(String   args[ ])
    {
            String    s1=new    String("Hello");
            String    s2=new    String("Hello");
            if(s1.equals(s2)==true)
                    System.out.println("Strings are same");
            else
                    System.out.println("Strings are NOT same");
    }
}

OUTPUT:   Strings are same
```

- In above example, we got output "Strings are same" which is right because *equals( )* method compares strings content, *not their addresses*.

  Above mentioned concepts shown in following figures:


s1
12e25d
s2
5c35d
Hello
Hello
= =
equals ( )
Compares address of string objects
Compares content of string objects

Now, consider One another example: There is slightly change in object creation of String. Here, object of String is created without using 'new' object:

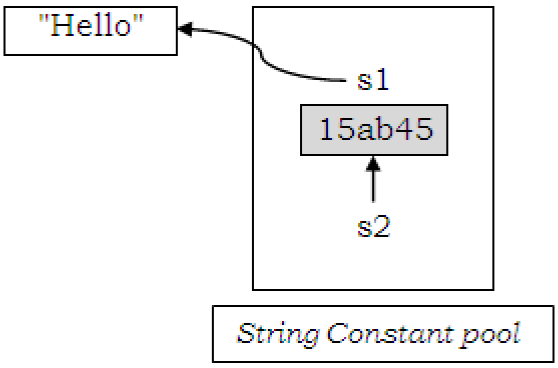
```
class       stringCmp
{
    public static void main(String   args[ ])
    {
            String    s1="Hello";
            String    s2="Hello";
            if(s1==s2)
                    System.out.println("Strings are same");
            else
                    System.out.println("Strings are NOT same");
    }
}
OUTPUT:   Strings are same
```

- In above example, we got output "Strings are same". Still we are using relational operator (= =) for string comparison. How it is possible?

→
  - ➢ Here, String objects are created without using 'new' operator at that time JVM uses separate memory block which is called *'string constant pool'* and such objects are store there.
  - ➢ When first statement i.e. String *s1= "Hello"* is executed then JVM insert object '*s1*' into '*string constant pool'* with separate address.

- ➢ And when second statement i.e. String *s2 "Hello"* is executed then JVM, searches in '*string constant pool*' to know whether the object with same content is already exist there or not?
  - ❖ If any object with same content is *found* in '*string constant pool*' then *JVM just attach second object at reference (address) of first object*.
  - ❖ If any object with same content is *NOT found* in '*string constant pool*' then JVM *allocate separate reference (address) to second object.*
- ➢ Here, content of String 's1' and 's2' are same therefore JVM just attach 's2' at address of 's1'. And that's made both objects addresses are same. And hence, we got output "Strings are same"

Above mentioned concept is shown in following fig.



Immutability of String:

- We know that, an 'object' is basic runtime entity that holds data or some content.
- And every Object can broadly classified into two categories viz: 1) Mutable object 2) Immutable object.

Let's see them in details:

1) Mutable Object:

- The objects whose *content* can be changeable or modifiable are called 'Mutable' objects.

2) Immutable Object:

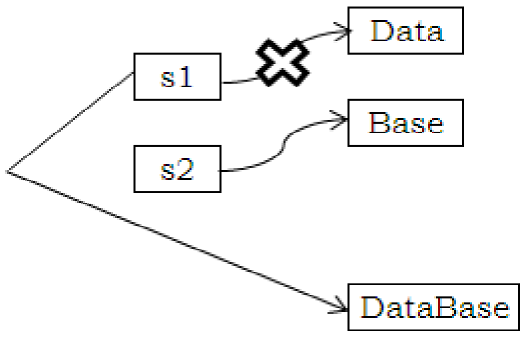
- The objects whose content can NOT be changeable or modifiable are called 'Immutable' Objects.
- And Object of String class is Immutable i.e. we cannot modify the content of String object. And hence, we can say that **'String' Class is Immutable class**.

Consider following example to test immutability of String.

```
class       myString
{
    public static void main(String   args[ ])
    {
            String    s1="Data";
            String    s2="Base";
            s1=s1+s2;
            System.out.println(s1);
    }
}

OUTPUT: DataBase
```

- In above example we got output "DataBase". It seems that, the content of 's1' is modified. Because, earlier 's1' has content "Data" and 's2' has content "Base". And after 's1+s2' they are joined together and total string becomes "DataBase". This result is assigned to 's1'. If 's1' is mutable then it gets new string "DataBase" and we also got "DataBase" as result.
- But 's1' is object of String class & we learned that 'String objects are immutable', then how we got result "DataBase"?

➔ Definitely String object 's1' is immutable. Consider following figure that will explain this concept briefly:



- In the program, JVM creates two objects 's1' and 's2' separately, as shown in above figure.
- When 's1+s2' is done then JVM creates new object and stores 'DataBase' in that created object.
- After creating new object, the reference 's1' is just attached or assigned to newly created object.
- Remember, it ***does not modify existing content of 's1'***. And that's why we are called *'String objects are immutable.'*
- The old object that contains 'Data' has lost its reference. So it is called '*unreferenced object'* and garbage collector remove it from memory.

StringBuffer And StringBuilder

StringBuffer:

- In previous chapter, we learnt that Strings are immutable and cannot modified. To overcome this, Java language provide new String handling class called 'StringBuffer' which represents strings in such a way that their data can be modified.
- It means 'StringBuffer' objects are *mutable* i.e. we can modified their data. Also, the methods present in this class that directly manipulate and change the data of object very easily.
- *Also, there are several classes present in Java which are immutable such as Character, Byte, Integer, Float, Double, Long. . . . etc wrapper classes are immutable.*

Creating StringBuffer Object:

There are three ways of creating *StringBuffer* object and fill that object with string:

I) Creating s*tringBuffer* object by using *new* operator and pass String to that object:

```
StringBuffer    sb=new    StringBuffer("Hello");
```

Here, we are passing "Hello" string to *stringBuffer* object 'sb'.

II) Creating *StringBuffer* object first using new operator & then storing string to created object:

```
StringBuffer   sb=new    StringBuffer( );
```

Here, we create *StringBuffer* object 'sb' as an empty object and not passing any string to it.
In this case, a *StringBuffer* object will be created with *default capacity of 16 characters*.
After creating *StringBuffer* object, we can store string to created object using method append() or insert().
E.g. sb.append("Hello");

```
StringBuffer    sb=new    StringBuffer(50);
```

Here, we create *StringBuffer* object 'sb' as an empty object with capacity of 50 characters.
But, we can store *more than 50 characters to 'sb' object because StringBuffer objects are mutable and can expand dynamically*.

III) First create object of String that holds one string and then pass that object to *StringBuffer*:

```
String        str= "Hello";
StringBuffer   sb=new    StringBuffer(str);
```

Here, we create String object 'str' that has string "Hello" and pass it to *StringBuffer* object 'sb'.
In this case, *StringBuffer* object stores data of *str i.e. "Hello"*

StringBuffer Class Methods:

There are several methods belonging to *StringBuffer* class and they are as follow:
Note that: Following methods of StringBuffer class are non-static therefore they are called with <u>*object*</u> *of StringBuffer class.*

1) append ( ) :

- This method is used to *append* given string into object of *StringBuffer* class.

  Syntax:

```
sb.append(x);
```

Here, *'sb'* is an object of *StringBuffer* class.
'x' is argument accept by append( ) method that may be boolean, int, long, float, char, String or another *StringBuffer*.

E.g.

```
class    datastr
{
    public static void main(String args[ ])
    {
            StringBuffer     sb=new   StringBuffer("Uni");
            sb.append("versity");
            System.out.println(sb);
    }
}
OUTPUT:          University
```

2) insert ( ) :

- This method is used to *inserts* given string into object of *StringBuffer* class from given position.

  Syntax:

```
sb.insert(pox,x);
```

Here, *'sb'* is an object of *StringBuffer* class.
'x' is argument accept by append( ) method that may be boolean, int, long, float, char, String or another *StringBuffer*.
'*pos'* is integer value which is position at which new string will be inserted.

E.g.

```
class    Mydemo
{
    public static void main(String args[ ])
    {
            StringBuffer     sb=new   StringBuffer("India Is My Country");
            sb.insert(9,"Great");
            System.out.println(sb);
    }
}
OUTPUT:          India Is GreatMy Country
```

In above program, "Great" string is inserted into '*sb'* object from index 9.

3) delete ( ) :

- This method is used to *deletes all the characters from first position 'i' to second postion 'j-1' p*osition.

  Syntax:

```
sb.delete(i,j);
```

Here, *'sb'* is an object of *StringBuffer* class.
'*i'* is integer value which is first position
'j' is integer value which is second position.

E.g.

```
class      Mydemo
{
    public static void main(String    args[ ])
    {
            StringBuffer       sb=new    StringBuffer("University");
            sb.delete(3,6);
            System.out.println(sb);
    }
}
OUTPUT:          Unisity
```

In above program, delete( ) methoddeletes all character form position 3 to 5.

4) reverse ( ) :

- This method is used to *reverse all the characters of StringBuffer*object.

  Syntax:

```
sb.reverse( );
```

Here, *'sb'* is an object of *StringBuffer* class that contains the string.

E.g.

```
class       Mydemo
{
    public static void main(String    args[ ])
    {
            StringBuffer        sb=new    StringBuffer("ABC");
            sb.reverse( );
            System.out.println(sb);
    }
}
OUTPUT:          CBA
```

5) length ( ) :

- This method is used to find number of characters present in *StringBuffer* object.
- This method returns one integer value which is total characters stored in *StringBuffer*.

  Syntax:

```
intlen = sb.length( );
```

Here, *'sb'* is an object of *StringBuffer* class and that contains the string.
'len' is integer variable used to store length.

E.g.

```
class          stringLen
{
    public static void main(String args[ ])
    {
            StringBuffer      sb=new    StringBuffer("Hello");
            intlen=sb.length( );
            System.out.println("Length="+len);
    }
}
OUTPUT:
            Length=5
```

6) indexOf( ) :

- This method returns integer value which is nothing but first position of substring into *StringBuffer* object.
- If substring is not found in *StringBuffer* object then it returns **-1** negative value.

Syntax:

```
int    m = sb.indexOf(str);
```

Here, 'sb' is an object of *StringBuffer* class and that contains first string.
'str' is sub string .
'm' is integer variable to store returned value.

E.g.
```
class        stringFirstIndex
{
    public static void main(String   args[ ])
    {
            StringBuffer      sb=new  StringBuffer("Box is heavy and it is dirty");
            int     m= sb.indexOf("is");
            if (m < 0 )
                    System.out.print("Substring Not found");
            else
                    System.out.print("Substring found at= "+m);
    }
}
OUTPUT: Substring found at= 4
```

7) lastIndexOf( ) :

- This method returns integer value which is nothing but last position of substring into *StringBuffer* object.
- If substring is not found in *StringBuffer* object then it returns **-1** negative value.

Syntax:
```
int    m = sb.lastIndexOf(str);
```

Here, 'sb' is an object of *StringBuffer* class and that contains first string.
'str' is sub string.
'm' is integer variable to store returned value.

E.g.
```
class           stringLastIndex
{
    public static void main(String   args[ ])
    {
            StringBuffer      sb=new    StringBuffer("Box is heavy and it is dirty");
            int m= s1.lastIndexOf("is");
            if (m < 0 )
                    System.out.print("Substring Not found");
            else
                    System.out.print("Substring found at= "+m);
    }
}
OUTPUT: Substring found at= 20
```

8) replace( ) :

- This method replaces characters from 'i' to 'j-1' position by given string of *StringBuffer* object

Syntax:
```
sb.replace(i,j, "str");
```

Here, 'i' is first position from which character replaces
'j' is second position and up to "j-1" position character are replaced by string "str"

E.g.
```
class        stringReplace
{
    public static void main(String   args[ ])
    {
            StringBuffer      sb=new StringBuffer("Box is heavy");
            sb.replace(0,3,"zzz");
            System.out.print(sb);
    }
}
OUTPUT: zzzis heavy
```

9) substring( ) :

- This method has two forms:-

I) substring(int):

- This method returns a new string consisting of all characters from *given position* to the *end of string from StringBuffer object*.

Syntax:

```
String    str = sb.substring(pos);
```

Here, 'sb' is an object of *StringBuffer* class and that contains string.
'pos' is an integer value from which new string is to be started.
'str' is string object to store resultant string.

E.g.

```
class          stringSubstr
{
    public static void main(String   args[ ])
    {
            StringBuffer         sb=new StringBuffer("Box is heavy");
            String    str= sb.substring(4);
            System.out.print(str);
    }
}
OUTPUT: is heavy
```

II) substring(int,int):

- This method returns a new string consisting of all characters from *given first position* to *the* ***last position -1*** from *StringBuffer* object.

Syntax:

```
String    str = sb.substring(pos1,pos2);
```

Here, 'sb' is an object of StringBuffer class and that contains string.
'pos1' is an integer value i.e. *first position* from which new string is to be started.
'pos2' is an integer value i.e. *last position* and new string ends at *last postion-1*.
'str' is string object to store resultant string.

E.g.

```
class    stringSubstr1
{
    public static void main(String   args[ ])
    {
            StringBuffer          sb=new StringBuffer("Box is heavy");
            String    str= sb.substring(7,11);
            System.out.print(str);
    }
}
OUTPUT: heav
```

StringBuilder Class:

- *StringBuilder* class has been added in JDK 1.5 which has same features of *StringBuffer* class.
- *StringBuilder* class objects are also mutable like *StringBuffer* class i.e. modification is allowed on object of *StringBuilder* class.

We can create object of *StringBuilder*class as follows:

- ❖ StringBuilder   sb=new StringBuilder("Hello");
- ❖ StringBuilder   sb=new StringBuilder( );
- ❖ StringBuilder   sb=new StringBuilder(50);

StringBuilder Class methods:

The following are the important methods of *StringBuilder* class whose functionality is same as *StringBuffer* class:

- StringBuilder append(x)
- StringBuilder insert(i,x)
- StringBuilder delete(i,j)
- StringBuilder reverse( )
- String toString( )
- int length( )
- int indexOf(String str)
- int lastIndexOf(String str)
- StringBuilder replace(i,j,Str)
- String substring(i);
- String substring(i,j);

Difference between *StringBuffer* and *StringBuilder* class:

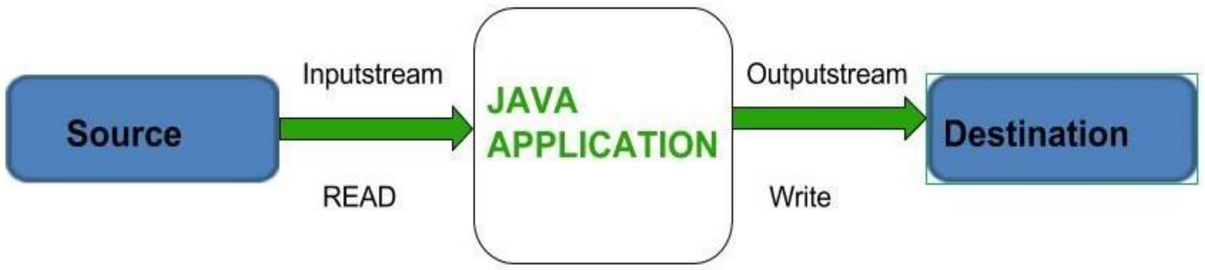
StringBuffer	**StringBuilder**
1) *StringBuffer* class is synchronized	1) *StringBuilder* class is not synchronized
2) This class is <u>synchronized</u> therefore multiple threads *cannot* act simultaneously onto object of *StringBuffer* class. i.e. threads act on object one after another.	2) This class is <u>not synchronized</u> therefore multiple threads *acts simultaneously onto object* of *StringBuilder* class. i.e. threads act on object at one time.
3) In this case, Threads act on single object one after another that gives reliable or correct result.	3) In this case, multiple threads act on single object at one time that gives unreliable or undependable or inaccurate result.
4) *StringBuffer* class will take <u>more execution time</u>. Because, when one thread acts on object at that time other threads are in waiting stage. After first thread completes its working then another thread start it work such process is happen for every thread.	4) *StringBuilder* class will take <u>less execution time than *StringBuffer*</u> class. Because, at one time multiple threads will act on same object of *StringBuilder* class. Here, no question for waiting of thread.

<u>Input / Output Handling</u>

- We know to solve any problem, we need an input and output operations. To perform input and output (I/O) in Java, it has *java.io* package that contains the classes which performs I/O.
- Such <u>I/O operations done via streams that represents an input source and an output destination</u>. The stream in the java.io package supports many data such as primitives, object, localized characters, etc.
- Java Program performs input / output through stream. So let's understand stream.

<u>Stream:</u>

- A stream can be defined as a sequence of bytes (data) which flows from input device to output device and vice-versa.
- There are two kinds of Streams:
- **InPutStream**: The InputStream is used to read data from a source i.e. from keyboard, file etc.
- **OutPutStream:** The OutputStream is used for writing data to a destination i.e. to monitor, file etc.
- **Following diagram shows I/O stream in Java-**



- To perform both Input stream and Output stream operations, Java language has two classes viz.-

1) Byte Stream (Binary Stream) class
2) Character Stream (Text stream) class

Before learning these classes, we have to know file and its type which is discussed below-

File:

- File is a place on secondary storage (Hard disk, CD, DVD, Pendrive etc.) where large amount of data can be stored permanently.

Types of file:

There are two types of files-
1) Binary files
2) Text Files

Let's see these types in details:

1) Binary Files:

- The files which operates data in terms of binary stream (8 bit data) those files are called "Binary Files"
- Binary files used to operate binary data such as images, videos etc.
- Binary files are secured file because data in not directly interpreted by human being.

2) Text files:

- The files which operates data in terms of text streams (16 bit Unicode data) or characters those files are called "Text files".
- Text files are Not secured file because its data directly interpreted by human being.

To deal with file in java, it has following classes:

1) Byte Streams class:

- Java byte streams are used to perform input and output in terms of 8-bit bytes.
- Though there are many classes related to byte streams but the most frequently used classes are, *FileInputStream* and *FileOutputStream*.
- Following program shows use of ByteStream classes.

Program 1) Program that reads content of file.

```
import    java.io.*;
public    class    FileRead
{
  public  static  void  main(String   args[])   throws    IOException
 {
      FileInputStream in = new  FileInputStream("d:\\myfile\\input.txt");
      int c;
    try
    {
      while ((c = in.read()) != -1)   //While not end of File
      {
           System.out.print((char)c);
      }
    }
    catch(FileNotFoundException   ex)
    {
         System.out.println(ex);
    }
    finally
    {
        in.close();
    }
  }
}
```

Program 2) Program that write content into file.

```
import      java.io.*;
public   class   FileWrite
{
  public  static  void  main(String args[]) throws IOException
 {
      FileOutputStream   out = new FileOutputStream("d:\\myfile\\write.txt");
      String   str="Box is Heavy";
      int  len=str.length();
      int  i;
    try
   {
     for(i=0;i<=len-1;i++)
     {
       out.write(str.charAt(i));
     }
    }
    catch(FileNotFoundException   ex)
    {
         System.out.println(ex);
    }
    finally
   {
     out.close();
    }
  }
}
```

Program 3) Program that copies content of file into another file.

```
import    java.io.*;
public  class   FileCopy
{
  public    static   void   main(String args[])   throws   IOException
 {
       FileInputStream      in = new   FileInputStream("d:\\myfile\\input.txt");
       FileOutputStream   out = new  FileOutputStream("d:\\myfile\\output.txt");
       int   c;
     try
    {
       while ((c = in.read()) != -1)       //while there is not end of File
      {
         out.write(c);
      }
     }
     catch(FileNotFoundException     ex)
     {
          System.out.println(ex);
     }
     finally
    {
         in.close();
         out.close();
     }
  }
}
```

2) Character Streams class:

- Java Byte streams are used to perform input and output of 8-bit bytes, whereas Java Character streams are used to perform input and output for 16-bit Unicode.
- There are many classes related to character streams but the most frequently used classes are, *FileReader and FileWriter*.
- Though internally FileReader uses FileInputStream and FileWriter uses FileOutputStream but there is **major difference is that FileReader reads two bytes at a time and FileWriter writes two bytes at a time**.
- Following programs shows use of CharacterStream classes:

Program 1) Program that read the content of file.

```
import    java.io.*;
public class   NewReadFile
{
   public static void main(String args[]) throws IOException
   {
      FileReader  in = new   FileReader("d:\\myfile\\read.txt");
       int   c;
```

```
    try
    {
        while ((c = in.read()) != -1)    //while not end of file
        {
            System.out.print((char)c);
        }
    }
    catch(FileNotFoundException ex)
    {
         System.out.println(ex);
    }
    finally
    {
        in.close();
    }
   }
  }
```

Program 2) Program to write the content in file.

```
import    java.io.*;
public   class   NewWriteFile
{
  public  static  void  main(String   args[])  throws   IOException
  {
    FileWriter    out = new   FileWriter("d:\\myfile\\NewWrite.txt");
    String    str="Box is Heavy";
    int    len=str.length();
    int    i;
    try
    {
        for(i=0;i<=len-1;i++)
        {
                  out.write(str.charAt(i));
        }
    }
    catch(FileNotFoundException ex)
    {
         System.out.println(ex);
    }
    finally
    {
        out.close();
    }
   }
  }
```

Program 3) Program that copies content of file into another file.

```
import     java.io.*;
public   class     NewCopyFile
{
  public static void main(String args[])  throws  IOException
  {
    FileReader    in =   new  FileReader("d:\\myfile\\read.txt");
    FileWriter     out = new  FileWriter("d:\\myfile\\write.txt");
    int     c;
    try
    {
        while ((c = in.read()) != -1)  //while not end of file
        {
          out.write(c);
        }
    }
    catch(FileNotFoundException   ex)
    {
         System.out.println(ex);
    }
    finally
    {
        in.close();
        out.close();
    }
   }
  }
```

Java finally block:

- It is a block used to execute important code such as closing the opened file, closing database connection, database backup code etc.
- Java finally block is always executed whether an exception is handled or not. Therefore, it contains all the necessary statements that need to be printed regardless of the exception occurs or not.

Theory Assignment No: 02

1) What is Array? Explain all types of array in details.
2) How array can be initialized? Explain with example.
3) What is String? How string is initialized?
4) Explain following String class methods with example:
   1) length( ) 2) concat( ) 3) charAt( ) 4) equals( ) 5) compareTo( ) 6) startsWith( )
   7) endsWith( ) 8) indexOf( ) 9) replace( ) 10) substring( ) 11) split( )
5) Explain immutability of string.
6) Explain following methods of StringBuffer class with example:
   1) append( ) 2) insert( ) 3) delete( ) 4) reverse( ) 5) length( ) 6) replace( )
   7) substring( )
7) Give the difference between StringBuffer and StringBuilder class.
8) What is File? How file manipulation is done in Java?

Practical Assignment No: 4

1) Write a program that accept any 10 numbers and print in reverse order.
2) Write a program that finds sum of all elements of array.
3) Write a program that finds sum of even numbers and sum of odd numbers in an array.
4) Write a program which check entered number is present in an array or not.
5) Write a program which find maximum and minimum number in an array.
6) Write a program which print given number into word.(e.g. 45=FourFive)
7) Write a program which convert decimal number into binary equivalent number.
8) Write a program which convert decimal number into octal equivalent number.
9) Write a program which find sum first and last digit of the entered number.
10) Write a program which sorts array element in ascending order.
11) Write a program which sorts array element in descending order.
12) Write a program which swaps two integer arrays.
13) Write a program which finds the sum of all elements of the matrix.
14) Write a program which prints only diagonal elements of the matrix.
15) Write a program which find sum of diagonal elements of the matrix.
16) Write a program which find sum of each row and each column of the matrix.
17) Write a program to calculate addition of two matrices.
18) Write a program to calculate subtraction of two matrices.
19) Write a program to calculate multiplication of two matrices.
20) Write a program which check entered string is Palindrome or not.
21) Write a program which count total number of vowels present in string.
22) Write a program to read the content of file.
23) Write a program to write the data into file.
24) Write a program to copy the content of one file into another.

UNIT-03. Object-Oriented Programming Overview

What is Object Oriented Programming (OOP)?

- Object oriented programming (OOP) is a programming paradigm (Model) that depends on the concept of **classes and objects**.
- It is used to develop a software program which is simple, reusable using classes, which are used to create individual instances i.e. objects.
- There are many object-oriented programming languages like JavaScript, C++, C#, Java, PHP, Python etc.

Principles of OOP (OOP's Concepts):

- OOP totally depends on several OOP principles like class, object, data encapsulation, data abstraction, polymorphism, inheritance, message passing, Persistence etc.
- Let's see these principles in details-

1) Class:

- 'Class' is user defined data type which is collection of variables and methods.
- That is, these variables are called 'properties or attributes' and methods are called 'actions'
- Also, 'Class' is blue print for object because everything in class can also hold by object. i.e. class decides how the object was ?
- Since, all variables and methods are also available in objects, because they are created from class therefore they are called 'instance variable' and 'instance method'.

Consider, following defined class for 'person'.

```
class           person
{
          int      age;
         String   name;      //Properties-  i.e. instance variables of class

        void    talk( )    // action- i.e. instance method
       {
             System.out.println("Hi, My name is "+name);
             System.out.println("My age is "+age);
        }
}
```

In above, example *person* is class that consists of instance variables (properties) *age, name* and having instance method (action) *talk( )*.

2) Object:

- 'Object' is basic run time entity that holds entire data (variables and methods) of class.
- Object is also called as 'instance' of class.
- When object is created then that created object is stored in 'Heap area' by JVM.
- Also, whenever an object is created then JVM allocates separate memory reference (address) for created object called 'hash code'
- This 'hash code' is unique hexadecimal number created by JVM for object.
- We can also see *hash code* of every created object using *hashCode()* method of *Object* class of *java.lang* package.

Syntax to create object:

```
class_name      obj= new     class_name( );
```

Here, *class_name* is name of the class which is an identifier
obj is created object of class.

Purpose to create object:

There are following purpose to create the object:

1) To store entire data (variables and methods) of class.
2) To access variable or to make call for methods of class from outside class.

Following program demonstrate the use of *hashCode()* method.

```
Class       Demo
{
        public   static   void   main(String [ ] args)
        {
                Demo   a=new   Demo();
                Demo   b=new   Demo();
                System.out.println("First Object's Hash Code="+a.hashCode( ));
                System.out.println("Second Object's Hash Code="+b.hashCode( ));
        }
}

OUT PUT:
        First Object's Hash Code=53ab83406
        Second Object's Hash Code=1b6164678
```

3) Data Encapsulation:

- The binding of data and functions into single unit is called "Data encapsulation". And that keeps both data and functions safe from outside the world and misuse.
- Data encapsulation lead to the important OOP concept of data hiding and is accomplished by class. Because, by default all the data of class is 'default' therefore this data is only accessed within package and it is not accessed by outside the world (Other package).
- Java supports the properties of encapsulation and data hiding through the creation of user-defined types, called 'classes'
- A benefit of encapsulation is that it can reduce system complexity, and thus increases robustness, by allowing the developer to limit the interdependencies between software components.
- This feature supports the data security in OOP.
- Encapsulation in Java is a mechanism of wrapping the data (variables) and code acting on the data (methods) together as a single unit. In encapsulation, the variables of a class will be hidden from other classes, and can be accessed only through the methods of their current class. Therefore, it is also known as data hiding.
- To achieve encapsulation in Java −
  - ➢ Declare the variables of a class as private.
  - ➢ Provide public setter and getter methods to modify and view the variables values.

4) Data Abstraction:

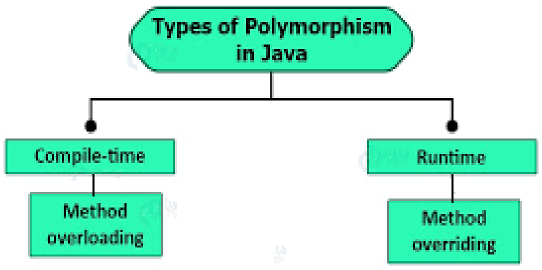
- Data abstraction refers to, providing only essential information to the outside word and hiding their background details.
- This is closely related to encapsulation because abstraction is implemented by encapsulation.
- In Java, classes provide great level of data abstraction. They provide sufficient public methods to the outside world to play with the functionality of the object and to manipulate object data. Without actually knowing how class has been implemented internally.
- If we have to give accessibility to some data of the class then it has to be placed in public section and those we have to make hide from outside world then it is placed in private section of class. Thus, Classes makes data abstraction with the help of access specifier or visibility mode.
- E.g.
- Let's take one real life example of a TV. Which you can turn on and off, change the channel, adjust the volume, and add external components such as speakers, VCRs, and DVD players.
- But you do not know its internal details such as,
  - ▪ you do not know how it receives signals over the air or through a cable
  - ▪ How it translates them, and finally displays them on the screen.
- Thus we can say, a television clearly separates its internal implementation from its external interface and you can play with its interfaces like the power button, channel changer, and volume control without having zero knowledge of its internals

5) Polymorphism:

- The word polymorphism is made up of two Greek words “poly” and “morphism”.
- “Poly” means many and “morphism” means forms, so polymorphism means many-forms.
- In OOP, we implement many functions with same name but these functions are differ from each other according to their signature. (**Signature** refers to type of argument and number of argument accepts by the functions)
- In OOP, polymorphism means “one name” for multiple functions that have different behaviors.
- E.g. Consider following prototype of functions:

```
int     add( );
void    add(int);
int     add(int, int);
```

- In above example, many functions having same name but all these differ from each other by signatures therefore here polymorphism occurs.
- Following figure shows polymorphism with its types:



Basically polymorphism having two types:

1) Compile Time polymorphism (Static linkage/Static Binding/Early Binding):

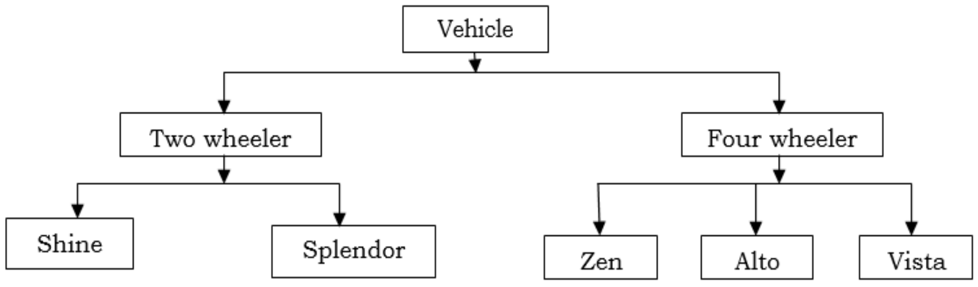
- Definition: Choosing the functions at compile time of program is called as “Compile time polymorphism or Static linkage/Static Binding/Early Binding”
- Compile time polymorphism is achieved by ‘Method overloading’
- Note that: Java **does not** supports for ‘operator overloading’

2) Run Time polymorphism (Dynamic linkage/Dynamic Binding/Late Binding):

- Definition: Choosing the functions at run time of program is called as “run time polymorphism or dynamic linkage/dynamic Binding/late binding”
- Runtime polymorphism is achieved by ‘Method overriding”

6) Inheritance:

- Inheritance is a one of the most important feature of OOP, in which a class can inherits data members and member function (methods) of existing class.
- Inheritance is nothing but the process of deriving (Creating) new classes from existing class. The new class is called as ‘Derived class or sub class or child class’ where as existing class is called as ‘Base class or Super class or parent class’.
- In inheritance, any number of classes can be linked with one another.
- Inheritance supports for “Is-A” relationship.
- While creating new class from existing one, the derived class accesses the data from base class and also add its own features into existing class without modifying base class. Thus, existing class is reused in derived class.
- The main concept behind inheritance is “Reusability” of code is possible. i.e. we use existing program again and again with addition of extra features.
- Inheritance reduces software developing time.
- Inheritance reduces software developing cost also.
- Following figure shows concept of inheritance:

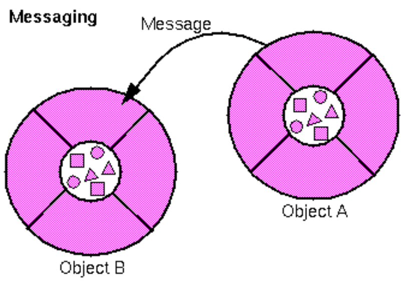


In above figure, ‘Vehicle’ is super base class from which two classes are derived ‘Two wheeler’ and ‘Four wheeler’. Also, ‘Shine’ and ‘Splendor’ are derived classes from ‘Two wheeler’. And ‘Zen’, ‘Alto’ and ‘Vista’ are derived classes from ‘Four wheeler’ base class.

7) Message Passing:

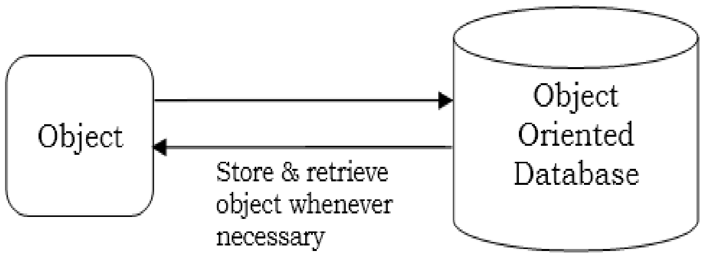
- The act of communicating with an object to get something done is called as “Message Passing”.
- In message passing sending and receiving of information is done by the objects. Therefore it helps in building real life systems.
- Following are the basic steps in message passing.
    - ➢ Creating classes that define attributes and its behavior.
    - ➢ Creating objects from class definitions.
    - ➢ Establishing communication among objects.
- In OOPs, Message Passing involves the name of objects, the name of the function, and the information to be sent.
- Objects, which are usually instances of classes, are used to interact with one another to design strong applications.
- In OOP, each object is capable of **receiving messages**, processing data, and **sending messages** to other objects and can be viewed as an independent ‘unit’ with a distinct role or responsibility.
- Objects react when they receive messages by applying methods on themselves.
- A message is a request to an object to invoke one of its methods; in other words, a message for an object is a simply a call to one of its methods through the object.
- When objects communicate with one another, we say that they send and receive messages.
- E.g. X.getdata(a,b);
- Here, we are passing a message getdata() to the objects ‘X’ with parameters ‘a’ and ‘b’.

Following figure shows messaging between object A and Object B:



8) Persistence:

- In OOP, Persistence is simply related with the ‘objects’ that "Stick around" between the programs.
- This is just serialization (It is the process of translating an ‘object’ into a format that can be stored and reconstructed or retrieve later whenever required) an object from Object Oriented database (OO-database).
- In short, due to persistence concept, OOP language is capable to deal with the object oriented database.



Variable:

- Variable is the name given to the memory location where the value or data will be stored.
- Variable is such thing that has ability to store the value or data.

Types of Variable in Java:

There are three kinds of variable found in Java language:

1) Local Variable
2) Instance Variable (Non-static Variable)
3) Class Variable (Static Variable)

Let’s see all variables in details:

1) Local Variable:

- The variable is declared within body any method is called 'Local Variable'.
- The scope of Local variable is limited to that method in which it is declared i.e. it is not accessible inside other method.

2) Instance Variable (Non-static Variable):

- The variable is declared within class body without using static keyword is called 'Instance Variable'.
- These variables are called instance variable because their content is in object of class.
- Such, instance variable can easily be accessed by class methods of same class in which it is declared.
- Instance variable's Separate copy is given all objects of class.
- Instance variables are stored at Heap area by JVM.
- Following table shows default values allocated by Java compiler to Instance variable.

Data Type	Default Value
byte	0
short	0
int	0
long	0
float	0.0
double	0.0
char	space
String	null
any class type (i.e. Object)	null
boolean	false

3) Class Variable (Static Variable):

- The variable is declared within class body with using static keyword is called 'Class Variable' or 'Static variable'.
- Static variables are common to all objects of class i.e. Common copy of static variable is shared among all objects of class.
- These variables are called class variable because they are in class scope.
- Such, class variable can easily be accessed by class methods of same class in which it is declared.
- Class variables are stored at method area by JVM.
- Following program shows different java varialbes-

```
class          person
{
         static    int      age;     //class variable
         String   name;      //instance variable
        void    talk( )    // action- i.e. instance method
       {
            int    p;     // Local variable
            System.out.println("Hi, My name is "+name);
            System.out.println("My age is "+age);
        }
}
```

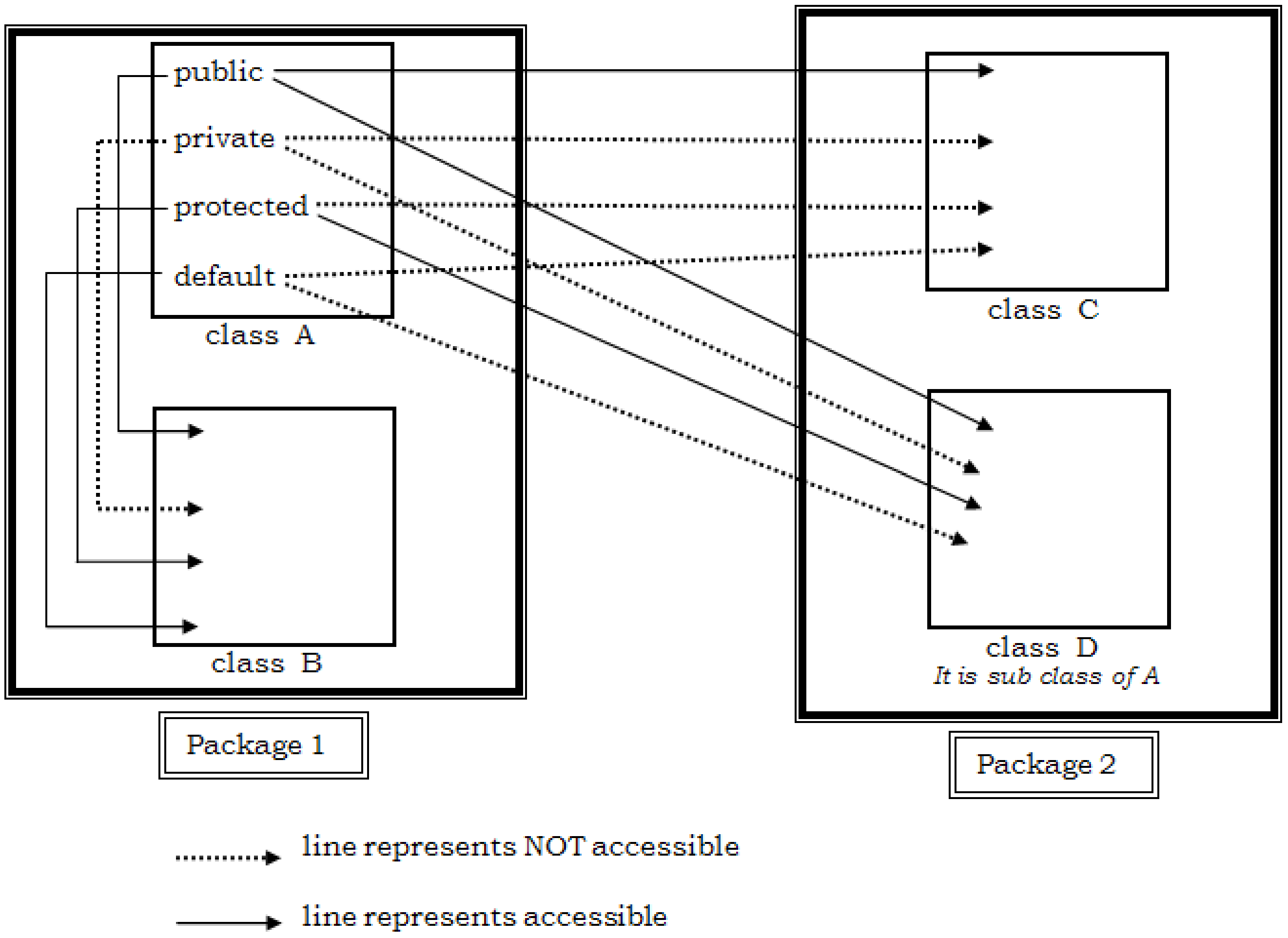
Access Specifier (Member Access Controls) in Java:

- Access Specifier is a keyword that specifies, How to access the members of class or a class itself.
- We can also, use access Specifier before a class and its members.

There are 4 access specifier in Java language VIZ.

1) public 2) private 3) protected 4) default

Following figure shows different access Specifier with their accessibility:



Let us see all these access Specifier in details:

1) public:

- 'public' members of class are <u>accessible everywhere therefore they have **global scope**</u>.
- They are accessible:
  1) Within methods of same class.
  2) Within methods of sub classes.
  3) Within methods of other class of same package.
  4) Within methods of class of other package.

2) private:

- 'private' members of class are only accessible by methods of same class therefore they have class scope. That is private members are not accessible outside the class.
- They are accessible:
  1) Within methods of same class.
- They are NOT accessible:
  1) Within methods of sub classes.
  2) Within methods of other class of same package.
  3) Within methods of class of other package.

3) protected:

- The accessibility of 'protected' members of class is given as follow;
- They are accessible:
  1) Within methods of same class.

2) Within methods of sub classes.
3) Within methods of other class of same package.
4) Within methods of classes of other package (Since, other package class should be sub class)

- They are NOT accessible:
  1) Within methods of classes of other package (If other package class should **NOT** be sub Class)

4) default:

- If no access specifier is written by the programmer then, Java compiler uses 'default' access specifier.
- The 'default' members of class should be accessible by the methods of class of same package i.e. they are not accessible out the package. Therefore 'default' access specifier has 'package' scope.
- They are accessible:
  1) Within methods of same class.
  2) Within methods of sub classes.
  3) Within methods of other class of same package.
- They are NOT accessible:
  1) Within methods of classes of other package.

Methods in Java:

In Java, there are two kinds of methods found;
1) Instance Methods (Non-Static Methods) 2) Class Methods (Static Methods)
Let us see all these methods in details:

1) Instance Methods (Non-Static Methods):

- The method which is defined within class body *without* using keyword 'static' are called as 'Instance methods or Non-static methods'.
- These types of methods are act on 'instance variable' of class.
- They are called 'Instance' method because their content is in instance (object) of class.
- Instance methods are called with the help of object of class using syntax:

object.method(arg1,arg2, ----);

- One specialty of instance method is that they are capable to access instance variable (non-static variable) and class variable (static variable) directly.

Types of Instance Methods:

Further, Instance methods are of two types:
1) Accessor Method 2) Mutator Method
Let's see these methods in details:

1) Accessor Method: (Getter Method)

- This type of instance methods are *only* access or read instance variables i.e. they do not modify value of instance variable are called as "Accessor Method".
- This method is also called the Getter method. Because, this method returns the variable value.

2) Mutator Method: (Setter Method)

- This type of instance methods are access or read instance variables and also modify value of instance variable are called as "Mutator Method".
- This method is also called the Setter method. Because, this method sets the variable value.

Following program shows the use of Accessor and Mutator methods.

```
class    Access
{
        int     age;
        String  name;
        int      getAge( )   //Accessor method
        {
                return(age);
        }
        String   getName()    //Accessor method
        {
                return(name);
        }
        void   setAge(int      x)    //mutator method
        {
              age=x;
        }
void   setName(String    nm)      //mutator method
{
                name=nm;
}
public  static  void  main(String  [ ] args)
{
        Access   t=new   Access();
        t.setAge(55);
        t.setName("RAHUL");
        System.out.println("Age="+t.getAge());
        System.out.println("Name="+t.getName());
        }
}
```

2) Class Methods (Static Methods):

- The method which is defined within class body using keyword 'static' is called as 'class methods or static methods'.
- These types of methods are act on 'class variable (static variable)' of class.
- They are called 'Class' methods because they are defined using 'static' keyword & it is related with class.
- Class methods are called with the help of class name using syntax:

  Class_Name.method(arg1,arg2, ----);
- Class methods are capable to access only class variable (static variable) directly.
- Still, if we want to access instance variable inside class method then it can be accessed only by using object of class.

Following program shows use of class method:

```
public   class      Empolyee
{
    static  int       no;
    static  String   name;
    static  void  get()
    {
         no=65;
         name="KIRAN";
    }
    static   void   show()
    {
         System.out.println("Number="+no);
         System.out.println("Name="+name);
    }
   public   static   void   main(String  []args)
         {
                  Empolyee.get( );              //class methods accessed with class name.
                  Empolyee.show( );
         }
}
```

(**HOME WORK**: *Write Difference between Instance Methods and Class Methods)*

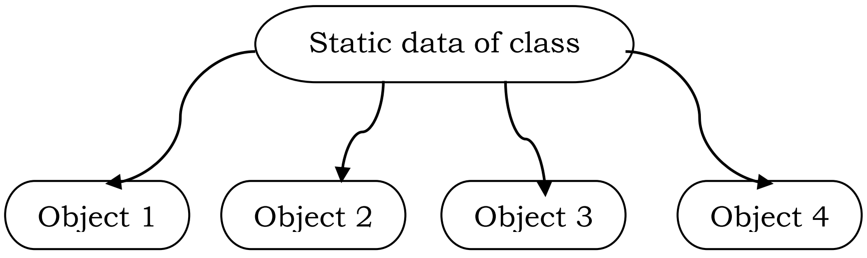
Static data:

- We know that, for single class we can create multiple objects. And each object holds their individual record. But there are some data which are common to all objects in such situation that data can be made as static.
- Once the data is made as static then this data is common to all objects of class.
- In short, static data is common data and which shared by all objects of class.
- Only one copy of static data members is maintained for entire class therefore it is shared among all objects.
- Static data by default initialized to <u>zero</u>.
- Static data are initialized at <u>once therefore it is used like a counter</u>.
- <u>Static data are stored separately rather than as a part of an object</u>.
- Syntax to <u>declare</u> static variable:

```
static      datatype     variable;
```

Here,	'static'	is keyword which is used to declare member as static
	'datatype'	is any valid data type in Java.
	'variable'	is name of static variable.

Following fig. shows sharing of static data by different objects of class:



In above fig. static data of class is common to all objects Object1, Object2, Object3, Object4. Or the same copy of static data is shared among all objects.

Following program demonstrate the use of static variable that counts total object created for class:

```
class    count
{
          static   int     cnt;
          count( )
          {
              cnt++;
          }
          static void show( )
          {
              System.out.println("Total Objects= "+cnt);
         }
```

```
public    static   void   main(String  [ ]args)
{
                    count  a=new count( );
                    count  b=new count( );
                    count  c=new count( );
                    count  d=new count( );
                    count.show( );
}
}
OUTPUT:       Total Objects= 4
```

Method Overloading:

- Method overloading is type of compile time polymorphism where we can take or define many methods having same name but all these methods are differ from each other according to their signature (type of argument and number of argument accept by method).
- It is type of compile time polymorphism, hence all overloaded methods are get selected by JVM at compile time of java program.
- Note that: When we overload the methods then all overloaded methods functionality must be same.
- Following program shows method overloading that finds addition of two numbers:

```
public    class    MethodOverload
{
    int   a,b,c;
    void   add( )
    {
          a=5;
          b=7;
          c=a+b;
          System.out.println("Addition="+c);
    }
    void   add(int   x)
    {
          a=10;
          c=a+x;
          System.out.println("Addition="+c);
    }
```

```
    void    add(int    x,int   y)
    {
          c=x+y;
          System.out.println("Addition="+c);
    }
    public   static   void   main(String    []args)
    {
          MethodOverload     p=new MethodOverload();
          p.add();
          p.add(15);
          p.add(45,65);
    }
}
```

Constructor:

- The constructor is special method whose name is same as that of class name.
- The main task of constructor is to initialize values for object of its class.
- It is called constructor because it construct the values for object of the class.
- The constructors are implicitly invoked by the compiler whenever an object of class is created.
- A constructor is implicitly called whenever the object of class is created.
- Example:

Consider, there is one class having name "person" then its constructor is given as:

```
class              person
{
          person( )        // constructor
          {
                    -------
                    -------
          }
}
```

Properties/Characteristics/Features of Constructor:

1) Constructor name is same as that of class name.
2) Constructor does not return any value but it can accept some parameter as input.
3) Constructor does not have any return type even void.

4) Constructors are implicitly invoked by the compiler whenever object of class is created.
5) Constructor can be overloaded.
6) Constructors are not static.

Types of constructor:

Depending upon arguments passed to constructor, it has two types.

1) Default Constructor
2) Parameterized constructor

NOTE:

1) *Java language does not support for copy constructor.*
2) *If we specify return type for constructor then Java compiler treats that method as normal method.*

1) Default constructor:

- The constructor that does not have any argument or parameter as input, such constructor is called as "Default Constructor".
- Default constructor is implicitly invoked by compiler whenever object of class is crated

Syntax:

```
class     class_name
{
          class_name( )        // default constructor.
          {
                 -------
                 -------
          }
}
```

Following program shows implementation of default constructor:

```
class     person
{
   int      age;
   String   name,add;
   double   sal;
    person( )   // default constructor
   {
     age=70;
     name= "Sachin";
     add= "Pune";
     sal= 4568.50;
    }
```

```
        void  show( )
        {
            System.out.println("Age="+age);
            System.out.println("Name="+name);
            System.out.println("Address="+add);
            System.out.println("Salary="+sal);
        }
public   static  void    main(String [ ] args )
{
        person   x=new person( );   //call to def. constructor
        x.show( );
 }
}
```

2) Parameterized constructor:

- The constructor that accepts any argument or parameter as input, such constructor is called as "parameterized Constructor".
- Parameterized constructor is implicitly invoked by compiler when we pass some values direct to object at the time of its creation.

Syntax:

```
class     class_name
{
         class_name( int   x, String  y)        // parameterized constructor.
          {
                 -------
                 -------
          }
}
```

Following program shows implementation of parameterized constructor:

```
class     person
{
   int      age;
   String   name,add;
   double   sal;
```

```
void  show( )
{
   System.out.println("Age="+age);
   System.out.println("Name="+name);
   System.out.println("Address="+add);
   System.out.println("Salary="+sal);
 }
```

```
   person(int  a, String  n ) //para.constructor
   {
      age=a;
      name=n;
      add= “Pune”;
      sal= 4568.50;
    }
```

```
public   static  void     main(String [ ] args )
{
    person   x=new person(45, “Raj” );  //call to parameter constructor
    x.show( );
 }
}
```

Overloading of Constructor (Multiple Constructor in Class):

- We know that in case of Method overloading, we can define or implement many methods with same name within same class, but all these methods are differ from each other according to their signature (Type of argument and number of argument accept by methods).
- Like method overloading, we are also able to overload the constructor.
- Constructor overloading means implementing or defining multiple constructors for single class. But all these constructors are differing from each other according to their signatures.
- Following program shows constructor overloading:

```
class     person
{
   int       age;
   String   name,add;
   double  sal;

    person( )    // default constructor
   {
      age=70;
      name= “Sachin”;
      add= “Pune”;
      sal= 4568.50;
    }
  person(int  a, String  n ) //para.constructor
  {
      age=a;
      name=n;
      add= “MUMBAI”;
      sal= 54508.75;
  }
```

```
 void  show( )
 {
   System.out.println(“Age=”+age);
   System.out.println(“Name=”+name);
   System.out.println(“Address=”+add);
   System.out.println(“Salary=”+sal);
  }
public   static  void    main(String [ ]  args )
{
    person   p=new  person( );
    person   q=new person(45, “Raj” );
    p.show( );
    q.show( );
 }
}
```

What is the destructor in Java?

- We know that, destructor is a method that destroy (release the memory) the object. It is opposite of constructor.
- It is a special method that automatically gets called when an object is no longer used (i.e. destroyed)
- When an object completes its life-cycle the garbage collector automatically deletes that object and de-allocates or releases the memory occupied by the object.
- In Java, automatically memory management is done by garbage collector and hence Java language does not have concept of destructor. But Java language has finalize( ) method whose work is same as that of destructor.
- For automatically memory management, JVM implicitly called gc( ) method. (we called it explicitly as System.gc())

finalize( ) Method:

- It is difficult for the programmer to forcefully execute the garbage collector to destroy the object.
- But Java provides an alternative way to forcefully execute the garbage collector to destroy the object.
- The Java *Object* class provides the *finalize()* method that works the same as the destructor.
- It is a protected method of the Object class that is defined in the *java.lang* package.
- It can be called only once.
- We need to call the finalize() method explicitly if we want to override the method.
- The syntax of the finalize() method is as follows:

Syntax:

```
        protected void finalize ( )
        {
            //resources to be close
        }
```

Following program shows use of finalize( ) method:

```
public  class  DestructorDemo
{
        DestructorDemo()
       {
            System.out.println("This is Constructor");
        }
public  static  void  main(String[] args)
{
     DestructorDemo de = new   DestructorDemo();
     de.finalize();
}
protected  void  finalize()
{
     System.out.println("Object is destroyed ...");
}
}
```

Parameter passing technique in Java:

- In JAVA, when we pass any variable or any object or even object-reference to method then they are passed as ‘Pass by value’ or ‘Call by value’
- That is, Default parameter passing technique in Java is ‘Pass by value’ or ‘Call by value’.
- Pass by address (Pass by pointer) or call by address (Call by pointer) is invalid in Java, because Java does not support for pointer.
- Everything is passed to method in Java by ‘Pass by value’ or ‘Call by value’ concept.
- In pass by value concept, when we do some operations on formal parameter that does not affects value of actual parameter.
- Following program shows pass by value concept:

```
public  class  parapass
{
            void change(int a)
            {
                        a=90;
            }
```

```
   public static void main(String aaa[])
   {
            int x=7;
            parapass p=new parapass();
            System.out.println(“Before Val=”+x);
            p.change(x);
            System.out.println(“After Val=”+x);
   }
}
```

Passing Object as ‘pass by value’:

- We know that, everything is passed to method in Java by ‘Pass by value’ or ‘Call by value’ concept whether it is normal variable or any objet or any object-reference.

Consider, following program that illustrate passing object as pass by value.

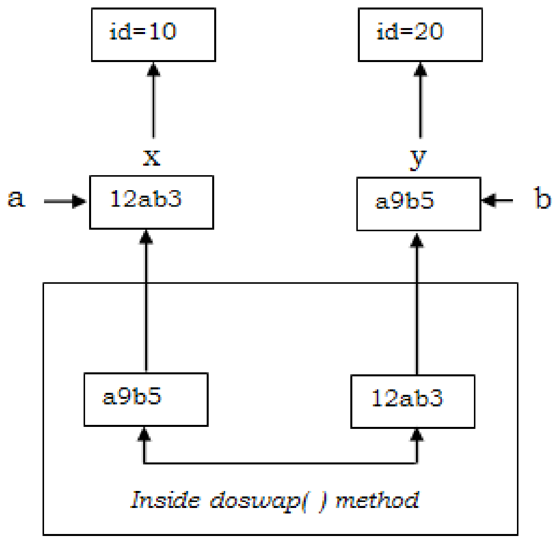
```
class    emp
{
            int   id;
            emp(int   k)
            {
                        id=k;
            }
}
class    ParaPass
{
            emp   c;
            void   doswap(emp  a,emp   b)
            {
                        c=a;
                        a=b;
                        b=c;
            }
```

```
public   static   void   main(String   [ ]sd)
   {
            emp   x=new   emp(10);
            emp   y=new   emp(20);
            ParaPass   p=new   ParaPass();
            System.out.println("Before swapping ID's= "+x.id+"   "+y.id);
            p.doswap(x,y);
           System.out.println("After swapping ID's=  "+x.id+"   "+y.id);
   }
}
```

OUT PUT:

```
Before swapping ID's=10      20
After swapping ID's= 10    20
```

- In above program, we pass two objects *'x', 'y'* of *'emp'* class to *doswap( )* method. Here, objects are pass by value. These passed objects are stored in corresponding formal objects *'a'* and *'b'*.
- We do some swapping mechanism over objects *'a'* and *'b'* inside *doswap( )* method that does not effects on values of *'id'* hold by objects *'x'* and *'y'* because they are passed as pass by value.
- Here, we interchange the objects that ***interchange the references*** of objects. That's why we got same output corresponding to input i.e. no interchanging of *'id'* is done.
- This is shown in following figure.



Static Block:

A static block is block of statements declared with keyword 'static', something like this:

```
static
{
        ------------
        ------------
        ------------
}
```

- *Static block has highest priority* therefore JVM interprets this block first even before main( ) method also.
- In, Java interpretation, JVM first searches for main( ) method, if main( ) found and also static blocks are found then execution of static blocks are done first and then main( ) method is executed.
- If JVM not found main( ) method in java source program then this source code will not execute.
- Also, Single Java program may contains multiple static blocks; in such situation JVM executes them one after another in sequence (FIFO) manner.
- If we have to execute some statements before the main( ) method then they are placed inside static block.

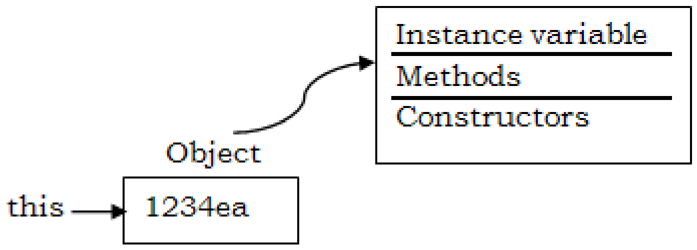
Following program demonstrate use of static block.

```
class      Test
{
    static
    {
       System.out.println("First");
    }
   public  static  void  main(String  [ ] as)
   {
      System.out.println("Main Method");
   }
   static
    {
       System.out.println("Third");
    }
    static
    {
       System.out.println("Second");
    }
 }
```
	OUTPUT:
	```
	          First
	          Third
	          Second
	          Main Method
	```

‘this’ keyword in Java:

- When an object is created to a class, then *JVM implicitly create default reference to created object* and that *default reference* is called ‘this’.
- ‘this’ keyword always refers to *current object of class*.
- ‘this’ is non-static therefore it *cannot use within static method or static block*.
- ‘this’ is hidden parameter for every non-static method of class.
- Generally, we write instance variable, methods and constructors in a class. All these members are stored in object. As well as *all these members are also refers by ‘this’ reference*.

‘this’ reference is shown in following figure:



Use of ‘this’:

We know that ‘this’ is reference for current object of class therefore it is used like object i.e.

1) ‘this’ is used to invoke default constructor of present class using syntax: **this( );**
   Also, we made call to parameterized constructor of present class using syntax: **this (arg1, arg2, ...);**
2) ‘this’ is also used to call instance methods of class using syntax: **this.method(arg1,arg2,...);**
3) ‘this’ is also used to access instance variable of class using syntax: **this.variable;**
4) When instance variables of class and formal parameters of methods are same then that creates confusion. To solve this confusion ‘*this’ keyword is used to access instance variables* of class.

Following program shows use of ‘this’ keyword:

```
class    sample
{
        int     age;
        String  name;
        sample( )
        {
                this(45,"Sachin");        //call to parameterized constructor
        }
        sample( int   age, String    name)
        {
                /* Instance variable & formal parameters are same. Therefore 'this' is used to access instance variable*/
                this.age=age;
                this.name=name;
        }
        void   show( )
        {
                System.out.println("Age="+age);
                System.out.println("Name="+name);
        }
        public   static   void main(String   [ ]args)
        {
                sample    p=new   sample( );
                p.show( );
        }
}
```

OUTPUT:

```
        Age= 45
        Name=Sachin
```

Final class in Java: (We will discuss in Inheritance)

Object class in Java:

- Object class is present in *java.lang* package.
- Every class in Java is directly or indirectly derived from the Object class.
- If a class does not extend any other class then it is a direct child class of Object and if extends another class then it is indirectly derived. <u>Therefore the Object class methods are available to all Java classes</u>.
- Hence, <u>Object class acts as a root of inheritance hierarchy in any Java Program</u>.
- The <u>Object class</u> is the parent class of all the classes in java by default. In other words, it is the topmost class of java.
- The Object class is beneficial if you want to refer any object whose type you don't know. Notice that parent class reference can always refers the child class object, known as "upcasting"
- Let's take an example, there is getObject() method that returns an object but it can be of any type like Employee, Student etc, we can use Object class reference to refer that object. For example:

  Object obj=getObject();   //we don't know what object will be returned from this method
- The Object class provides some common behaviors to all the objects such as object can be compared, object can be cloned, object can be notified etc.

Methods of Object class

- The Object class provides many methods. They are as follows:

Method	Description
public final Class getClass()	returns the Class class object of this object. The Class class can further be used to get the metadata of this class.
public int hashCode()	returns the hashcode number for this object.
public boolean equals(Object obj)	compares the given object to this object.
protected Object clone() throws CloneNotSupportedException	creates and returns the exact copy (clone) of this object.
public String toString()	returns the string representation of this object.
public final void notify()	wakes up single thread, waiting on this object's monitor.
public final void notifyAll()	wakes up all the threads, waiting on this object's monitor.
public final void wait(long timeout) throws InterruptedException	causes the current thread to wait for the specified milliseconds, until another thread notifies (invokes notify() or notifyAll() method).
public final void wait(long timeout,int nanos)throws InterruptedException	causes the current thread to wait for the specified milliseconds and nanoseconds, until another thread notifies (invokes notify() or notifyAll() method).
public final void wait() throws InterruptedException	causes the current thread to wait, until another thread notifies (invokes notify() or notifyAll() method).
protected void finalize()throws Throwable	is invoked by the garbage collector before object is being garbage collected.

Garbage Collection:

- Garbage collection in Java is a process by which Java programs perform automatic memory management. We know that when Java programs compile it would create byte code and that can be interpret by JVM. When Java programs run on the JVM, objects are created on the heap, and memory is allocated to each object. Eventually, some objects will no longer be needed. Then garbage collector finds these unused objects and deletes them to free up memory.
- Garbage Collection is process of retrieving the runtime unused memory automatically. In other words, garbage collection is a way to destroy the unused objects and free the memory.
- To do so, we were using free() function in C language and *delete* in C++. But, in java it is performed automatically. So, java provides better memory management.

- In C/C++, a programmer is responsible for both the creation and destruction of objects. Usually, programmer neglects the destruction of useless objects. Due to this negligence, at a certain point, sufficient memory may not be available to create new objects, and the entire program will terminate abnormally, causing OutOfMemoryErrors.
- But in Java, the programmer need not care for all those objects which are no longer in use. Garbage collector destroys these objects.
- The main objective of Garbage Collector is to free heap memory by destroying unreachable objects.
- The garbage collector is the best example of the Daemon thread as it is always running in the background.

How Does Garbage Collection in Java works?

- Java garbage collection is an automatic process. Automatic garbage collection is the process of looking at heap memory, identifying which objects are in use and which are not, and deleting the unused objects.
- An in-use object, or a referenced object, means that some part of your program still maintains a pointer to that object. An unused or unreferenced object is no longer referenced by any part of your program. So the memory used by an unreferenced object should be destroyed.
- The programmer does not need to mark objects to be deleted explicitly. This task automatic done by garbage collector.

Advantage of Garbage Collection:

- It makes java memory efficient because garbage collector removes the unreferenced objects from heap memory.
- It is automatically done by the garbage collector (a part of JVM) so we don't need to make extra efforts.

Note: The Garbage collector of JVM collects only those objects that are created by new keyword. So if you have created any object without new, you can use finalize method to perform cleanup processing (destroying remaining objects)

Theory Assignment No: 03

1) Explain following OOP's concepts:
   1) Class 2) Object 3) Data Encapsulation 4) Data Abstraction 5) Polymorphism
   6) Inheritance 7) Message Passing 8) Persistence
2) Explain member access controls (Access Specifier) in Java.
3) What is Variable? Explain types of variable.
4) Explain Java methods with types.
5) What is method overloading?
6) What is constructor? Explain types of constructor with example.
7) Write short note on:
   1) finalize( ) method 2) static block 3) 'this' keyword 4) Parameter passing technique
   5) Garbage collection

Practical Assignment No: 05

1) Write a program to overload method that calculate area of triangle.
2) Write a program that demonstrate use of getter and setter method.
3) Write a program to implement default constructor.
4) Write a program to implement parameterized constructor.
5) Write a program for constructor overloading. (Multiple constructors in a class)
6) Write following programs by using Constructor.
   a) Find the largest and smallest number from the array.
   b) Swap two numbers by using parameterized constructor.
   c) Find the sum of diagonal elements of the matrix.
   d) Print the Fibonacci series up to 'n' numbers.
7) Write a program that demonstrate use of finalize( ) method.
8) Write a program that demonstrate use of 'this' keyword.
9) Write a program that demonstrate use of static block.
10) Write a program that demonstrate passing object as pass by value.

04. Inheritance, Interfaces, Packages and Enum.

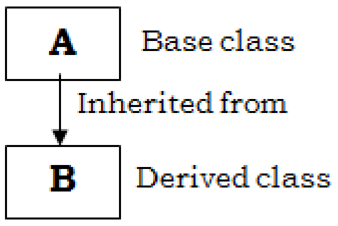
Introduction:

- We know that, inheritance is one of the most important object oriented concept and Java language supports it.
- The main concept behind inheritance is reusability of code (Software) is possible by adding some extra feature into existing software without modifying it.
- In java language, '*Object'* is super class of all classes even your own defined class also.

Definition:

- "The mechanism of creating new class from existing class is called as Inheritance".
- The existing or old class is called as 'Base class' or 'Parent class' or 'Super class' and new class is called as 'Derived class' or 'Child class' or 'Sub class'.
- While deriving the new class from exiting one then, the derived class will have some or all the features of existing class and also it can add its own features i.e sub class will acquires features of base class without modifying it.
- When a new class is derived from exiting class then all public, protected and default data of base class can easily accessed into its derived class where as private data of base class cannot inherited i.e. this private data cannot accessed in derived class.

Consider following example:



In above figure class B is inherited from class A. Therefore class A is called as *Super class* or or *parent class or base class* of class B where as class B is called as *Sub class or child class or derived class*.

Need or Features of Inheritance:

1) It is always nice to use existing software instead of creating new one by adding some extra features without modifying it. This can be done by only inheritance.
2) Due to inheritance software cost is reduces.
3) Due to inheritance software developing time is also reduces.
4) Inheritance allows reusability of code.
5) Due to inheritance we can add some enhancement to the base class without modifying it this is due to creating new class from exiting one.

Syntax to derive new class from existing class:

```
   class        derived_class_name     extends    base_class_name
 {
          //Declaration of Variables and definition of methods.
 }
```

Here;

class	is keyword to define class.
extends	is keyword used to create or derive or extends new class.
derived_class_name	is name of newly created class which is an identifier.
base_class_name	is name of existing class which is also an identifier.

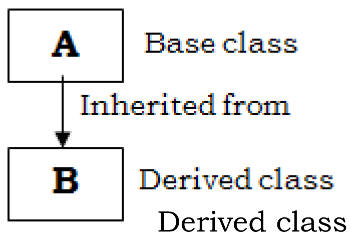
Types (forms) of Inheritance:

Depending upon number of base classes and number of derived classes and their arrangement, inheritance has following types.

1) Single Inheritance:

- In case of Single inheritance there is only *one base class from which we derive only one new class*.

e.g.



Derived class

From above figure we can write single inheritance as:

```
class       A
{
   //Declaration of Variables and definition of methods.
}

class       B   extends       A
{
      //Declaration of Variables and definition of methods.
}
```

Following program shows implementation of single inheritance:

```
import      java.util.Scanner;
class       A
{
            int    x,y;
            Scanner   sc=new   Scanner(System.in);
            void get( )
           {
            System.out.println("Enter two numbers ");
                        x=sc.nextInt( );
                        y=sc.nextInt( );
           }
}
```

```
class      add    extends      A
{
           int z;
           void       doadd( )
           {
                      super.get( );
                      z=super.x+super.y;
                      System.out.println("Addition="+z);
           }
}
public  class   single
{
           public    static   void   main(String [ ] args)
           {
                      add   p=new  add( );
                      p.doadd( );
           }
}
```

'super' keyword:

- If we create an object of super class then we can access only super class members, i.e. we are not able to access the sub class members using object of super class.
- But, *if we create an object of sub class, then all the members of super class as well as sub class are easily accessible by object of sub class*. And that's why we always create an object of sub class instead of super class.
- Some time, both super class and sub class may have members (variable or methods) with same name in such situation, *only members of sub class is accessible with its object*. Here, super class members are not considered. And hence, in such condition if we want to access super class members into its sub class then 'super' keyword is used along with member of base class.

Use of 'super' keyword:

- ➢ 'super' keyword is used to access members of base class into its derived class.
- ➢ We can access, instance variables of base class into its derived class using syntax:

```
super.variable;
```

- We can access, methods of base class into its derived class using syntax:

```
super.method(arg1,arg2, ...);
```

- Note that, *we need not call the default constructor of super class because it is defaultly available to its sub class*. And hence, We can made call to parameterized constructor of base class into its derived class using syntax:

```
super( arg1,arg2,.....);
```

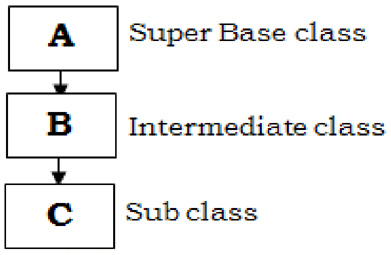
Consider following example that demonstrate use of 'super' keyword:

```
class   A
{
        int   x=55;

}
class   B   extends   A
{
        int    x=70;
        void   get( )
        {
          System.out.println("Super Class variable="+super.x);   //access 'x' of base class using 'super' keyword
          System.out.println("Sub Class variable="+x);
        }
        public  static  void   main(String [ ]  args)
        {
                B  p=new  B( );
                p.get( );
        }
}
OUTPUT:
                Super Class variable= 55
                Sub Class variable= 70
```

2) Multilevel Inheritance:

- In case of multilevel inheritance we can derive *new class from another derived classes, further from derived classes we again derive new class and so on*.
- Also, there are some levels of inheritances therefore it is called as "Multilevel inheritance"

E.g.

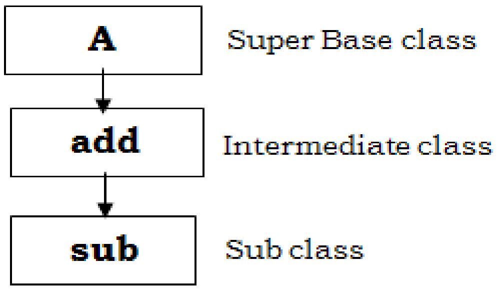


From above figure we can write multilevel inheritance as fallow:

```
class     A
{
   //Declaration of Variables and definition of methods.
}
class     B   extends    A
{
     //Declaration of Variables and definition of methods.
}
class     C    extends   B
{
   //Declaration of Variables and definition of methods.
}
```

Implementation of Multilevel Inheritance is as fallow:

Figure:



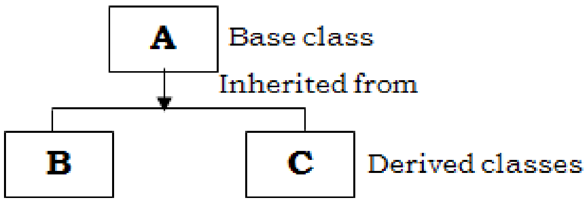
```
import  java.util.Scanner;
class A
{
        int   a,b;
        void get()
        {
          Scanner sc=new Scanner(System.in);
          System.out.println("Enter two no=");
          a=sc.nextInt();
          b=sc.nextInt();
        }
}
class   add  extends   A
{
        int   c;
        void doadd()
        {
           super.get();
           c=super.a+super.b;
           System.out.println("Addition="+c);
        }
}
```

```
class  sub  extends  add
{
        int  c;
        void  dosub()
        {
          super.doadd();
          super.get();
          c=super.a-super.b;
          System.out.println("Subraction="+c);
        }
}
public  class  Multilevel
{
    public static void main(String []args)
    {
        sub  t=new sub();
        t.dosub();
    }
}
```

3) Hierarchical Inheritance:

- In case of hierarchical inheritance there is *only one base class from that class we can derive multiple new classes*.
- The base class provides all its features to all derived classes which are common to all sub classes.
- The hierarchical inheritance comes into picture when certain feature of one level is shared by many other derived classes.

e.g.

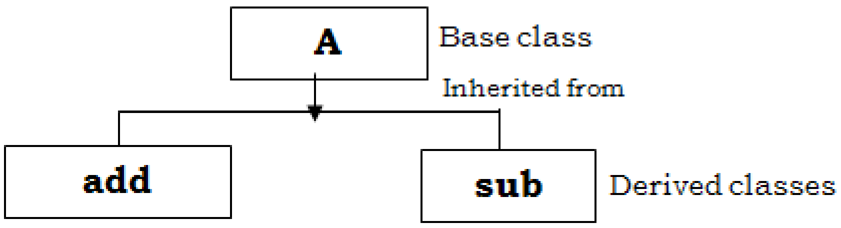


From above figure we can write hierarchical inheritance as fallow:

```
class    A
{
  //Declaration of Variables and definition of methods.
}
class    B    extends    A
{
     //Declaration of Variables and definition of methods.
}
class    C     extends    A
{
   //Declaration of Variables and definition of methods.
}
```

Implementation of Hierarchical Inheritance is as fallow:

Figure:



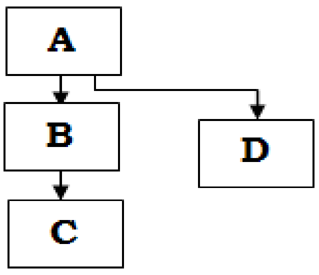
```
import   java.util.Scanner;
class  A
{
        int    a,b;
        void   get()
        {
          Scanner sc=new Scanner(System.in);
          System.out.println("Enter two No=");
          a=sc.nextInt();
          b=sc.nextInt();
        }
}
class add extends A
{
        int    c;
        void    doadd()
        {
          super.get();
          c=super.a+super.b;
          System.out.println("Addition="+c);
        }
}
```

```
class  sub  extends  A
{
        int   c;
        void   dosub()
        {
          super.get();
          c=super.a-super.b;
          System.out.println("Subtraction="+c);
        }
}

public class Hierarchical
{
        public static void main(String []args)
        {
          add  p=new add();
          p.doadd();
          sub q=new sub();
          q.dosub();
        }
}
```

4) Hybrid Inheritance:

- Hybrid inheritance is the *combination of two or more forms of inheritances*.
- To implement hybrid inheritance, we have to combine two or more forms of inheritances and such a combination of different forms of inheritances is called "Hybrid inheritance".

E.g.

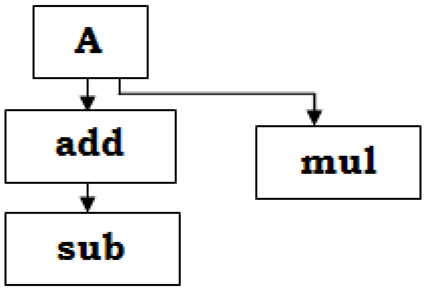


The above inheritance is a combination of multilevel and hierarchical inheritances therefore it is "Hybrid Inheritance".

From above figure we can write hybrid inheritance as fallow:

```
class    A
{
   //Declaration of Variables and definition of methods.
}
class    B   extends    A
{
     //Declaration of Variables and definition of methods.
}
class    C     extends    B
{
    //Declaration of Variables and definition of methods.
}
class    D   extends    A
{
     //Declaration of Variables and definition of methods.
}
```

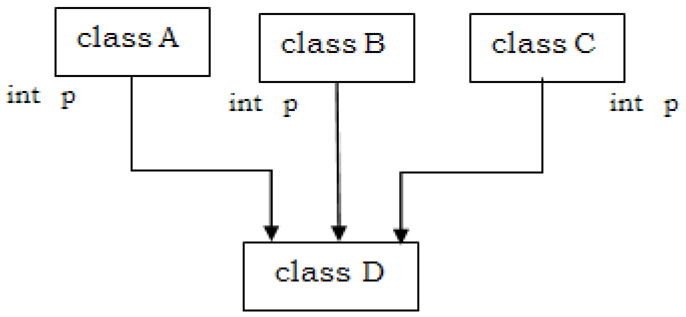
<u>Implementation of Hybrid Inheritance is as fallow:</u>



```
import java.util.Scanner;
class  A
{
        int a,b;
        void get()
        {
          Scanner sc=new Scanner(System.in);
          System.out.println("Enter two No=");
          a=sc.nextInt();
          b=sc.nextInt();
        }
}
class  add  extends  A
{
        int  c;
        void  doadd()
        {
          super.get();
          c=super.a+super.b;
          System.out.println("Addition="+c);
        }
}
class  sub  extends  add
{
        int   c;
        void  dosub()
        {
          super.doadd();
          super.get();
          c=super.a-super.b;
          System.out.println("Subtraction="+c);
        }
}
class  multi  extends  A
{
        int c;
        void  domulti()
        {
          super.get();
          c=super.a*super.b;
          System.out.println("Multiplication="+c);
        }
}
public  class  Hybrid
{
        public static void main(String []args)
        {
          sub  p=new sub();
          p.dosub();
          multi   q=new multi();
          q.domulti();
        }
}
```

5) Multiple Inheritance:

- Multiple inheritance means deriving or creating *only one sub class from multiple super classes*.
- But, Java does not support for multiple inheritance. And that multiple inheritance leads confusion for programmer, which is shown in following figure:



- ➢ In above fig, Super classes i.e. *Class A*, *Class B* and *Class C* has same member 'int p' and from these classes *Class D* is derived or extended. In this case, *Class D* has confusion regarding with copy of 'int p'. That is *Class D* does not know whose copy of 'int p' is accessed.
- ➢ Also, *class D extends A, B, C* such syntax is invalid in Java.
- ➢ If Java does not support for 'Multiple inheritance' then it is not considered as pure OOP language. In fact Java is pure OOP language.
- ➢ But, In Java, *multiple inheritance can be implemented using 'interface'* concept. So let's see the 'Interface' concept in details.

Interface

Introduction:

We know that, Java language does not supports for multiple inheritance but we can implement multiple inheritance using 'interface'.

Interface:

- An 'interface' is collection of only abstract methods.
- All the methods of interface by default *public & abstract*.
- Also, 'interface' has variables but by default all interface variables are *static, final and public*.
- An interface contains only abstract methods which are all incomplete therefore it is not possible to create object of interface.
- But, we can create separate classes from interface where we can implement all the methods of interface. These classes are called as 'implementation' classes.
- Since, 'implementation classes' contains method definition therefore we can create object of implementation classes.
- Every implementation class can have its own implementation of abstract methods of the interface.

Syntax to define interface:

```
interface    Interface_name
{
        datatype     var1=value;
                    ⋮
                    ▼
        return_type     Method1( );
                    ⋮
                    ▼
}
```

here,

'interface' is keyword used to define interface.
'Interface_name' is name of interface which is an identifier.

Syntax to define new class (implementation class) from interface:

```
class     implement_class_name      implements   Interface_name
{
        datatype      var1, varN;

        public  return_type    Method1( )
        {
                --------
                --------
        }

        public  return_type    MethodN( )
        {
                --------
                --------
        }
}
```

here,

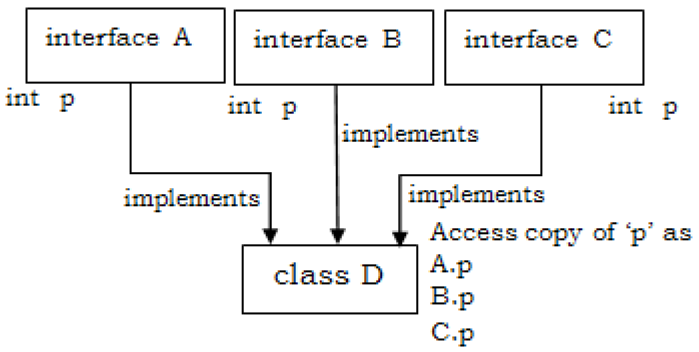
'implements' is keyword used <u>to define or create new class from interface</u>.
'Interface_name' is name of interface from which *implement_class_name* is created.

Characteristic or properties of Interface:

1) An '<u>interface</u>' is a *specification of method prototype* only i.e. methods does not have any body
2) An interface will have <u>zero or more abstract methods</u> which are all <u>public & abstract by default.</u>
3) An interface can have <u>variables</u> which are <u>public, static and final</u> by default. This means all variables of interface are <u>constant therefore we have to assign them value</u>.
4) None of the methods in interface can be private, protected or static.
5) We cannot create an object of interface but we can create reference to interface.
6) All methods of interface should be defined in its implementation classes. <u>If any method is not implemented then that implementation class should be declared as 'abstract'</u>
7) An interface can extend another interface.
8) An interface cannot implement another interface.
9) A class can implement (not extends) multiple interfaces.
   For Ex: class MyClass implements interface1, interface2, interface3 . . . .

Multiple Inheritance using Interfaces:

- Multiple inheritance means deriving or creating *<u>only one sub class from multiple super classes</u>*.
- But, <u>Java does not support for multiple inheritance</u>. And that multiple inheritance leads confusion for programmer, which is shown in following figure:



- In above fig, *class D* is implementation class which is implemented from three interfaces viz. *interface A*, *interface B* and *interface C*. In this case, *Class D* can access individual copy of *'int p'* from all interfaces using *A.p, B.p and C.p*. Because, by default all <u>variables of interface are static</u> therefore they are <u>accessed by interface name</u>. Thus, confusion regarding copy of *'int p'* in *Class D* is solved.
- Also, *class D implements A, B, C* such syntax is valid in Java.
- Thus, In Java*<u>, multiple inheritance is implemented using 'interface'</u>* concept.

Following program shows multiple inheritance using interface.

```
interface   A
{
            int   p=10;
         void  calculate( );
}
interface   B
{
            int   p=20;
         void  calculate( );
}
interface   C
{
          int   p=30;
          void  calculate( );
}
```

```
class   D   implements   A,B,C
{
    public   void  calculate( )
     {
        int z;
        z=A.p+B.p+C.p;
        System.out.println("Addition="+z);
      }
}
class    multiple
{
     public  static  void  main(String s[ ])
      {
         D    obj=new    D( );
        obj.calculate();
       }
}
```

In above program:

class D is implementation class which is implemented from interfaces A,B and C.

The *calculate( )* method in implementation *'class D'* must be defined with **'public'** modifier because it overrides its abstract methods of interfaces A,B,C. And by default all interface methods are **public**.

If do not define it with public modifier then compiler gives compilation error.

Method Overriding:

- When both classes i.e. super class & sub class have same methods with same signatures i.e. both super class and sub class have same method prototype then sub class method overrides (replaces) the super class method such a concept is called as 'Method overriding'
- We know that, super class methods are also accessed by object of sub class. But, in method overriding *both classes has same methods with same signatures in such situation; JVM only executes methods of sub class i.e. here sub class method replaces (overrides) the super class method and hence super class method not executes*.
- That is we can say that, super class method is overridden by sub class method. And hence super class method not executed by sub class object.
- Consider following example which shows Method overriding concept:

```
class      first
{
           void    get(int     p)
           {
                   ----------
                   ----------
                   ----------
           }
}
class      second   extends    first
{
           void   get(int     x)
           {
                   ----------
                   ----------
                   ----------
           }
}
```

- In above example, 'first' is super class whereas 'second' is sub class of 'first' class.
- Also, both classes has *get( )* method with same signatures. And if we made call to *get( )* method *using object of sub class* then only sub class *get( )* method is invoked i.e. sub class method overrides the base class method & *get( )* method of 'first' class never executed. Hence, here method overriding is done.

Following program demonstrate the Method overriding concept using instance method:

```
class     one
{
      void   add( )
       {
         System.out.println("I am in Base class");
       }
}
class  two   extends   one
{
    void  add( )
     {
       System.out.println("I am in Derived Class");
     }
 }
```

```
class    all
{
    public   static   void   main( String  [ ]ar)
   {
        two     t=new   two( );
        t.add( );
    }
}

OUTPUT:
          I am in Derived Class
```

Dynamic Method Dispatch or Runtime Polymorphism in Java:

- We know that, Runtime polymorphism is achieved in Java by 'method overriding'.
- Dynamic method dispatch is the mechanism by which a call to an overridden method is resolved at run time, rather than compile time.
- When an overridden method is called through a superclass reference, Java determines which version (superclass/subclasses) of that method is to be executed based upon the type of the object being referred to at the time the call occurs. Thus, this determination is made at run time.
- At run-time, it depends on the type of the object being referred to (not the type of the reference variable) that determines which version of an overridden method will be executed
- A superclass reference variable can refer to a subclass object. This is also known as "upcasting". Java uses this fact to resolve calls to overridden methods at run time.
- Therefore, if a superclass contains a method that is overridden by a subclass, then when different types of objects are referred to through a superclass reference variable, different versions of the method are executed.
- Following program shows Dynamic method dispatch concept:

```
class A
{
   void  m1()
   {
      System.out.println("Inside A's m1 method");
   }
}

class B extends A
{
                        // overriding m1()
   void  m1()
   {
      System.out.println("Inside B's m1 method");
   }
}

class C extends A
{
                      // overriding m1()
   void  m1()
   {
      System.out.println("Inside C's m1 method");
   }
}
```

```
class    Dispatch
{
   public static void main(String   args[])
   {
      A   x = new A();        // object of type A
      B   y = new B();       // object of type B
      C   z = new C();       // object of type C
      A   ref;     // reference of type A
      ref = x;     // ref refers to an A object
      ref.m1();  // calling A's version of m1()
      ref = y;   // now ref refers to a B object
      ref.m1();  // calling B's version of m1()
      ref = z;   // now ref refers to a C object
     ref.m1();   // calling C's version of m1()
   }
}
```

OUTPUT:

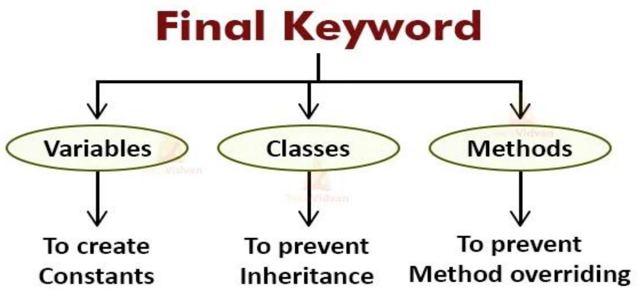
```
Inside A's m1 method
Inside B's m1 method
Inside C's m1 method
```

Difference between method overloading and method overriding:

Method Overloading	Method Overriding
1) Defining multiple method with same name with different signature is called 'Method overloading'	1) Defining multiple method with same name with same signature is called 'Method overriding'
2) Method overloading is done within same class.	2) Method overriding is done within different classes i.e. in super class & sub class.
3) In method overloading, return type of methods can be same or different.	3) In method overriding, return type of methods must be same.
4) JVM decides which method has to be called depending upon parameters of methods.	4) JVM decides which method has to be called depending upon object of sub class or super class.
5) Method overloading is code development. Same methods are developed to perform same task depending on parameters.	5) Method overriding is code replacement. The sub class method overrides (replaces) the super class method.

'final' Keyword:

- The final keyword is a non-access modifier used for classes, class attributes and class methods, which makes them non-changeable (impossible to inherit or override).
- We can use 'final' keyword *before variable declaration or before method definition or before class definition*. We explain all these as follows:



1) final Variable:

- If we use 'final' keyword before variable declaration then that variable becomes 'final' variable.
- And final variables are treated as constant in Java.
- That is, to declare constant in Java, 'final' keyword is used.
- *final variable* cannot be modified because they treated as constant.

Syntax:

```
final    datatype    name=value;
```

here,

'final' is keyword.
'datatype' is valid data type in java.
'name' is an identifier which is constant name .
'value' is any constant value assigned to final variable.

Ex: final double PI= 3.14;

Here, Incrementation or Decrementation or initialization other value to *'PI'* is invalid because it is 'final' and not modified.

2) final Method:

- If we use 'final' keyword before method definition then that method becomes 'final' method.
- And *final methods* cannot override by its derived or sub class. That is sub class method *cannot replace* code of its base class *final method*.
- *Note that: 'final' methods of base class* ***can be*** *accessible into its derived class*.

Syntax:

```
final    return_type    Method_name( )
{
          ----------
          ----------
          ----------
}
```

Note:

- *Final methods of base class cannot override into its derived class therefore method overriding using final methods are not possible. That is final methods are not overridden.*
- *But, 'final' methods of base class can be accessible into its derived class.*

Program:

Program:- 1)

```
class   A
{
          int   a;
 final   void   get()
 {
          a=10;
 }
}
class  B  extends   A
{
          int   b;
          void   get()
          {
                    b=50;
          }
}
public  class   FinalMethod
{
  public static void main(String []args)
  {
                    B     obj=new   B();
                    obj.get();
  }
}
```

Above program gives compile error as:
get() in B cannot override get() in A
Since, final method cannot override.

Program:- 2)

```
class   A
{
          int a;
         final  void  get()
          {
                    a=10;
          }
}
class B extends A
{

          void   show()
          {
          super.get();    //access final method in sub class
          System.out.println("Value="+super.a);
          }
}
public  class  FinalMethod
{
  public static void main(String []args)
  {
                    B     obj=new   B();
                    obj.show();
  }
}
```

Above program successfully executes. Since, here is no method overriding concept.

3) final Class:

- If we use ‘final’ keyword before class definition then that class becomes ‘final’ class.
- And <u>*final class*</u> prevents inheritance. That is, *we are not able to create or extends new class from ‘final class*’
- If we <u>don’t want to allow sub class to access features of its base class</u> then base class has to made as ‘final’

Syntax:

```
final     class       class_name
{
          ----------
          ----------
          ----------
}
```

Ex.

```
final     class      A
{
          --------
          --------
}
class    B   extends    A            // Is invalid.....
```

Program:- 1)

```
final   class     A
{
          void get()
          {
                    System.out.println("Base class");
          }
}
class  B  extends  A
{
```

Program:- 2)

```
class    A
{
          void get()
          {
                    System.out.println("Base class");
          }
}
class  B  extends  A
{
```

void show()
{
super.get();
System.out.println("Derived class");
}
}
public class FinalClassDemo
{
public static void main(String []args)
{
B obj=new B();
obj.show();
}
}
	void show()
	{
	super.get();
	System.out.println("Derived class");
	}
	}
	public class FinalClass
	{
	public static void main(String []args)
	{
	B obj=new B();
	obj.show();
	}
	}

Above program gives compile error as: **cannot inherit from final A** Since, final class cannot inherited.	Above program successfully executes. Since, here no final class.

Abstract Method and Abstract class

Abstract Method:

- Abstract method is such method that does not have any definition i.e. it has no body.
- Abstract method contains only method header i.e. method prototype.
- Abstract method has no body therefore they are also called 'incomplete method'
- Abstract method should be declared with keyword '*abstract*'
- When abstract method is declared into super class then <u>*it is compulsory to define it into its sub classes*</u>.
- An abstract method is written when <u>*same method has to perform different tasks depending upon object requirements*</u>.

Ex:

```
abstract  class   ABC
{
      abstract  void   get( );
}
```

- In above example, get( ) method is declared as abstract using keyword 'abstract'. And that method does not have any definition.

Abstract class:

- Abstract class is such class that declared with keyword 'abstract' and that *contains zero or more abstract methods*.
- Abstract class may also contain instance variable & concrete methods.
- When abstract method is declared into super class then <u>*it is compulsory to define it into its sub classes*</u>.
- We *cannot create object of Abstract class* i.e. <u>Abstract class cannot instantiated</u>. Because abstract class contains incomplete abstract methods.
- But *we can create <u>reference</u> of Abstract class*. And that *reference* may refer to *object of its sub classes*.
- <u>Abstract class may have constructors</u>. Then, question is that who invoke constructor of abstract class? Since, object of abstract class cannot be created.
- <u>Answer is that</u>: constructors of abstract class is <u>invoked by object of its sub classes</u>.
- <u>We cannot declare a class as *final* and *abstract* at a time.</u> Because, *final* keyword prevents to create sub classes and *abstract* keyword allows super class to have sub classes.

Ex:

```
abstract   class    ABC
{       abstract   void    get( );
        void    show( )
        {
                ----------
                ----------
         }
}
```

- In above example, class *ABC* is declared as abstract using keyword ‘abstract’. And that contains get( ) abstract method and show( ) concrete method.

Following program demonstrate the use of abstract class and abstract methods.

```
abstract   class   MyBase
{
         abstract   void   calculate(double x);
}
class   Square   extends   MyBase
{
         void    calculate(double  a)
         {
            System.out.println("Square="+(a*a));
         }
}
class   squareRoot   extends   MyBase
{
    void    calculate(double  a)
     {
    System.out.println("SquareRoot="+Math.sqrt(a));
     }
}
```

```
class   Cube   extends   MyBase
{
         void  calculate(double  a)
         {
           System.out.println("Cube="+(a*a*a));
         }
}
class    abstractDemo
{
         public  static  void  main(String as[ ])
         {
                  Square         t1=new  Square ();
                  squareRoot  t2=new   squareRoot ();
                  Cube   t3=new   Cube ();
                  t1.calculate(5);
                  t2.calculate(36);
                  t3.calculate(4);
         }
}
```

Difference between Abstract class and Interface:

Abstract Class	Interface
1) An abstract class contains some abstract methods and also some concrete methods.	1) An interface only contains abstract methods.
2) An abstract class contains instance variable	2) An interface cannot contain instance variables. It contains only constants.
3) By default abstract methods of abstract class is not public.	3) By default abstract methods of interface is public.
4) All abstract methods of abstract class should be implemented into its sub classes.	4) All abstract methods of interface should be implemented into its implementation classes.
5) Abstract class is declared using ‘abstract’ keyword.	5) Interface is declared using ‘interface’ keyword.
6) Methods in abstract class can be private, protected or static.	6) Methods in interface cannot be private, protected or static.
7) Abstract class may have ‘constructors’	7) Interface does not have ‘constructors’

Packages:

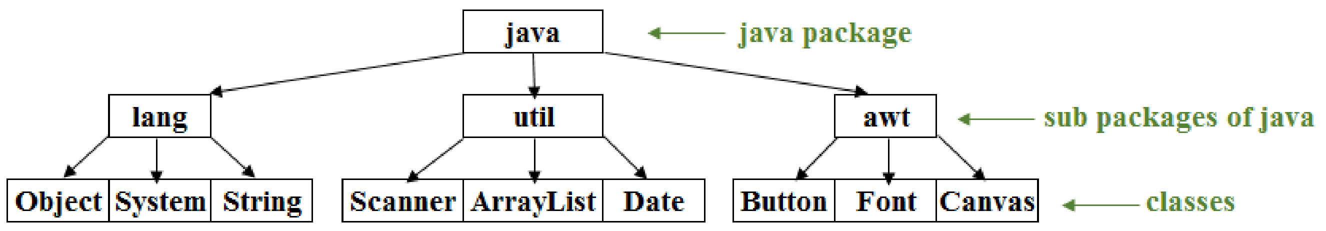
- A java package is a directory or folder which is group of similar types of classes, interfaces and sub-packages.
- We use packages to avoid name conflicts, and to write a better maintainable code.
- **Advantage of Java Package:**
  1) Java package is used to categorize the classes and interfaces so that they can be easily maintained.
  2) Java package provides access protection.
  3) Java package removes naming collision.
- Packages are divided into two categories:
  1) Built-in Packages (packages from the Java API)
  2) User-defined Packages (create your own packages)

Let’s see these types in details-

1) Built-in Packages:

- Built-in packages are those packages which are predefined in Java Library (in JDK).
- This library contains methods which are useful for managing input, database programming, and much more.
- This library is divided into packages and classes. Means we can either import a single class (along with its methods and attributes), or a whole package that contain all the classes that belong to the specified package.
- There are many built-in packages such as java, lang, awt, javax, swing, net, io, util, sql etc.

- Following diagram shows built-in packages:



- To use a class or a package from the library, we use the **import** keyword as follow:

```
import    package.name.Class;     // Import a single class
import    package.name.*;         // Import the whole package
```

Import a single Class:

- We can import single class as follow:

```
import    java.util.Scanner;     // Import a single class
```

In the example above, java.util is a package, while Scanner is a class of the java.util package.

Import a whole package:

We can import whole java.util package as follow:

```
import    java.util.*;      // Import whole util package
```

2) User defined Packages:

- The packages which are created or defined by user is called “User defined packages”.
- To create user defined package, **package** keyword is used as follow:

```
package       packageName;
```

- We create user defined package **mypack** as follow:

```
package    mypack;
public   class        MyPackageClass
 {
   public static void main(String[] args)
  {
    System.out.println("This is my package!");
  }
}
```

Compiling user defined package:

Syntax:

```
D:\2101>javac  -d   directoryWithPath    JavaSourceProgram.java
```

The **-d** keyword specifies the destination for where to save package and the class file.
We can use any directory name, like d:\2101   etc.
or, if we want to keep the package within the same directory then use the dot sign "**.**" at directory.

Example: Above MyPackageClass.java package source file can be compiled as-

```
D:\2101>javac  -d    .   MyPackageClass.java
```

➢ This forces the compiler to create the "mypack" package within same directory.
➢ After successful compilation of given program, a new folder was created, called "mypack" that contains MyPackageClass.

Note: The package name should be written in lower case to avoid conflict with class names.

Running user defined package:

Syntax:

```
D:\2101>java   packageName.ByteCodeFileName
```

Example: Above package can be run as follow:

```
D:\2101>java   mypack.MyPackageClass
```

Creating and using (importing) User defined package:

- **Step 1:** Create java source program with following code:

```
package       newpackage;
public   class    all
{
      public   double   add(double    a, double    b)
       {
              return(a+b);
        }
      public   double   sub(double    a, double    b)
       {
              return(a-b);
        }
       public   double   multi(double    a, double    b)
       {
              return(a*b);
        }

       public   double   div(double    a, double    b)
       {
              return(a/b);
        }
}
```

- **Step 2:** Save above program with **all.java** in your local directory.
- **Step 3:** Compile program as: d:\2201>javac -d . Multiplication.java
  After compiling above program it would create **newpackage** package in your local directory with **Multiplication.class** byte code.
- **Step 4:** Now to use this package in our program. Create new java source program and import **newpackage** package as-

```
import       newpackage.all;
class    UseDemo
{
       public   static void main(String   []arg)
       {
               all    m=new   all();
              double   p=m.add(15,4);
              double   q=m.sub(55,7);
              double   r=m.multi(5,4);
              double   s=m.div(65,7);
              System.out.println(“Add=”+p);
              System.out.println(“Sub=”+q);
              System.out.println(“Multi=”+r);
              System.out.println(“Division=”+s);
        }
}
```

- **Step 5:** Now compile above program as-   d:\2201>javac   UseDemo.java
- **Step 6:** Run above program as-   d:\2201>java UseDemo

Wrapper classes:

- The wrapper class in Java provides the mechanism to convert primitive data types into object and object into primitive data types.
- Wrapper class uses autoboxing and unboxing feature to convert primitives into objects and objects into primitives automatically.
- The automatic conversion of primitive data types into an object is known as autoboxing whereas conversion of object into primitive type is called unboxing.

Use of Wrapper classes in Java:

- Java is an object-oriented programming language, so we need to deal with objects many times like in Collections, Serialization, Synchronization, etc. where we need to use the wrapper classes.

- Change the value in Method: Java supports only call by value. So, if we pass a primitive value, it will not change the original value. But, if we convert the primitive value in an object, it will change the original value.
- Serialization: We need to convert the objects into streams to perform the serialization. If we have a primitive value, we can convert it in objects through the wrapper classes.
- Synchronization: Java synchronization works with objects in Multithreading.
- java.util package: The java.util package provides the utility classes to deal with objects.
- Collection Framework: Java collection framework works with objects only. All classes of the collection framework (ArrayList, LinkedList, Vector, HashSet, LinkedHashSet, TreeSet, PriorityQueue, ArrayDeque, etc.) deal with objects only.

- Following eight classes of the java.lang package are known as wrapper classes in Java. The list of eight wrapper classes are given below:

Primitive Type	**Wrapper class**
boolean	Boolean
char	Character
byte	Byte
short	Short
int	Integer
long	Long
float	Float
double	Double

Autoboxing:

- The automatic conversion of primitive data type into its corresponding wrapper class is known as "Autoboxing". For example, byte to Byte, char to Character, int to Integer, long to Long, float to Float, boolean to Boolean, double to Double, and short to Short referred as Autoboxing.
- Wrapper class Example: Primitive to Wrapper

```
   //Java program to convert primitive data types into objects
   public class AutoBox
{
   public static void main(String args[])
{
   int          a=20;
   boolean   b=false;
   Integer     j=a;      //autoboxing, compiler will write Integer.valueOf(a) internally
   Boolean    k=b;  //autoboxing
   System.out.println("j="+j);
   System.out.println("k="+k);
   }
}
```

Unboxing:

- The automatic conversion of wrapper type into its corresponding primitive type is known as "unboxing".
- It is the reverse process of Autoboxing. Wrapper class Example: Wrapper to Primitive

```
//Java program that converts object to primitive data types
public  class  UnBox
{
   public static void main(String    args[])
{
   Integer   a=new    Integer(3);
   int     j=a;            //unboxing, compiler will write a.intValue() internally
   Boolean   b=new  Boolean(false);
   boolean    c=b;
   System.out.println("int="+j);
   System.out.println("boolean="+c);
   }
}
```

Enumerations:

- The Enumerations or Enum in Java is a data type which contains a 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), season (SPRING, SUMMER, WINTER, and AUTUMN or FALL), colors (RED, YELLOW, BLUE, GREEN, WHITE, and BLACK) etc. According to the Java naming conventions, we should have all constants in capital letters. So, we have enum constants in capital letters.
- Java Enums can be thought of as classes which have a fixed set of constants (a variable that does not change).
- The Java enum constants are static and final implicitly. It is available since JDK 1.5.
- Enums are used to create our own data type like classes. The enum data type (also known as Enumerated Data Type) is used to define an enum in Java. Unlike C/C++, enum in Java is more powerful. Here, we can define an enum either inside the class or outside the class.

Points to remember for Java Enum:

- Enum improves type safety.
- Enum can be easily used in switch.
- Enum can be traversed.
- Enum can have fields, constructors and methods.
- Enum may implement many interfaces but cannot extend any class because it internally extends Enum class

Example of Java Enum:

```
class      EnumDemo
{
    enum  direction {NORTH,SOUTH,EAST,WEST };
    public  static  void   main(String[] args)
   {
      for (direction   s : direction.values())
      System.out.println(s);
    }
}
```

Note: The enum type has a values() method, which returns an array of all enum constants. This method is useful when you want to loop through the constants of an enum

Theory Assignment No: 04

1) What is Inheritance? Write need or features of inheritance.
2) Explain Single inheritance with example.
3) Explain Multilevel inheritance with example.
4) Explain Hierarchical inheritance with example.
5) Explain Hybrid inheritance with example.
6) How multiple inheritance creates confusion or problem in Java?
7) What is Interface? List out characteristics or properties of interface.
8) How multiple inheritance is implemented using interface in Java?
9) Explain Method overriding with example.
10) What is Dynamic method dispatch? Explain with example.
11) Write difference between method overloading and method overriding.
12) Write difference between Abstract class and interface.
13) What is wrapper class? Write its use.
14) What is Autoboxing and Unboxing? Explain with example.
15) Write short note on:
    a) ‘final’ keyword
    b) ‘super’ keyword
    c) Abstract method
    d) Abstract class
    e) Packages
    f) Enum

Practical Assignment No: 06

1) Write a program to implement single inheritance.
2) Write a program to implement multilevel inheritance.
3) Write a program to implement hierarchical inheritance.
4) Write a program to implement hybrid inheritance.
5) Write a program to implement multiple inheritance using interface.
6) Write a program that demonstrate use of method overriding.
7) Write a program that demonstrate dynamic method dispatch.
8) Write a program that demonstrate use ‘final’ keyword.
9) Write a program that demonstrate Autoboxing concept.
10) Write a program that demonstrate Unboxing concept.
11) Write a program that demonstrate use of **enum** data type.
12) Write a program that creates user defined package and use it in new java source program.
13) Write a program demonstrate the use of abstract class and abstract methods.

05. Exception Handling, Thread, Networking

Introduction:

While doing programming in Java, there is possibility to happen errors. These errors may be encountered at two times-
1) Compile time error
2) Runtime error
Let's see these types of errors in details-

1) Compile time error:

- These type of errors detected by compiler at compile time of program.
- Such type of errors are due to wrong syntax of programming languages.
- Or when programmer violate the rules of writing Java statements.
- These compiler error indicates something that must be fixed before the code can be compiled.
- All these errors are detected by compiler and thus are known as "compile-time errors"
- Most frequent compile time errors happen if we make following mistake:
  - ➢ Missing Parenthesis (})
  - ➢ Use of variable without declaring it.
  - ➢ Missing semicolon at end of statement.
- Note that such type of errors are handled by compiler and it generate corresponding error message such that we can correct it.
- These type of errors are not harmful for computer because it is easily detected by compiler at compile time.

2) Run time error (Exception):

- Exception is an unwanted or unexpected event, which occurs during the execution of a program i.e at run time and that disturbs the normal flow of the program's instructions.
- An exception is a problem or error that arises during the execution of a program (program running).
- Exceptions are run-time abnormal conditions that a program encounters during its execution.
- An exception is a response to an exceptional situation that arises while a program is running, such as an attempt to divide a number by zero.
- When an Exception occurs the normal flow of the program is disturbed and the program/Application/software terminates abnormally that cause's application failure which is harmful, which is not recommended, therefore, these exceptions should to be handled.
- An exception can occur for many different reasons. Following are some scenarios where an exception occurs.

➢ A user has entered an invalid data.
➢ Attempt to open a file that cannot be found.
➢ A network connection has been lost in the middle of communications.

What happen if we do not handle exceptions?

- When an exception occurred, if we don't handle it, the program terminates abnormally and remaining program code will not get executed that cause's application failure which is harmful.

Example-

- We know that, an array is of fixed size and each element is accessed using the indices. Consider, we have created an array with size 7. Then the valid expressions to access the elements of this array will be a[0], a[1], .. to a[6] (i.e. length-1).
- Whenever, we used a negative index value or, the index value greater than the size of the array, then there will be exception occurs.

Types of Exceptions:

Java has two types of exceptions.
1. Checked exception
2. Unchecked exception

Let's see these exception in details-

1) Checked Exceptions:

- The exception which occur at compile time of program is called "Checked Exception".
- Java compiler checks program contains the checked exception or not at the time of compilation.
- All these exceptions are subclass of Exception class.
- Developer has overall control on checked exception because these occur at compile time.
- For example: SQLException, IOException, ClassNotFoundException, MalformedURLException etc.

Example: Sample program for checked exception.

```
import   java.io.*;
public  class  CheckExpDemo
{
    public  static  void   main(String args[])
    {
                  FileInputStream    in = new  FileInputStream(“read.txt”);
                  int    c;
                  while((c=in.read())!=-1)
                  {
                         System.out.println((char)c);
                  }
                  in.close( );
    }
}
```

Above program generates compile time exception like-
error: unreported exception IOException
Because IOException must be caught or declared to be thrown.
Here, IOException occur at compile time therefore it is checked exception.

2. Unchecked Exceptions:

- The exception which occur at run time of program is called “Unchecked Exception”.
- JVM detects for unchecked exception at runtime of program.
- Such exception occur due to logical errors or improper use of API.
- Developer has no control on unchecked exception because these occur at run time of program.

For example: ArrayIndexOutOfBoundException, NullPointerException, ArithmeticException.
Example : Sample program for unchecked exception

```
class    UnCheckExpDemo
{
     public  static  void main(String args[])
     {
         int  a = 10;
         int  b = 0;
         int  c = a/b;
         System.out.println(c);
     }
}
```

Above program raise run time exception like-
Exception in thread "main" java.lang.ArithmeticException: / by zero

Difference between error and exception:

Error	Exception
It cannot be caught.	It can be handled by try and catch block.
Errors are by default unchecked type.	Exception can be either checked or unchecked type.
It is defined in *java.lang.Error* package.	It is defined in *java.lang.Exception* package.
Errors are generally caused by environment in which application is running.	Exceptions are generally caused by application itself.
Example: StackOverflowError, OutOfMemoryError etc.	**Example:** SQLException, IOException, ClassCastException, ArrayOutOfBoundException

Built-in exceptions:

- Built-in exceptions are the exceptions which are available in Java libraries. These exceptions are suitable to explain certain error situations. Below is the list of important built-in exceptions in Java.

1. ArithmeticException

It is thrown when an exceptional condition has occurred in an arithmetic operation.

2. ArrayIndexOutOfBoundsException

It is thrown to indicate that an array has been accessed with an illegal index. The index is either negative or greater than or equal to the size of the array.

3. ClassNotFoundException
This Exception is raised when we try to access a class whose definition is not found
4. FileNotFoundException
This Exception is raised when a file is not accessible or does not open.
5. IOException
It is thrown when an input-output operation failed or interrupted
6. InterruptedException
It is thrown when a thread is waiting, sleeping, or doing some processing, and it is interrupted.
7. NoSuchFieldException
It is thrown when a class does not contain the field (or variable) specified.
8. NoSuchMethodException
It is thrown when accessing a method which is not found.
9. NullPointerException
This exception is raised when referring to the members of a null object. Null represents nothing
10. NumberFormatException
This exception is raised when a method could not convert a string into a numeric format.
11. RuntimeException
This represents any exception which occurs during runtime.
12. StringIndexOutOfBoundsException
It is thrown by String class methods to indicate that an index is either negative than the size of the string

Examples of Built-in Exception:

```
// Java program to demonstrate ArithmeticException
class      ArithmeticException_Demo
{
public  static  void  main(String args[])
{
  try
 {
       int a = 30, b = 0;
       int c = a/b;    // cannot divide by zero
      System.out.println ("Result = " + c);
  }
 catch(ArithmeticException    e)
 {
  System.out.println ("Can't divide a number by 0");
 }
}
}
```
Output:
```
Can't divide a number by 0
```

```
//Java program to demonstrate NullPointerException
class    NullPointer_Demo
{
public static void main(String args[])
{
  try
  {
      String a = null; //null value
      System.out.println(a.charAt(0));
  }
 catch(NullPointerException    e)
 {
     System.out.println("NullPointerException !!");
  }
}
}
```
Output:
```
NullPointerException !!
```

```
//Java program to demonstrate StringIndexOutOfBoundsException
class      StringIndexOutOfBoundDemo
{
  public static void main(String args[])
  {
     try
    {
        String a = "Box is Heavy ";  // length is 12
        char c = a.charAt(20);     // 20th index element
        System.out.println(c);
     }
     catch(StringIndexOutOfBoundsException    e)
    {
System.out.println("StringIndexOutOfBoundsException");
    }
 }
}
```
Output:
```
StringIndexOutOfBoundsException
```

```
//Java program to demonstrate NumberFormatException
class     NumberFormat_Demo
{
public static void main(String args[])
{
try
{
     int   n= Integer.parseInt ("GOOD") ;
     System.out.println(n);
  }
catch(NumberFormatException     e)
 {
  System.out.println("Number format exception");
}
}
}
```
Output:
```
Number format exception
```

```
// Java program to demonstrate ArrayIndexOutOfBoundException
class      ArrayIndexOutOfBound_Demo
{
public static void main(String args[])
{
 try
{
     int a[] = new  int[5];
     a[6] = 9;   // assigning 9 element at 6th index but array size is 5
 }
catch(ArrayIndexOutOfBoundsException     e)
{
   System.out.println ("Array Index is Out Of Bounds");
}
}
}
```

Output:

```
Array Index is Out Of Bounds
```

```
// Java program to demonstrate FileNotFoundException
import     java.io.*;
class      FileNotFoundException _Demo
{
public static void main(String args[])
{
 try
{
    FileReader   in=new FileReader("Read.txt");
}
catch(FileNotFoundException      e)
{
   System.out.println("File Not found..");
}
}
}
```

Output:

```
File Not found..
```

Exception Handling in Java:

- Exception handling is the mechanism to handle the abnormal termination of the program without failure.
- By handling exception we can protect application from failure.
- In Java, there are different kinds of exceptions which are handled by **Exception** class which was derived from **Throwable** class.
- The Exception class Hierarchy in Java is shown as-

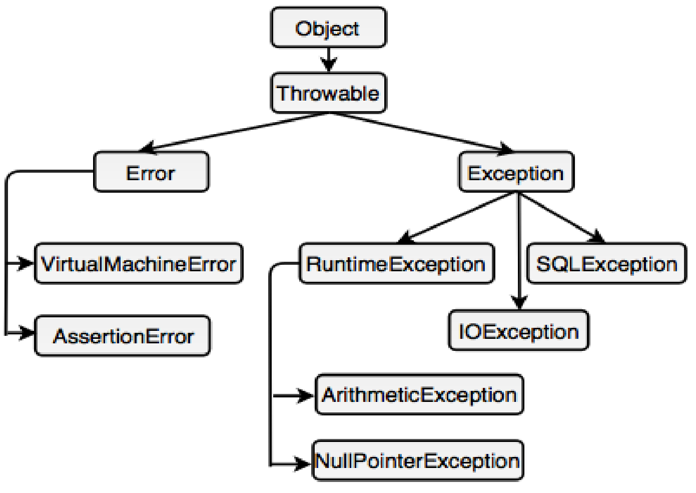


Fig: Exception Hierarchy

- To protect application it is necessary to handle the exception.
- In java, exceptions are handled using five keywords which are as follow-
  1) **try**
  2) **catch**
  3) **throw**
  4) **throws**
  5) **finally**

Let's see these keywords in details-

1) The try block:

- The try block contains the code or statements that might be possibility to occur an exception.
- The try block must have least one catch block or finally block.

2) The catch block:

- A catch block must be declared after try block. It contains the error handling code.
- A catch block executes if an exception occurs in corresponding try block.

- Syntax of try and catch:

```
try
{
     //code that cause exception;
}
catch(Exception_type      e)
{
     //exception handling code
}
```

Example: Following example shows use of try and catch block in java.

Program 1)

```
class      TryCatchDemo
{
        public  static  void  main(String args[])
        {
             int    a  = 30, b = 0;
             int    res;
             try
             {
                   res = a / b;
             }
             catch(ArithmeticException        ae)
             {
                   System.out.println("Divided by zero: "+ae);
             }
        }
}
```

Output :

```
Divided by zero: java.lang.ArithmeticException: / by zero
```

Program 2)

```
class     ExceptionDemo
{
   public static void main(String args[])
   {
        try
     {
     int    arr[] = new int[5];
     arr[2] = 5;
     System.out.println("Access element two: " + arr[2]);
     arr[7] = 10;    //invalid
    System.out.println("Access element seven: "+ arr[7]);
     }
   catch(ArrayIndexOutOfBoundsException     e)
   {
        System.out.println("Exception thrown:" + e);
    }
   }
}
```

Output :

```
Access element two: 5
Exception thrown:
java.lang.ArrayIndexOutOfBoundsException: 7
```

Multiple catch blocks:

- In a single program, there is a possibility to occur multiple exceptions of different types. Then to handle such multiple exceptions, we requires multiple catch blocks.
- A single try block has multiple catch blocks to handle different types of exceptions.
- Syntax:

```
try
{
     // code which generate exception
}
catch(Exception_type1      e1)
{
     //exception handling code
}
catch(Exception_type2       e2)
{
     //exception handling code
}
```

Example: Following program illustrating multiple catch blocks that handles different exception.

```
import   java.util.*;
public  class     MultiCat
{
      public static void main(String args[])
      {
                        Scanner   sc=new Scanner(System.in);
                        int     a,b,c;
              try
              {
                System.out.println("Enter Two NO=");
                a = sc.nextInt();
                b = sc.nextInt();
                c = a / b;
               System.out.println("Result = "+c);
              }
              catch(ArithmeticException    e)
              {
                    System.out.println("Number cannot divide by zero.");
              }
              catch(InputMismatchException    e)
              {
                      System.out.println("Enter only numeric value.");
              }
      }
}
```

Output:

Enter Two NO= 6 2 Result = 3	Enter Two NO= 7 0 Number cannot divide by zero.	Enter Two NO= 5 k Enter only numeric value.

3) The finally block:

- The code present in finally block will always be executed even if try block generates some exception.
- Finally block must be followed by try or catch block.
- It is used to execute some important code like closing file, closing database connection, taking back up etc.
- Finally block executes after try and catch block.
- Syntax:

```
try
{
      // code
}
catch(Exception_type1   ex)
{
      // catch block1
}
catch(Exception_type2    ex)
{
      //catch block 2
}
finally
{
      //finally block
      //always execute
}
```

Example: A program to implement finally block

```
class    FinallyTest
{
      public   static  void   main(String args[])
      {
            int arr[] = new   int[5];
```

```
          try
          {
               arr[7]=10;
          }
          catch(ArrayIndexOutOfBoundsException        ai)
          {
               System.out.println("Exception thrown : " + ai);
          }
          finally
          {
               System.out.println("The finally statement is executed");
          }
     }
}
```

Output:

```
Exception thrown : java.lang.ArrayIndexOutOfBoundsException: 7
The finally statement is executed
```

4) 'throw' keyword in Java:

- The **throw** keyword in Java is used <u>to explicitly or forcefully</u> throw the exception.
- That is with the help of **throw** any kind of exception like ArithmeticException, FileNotFoundException etc. can be explicitly thrown.
- It can be used for both checked and unchecked exception.

Syntax:

throw new Exception_subclass;

Here,

throw is keyword that throws exception.
new is operator
Exception_subclass is any exception class.

Example: A program to illustrate throw keyword in java

```
public   class   throwDemo
{
   public static void main(String args[])
   {
          try
         {
              throw   new   ArithmeticException("Not valid ");
         }
         catch(ArithmeticException   e)
         {
              System.out.println("ArithmeticException occur "+e);
         }
   }
}
```

OUTPUT:

```
ArithmeticException occur java.lang.ArithmeticException: Not valid
```

5) 'throws' keyword in Java

- The throws keyword is generally used for handling checked exception.
- If we do not want to handle exception by try and catch block, then it can be handled by throws.
- Without handling checked exceptions program can never be compiled.
- The **throws** keyword is always <u>added after the method signature</u>.
- Syntax of throws:

```
Return_type      method_name() throws    Exception_Class_Name
{
     Body of method
}
```

Example: A program to illustrate uses of throws keyword

```
import    java.io.*;
class   MyClass
{
 void   method()  throws IOException
 {
  System.out.println("My method…");
 }
}
class     throwsDemo
{
   public static void main(String args[])throws IOException
  {
     MyClass   m=new   MyClass();
     m.method();
     System.out.println("normal flow...");
  }
}
```

Difference between throw and throws keyword in Java:

Throw	Throws
It is used to throw own exception.	It is used when program does not handle exception via try block.
It handles explicitly thrown exceptions.	It handles all exceptions generated by program.
Cannot throw multiple exceptions.	Declare multiple exception E.g. public void method() throws, IOException, SQLException.
It is used within the method.	It is used within method signature.

Creating Custom Exceptions: (User defined exception)

- The exceptions that are created by programmer are known as “Custom exception.”
- These exceptions are created to generate a solution for anticipated errors as per programmer’s need.
- To create user defined exception, “Custom exception class” must be extended from “Exception” class.
- Syntax to create custom exception:

```
class    Custom_Excep_ClassName    extends    Exception
{
      Body of exception class
}
```

Example : Program to create custom exception in Java

```
import java.util.*;
class   custom   extends   Exception
{
   public custom(String   msg)
   {
                  super(msg);
   }
}
public   class   MyExcep
{
   public static void main(String  arg[])
   {
                  Scanner   sc=new Scanner(System.in);
                  int    age;
                  try
                  {
                    System.out.println("Enter your Age");
                    age=sc.nextInt();
                    if(age<18)
                    {
                              throw   new  custom("Age should be greater than 18");
                    }
```

```
                        else
                        {
                                    System.out.println("You are Eligible for voting..");
                        }
            }
            catch(custom   e)
            {
                        System.out.println(e);
            }
}
}
```
OutPut:

Enter your Age 52 You are Eligible for voting..	Enter your Age 15 custom: Age should be greater than 18

Thread:
- We know that, to get result or output from method its statements must be executed. And Thread is such a concept which is responsible to execute the statements of method.
- In short, we can say that because of thread methods of program executes and we will get result.
- C, C++ language supports for single threading concept i.e. to execute entire C, C++ program there is only one thread that execute entire statements of program.
- Java language supports for multithreading concept i.e. to execute Java program there will be allocate more threads that can execute multiple methods of program concurrently. (All methods will executes parallel with the help of threads)

Multithreading:
- Multithreading means executing multiple parts (methods) of the same program concurrently.
- Multithreading is a part of multitasking.
- A program may contain more than one thread and each thread complete its task simultaneously.
- Following diagram shows multithreading:

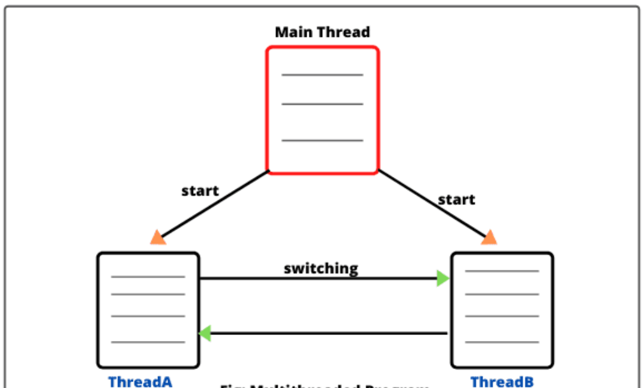


Fig: Multithreaded Program

Life Cycle of Thread:
- Life Cycle of Thread in Java represents all states of a thread from birth to its death.
- Every thread in Java has life cycle which is shown in following diagram:

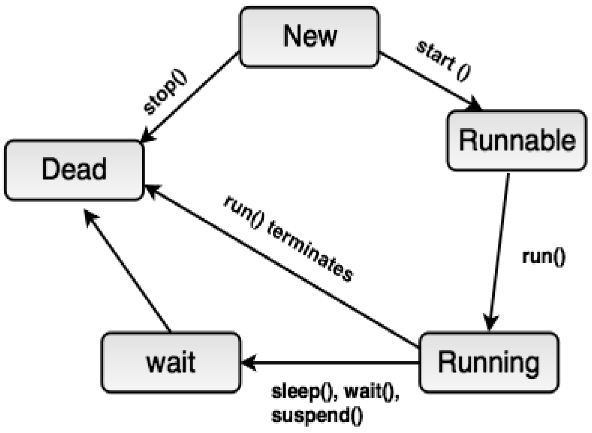


Fig: Life Cycle of Thread

Thread life cycle is explained bellow-

1) **New State:** When we create a thread, then it is said to be in the new state.
2) **Runnable State:** The life of a thread starts after invoking the start () method. The start() method invokes the run () method.
3) **Running State:** When the thread is currently running its task, then it is in running state.
4) **Wait State:** When other threads are holding the resources, the current thread is said to be in wait state
5) **Dead State:** When the thread has completed its task from run () method, then it is said to be in dead state.

Thread Methods in Java:

Following are the some important methods available in thread class.

Methods	Description
public void start( )	This method invokes the **run()** method, which causes the thread to begin its execution.
public void run( )	Contains the actual functionality of a thread and is usually invoked by **start()** method.
public static void sleep (long millisec)	Blocks currently running thread for at least certain millisecond.
public static void yield ( )	Causes the current running thread to halt temporarily and allow other threads to execute.
public static Thread currentThread ( )	It returns the reference to the current running thread object.
public final void setPriority (int priority)	Changes the priority of the current thread. The minimum priority number is 1 and the maximum priority number is 10.
public final void join (long millisec)	Allows one thread to wait for completion of another thread.
public final boolean isAlive ( )	Used to know whether a thread is live or not. It returns true if the thread is alive. A thread is said to be alive if it has been started but has not yet died.
public void interrupt ( )	Interrupts the thread, causing it to continue execution if it was blocked for some reason.

Creating Thread:

There are two ways to create a thread in Java.
1. By extending a class from **Thread** class.
2. By implementing a class from **Runnable** interface.
Let's see these way in details-

1. Extending a class from Thread class:

We can create a new thread by **extending from Thread class**.

- Thread class has constructor and method to create and perform the operation on thread.
- Following program shows creating Thread by extending Thread class

```
class   MyThread  extends   Thread
{
     public  void    run()
     {
              for(int i=1;i<=5;i++)
             {
                   System.out.print(" "+i);
             }
             for(int j=100;j>=95;j--)
            {
                      System.out.print(" "+j);
            }
      }
}
```

```
public   class   ThreadDemo
{
      public static void main(String[] args)
      {
            MyThread   t1 = new  MyThread();
            MyThread   t2 = new  MyThread();
            t1.start();
            t2.start();
      }
}
```

OUTPUT:
Whenever we run this program we will get different-different outputs. Since, Thread t1 and t2 parallely execute same block of code.

2. Implementing a class from Runnable Interface:

- We can also create a new thread by **implementing Runnable interface.**
- Runnable interface is available in *java.lang* package.
- The purpose of Runnable interface is to provide a set of rules common for objects to execute the functionality of thread while they are active.
- Example : Sample program to create thread class by implementing Runnable interface

```
class  MyThread   implements   Runnable
{
     public void run()
     {
         for(int i=1;i<=5;i++)
        {
            System.out.print(" "+i);
         }
        for(int j=100;j>=95;j--)
         {
                     System.out.print(" "+j);
         }
    }
}
```

```
public  class  MyRunThread
{
     public static void main(String[] args)
     {
            MyThread   dt1=new  MyThread();
            MyThread   dt2=new  MyThread();
            Thread  t1 = new  Thread(dt1);
            Thread  t2 = new  Thread(dt2);
            t1.start();
            t2.start();
     }
}
```

OUTPUT:
Whenever we run this program we will get different-different outputs. Since, Thread t1 and t2 parallely execute same block of code.

Thread Priorities:

- Thread priorities is the integer number from 1 to 10 assigned to thread which helps to determine the order in which threads are to be scheduled for running.
- Each thread has a priority. Priorities are represented by a number between 1 and 10. In most cases, the thread scheduler schedules the threads according to their priority (known as preemptive scheduling). But it is not guaranteed because it depends on JVM specification that which scheduling it chooses.
- It decides when to switch from one running thread to another thread.
- Default priority of a thread is 5 (NORM_PRIORITY). The value of MIN_PRIORITY is 1 and the value of MAX_PRIORITY is 10.
- Thread scheduler selects the thread for execution on the first-come, first-serve basis. That is, the threads having equal priorities share the processor time on the first-come, first-serve basis.
- When multiple threads are ready for execution, the highest priority thread is selected and executed by JVM. In case when a high priority thread stops, yields, or enters into the blocked state, a low priority thread starts executing.
- If any high priority thread enters into the runnable state, it will preempt the currently running thread forcing it to move to the runnable state. Note that the highest priority thread always preempts any lower priority thread.

Setter & Getter Method of Thread Priority:

- The Thread class has getter and setter methods related with Thread priority which are discussed bellow-

1) int   getPriority():

- This method of Thread class returns the priority of the given thread.

2) void setPriority(int newPriority):

- This method updates or assign the priority of the thread to newPriority.
- The method throws IllegalArgumentException if the value newPriority goes out of the range, which is 1 (minimum) to 10 (maximum).
- Following program shows use of MAX_PRIORITY, MIN_PRIORITY and NORM_PRIORITY

```
public class ThreadPriorityDemo extends Thread
{
   public void run()
   {
      System.out.println("Priority of thread is: "+Thread.currentThread().getPriority());
   }
   public static void main(String args[])
   {
     ThreadPriorityDemo  t1=new   ThreadPriorityDemo ();
     ThreadPriorityDemo  t2=new   ThreadPriorityDemo ();
     ThreadPriorityDemo  t3=new   ThreadPriorityDemo ();
```

```
      t1.setPriority(Thread.MAX_PRIORITY);
      t2.setPriority(Thread.MIN_PRIORITY);
      t3.setPriority(Thread.NORM_PRIORITY);
      t1.start();
      t2.start();
      t3.start();
    }
}
```

OUTPUT:

```
Priority of thread is: 10
Priority of thread is: 1
Priority of thread is: 5
```

Example : Following program shows use of getPriority() and setPriority() method.

```
class  newThread  extends  Thread
{
                    public void run()
                    {
                       System.out.println(Thread.currentThread()+" has Priority "+Thread.currentThread().getPriority());
                    }
}

public class AllThread
{
   public static void main(String []ar)
   {
                       newThread t1=new newThread();
                       newThread t2=new newThread();
                       newThread t3=new newThread();
                       newThread t4=new newThread();
       t1.setPriority(Thread.MAX_PRIORITY);
       t2.setPriority(Thread.NORM_PRIORITY);
       t3.setPriority(Thread.MIN_PRIORITY);
       t4.setPriority(4);
       t1.start();
       t2.start();
       t3.start();
       t4.start();
   }
}
```

OUTPUT:

```
Thread[Thread-0,10,main] has Priority 10
Thread[Thread-1,5,main] has Priority 5
Thread[Thread-3,4,main] has Priority 4
Thread[Thread-2,1,main] has Priority 1
```

Example : Following program shows execution of thread depending on their priority.

```
public  class  ThreadExec  extends  Thread
{
public  void  run()
{
  System.out.println(Thread.currentThread());
}
public static void main(String[] args)
{
 ThreadExec   t1 = new  ThreadExec();
 ThreadExec   t2 = new  ThreadExec();
 ThreadExec   t3 = new  ThreadExec();

 t1.setPriority(4);
 t2.setPriority(2);
 t3.setPriority(8);
 t1.start();
 t2.start();
 t3.start();
 }
}
```

Note that: By executing above program again and again, we will get different-different results.

OUTPUT:

Thread[Thread-2,8,main] Thread[Thread-0,4,main] Thread[Thread-1,2,main]	Thread[Thread-0,4,main] Thread[Thread-2,8,main] Thread[Thread-1,2,main]	Thread[Thread-2,8,main] Thread[Thread-1,2,main] Thread[Thread-0,4,main]

Thread Synchronization:

- We know that in multi-threaded programs where multiple threads try to access the same resources (like variable, memory, printer etc.) at same time and finally produce incorrect and unpredicted results.
- Therefore it is necessary to allow only one thread can access to the resource at one time where as other threads may remain in waiting state. And this task in Java is done with the help of synchronized block and synchronized method.
- Definition: Thread synchronization is such a mechanism that allows only one thread to act on object or shared resources while other threads are in waiting state. The next thread only allow to enter in shared resources, if and only if current working thread finishes its task.
- In short, because of Thread synchronization multiple threads cannot enters in shared resources at one time.
- Java provides a way of creating threads and synchronizing their tasks using **synchronized blocks**.
- Synchronized blocks in Java are marked with the **synchronized** keyword.
- All synchronized blocks synchronize on the same object can only have one thread executing inside them at a time. All other threads attempting to enter the synchronized block are blocked until the thread inside the synchronized block exits that block.
- Thread synchronization achieved by two ways-
  1) synchronized block 2) synchronized method.
- Let's see these in details-

1) Thread synchronization by synchronized block-

Following is the general form of a synchronized block:

```
synchronized(sync_object)
{
  // Access shared variables and other shared resources
}
```

From above syntax-

- ➢ Only one thread can execute synchronized block at a time.
- ➢ sync_object is a reference to an object whose lock associates with the monitor.
- ➢ The code is said to be synchronized on the monitor object

- This synchronization is implemented in Java with a concept called monitors. Only one thread can own a monitor at a given time.
- When a thread acquires a lock, it is said to have entered the monitor. All other threads attempting to enter the locked monitor will be suspended until the first thread exits the monitor.
- Following is an example of **multi-threading with synchronized block**:

```
class  ThreadJob
{
                public  void  myjob()
                {
                  for(int  i=11;i<=15;i++)
                  {
                            System.out.println(i);
                  }
                }
}
class  newThread  extends  Thread
{
               ThreadJob     TJ;
               newThread(ThreadJob  tj)
```

```
                {
                    TJ=tj;
                }
                public  void  run()
                {
                  synchronized(TJ)
                  {
                    System.out.println(Thread.currentThread());
                    TJ.myjob();
                  }
                }
}
public  class  ThreadSyncDemo
{
    public  static  void  main(String []arg)
    {
                ThreadJob    obj=new ThreadJob();
                newThread  t1=new  newThread(obj);
                newThread  t2=new  newThread(obj);
                newThread  t3=new  newThread(obj);
                newThread  t4=new  newThread(obj);
                t1.start();
                t2.start();
                t3.start();
                t4.start();
    }
}
```

OUTPUT:

```
Thread[Thread-0,5,main]
11
12
13
14
15
Thread[Thread-2,5,main]
11
12
13
14
15
Thread[Thread-1,5,main]
11
12
13
14
15
Thread[Thread-3,5,main]
11
12
13
14
15
```

2) Thread synchronization by Synchronization Method:

- This is another way of doing multithreading with synchronized method.
- If we declare any method using **synchronized** keyword then it is known as “synchronized method”
- The working of synchronized block and Synchronized method is same which is used to lock an object for any shared resource if any thread enters it.
- When a thread invokes a synchronized method, it automatically acquires the lock for that object and releases it when the thread completes its task.

```
class   ThreadDemo2  extends Thread
{
                synchronized  void  threadWork()
                {
                  System.out.println("Active Thread="+Thread.currentThread());
                  for(int   i=11;i<=15;i++)
                  {
                          System.out.println(i);
                  }
                }
                public  void  run()
                {
                    threadWork();
                }
}

public class MultiThread
 {
   public  static  void  main(String arg[] )
   {
                ThreadDemo2    t1=new ThreadDemo2();
                ThreadDemo2    t2=new ThreadDemo2();
                t1.start();
                t2.start();
     }
}
```

OUTPUT:

```
Active Thread=Thread[Thread-0,5,main]
11
12
13
14
15
Active Thread=Thread[Thread-1,5,main]
11
12
13
14
15
```

Thread Communication: (Inter Thread Communication)

- Inter-thread communication in Java is a technique through which multiple threads communicate with each other to achieve specific task.
- It provides an efficient way through which more than one thread communicate with each other by reducing CPU idle time. (CPU idle time is a process in which CPU cycles are not wasted.)
- When more than one threads are executing simultaneously, sometimes they need to communicate with each other by exchanging information with each other. A thread exchanges information before or after it changes its state.
- There are several situations where communication between threads is important.
- For example, suppose that there are two threads A and B. Thread B uses data produced by Thread A and performs its task.
- If Thread B waits for Thread A to produce data, it will waste many CPU cycles. But if threads A and B communicate with each other when they have completed their tasks, they do not have to wait and check each other’s status every time.
- Thus, CPU cycles will not waste. This type of information exchanging between threads is called “inter-thread communication” in Java
- Inter thread communication is used when an application has two or more threads that exchange same information.
- Inter thread communication helps in avoiding thread pooling.

- Inter thread communication in Java can be achieved by using three methods provided by Object class of java.lang package. They are:
  1. wait()
  2. notify()
  3. notifyAll()
- These methods can be called only from within a synchronized method or synchronized block of code otherwise, an exception named *IllegalMonitorStateException* is thrown.
- All these methods are declared as final. Since it throws a checked exception, therefore, you must be used these methods within Java try-catch block.
- Let's see these methods in details-

1. wait() Method:

- wait() method in Java keeps current thread in waiting state.
- This method will throws InterruptedException.
- Various forms of wait() method allow us to specify the amount of time a thread can wait. They are as follows:

Syntax:

public final void wait()
public final void wait(long millisecond) throws InterruptedException
public final void wait(long millisecond, long nanosecond) throws InterruptedException

- All overloaded forms of wait() method throw InterruptedException.
- If time is not specified in the wait() method, a thread can wait for maximum time.

2. notify() Method:

- The notify() method wakes up a single thread that which is already in wait() state. If more than one thread is waiting, this method will awake one of them.
- The general syntax to call notify() method is as follows:

Syntax:

public final void notify()

3. notifyAll() Method:

- The notifyAll() method is used to wake up all threads that are already in wait() state.
- The general syntax to call notifyAll() method is as follows:

Syntax:

public final void notifyAll()

Following program shows inter thread communication between producer and consumer thread

```
class   producer   extends   Thread
{
            StringBuffer    sb;
            producer()
            {
              sb=new   StringBuffer();  //allot memory
            }
            public  void  run()
            {
              synchronized(sb)
              {
                        try
                        {
                           for (int i=1;i<=10 ;i++ )
                          {
                                     Thread.sleep(100);     //that will causes InterruptedException
                                     sb.append(i+" : ");
                                     System.out.println("Producing...."+i);
                          }
                        }
                        catch (InterruptedException      e)
                        {

                        }
                      System.out.println("Production Is over..");
                      sb.notify();   //production is Over & notify to consumer thread
```

```
                        }
                   }
}
class   consumer   extends  Thread
{
                   producer     p;
                   consumer(producer     t)
                   {
                      p=t;
                   }
                   public  void   run()
                   {
                      synchronized(p)
                      {
                                 try
                                 {
                                            p.wait();  // Wait till notification is received
                                 }
                                 catch (Exception e)
                                 {

                                 }
                                 System.out.println("Consumer Receive Production= "+p.sb);
                      }
                   }
}
class   ThreadCommunicate
{
                   public static void main(String []as)
                   {
                      producer     t1=new   producer();
                      consumer    t2=new   consumer(t1);
                      t1.start();
                      t2.start();
                   }
}
```

OUTPUT:

```
Producing....1
Producing....2
Producing....3
Producing....4
Producing....5
Producing....6
Producing....7
Producing....8
Producing....9
Producing....10
Production Is over..
Consumer Receive Production= 1 : 2 : 3 : 4 : 5 : 6 : 7 : 8 : 9 : 10 :
```

Networking in Java:

- Networking is the concept of connecting multiple remote or local networking devices (computer, mobiles, printer, scanner etc.) together. Java program communicates over the network at application layer.
- All the Java networking classes and interfaces use **java.net** package to implement networking application. These classes and interfaces provide the functionality to develop system-independent network communication system.
- The java.net package provides the functionality for two common protocols and they are-

TCP (Transmission Control Protocol)

- TCP is a connection based protocol that provides a reliable flow of data between two devices.
- This protocol provides the reliable connections between two applications so that they can communicate easily.
- It is a connection based protocol.

UDP (User Datagram Protocol)

- UDP protocol sends independent packets of data, called datagram from one computer to another with no guarantee of arrival.
- It is not connection based protocol.

Networking Terminology:

i) Request and Response:

- When an input data is sent to an application via network, it is called request.
- The output data coming out from the application back to the client program is called response.

ii) Protocol:

- A protocol is basically a set of rules and guidelines which provides the instructions to send request and receive response over the network.
- For example: TCP, UDP, SMTP, FTP etc.

iii) IP Address:

- IP Address stands for Internet protocol address. It is an identification number that is assigned to a node of a computer in the network.
- For example: 192.168.2.01
- Range of the IP Address
  0.0.0.0 to 255.255.255.255

iv) Port Number:

- The port number is an identification number of server software. The port number is unique for different applications. It is a 32-bit positive integer number having between ranges 0 to 65535.

v) Socket

- Socket is a listener through which computer can receive requests and responses. It is an endpoint of two way communication link. Every server or programs runs on the different computers that has a socket and is bound to the specific port number.

Java URL Class

- While dealing with network programming in Java, it has URL class.
- URL stands for Uniform Resource Locator.
- The URL provides the logically understandable form of uniquely identifying the resource or address information on the Internet.
- The URL class provides a simple, concise API to access information over the Internet.
- URL Class Methods:

Method	Description
String getAuthority()	Returns the authority part of the URL.
int defaultPort()	Returns the default port number of the protocol associated with given URL.
String getFile()	Returns the file name of the given URL.
String getHost()	Returns the host name of the URL.
String getPath()	Returns the complete path of the given URL.
int getPort()	Returns the port number of the given URL. If it is not applicable, it returns -1.
String getQuery()	Returns the query part of the given URL.
String getProtocol()	Returns the name of the protocol of the given URL.

Following program shows use of different methods of URL class:

```
import    java.net.*;
public   class   urlDemo
{
   public   static   void    main(String args[]) throws     MalformedURLException
   {
      URL    url = new     URL("http://www.google.com/java.htm");
      System.out.println("URL: " + url.toString());
      System.out.println("Protocol: "+url.getProtocol());
      System.out.println("path: " + url.getPath());
      System.out.println("Port: "+url.getPort());
      System.out.println("Host: "+url.getHost());
      System.out.println("Authority: " + url.getAuthority());
      System.out.println("File: "+url.getFile());
      System.out.println("Query: "+url.getQuery());
      System.out.println("HashCode: "+url.hashCode());
      System.out.println("External form: "+url.toExternalForm());
   }
}
```

OUTPUT:

```
URL: http://www.google.com/java.htm
Protocol: http
path: /java.htm
Port: -1
Host: www.google.com
Authority: www.google.com
File: /java.htm
Query: null
HashCode: -1526831008
External form: http://www.google.com/java.htm
```

Theory Assignment No: 05

1) What is Exception? Explain types of exceptions in Java.
2) Write difference between Error and Exception
3) How exceptions are handle in Java? Explain with example.
4) How exceptions are handled with multiple catch blocks.
5) Give the difference between throw and throws.
6) What is Thread? Explain Multithreading.
7) How we can create thread in Java?
8) What is synchronization? How thread is synchronized in Java?
9) Write short note on:

1) finally block	2) throw keyword	3) throws keyword	4) Custom exception
5) Thread life cycle	6) Thread Priority	7) Thread communication	8) URL class in Java.

Practical Assignment No: 07

1) Write a program that demonstrate use of checked exception.
2) Write a program that demonstrate use of unchecked exception.
3) Write a program that shows use of ArrayIndexOutOfBoundsException.
4) Write a program that shows use of ArithmeticException.
5) Write a program that shows use of NullPointerException.
6) Write a program that shows use of NumberFormatException.
7) Write a program that shows use of FileNotFoundException.
8) Write a program that demonstrate use of multiple catch blocks to handle different exceptions.
9) Write a program that demonstrate use of finally block.
10) Write a program that demonstrate use of throw keyword.
11) Write a program that demonstrate use of throws keyword.
12) Write a program that creates user defined exception. (Custom exception)
12) Write a program that shows creating thread by extending Thread class.
13) Write a program that shows creating thread by implementing Runnable interface.
14) Write a program that shows use of MAX_PRIORITY, MIN_PRIORITY and NORM_PRIORITY.
15) Write a program that set user defined priority for thread and retrieve it.
16) Write a program that demonstrate multi-threading using synchronized block.
17) Write a program that demonstrate multi-threading using synchronized method.
18) Write a program that shows inter thread communication.
19) Write a program that shows use of different methods of URL class.

06. Applets, AWT, Swing and Event Handling

Applets:

- An applet is a special kind of Java program that runs in a Java enabled browser. This is the first Java program that can run over the network using the browser. Applet is typically embedded inside a web page and runs in the browser.
- In other words, we can say that Applets are small Java applications that can be accessed on an Internet server, transported over Internet, and can be automatically installed and run as apart of a web document.
- After a user receives an applet, the applet can produce a graphical user interface. It has limited access to resources so that it can run complex computations without introducing the risk of viruses or breaching data integrity.
- To create an applet, a class must extends from **Applet** class which was found under **java.applet** package.
- An Applet class does not have any main() method. But it is viewed using JVM by appletviewr.
- The JVM can use either a plug-in of the Web browser or a separate runtime environment to run an applet application.
- JVM creates an instance of the applet class and invokes init() method to initialize an Applet.

Applet class:

- Applet class provides all necessary support for applet execution, such as initializing and destroying of applet. It also provide methods that load and display images and methods that load and play audio clips.

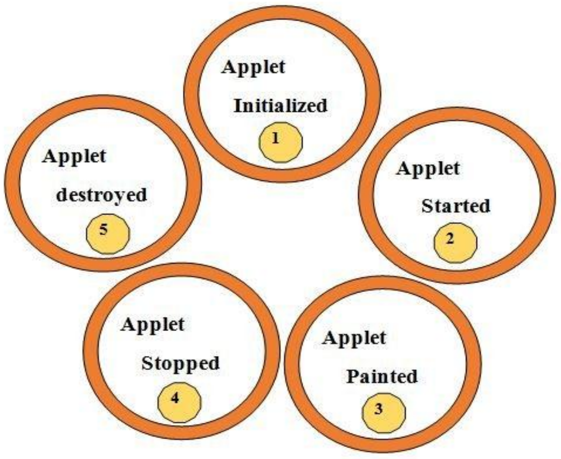
Lifecycle of Java Applet:

- Applet lifecycle describes all states of applet from initialization to destroy.

Following are the stages shows Applet life cycle:

1) Applet is initialized.
2) Applet is started
3) Applet is painted.
4) Applet is stopped.
5) Applet is destroyed.

Following diagram shows Applet life cycle:



Most applets override these four methods. These four methods forms Applet lifecycle.

1) **init() :** init() is the first method to be called. This is where variable or componets of applet are initialized. This method is called only once during the runtime of applet.
2) **start() :** start() method is called after init(). This method is called to restart an applet after it has been stopped.
3) **paint(Graphics g):** paint( ) is used to paint the Applet. It provides Graphics class object that can be used for drawing line, circle, oval, rectangle, arc etc. Graphics componets.
4) **stop() :** stop() method is used to suspend thread that does not need to run the applet.
5) **destroy() :** destroy() method is called when your applet needs to be removed completely from memory.

Note:

- To use above **init( ), start( ), stop( ) and destroy( )** methods, we have to import package **java.applet.Applet;**
- To use **paint( )** method, we have to import package **java.awt.*;**
- The stop() method is always called before destroy() method.

A simple Java applet Program:

```
import   java.awt.*;
import   java.applet.*;
public  class  Simple  extends  Applet
{
   String  msg;
   public  void  init()
   {
                    // set the foreground and background colors.
     setBackground(Color.cyan);
     setForeground(Color.red);
     msg = "Inside init( ) method";
   }
   // Initialize the string to be displayed.
   public  void  start()
   {
     msg =msg+ " Inside start( ) method";
   }
   // Display msg in applet window.
   public void paint(Graphics g)
   {
     msg =msg+" Inside paint( ).";
     g.drawString(msg, 10, 30);
   }
  }
/*
  <applet code="Simple.class" width=300 height=50>
  </applet>
  */
```

How to run an Applet?

There are two ways to run an applet

1. By creating separate html file.
2. By embedding html code to applet java source file.

Let's see ways in details-

1. By creating separate html file:

Follow the steps:

1) Create new java source program (.java file) that contains applet source code. As follow:

```
import   java.awt.*;
import   java.applet.*;
public  class  First  extends  Applet
{
     public   void init( )
    {
           setBackground(Color.white);
    }
     public  void  paint(Graphics g)
    {
         g.drawString("My First Applet", 20, 20);
    }
}
```

2) Save above code with First.java and then compile it as follow:

```
d:\2201>javac   First.java
```

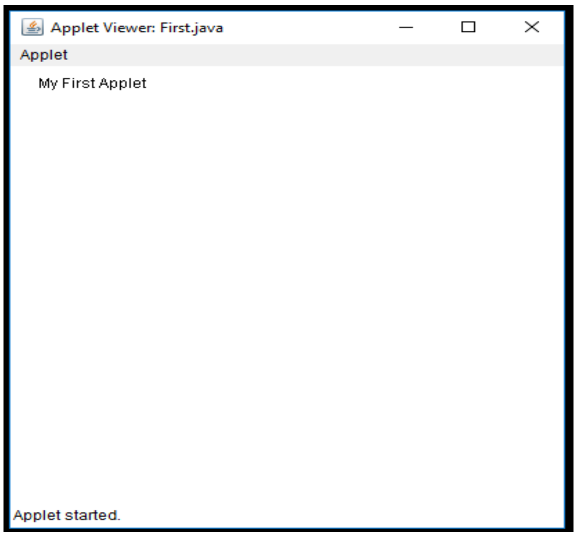
3) Create new separate html file that uses created byte code (First.class) As follow:

```
<html>
<body>
<applet    code= "First.class"  height=400   width=400></applet>
</body>
</html>
```

4) Save above html code with **.html extension**. Let I store it with name **First.html**. And then **interpret this html file with JDK tool appletviewer** as follow:

d:\2201>appletviewer First.html

After executing above applet we will get output as-



```
My First Applet
```

```
Applet started.
```

2. By embedding html code to applet java source file.

Step 1) Create new java source program (.java file) that contains applet source code and also embed html applet code using multiline comments as follow:

```
import   java.awt.*;
import   java.applet.*;
public  class  Second  extends  Applet
{
   public   void init( )
    {
           setBackground(Color.white);
    }
  public  void  paint(Graphics g)
    {
         g.drawString("My Second Applet", 20, 20);
    }
}
/*
<applet    code= “Second.class”  height=400   width=400></applet>
*/
```

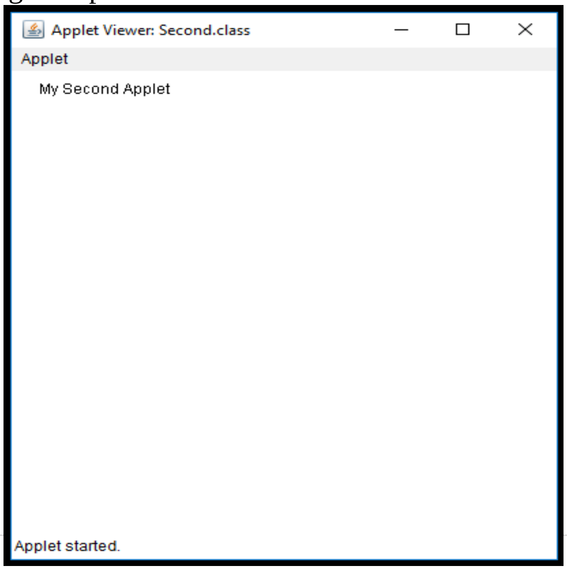
Step 2) Save above code with Second.java and then compile it as follow:

d:\2201>javac Second.java

Step 3) Then **interpret this java file with JDK tool appletviewer** as follow:

d:\2201>appletviewer Second.java

After executing above applet we will get output as-



```
My Second Applet
```

```
Applet started.
```

Displaying Graphics in Applet

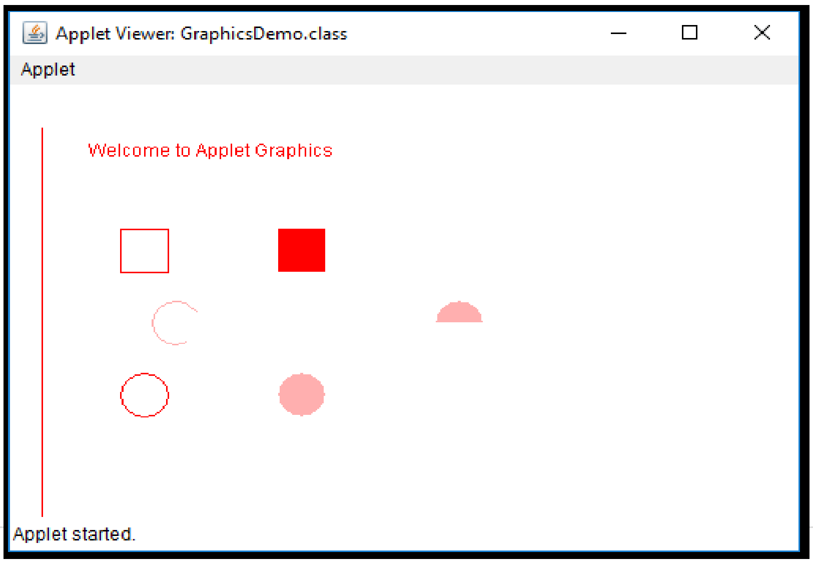
- **java.awt.Graphics** class provides many methods for graphics programming. Commonly used methods of Graphics class are discussed bellow-

1) **drawString(String str, int x, int y):** It is used to draw the specified string.
2) **drawRect(int x, int y, int width, int height):** It draws a rectangle with the specified width and height.
3) **fillRect(int x, int y, int width, int height):** It is used to fill rectangle with the default color Black and specified width and height.
4) **drawOval(int x, int y, int width, int height):** It is used to draw oval with the specified width and height.
5) **fillOval(int x, int y, int width, int height):** It is used to fill oval with the default color and specified width and height.
6) **drawLine(int x1, int y1, int x2, int y2**): It is used to draw line between the points(x1, y1) and (x2, y2).
7) **drawImage(Image img, int x, int y, ImageObserver observer):** It is used draw the specified image.
8) **drawArc(int x, int y, int width, int height, int startAngle, int arcAngle**): It is used draw a circular or elliptical arc.
9) **fillArc(int x, int y, int width, int height, int startAngle, int arcAngle**): It is used to fill a circular or elliptical arc.
10) **setColor(Color c):** It is used to set the graphics current color to the specified color.
11) **setFont(Font font):** It is used to set the graphics current font to the specified font.

Following program shows use of above mentioned graphics methods:

```
import   java.applet.Applet;
import   java.awt.*;
public  class  GraphicsDemo  extends  Applet
{
   public  void init( )
    {
           setBackground(Color.white);
    }
    public  void  paint(Graphics g)
   {
          g.setColor(Color.red);
          g.drawString("Welcome",50, 50);
          g.drawLine(20,30,20,300);
          g.drawRect(70,100,30,30);
          g.fillRect(170,100,30,30);
          g.drawOval(70,200,30,30);
          g.setColor(Color.pink);
          g.fillOval(170,200,30,30);
         g.drawArc(90,150,30,30,30,270);
         g.fillArc(270,150,30,30,0,180);
     }
}
/*<applet code="GraphicsDemo.class" width="300" height="300">
</applet>*/
```

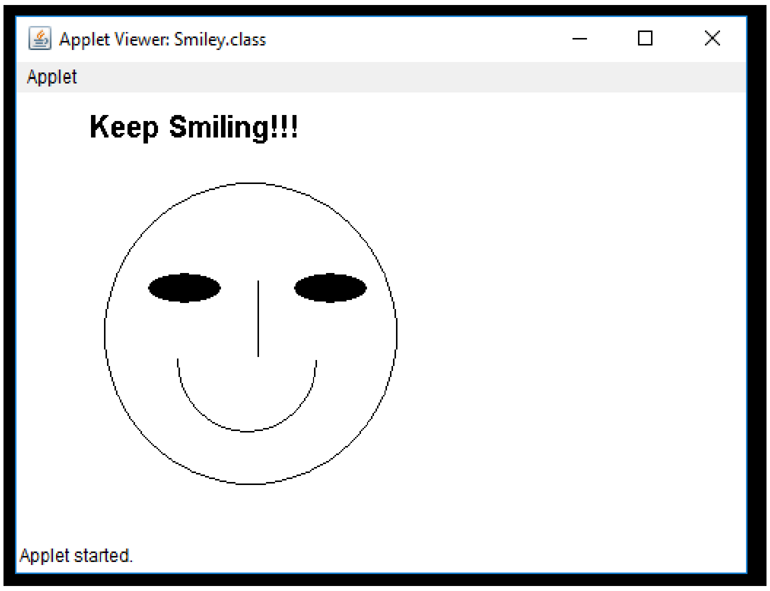
Output of above code:



//Display Smiley in Applet.

```
import   java.awt.*;
import   java.applet.*;
public  class  Smiley  extends   Applet
{
    Font    f = new   Font("Helvetica", Font.BOLD,20);
    public   void init( )
   {
         setBackground(Color.white);
         setFont(f);
   }

                  public void paint(Graphics g)
                  {
                     g.drawString("Keep Smiling!!!", 50, 30);
                     g.drawOval(60, 60, 200, 200);
                     g.fillOval(90, 120, 50, 20);
                     g.fillOval(190, 120, 50, 20);
                     g.drawLine(165, 125, 165, 175);
                     g.drawArc(110, 130, 95, 95, 0, -180);
                  }
}
/*
<html>
<body>
<applet    code   = "Smiley.class"     width= "500"    height = "300"> </applet>
</body>
</html>    */
```

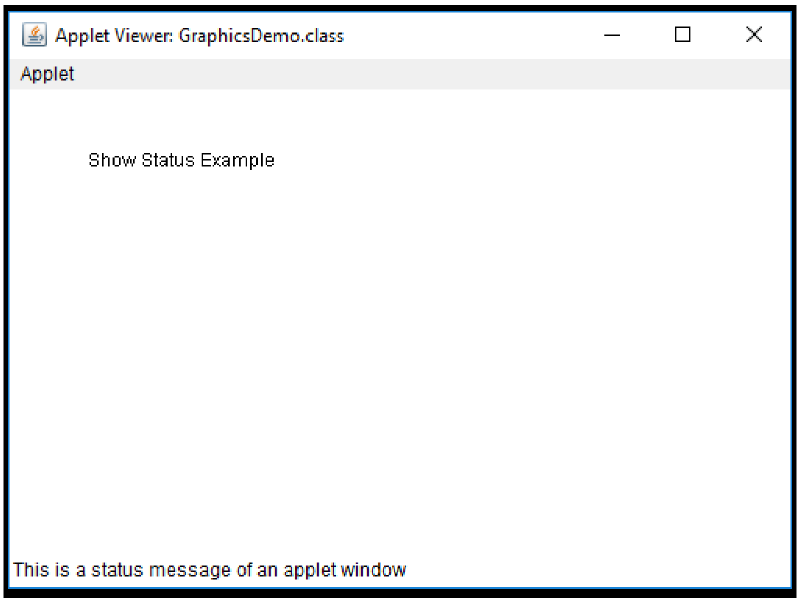
Output of above code:



//Set Status Message in Applet Window Example

```
import     java.applet.Applet;
import     java.awt.Graphics;
 public   class   StatusMessage    extends Applet
{
   public    void   paint(Graphics g)
   {
     g.drawString("Show Status Example", 50, 50);   //this will be displayed inside an applet
     showStatus("This is a status message of an applet window"); //this will be displayed in a status bar of an applet
    }
}
/*
<applet    code= “StatusMessage.class”     width=300   height=500></appet>   */
```

OUTPUT:



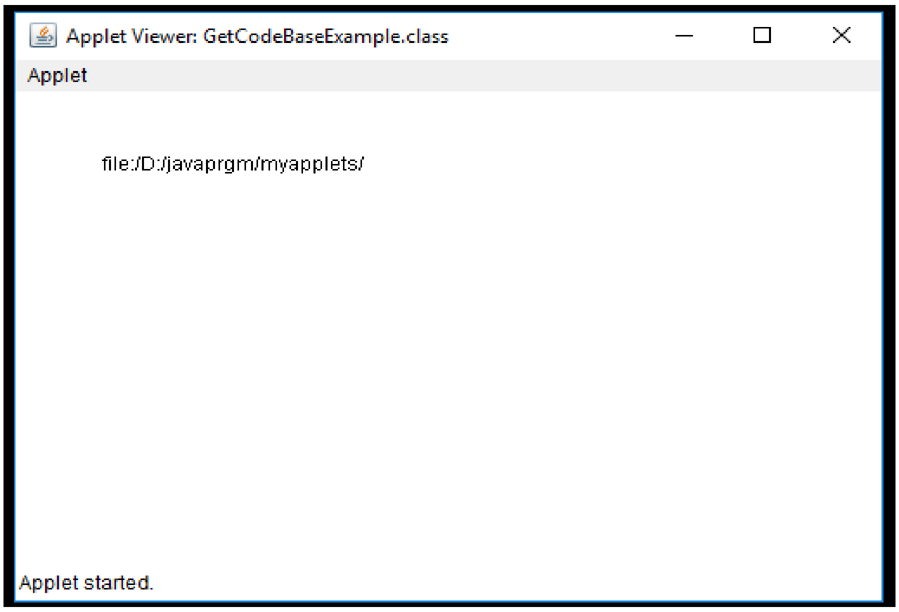
```
Show Status Example
```

```
This is a status message of an applet window
```

//Get Applet Directory URL (i.e path of applet source code) or Code Base Example

```
import    java.applet.Applet;
import    java.awt.Graphics;
import    java.net.URL;
public   class   GetCodeBaseExample    extends Applet
{
 public void paint(Graphics g)
{
    URL   appletDir = getCodeBase();
    g.drawString(appletDir.toString(), 50, 50);
}
}
/*
<applet code="GetCodeBaseExample.class"   width=200   height=200>
</applet>
*/
```

OUTPUT:



```
file:/D:/javaprgm/myapplets/
```

```
Applet started.
```

Displaying Image in Applet:

- Applet is mostly used in games and animation. For this purpose image is required to be displayed on applet.
- The java.awt.Graphics class provide a method drawImage() to display the image.
- Syntax of drawImage() method:

drawImage(Image img, int x, int y, ImageObserver observer);

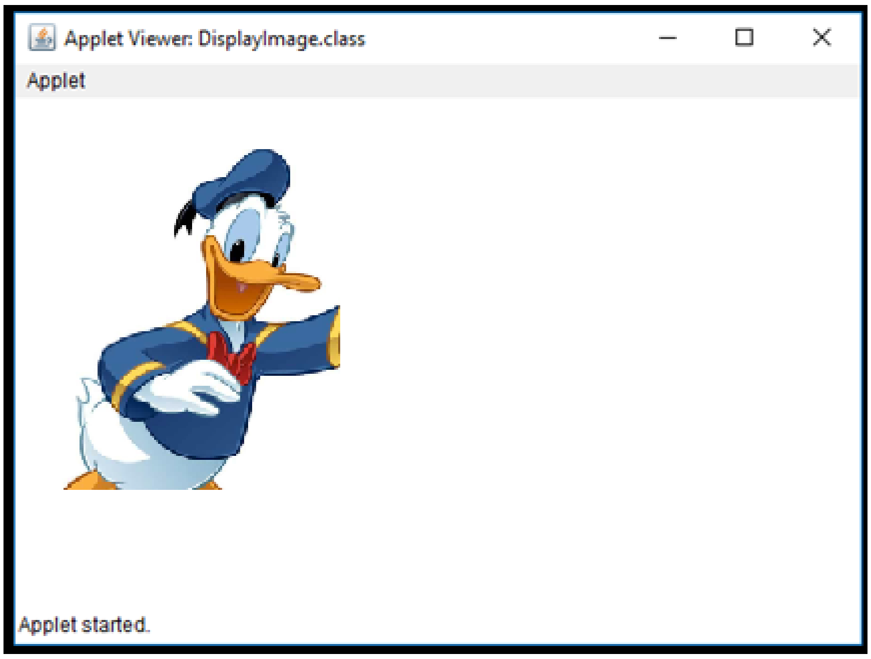
In above syntax-

- We get the object of Image by using getImage() method of Appet class that returns the object of Image.
  Syntax: **getImage(URL u, String imageName);**
- In above syntax we have to pass URL of image as first argument by calling getDocumentBase() method and second argument string which is image name.
- x, y is location of image where image will display
- The 4th is ImageObserver which is an interface. So current class reference would also be treated as ImageObserver which is **this.**

Following program shows displaying an image in Applet

```
import    java.awt.*;
import    java.applet.*;
public  class  DisplayImage  extends  Applet
{
 Image  picture;
 public  void  init()
 {
  picture = getImage(getDocumentBase(),"duck.jpg");
 }
 public  void  paint(Graphics g)
 {
    g.drawImage(picture, 30,30, this);
 }
}
/*<applet code="DisplayImage.class" width="300" height="300">
</applet> */
```

OUTPUT:



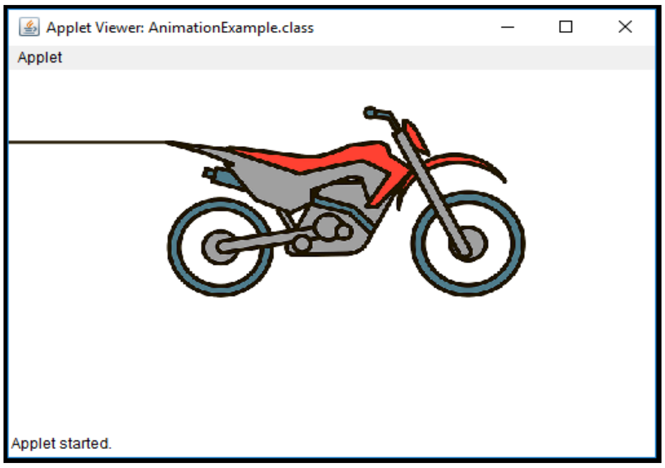
Animation in Applet:

Applet is mostly used in games and animation. For this purpose image is required to be moved.

Following program shows moving image in Applet (Animation using Appet)

```
import   java.awt.*;
import   java.applet.*;
public class AnimationExample extends Applet
{
  Image   picture;
  public  void  init()
 {
    picture =getImage(getDocumentBase(),"bike.png");
 }

  public void paint(Graphics g)
{
     try
         {
          for(int i=10;i<500;i++)
          {
            g.drawImage(picture, i,30, this);
            Thread.sleep(10);
          }
         }
      catch(Exception e){}
}
}
/*<applet code="AnimationExample.class" width="300" height="300">
</applet> */
```

OUTPUT:



Java Event Handling (Event and Listener):

- An event can be defined as changing the state of an object or behavior by performing actions.
- Actions can be a button click, cursor movement, entering a character in Textbox, Clicking or dragging a mouse, keypress through keyboard or page scrolling, etc.
- For event handling in AWT, we have to import package **java.awt.event.*;**
- Java Event classes and Listener interfaces are shown in following table-

Event Classes	Listener Interfaces
ActionEvent	ActionListener
MouseEvent	MouseListener and MouseMotionListener
MouseWheelEvent	MouseWheelListener
KeyEvent	KeyListener

ItemEvent	ItemListener
TextEvent	TextListener
AdjustmentEvent	AdjustmentListener
WindowEvent	WindowListener
ComponentEvent	ComponentListener
ContainerEvent	ContainerListener
FocusEvent	FocusListener

Steps to perform Event Handling:

Following steps are required to perform event handling:

Step 1: Add the components on to the Applet or java GUI application.

Step 2: Register the component with the Listener.

- For registering the component with the Listener, many classes provide the registration methods and these methods are –
- Following are the respective methods of components to register the component with listener:
- **Button**
  - public void addActionListener(ActionListener a){}
- **MenuItem**
  - public void addActionListener(ActionListener a){}
- **TextField**
  - public void addActionListener(ActionListener a){}
  - public void addTextListener(TextListener a){}
- **TextArea**
  - public void addTextListener(TextListener a){}
- **Checkbox**
  - public void addItemListener(ItemListener a){}
- **Choice**
  - public void addItemListener(ItemListener a){}
- **List**
  - public void addActionListener(ActionListener a){}
  - public void addItemListener(ItemListener a){}

Step 3: Java Event Handling Code

We can put the event handling code into actionPerformed(ActionEvent e) method shown as follow:

```
public  void    actionPerformed(ActionEvent       aa)
{
   if(aa.getSource()==b1)               //    b1 is Button object.
     setBackground(Color.red);         //     getSource( ) method  returns object of clicked Button object
}
```

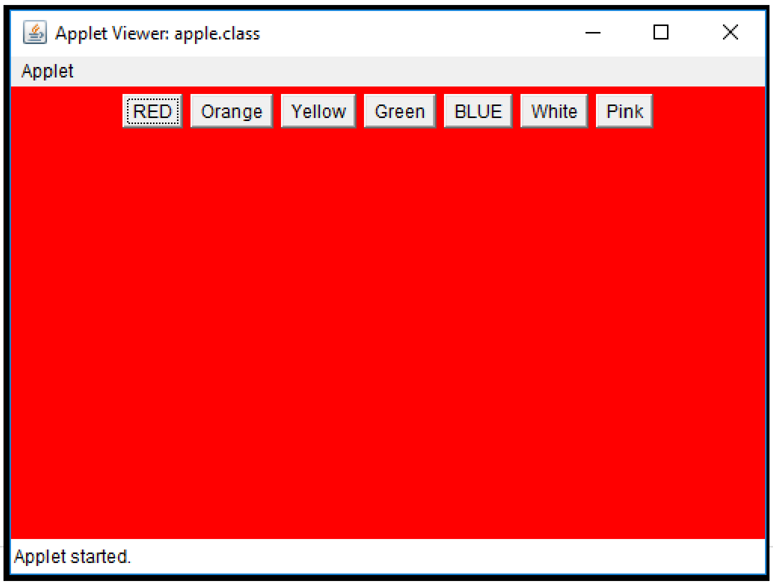
Ex-

Following applet program shows Event handling demo that changes background color of applet by clicking on appropriate button:

```
import java.applet.Applet;
import java.awt.*;
import java.awt.event.*;
public class apple extends Applet implements ActionListener
{
    Button      b1,b2,b3,b4,b5,b6,b7;
                     public void init()
    {
                              // creating objects for Buttons
                    b1=new Button("RED");
                    b2=new Button("Orange");
                    b3=new Button("Yellow");
```

```
                b4=new Button("Green");
                b5=new Button("BLUE");
                b6=new Button("White");
                b7=new Button("Pink");
                    //add buttons onto applet
                add(b1);
                add(b2);
                add(b3);
                add(b4);
                add(b5);
                add(b6);
                add(b7);
                   //register Listner for buttons
                b1.addActionListener(this);
                b2.addActionListener(this);
                b3.addActionListener(this);
                b4.addActionListener(this);
                b5.addActionListener(this);
                b6.addActionListener(this);
                b7.addActionListener(this);
    }
                public void   actionPerformed(ActionEvent    aa)
                {
                  if(aa.getSource()==b1)
                          setBackground(Color.red);
                  else if(aa.getSource()==b2)
                          setBackground(Color.orange);
                  else if(aa.getSource()==b3)
                          setBackground(Color.yellow);
                  else if(aa.getSource()==b4)
                          setBackground(Color.green);
                  else if(aa.getSource()==b5)
                          setBackground(Color.blue);
                  else if(aa.getSource()==b6)
                          setBackground(Color.white);
                  else if(aa.getSource()==b7)
                          setBackground(Color.pink);
                }
}
/*
<applet         code       = "apple.class"          width   = "500"          height   = "300" >
</applet>
*/
```

OUTPUT:

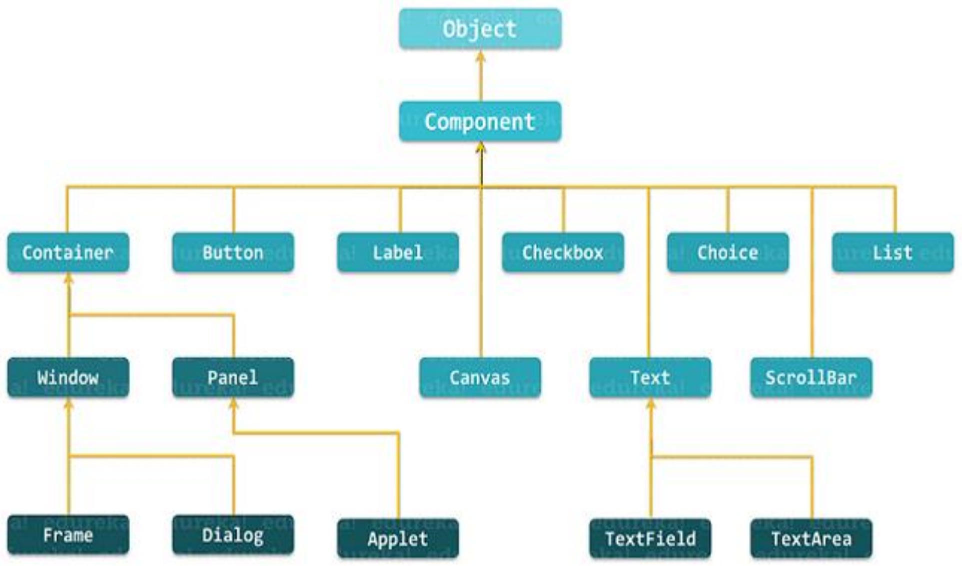


Java AWT:

- Java **AWT** (**Abstract Window Toolkit**) is an API to develop GUI or window-based applications in java.
- Java AWT components are platform-dependent i.e. components are displayed according to the view of operating system.
- AWT is heavyweight i.e. its components are using the resources of OS.
- Java AWT is an API that contains large number of classes and methods to create and manage graphical user interface (GUI) applications.
- The AWT was designed to provide a common set of tools for GUI design that could work on a variety of platforms. The tools provided by the AWT are implemented using each platform's native GUI toolkit, hence preserving the look and feel of each platform. This is an advantage of using AWT.
- But the disadvantage of such an approach is that GUI designed on one platform may look different when displayed on another platform that means AWT component are platform dependent.
- The java.awt package provides classes for AWT api such as TextField, Label, TextArea, RadioButton, CheckBox, Choice, List etc.

Java AWT Hierarchy

The hierarchy of Java AWT classes are given below.



- ➢ In the above diagram, Component is the superclass of all the GUI controls. It is an abstract class which encapsulates all the attributes of a visual component and represents an object with graphical representation. A component class instance is basically responsible for the look and feel of the current interface.

Container

- The Container is a component in AWT that can contain another components like buttons, textfields, labels etc.
- The classes that extends Container class are known as container such as Frame, Dialog and Panel.

Window

- The window is the container that have no borders and menu bars.
- We must use frame, dialog or another window for creating a window.

Panel

- The Panel is the container that doesn't contain title bar and menu bars. It can have other components like button, textfield etc.

Frame

- The Frame is the container that contain title bar and can have menu bars. It can have other components like button, textfield etc.

Useful Methods of Component class

Method	Description
public void add(Component c)	inserts a component on the Window,frame,Applet etc.
public void setSize(int width,int height)	sets the size (width and height) of the component.
public void setLayout(LayoutManager m)	defines the layout manager for the component.
public void setVisible(boolean status)	changes the visibility of the component, by default false.

Creating Frame:

- To create simple awt example, we need a frame. A frame is basic component in AWT.
- The frame has to be created before any other component because all other componets (button, textfield etc) can be displayed in frame.
- There are two ways to create a frame in AWT.
  1) By extending Frame class (inheritance)
  2) By creating the object of Frame class (association)

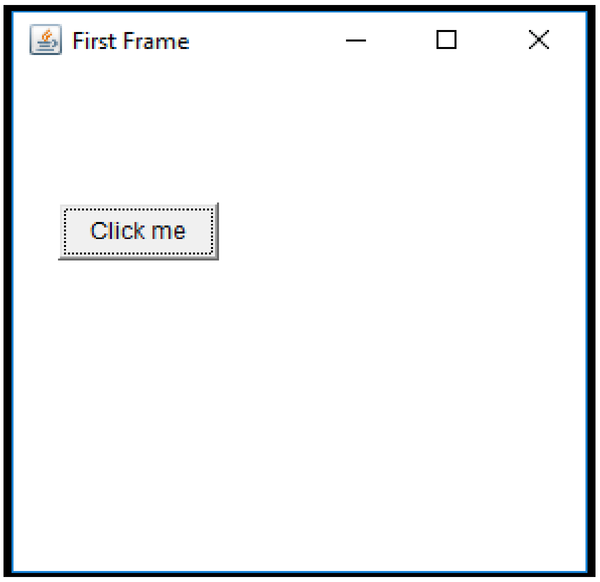
Let's see these ways in details-

1) Creating Frame by extending Frame class (Using inheritance)

Let's see a simple example of AWT where we are inheriting Frame class.
Here, we are showing Button component on the Frame.

```
import   java.awt.*;
class   FrameDemo1   extends   Frame
{
public  static  void    main(String args[])
{
   FrameDemo1    f=new FrameDemo1();
   f.setTitle("First Frame");    //set frame title
   f.setSize(300,300);       //frame size 300 width and 300 height
   f.setLayout(null);       //no layout manager
   f.setVisible(true);      //now frame will be visible, by default not visible
   Button b=new Button("Click me");
   b.setBounds(30,100,80,30);   // setting button position
   f.add(b);     //adding button on frame
}
}
```

OUTPUT:



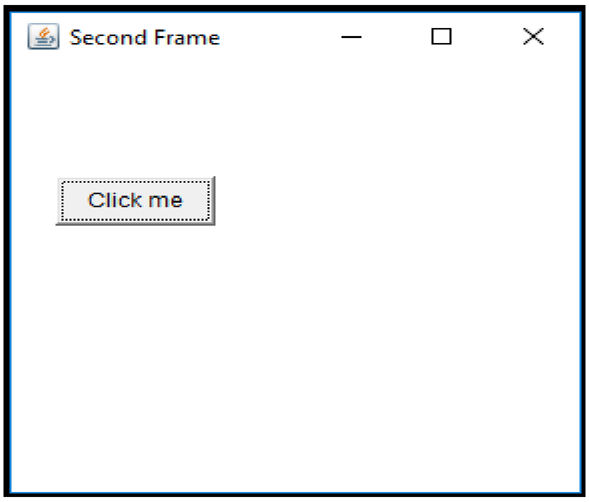
➢ In above example **setBounds (int x, int y, int width, int height)** method is used in to set the position of the awt button.

2) Creating Frame by object of Frame class (Association):

Let's see a simple example of AWT where we are creating object i.e. instance of Frame class.
Here, we are showing Button component on the Frame.

```
import      java.awt.*;
class     FrameDemo2
{
public   static   void    main(String args[])
{
   Frame    f=new     Frame();   //create frame
   f.setTitle("Second Frame ");   //set frame title
   f.setSize(300,300);       //frame size 300 width and 300 height
   f.setLayout(null);       //no layout manager
   f.setVisible(true);      //now frame will be visible, by default not visible
   Button    b=new Button("Click me");
   b.setBounds(30,100,80,30);   // setting button position
   f.add(b);     //adding button into frame
}
}
```

OUTPUT:



Closing the Frame:

- When we run application of Frame then it does not close by clicking on close button.
- If we have to close Frame application, then we have to **addWindowListener** containing **windowClosing ( ) method** that has code **System.exit(0) it exit from entire awt application.**
- **OR.** If we have to close current running Frame, then we have to **addWindowListener** containing **windowClosing ( ) method** that uses Frame class method **dispose ( ) it closes only current Frame.**

Following program shows frame closing demo.

```
import   java.awt.*;
import   java.awt.event.*;
class  FrameClose  extends  Frame
{
public  static  void  main(String args[])
{
   FrameClose   f=new FrameDemo2();
   f.setTitle("CLOSE DEMO");  //set frame title
   f.setSize(300,300);        //frame size 300 width and 300 height
   f.setLayout(null);       //no layout manager
   f.setVisible(true);      //now frame will be visible, by default not visible
   Button b=new Button("Click me");
   b.setBounds(30,100,80,30);   // setting button position
   f.add(b);     //adding button into frame
   f.addWindowListener(new   WindowAdapter( )
   {
                  public  void  windowClosing(WindowEvent    e)
                  {
                    System.exit(0);
                  }
   });
}
}
```

Java awt Controls:

- Java AWT controls are the controls that are used to design graphical user interfaces for java applications.
- To make an effective GUI, Java provides **java.awt** package that supports various AWT controls like Label, Button, CheckBox, CheckBox Group, List, Text Field, Text Area, Choice, Canvas, Image, Scrollbar, Dialog, File Dialog, etc that creates or draw various components on web and GUI based application.
- To design and manipulate Java GUI application, java provides different awt controls which are discussed bellow **(Note that all these are built in classes of java.awt package)** –

1) Label

- A label is a GUI control which can be used to display static text. Label can be created using the Label class.
- Label class has constructors which are listed below:
  Label()
  Label(String str)
  Label(String str, int how)
- The parameter *how* specifies the text alignment. And its valid values are Label.LEFT, Label.CENTER or Label.RIGHT

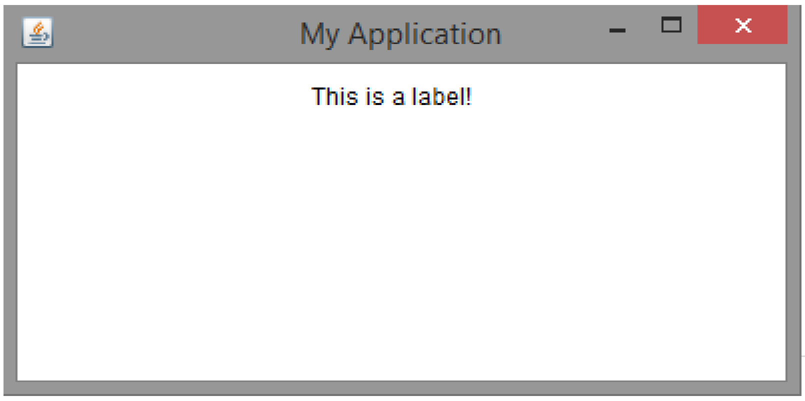
Methods available in the Label class are as follows:

- void setText(String str) – To set or assign text to the label.
- String getText() – To retrieve the text of a label.
- void setAlignment(int how) – To set the alignment of text in a label.
- int getAlignment() – To get the alignment of text in a label.

- Following program shows use of Label class:

```
import    java.awt.*;
import    java.awt.event.*;
public class   MyFrame
{
              Label    myLabel=new   Label(“This is a label!”);
              MyFrame()
              {
                Frame      f=new   Frame();
                f.setSize(400, 200);
                f.setTitle("My Application");
                f.setLayout(new FlowLayout());
                f.setVisible(true);
                myLabel.setBounds(150,20,180,25);
                f.add(myLabel);
                f.addWindowListener(new WindowAdapter()
                {
                        public void windowClosing(WindowEvent we)
                        {
                                System.exit(0);
                        }
                });
              }
              public static void main(String[] args)
              {
                  MyFrame    mf = new MyFrame();
              }
}
```

OUTPUT:



2) Button

- A Button or push button is the frequently used GUI control.
- A push button or a button can be created by using the Button class.
- Button class has constructors which are given below:
  - Button()
  - Button(String str)

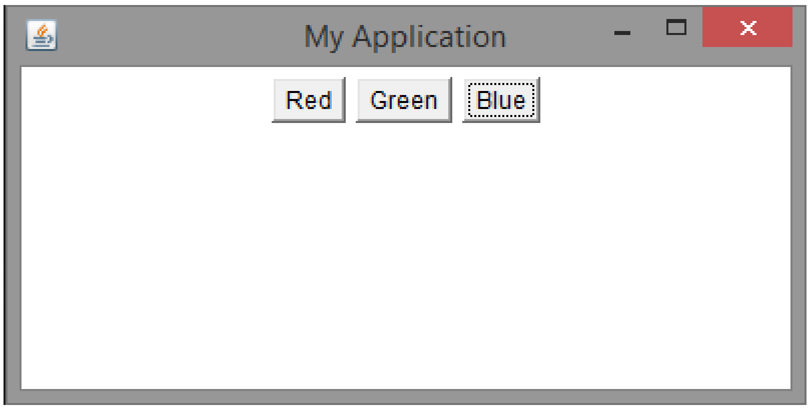
Methods available in the Button class are as follows:

- ➢ void setLabel(String str) – To set or assign the text to be displayed on the button.
- ➢ String getLabel() – To retrieve the text on the button.

- When a button is clicked, it generates an ActionEvent which can be handled using the ActionListener interface and the event handling method is actionPerformed(). If there are multiple buttons we can get the label of the button which was clicked by using the method getActionCommand().
- Following program shows use of Button class:

```
import   java.awt.*;
import   java.awt.event.*;
public class   MyFrame
{
              Button   b1  = new   Button("Red");
              Button   b2  = new   Button("Green");
              Button   b3  = new   Button("Blue");
              MyFrame()
              {
                Frame     f=new    Frame();
                f.setSize(400, 200);
                f.setTitle("My Application");
                f.setLayout(null);
                f.setVisible(true);
                 b1.setBounds(100,30,60,30);
                 b2.setBounds(180,30,60,30);
                 b3.setBounds(260,30,60,30);
                f.add(b1);
                f.add(b2);
                f.add(b3);
                f.addWindowListener(new WindowAdapter()
                {
                        public void windowClosing(WindowEvent we)
                        {
                                System.exit(0);
                        }
                });
              }
              public static void main(String[] args)
              {
                        MyFrame    mf = new    MyFrame();
              }
}
```

OUTPUT:



3) Checkbox

- A checkbox control can be created using the Checkbox class.
- Checkbox class has following constructors:
  - Checkbox( )
  - Checkbox(String str)
  - Checkbox(String str, boolean on)
  - Checkbox(String str, boolean on, CheckboxGroup cbGroup)
  - Checkbox(String str, CheckboxGroup cbGroup, boolean on)

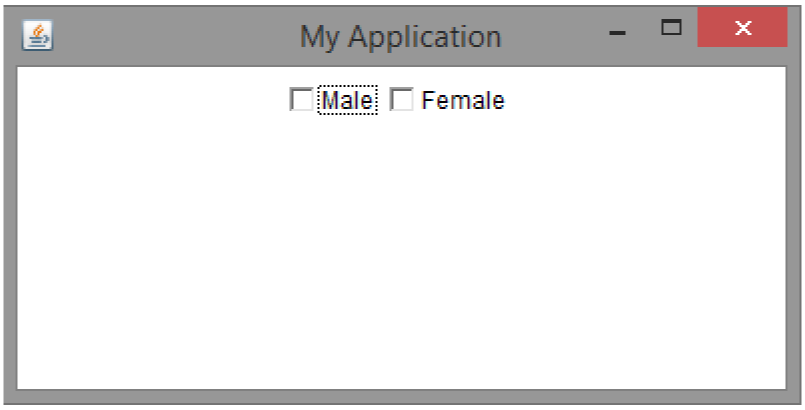
Methods available in the Checkbox class:

- ➢ boolean   getState() – To retrieve the state of a checkbox.
- ➢ void   setState(boolean on)– To set the state of a checkbox.
- ➢ String getLabel() – To retrieve the text of a checkbox.
- ➢ void setLabel(String str) – To set the text of a checkbox.

- A checkbox when selected or deselected, generates an ItemEvent which can be handled using the ItemListener interface and the corresponding event handling method is itemStateChanged().
- Following program shows use of Checkbox class:

```
import   java.awt.*;
import   java.awt.event.*;
public  class  MyFrame
{
            Checkbox   c1 = new   Checkbox("Male");
            Checkbox   c2 = new   Checkbox("Female");
            MyFrame()
            {
              Frame    f=new    Frame();
              f.setSize(400, 200);
              f.setTitle("My Application");
              f.setLayout(null);
              f.setVisible(true);
              c1.setBounds(100,40,80,30);
              c2.setBounds(200,40,80,30);
              f.add(c1);
              f.add(c2);
              f.addWindowListener(new WindowAdapter()
              {
                      public void windowClosing(WindowEvent we)
                      {
                              System.exit(0);
                      }
              });
            }
            public static void main(String[] args)
            {
                MyFrame mf = new MyFrame();
            }
}
```

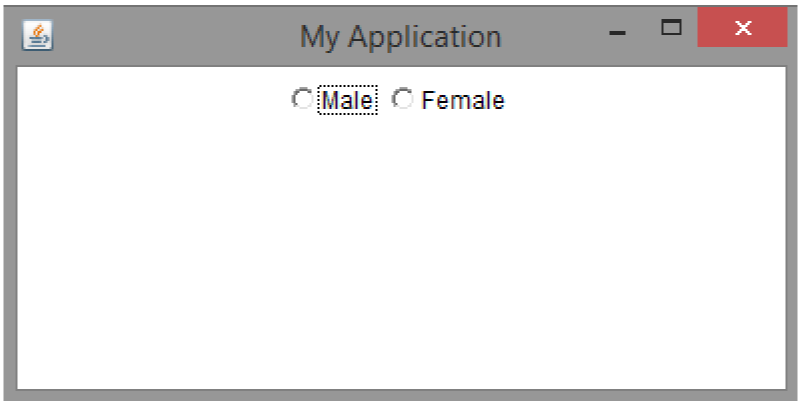
Output:



Note: In AWT, there is no separate class for creating radio buttons. The difference between a checkbox and radio button is, a user can select one or more checkboxes. Whereas, a user can select only one radio button in a group. **Radio buttons can be create by using Checkbox class and CheckboxGroup class** as shown in the below code:

```
import   java.awt.*;
import   java.awt.event.*;
public  class  MyFrame
{
                 CheckboxGroup   cbg= new   CheckboxGroup();
                 Checkbox    c1 = new    Checkbox("Male", cbg, false);
                 Checkbox    c2 = new    Checkbox("Female", cbg, false);

               MyFrame()
               {
                  Frame     f=new    Frame();
                 f.setSize(400, 200);
                 f.setTitle("My Application");
                 f.setLayout(null);
                 f.setVisible(true);
                 c1.setBounds(100,40,80,30);
                 c2.setBounds(200,40,80,30);
                 f.add(c1);
                 f.add(c2);
                 f.addWindowListener(new WindowAdapter()
                 {
                          public void windowClosing(WindowEvent we)
                          {
                                   System.exit(0);
                          }
                 });
               }
               public static void main(String[] args)
               {
                   MyFrame    mf = new    MyFrame();
               }
}
```

Output:



4) Choice class (Dropdown list)

- A drop down box or a combo box contains a list of items (strings).
- When a user clicks on a drop down box, it pops up a list of items from which user can select a single item.
- A drop down list can be created using the **Choice class**.
- There is only one constructor in the choice class using which we can create an empty list.

Methods available in Choice class:

- void   add(String name) – To add an item to the drop down list.
- String   getSelectedItem() – To retrieve the item selected by the user.
- int   getSelectedIndex() – To retrieve the index of the item selected by the user.
- int   getItemCount() – To retrieve the number of items in the drop down list.
- void select(int   index) – To select an item based on the given index.

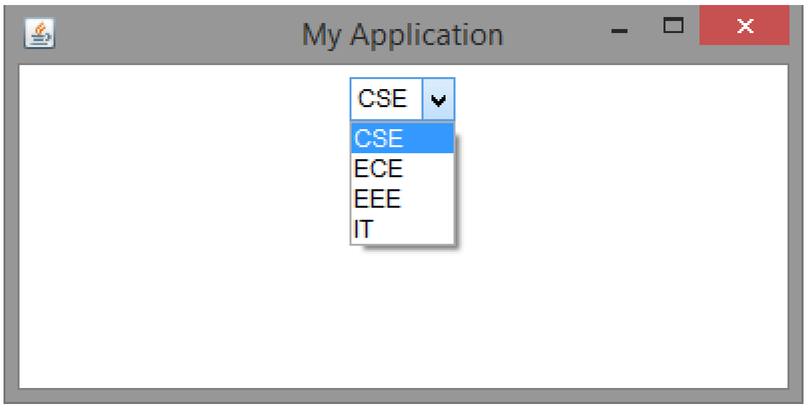
- ➢ void select(String name) – To select an item based on the given item name.
- ➢ void getItem(int index) – To retrieve an item at the given index.

- Whenever an user selects an item from the drop down box, an ItemEvent is generated. It can be handled using the ItemListener interface and the event handling method is itemStateChanged().

Following code demonstrates working with drop down boxes:

```
import   java.awt.*;
import   java.awt.event.*;
public   class   MyFrame
{
                Choice   myList = new    Choice();
                MyFrame()
                {
                  Frame     f=new    Frame();
                  f.setSize(400, 200);
                  f.setTitle("My Application");
                  f.setLayout(null);
                  f.setVisible(true);
                  myList.setBounds(100,50,120,30);
                  myList.add("CSE");
                  myList.add("ECE");
                  myList.add("EEE");
                  myList.add("IT");
                  f.add(myList);
                  f.addWindowListener(new WindowAdapter()
                  {
                           public void windowClosing(WindowEvent we)
                           {
                                    System.exit(0);
                           }
                  });
                }
                public static void main(String[] args)
                {
                  MyFrame   mf = new    MyFrame();
                }
}
```

Output:



5) List Box

- A List box contains a list of items among which the user can select one or more items.
- More than one items in the list box are visible to the user.
- A list box can be created using the **List class.**
- List class has following constructors:

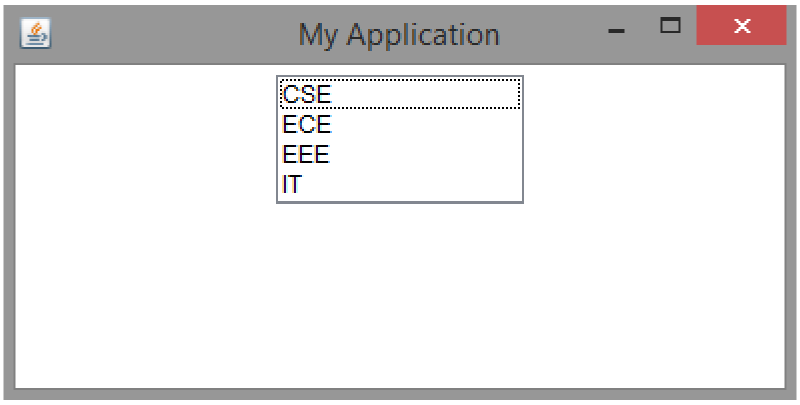
  List()
  List(int numRows)
  List(int numRows, boolean multipleSelect)

- In the above constructors, numRows specifies the number of items to be visible to the user and multipleSelect specifies whether the user can select multiple items or not.

- When a list item is double clicked, ActionEvent is generated. It can be handled with ActionListener and the event handling method is actionPerformed(). We can get the name of the item using getActionCommand() method.
- When a list item is selected or deselected, ItemEvent is generated. It can be handled with ItemListener and the event handling method is itemStateChanged(). We can use getItemSelectable() method to obtain a reference to the object that raised this event.
- **Methods available in the List class:**
  - void add(String name) – To add an item to the list box.
  - void add(String name, int index) – To add an item at the specified index in the list box.
  - String getSelectedItem() – To get the item name which is selected by the user.
  - int getSelectedIndex() – To get the item index which is selected by the user.
  - String[] getSelectedItems() – To retrieve the selected item names by the user.
  - int[] getSelectedIndexes() – To retrieve the selected item indexes by the user.
  - int getItemCount() – To retrieve the number of items in the list box.
  - void select(int index) – To select an item based on the given index.
- Following code demonstrates working with list boxes:

```
import  java.awt.*;
import  java.awt.event.*;
public class MyFrame extends Frame
{
                List   myList=new    List();
                MyFrame()
                {
                  Frame     f=new    Frame();
                  f.setSize(400, 200);
                  f.setTitle("My Application");
                  f.setLayout(null);
                  f.setVisible(true);
                  myList.setBounds(150,50,100,90);
                  myList = new List();
                  myList.add("CSE");
                  myList.add("ECE");
                  myList.add("EEE");
                  myList.add("IT");
                  f.add(myList);
                  f.addWindowListener(new WindowAdapter()
                  {
                          public void windowClosing(WindowEvent we)
                          {
                                  System.exit(0);
                          }
                  });
                 }
                public static void main(String[] args)
                {
                   MyFrame   mf = new    MyFrame();
                }
}
```

Output:



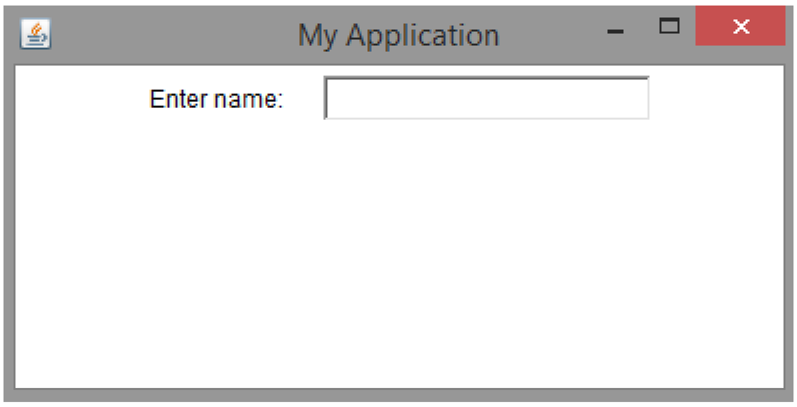
6) Text Fields

- A text field or text box is a single line text entry control which allows the user to enter a single line of text.
- A text field can be created using the **TextField class**
- TextFields has following constructors:
  TextField()
  TextField(int numChars)
  TextField(String str)
  TextField(String str, int numChars)
- In the above constructors numChars specifies the width of the text field, and str specifies the initial text in the text field.
- When an user hits 'Enter' key on the keyboard in a text field, an ActionEvent is generated. It can be handled using ActionListener and the event handling method is actionPerformed().
- Whenever an user modifies the text in the text field, a TextEvent is generated which can be handled using TextListener and the event handling method is textValueChanged().
- **Methods available in TextField class:**
  - String getText() – Retrieves the text in the text field.
  - void setText(String str) – Assigns or sets text in the text field.
  - String getSelectedText() – Retrieves the selected text in the text field.
  - void select(int startindex, int endindex) – To select the text in text field from startindex to endindex – 1.
  - boolean isEditable() – To check whether the text field is editable or not.
  - void setEditable(boolean canEdit) – To make a text field editable or non-editable.
  - void setEchoChar(char ch) – To set the echo character of a text field. This is generally used for password fields.
  - boolean echoCharIsSet() – To check whether the echo character for the text field is set or not.
  - char getEchoChar() – To retrieve the current echo character.

Following code demonstrates working with text fields:

```
import   java.awt.*;
import   java.awt.event.*;
public  class   MyFrame
{
              Label   myLabel= new Label("Enter name: ");
              TextField  tf= new TextField();
              MyFrame()
              {
                Frame    f=new  Frame();
                f.setSize(400, 200);
                f.setTitle("My Application");
                f.setLayout(null);
                f.setVisible(true);
                myLabel.setBounds(80,50,110,30);
                tf.setBounds(200,50,150,30);
                f.add(myLabel);
                f.add(tf);
                f.addWindowListener(new WindowAdapter()
                {
                        public  void  windowClosing(WindowEvent we)
                        {
                                System.exit(0);
                        }
                });
                }
              public static void main(String[] args)
              {
                MyFrame   mf = new  MyFrame();
              }
}
```

Output:



7) Text Areas

- A text area is a multi-line text entry control in which user can enter multiple lines of text.
- A text area can be created using the **TextArea** class.
- It has following constructors:

  TextArea()
  TextArea(int numLines, int numChars)
  TextArea(String str)
  TextArea(String str, int numLines, int numChars)
  TextArea(String str, int numLines, int numChars, int sBars)
- In the above constructors, numLines specifies the height of the text area, numChars specifies the width of the text area, str specifies the initial text in the text area and sBars specifies the scroll bars. Valid values of sBars can be any one of the following:

SCROLLBARS_BOTH
SCROLLBARS_NONE
SCROLLBARS_HORIZONTAL_ONLY
SCROLLBARS_VERTICAL_ONLY

Methods available in the TextArea class:

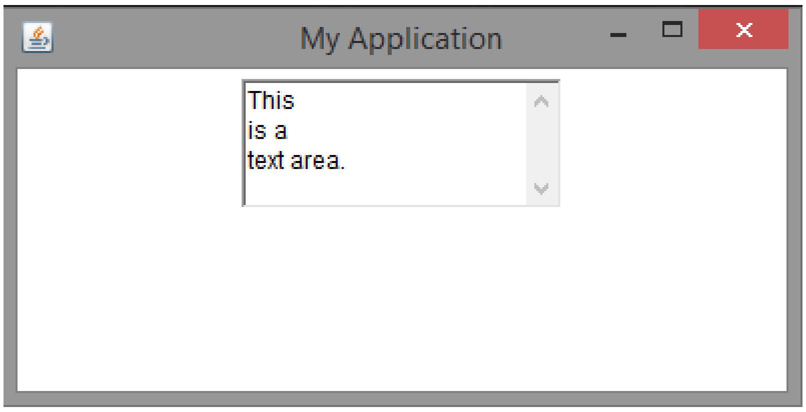
- String getText() – To retrieve the text in the text area.
- void setText(String str) – To assign or set the text in a text area.
- String getSelectedText() – To retrieve the selected text in a text area.
- void select(int s tartindex, int endindex) – To select the text in text field from startindex to endindex – 1.
- boolean isEditable() – To check whether the text field is editable or not.
- void setEditable(boolean canEdit) – To make a text field editable or non-editable.
- void append(String str) – To append the given string to the text in the text area.
- void insert(String str, int index) – To insert the given string at the specified index.
- void replaceRange(String str, int startIndex, int endIndex) – To replace the text from startIndex to endIndex – 1 with the given string.

Following code demonstrates use of Textarea class:

```
import  java.awt.*;
import  java.awt.event.*;
public  class  MyFrame
{
                TextArea    ta= new TextArea();
                MyFrame()
                {
                  Frame    f=new    Frame();
                  f.setSize(400, 200);
                  f.setTitle("My Application");
                  f.setLayout(null);
                  f.setVisible(true);
                  ta.setBounds(150,70,120,100);
                  f.add(ta);
                  f.addWindowListener(new WindowAdapter()
                  {
                          public void windowClosing(WindowEvent we)
                          {
                                  System.exit(0);
```

```
                        }
                });
            }
            public  static  void   main(String[] args)
            {
              MyFrame    mf = new    MyFrame();
            }
}
```

OUTPUT:



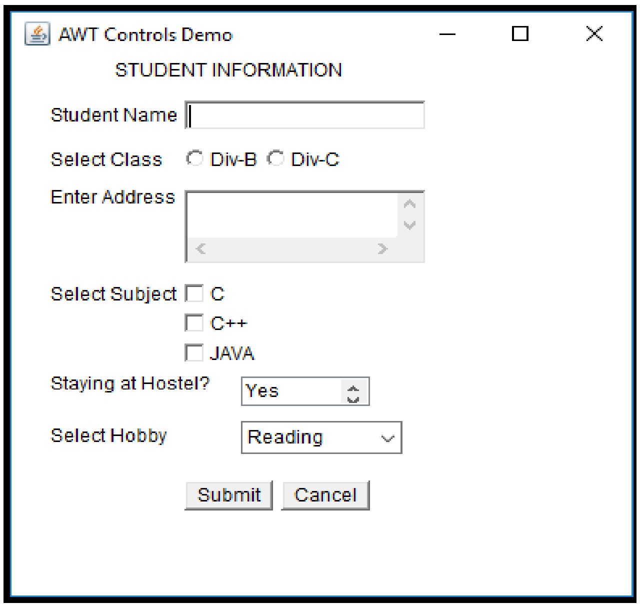
```
This
is a
text area.
```

Following program demonstrate use of different awt controls.

```
import    java.awt.event.*;
import    java.awt.*;
class   college
{
    TextField       tx1=new TextField();
    TextArea       ta=new    TextArea();

    Label         l1=new   Label("STUDENT INFORMATION");
    Label         l2=new  Label("Student Name");
    Label         l3=new  Label("Select Class");
    Label         l4=new  Label("Enter Address");
    Label         l5=new  Label("Select Subjects");
    Label         l6=new  Label("Staying at Hostel?");
    Label         l7=new  Label("Select Hobby");

    CheckboxGroup cbg=new  CheckboxGroup();   /*in java.awt, when we add check boxes on checkbox group
                                                    then they became radiobox automatically*/
    Checkbox       chk1=new   Checkbox("Div-B",cbg,false);
    Checkbox       chk2=new   Checkbox("Div-C",cbg,false);

   Checkbox   chkSub1=new   Checkbox("C");
   Checkbox   chkSub2=new   Checkbox("C++");
   Checkbox   chkSub3=new   Checkbox("JAVA");

   List          lb=new   List(4,false);
   Choice   hoby=new   Choice();

   Button   btn1=new   Button("Submit");
   Button   btn2=new   Button("Cancel");
   college( )
  {
    Frame   f=new Frame("AWT Controls Demo");
              //set size, layout and visibility of frame
    f.setSize(400,400);
    f.setLayout(null);
    f.setVisible(true);
              //add items into List control
   lb.add("Yes");
```

```
     lb.add("No");
               //add items into Choice control
     hoby.add("Reading");
     hoby.add("Swimming");
     hoby.add("Watching TV");
     hoby.add("Internet Surfing");
                // set position of controls onto frame
      l1.setBounds(70,30,150,20);
      tx1.setBounds(115,60,150,20);
      ta.setBounds(115,120,150,50);
      l2.setBounds(30,60,80,20);
      l3.setBounds(30,90,80,20);
      l4.setBounds(30,115,80,20);
      l5.setBounds(30,180,80,20);
      l6.setBounds(30,240,120,20);
      l7.setBounds(30,275,120,20);
      chk1.setBounds(115,90,50,20);
      chk2.setBounds(165,90,50,20);
      chkSub1.setBounds(115,180,80,20);
      chkSub2.setBounds(115,200,80,20);
      chkSub3.setBounds(115,220,80,20);
      lb.setBounds(150,245,80,20);
      hoby.setBounds(150,275,100,20);
      btn1.setBounds(115,315,55,20);
      btn2.setBounds(175,315,55,20);
                 //add controls onto frame
      f.add(tx1);
      f.add(l1);
      f.add(l2);
      f.add(l3);
      f.add(chk1);
      f.add(chk2);
      f.add(ta);
      f.add(l4);
      f.add(l5);
      f.add(chkSub1);
      f.add(chkSub2);
      f.add(chkSub3);
      f.add(l6);
      f.add(lb);
      f.add(hoby);
      f.add(l7);
      f.add(btn1);
      f.add(btn2);
                 // to exit from running frame
      f.addWindowListener(new WindowAdapter()
      {
           public void windowClosing(WindowEvent  e)
            {
                 System.exit(1);
            }
       });
}
public static void main(String  arg[])
{
   college     p=new college();
}
}
```

OUTPUT:



Java GUI Event Handling: Event and Listener (Java Event Handling):

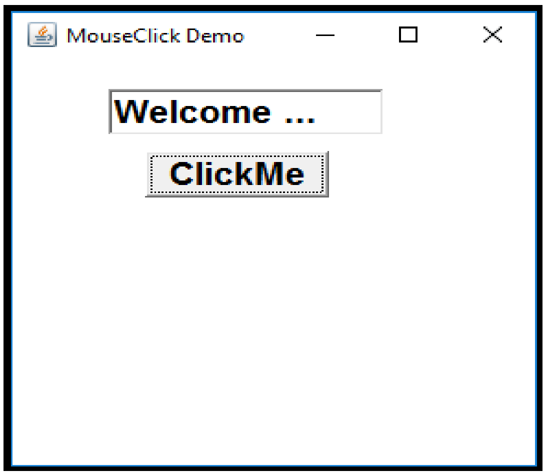
- Any program that uses GUI (graphical user interface) such as Java application written for windows, is event driven. Event describes the change in state of any object. For Example: Pressing a button, entering a character in Textbox, Clicking or dragging a mouse, etc.
- **Note: For GUI event handling notes Refere above mentioned notes in Applet part on (Page No- 8 and Page No- 9)**

Program-1) Following program shows GUI event handling that displayed message in TextField by clicking on Button.

```
import java.awt.*;
import java.awt.event.*;

public  class  MouseClkDemo  implements  ActionListener
{
   TextField   txt=new  TextField();
   Button      btn=new   Button("ClickMe");
   public MouseClkDemo()
   {
                 Frame  fr=new  Frame("MouseClick Demo");
                   //set size,layout and visibilty for frame
                 fr.setSize(300,300);
                 fr.setLayout(null);
                 fr.setVisible(true);
                   //create font
                 Font  fnt=new Font("Arial",Font.BOLD,20);
                   //set display possition for controls
                 txt.setBounds(60,50,150,30);
                 btn.setBounds(80,90,100,30);
                    //apply created font to controls
                 btn.setFont(fnt);
                 txt.setFont(fnt);
                   //adding controls onto frame
                 fr.add(txt);
                 fr.add(btn);
                    //register listner to button
                 btn.addActionListener(this);
                             // to exit from frame
                 fr.addWindowListener(new WindowAdapter()
                 {
                   public void windowClosing(WindowEvent e)
                   {
```

```
                              System.exit(1);
                      }
                  });
   }
   // code to execute after event occure
                  public  void   actionPerformed(ActionEvent     e)
                  {
                      txt.setText("Welcome ...");
                  }
 public static void main(String []arg)
 {
                   MouseClkDemo   m=new MouseClkDemo();
 }
}
```

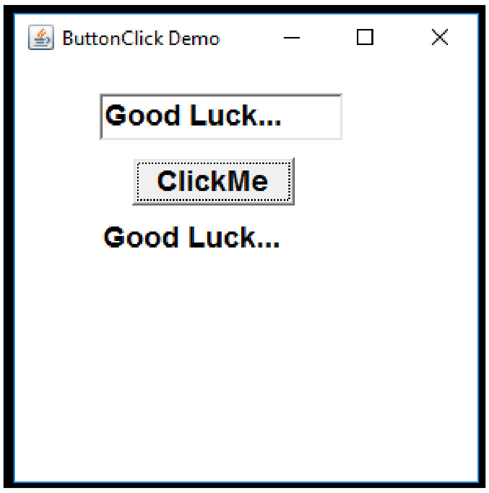
OUTPUT:



Program-2) Following program shows GUI event handling that displayed entred text of TextField to Label by clicking on Button.

```
import  java.awt.*;
import  java.awt.event.*;
public class  LabelAction  implements  ActionListener
{
   TextField  txt=new TextField();
   Button  btn=new Button("ClickMe");
   Label   lb=new Label();
   public LabelAction()
   {
                  Frame  fr=new Frame("ButtonClick Demo");
                    //set size,layout and visibilty for frame
                  fr.setSize(300,300);
                  fr.setLayout(null);
                  fr.setVisible(true);
                    //create font
                  Font  fnt=new Font("Arial",Font.BOLD,18);
                    //set display possition for controls
                  txt.setBounds(60,50,150,30);
                  btn.setBounds(80,90,100,30);
                  lb.setBounds(60,125,120,30);
                     //apply created font to controls
                  btn.setFont(fnt);
                  txt.setFont(fnt);
                  lb.setFont(fnt);
                    //adding controls onto frame
                  fr.add(txt);
                  fr.add(btn);
                  fr.add(lb);
                     //register listner to button
                  btn.addActionListener(this);
                          // to exit from frame
                  fr.addWindowListener(new WindowAdapter()
                  {
                    public void windowClosing(WindowEvent e)
                     {
                               System.exit(1);
                     }
                  });
   }
   // code to execute after event occure
                  public void   actionPerformed(ActionEvent   e)
                  {
                     String   str=txt.getText();   //retrieve text of TextField
                    lb.setText(str);     // set text str to Label
                  }
public  static  void  main(String []arg)
{
                   LabelAction   m=new LabelAction();
}
}
```

OUTPUT:



Good Luck...

Good Luck...

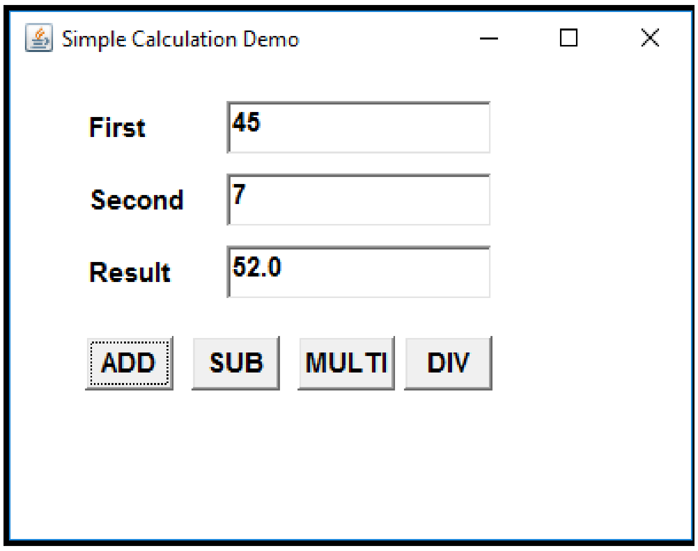
Program-3) Program shows GUI application that finds addition, subtraction, multiplication and division.

```
import   java.awt.*;
import   java.awt.event.*;
public class Calculation implements ActionListener
{
    TextField    txt1=new TextField();
    TextField    txt2=new TextField();
    TextField   txt3=new TextField();
    Button   btn1=new Button("ADD");
    Button   btn2=new Button("SUB");
    Button   btn3=new Button("MULTI");
    Button   btn4=new Button("DIV");
    Label    lb1=new Label("First");
    Label    lb2=new Label("Second");
    Label    lb3=new Label("Result");
    public Calculation()
    {
                    Frame  fr=new Frame("Simple Calculation Demo");
                       //set size,layout and visibilty for frame
                    fr.setSize(400,300);
                    fr.setLayout(null);
                    fr.setVisible(true);
                       //create font
```

```
                Font  fnt=new Font("Arial",Font.BOLD,15);
                  //set display possition for controls
                txt1.setBounds(130,50,150,30);
                txt2.setBounds(130,90,150,30);
                txt3.setBounds(130,130,150,30);
                lb1.setBounds(50,50,150,30);
                lb2.setBounds(50,90,150,30);
                lb3.setBounds(50,130,150,30);

                btn1.setBounds(50,180,50,30);
                btn2.setBounds(110,180,50,30);
                btn3.setBounds(170,180,55,30);
                btn4.setBounds(230,180,50,30);


                   //apply created font to controls
                btn1.setFont(fnt);
                btn2.setFont(fnt);
                btn3.setFont(fnt);
                btn4.setFont(fnt);
                txt1.setFont(fnt);
                txt2.setFont(fnt);
                txt3.setFont(fnt);
                lb1.setFont(fnt);
                lb2.setFont(fnt);
                lb3.setFont(fnt);

                  //adding controls onto frame
                fr.add(txt1);
                fr.add(txt2);
                fr.add(txt3);
                fr.add(btn1);
                fr.add(btn2);
                fr.add(btn3);
                fr.add(btn4);
                fr.add(lb1);
                fr.add(lb2);
                fr.add(lb3);
                   //register listner to buttons
                btn1.addActionListener(this);
                btn2.addActionListener(this);
                btn3.addActionListener(this);
                btn4.addActionListener(this);

                   // to exit from frame
                fr.addWindowListener(new WindowAdapter()
                {
                  public void windowClosing(WindowEvent e)
                  {
                          System.exit(1);
                  }
                });
  }
  // code to execute after event occure
                public void    actionPerformed(ActionEvent    e)
                {
                  String  s1=txt1.getText();
                  String  s2=txt2.getText();
                  double d1=Double.parseDouble(s1);
```

```
                        double d2=Double.parseDouble(s2);
                        double d3;
                        if(e.getSource()==btn1)
                        {
                                d3=d1+d2;
                                String res=Double.toString(d3);
                                txt3.setText(res);
                        }
                        else if(e.getSource()==btn2)
                        {
                                d3=d1-d2;
                                String res=Double.toString(d3);
                                txt3.setText(res);
                        }
                        else if(e.getSource()==btn3)
                        {
                                d3=d1*d2;
                                String res=Double.toString(d3);
                                txt3.setText(res);
                        }
                        else if(e.getSource()==btn4)
                        {
                                d3=d1/d2;
                                String res=Double.toString(d3);
                                txt3.setText(res);
                        }
                      }
  public static void main(String []arg)
  {
                      Calculation   m=new Calculation();
  }
}
```

OUTPUT:

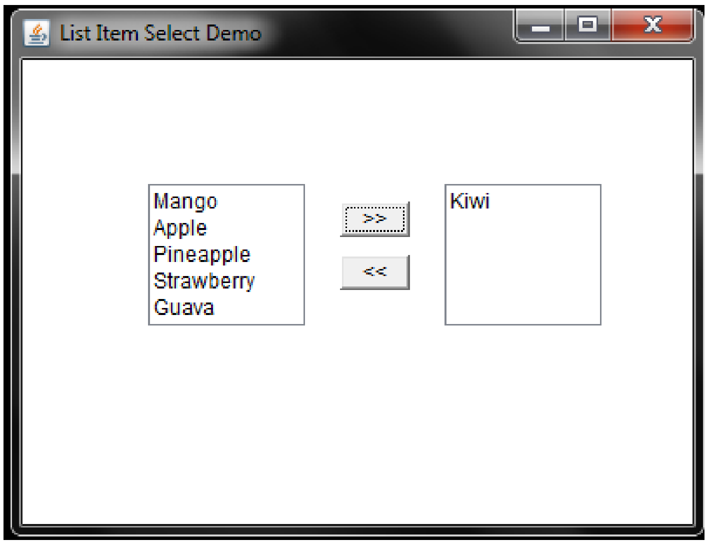


Program-4) Program shows GUI application that display selected item of First List into Second List and vice versa by clicking on respective buttons.

```
import   java.awt.*;
import    java.awt.event.*;
public  class  ListItemSelect  implements  ActionListener
{

   List  lst_one=new   List();
   List  lst_two=new   List();
   Button  b1=new    Button(">>") ;
```

```
  Button   b2=new   Button("<<");
  ListItemSelect()
  {
      Frame   f=new Frame("List Item Select Demo");
      f.setSize(400,300);
      f.setVisible(true);
      f.setLayout(null);
      lst_one.setBounds(80,100,90,80);
      lst_two.setBounds(250,100,90,80);
      b1.setBounds(190,110,40,20);
      b2.setBounds(190,140,40,20);
        //add items in First List Box
      lst_one.add("Mango");
      lst_one.add("Apple");
      lst_one.add("Kiwi");
      lst_one.add("Pineapple");
      lst_one.add("Strawberry");
      lst_one.add("Guava");
          //add controls on to Frame
      f.add(lst_one);
      f.add(lst_two);
      f.add(b1);
      f.add(b2);
        // register listener to Buttons
      b1.addActionListener(this) ;
      b2.addActionListener(this);
        //to close frame
      f.addWindowListener(new WindowAdapter()
      {
                public void windowClosing(WindowEvent   e)
                {
                  System.exit(1);
                }
      });
  }

  public  void  actionPerformed(ActionEvent   e)
  {
                   if(e.getSource()==b1)
                   {
                     String  s=lst_one.getSelectedItem();
                     lst_two.add(s);
                     lst_one.remove(s);
                   }
                   else if(e.getSource()==b2)
                   {
                     String   s=lst_two.getSelectedItem();
                     lst_one.add(s);
                     lst_two.remove(s);
                   }
  }
 public static void main(String []ar)
  {
                  ListItemSelect     p=new   ListItemSelect();
  }
}
```

OUTPUT:



Java Swing

- Java Swing tutorial is a part of Java Foundation Classes (JFC) that is used to create window-based applications. It is built on the top of AWT (Abstract Windowing Toolkit) API and entirely written in java.
- We know that Java AWT is platform dependent but Java Swing provides platform-independentcy and it has lightweight components.
- The java swing provides decent look for all component as compared to java awt components that's why now a days Java swing is widly used to develop GUI application.
- The **javax.swing** package provides classes for java swing programming such as JButton, JTextField, JTextArea, JRadioButton, JCheckbox, JMenu, JColorChooser etc.

Difference between AWT and Swing

There are many differences between java awt and swing that are given below.

No.	Java AWT	Java Swing
1)	AWT components are **platform-dependent**.	Java swing components are **platform-independent**.
2)	AWT components are **heavyweight**.	Swing components are **lightweight**.
3)	AWT **doesn't support pluggable look and feel**.	Swing **supports pluggable look and feel**.
4)	AWT provides **less components** than Swing.	Swing provides **more powerful components** such as tables, lists, scrollpanes, colorchooser, tabbedpane etc.
5)	AWT **doesn't follows MVC** (Model View Controller) where model represents data, view represents presentation and controller acts as an interface between model and view.	Swing **follows MVC**.

Main Features of Swing Toolkit:

- ➢ Platform Independent
- ➢ Customizable
- ➢ Extensible
- ➢ Configurable
- ➢ Lightweight
- ➢ Rich Controls
- ➢ Pluggable Look and Feel

What is JFC?

- The Java Foundation Classes (JFC) are a set of GUI components which simplify the development of desktop applications.
- Features of JFC
  - ➢ Swing GUI components.
  - ➢ Look and Feel support.

Swing and JFC

- JFC is an abbreviation for Java Foundation classes, which encompass a group of features for building Graphical User Interfaces (GUI) and adding rich graphical functionalities and interactivity to Java applications. Java Swing is a part of Java Foundation Classes (JFC).
- Commonly used Methods of swing Component class are-
- The methods of Component class are widely used in java swing that are given below.

Method	Description
public void add(Component c)	add a component on another component.
public void setSize(int width,int height)	sets size of the component.
public void setLayout(LayoutManager m)	sets the layout manager for the component.
public void setVisible(boolean b)	sets the visibility of the component. It is by default false.

Creating JFrame:

- To create simple swing example, we need a Jframe. A Jframe is basic component in AWT.
- The Jframe has to be created before any other component because all other componets (JButton, JTextfield etc) can be displayed on Jframe.
- There are two ways to create a Jframe in Swing-
  3) By extending JFrame class (inheritance)
  4) By creating the object of JFrame class (association)

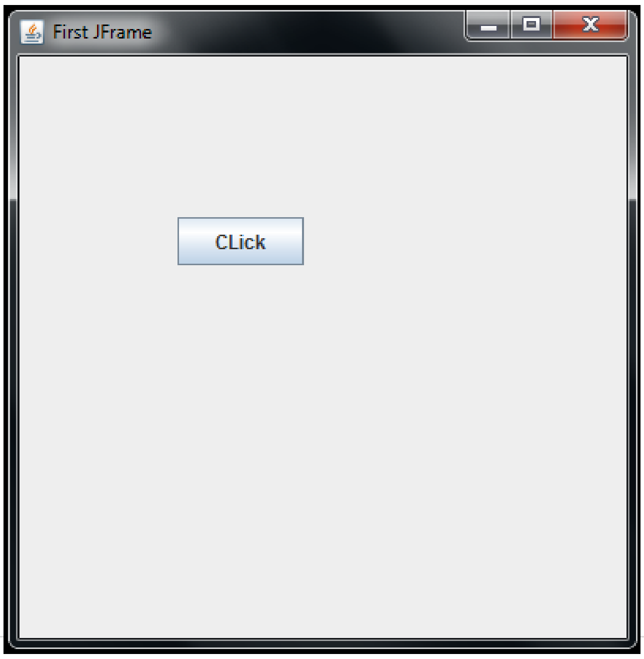
Let’s see these ways in details-

1) Creating Frame by extending JFrame class (Using inheritance)

Let's see a simple example of Swing where we are inheriting JFrame class.
Here, we are showing JButton component on the JFrame.

```
import     javax.swing.*;
public  class  NewFrameDemo  extends  JFrame
{

  public static void main(String []ar)
  {
     NewFrameDemo    f=new     NewFrameDemo();
     f.setSize(400,400);
     f.setVisible(true);
     f.setLayout(null);
     f.setTitle("First JFrame");
     JButton   btn=new JButton("CLick");
     btn.setBounds(100,100,80,30);
     f.add(btn);
  }
}
```

OUTPUT:



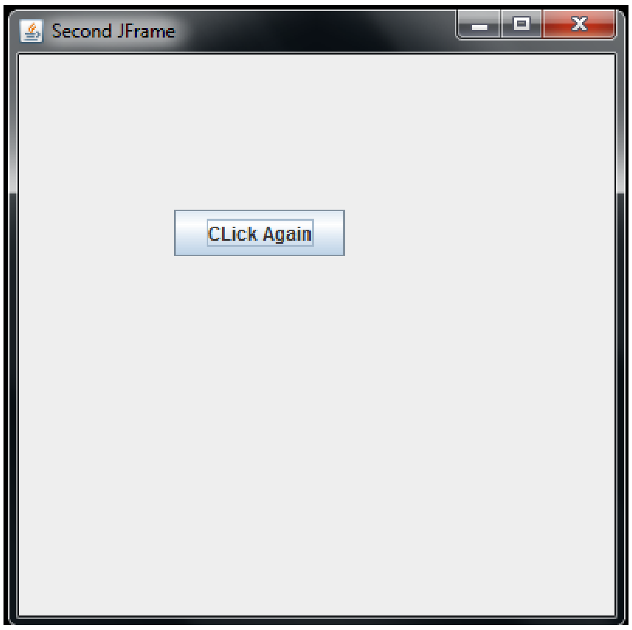
- In above example **setBounds (int x, int y, int width, int height)** method is used in to set the position of the swing JButton.

2) Creating JFrame by object of JFrame class (Association):

Let's see a simple example of swing where we are creating object i.e. instance of JFrame class.
Here, we are showing JButton component on the JFrame.

```
import    javax.swing.*;
public   class   SecondJFrame
{
   public  static  void  main(String []ar)
   {
        JFrame    f=new    JFrame();
        f.setSize(400,400);
        f.setVisible(true);
        f.setLayout(null);
        f.setTitle("Second JFrame");
        JButton   btn=new   JButton("CLick Again");
        btn.setBounds(100,100,110,30);
        f.add(btn);
   }
}
```

OUTPUT:



Closing the JFrame:

- When we run application of JFrame then it closes by clicking on close button but still this JFrame application remains active in RAM.
- If we have to close JFrame application totally (from RAM also), then we have to use JFrame method **setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE); that closes entire swing application.**
- **OR** If we have to close current running JFrame only (from RAM also), then we have to use JFrame method **setDefaultCloseOperation(JFrame. DISPOSE_ON_CLOSE); that closes current JFrame only.**
- Following program shows JFrame closing demo.

```
import      java.awt.*;
import     javax.swing.*;
class     MySecFrame
{
public   static   void    main(String args[])
{  JFrame    f=new     JFrame("MY Second JFrame");    //create Jframe with title
    f.setSize(300,300);        //frame size 300 width and 300 height
    f.setLayout(null);        //no layout manager
    f.setVisible(true);      //now Jframe will be visible, by default not visible
    JButton b=new JButton("Click...");
    b.setBounds(70,70,80,30);   // setting button position
    f.add(b);     //adding Jbutton on Jframe
```

```
   f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
}
}
```

Java Swing Controls:

- Java swing controls are the controls that are used to design graphical user interfaces for java applications.
- To make an effective GUI, Java provides **javax.swing** package that supports various swing controls like JLabel, JButton, JCheckBox, JTextField, JList, JRadioButton, JComboBox etc that creates or draw various components on web and GUI based application.
- To design and manipulate Java GUI application, java provides different swing controls which are discussed bellow **(Note that all these are built in classes of javax.swing package)** –

1) JLabel

- The JLabel class is used to display a label i.e., static text onto GUI application.
- A JLabel class has following constructors:
  JLabel(Icon icon)
  JLabel(String str)
  JLabel(String str, Icon icon, int align)
- In the above constructors icon is used to specify an image to be displayed as a label. Icon is a predefined interface which is implemented by the ImageIcon class. str is used to specify the text to be displayed in the label and align is used to specify the alignment of the text.
- **Methods of JLabel class are as follows:**
  - void setText(String str) – To set the text of the label
  - String getText() – To get the text of the label
  - void setIcon(Icon icon) – To display an image in the label
  - Icon getIcon() – To retrieve the image displayed in the labelA label is a GUI contro

2) JButton

- The JButton class is used to display a push button.
- A JButton has following constructors:
  JButton(Icon   icon)
  JButton(String str)
  JButton(String str, Icon icon)
- In the above constructors, icon specifies the image to be displayed as button and str specifies the text to be displayed on the button.
- JButton objects raise ActionEvent when the button is clicked. It can be handled by implementing ActionListener interface and the event handling method is actionPerformed().
- **The JButton class implements the abstract class AbstractButton which provides the following methods:**
  - void setDisableIcon(Icon di) – To set the icon to be displayed when button is disabled
  - void setPressedIcon(Icon pi) – To set the icon to be displayed when button is pressed
  - void setSelectedIcon(Icon si) – To set the icon to be displayed when button is selected
  - void setRolloverIcon(Icon ri) – To set the icon to be displayed when the button is rolled over
  - void setText(String str) – To set the text to be displayed on the button
  - String getText() – To retrieve the text displayed on the button

3) JCheckBox

- Check boxex can be created using the JCheckBox class which inherits the JToggleButton class.
- JCheckBox has following constructor:
  JCheckBox() : creates a new checkbox with no text or icon
  JCheckBox(Icon i) : creates a new checkbox with the icon specified
  JCheckBox(Icon icon, boolean s) : creates a new checkbox with the icon specified and the boolean value specifies whether it is selected or not.
  JCheckBox(String t) :creates a new checkbox with the string specified
  JCheckBox(String text, boolean selected) :creates a new checkbox with the string specified and the boolean value specifies whether it is selected or not.
  JCheckBox(String text, Icon icon) :creates a new checkbox with the string and the icon specified.

JCheckBox(String text, Icon icon, boolean selected): creates a new checkbox with the string and the icon specified and the boolean value specifies whether it is selected or not.

Methods of JCheckBox-

- setIcon(Icon i) : sets the icon of the checkbox to the given icon
- setText(String s) :sets the text of the checkbox to the given text
- setSelected(boolean b) : sets the checkbox to selected if boolean value passed is true or vice versa
- getIcon() : returns the image of the checkbox
- getText() : returns the text of the checkbox
- updateUI() : resets the UI property with a value from the current look and feel.
- getUI() : returns the look and feel object that renders this component.
- paramString() : returns a string representation of this JCheckBox.
- getUIClassID() : returns the name of the Look and feel class that renders this component.
- getAccessibleContext() : gets the AccessibleContext associated with this JCheckBox.
- isBorderPaintedFlat() : gets the value of the borderPaintedFlat property.
- setBorderPaintedFlat(boolean b) : sets the borderPaintedFlat property,

4) JTextField

- A text field is a GUI control which allows the user to enter a single line of text.
- A text field can be created using the class JTextField which inherits the class JTextComponent.
- A JTextField class has following constructors:
  JTextField(int cols)
  JTextField(String str)
  JTextField(String str, int cols)
- In the above constructors cols specifies the size of the text field and str specifies the default string to be displayed in the text field.

Methods of JTextField:

- setColumns(int n) :set the number of columns of the text field.
- setFont(Font f) : set the font of text displayed in text field.
- addActionListener(ActionListener l) : set an ActionListener to the text field.
- int getColumns() :get the number of columns in the textfield.

5) JList

- JList is part of Java Swing package.
- JList is a component that displays a set of Objects and allows the user to select one or more items.
- JList inherits JComponent class. JList is a easy way to display an array of Vectors.
- Constructors of JList are :
  JList(): creates an empty blank list
  JList(E [ ] l) : creates an new list with the elements of the array.
  JList(ListModel d): creates a new list with the specified List Model
  JList(Vector l) : creates a new list with the elements of the vector

Methods of JList are:

getSelectedIndex()	returns the index of selected item of the list
getSelectedValue()	returns the selected value of the element of the list
setSelectedIndex(int i)	sets the selected index of the list to i
setSelectionBackground(Color c)	sets the background Color of the list
setSelectionForeground(Color c)	Changes the foreground color of the list
setListData(E [ ] l)	Changes the elements of the list to the elements of l .
setVisibleRowCount(int v)	Changes the visibleRowCount property
setSelectedValue(Object a, boolean s)	selects the specified object from the list.
setSelectedIndices(int[] i)	changes the selection to be the set of indices specified by the given array.
setListData(Vector l)	constructs a read-only ListModel from a Vector specified.
setLayoutOrientation(int l)	defines the orientation of the list
setFixedCellWidth(int w)	Changes the cell width of list to the value passed as parameter.
setFixedCellHeight(int h)	Changes the cell height of the list to the value passed as parameter.
isSelectedIndex(int i)	returns true if the specified index is selected, else false.
indexToLocation(int i)	returns the origin of the specified item in the list's coordinate system.
getToolTipText(MouseEvent e)	returns the tooltip text to be used for the given event.

- getSelectedValuesList() returns a list of all the selected items.
- getSelectedIndices() returns an array of all of the selected indices, in increasing order
- getMinSelectionIndex() returns the smallest selected cell index, or -1 if the selection is empty.
- getMaxSelectionIndex() returns the largest selected cell index, or -1 if the selection is empty.
- getListSelectionListeners() returns the listeners of list
- getLastVisibleIndex() returns the largest list index that is currently visible.
- getDragEnabled() returns whether or not automatic drag handling is enable
- addListSelectionListener(ListSelectionListener l) adds a listSelectionlistener to the list

6) JComboBox

- JComboBox is a part of Java Swing package.
- JComboBox inherits JComponent class.
- JComboBox shows a popup menu that shows a list and the user can select a option from that specified list. JComboBox can be editable or read- only depending on the choice of the programmer.
- Constructors of the JComboBox are:
  JComboBox() : creates a new empty JComboBox .
  JComboBox(ComboBoxModel M) : creates a new JComboBox with items from specified ComboBoxModel
  JComboBox(E [ ] i) : creates a new JComboBox with items from specified array.
  JComboBox(Vector items) : creates a new JComboBox with items from the specified vector

Methods of JComboBox are:

- addItem(E item) : adds the item to the JComboBox
- addItemListener(ItemListener l) : adds a ItemListener to JComboBox
- getItemAt(int i) : returns the item at index i
- getItemCount(): returns the number of items from the list
- getSelectedItem() : returns the item which is selected
- removeItemAt(int i) : removes the element at index i
- setEditable(boolean b) : the boolean b determines whether the combo box is editable or not .If true is passed then the combo box is editable or vice versa.
- setSelectedIndex(int i): selects the element of JComboBox at index i.
- showPopup() :causes the combo box to display its popup window.
- setUI(ComboBoxUI ui): sets the L&F object that renders this component.
- setSelectedItem(Object a): sets the selected item in the combo box display area to the object in the argument.
- setSelectedIndex(int a): selects the item at index anIndex.
- setPopupVisible(boolean v): sets the visibility of the popup.
- setModel(ComboBoxModel a) : sets the data model that the JComboBox uses to obtain the list of items.
- setMaximumRowCount(int count): sets the maximum number of rows the JComboBox displays.
- setEnabled(boolean b): enables the combo box so that items can be selected.
- removeItem(Object anObject) : removes an item from the item list.
- removeAllItems(): removes all items from the item list.
- removeActionListener(ActionListener l): removes an ActionListener.
- isPopupVisible() : determines the visibility of the popup.
- addPopupMenuListener(PopupMenuListener l) : adds a PopupMenu listener which will listen to notification messages from the popup portion of the combo box.
- getActionCommand() : returns the action command that is included in the event sent to action listeners.
- getEditor(): returns the editor used to paint and edit the selected item in the JComboBox field.
- getItemCount() : returns the number of items in the list.
- getItemListeners(): returns an array of all the ItemListeners added to this JComboBox with addItemListener().
- createDefaultKeySelectionManager() : returns an instance of the default key-selection manager.
- fireItemStateChanged(ItemEvent e) : notifies all listeners that have registered interest for notification on this event type.
- firePopupMenuCanceled() : notifies PopupMenuListeners that the popup portion of the combo box has been canceled.
- firePopupMenuWillBecomeInvisible() : notifies PopupMenuListeners that the popup portion of the combo box has become invisible.
- firePopupMenuWillBecomeVisible() : notifies PopupMenuListeners that the popup portion of the combo box will become visible.

- ➢ setEditor(ComboBoxEditor a): sets the editor used to paint and edit the selected item in the JComboBox field.
- ➢ setActionCommand(String a) : sets the action command that should be included in the event sent to actionListeners.
- ➢ getUI() : returns the look and feel object that renders this component.
- ➢ paramString() : returns a string representation of this JComboBox.
- ➢ getUIClassID() : returns the name of the Look and feel class that renders this component.
- ➢ getAccessibleContext() : gets the AccessibleContext associated with this JComboBox

7) JRadioButton

- We use the JRadioButton class to create a radio button.
- Radio button is use to select one option from multiple options.
- Basically it is used in filling forms, online objective papers and quiz.
- We add radio buttons in a ButtonGroup so that we can select only one radio button at a time. We use "ButtonGroup" class to create a ButtonGroup and add radio button in a group.

Methods of JRadioButton:

- ➢JRadioButton() : Creates a unselected RadioButton with no text.
- ➢JButton(String s) : Creates a JButton with a specific text.
- ➢JLabel(String s) : Creates a JLabel with a specific text.

Steps to Group the radio buttons together:

- ➢ ButtonGroup() : Use to create a group, in which we can add JRadioButton. We can select only one JRadioButton in a ButtonGroup.
- ➢ Create a ButtonGroup instance by using "ButtonGroup()" Method.
- ➢ ButtonGroup   G = new   ButtonGroup();
- ➢ Now add buttons in a Group "G", with the help of "add()" Method.
- ➢ Example:
  G.add(Button1);
  G.add(Button2);

8) JTextArea

- JTextArea is a part of java Swing package.It represents a multi line area that displays text. It is used to edit the text.
- JTextArea inherits JComponent class.
- The text in JTextArea can be set to different available fonts and can be appended to new text.
- A text area can be customized to the need of user.
- **Constructors of JTextArea are:**
  JTextArea() : constructs a new blank text area .
  JTextArea(String s) : constructs a new text area with a given initial text.
  JTextArea(int row, int column) : constructs a new text area with a given number of rows and columns.
  JTextArea(String s, int row, int column) : constructs a new text area with a given number of rows and columns and a given initial text.

Methods of JTextArea:

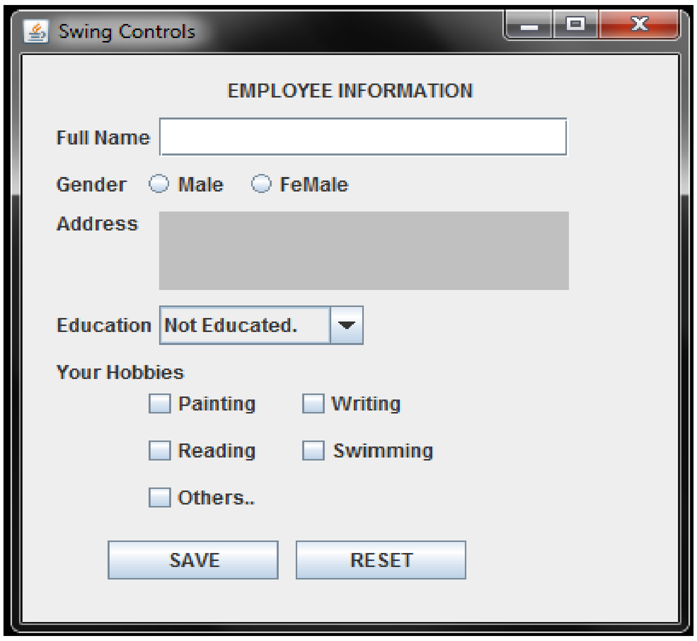
- ➢ append(String s) : appends the given string to the text of the text area.
- ➢ getLineCount() : get number of lines in the text of text area.
- ➢ setFont(Font f) : sets the font of text area to the given font.
- ➢ setColumns(int c) : sets the number of columns of the text area to given integer.
- ➢ setRows(int r) : sets the number of rows of the text area to given integer.
- ➢ getColumns() : get the number of columns of text area.
- ➢ getRows() : get the number of rows of text area.

Following program shows java swing application uses different java swing controls

```
import    javax.swing.*;
import    java.awt.*;
public    class    SwingControls
{
   JLabel  headLbl=new JLabel("EMPLOYEE INFORMATION") ;
   JLabel  nameLbl=new JLabel("Full Name") ;
   JLabel  genLbl=new JLabel("Gender") ;
   JLabel  addLbl=new JLabel("Address") ;
   JLabel  eduLbl=new JLabel("Education");
   JLabel  hobbyLbl=new JLabel("Your Hobbies");

   JRadioButton genRadM=new JRadioButton("Male");
   JRadioButton genRadF=new JRadioButton("FeMale");
   ButtonGroup G = new ButtonGroup();

   JTextField  nameTxt=new JTextField();
   JTextArea   addTxtA=new JTextArea();

   JCheckBox  chk1=new JCheckBox("Painting");
   JCheckBox  chk2=new JCheckBox("Reading");
   JCheckBox  chk3=new JCheckBox("Writing");
   JCheckBox  chk4=new JCheckBox("Swimming");
   JCheckBox  chk5=new JCheckBox("Others..");

   JButton  subBtn=new JButton("SAVE");
   JButton  resBtn=new JButton("RESET");

   String s1[] = {"Not Educated.", "Std 1 To 9", "SSC", "HSC", "Graduate","Post Graduate"};
   JComboBox   eduCom=new JComboBox(s1);

   public SwingControls()
   {
               JFrame   f=new  JFrame("Swing Controls");
               f.setSize(400,400);
               f.setLayout(null);
               f.setVisible(true);
                 //Add RadioButton to ButtonGroup
               G.add(genRadM);
               G.add(genRadF);
                   //Add background color to lightgray for JTextArea
               addTxtA.setBackground(Color.lightGray);
                  //set position of controls
               headLbl.setBounds(120,10,200,25);
               nameLbl.setBounds(20,40,200,25);
               genLbl.setBounds(20,70,200,25);
               addLbl.setBounds(20,95,200,25);
               eduLbl.setBounds(20,160,200,25);
               hobbyLbl.setBounds(20,190,200,25);
               nameTxt.setBounds(80,40,240,25);
               genRadM.setBounds(70,70,60,25);
               genRadF.setBounds(130,70,70,25);
               addTxtA.setBounds(80,100,240,50);
               eduCom.setBounds(80,160,120,25);
               chk1.setBounds(70,210,90,25);
               chk2.setBounds(70,240,90,25);
               chk3.setBounds(160,210,90,25);
               chk4.setBounds(160,240,90,25);
               chk5.setBounds(70,270,90,25);
               subBtn.setBounds(50,310,100,25);
               resBtn.setBounds(160,310,100,25);
                 //add controls on JFrame
               f.add(headLbl);
               f.add(nameLbl);
               f.add(genLbl);
               f.add(addLbl);
               f.add(addLbl);
               f.add(eduLbl);
               f.add(hobbyLbl);
```

```
                f.add(nameTxt);
                f.add(genRadM);
                f.add(genRadF);
                f.add(addTxtA);
                f.add(eduCom);
                f.add(chk1);
                f.add(chk2);
                f.add(chk3);
                f.add(chk4);
                f.add(chk5);
                f.add(subBtn);
                f.add(resBtn);
                    //to close JFrame
                f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }
    public static void main(String []arg)
    {
                SwingControls    obj=new SwingControls();
    }
}
```

OUTPUT:



Note:

- ➢ Event handling in Java Swing is similary done as Java Applet or Java AWT. For that we have to import package **java.awt.event.*;**
- ➢ **Reffer Event Handling Notes on Page No. 8 and Page No. 9**

Java Servlet Overview:

- Servlet technology which is used to create a web application using java language that resides at server side and generates a dynamic web page.
- Java Servlets are java programs that run on a Web server and act as a middle layer between a requests coming from a Web browser or other HTTP client and databases or applications on the HTTP server.
- There are many interfaces and classes in the Servlet API such as Servlet, GenericServlet, HttpServlet, ServletRequest, ServletResponse, etc.
- Using Servlets, we can collect input from users through web page forms, present records from a database or another source, and create web pages dynamically.

What is servlet?

Servlet can be described in many ways, depending on the context some of these as follow-

- ➢ Servlet is a technology which is used to create a server side web application.
- ➢ Servlet is an API that provides many interfaces and classes to create web applications.

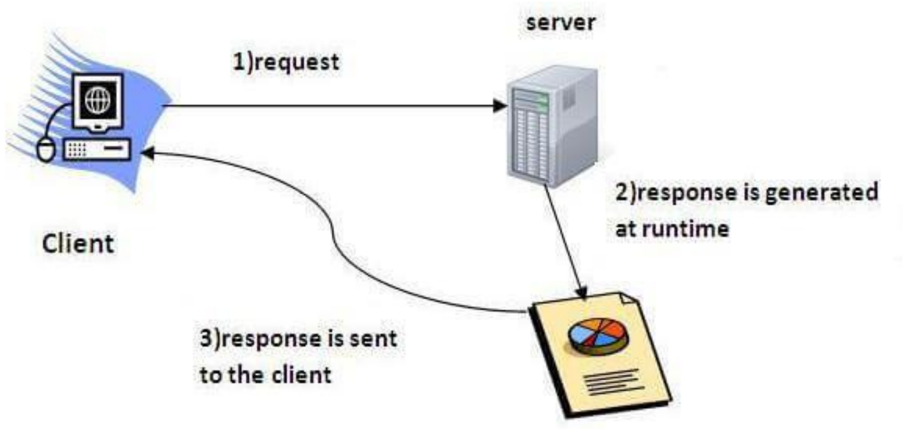
- Servlet is an interface that must be implemented for creating any Servlet.
- Servlet is a class that extends the capabilities of the servers and responds to the incoming requests. It can respond to any requests that come from client.
- Servlet is a web component that is deployed on the server to create a dynamic web page.

Characteristics or Properties of Servlet:

- Servlet technology is robust and scalable because of java language.
- Servlets execute within the address space of a Web server. It is not necessary to create a separate process to handle each client request.
- Servlets are platform-independent because they are written in Java.
- Servlets are trusted: Java security manager on the server enforces a set of restrictions to protect the resources on a server machine.
- The full functionality of the Java class libraries is available to a servlet. Therefore servlet can communicate with applets, databases, or other software via the sockets and RMI mechanisms.
- Servlet performance is significantly better as compared to other server applications.

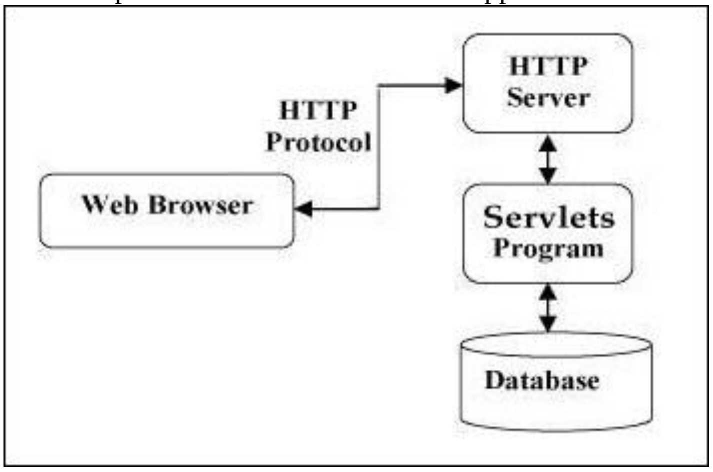
Working of Servlet:

Following diagram shows working of servlet.


server
1)request
2)response is generated at runtime
Client
3)response is sent to the client

Servlets Architecture:

The following diagram shows the position of Servlets in a Web Application.



Servlets Tasks:

Servlets perform the following major tasks –

- Servlet can read the **explicit data** sent by the clients (browsers). This includes an HTML form on a Web page or it could also come from an applet or a custom HTTP client program.
- Servlet can read the **implicit HTTP request data** sent by the clients (browsers). This includes cookies, media types and compression schemes the browser understands.
- Servlet process the data and generate the results as per client request. This process may require access to a database, executing an RMI or CORBA call, invoking a Web service, or computing the response directly.
- Servlet also capable to send the explicit data (i.e., the document) to the clients (browsers). This document can be sent in a variety of formats, including text (HTML or XML), binary (GIF images), Excel, etc.

Theory Assignment No: 06

1) What is Applet? Explain life cycle of Applet.
2) What are the steps to perform event handling?
3) What is Java AWT? Explain AWT hierarchy.
4) How to create Frame in Java awt?
5) Explain following Java awt controls-
   1) Label
   2) Button
   3) Checkbox
   4) Choice (Dropdown Boxes)
   5) List
   6) TextField
   7) TextArea
6) What is Java Swing?
7) Write the difference between Java awt and Javax Swing.
8) Explain following Java Swing controls-
   1) JLabel
   2) JButton
   3) JCheckbox
   4) JChoiceBox
   5) JComboBox
   6) JTextField
   7) JTextArea
   8) JList
9) What is Java Servlet? Listout characteristics or properties of servlet.
10) List out different tasks performed by servlet.

Practical Assignment No: 08

1) Write a program to implement Applet containing different Graphics objects.
2) Write a program to implement Applet showing smiley face.
3) Write a program to implement Applet showing event handling that changes background color of applet
4) Write a program to implement Applet that display an image.
5) Write a program to implement Applet that moves image (Animation using Appet)
6) Write a program that add different AWT controls onto Applet.
7) Write a program that demonstrate use of different awt controls on Frame.
8) Write an awt GUI application that finds addition, subtraction, multiplication and division.
9) Write a program that shows awt GUI application that display selected item of First List into Second List and vice versa by clicking on respective buttons.
10) Write a program that shows java swing application with different java swing controls.
11) Write any program in java swing that shows use of multiple JFrames.