

Presentation on Java Programming (IT207G)

UNIT No. I- 2



What is String in Java?

- Generally, String is a sequence of characters. But in Java, **string** is an **object that represents a sequence of characters.**
- The `java.lang.String` class is used to create a string object.

How to create a string object?

1) String Literal

Java String literal is created by using double quotes. For Example:

String s="welcome";

2) By new keyword

String s=new String("Welcome");

Java String class methods

No.	Method	Description
1	<u>char charAt(int index)</u>	It returns char value for the particular index
2	<u>int length()</u>	It returns string length
3	<u>String substring(int beginIndex)</u>	It returns substring for given begin index.
4	<u>boolean equals(Object another)</u>	It checks the equality of string with the given object.
5	<u>String concat(String str)</u>	It concatenates the specified string.
6	<u>String replace(char old, char new)</u>	It replaces all occurrences of the specified char value.
7	<u>String toLowerCase()</u>	It returns a string in lowercase.
8	<u>String toUpperCase()</u>	It returns a string in uppercase.

Java String Class Methods

```
public class Stringoperation1
{
    public static void main(String ar[])
    {
        String s="Sachin";
        System.out.println(s.toUpperCase());//SACHIN
        System.out.println(s.toLowerCase());//sachin
        System.out.println(s);//Sachin(no change in original)
    }
}
```

Output:

SACHIN
sachin
Sachin

Java String Class Methods

```
public class Stringoperation4
{
    public static void main(String ar[])
    {
        String s="Sachin";
        System.out.println(s.charAt(0));//S
        System.out.println(s.charAt(3));//h
    }
}
```

Output:

S

h

Java String Class Methods

```
public class Stringoperation8
```

```
{
```

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

```
{
```

```
String s1="Java is a programming language. Java is a platform. Java is  
an Island.;"
```

```
String replaceString=s1.replace("Java","Kava");
```

```
System.out.println(replaceString);
```

```
}
```

```
}
```

Output:

Kava is a programming language. Kava is a platform. Kava is an Island.

What is StringBuffer in Java?

- Java StringBuffer class is used to create mutable (modifiable) String objects.
- The StringBuffer class in Java is the same as String class except it is mutable i.e. it can be changed..

Important Constructors of StringBuffer Class

Constructor	Description
StringBuffer()	It creates an empty String buffer with the initial capacity of 16.
StringBuffer(String str)	It creates a String buffer with the specified string..
StringBuffer(int capacity)	It creates an empty String buffer with the specified capacity as length.

Java StringBuffer class methods

No.	Method	Description
1	append(String s)	It is used to append the specified string with this string.
2	insert(int offset, String s)	It is used to insert the specified string with this string at the specified position.
3	replace(int startIndex, int endIndex, String str)	It is used to replace the string from specified startIndex and endIndex.
4	reverse()	is used to reverse the string.
5	capacity()	It is used to return the current capacity.
6	charAt(int index)	It is used to return the character at the specified position.
7	delete(int startIndex, int endIndex)	It is used to delete the string from specified startIndex and endIndex.
8	length()	It is used to return the length of the string

What is a mutable String?

A String that can be modified or changed is known as mutable String.

Example 1:

```
class StringBufferExample{  
    public static void main(String args[]){  
        StringBuffer sb=new StringBuffer("Hello ");  
        sb.append("Java");//now original string is changed  
        System.out.println(sb);//prints Hello Java  
    }  
}
```

Output:

Hello Java

Example 2:

StringBufferExample2.java

```
class StringBufferExample2{  
    public static void main(String args[]){  
        StringBuffer sb=new StringBuffer("Hello ");  
        sb.insert(1,"Java");//now original string is changed  
        System.out.println(sb);//prints HJavaello  
    }  
}
```

Output:

HJavaello

Example 3:

StringBufferExample3.java

```
class StringBufferExample3{
public static void main(String args[]){
StringBuffer sb=new StringBuffer("Hello");
sb.replace(1,3,"Java");
System.out.println(sb);//prints HJava
}
}
```

Output:

HJava

Difference between String and StringBuffer

No.	String	StringBuffer
1)	The String class is immutable.	The StringBuffer class is mutable.
2)	String is slow and consumes more memory when we concatenate too many strings because every time it creates new instance.	StringBuffer is fast and consumes less memory when we concatenate strings.
3)	String class overrides the equals() method of Object class. So you can compare the contents of two strings by equals() method.	StringBuffer class doesn't override the equals() method of Object class.
4)	String class is slower while performing concatenation operation.	StringBuffer class is faster while performing concatenation operation.
5)	String class uses String constant pool.	StringBuffer uses Heap memory

Java Command Line Arguments

- The java command-line argument is an argument i.e. passed at the time of running the java program.
- The arguments passed from the console can be received in the java program and it can be used as an input.
- You can pass N (1,2,3 and so on) numbers of arguments from the command prompt.
- **public static void main(String args[])**

example of command-line argument in java

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

compile by > javac CommandLine.java

run by > java CommandLine Ajay

Output: **Your first argument is: Ajay**

Example of command-line argument in java

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

compile by > javac ABC.java
run by > java ABC Ajay Love Java 7
Output: Ajay

Love
Java

Wrapper classes in Java

- The **wrapper class in Java** provides the mechanism *to convert primitive into object and object into primitive.*
- **autoboxing** and **unboxing** feature convert primitives into objects and objects into primitives automatically.
- Java is an object-oriented programming language, so we need to deal with objects many times , where we need to use the wrapper classes.

Wrapper classes in Java

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

Wrapper classes in Java

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.

```
public class WrapperExample1{
    public static void main(String args[]){
        //Converting int into Integer
        int a=20;
        Integer i=Integer.valueOf(a);//converting int into Integer explicitly
        Integer j=a;//autoboxing, now compiler will write Integer.valueOf(a) internally
        System.out.println(a+" "+i+" "+j);
    }
}
```

Wrapper classes in Java

Unboxing:

- The automatic conversion of wrapper type into its corresponding primitive type is known as unboxing.
- It is the reverse process of autoboxing.

```
public class WrapperExample2{  
    public static void main(String args[]){  
        //Converting Integer to int  
        Integer a=new Integer(3);  
  
        int i=a.intValue(); //converting Integer to int explicitly  
        int j=a; //unboxing, now compiler will write a.intValue() internally  
  
        System.out.println(a+" "+i+" "+j);  
    }  
}
```

Vector class in Java

- **Vector** is like the *dynamic array* which can grow or shrink its size. Unlike array, we can store n-number of elements in it as there is no size limit.
- It is found in the `java.util` package and implements the *List* interface, so we can use all the methods of List interface here.

Java Vector Constructors:

SN	Constructor	Description
1)	<code>vector()</code>	It constructs an empty vector with the default size as 10.
2)	<code>vector(int initialCapacity)</code>	It constructs an empty vector with the specified initial capacity and with its capacity increment equal to zero.

Vector class Methods

SN	Method	Description
1)	<u>add()</u>	It is used to append the specified element in the given vector.
2)	<u>addAll()</u>	It is used to append all of the elements in the specified collection to the end of this Vector.
3)	<u>equals()</u>	It is used to compare the specified object with the vector for equality.
4)	<u>capacity()</u>	It is used to get the current capacity of this vector.
5)	<u>elements()</u>	It returns an enumeration of the components of a vector.
6)	<u>elementAt()</u>	It is used to get the component at the specified index.
7)	<u>contains()</u>	It returns true if the vector contains the specified element.
8)	<u>containsAll()</u>	It returns true if the vector contains all of the elements in the specified
9)	<u>get()</u>	It is used to get an element at the specified position in the vector.

Vector class Methods

SN	Method	Description
10)	<u>firstElement()</u>	It is used to get the first component of the vector.
11)	<u>get()</u>	It is used to get an element at the specified position in the vector.
12)	<u>insertElementAt()</u>	It is used to insert the specified object as a component in the given vector at the specified index.
13)	<u>lastElement()</u>	It is used to get the last component of the vector.
14)	<u>remove()</u>	It is used to remove the specified element from the vector. If the vector does not contain the element, it is unchanged.
15)	<u>removeElementAt()</u>	It is used to delete the component at the specified index.
16)	<u>setElementAt()</u>	It is used to set the component at the specified index of the vector to the specified object.

Java Vector Example

```
import java.util.*;
public class VectorExample {
    public static void main(String args[]) {
        //Create a vector
        Vector vec = new Vector();
        //Adding elements using add() method of List
        vec.add("Tiger");
        vec.add("Lion");
        vec.add("Dog");
        vec.add("Elephant");
        //Adding elements using addElement() method of Vector
        vec.addElement("Rat");
        vec.addElement("Cat");
        vec.addElement("Deer");
        System.out.println("Elements are: "+vec);
    }
}
```

O/P: Elements are: [Tiger, Lion, Dog, Elephant, Rat, Cat, Deer]

Java Vector Example

```
import java.util.*;  
  
public class VectorExample1 {  
  
    public static void main(String args[]) {  
  
        //Create an empty vector with initial capacity 4  
        Vector<String> vec = new Vector<String>(4);  
  
        //Adding elements to a vector  
        vec.add("Tiger");  
        vec.add("Lion");  
        vec.add("Dog");  
        vec.add("Elephant");  
  
        //Check size and capacity  
        System.out.println("Size is: " + vec.size());  
        System.out.println("Default capacity is: " + vec.capacity());  
  
        //Display Vector elements  
        System.out.println("Vector element is: " + vec);  
        vec.addElement("Rat");  
        vec.addElement("Cat");  
        vec.addElement("Deer");  
  
        //Again check size and capacity after two insertions  
        System.out.println("Size after addition: " + vec.size());  
        System.out.println("Capacity after addition is: " + vec.capacity());  
  
        //Display Vector elements again  
        System.out.println("Elements are: " + vec);  
  
        //Checking if Tiger is present or not in this vector  
        if(vec.contains("Tiger"))  
        {  
            System.out.println("Tiger is present at the index " + vec.indexOf("Tiger"));  
        }  
        else  
        {  
            System.out.println("Tiger is not present in the list.");  
        }  
  
        //Get the first element  
        System.out.println("The first animal of the vector is = " + vec.firstElement());  
  
        //Get the last element  
        System.out.println("The last animal of the vector is = " + vec.lastElement());  
    }  
}
```

Java Vector Example

Output:

Size is: 4

Default capacity is: 4

Vector element is: [Tiger, Lion, Dog, Elephant]

Size after addition: 7

Capacity after addition is: 8

Elements are: [Tiger, Lion, Dog, Elephant, Rat, Cat, Deer]

Tiger is present at the index 0

The first animal of the vector is = Tiger

The last animal of the vector is = Deer

Presentation on Java Programming (IT207G)

UNIT No. 1



Content :

- What is java?
- Where is java used?
- Feature of java
- Java program translation
- Java virtual machine
- Java system overview
- Java program-development phase
- Advantage of java
- Disadvantage of java



What is java?

- Java is a general – purpose , object-oriented programming language developed by sun Microsoft of USA in 1991.
- Originally called Oak by james gosling,one of the inventors of the language.
- Java is a first programming language which provide the concept of writing programs that can be executed using the web.

where is java used?

- Desktop application :-acrobat reader,Media player,antiviruse etc.
- Enterprise application :- banking application,Business application.
- Mobile.
- Embedded system.
- Games.
- Robotics.

java Features :

- Compiled and interpreted
- Platform-independent and portable
- Object-oriented
- Robust and secure
- Distributed
- Familiar
- Simple and small
- High performance

java Features :

Simple

Java is very easy to learn, and its syntax is simple, clean and easy to understand. According to Sun Microsystem, Java language is a simple programming language because:

Object-oriented

Java is an object-oriented programming language. Everything in Java is an object. Object-oriented means we organize our software as a combination of different types of objects that incorporate both data and behavior.

java Features :

Platform Independent

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

Secured

Java is best known for its security. With Java, we can develop virus-free systems. Java is secured because:

- **No explicit pointer**
- **Java Programs run inside a virtual machine sandbox**

java Features :

Robust

The English mining of Robust is strong. Java is robust because:

- It uses strong memory management.
- There is a lack of pointers that avoids security problems.
- Java provides automatic garbage collection which runs on the Java Virtual Machine to get rid of objects which are not being used by a Java application anymore.
- There are exception handling and the type checking mechanism in Java. All these points make Java robust.

java Features :

Portable

Java is portable because it facilitates you to carry the Java bytecode to any platform. It doesn't require any implementation.

Multi-threaded

A thread is like a separate program, executing concurrently. We can write Java programs that deal with many tasks at once by defining multiple threads.

C++ vs Java

Comparison Index	C++	Java
Platform-independent	C++ is platform-dependent.	Java is platform-independent.
Goto	C++ supports the goto statement.	Java doesn't support the goto statement.
Multiple inheritance	C++ supports multiple inheritance.	Java doesn't support multiple inheritance through class.
Operator Overloading	C++ supports operator overloading .	Java doesn't support operator overloading.
Pointers	C++ supports pointers . You can write a pointer program in C++.	Java has restricted pointer support in java.
Call by Value and Call by reference	C++ supports both call by value and call by reference.	Java supports call by value only. There is no call by reference in java.

File name: Hello.java

```
/* Our first Java program – Hello.java */
```

```
public class Hello {
```

```
    //main()
```

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

```
        System.out.println( "hello world!" );
```

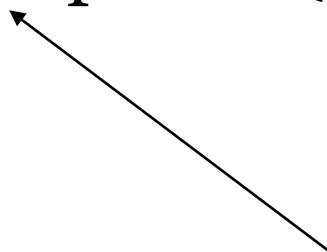
```
}
```

```
}
```

Command line
arguments



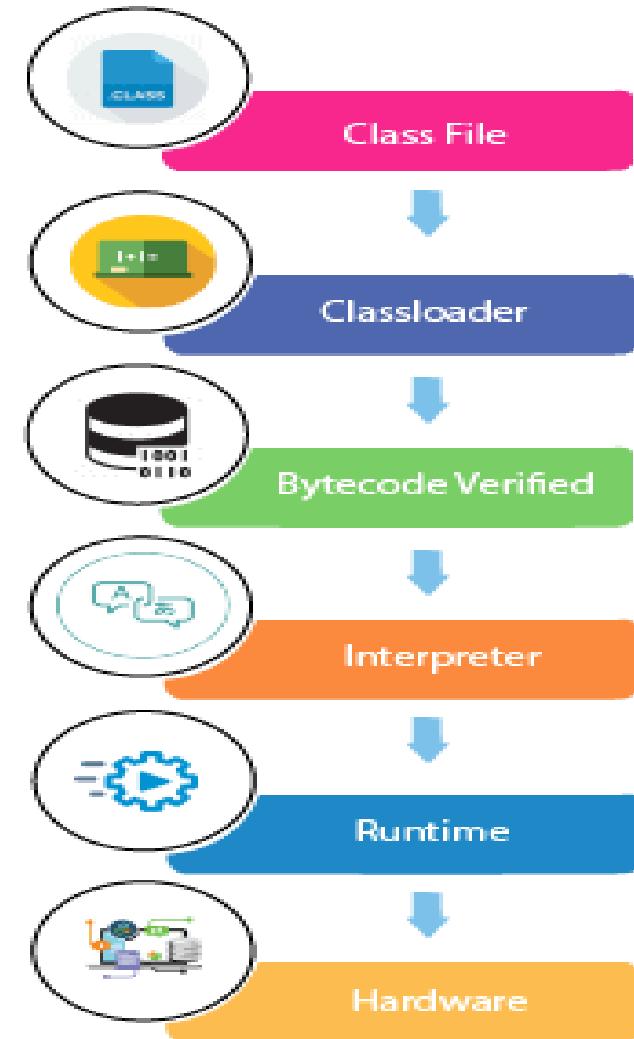
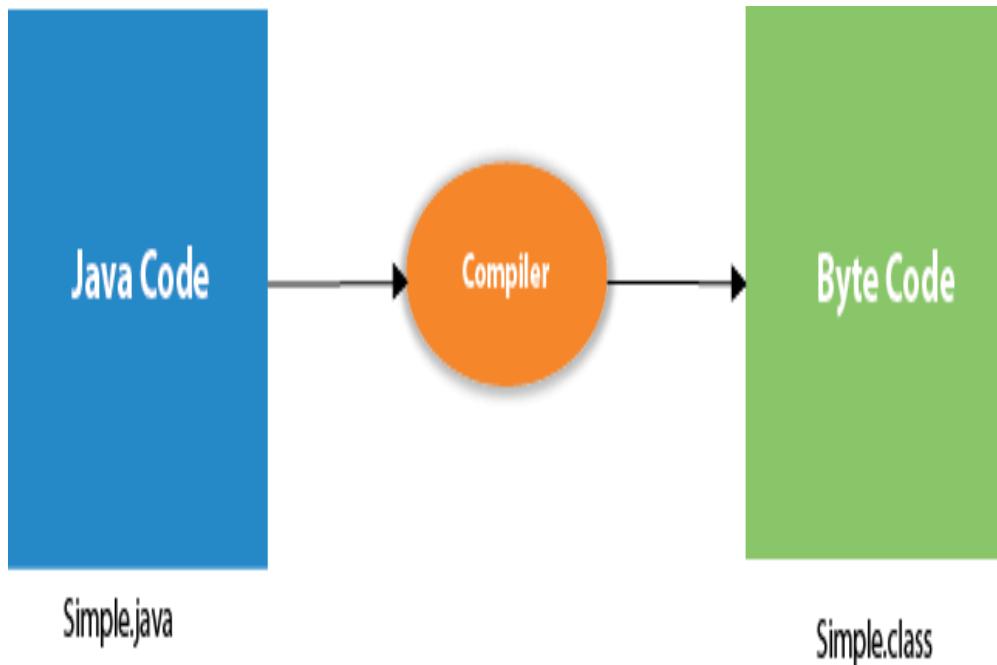
Standard output, print with new line



Parameters used in First Java Program:

- **class** keyword is used to declare a class in Java.
- **public** keyword is an access modifier that represents visibility.
- **static** is a keyword. If we declare any method as static, it is known as the static method. The core advantage of the static method is that there is no need to create an object to invoke the static method.
- **void** is the return type of the method. It means it doesn't return any value.
- **main** represents the starting point of the program.
- **String[] args** or **String args[]** is used for command line argument. We will discuss it in coming section.
- **System.out.println()** is used to print statement.

What happens at compile time and run time?



Difference between JDK, JRE, and JVM:

JVM (Java Virtual Machine)

It is a specification that provides a runtime environment in which Java bytecode can be executed.

JRE (Java Runtime Environment)

The Java Runtime Environment is a set of software tools which are used for developing Java applications. It contains a set of libraries + other files that JVM uses at run time.

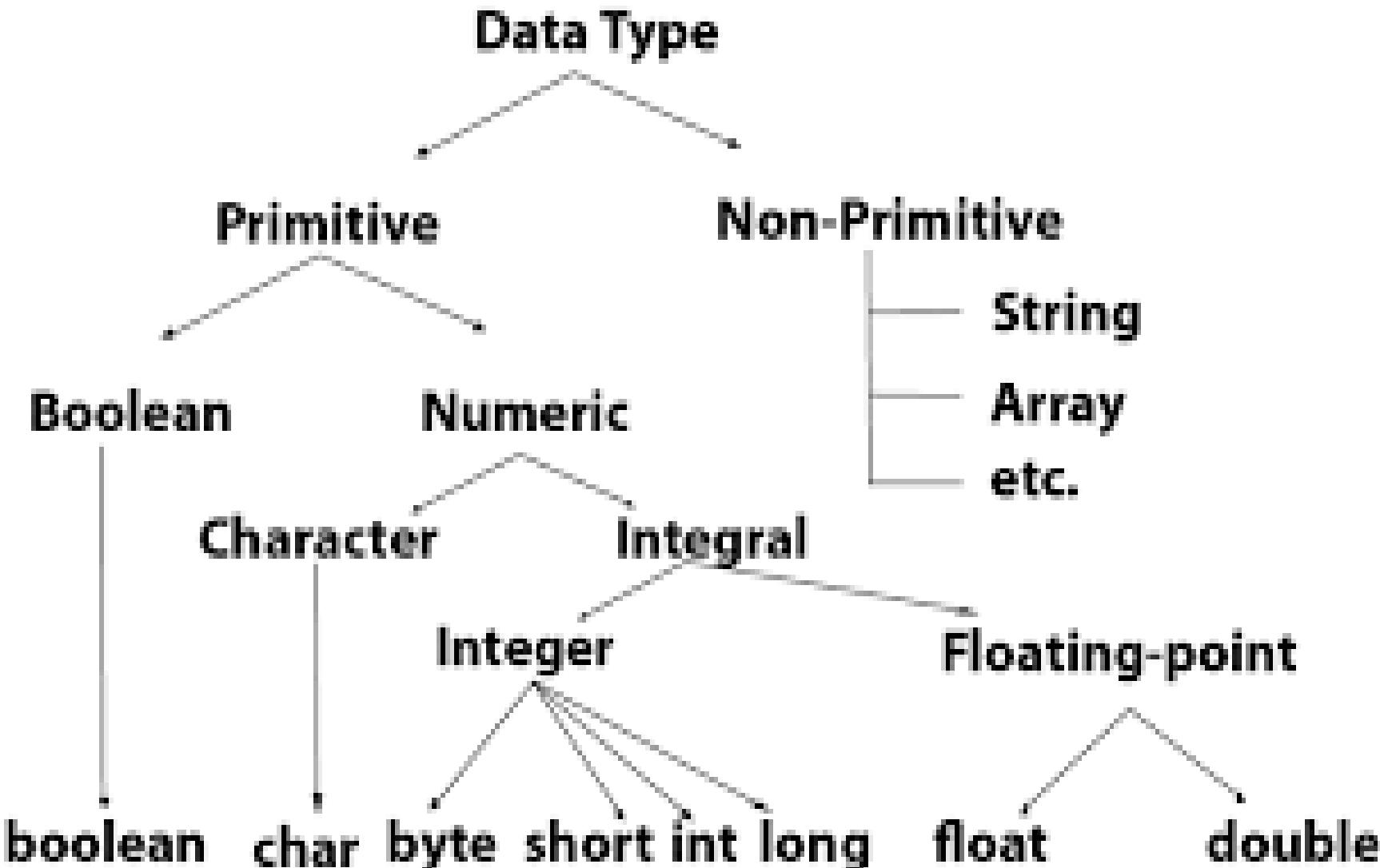
JDK (Java Development Kit)

The Java Development Kit (JDK) is a software development environment which is used to develop Java applications and applets.

The JDK contains a private Java Virtual Machine (JVM) and a few other resources such as an interpreter/loader (java), a compiler (javac), an archiver (jar), a documentation generator (Javadoc), etc.

Advantages of java

- java is simple language.
- Java does not support POINTERS.
- Java is first language in which programs can be executed using web.
- Write once run anywhere(WORA).



Primitive types:

Primitive type	Size	Minimum	Maximum	Wrapper type
boolean	1-bit	—	—	Boolean
char	16-bit	Unicode 0	Unicode $2^{16}-1$	Character
byte	8-bit	-128	+127	Byte
short	16-bit	-2^{15}	$+2^{15}-1$	Short
int	32-bit	-2^{31}	$+2^{31}-1$	Integer
long	64-bit	-2^{63}	$+2^{63}-1$	Long
float	32-bit	IEEE754	IEEE754	Float
double	64-bit	IEEE754	IEEE754	Double

The Scope of Variables in Java

- Each variable has a scope that specifies how long it will be seen and used in a program.

Local Variables:

Local variables are those that are declared inside of a method, constructor, or code block. Only the precise block in which they are defined is accessible.

```
public SumExample
{
    public void calculateSum() {
        int a = 5; // local variable
        int b = 10; // local variable
        int sum = a + b;
        System.out.println("The sum is: " + sum);
    } // a, b, and sum go out of scope here
}
```

The Scope of Variables in Java

Instance Variables:

Within a class, but outside of any methods, constructors, or blocks, instance variables are declared. They are accessible to all methods and blocks in the class and are a part of an instance of the class.

```
public class Circle {  
    double radius; // instance variable  
    public double calculateArea() {  
        return Math.PI * radius * radius;  
    }  
}
```

The Scope of Variables in Java

Class Variables (Static Variables):

In a class but outside of any method, constructor, or block, the static keyword is used to declare class variables, also referred to as static variables. Class variables can be accessed by using the class name and are shared by all instances of the class.

Example:

```
public class Bank {  
    static double interestRate; // class variable  
    // ...  
}
```

The Scope of Variables in Java

Method Parameters:

Variables that are supplied to a method when it is invoked are known as method parameters. The scope of method parameters is restricted to the method in which they are defined, making them local to that method.

Example:

```
public void printName(String name) { // name is a method parameter  
    System.out.println("Hello, " + name + "!");  
}
```

Presentation on Java Programming (IT207G)

UNIT No. II- 1



Wrapper classes in Java

- The **wrapper class in Java** provides the mechanism *to convert primitive into object and object into primitive.*
- **autoboxing** and **unboxing** feature convert primitives into objects and objects into primitives automatically.
- Java is an object-oriented programming language, so we need to deal with objects many times , where we need to use the wrapper classes.

Wrapper classes in Java

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

Wrapper classes in Java

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.

```
public class WrapperExample1{
    public static void main(String args[]){
        //Converting int into Integer
        int a=20;
        Integer i=Integer.valueOf(a);//converting int into Integer explicitly
        Integer j=a;//autoboxing, now compiler will write Integer.valueOf(a) internally
        System.out.println(a+" "+i+" "+j);
    }
}
```

Wrapper classes in Java

Unboxing:

- The automatic conversion of wrapper type into its corresponding primitive type is known as unboxing.
- It is the reverse process of autoboxing.

```
public class WrapperExample2{  
    public static void main(String args[]){  
        //Converting Integer to int  
        Integer a=new Integer(3);  
  
        int i=a.intValue(); //converting Integer to int explicitly  
        int j=a; //unboxing, now compiler will write a.intValue() internally  
  
        System.out.println(a+" "+i+" "+j);  
    }  
}
```

Java Classes and Objects

- Java is an Object-Oriented programming language. In Java, the **classes and objects** are the basic and important features of object-oriented programming system

➤ **Java Classes:**

- A **class** is a blueprint from which individual objects are created (or, we can say a class is a data type of an object type).
- Each class has its methods and attributes that can be accessed and manipulated through the objects.
- For example, if you want to create a class for *students*. In that case, "Student" will be a class, and student records (like *student1*, *student2*, etc) will be objects.

Types of Java Class Variables

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

- **Creating (Declaring) a Java Class**
- To create (declare) a class, you need to use access modifiers followed by **class** keyword and *class_name*.

Syntax to create a Java class

access_modifier class class_name

{

data members;

constructors;

methods;

...;

}

Java Objects

- An **object** is a variable of the type **class**, it is a basic component of an object-oriented programming system.
- A class has the methods and data members (attributes), these methods and data members are accessed through an **object**.
- Thus, an object is an instance of a class.

Creating (Declaring) a Java Object

- **Declaration** – A variable declaration with a variable name with an object type.
- **Instantiation** – The 'new' keyword is used to create the object.
- **Initialization** – The 'new' keyword is followed by a call to a constructor. This call initializes the new object.

Syntax to Create a Java Object

`Class_name object_name = new Class_name([parameters]);`

Accessing Instance Variables and Methods

- Instance variables and methods are accessed via created objects. To access an instance variable, following is the fully qualified path –

```
/* First create an object */  
ObjectReference = new Constructor();  
/* Now call a variable as follows */  
ObjectReference.variableName;  
/* Now you can call a class method as follows */  
ObjectReference.MethodName();
```

```
// Creating a Java class
class Dog {
    String breed;
    int age;

    public void input(String b, int a)
    {
        breed = b;
        age=a;
    }
    public void printDetails()
    {
        System.out.println("Dog details:");
        System.out.println(breed);
        System.out.println(age);
    }
}
public class Main {
    public static void main(String[] args) {
        Dog Dog obj = new Dog();
        obj.input("Golden Retriever",15);
        obj.printDetails(); }
}
```

```
import java.io.*;
public class Employee {
String name;
int age;
String designation;
double salary;
public Employee(String name) {
this.name = name; }
public void empAge(int empAge) {
age = empAge;
}
public void empDesignation(String empDesig) {
designation = empDesig;
}
public void empSalary(double empSalary) {
salary = empSalary;
}
public void printEmployee() {
System.out.println("Name:" + name );
System.out.println("Age:" + age );
System.out.println("Designation:" + designation );
System.out.println("Salary:" + salary); }
}
```

```
import java.io.*;  
  
public class EmployeeTest {  
  
    public static void main(String args[]) {  
  
        /* Create two objects using constructor */  
  
        Employee empOne = new Employee("James Smith");  
  
        Employee empTwo = new Employee("Mary Anne");  
  
        // Invoking methods for each object created  
  
        empOne.empAge(26);  
  
        empOne.empDesignation("Senior Software Engineer");  
  
        empOne.empSalary(1000);  
  
        empOne.printEmployee();  
  
        empTwo.empAge(21);  
  
        empTwo.empDesignation("Software Engineer");  
  
        empTwo.empSalary(500);  
  
        empTwo.printEmployee();  
  
    }  
  
}
```

OUTPUT

Name:James Smith
Age:26
Designation:Senior Software Engineer
Salary:1000.0

Name:Mary Anne Age:21
Designation:Software Engineer Salary:500.0

Java - Constructors

- A constructor initializes an object when it is created.
- It has the same name as its class and is syntactically similar to a method.
However, constructors have no explicit return type.
- Typically, you will use a constructor to give initial values to the instance variables defined by the class, or to perform any other start-up procedures required to create a fully formed object.
- All classes have constructors, whether you define one or not, because Java automatically provides a default constructor that initializes all member variables to zero.
- However, once you define your own constructor, the default constructor is no longer used.

No argument Constructors

As the name specifies the no argument constructors of Java does not accept any parameters instead, using these constructors the instance variables of a method will be initialized with fixed values for all objects.

```
Public class MyClass {  
    int num;  
    MyClass()  
    {  
        num = 100;  
    }  
}
```

```
public class ConsDemo {  
    public static void main(String args[]) { MyClass  
        t1 = new MyClass();  
        MyClass t2 = new MyClass();  
        System.out.println(t1.num + " " + t2.num); } }
```

Output
100 100

Parameterized Constructors

Most often, you will need a constructor that accepts one or more parameters. Parameters are added to a constructor in the same way that they are added to a method, just declare them inside the parentheses after the constructor's name.

```
class MyClass
{
    int x;
    MyClass(int i )
    {
        x = i;
    }
}
```

```
public class ConsDemo {
    public static void main(String args[]) {
        MyClass t1 = new MyClass( 10 );
        MyClass t2 = new MyClass( 20 );
        System.out.println(t1.x + " " + t2.x);
    }
}
```

Output

10 20

Constructor Overloading in Java

- Constructor overloading in Java is a technique of having more than one constructor with different parameter lists.
- They are arranged in a way that each constructor performs a different task.
- They are differentiated by the compiler by the number of parameters in the list and their types.

```
class Student5{
    int id;
    String name;
    int age;
    //creating two arg constructor
    Student5(int i, String n){
        id = i;
        name = n;
    }
    //creating three arg constructor
    Student5(int i, String n, int a){
        id = i;
        name = n;
        age = a;
    }
    void display(){
        System.out.println(id + " " + name + " " + age);
    }
}
```

```
public static void main(String args[]){
    Student5 s1 = new Student5(111, "Karan");
    Student5 s2 = new Student5(222, "Aryan", 25);
    s1.display();
    s2.display();
}
```

Output:

111	Karan	0
222	Aryan	25

Input from Keyboard

1. import java.util.Scanner;
2. Create object of Scanner class as follows:

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

3. Take input as follows
 - String name=sc.nextLine();
 - int age=sc.nextInt();
 - float salary=sc.nextFloat();

Practical Programs:

- WAP to create a class Arithmetic having data member a and b , also having six methods named 1. input() 2. add() 3. sub() 4. mul()
5. div() 6. output()
- WAP to create class Bike having data member name, engine, mileage and cost. Create three object by using constructor overloading and display details using display() method.
- WAP to create class Bank having data member name, accno and amount. Having methods input(), withdraw(), deposit() and checkBalance().

Presentation on Java Programming (IT207G)

UNIT No. II- 2



Access Modifiers in Java

- The access modifiers in Java specifies the accessibility or scope of a field, method, constructor, or class.
- We can change the access level of fields, constructors, methods, and class by applying the access modifier on it.
- There are four types of Java access modifiers:

Access Modifier	within class	within package	outside package by subclass only	outside package
Private	Y	N	N	N
Default	Y	Y	N	N
Protected	Y	Y	Y	N
Public	Y	Y	Y	Y

Access Modifiers in Java

Private: The access level of a private modifier is only within the class. It cannot be accessed from outside the class.

```
class A{  
    private int data=40;  
    private void msg(){System.out.println("Hello java");}  
}  
  
public class Simple{  
    public static void main(String args[]){  
        A obj=new A();  
        System.out.println(obj.data); //Compile Time Error  
        obj.msg(); //Compile Time Error  
    } }
```

Access Modifiers in Java

Default: The access level of a default modifier is only within the package. It cannot be accessed from outside the package. If you do not specify any access level, it will be the default.

- In this example, we have created two packages pack and mypack. We are accessing the A class from outside its package, since A class is not public, so it cannot be accessed from outside the package.

```
//save by A.java
package pack;
class A{
    void msg()
{
    System.out.println("Hello");
} }
```

```
package mypack;
import pack.*;
class B{
    public static void main(String args[]){
        A obj = new A(); //Compile Time Error
        obj.msg(); //Compile Time Error
    } }
```

Access Modifiers in Java

Protected: The access level of a protected modifier is within the package and outside the package through child class. If you do not make the child class, it cannot be accessed from outside the package.

- The protected access modifier can be applied on the data member, method and constructor. It can't be applied on the class.

```
//save by A.java
package pack;
class A{
    protected void msg()
    {
        System.out.println("Hello");
    }
}
```

Output:Hello

```
package mypack;
import pack.*;
class B extends A{
    public static void main(String args[]){
        A obj = new A(); //No Error
        obj.msg(); //No Error
    }
}
```

Access Modifiers in Java

Public: The access level of a public modifier is everywhere. It can be accessed from within the class, outside the class, within the package and outside the package.

- The **public access modifier** is accessible everywhere. It has the widest scope among all other modifiers.

```
//save by A.java
package pack;
public class A{
    public void msg()
{
    System.out.println("Hello");
} }
```

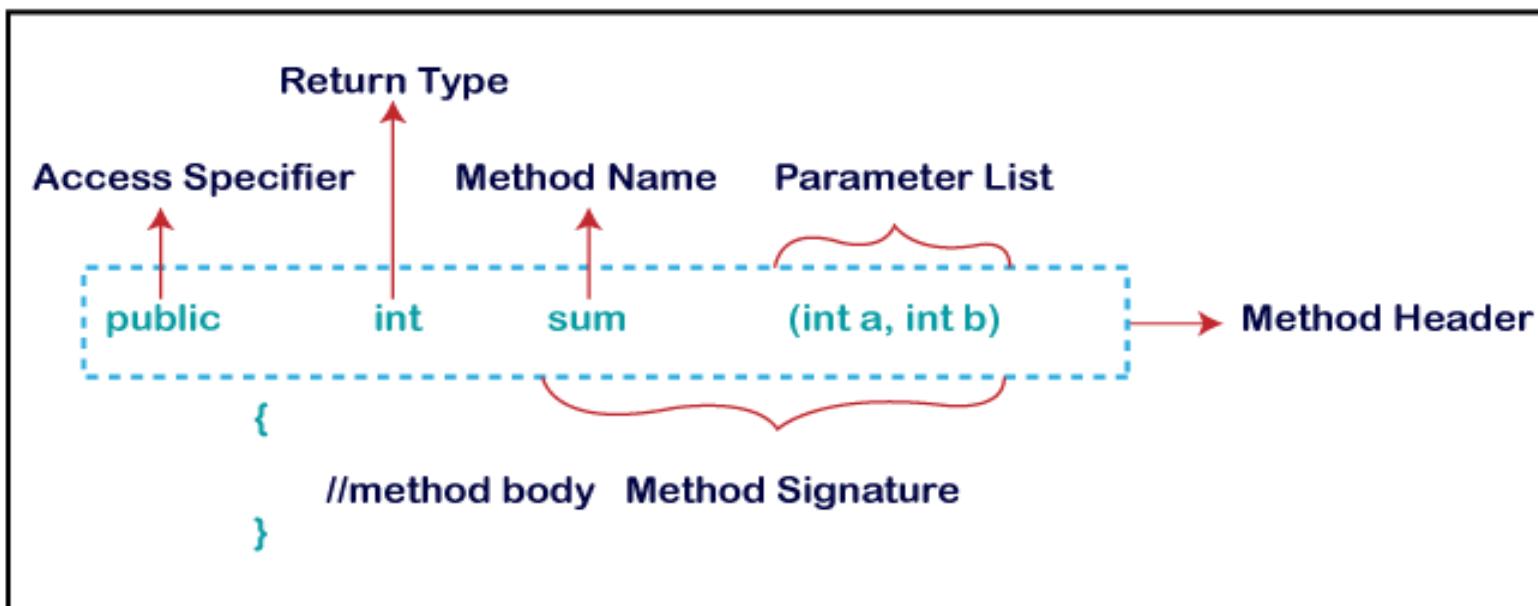
Output:Hello

```
package mypack;
import pack.*;
class B {
    public static void main(String args[]){
        A obj = new A(); //No Error
        obj.msg(); //No Error
    }
}
```

Method in Java

- A **method** is a block of code or collection of statements or a set of code grouped together to perform a certain task or operation. It is used to achieve the **reusability** of code. We write a method once and use it many times.

Method Declaration



Types of Method

➤ Predefined Method

In Java, predefined methods are the method that is already defined in the Java class libraries known as predefined methods. It is also known as the **standard library method** or **built-in method**.

```
public class Demo
{
    public static void main(String[] args)
    {
        // using the max() method of Math class
        System.out.print("The maximum number is: " + Math.max(9,7));
    }
}
```

Output: The maximum number is: 9

Types of Method

➤ User-defined Method

The method written by the user or programmer is known as **a user-defined** method. These methods are modified according to the requirement.

```
//user defined method
public static void findEvenOdd(int num)
{
    //method body
    if(num%2==0)
        System.out.println(num+" is even");
    else
        System.out.println(num+" is odd");
}
```

```
import java.util.Scanner;
public class EvenOdd
{
    public static void main (String args[])
    {
        //creating Scanner class object
        Scanner scan=new Scanner(System.in);
        System.out.print("Enter the number: ");
        //reading value from the user
        int num=scan.nextInt();
        //method calling
        findEvenOdd(num);
    }
}
```

Calling a Java Method

- For using a method, it should be called. There are two ways in which a method is called i.e., **method returns a value** or **returning nothing** (no return value).
- The methods returning void is considered as call to a statement. Lets consider an example –

```
public class ExampleMinNumber
{
    public static void main(String[] args)
    {
        int a = 11; int b = 6;
        int c = minFunc(a, b);
        System.out.println("Min Value =" + c);
    }
}
```

```
/** returns the minimum of two numbers */
public static int minFunc(int n1, int n2) {
    int min;
    if (n1 > n2)
        min = n2;
    else min = n1;
    return min;
}
```

The void Keyword with Java Methods

- The void keyword allows us to create methods which do not return a value.

```
public class ExampleVoid
{
    public static void main(String[] args) {
        methodRankPoints(255.7);
    }

    public static void methodRankPoints(double points)
    {
        if (points >= 202.5) {
            System.out.println("Rank:A1");
        }
        else if (points >= 122.4)
        {
            System.out.println("Rank:A2");
        }
        else
        {
            System.out.println("Rank:A3");
        }
    }
}
```

Java Methods Overloading

- When a class has two or more methods by the same name but different parameters, it is known as method overloading..

```
public class ExampleOverloading {  
  
    public static void main(String[] args) {  
        int a = 11;  
        int b = 6;  
        double c = 7.3;  
        double d = 9.4;  
        int result1 = minFunction(a, b);  
  
        // same function name with different parameters  
        double result2 = minFunction(c, d);  
  
        System.out.println("Minimum Value = " + result1);  
        System.out.println("Minimum Value = " + result2);  
    }  
}
```

Java Methods Overloading

```
// for integer
public static int minFunction(int n1, int n2) {
    int min;
    if (n1 > n2)
        min = n2;
    else
        min = n1;
    return min;
}

// for double
public static double minFunction(double n1, double n2)
{
    double min;
    if (n1 > n2)
        min = n2;
    else
        min = n1;
    return min;
}
```

Java Class Methods

- The **class methods** are methods that are declared within a [class](#).
- They perform specific operations and can access, modify the [class attributes](#).

Syntax:

```
public class class_name
{
    modifier returnType nameOfMethod(Parameter List)
    {
        // method body
    }
}
```

Accessing Java Class Methods:

To access a class method (public class method), you need to create an object first, then by using the object you can access the class method.

object_name.method_name([parameters]);

Example Program

```
class MyTest {  
    public int min (int n1, int n2)  
    {  
        int min;  
        if (n1 > n2)  
            min = n2;  
        else  
            min = n1;  
        return min;  
    }  
}
```

```
//Accessing Methods by Object  
  
public class Tester {  
  
    public static void main(String[] args) {  
        int a = 11;  
        int b = 6;  
  
        MyTest util = new MyTest();  
  
        int c = util.min(a, b);  
  
        System.out.println("MinValue = " + c);  
    }  
}
```

Java static keyword

- The **static keyword** in [Java](#) is used for memory management mainly.
- We can apply static keyword with [variables](#), methods, blocks and [nested classes](#).

1) Java static variable

If you declare any variable as static, it is known as a static variable.

- The static variable can be used to refer to the common property of all objects (which is not unique for each object).
- The static variable gets memory only once in the class area at the time of class loading.

Advantages of static variable:

It makes your program **memory efficient** (i.e., it saves memory).

Program of the counter without static variable

```
class Counter{  
    int count=0;//will get memory each time when the instance is created
```

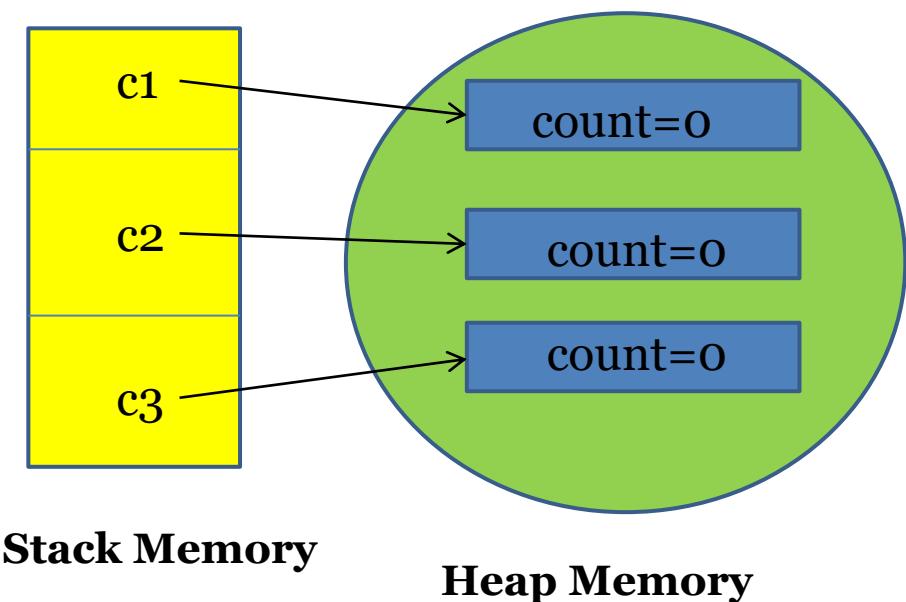
```
Counter(){  
    count++; //incrementing value
```

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

```
public static void main(String args[]){  
    //Creating objects  
    Counter c1=new Counter();  
    Counter c2=new Counter();  
    Counter c3=new Counter();  
}  
}
```

Output:

1
1
1



Program of the counter with static variable

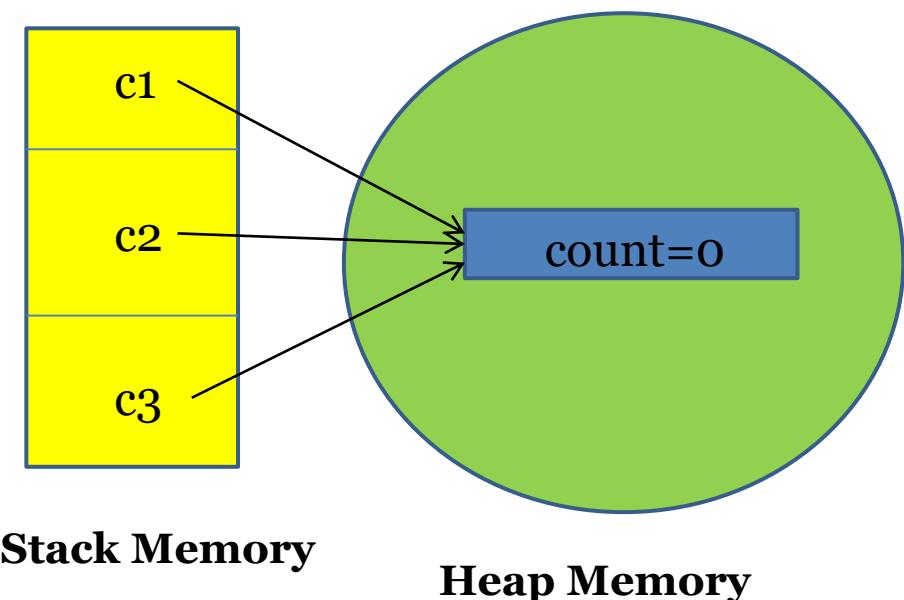
```
class Counter{  
    static int count=0;//will only once and retain its value get memory
```

```
Counter(){  
    count++; //incrementing value  
    System.out.println(count);  
}
```

```
public static void main(String args[]){  
    //Creating objects  
    Counter c1=new Counter();  
    Counter c2=new Counter();  
    Counter c3=new Counter();  
}
```

Output:

1
2
3



2) Java static method

- If you apply static keyword with any method, it is known as static method.
- A static method belongs to the class rather than the object of a class.
- A static method can be invoked without the need for creating an instance of a class.
- A static method can access static data member and can change the value of it.

```

class Student{
    int rollno;
    String name;
    String college;
static void change()
{
    college = "BBDIT";
}
//constructor to initialize the variable
Student(int r, String n)
{
    rollno = r;
    name = n;
}
//method to display values
void display()
{
    System.out.println(rollno+" "+name+" "+college);
}
}

```

```

public class TestStaticMethod{
public static void main(String args[]){
Student.change();

//creating objects
Student s1 = new Student(111,"Karan");
Student s2 = new Student(222,"Aryan");
Student s3 = new Student(333,"Sonoo");
//calling display method
s1.display();
s2.display();
s3.display();
}
}

```

Output:

111 Karan BBDIT
 222 Aryan BBDIT
 333 Sonoo BBDIT

3) Java static block

- Is used to initialize the static data member.
- It is executed before the main method at the time of classloading.
- Example of static block

```
class A2{  
    static  
    {  
        System.out.println("static block is invoked");  
    }  
    public static void main(String args[])  
    {  
        System.out.println("Hello main");  
    }  
}
```

Output:

static block is invoked
Hello main



Presentation on Java Programming (IT207G)

UNIT No. II- 3



Java static keyword

- The **static keyword** in [Java](#) is used for memory management mainly.
- We can apply static keyword with [variables](#), methods, blocks and [nested classes](#).

1) Java static variable

If you declare any variable as static, it is known as a static variable.

- The static variable can be used to refer to the common property of all objects (which is not unique for each object).
- The static variable gets memory only once in the class area at the time of class loading.

Advantages of static variable:

It makes your program **memory efficient** (i.e., it saves memory).

Program of the counter without static variable

```
class Counter{  
    int count=0;//will get memory each time when the instance is created
```

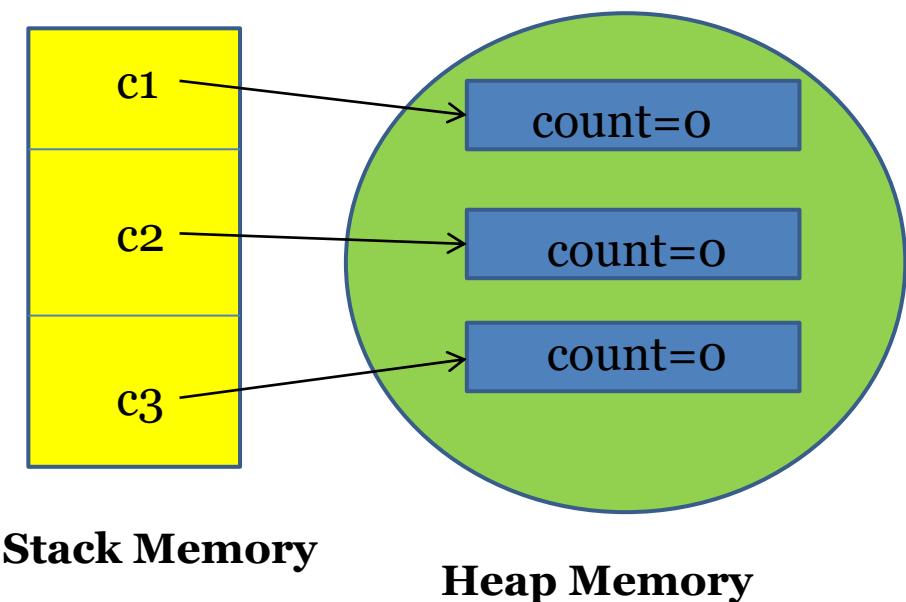
```
Counter(){  
    count++; //incrementing value
```

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

```
public static void main(String args[]){  
    //Creating objects  
    Counter c1=new Counter();  
    Counter c2=new Counter();  
    Counter c3=new Counter();  
}  
}
```

Output:

1
1
1



Program of the counter with static variable

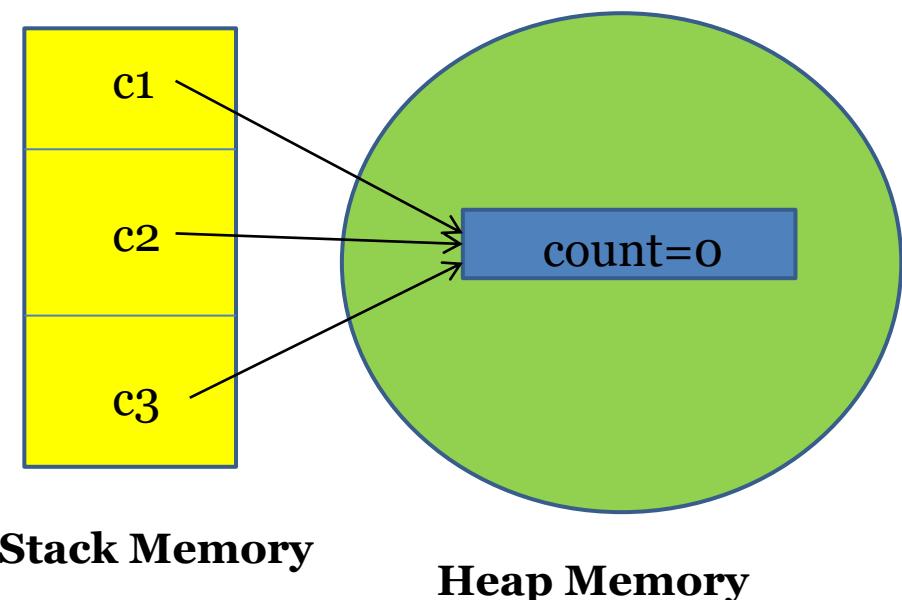
```
class Counter{  
    static int count=0;//will only once and retain its value get memory
```

```
Counter(){  
    count++; //incrementing value  
    System.out.println(count);  
}
```

```
public static void main(String args[]){  
    //Creating objects  
    Counter c1=new Counter();  
    Counter c2=new Counter();  
    Counter c3=new Counter();  
}
```

Output:

1
2
3



2) Java static method

- If you apply static keyword with any method, it is known as static method.
- A static method belongs to the class rather than the object of a class.
- A static method can be invoked without the need for creating an instance of a class.
- A static method can access static data member and can change the value of it.

```

class Student{
    int rollno;
    String name;
    String college;
static void change()
{
    college = "BBDIT";
}
//constructor to initialize the variable
Student(int r, String n)
{
    rollno = r;
    name = n;
}
//method to display values
void display()
{
    System.out.println(rollno+" "+name+" "+college);
}
}

```

```

public class TestStaticMethod{
public static void main(String args[]){
Student.change();

//creating objects
Student s1 = new Student(111,"Karan");
Student s2 = new Student(222,"Aryan");
Student s3 = new Student(333,"Sonoo");
//calling display method
s1.display();
s2.display();
s3.display();
}
}

```

Output:

111 Karan BBDIT
 222 Aryan BBDIT
 333 Sonoo BBDIT

3) Java static block

- Is used to initialize the static data member.
- It is executed before the main method at the time of classloading.
- Example of static block

```
class A2{  
    static  
    {  
        System.out.println("static block is invoked");  
    }  
    public static void main(String args[])  
    {  
        System.out.println("Hello main");  
    }  
}
```

Output:

static block is invoked
Hello main

HOME WORK

- WAP create class Store having data member storeName, itemprice and amount. Also having two methods sname(), sale() and display() . sname() is static method which contains store name, sale () method contains itemprice as a parameter which increases static amount variable and display() is used to display all data member name.
Create three object of class

Java final keyword

- The **final keyword** in java is used to restrict the user. The java final keyword can be used in many context. Final can be:
- **Variable:** If you make any variable as final, you cannot change the value of final variable(It will be constant).
- **Method:** If you make any method as final, you cannot override it.
- **Class:** If you make any class as final, you cannot extend it.

Example of final variable

```
class Bike9{  
    final int speedlimit=90;//final variable  
    void run(){  
        speedlimit=400;  
    }  
    public static void main(String args[]){  
        Bike9 obj=new Bike9();  
        obj.run();  
    }  
}//end of class
```

Output: Compile Time Error

Example of final method

```
class Bike{  
    final void run(){System.out.println("running");}  
}  
  
class Honda extends Bike{  
    void run()  
    {  
        System.out.println("running safely with 100kmph");  
    }  
    public static void main(String args[]){  
        Honda honda= new Honda();  
        honda.run();  
    }  
}
```

Output: Compile Time Error

Example of final class

```
final class Bike  
{ }
```

```
class Honda1 extends Bike{  
    void run()  
{  
    System.out.println("running safely with 100kmph");  
}  
    public static void main(String args[]){  
        Honda1 honda= new Honda1();  
        honda.run();  
    }  
}
```

Output: Compile Time Error

Java this keyword

There can be a lot of usage of **Java this keyword**. In Java, this is a **reference variable** that refers to the current object.

Usage of Java this Keyword

There can be a lot of usage of java this keyword. In java, this is a reference variable that refers to the current object.

01

this can be used to refer current class instance variable.

02

this can be used to invoke current class method (implicity)

03

this() can be used to invoke current class Constructor.

04

this can be passed as an argument in the method call.

05

this can be passed as argument in the constructor call.

06

this can be used to return the current class instance from the method

1) this: to refer current class instance variable

The this keyword can be used to refer current class instance variable. If there is ambiguity between the instance variables and parameters, this keyword resolves the problem of ambiguity.

```
class Student{  
int rollno;  
String name;  
float fee;  
  
Student(int rollno,String name,float fee)  
{  
rollno=rollno;  
name=name;  
fee=fee;  
}  
void display()  
{  
System.out.println(rollno+" "+name+" "+fee);  
}
```

```
class TestThis1{  
public static void main(String args[]){  
Student s1=new Student(111,"ankit",5000);  
Student s2=new Student(112,"sumit",6000);  
s1.display();  
s2.display();  
} }
```

OUTPUT:
0 null 0.0
0 null 0.0

1) Solution of the above problem by this keyword

```
class Student{  
    int rollno;  
    String name;  
    float fee;  
  
    Student(int rollno, String name, float fee)  
    {  
        this.rollno=rollno;  
        this.name=name;  
        this.fee=fee;  
    }  
    void display()  
    {  
        System.out.println(rollno+ " "+name+ " "+fee);  
    }  
}
```

```
class TestThis1{  
    public static void main(String args[]){  
        Student s1=new Student(111, "ankit", 5000);  
        Student s2=new Student(112, "sumit", 6000);  
        s1.display();  
        s2.display();  
    } }  
}
```

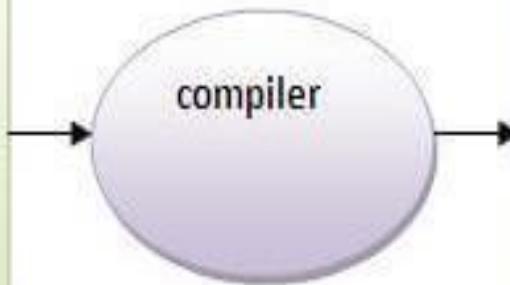
OUTPUT:

```
111 ankit 5000.0  
112 sumit 6000.0
```

2) this: to invoke current class method

- You may invoke the method of the current class by using the this keyword.
- If you don't use the this keyword, compiler automatically adds this keyword while invoking the method. Let's see the example

```
class A{  
    void m(){  
    }  
    void n(){  
        m();  
    }  
    public static void main(String args[]){  
        new A().n();  
    }  
}
```



```
class A{  
    void m(){  
    }  
    void n(){  
        this.m();  
    }  
    public static void main(String args[]){  
        new A().n();  
    }  
}
```

3) this() : to invoke current class constructor

- The this() constructor call can be used to invoke the current class constructor.
- It is used to reuse the constructor. In other words, it is used for constructor chaining.

```
class A{  
    A()  
    {  
        System.out.println("hello a");  
    }  
  
    A(int x)  
    {  
        this();  
  
        System.out.println(x);  
    } }
```

```
class TestThis5  
{  
    public static void main(String args[])  
    {  
        A a=new A(10);  
    }  
}
```

Output:
hello a
10

Java Garbage Collection

- In java, garbage means unreferenced objects.
- Garbage Collection is process of reclaiming the runtime unused memory automatically.
- In other words, it is a way to destroy the unused objects.
- 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.

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.

finalize() method

- The finalize() method is invoked each time before the object is garbage collected.
- This method can be used to perform cleanup processing.
- This method is defined in Object class as:

protected void finalize(){}

```
public class TestGarbage1{
    public void finalize()
    {
        System.out.println("object is garbage collected");
    }
    public static void main(String args[])
    {
        TestGarbage1 s1=new TestGarbage1();
        TestGarbage1 s2=new TestGarbage1();
        s1=null;
        s2=null;
    }
}
```

OUTPUT:

object is garbage collected
object is garbage collected

Home Work

1. **WAP to create** class ElectricityBill having data members Name, conNo, address and units. Also having one constructor which takes input from keyboard , two methods i.e. calculateBill() and display().

To calculates the electricity bill based on the following rates:

For the first 100 units, the rate is 1.20 rs. per unit.

For the next 200 units (101-300), the rate is 2.00 rs. per unit.

For the next 200 units (301-500), the rate is 3.00 rs. per unit.

For any units above 500, the rate is 4.00 rs. per unit.

Home Work (using this keyword)

1. **WAP to create** Employee class with a constructor to initialize the name and basic salary of the employee.

It also calculates the House Rent Allowance (HRA) and Dearness Allowance (DA) based on the basic salary.

The calculateGrossSalary() method calculates the gross salary by adding the basic salary, HRA, and DA.

Finally, the displaySalary() method displays all the salary components along with the gross salary.

Presentation on Java Programming (IT207G)

UNIT No. III- 1



Inheritance in Java

- **Inheritance in Java** is a mechanism in which one object acquires all the properties and behaviors of a parent object.
- The idea behind inheritance in Java is that you can create new classes that are built upon existing classes.
- When you inherit from an existing class, you can reuse methods and fields of the parent class.
- Moreover, you can add new methods and fields in your current class also.

Why use inheritance in java

- For Method Overriding (so runtime polymorphism can be achieved).
- For Code Reusability

Inheritance in Java

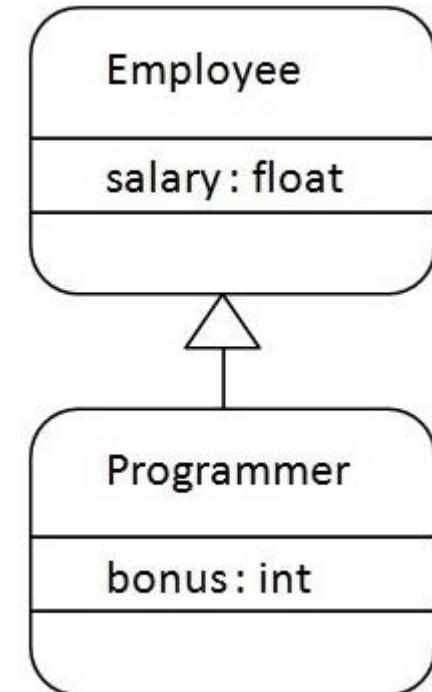
The syntax of Java Inheritance

```
class Subclass-name extends Superclass-name
{
    //methods and fields
}
```

The **extends keyword** indicates that you are making a new class that derives from an existing class. The meaning of "extends" is to increase the functionality

Inheritance in Java

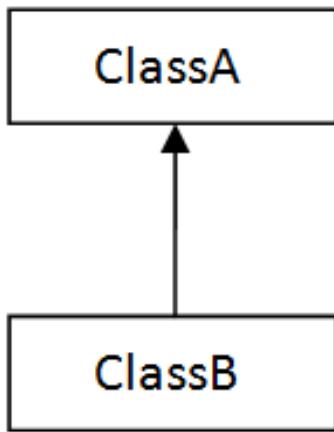
```
class Employee{  
    float salary=40000;  
}  
  
class Programmer extends Employee{  
    int bonus=10000;  
    public static void main(String args[]){  
        Programmer p=new Programmer();  
        System.out.println("Programmer salary is:"+p.salary);  
        System.out.println("Bonus of Programmer is:"+p.bonus);  
    }  
}
```



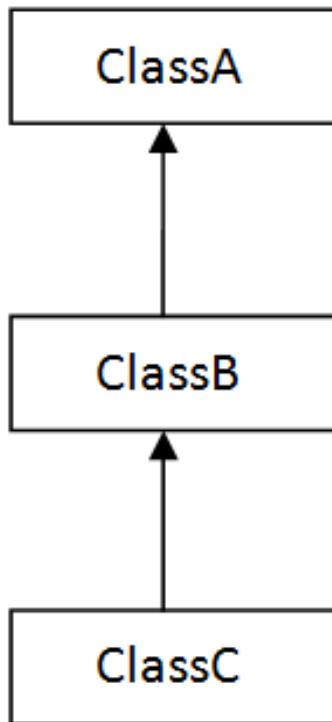
Output:

Programmer salary is:40000.0
Bonus of programmer is:10000

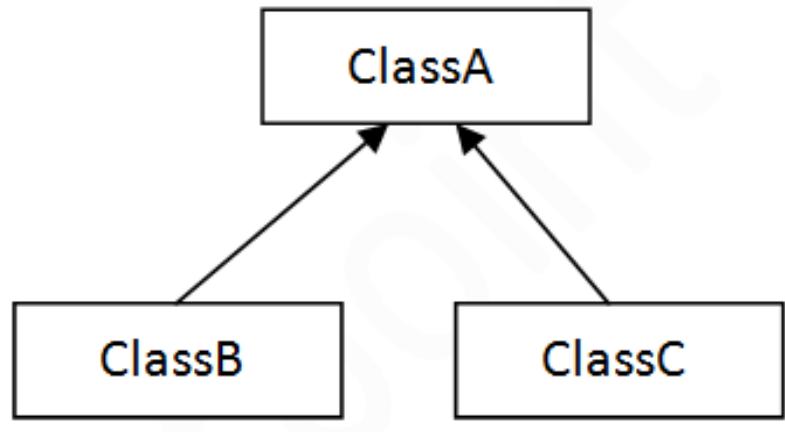
Types of inheritance in java



1) Single

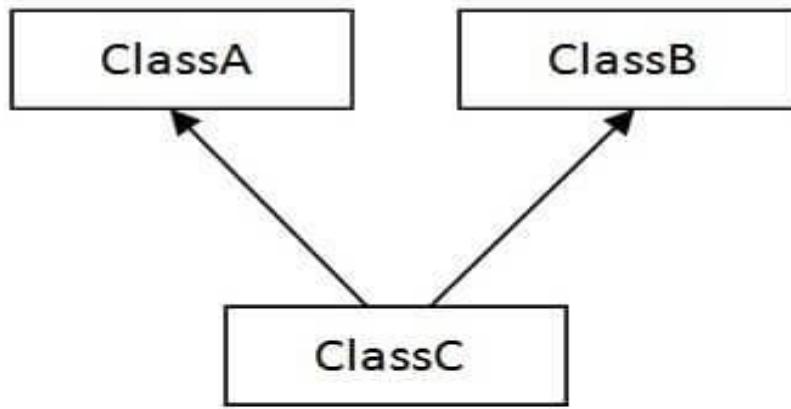


2) Multilevel

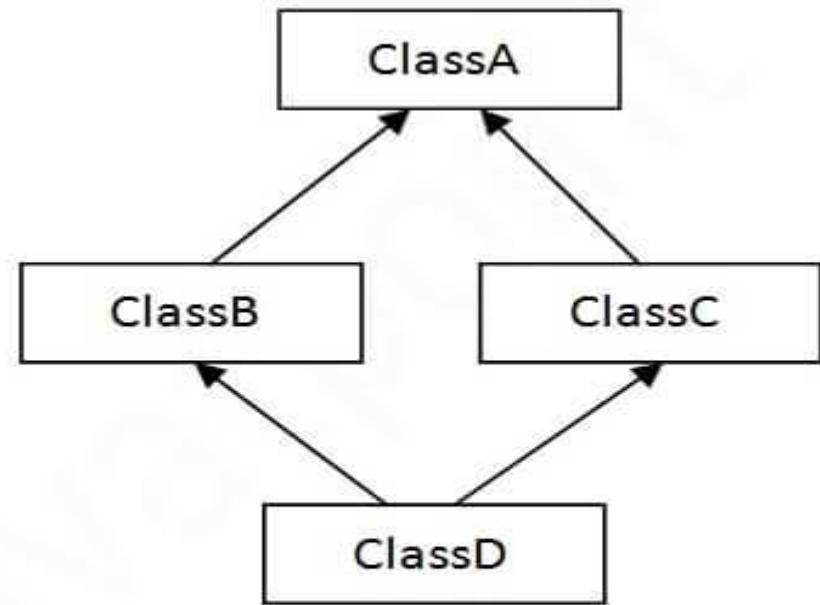


3) Hierarchical

Types of inheritance in java

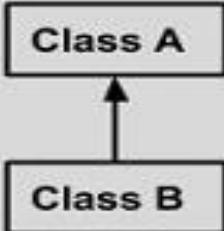
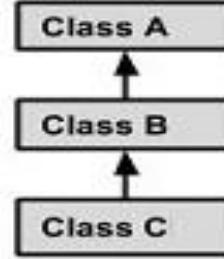
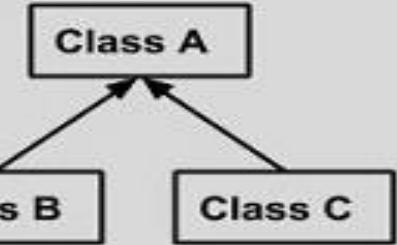
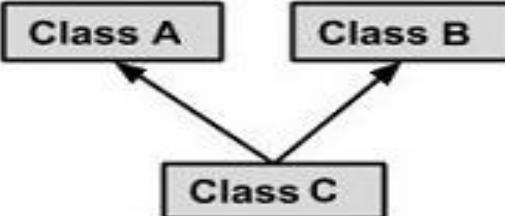


4) Multiple



5) Hybrid

Types of inheritance in java

Single Inheritance		public class A { } public class B extends A { }
Multi Level Inheritance		public class A { } public class B extends A {..... } public class C extends B {..... }
Hierarchical Inheritance		public class A { } public class B extends A {..... } public class C extends A {..... }
Multiple Inheritance		public class A { } public class B {..... } public class C extends A,B { } <i>// Java does not support multiple Inheritance</i>

Single Inheritance Example

```
class Animal{
void eat()
{System.out.println("eating...");}
}

class Dog extends Animal{
void bark()
{
System.out.println("barking...");}
}

class Test {
public static void main(String args[]){
Dog d=new Dog();
d.bark();
d.eat();
}}
```

Output:
barking...
eating...

Multilevel Inheritance Example

```
class Animal{  
void eat(){System.out.println("eating...");}  
}  
  
class Dog extends Animal{  
void bark(){System.out.println("barking...");}  
}  
  
class BabyDog extends Dog{  
void weep(){System.out.println("weeping...");}  
}  
  
class Test{  
public static void main(String args[]){  
BabyDog d=new BabyDog();  
d.weep();  
d.bark();  
d.eat();  }}  

```

Output:

weeping...
barking...
eating...

Hierarchical Inheritance Example

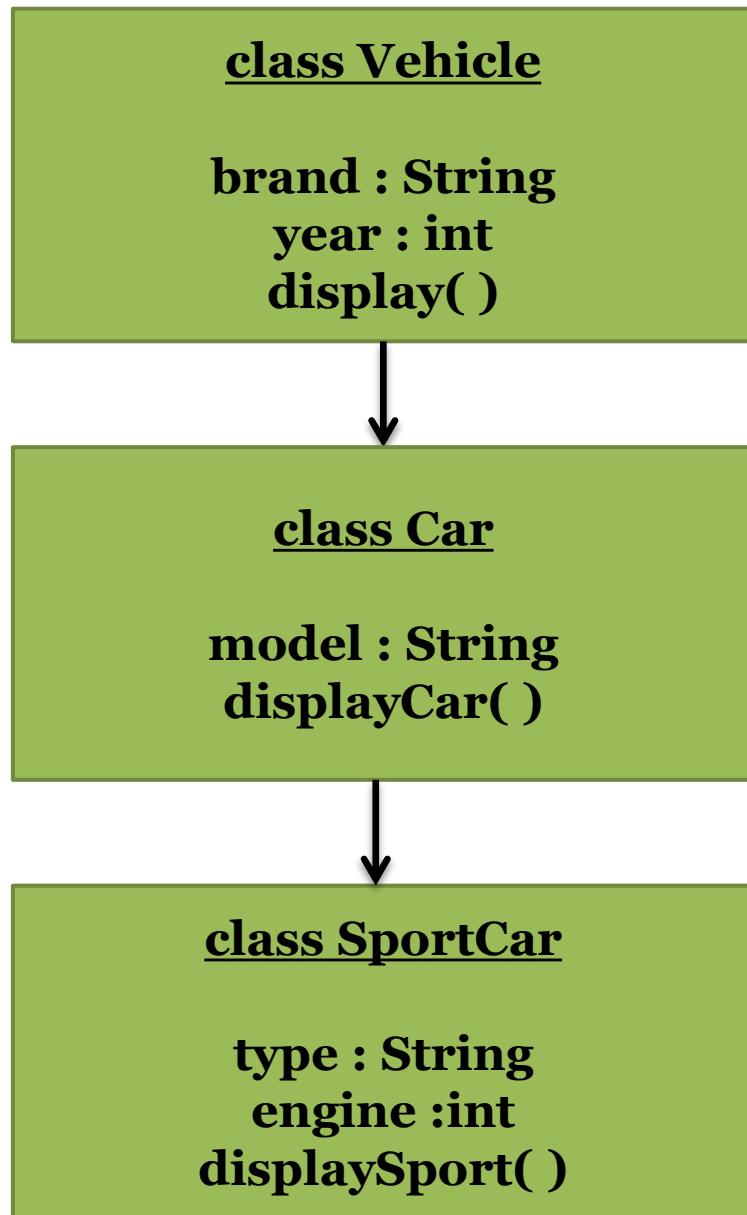
```
class One {  
    public void printOne() {  
        System.out.println("printOne() Method of Class One");  
    }  
}  
// Derived class 1  
class Two extends One {  
    public void printTwo() {  
        System.out.println("Two() Method of Class Two"); }  
}  
  
// Derived class 2  
class Three extends One {  
    public void printThree() {  
        System.out.println("printThree() Method of Class 3");  
    }  
}
```

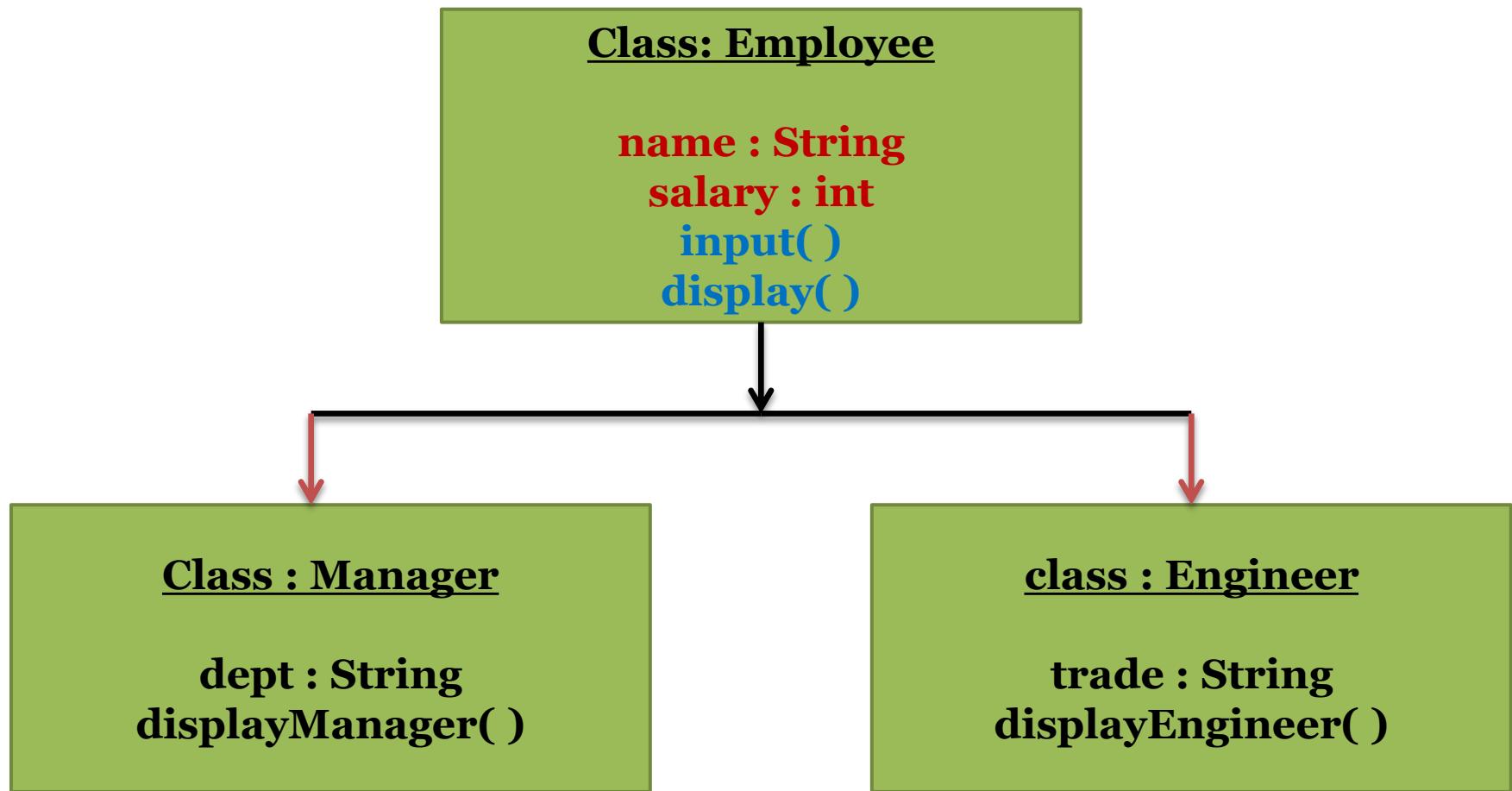
Hierarchical Inheritance Example

```
public class Test
{
    public static void main(String args[])
    {
        Two obj1 = new Two();
        Three obj2 = new Three();
        obj1.printOne();
        obj2.printOne();
    }
}
```

OUTPUT:

printOne() Method of Class One
printOne() Method of Class One





Presentation on Java Programming (IT207G)

UNIT No. III- 2



Java Inheritance: The **super** Keyword

The **super** keyword in Java is a reference variable which is used to refer immediate parent class object.

Whenever you create the instance of subclass, an instance of parent class is created implicitly which is referred by super reference variable.

Java Inheritance: The super Keyword

Usage of Super Keyword

1

Super can be used to refer immediate parent class instance variable.

2

Super can be used to invoke immediate parent class method.

3

`super()` can be used to invoke immediate parent class constructor.

1) super is used to refer immediate parent class instance variable.

- We can use super keyword to access the data member or field of parent class.
- It is used if parent class and child class have same fields.

```
class Animal{  
String color="white";  
}  
  
class Dog extends Animal{  
String color="black";  
  
void printColor(){  
  
System.out.println(color);  
//prints color of Dog class  
  
System.out.println(super.color);  
//prints color of Animal class  
} }
```

```
class TestSuper1  
{  
public static void main(String args[])  
{  
Dog d=new Dog();  
d.printColor();  
}}
```

Output:
black
white

2) super can be used to invoke parent class method.

```
class Animal{
void eat()
{System.out.println("eating...");}
}
class Dog extends Animal{
void eat()
{
System.out.println("eating bread...");
}
void bark()
{System.out.println("barking...");}
void work()
{
super.eat();
bark();
} }
```

```
class TestSuper2{
public static void main(String args[])
{
Dog d=new Dog();
d.work();
}
}
```

Output:
eating...
barking...

3) super is used to invoke parent class constructor.

```
class Animal{  
Animal()  
{  
System.out.println("animal is created");  
}  
}  
  
class Dog extends Animal{  
Dog()  
{  
super();  
System.out.println("dog is created");  
}  
}
```

```
class TestSuper2{  
public static void main(String args[])  
{  
Dog d=new Dog();  
}  
}
```

Output:

animal is created
dog is created

Real Example

```
class Person
{
    int id;
    String name;
    Person(int id, String name)
    {
        this.id=id;
        this.name=name;
    }
}
```

```
class Emp extends Person{
    float salary;
    Emp(int id, String name, float salary)
    {
        super(id, name);
        //reusing parent constructor
        this.salary=salary;
    }
    void display()
    {
        System.out.println(id+ " "+name+ " "+salary);
    }
}
```

```
class TestSuper5{
    public static void main(String[] args)
    {
        Emp e1=new Emp(1, "ankit", 45000f);
        e1.display();
    }
}
```

Output:
1 ankit 45000

Q) Why multiple inheritance is not supported in java?

To reduce the complexity and simplify the language, multiple inheritance is not supported in java.

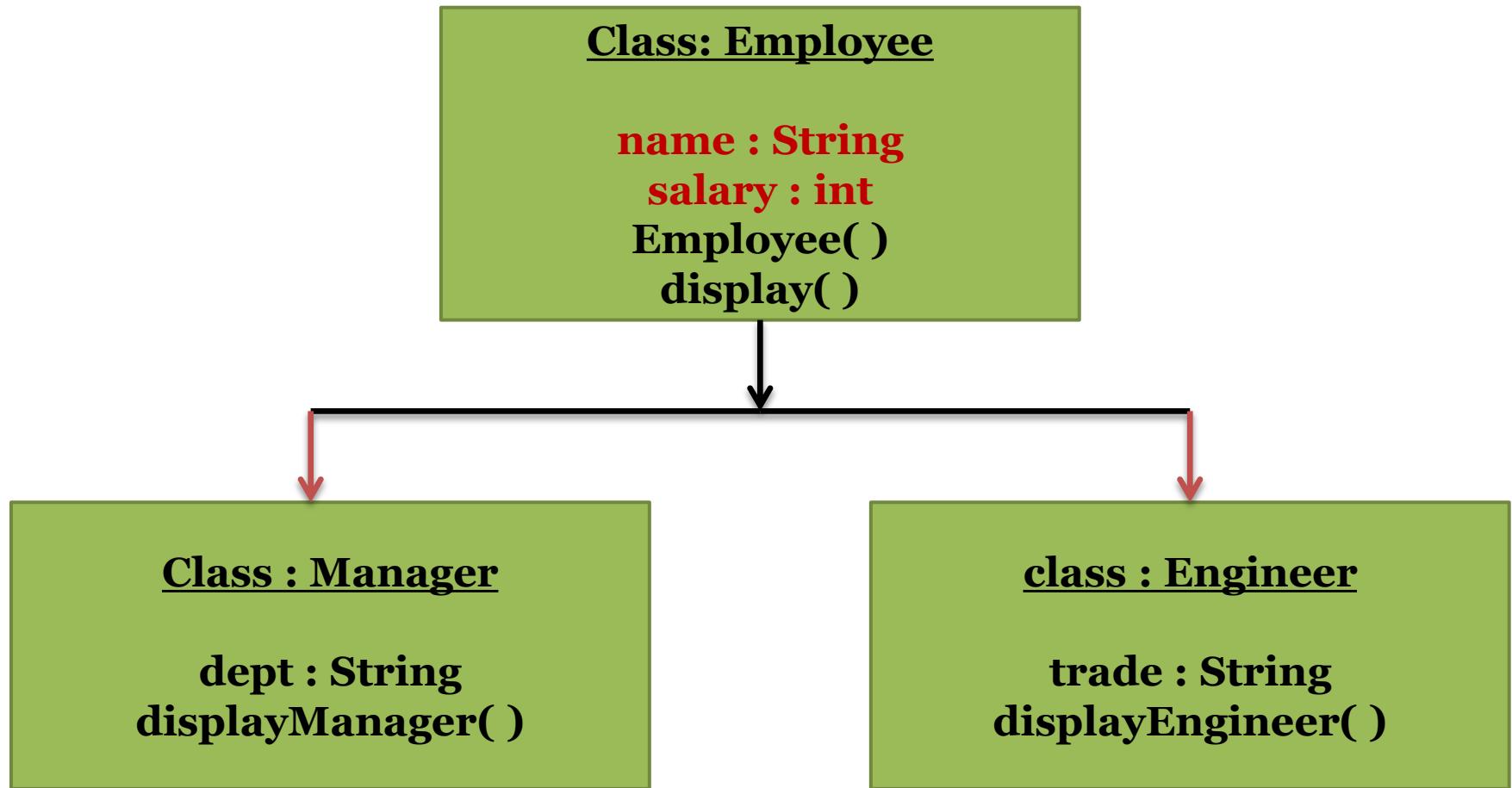
```
class A{
void msg()
{System.out.println("Hello");}
}

class B{
void msg()
{System.out.println("Welcome");}
}

class C extends A, B
{
public static void main(String args[]){
C obj=new C();
obj.msg();

}
}
```

Compile Time Error



Presentation on Java Programming (IT207G)

UNIT No. III- 3



Java Inheritance: Method Overriding

Method overriding allows us to achieve run time polymorphism and is used for writing specific definitions of a subclass method that is already defined in the superclass.

The method is superclass and overridden method in the subclass should have the same declaration signature such as parameters list, type, and return type..

Usage of Java Method Overriding

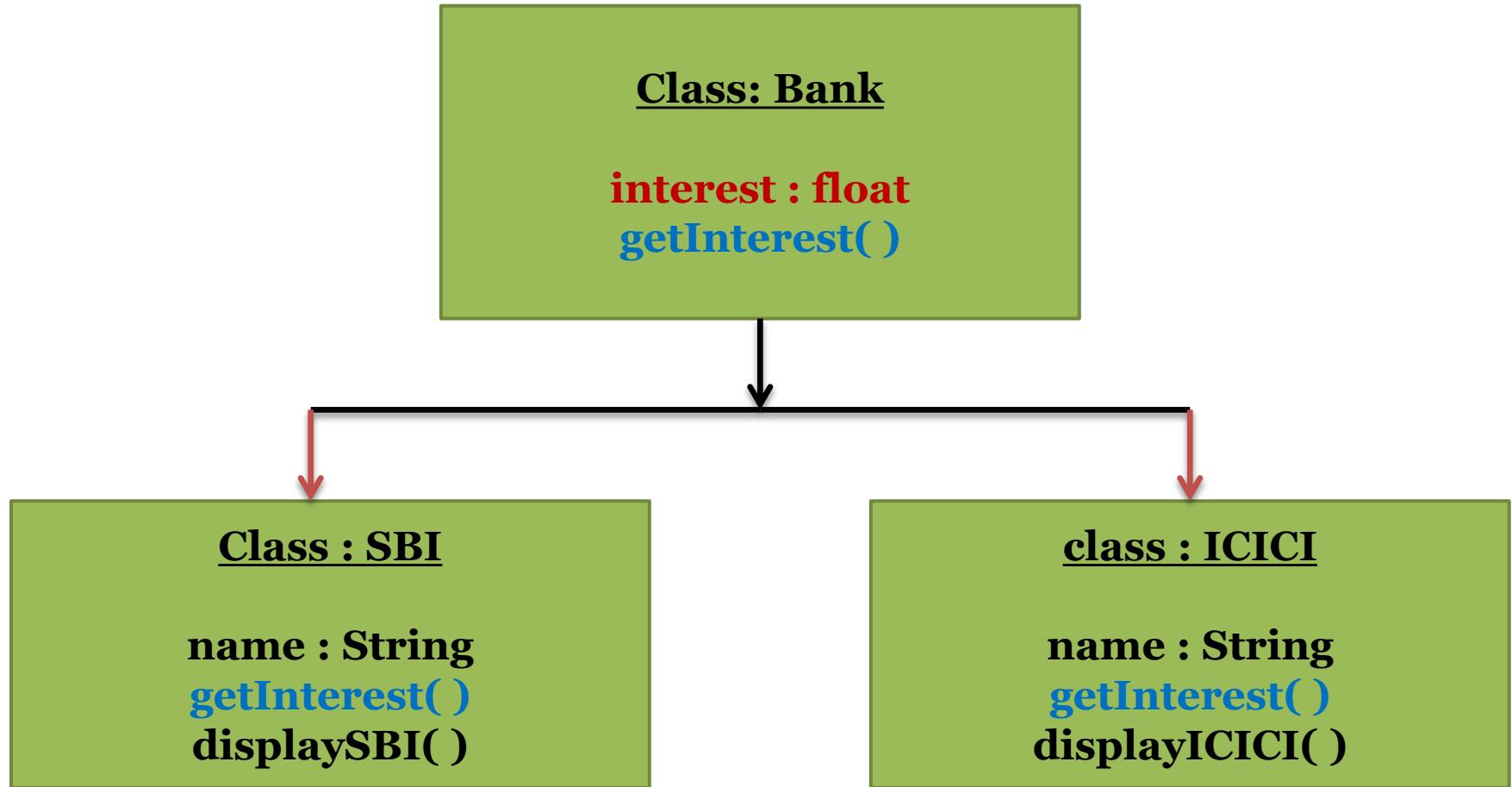
- Method overriding is used for achieving run-time polymorphism.
- Method overriding is used for writing specific definition of a subclass method (this method is known as the overridden method).

Example of Method Overriding in Java

```
class Animal {  
    public void move()  
    {  
        System.out.println("Animals can move");  
    } }  
class Dog extends Animal  
{ public void move()  
{  
    System.out.println("Dogs can walk and run");  
} }
```

```
public class TestDog {  
    public static void main(String args[])  
    {  
        Dog a = new Dog();  
        a.move();  
    } }
```

Output: Dogs can walk and run



Abstraction in Java

A class which is declared with the **abstract keyword** is known as an abstract class in [Java](#).

It can have **abstract and non-abstract methods** (method with the body).

Abstraction is a process of hiding the implementation details and showing only functionality to the user.

Another way, it shows only essential things to the user and hides the internal details, **for example**, sending SMS where you type the text and send the message. You don't know the internal processing about the message delivery.

Java Abstract Class

A class which is declared as abstract is known as an **abstract class**. It can have abstract and non-abstract methods. It needs to be extended and its method implemented. It cannot be instantiated.

Rules for Java Abstract class



- 1 An abstract class must be declared with an abstract keyword.
- 2 It can have abstract and non-abstract methods.
- 3 It cannot be instantiated.
- 4 It can have final methods
- 5 It can have constructors and static methods also.

Abstract Method in Java

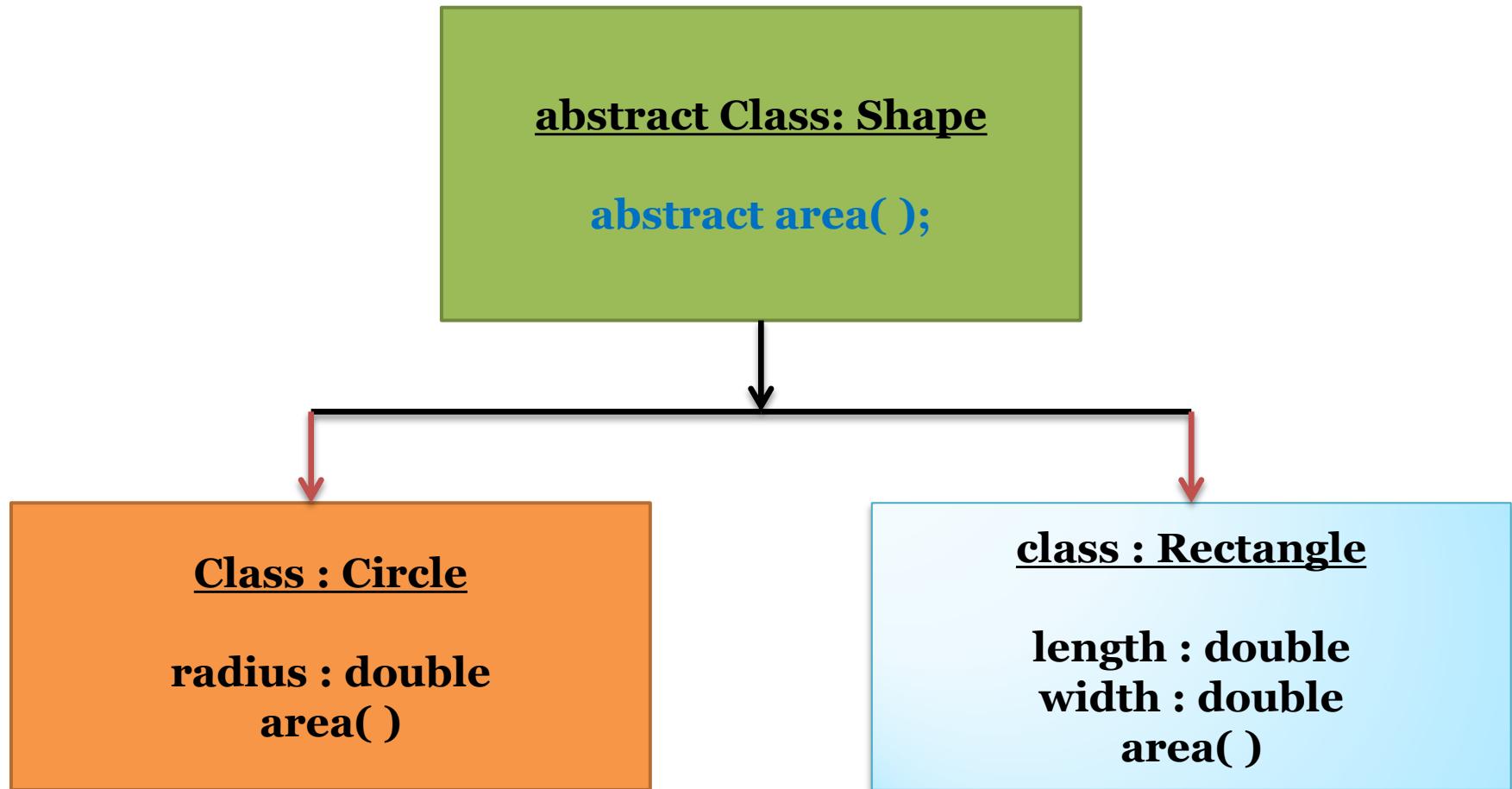
A method which is declared as abstract and does not have implementation is known as an abstract method.

For Example: **abstract void printStatus();**//no method body and abstract

```
abstract class Bike
{
    abstract void run();
}

class Honda extends Bike
{
    void run()
    {
        System.out.println("running safely");
    }

    public static void main(String args[])
    {
        Honda obj = new Honda();
        obj.run();
    }
}
```



Interface in Java

An **interface in Java** is a blueprint of a class. It has static constants and abstract methods.

- The interface in Java is *a mechanism to achieve [abstraction](#)*. There can be only abstract methods in the Java interface, not method body.
- It is used to achieve abstraction and multiple [inheritance in Java](#).
- In other words, you can say that interfaces can have abstract methods and variables. It cannot have a method body.

Why use Java interface?

It is used to achieve abstraction.

1

2

By interface, we can support the functionality of multiple inheritance.

3

It can be used to achieve loose coupling.

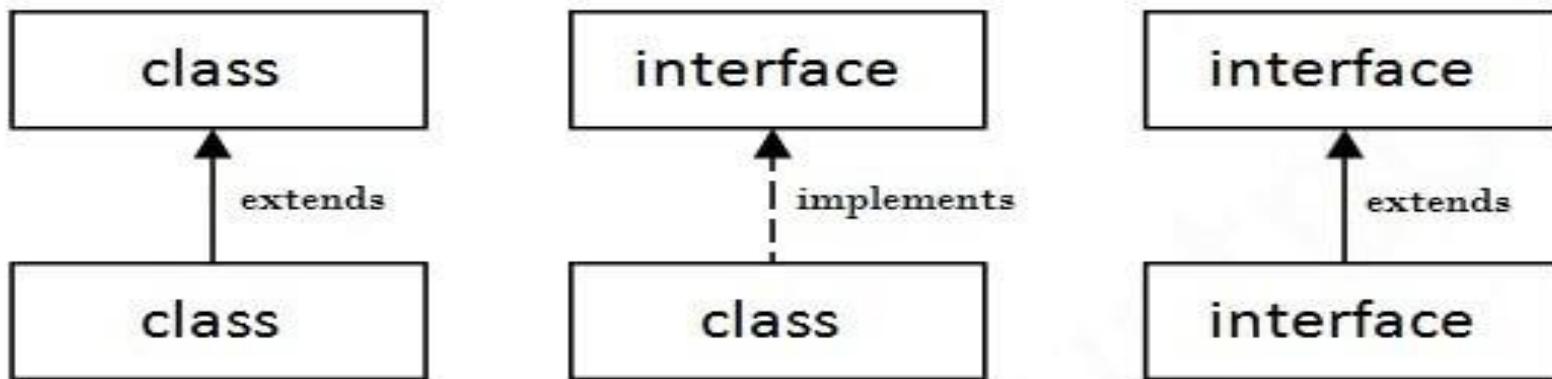
How to declare an interface?

1. An interface is declared by using the interface keyword.
2. It provides total abstraction; means all the methods in an interface are declared with the empty body, and all the fields are public, static and final by default.
3. A class that implements an interface must implement all the methods declared in the interface..

```
interface <interface_name>{  
    // declare constant fields  
    // declare methods that abstract by default.  
}
```

The relationship between classes and interfaces

As shown in the figure given below, a class extends another class, an interface extends another interface, but a **class implements an interface**.

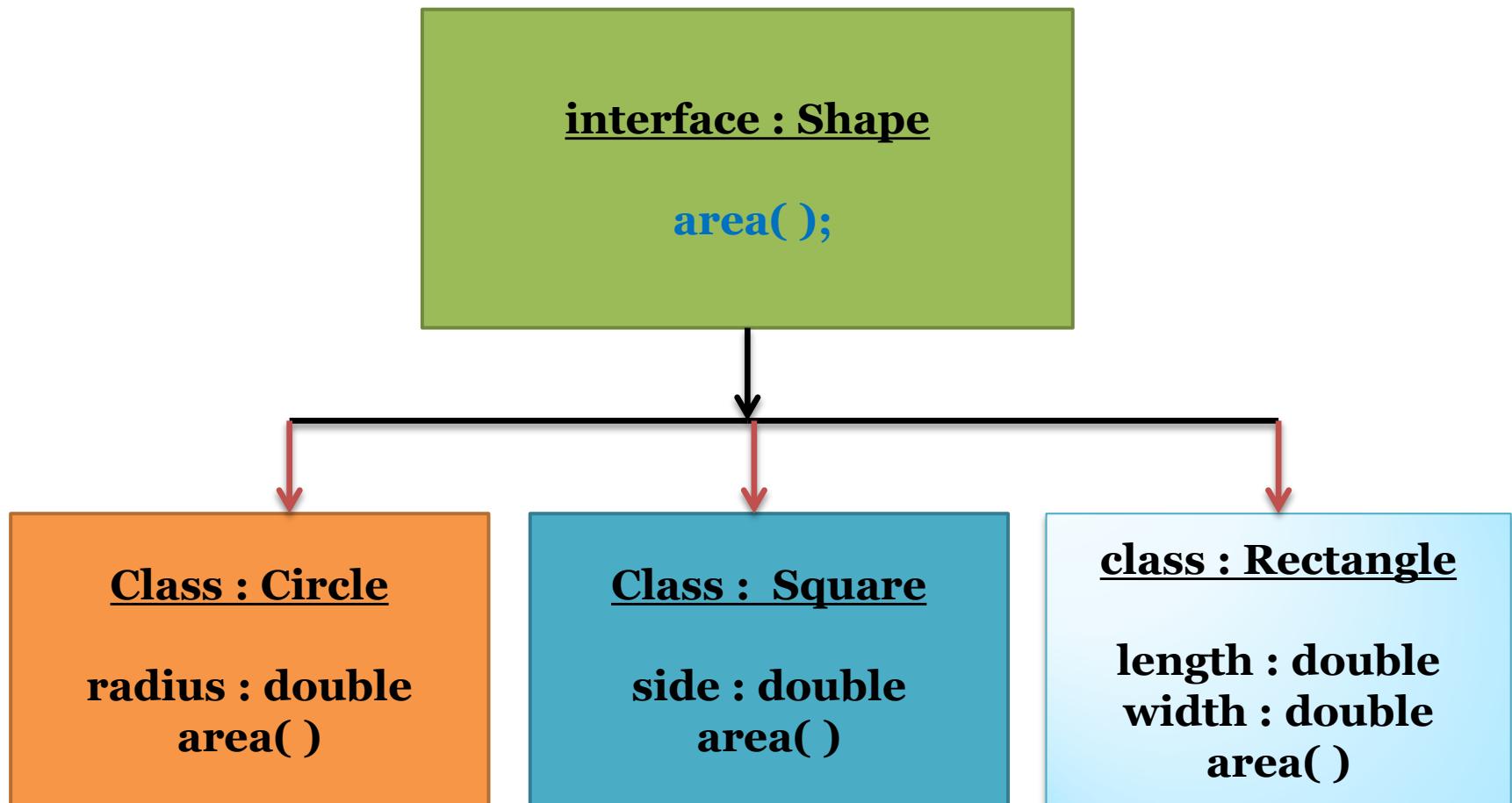


Java Interface Example

```
interface printable{
void print();
}

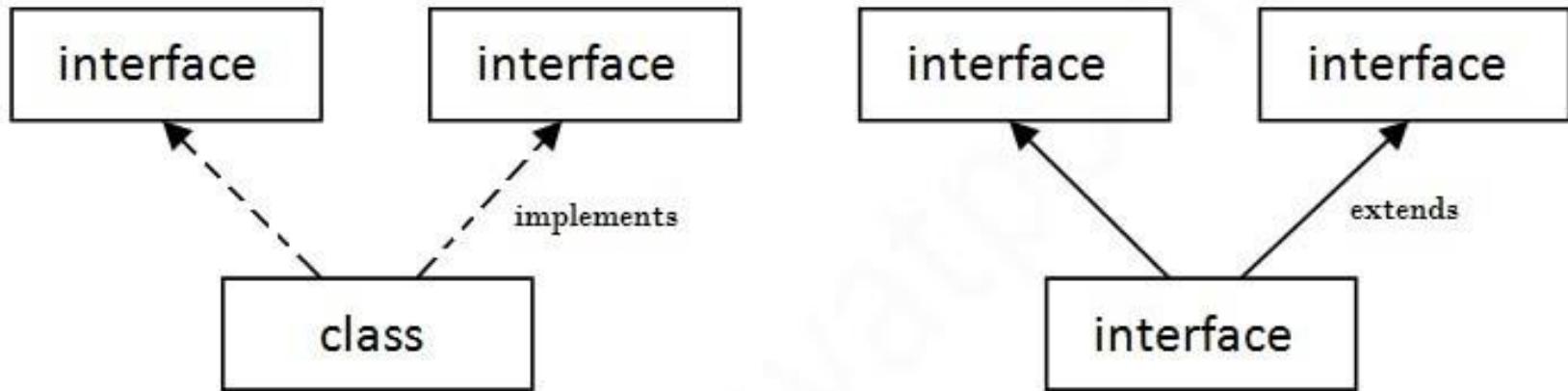
class A6 implements printable{
public void print(){System.out.println("Hello");}
}

public static void main(String args[]){
A6 obj = new A6();
obj.print();
}
}
```



Multiple inheritance in Java by interface

If a class implements multiple interfaces, or an interface extends multiple interfaces, it is known as multiple inheritance.



Multiple Inheritance in Java

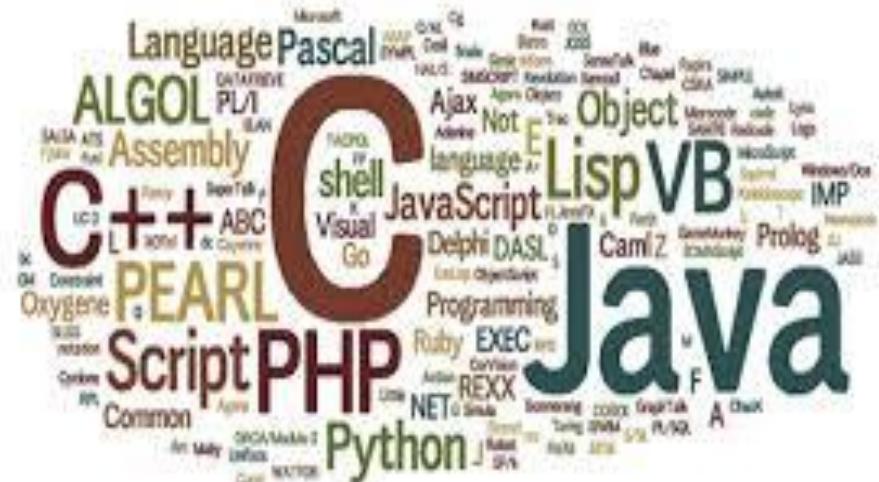
```
interface Printable{
void print();
}

interface Showable{
void print();
}

class Test implements Printable, Showable{
public void print()
{
System.out.println("Hello");
}
public static void main(String args[]){
Test obj = new Test();
obj.print();
} }
```

Presentation on Java Programming (IT207G)

UNIT No. III- 4



Interface in Java

An **interface in Java** is a blueprint of a class. It has static, constant variables and abstract methods.

- The interface in Java is *a mechanism to achieve [abstraction](#)*. There can be only abstract methods in the Java interface, not method body.
- It is used to achieve abstraction and multiple [inheritance in Java](#).
- In other words, you can say that interfaces can have abstract methods and variables. It cannot have a method body.

Why use Java interface?

It is used to achieve abstraction.

1

2

By interface, we can support the functionality of multiple inheritance.

3

It can be used to achieve loose coupling.

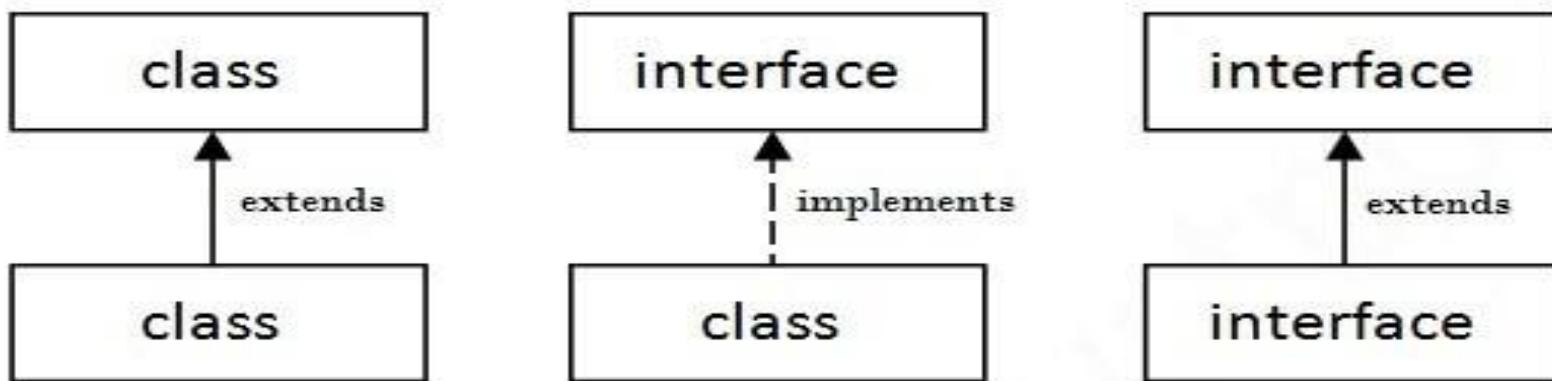
How to declare an interface?

1. An interface is declared by using the interface keyword.
2. It provides total abstraction; means all the methods in an interface are declared with the empty body, and all the fields are public, static and final by default.
3. A class that implements an interface must implement all the methods declared in the interface..

```
interface <interface_name>{  
    // declare constant fields  
    // declare methods that abstract by default.  
}
```

The relationship between classes and interfaces

As shown in the figure given below, a class extends another class, an interface extends another interface, but a **class implements an interface**.



Implementing Interfaces in Java

A class uses the **implements** keyword to implement an interface.

```
interface Animal {  
    public void eat();  
    public void travel();  
}
```

```
public class MammalInt implements Animal {  
  
    public void eat() {  
        System.out.println("Mammal eats");  
    }  
  
    public void travel() {  
        System.out.println("Mammal travels");  
    }  
  
    public static void main(String args[]) {  
        MammalInt m = new MammalInt();  
        m.eat();  
        m.travel(); } }
```

Output:

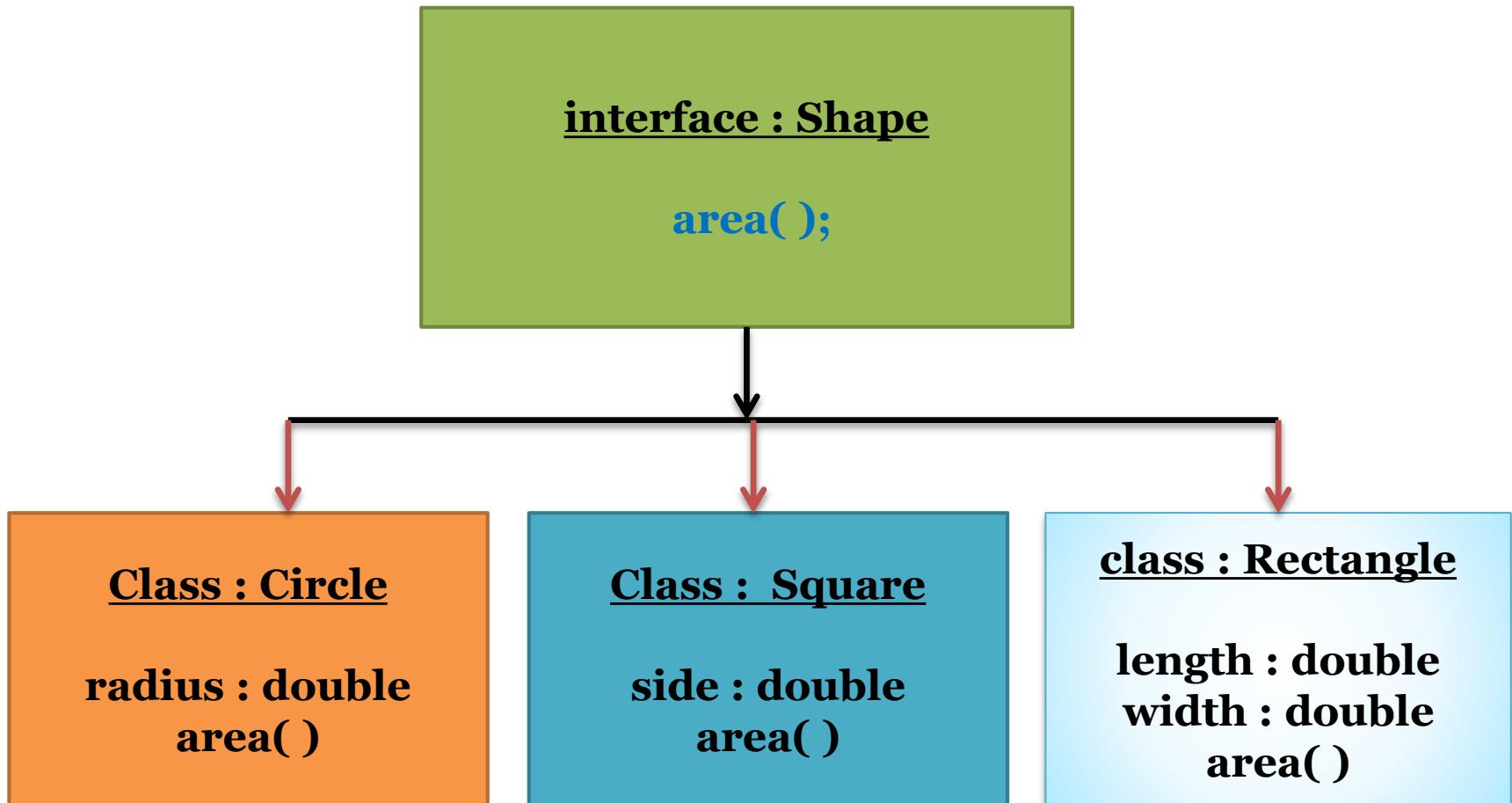
Mammal eats
Mammal travels

Java Interface Example

```
interface printable{
void print();
}

class A6 implements printable{
public void print()
{
System.out.println("Hello");
}

public static void main(String args[]){
A6 obj = new A6();
obj.print();
}
}
```

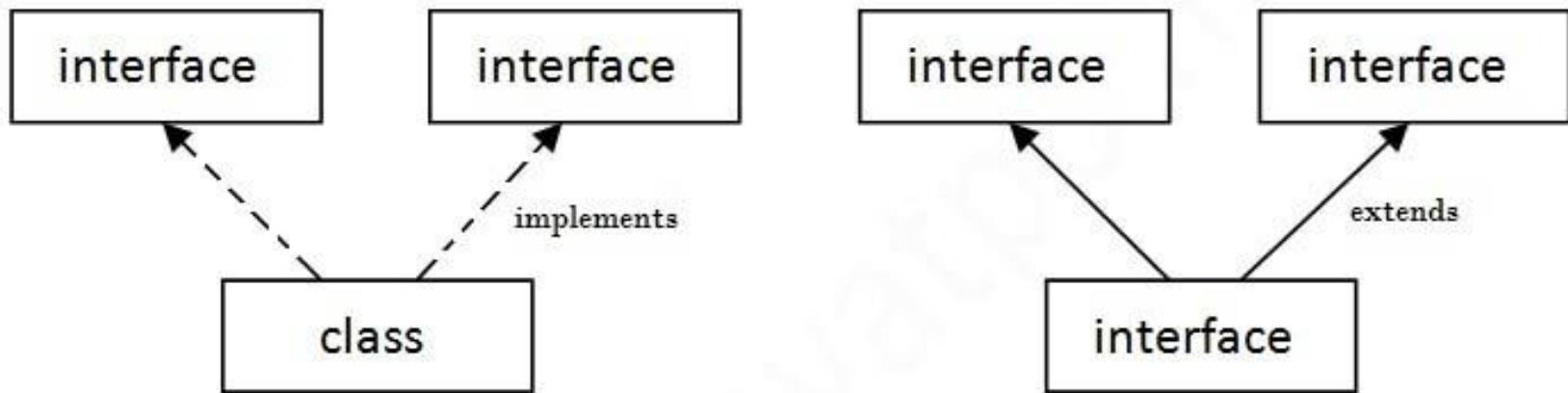


Practical Program:

Create a java program where the **BankAccount** interface defines methods for **deposit, withdraw, and check balance**. The **SavingsAccount** and **CheckingAccount** classes implement this interface and provide their own implementations for these methods. Finally, in the Test class, create instances of both types of accounts and perform various operations such as deposit, withdraw, and checking balance.

Multiple inheritance in Java by interface

If a class implements multiple interfaces, or an interface extends multiple interfaces, it is known as multiple inheritance.



Multiple Inheritance in Java

```
interface Printable{
void print();
}

interface Showable{
void print();
}

class Test implements Printable, Showable{
public void print()
{
System.out.println("Hello");
}
public static void main(String args[]){
Test obj = new Test();
obj.print();
} }
```

```
interface InternetConnectable {  
    void connectToInternet();  
}  
  
interface MediaPlayable {  
    void playMedia();  
}  
  
class Smartphone implements InternetConnectable, MediaPlayable  
{  
    public void connectToInternet() {  
        System.out.println("Connected to the internet");  
    }  
    public void playMedia() {  
        System.out.println("Playing media");  
    }  
}
```

```
public class Test {  
    public static void main(String[] args) {  
        Smartphone phone = new Smartphone();  
        phone.connectToInternet();  
        phone.playMedia();  
    } } 
```

Java - Packages

A **Java package** can be defined as a grouping of related types ([classes](#), [interfaces](#), [enumerations](#), and annotations) providing access protection and namespace management.

Types of Java Packages

1. Built-in Java Packages
2. User-defined Java Packages

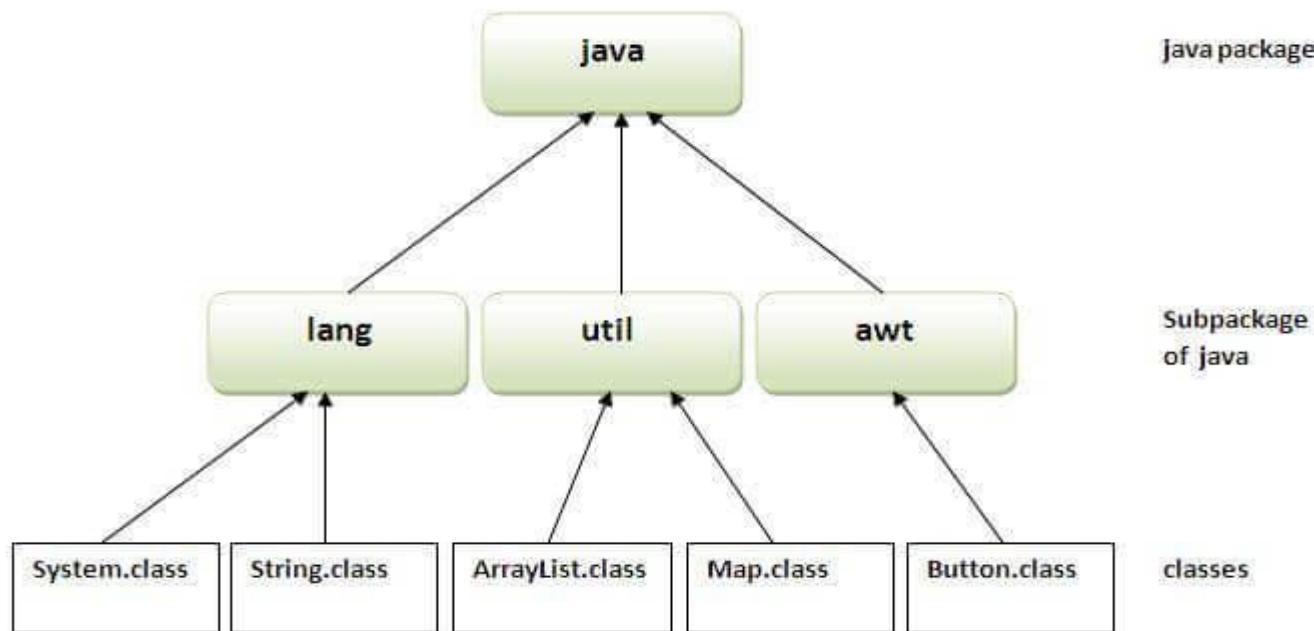
Some of the existing packages in Java are –

java.lang – bundles the fundamental classes

java.io – classes for input , output functions are bundled in this package

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.



User-defined Java Packages

- You can define your own packages to bundle groups of classes/interfaces, etc.
- While creating a package, you should choose a name for the package and include a **package** statement along with that name at the top of program.

package geometry;

- To compile the Java programs with package statements, you have to use -d option as shown below.

```
javac -d . file_name.java
```

How to access package from another package?

1. import package.*;

Example: import java.io.*;

2. import package.classname;

Example: import java.io.Scanner;

3. fully qualified name:

- It is generally used when two packages have same class name e.g. java.util and java.sql packages contain Date class.
- To access a package using its fully qualified name, you prepend the package name to the class name when importing or referencing the class.
- Example: sql.Date.showDate(); util.Date.setDate()

Java Package Example

```
package geometry;
```

```
public class Rectangle {  
    private double length;  
    private double width;  
  
    public Rectangle(double length, double width) {  
        this.length = length;  
        this.width = width;  
    }  
    public double calculateArea() {  
        return length * width;  
    }  
    public double calculatePerimeter() {  
        return 2 * (length + width);  
    }  
}
```

```
import geometry.Rectangle;  
  
public class Test {  
    public static void main(String[] args) {  
        Rectangle rectangle = new Rectangle(5, 3);  
  
        // Calculate and display area and perimeter for rectangle  
        System.out.println("Rectangle:");  
        System.out.println("Area: " + rectangle.calculateArea());  
        System.out.println("Perimeter: " + rectangle.calculatePerimeter());  
    } }
```

```
> javac -d . Rectangle.java  
> javac Test.java  
> java Test
```

OUTPUT:
Rectangle:
Area: 15.0
Perimeter: 16.0

Command Prompt

Microsoft Windows [Version 10.0.19045.4046]
(c) Microsoft Corporation. All rights reserved.

C:\Users\!TDEPT>cd\

C:\>cd C:\Users\!TDEPT\Desktop\Java AIML\ABC1

C:\Users\!TDEPT\Desktop\Java AIML\ABC1>javac -d . Rectangle.java

C:\Users\!TDEPT\Desktop\Java AIML\ABC1>javac Test.java

C:\Users\!TDEPT\Desktop\Java AIML\ABC1>java Test

Rectangle:

Area: 15.0

Perimeter: 16.0

C:\Users\!TDEPT\Desktop\Java AIML\ABC1>

Built-in Packages

These packages consist of a large number of classes which are a part of **Java API**. Some of the commonly used built-in packages are:

- 1) **java.lang**: Contains language support classes(e.g classed which defines primitive data types, math operations). This package is automatically imported.
- 2) **java.io**: Contains classed for supporting input / output operations.
- 3) **java.util**: Contains utility classes which implement data structures like Linked List, Dictionary and support ; for Date / Time operations.
- 4) **java.applet**: Contains classes for creating Applets.
- 5) **java.awt**: Contain classes for implementing the components for graphical user interfaces (like button , ;menus etc).
- 6) **java.net**: Contain classes for supporting networking operations.

Package: mathoperations

Class: MathUtil

```
int add( a,b);  
int sub(a,b);  
int mul(a,b);  
float div(a,b);
```



Class : Test

This class import
mathoperations package
and display result of all
methods of MathUtil class

Package: banking

Class: BankAccount

```
String : accountNumber  
Double : balance  
BankAccount()  
deposit(amount);  
withdraw(amount);  
checkBalance();
```



Class : Test

This class import banking package and display result of all methods of BankAccount class

Presentation on Java Programming (IT207G)

UNIT No. IV- 1



Java - Exceptions

- An exception (or exceptional event) is a problem that arises during the execution of a program.
- When an **Exception** occurs the normal flow of the program is disrupted and the program/Application terminates abnormally.
- The **Exception Handling in Java** is one of the powerful *mechanism to handle the runtime errors* so that the normal flow of the application can be maintained.

Why Exception Occurs?

- ✓ A user has entered an invalid data.
- ✓ A file that needs to be opened cannot be found.
- ✓ A network connection has been lost in the middle of communications or the JVM has run out of memory.

Common Scenarios of Java Exceptions

- 1) A scenario where ArithmeticException occurs If we divide any number by zero, there occurs an ArithmeticException.

```
int a=50/0;//ArithmeticException
```

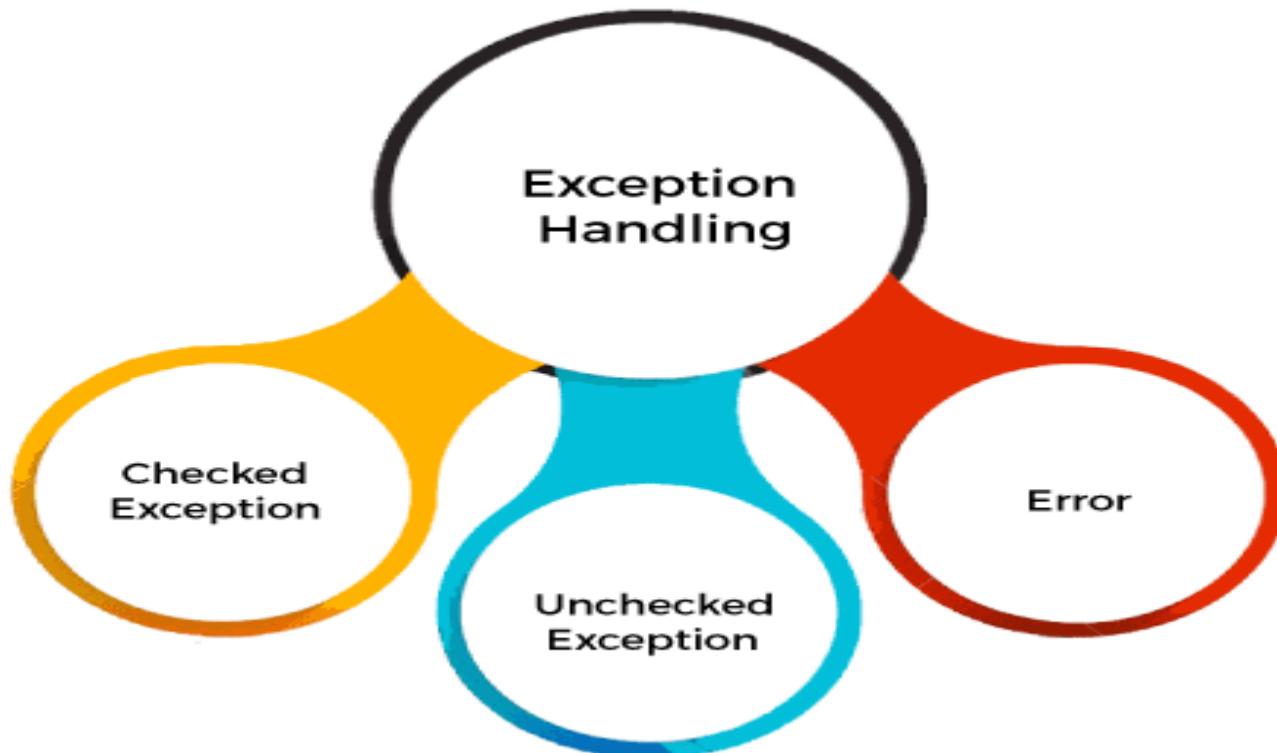
- 2) A scenario where NullPointerException occurs If we have a null value in any variable, performing any operation on the variable throws a NullPointerException.

```
String s=null;  
System.out.println(s.length());//NullPointerException
```

- 3) A scenario where ArrayIndexOutOfBoundsException occurs When an array exceeds to it's size, the ArrayIndexOutOfBoundsException occurs. there may be other reasons to occur ArrayIndexOutOfBoundsException.

```
int a[]={};  
a[10]=50; //ArrayIndexOutOfBoundsException
```

Types of Java Exceptions



1) Checked Exception

A checked exception is an exception that is checked (notified) by the compiler at compilation-time, these are also called as compile time exceptions.

For example, IOException, SQLException, etc. Checked exceptions are checked at compile-time.

```
import java.io.File;  
import java.io.FileReader;
```

```
C:\>javac FilenotFound_Demo.java  
FilenotFound_Demo.java:8: error: unreported exception FileNotFoundException  
        FileReader fr = new FileReader(file);  
                           ^  
1 error
```

2) Java Unchecked Exceptions

An unchecked exception is an exception that occurs at the time of execution. These are also called as Runtime Exceptions. These include programming bugs, such as logic errors or improper use of an API.

```
public class Unchecked_Demo {  
    public static void main(String args[])  
    {  
        int num[] = {1, 2, 3, 4};  
        System.out.println(num[5]);  
    } }
```

Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: 5
at Exceptions.Unchecked_Demo.main(Unchecked_Demo.java:8)

Exception Handling in Java

Java provides five keywords that are used to handle the exception.

1. try	The "try" keyword is used to specify a block where we should place an exception code. It means we can't use try block alone. The try block must be followed by either catch or finally.
2. catch	The "catch" block is used to handle the exception. It must be preceded by try block which means we can't use catch block alone.

Syntax:

```
try {  
    // Protected code  
} catch (ExceptionName e1) {  
    // Catch block  
}
```

```
public class JavaExceptionExample{
    public static void main(String args[]){
        try{
            //code that may raise exception
            int data=100/0;
        }
        catch(ArithmetcException e)
        {
            System.out.println(e);
        }
        //rest code of the program
        System.out.println("rest of the code... ");
    } }
```

```
Exception in thread main java.lang.ArithmetcException:/ by zero
rest of the code...
```

Java Multi-catch block

A try block can be followed by one or more catch blocks. Each catch block must contain a different exception handler.

```
public class Main {  
    public static void main(String[] args) {  
        try {  
            int result = 10 / 0;  
            int[] arr = {1, 2, 3};  
            int element = arr[5];  
        } catch (ArithmaticException e) {  
            System.out.println("Arithmatic Exception caught: ");  
        } catch (ArrayIndexOutOfBoundsException e) {  
            System.out.println("ArrayIndexOutOfBoundsException Exception caught: ");  
        } catch (Exception e) {  
            System.out.println("Exception caught: " + e.getMessage());  
        }  
    } }
```

Nested try Statement

In Java, using a try block inside another try block is permitted. It is called as nested try block. Every statement that we enter a statement in try block, context of that exception is pushed onto the stack.

```
try
{
    statement 1;
    statement 2;
//try catch block within another try block
```

```
try
{
    statement 3;
    statement 4;
//try catch block within nested try block
try
{
    statement 5;
    statement 6;
}
```

```
    catch(Exception e2)
    {
        //exception message
    }

}
catch(Exception e1)
{
    //exception message
}

}
//catch block of parent (outer) try block
catch(Exception e3)
{
    //exception message
}
....
```

```
public class NestedTryBlock{
    public static void main(String args[]){
        //outer try block
        try{
            try{
                System.out.println("going to divide by 0");
                int b =39/0;
            }
            catch(ArithmeticException e)
            {
                System.out.println(e);
            }
            try{
                int a[] =new int[5];
                a[5]=4;
            }
            catch(ArrayIndexOutOfBoundsException e)
            {
                System.out.println(e);
            }
            System.out.println("other statement");
        }
    }
}
```

```
catch(Exception e)
{
    System.out.println("handled the exception (outer catch)");
}

System.out.println("normal flow..");
}
}
```

throws keyword in Exception Handling

- ✓ In Java, the throws keyword is used in method declarations to indicate that the method may throw one or more types of exceptions during its execution.
- ✓ When a method uses throws to declare exceptions, it's essentially stating that it does not handle those exceptions internally and that calling methods or the caller of the method should handle them instead.

Here's how the throws keyword is used::

```
public void someMethod() throws SomeExceptionType {  
    // Method code that may throw SomeExceptionType  
}
```



Multiple exceptions can be declared using **throws** by separating them with commas:

```
public void someMethod() throws IOException, SQLException {  
    // Method code that may throw IOException or SQLException  
}
```

- ✓ Using throws can be helpful in documenting the exceptions that a method might throw, making it clear to other developers what exceptions need to be handled when using that method.
- ✓ However, it also means that callers of the method need to be aware of and handle these exceptions appropriately.

```
import java.io.FileNotFoundException;
import java.io.FileReader;
public class ThrowsExample {
    public static void readFile(String filename) throws FileNotFoundException
    {
        FileReader fileReader = new FileReader(filename);
    }
    public static void main(String[] args)
    {
        try
        {
            readFile("example.txt");
        }
        catch (FileNotFoundException e)
        {
            System.err.println("File not found: " + e.getMessage());
        }
    }
}
```

throw keyword in Exception Handling

- ✓ In Java, the throw keyword is used to explicitly throw an exception within a program.
- ✓ It allows you to create and throw instances of Throwable subclasses, such as Exception or Error, or custom exception classes that you've defined.

Here's how the throw keyword is used::

```
throw new SomeExceptionType("Error message");
```

After the **throw** statement is executed, the flow of control immediately transfers to the nearest enclosing **catch** block capable of handling the thrown exception.

```
public class ThrowExample {  
    public static void main(String[] args) {  
        try {  
            validateAge(15);  
        } catch (IllegalArgumentException e) {  
            System.err.println("Error: " + e.getMessage());  
        }  
  
        public static void validateAge(int age) {  
            if (age < 18) {  
                throw new IllegalArgumentException("Age must be 18 or above");  
            }  
            System.out.println("Age is valid");  
        }  
    }  
  
System Define Exception : IllegalArgumentException
```

```
class InvalidAgeException extends Exception {  
    public InvalidAgeException() {  
        System.out.println("Age must be 18 or above");    }  
}  
  
public class UserDefinedExceptionExample {  
    public static void main(String[] args) {  
        try {  
            registerUser("John", 15);  
        } catch (InvalidAgeException e)  
        {  
            System.err.println("Error: " + e.getMessage());  
        } //main close  
  
        public static void registerUser(String name, int age) throws InvalidAgeException {  
            if (age < 18) {  
                throw new InvalidAgeException();  
            }  
            System.out.println("User " + name + " registered successfully");  
        }  
    }  
}
```

User Define Exception Program: InvalidAgeException

finally block in Exception Handling

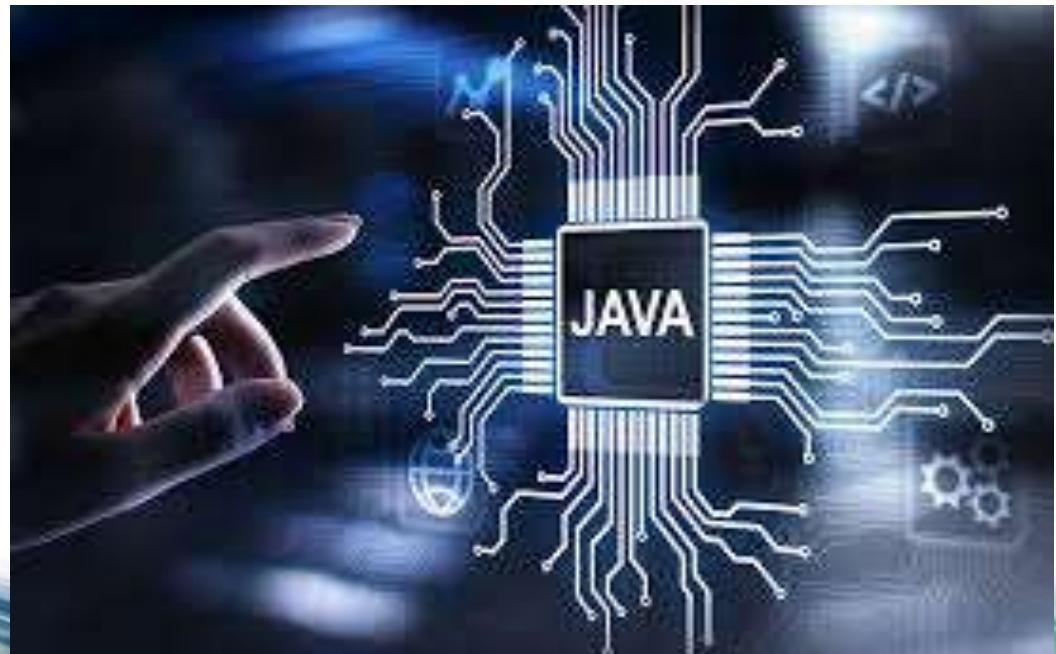
- ✓ In Java exception handling, the finally block is used to define a block of code that is always executed, regardless of whether an exception is thrown or not, and regardless of whether an exception is caught or not.
- ✓ The finally block is typically used to perform cleanup tasks such as closing files or releasing resources, ensuring that these tasks are executed regardless of the outcome of the code block.

Here's how the finally block is used::

```
try {
    // Code that may throw an exception
} catch (SomeException e) {
    // Exception handling code
} finally {
    // Code that is always executed, regardless of
}
```

Presentation on Java Programming (IT207G)

UNIT No. IV- 2



Multithreading in Java

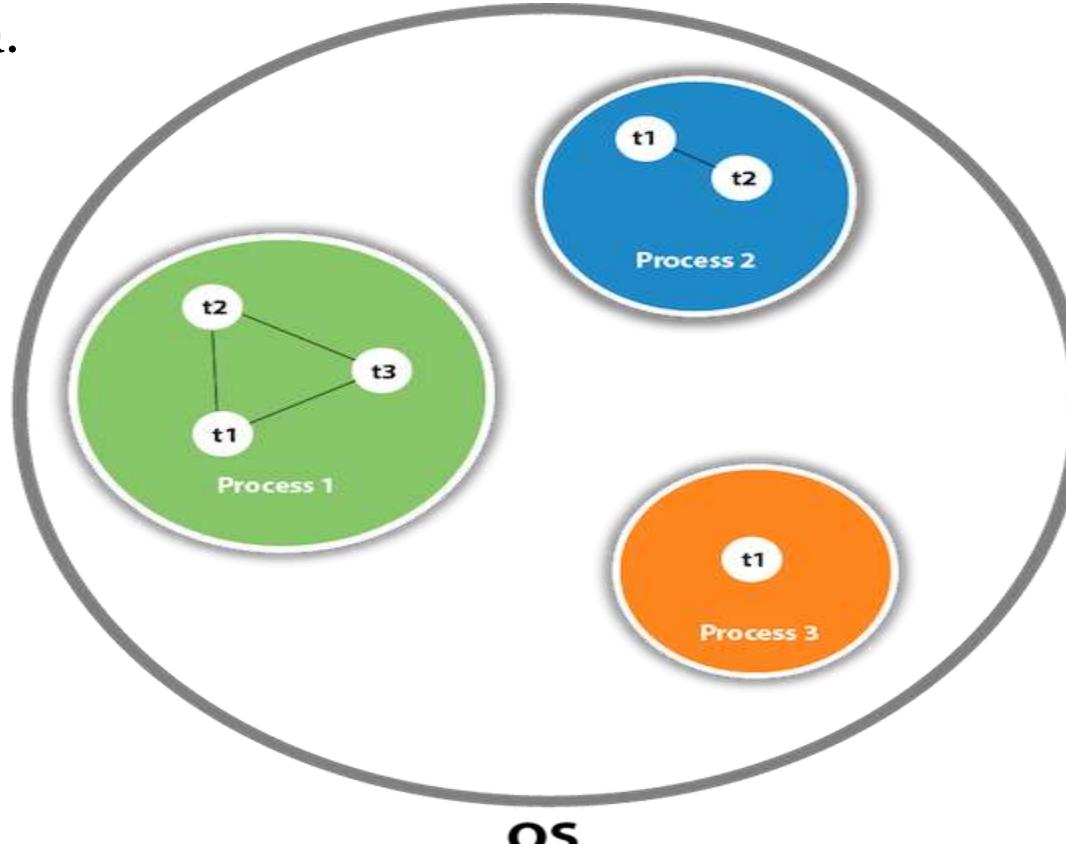
- **Multithreading in Java** is a process of executing multiple threads simultaneously.
- A thread is a **lightweight sub-process**, the **smallest unit** of processing. Multiprocessing and multithreading, both are used to achieve multitasking.
- However, we use multithreading than multiprocessing because threads use a shared memory area. They don't allocate separate memory area so saves memory, and context-switching between the threads takes less time than process.

Advantages of Java Multithreading:

- 1) It **doesn't block the user** because threads are independent and you can perform multiple operations at the same time.
- 2) Threads are **independent**, so it doesn't affect other threads if an exception occurs in a single thread.

What is Thread in java

- A thread is a lightweight subprocess, the smallest unit of processing. It is a separate path of execution.
- Threads are independent. If there occurs exception in one thread, it doesn't affect other threads. It uses a shared memory area.



Java Thread class

- Java provides **Thread class** to achieve thread programming. Thread class provides constructors and methods to create and perform operations on a thread.

S. N.	Modifier and Type	Method	Description
1)	void	<u>start()</u>	It is used to start the execution of the thread.
2)	void	<u>run()</u>	It is used to do an action for a thread.
3)	static void	<u>sleep()</u>	It sleeps a thread for the specified amount of time.
4)	static Thread	<u>currentThread()</u>	It returns a reference to the currently executing thread object.
5)	void	<u>join()</u>	It waits for a thread to die.
6)	int	<u>getPriority()</u>	It returns the priority of the thread.
7)	void	<u>setPriority()</u>	It changes the priority of the thread.
8)	String	<u>getName()</u>	It returns the name of the thread.
9)	void	<u>setName()</u>	It changes the name of the thread.

Thread Life Cycle

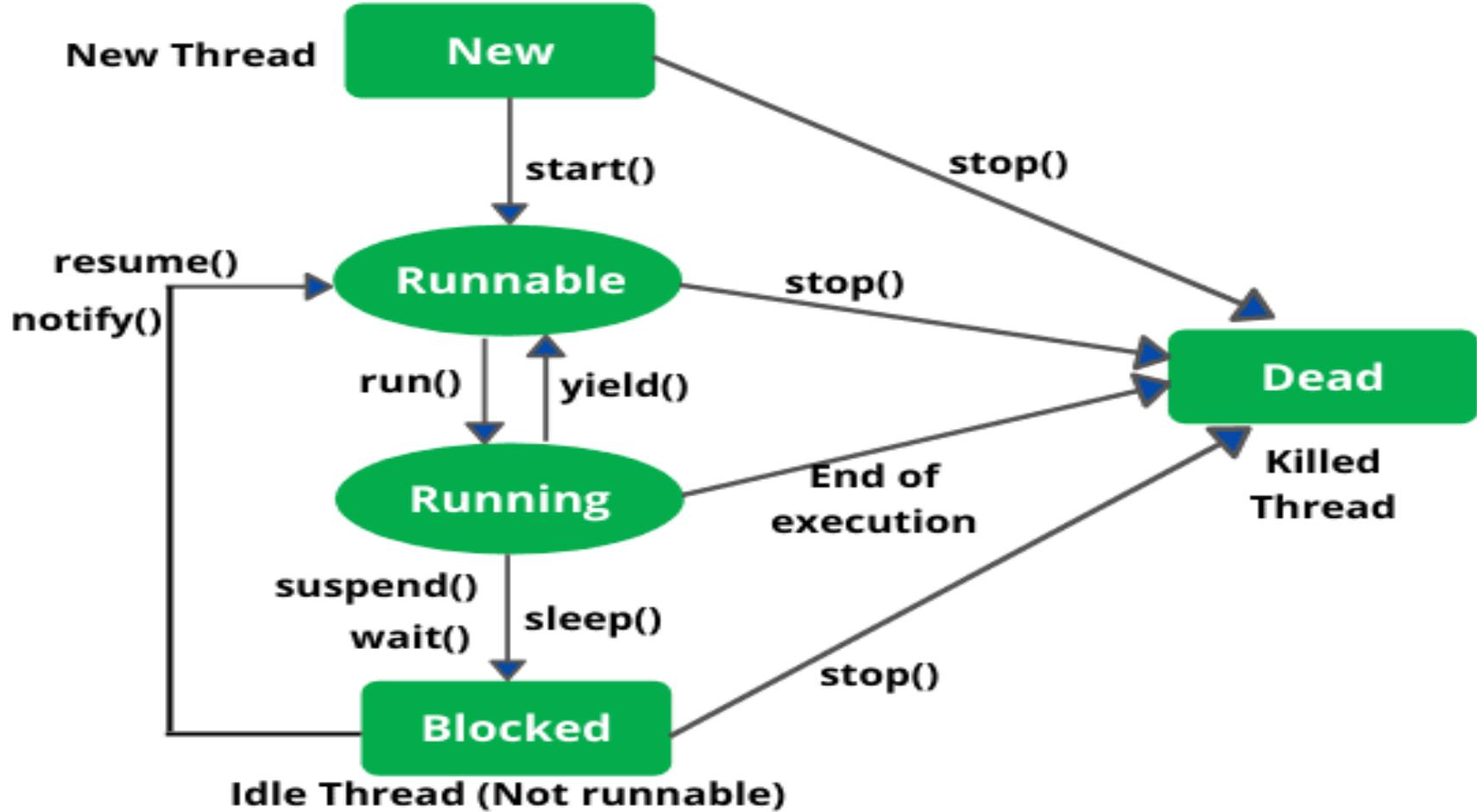


Fig: State Transition Diagram of a Thread

Following are the stages of the life cycle –

New – A new thread begins its life cycle in the new state. It remains in this state until the program starts the thread.

Runnable – After a newly born thread is started, the thread becomes runnable. A thread in this state is considered to be executing its task.

Waiting – Sometimes, a thread transitions to the waiting state while the thread waits for another thread to perform a task. A thread transitions back to the runnable state only when another thread signals the waiting thread to continue executing.

Timed Waiting – A runnable thread can enter the timed waiting state for a specified interval of time..

Terminated (Dead) – A runnable thread enters the terminated state when it completes its task or otherwise terminates.

Java Threads

There are two ways to create a thread:

1. By extending Thread class
2. By implementing Runnable interface.

1. By extending Thread class

Step 1: Create subclass of Thread class.

Step 2: Override run() Method: You will need to override **run()** and you will put your complete business logic inside this method.

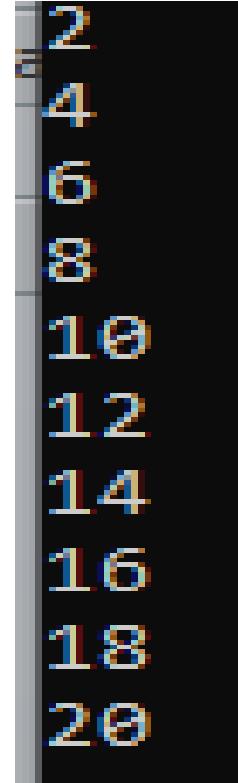
Syntax : public void run()

Step 3: Create object of class in main().

Step 4: Call Thread using start() Method: Once Thread object is created, you can start it by calling **start()** method, which executes a call to **run()** method.

Syntax: void start();

```
class EvenNumber extends Thread {  
  
    public void run() {  
        for (int i = 2; i <= 20; i += 2) {  
            System.out.println(i);  
        }  
    }  
    try {  
        Thread.sleep(500); // Delay for 5 second  
    }  
    catch(Exception e){  
        System.out.println(e.getMessage());  
    }  
}  
}  
}  
public static void main(String[] args) {  
    EvenNumber thread1 = new EvenNumber();  
    thread1.start();  
}  
}
```



2
4
6
8
10
12
14
16
18
20

```
class EvenNumber extends Thread {  
    public void run() {  
        for (int i = 2; i <= 20; i += 2) {  
            System.out.println(i);  
        }  
    }  
}  
try  
{    Thread.sleep(500);  
}  
catch(Exception e){ }  
}  
}// class close  
  
class OddNumber extends Thread {  
    public void run() {  
        for (int i = 1; i <= 20; i += 2) {  
            System.out.println(i);  
        }  
    }  
}  
try  
{    Thread.sleep(500);  
}  
catch(Exception e){ }  
}  
}// class close
```

```
class TestNumber  
{  
    public static void main(String[] args)  
    {  
        EvenNumber thread1 = new EvenNumber();  
        thread1.start();  
        OddNumber thread2=new OddNumber();  
        thread2.start();  
    }  
}
```

Thread Priorities

- Every Java thread has a priority that helps the operating system determine the order in which threads are scheduled.
- Java thread priorities are in the range between **MIN_PRIORITY** (a constant of 1) and **MAX_PRIORITY** (a constant of 10). By default, every thread is given priority **NORM_PRIORITY** (a constant of 5).
- Threads with higher priority are more important to a program and should be allocated processor time before lower-priority threads.

Setter & Getter Method of Thread Priority

- **public final int getPriority():** The java.lang.Thread.getPriority() method returns the priority of the given thread.
- **public final void setPriority(int newPriority):** The java.lang.Thread.setPriority() 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).

```
class EvenThread extends Thread {  
    public void run() {  
        for (int i = 2; i <= 10; i += 2) {  
            System.out.println("Even Thread: " + i);  
        }    }    }
```

```
class OddThread extends Thread {  
    public void run() {  
        for (int i = 1; i <= 10; i += 2) {  
            System.out.println("Odd Thread: " + i);  
        }    }    }
```

```
public class Main {  
    public static void main(String args[]) {  
        EvenThread evenThread = new EvenThread();  
        OddThread oddThread = new OddThread();  
evenThread.setPriority(Thread.MIN_PRIORITY);  
oddThread.setPriority(Thread.MAX_PRIORITY);  
        evenThread.start();  
        oddThread.start();  
    }    }
```

OUTPUT:

Odd Thread: 1
Even Thread: 2
Odd Thread: 3
Even Thread: 4
Odd Thread: 5
Even Thread: 6
Odd Thread: 7
Even Thread: 8
Odd Thread: 9
Even Thread: 10

create a thread by implementing the Runnable interface

- 1. Create a Runnable implementation:** First, you need to create a class that implements the Runnable interface. This interface has a single method run() that you need to override.
- 2. Instantiate the Thread object:** After implementing the Runnable interface, you need to create an instance of the Thread class and pass your Runnable object to its constructor.
- 3. Start the thread:** Finally, you can start the thread by calling the start() method on the Thread object.

```
class MyRunnable implements Runnable {  
    public void run() {  
        System.out.println("MyRunnable is running");  
    }  
}  
  
public class Main {  
    public static void main(String[] args) {  
        // Create an instance of MyRunnable  
        MyRunnable myRunnable = new MyRunnable();  
  
        // Create a Thread object and pass the MyRunnable instance to its constructor  
        Thread thread = new Thread(myRunnable);  
  
        // Start the thread  
        thread.start();  
    }  
}
```

Practical's

1. Develop a basic calculator application where arithmetic operations (addition, subtraction, multiplication, division) are performed concurrently by separate threads.
2. Write a program to create 3 threads in which one will display 1 to 50 numbers, second will display only even numbers from 1 to 50 and third will display only odd numbers from 1 to 50. Also set the different priorities to three threads and give delay of 1 second while displaying the output.

isAlive() and join() for inter Thread communication

- The **isAlive()** method in Java is used to determine whether a thread is currently alive or not.
- A thread is considered alive if it has been started and has not yet completed its execution (i.e., it has not terminated).

Here's the typical use of the **isAlive()** method:

- **Thread Monitoring:** You can use isAlive() to check the status of a thread and take appropriate actions based on its state.
- **Thread Termination Handling:** You can use isAlive() to check whether the thread has completed its execution before accessing its results or performing cleanup operations.

```
public class ThreadExample extends Thread {  
    public void run() {  
        try {  
            Thread.sleep(1000);  
        } catch (InterruptedException e) {  
            e.printStackTrace();  
        }  
    }  
  
    public static void main(String[] args) {  
        ThreadExample thread = new ThreadExample();  
        thread.start();  
  
        // Check if the thread is alive  
        if (thread.isAlive()) {  
            System.out.println("Thread is still alive.");  
        } else {  
            System.out.println("Thread has terminated.");  
        }  
    }  
}
```

OUTPUT: Thread is still alive.

In Java, the **join()** method is used to wait for a thread to finish its execution before proceeding with the execution of the current thread. It allows one thread to wait for another thread to complete.

- 1. Waiting for Thread Completion:** The primary use of the join() method is to ensure that a thread completes its execution before proceeding further.
- 2. Synchronization:** join() is often used for synchronization purposes. It allows threads to coordinate their execution and synchronize the flow of control.
- 3. Main Thread Coordination:** By calling join() on each thread that the main thread wants to wait for, it can ensure that all threads have finished before the program exits.

```
class MyThread1 extends Thread {  
    public void run()  
    {  
        System.out.println("Thread 1 started");  
        try {  
            Thread.sleep(2000);  
        } catch (InterruptedException e) {  
            e.printStackTrace();  
        }  
        System.out.println("Thread 1 finished");  
    } }
```

```
class MyThread2 extends Thread {  
    public void run()  
    {  
        System.out.println("Thread 2 started");  
        try {  
            Thread.sleep(3000);  
        } catch (InterruptedException e) {  
            e.printStackTrace();  
        }  
        System.out.println("Thread 2 finished");  
    } }
```

```
public class JoinExample {  
    public static void main(String[] args) throws InterruptedException {  
        MyThread1 thread1 = new MyThread1();  
        MyThread2 thread2 = new MyThread2();  
        thread1.start();  
        thread2.start();  
        System.out.println("Main thread waiting for threads to finish");  
        thread1.join();  
        thread2.join();  
        System.out.println("All threads have finished");  
    } }
```

OUTPUT:

Thread 1 started

Thread 2 started

Main thread waiting for threads to finish

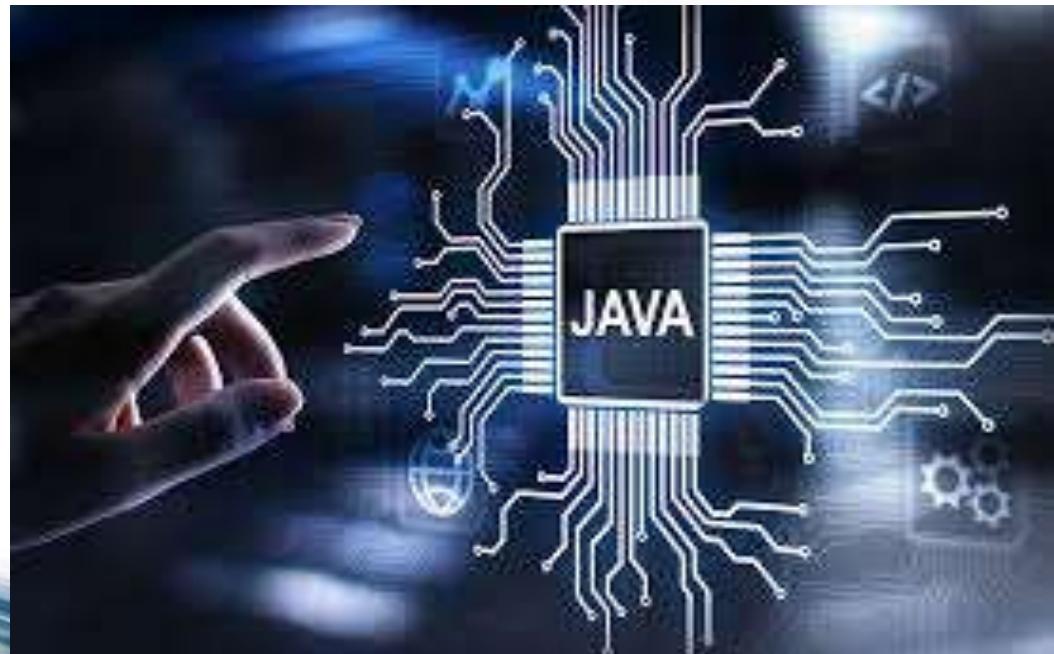
Thread 1 finished

Thread 2 finished

All threads have finished

Presentation on Java Programming (IT207G)

UNIT No. V- 1



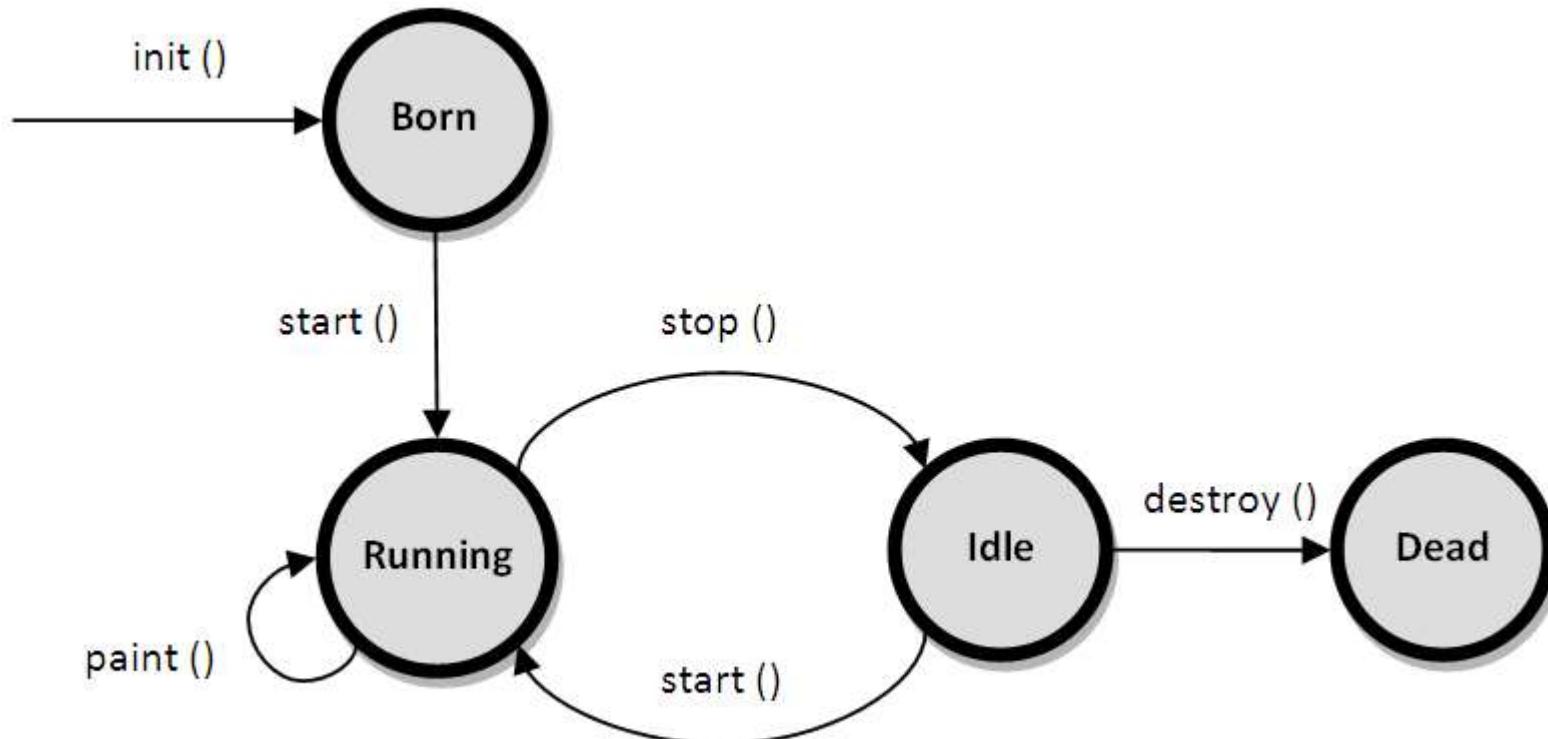
Java Applet

- An **applet** is a Java program that runs in a Web browser. An applet can be a fully functional Java application because it has the entire Java API at its disposal..
- An applet is a special kind of Java program that **runs in a Java enabled 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.
- To create an applet, a class must extend **java.applet.Applet** class.

Difference between Java Applet and Application

Parameters	Java Application	Java Applet
Meaning	A Java Application also known as application program that independently executes on the computer.	The Java applet works on the client side, and runs on the web browser.
Requirement of main() method	Its execution starts with the main() method only.	It does not require the use of any main() method. Java applet initializes through init() method.
Installation	We need to install the Java application first and obviously on the local computer.	Java applet does not need to be pre-installed.
Operation	It performs read and write tasks on a variety of files located on a local computer.	It cannot run the applications on any local computer.
Security	Java applications are pretty trusted, and thus, come with no security concerns.	Java applets are less reliable. So, they need to be safe.

Applet Life Cycle



Applet Life Cycle Methods

- Applet life cycle methods control the life of applet and they called automatically. It has 4 life cycle methods and java.awt.Component class provides 1 life cycle methods for an applet.
1. **public void init():** This method is called when the applet is first initialized. It is typically used for applet initialization tasks such as initializing the applet.
 2. **public void start():** This method is called when the applet is about to start running.
 3. **public void stop():** This method is called when the applet is about to stop running. It is called when the user navigates away from the page.
 4. **public void destroy():** This method is called when the applet is being destroyed.
 5. **public void paint(Graphics g):** This method is called whenever the applet needs to be redrawn, such as when it is first displayed

How to run an Applet?

There are two ways to run an applet

1. By html file on Web Browser
2. By appletViewer tool (for testing purpose) On command prompt

For Example:

c:\>javac First.java

c:\>appletviewer my.html

Basic guide to creating an applet program in Java

- 1. Create a Java class:** Start by creating a new Java class that extends the java.applet.Applet class.
- 2. Override the necessary methods:** In your applet class, you'll need to override the init(), start(), stop(), and paint() methods.
- 3. Implement the applet's functionality:** Inside the init() method, you can set up any initialization code for your applet. In the paint() method, you'll write code to draw graphics or display content on the applet's surface.
- 4. Create an HTML File:** To run your applet in a appletviewer, you'll need to create an <applet> tag with the necessary attributes to specify the applet's class name and size in html separate file.
- 5. Compile and run code:** Once you've written your applet class, compile it using the Java compiler (javac) and run (appletviewer)

<applet> Tag

- The HTML file need to create for writing <applet> tag
- This way, you have two files that is HTML file and Java source file.

```
<html>
<body>
<applet code = "HelloApplet.class" width = "320"
height = "120">

</applet>
</body>
</html>
```

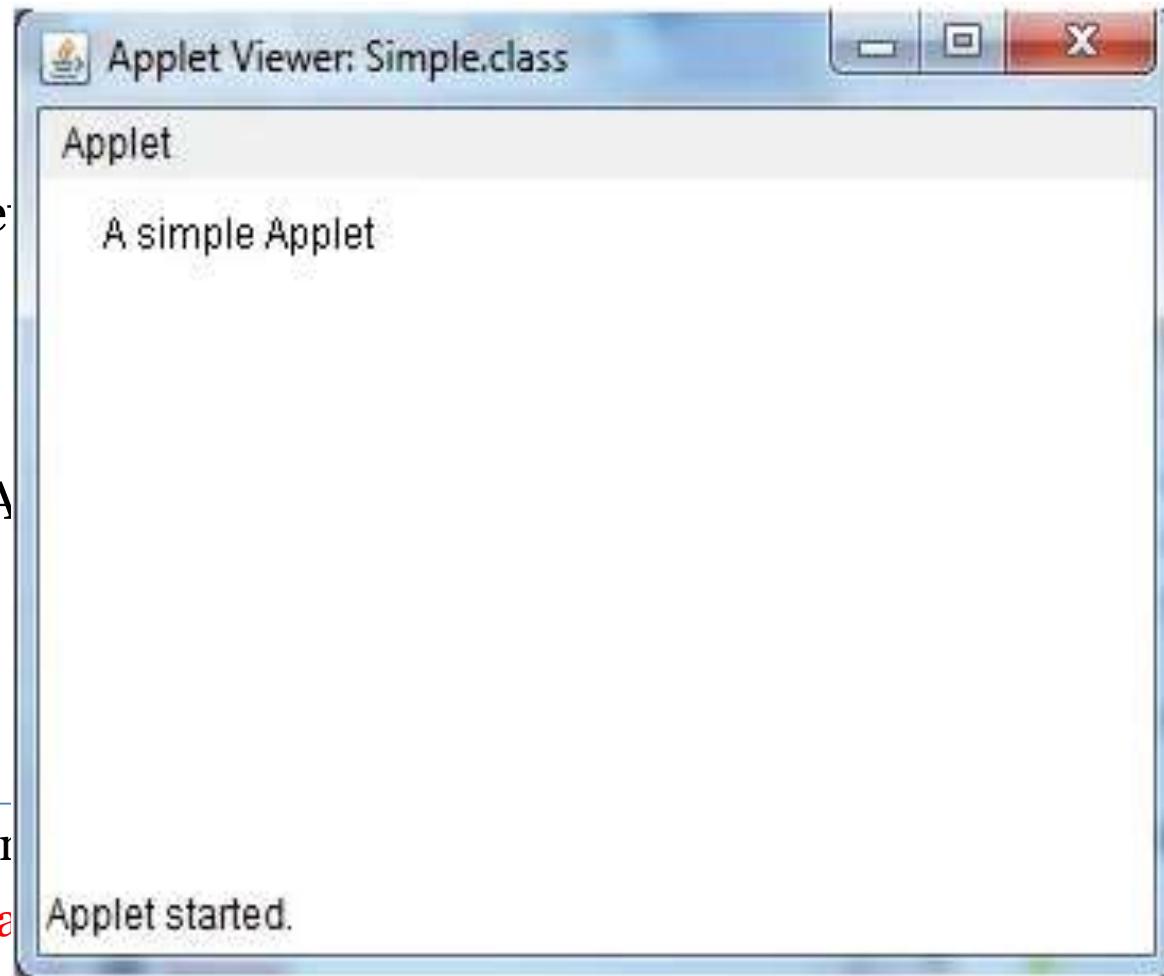
- The code attribute of the <applet> tag is required. It specifies the Applet class to run.
- Width and height are also required to specify the initial size of the panel in which an applet runs.

Simple Applet program

```
import java.applet.Applet;  
import java.awt.Graphics;  
  
public class MyApplet extends Applet  
{  
    public void paint(Graphics g)  
    {  
        g.drawString("A Simple Applet", 20, 20);  
    }  
}
```

Simple.html

```
<html>
<head>
    <title>Status Applet</title>
</head>
<body>
    <applet code="MyA...
</applet>
</body>
</html>
```



Command

c:\>java -jar appletviewer Simple.html

c:\>appletviewer Simple.html

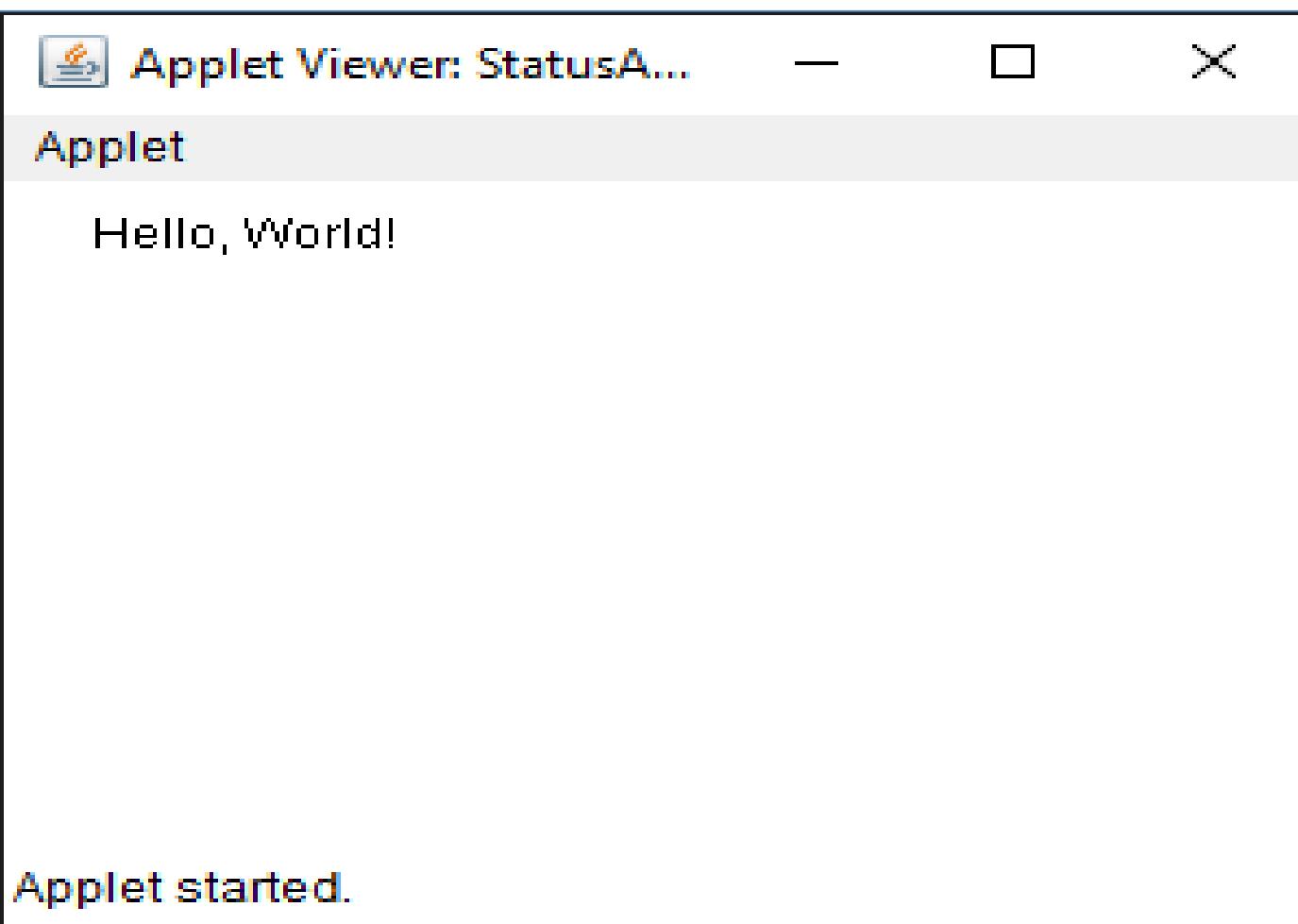
```
import java.applet.Applet;
import java.awt.Graphics;

public class StatusApplet extends Applet {
    public void init() {
        showStatus("Initializing applet...");
    }
    public void start() {
        showStatus("Starting applet...");
    }
    public void stop() {
        showStatus("Stopping applet...");
    }
    public void destroy() {
        showStatus("Destroying applet...");
    }
    public void paint(Graphics g)
    {
        g.drawString("Hello, World!", 20, 20);
    }
}
```

Applet program to show all life cycle methods of applet

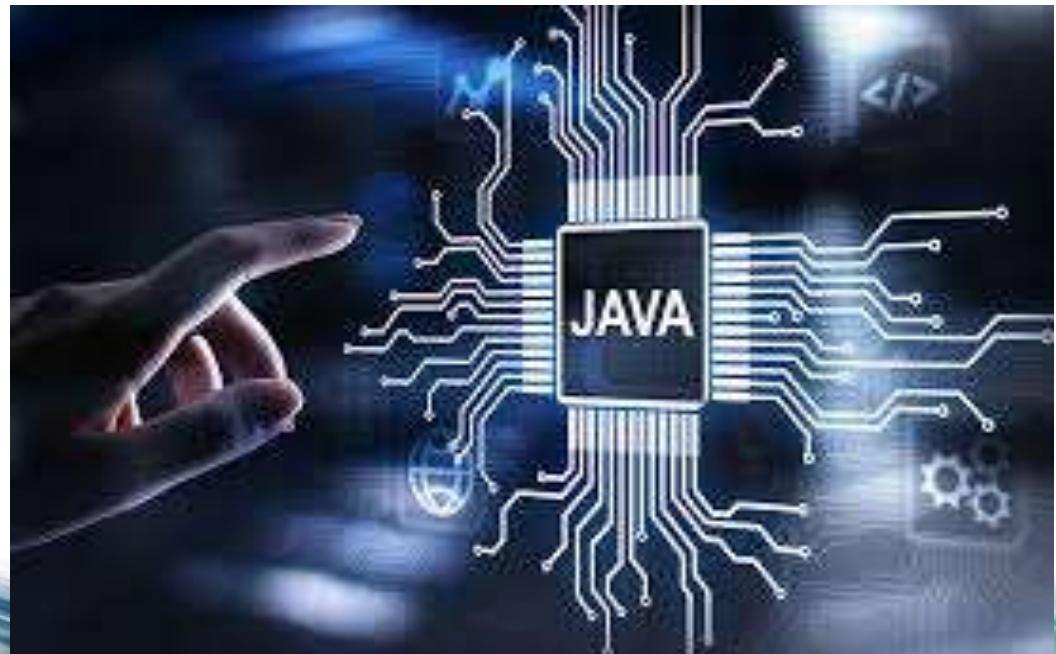
status.html

```
<html>
<head>
  <title>
    Applet
  </title>
</head>
<body>
  Hello, World!
  Your
</body>
</html>
```



Presentation on Java Programming (IT207G)

UNIT No. V- 2



Parameter in Applet

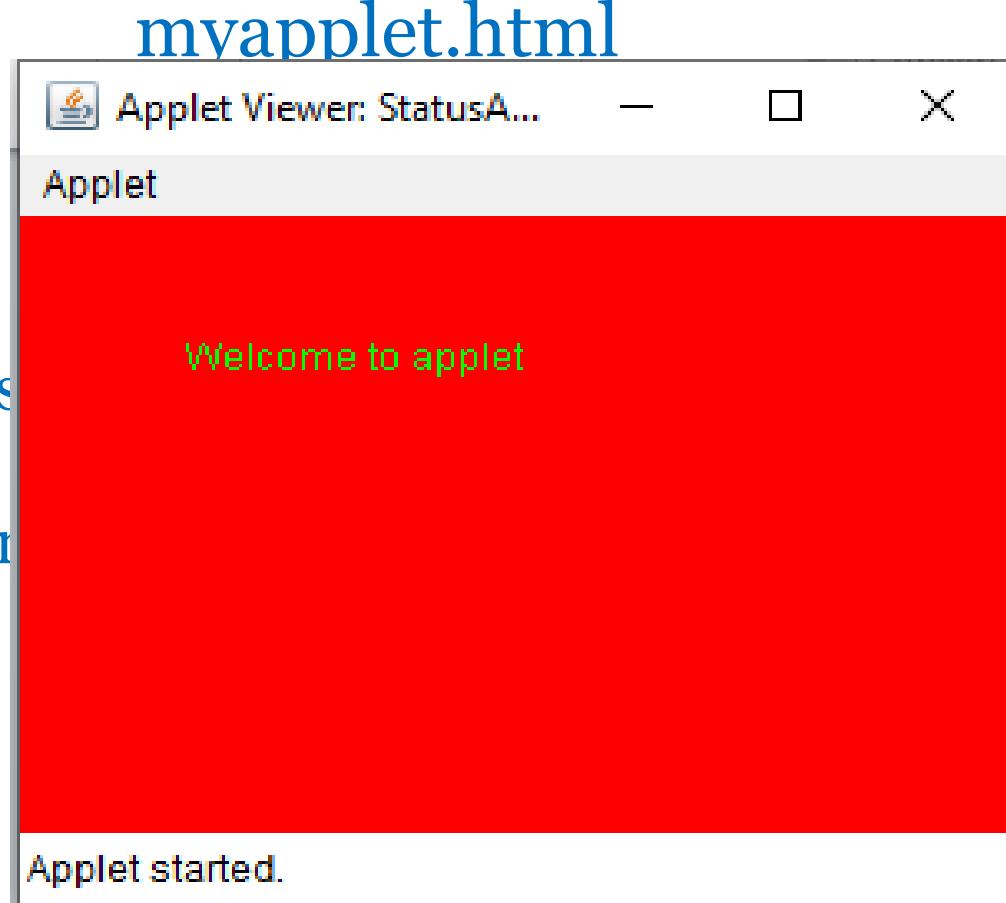
- parameter passing refers to the process of passing data or parameters from the HTML code that embeds the applet into a web page ..
- This mechanism allows you to customize and configure the behavior of the applet dynamically based on the values provided in the HTML code..
- **Step 1: Embedding the Applet in HTML:** To embed a Java applet in an HTML page, you typically use the `<applet>` tag. Within this tag, you can specify various parameters using the `<param>` tags.
`<param name="param1" value="value1">`
- **Step 2: Accessing Parameters in the Applet:** Inside the Java applet code, you can access the parameters passed from the HTML page using the `getParameter()` method of the `Applet` class..
`String str=getParameter("param name");`

Example of using parameter in Applet: UseParam.java

```
import java.applet.Applet;
import java.awt.*;

public class UseParam extends Applet{
    public void init()
    {
        setBackground(Color.red);
        setForeground(Color.green);
    }
    public void paint(Graphics g)
    {
        String str=getParameter("msg");
        g.drawString(str,50, 50);
    }
}
```

```
<html>
<body>
<applet code="Us...>
<param name="i...>
</applet>
</body>
</html>
```



Command Prompt:

c:\>javac UseParam.java

c:\>appletviewer myapplet.html

Class Work:

- Write a program to pass two parameters in Applet, One contains your full name and another will be your branch name. Display both the parameters on applet screen.

Displaying Graphics in Applet

- **java.awt.Graphics** class provides many methods for graphics programming..
1. **public abstract void drawString(String str, int x, int y):** is used to draw the specified string.
 2. **public void drawRect(int x, int y, int width, int height):** draws a rectangle with the specified width and height.
 3. **public abstract void fillRect(int x, int y, int width, int height):** is used to fill rectangle with the default color and specified width and height.
 4. **public abstract void drawOval(int x, int y, int width, int height):** is used to draw oval with the specified width and height.

5. public abstract void fillOval(int x, int y, int width, int height): is used to fill oval with the default color and specified width and height.

6. public abstract void drawLine(int x1, int y1, int x2, int y2): is used to draw line between the points(x1, y1) and (x2, y2).

7. public abstract void drawArc(int x, int y, int width, int height, int startAngle, int arcAngle): is used draw a circular or elliptical arc

8. public abstract void fillArc(int x, int y, int width, int height, int startAngle, int arcAngle): is used to fill a circular or elliptical arc.

9. public abstract void setColor(Color c): is used to set the graphics current color to the specified color.

10. public abstract void setFont(Font font): is used to set the graphics current font to the specified font.

How to drawArc() and fillArc() work?

Syntax:

```
drawArc(int x, int y, int w, int h, int sa, int aa)
```

Parameters: The drawOval method takes four arguments:

X=x-co-ordinate

Y=y co-ordinate

W=width;

H=height;

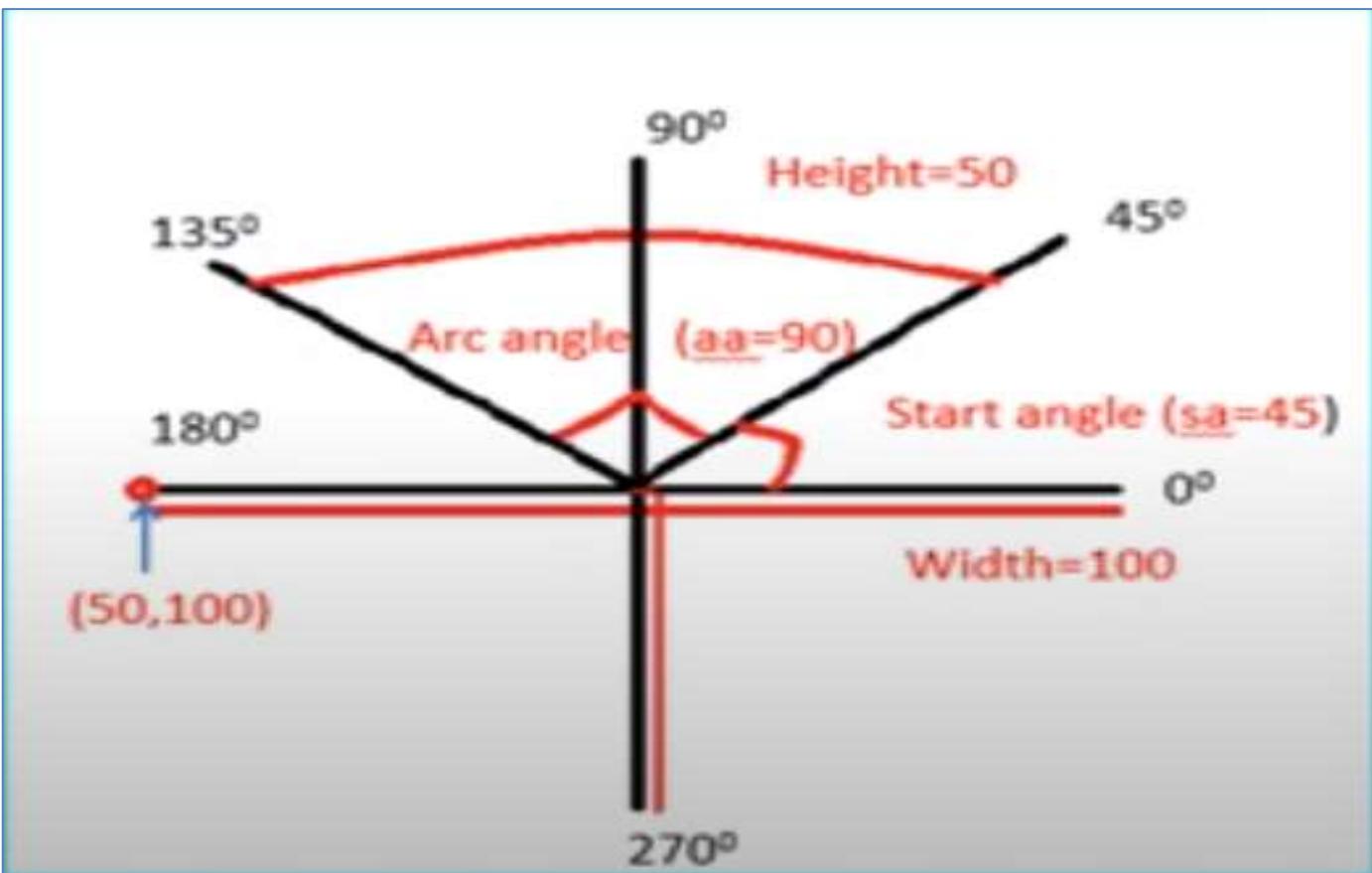
Sa=starting angle

Aa=arc angle (sweep angle)

How to drawArc() and fillArc() work?

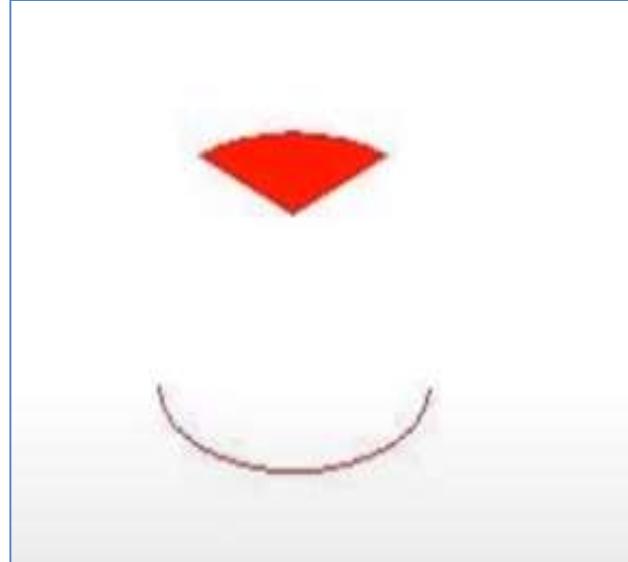
Example:

```
g.drawArc(50,100,100,70,45,90);
```



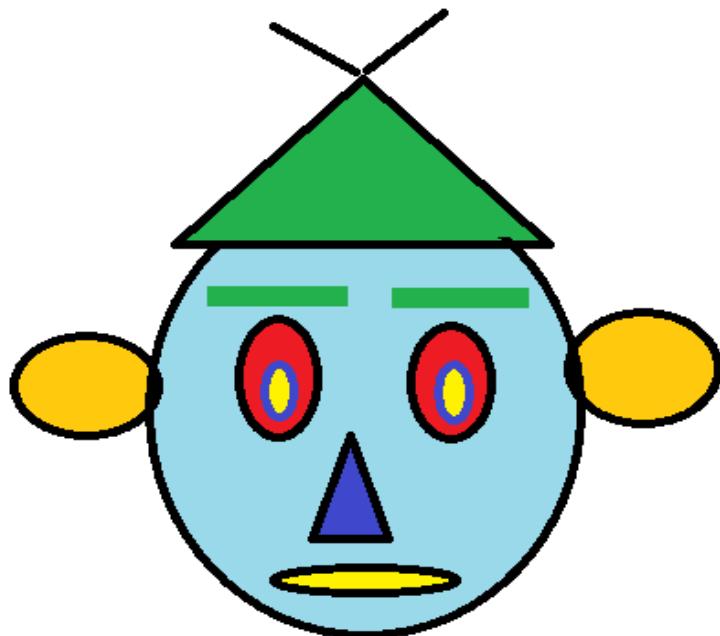
Example Program for drawArc() and fillArc() method

```
import java.awt.*;
import java.applet.*;
public class javaArc extends Applet
{
    public void paint(Graphics g)
    {g.setColor(Color.red);
    g.drawArc(100,150,100,50,180,180);
    g.fillArc(100,100,100,50,45,90);
    }
}
/*<applet code = javaArc.class width = 500
height = 500>
</applet>*/
```



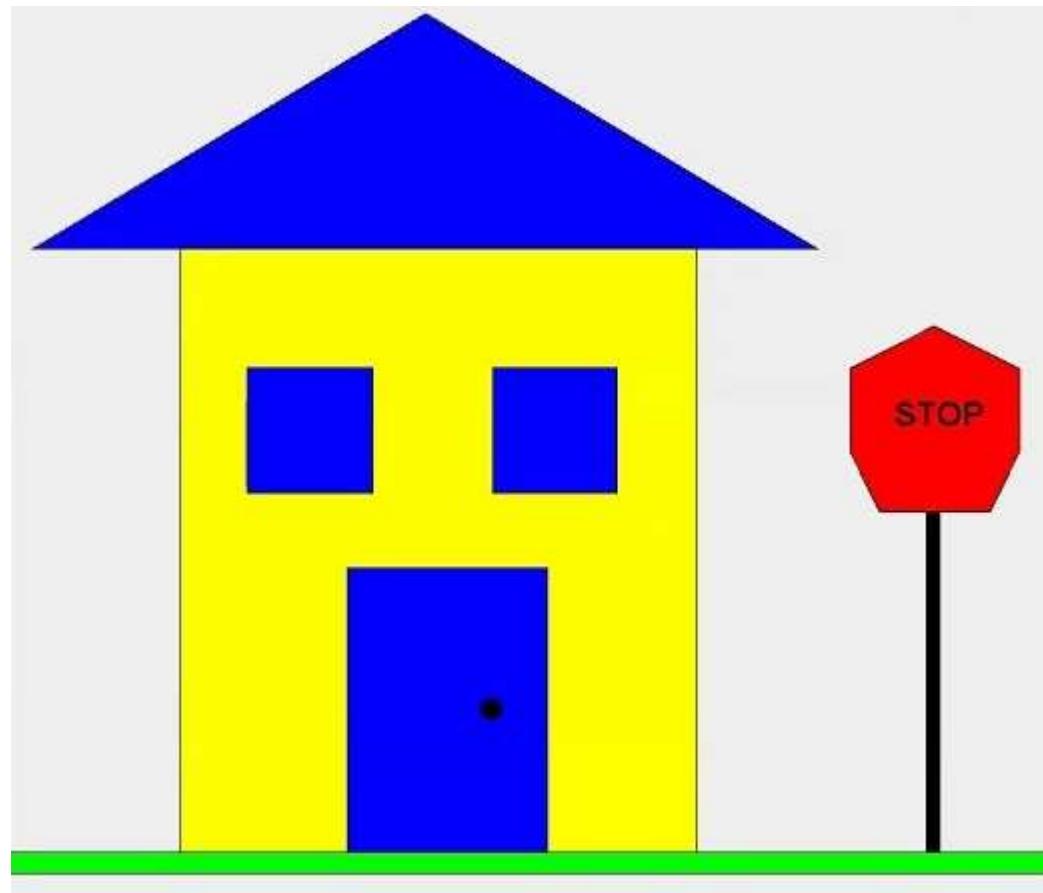
Practical:

- Write program to create applet having following graphics in it.



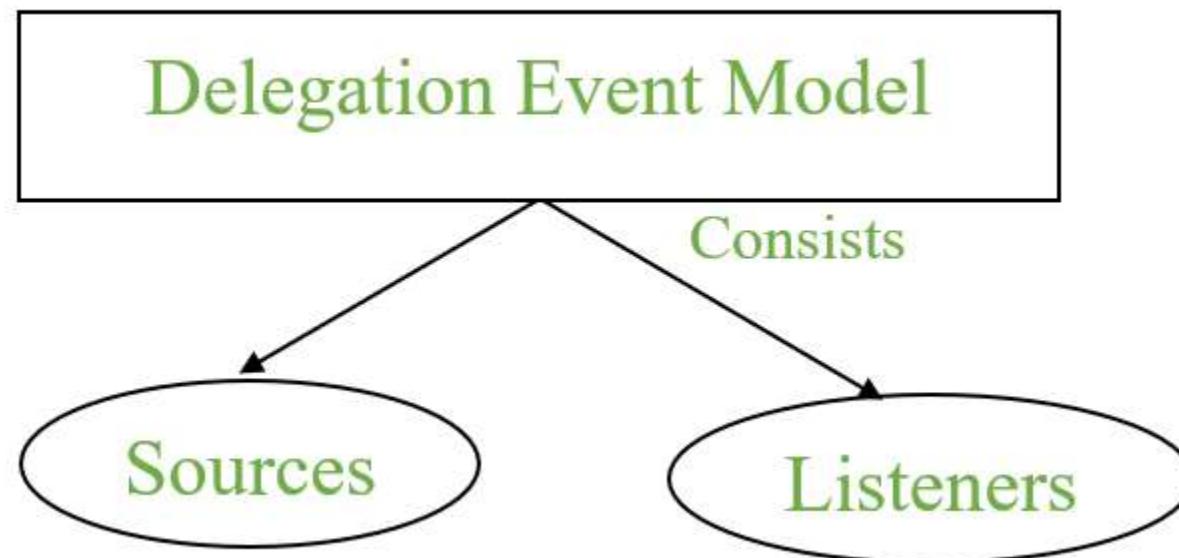
Practical:

- Write program to create applet having following graphics in it.



AWT Event Handling

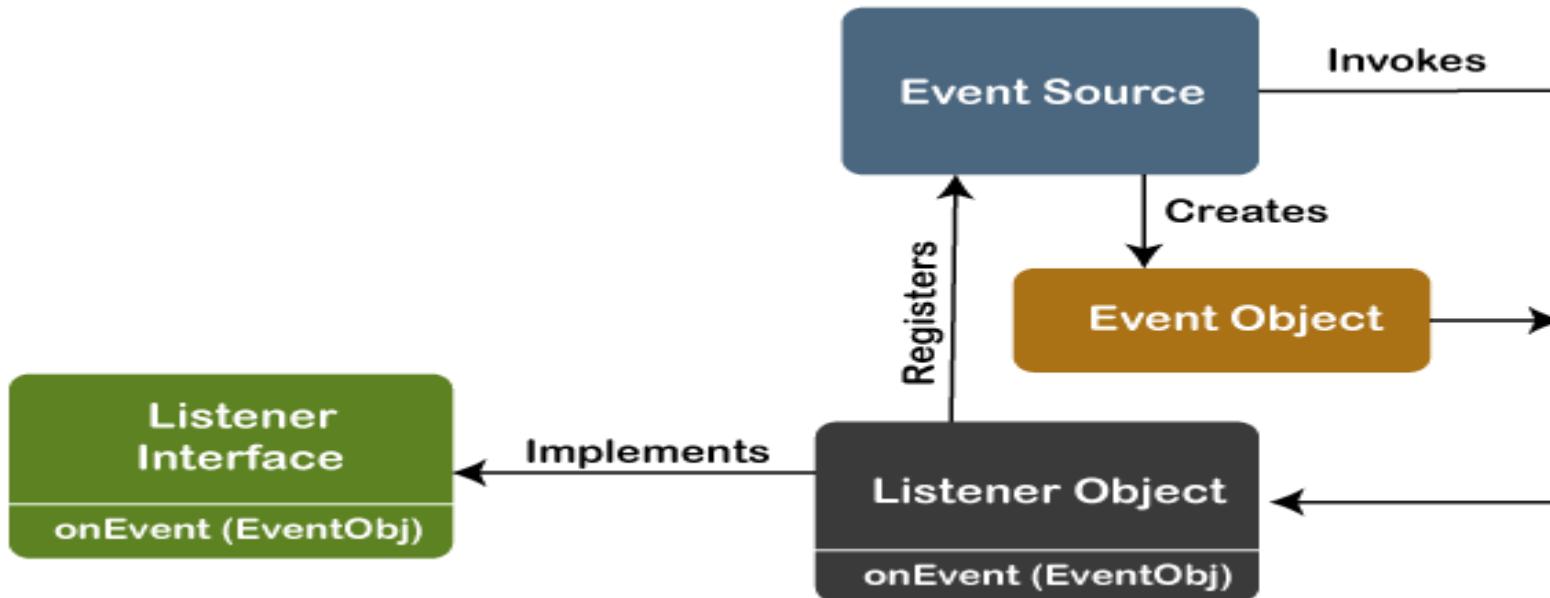
- It is a mechanism to **control the events** and to **decide what should happen after an event** occur. To handle the events, Java follows the ***Delegation Event model***.



Delegation Event model

Source: Events are generated from the source. There are various sources like buttons, checkboxes, list, menu-item, choice, scrollbar, text components, windows, etc., to generate events.

Listeners: Listeners are used for handling the events generated from the source. Each of these listeners represents interfaces that are responsible for handling events.



Steps to perform Event handling

Step 1: Create a new class which extends Applet class and implements appropriate Listener Interface.

Step 2: To perform Event Handling, we need to register the source with the listener.

Syntax: addTypeListener()

Example 1: For **KeyEvent** we use *addKeyListener()* to register.

Step 3: Finally implement abstract method of Listener Interface which contains the logic of event handling

Example 1: For KeyListener

- keyTyped()
- keyPressed()
- keyReleased()

Event Classes in Java

Event Class	Listener Interface	Description
ActionEvent	ActionListener	When a button click or selecting an item from the menu-item list.
ItemEvent	ItemListener	When an item was selected or not from list.
KeyEvent	KeyListener	When keypresses on the keyboard.
MouseEvent	MouseListener & MouseMotionListener	When the user interaction with the mouse (Pointing Device).
MouseWheelEvent	MouseWheelListener	When mouse wheel was rotated in a component.

Different interfaces consists of different methods

Listener Interface	Methods
ActionListener	•actionPerformed()
ItemListener	•itemStateChanged()
KeyListener	•keyTyped() •keyPressed() •keyReleased()
MouseListener	•mousePressed() •mouseClicked() •mouseEntered() •mouseExited() •mouseReleased()
MouseMotionListener	•mouseMoved() •mouseDragged()
MouseWheelListener	•mouseWheelMoved()

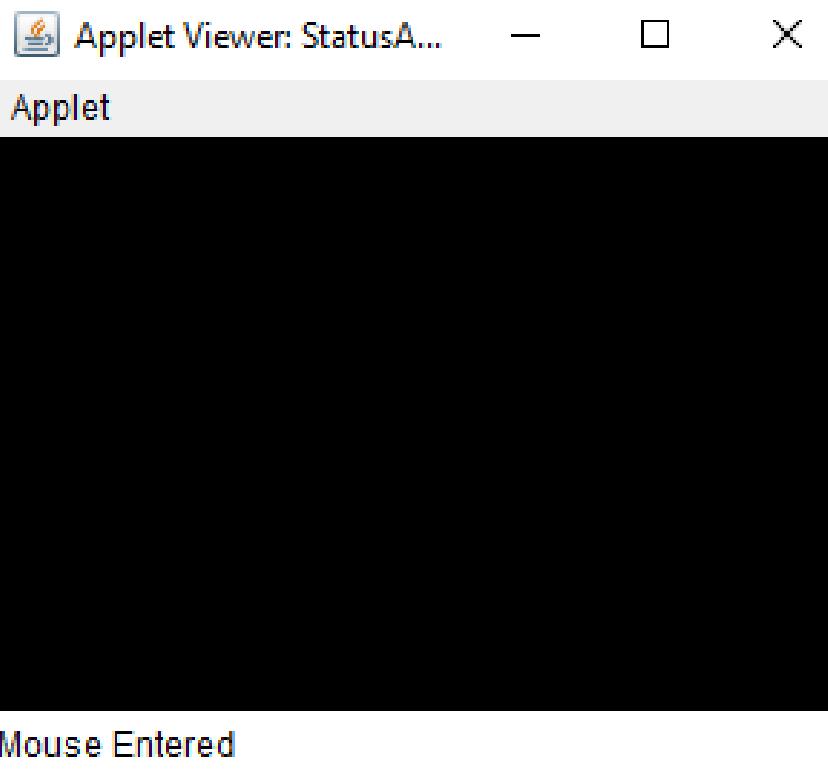
Mouse Event Handling

```
import java.awt.*;
import java.applet.Applet;
import java.awt.event.*;

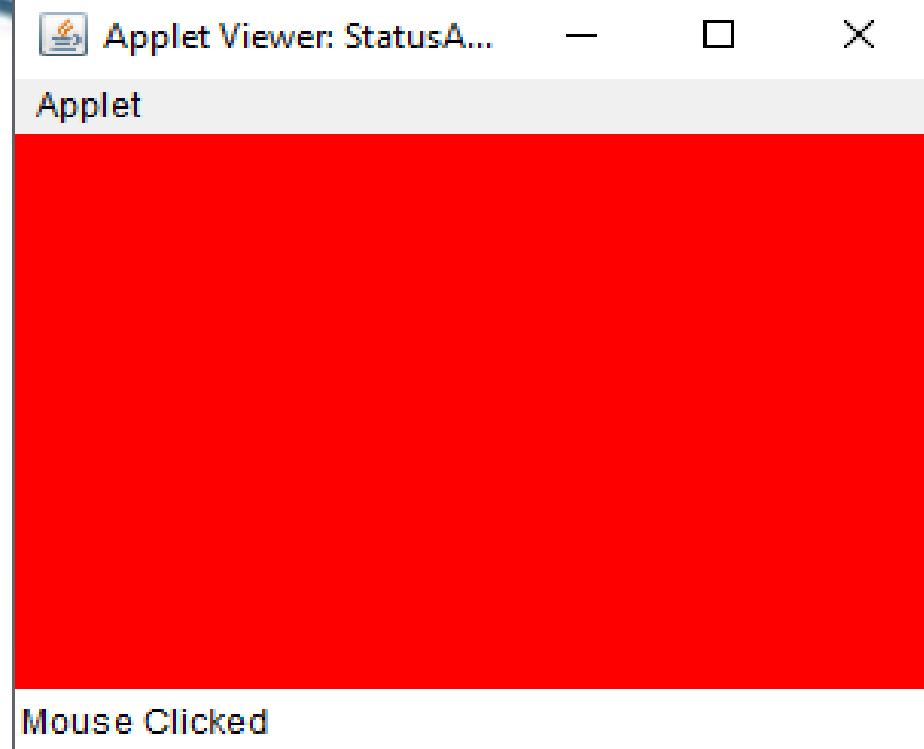
public class StatusApplet2 extends Applet implements MouseListener {
    public void init() {
        addMouseListener(this); //Step 1
    }

    public void mouseClicked(MouseEvent e) {
        showStatus("Mouse Clicked");
        setBackground(Color.red);
        repaint();
    }
}
```

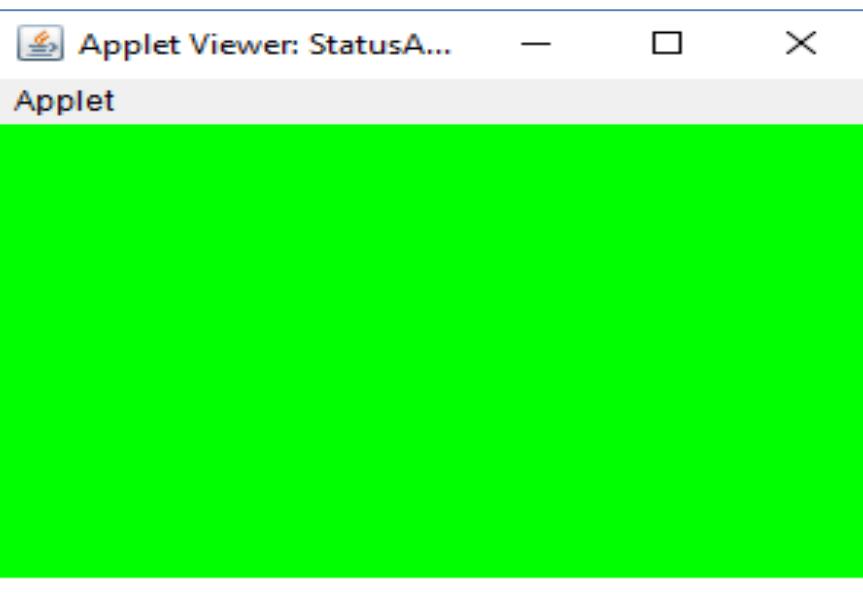
```
public void mousePressed(MouseEvent e) {  
    showStatus("Mouse Pressed");  
    setBackground(Color.green);  
}  
  
public void mouseReleased(MouseEvent e) {  
    showStatus("Mouse Released");  
    setBackground(Color.blue);  
}  
  
public void mouseEntered(MouseEvent e) {  
    showStatus("Mouse black");  
    setBackground(Color.black);  
}  
  
public void mouseExited(MouseEvent e) {  
    showStatus("Mouse Exited");  
    setBackground(Color.yellow);  
}  }
```



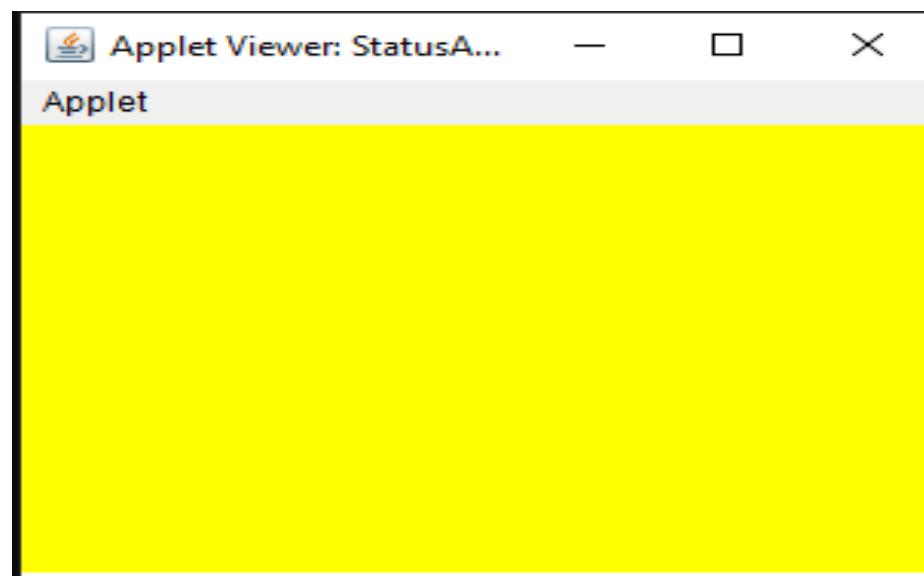
Mouse Entered



Mouse Clicked



Mouse Pressed



Mouse Exited

Introduction to Java Swing

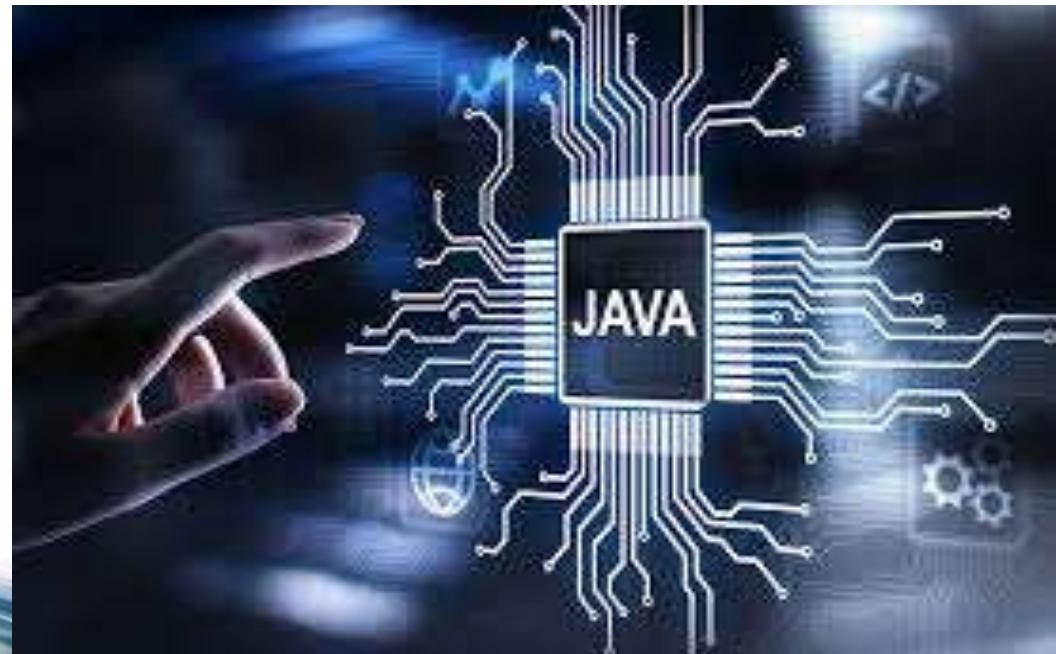
- **Swing** is a Java Foundation Classes [JFC] library and an extension of the Abstract Window Toolkit [AWT]. .
- Java Swing offers much-improved functionality over AWT, new components, expanded components features, and excellent event handling with drag-and-drop support.
- Swing is a Set of API (API- Set of Classes and Interfaces) which Provided to Design Graphical User Interfaces
- Swing is an Extension library to the AWT which Includes New and improved Components that have been enhancing the looks and Functionality of GUIs'

Difference between Java Swing and Java AWT

Java AWT	Java Swing
Java AWT is an API to develop GUI applications in Java.	Swing is a part of Java Foundation Classes and is used to create various applications.
Components of AWT are heavy weighted.	The components of Java Swing are lightweight.
Components are platform dependent.	Components are platform independent.
Execution Time is more than Swing.	Execution Time is less than AWT.
AWT components require java.awt package.	Swing components requires javax.swing package.

Presentation on Java Programming (IT207G)

UNIT No. VI- 1



JDBC

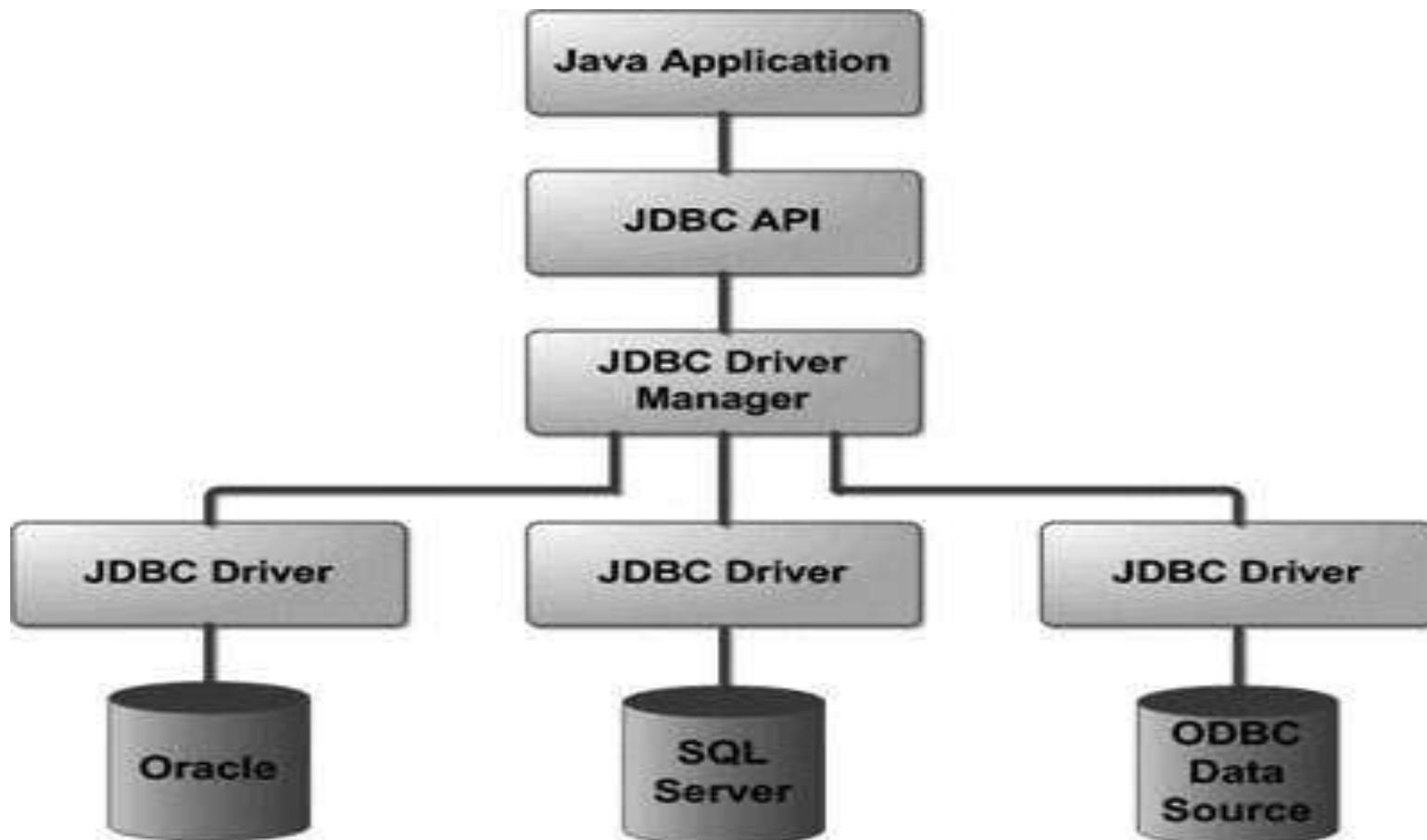
JDBC stands for **Java Database Connectivity**, which is a standard Java API for database-independent connectivity between the Java programming language and a wide range of databases.

The JDBC library includes APIs for each of the tasks mentioned below that are commonly associated with database usage.

- Making a connection to a database.
- Creating SQL or MySQL statements.
- Executing SQL or MySQL queries in the database.
- Viewing & Modifying the resulting records.

Fundamentally, JDBC is a specification that provides a complete set of interfaces that allows for portable access to an underlying database.

JDBC Architecture



Explanation

- 1. Application:** It is the Java [servlet](#) or an applet that communicates with the data source.
- 2. JDBC API:** This provides the application-to-JDBC Manager connection. A few of the crucial interfaces and classes defined in the JDBC API are the following:
 - Drivers
 - DriverManager
 - Statement
 - Connection
 - ResultSet
- 3. Driver:** A driver is an interface that manages database server connectivity. Communication is handled using DriverManager objects.
- 4. JDBC Driver Manager:** In a JDBC application, the Driver Manager loads database-specific drivers. This driver manager makes a database connection. To handle the user request, it additionally makes a database-specific call to the database.

Common JDBC Components

The JDBC API provides the following interfaces and classes –

DriverManager: This class manages a list of database drivers. Matches connection requests from the java application with the proper database driver using communication sub protocol.

Driver: This interface handles the communications with the database server.

Connection: This interface with all methods for contacting a database. The connection object represents communication context, i.e., all communication with database is through connection object only.

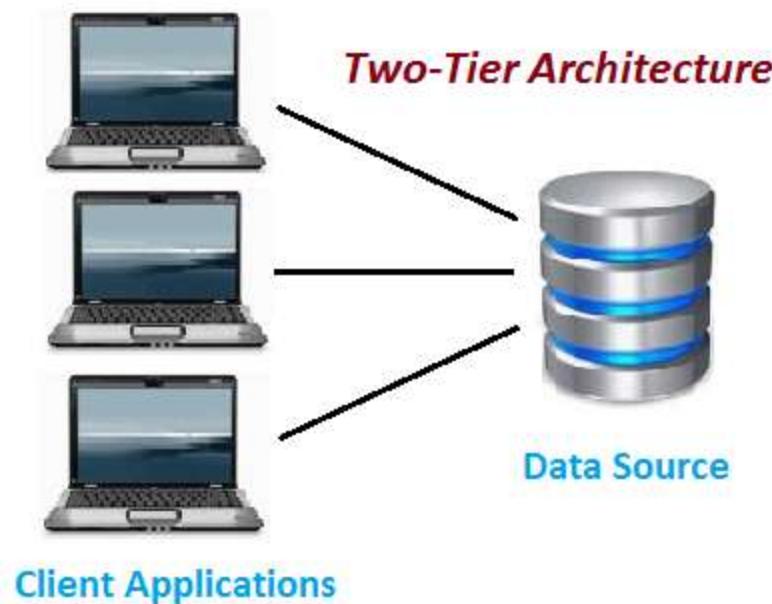
Statement: You use objects created from this interface to submit the SQL statements to the database. Some derived interfaces accept parameters in addition to executing stored procedures.

ResultSet: These objects hold data retrieved from a database after you execute an SQL query using Statement objects. It acts as an iterator to allow you to move through its data.

Different Types of Architecture of JDBC

1. Two-Tier Architecture:

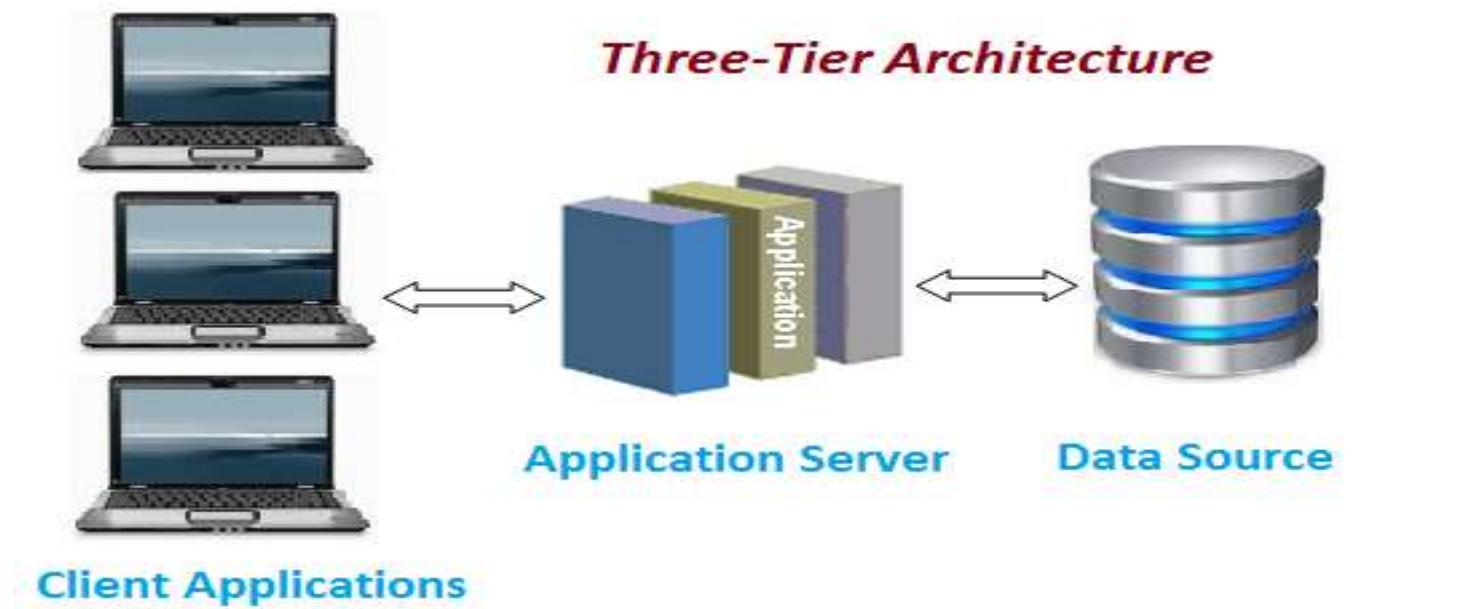
The two-tier is based on Client Server architecture. The two-tier architecture is like client server application. The direct communication takes place between client and server. There is no intermediate between client and server. Because of tight coupling a 2 tiered application will run faster.



Different Types of Architecture of JDBC

2. Three-Tier Architecture:

Three-tier architecture typically comprise a presentation tier (**Client layer**), a business and a data tier. Three layers in the three tier architecture are as follows:



Steps to connect to the database in java

1. Register the driver class:

The `forName()` method of `Class` class is used to register the driver class. This method is used to dynamically load the driver class.

```
Class.forName("oracle.jdbc.driver.OracleDriver");
```

2. Create the connection object

```
Connection con=DriverManager.getConnection( "jdbc:oracle:thin:@localhost:1521:xe"  
, "system","password");
```

3. Create the Statement object

```
Statement stmt=con.createStatement();
```

4. Execute the query

```
ResultSet rs=stmt.executeQuery("select * from emp");
```

5. Close the connection object

```
con.close();
```

```
import java.sql.*;
class OracleCon{
public static void main(String args[]){
try{
//step1 load the driver class
Class.forName("oracle.jdbc.driver.OracleDriver");

//step2 create the connection object
Connection con=DriverManager.getConnection(
"jdbc:oracle:thin:@localhost:1521:xe","system","oracle");

//step3 create the statement object
Statement stmt=con.createStatement();

//step4 execute query
int n=stmt.executeUpdate("insert into students values(109, 'Sanjay')");
System.out.println("Row inserted: "+n);

} catch(Exception e){ System.out.println(e);}
} }
```

GOVERNMENT POLYTECHNIC, GONDIA INFORMATION TECHNOLOGY

SUBJECT : ADVANCE JAVA (IT207G)

LESSON NO :1

UNIT V : INTRACTING WITH DATABASE

TEACHER: A.G. BARSAGADE

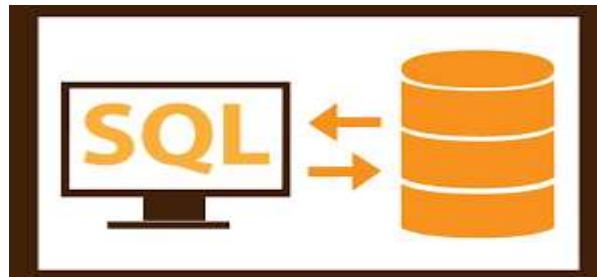
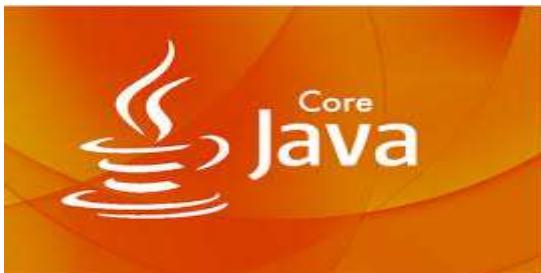


PRE-REQUISITE ?

- Before moving further, you need to have a good understanding of the following two subjects –

That you have learned in last semester (4th)

- 1) Core JAVA Programming
- 2) SQL or MySQL Database i.e. DBMS



LESSON NO :1

TOPIC & SUB TOPIC

- Introduction to JDBC
- JDBC Architecture
 - 1. Two Tier Architecture
 - 2. Three Tier Architecture
- Standard steps to connection with Database

OUTCOMES

- Student should able understand definition & basic of JDBC.
- Student should able understand two and three tier architecture of JDBC.
- Student should able to write first program of java database connectivity.

WHAT IS JDBC?

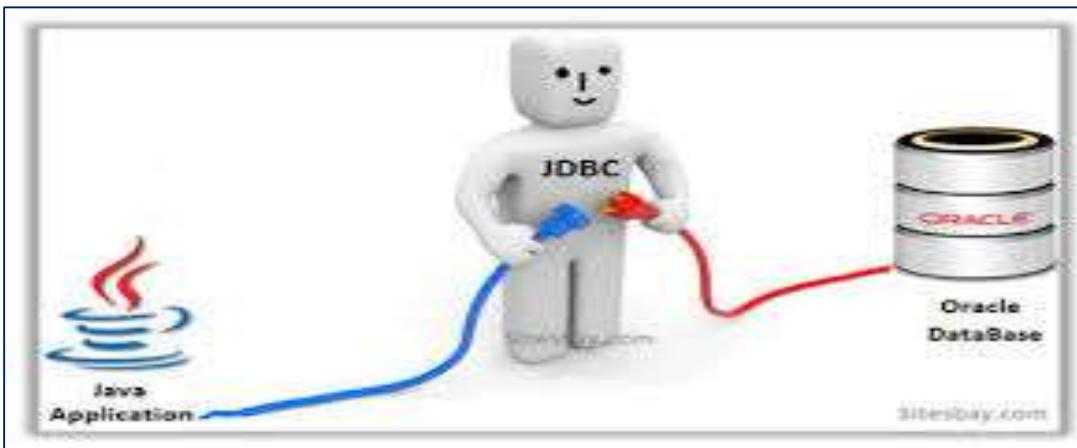
- JDBC stands for **Java Database Connectivity**, which is a standard Java API for database-independent connectivity between the Java programming language and a wide range of databases.
- The JDBC library includes APIs for each of the tasks mentioned below that are commonly associated with database usage.
 - 1) Making a connection to a database.
 - 2) Creating SQL or MySQL statements.
 - 3) Executing SQL or MySQL queries in the database.
 - 4) Viewing & Modifying the resulting records.

WHAT IS JAVA JDBC API?

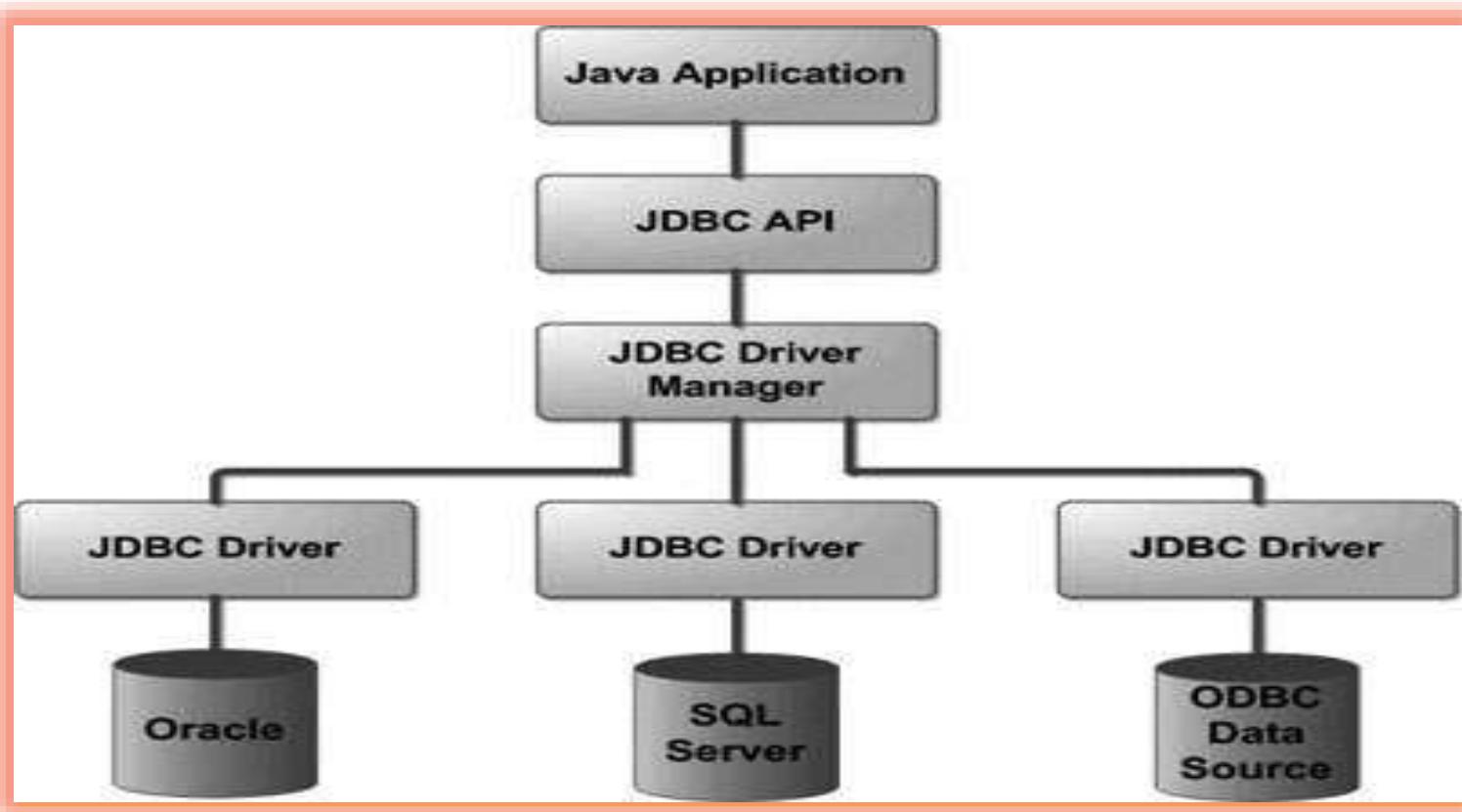
- The Java Database Connectivity (JDBC) API provides universal data access from the Java programming language. Using the JDBC API, you can access virtually any data source, from relational databases to spreadsheets and flat files. JDBC technology also provides a common base on which tools and alternate interfaces can be built.
- The JDBC API is comprised of two packages:
 - 1) `java.sql`
 - 2) `javax.sql`
- You automatically get both packages when you download the Java Platform Standard Edition (Java SE) 8.

NEED OF JDBC?

- We can use JDBC API to handle database using Java program and can perform the following activities:
 - 1) Connect to the database
 - 2) Execute queries and update statements to the database
 - 3) Retrieve the result received from the database.



JDBC ARCHITECTURE



COMMON JDBC COMPONENTS

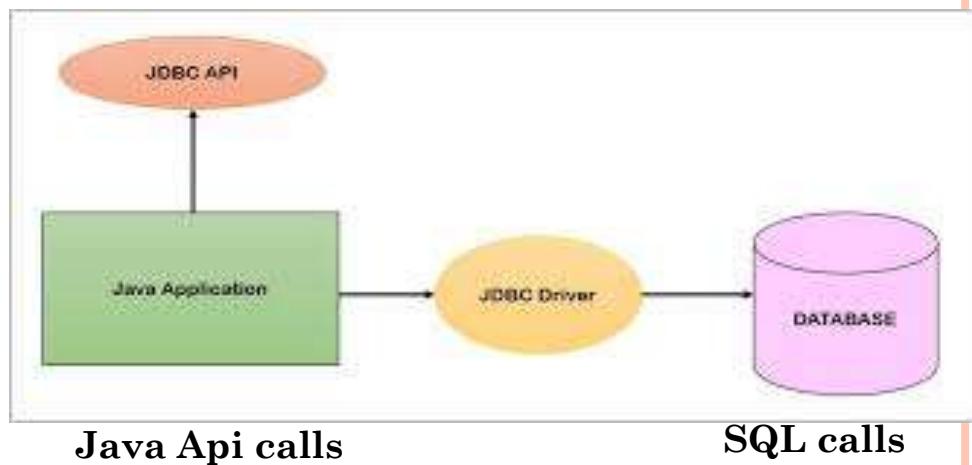
- 1) **Java Application:** It is our client program which contains JDBC API client logic and our business logic.
- 2) **JDBC API:** It provides the connectivity between Java application and Database. Java application can communicate with any Database with the help of **DriverManager** and Database specific **Driver**.
- 3) **DriverManager:** DriverManager is responsible to manage all Database Drivers available in our Application. DriverManager will register and un-register Database Drivers. DriverManager can establish the connection to the Database with the help of Driver Software.

Continue.....

4) DRIVER :

A driver is a translation software developed according to JDBC specifications. It performs two duties :

- 1) Receive the java api calls from java application, translating them into DBMS understandable SQL calls & submitting them to DBMS.
- 2) Receive the result from DBMS, translating them to Java api calls & giving them to java application.



TWO-TIER ARCHITECTURE FOR DATA ACCESS.

- Two-tier Architecture provides direct communication between Java applications to the database. It requires a **JDBC driver** that can help to communicate with the particular database.

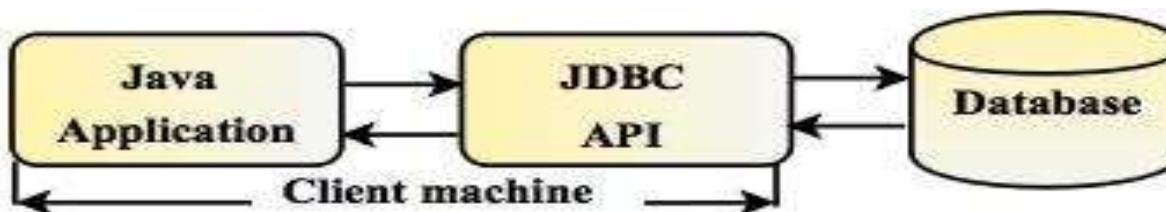


Fig: Two-tier Architecture of JDBC

THREE-TIER ARCHITECTURE FOR DATA ACCESS.

- In the three-tier model, commands are sent by the HTML browser to middle services i.e. Java application which can send the commands to the particular database. The middle tier has been written in **C or C++** languages. It can also provide better performance.

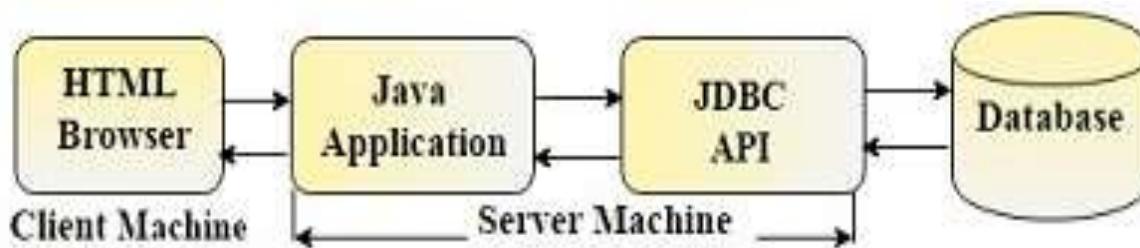


Fig: Three-tier Architecture of JDBC

STANDARD STEPS TO CONNECTION WITH DATABASE

Step 1:

Importing the sql
package

Step 2:

Loading the Driver

Step 3:

Creating a connection
String

Step 4:

Requesting for
connection

Step 5:

Closing the connection



STEP 1: IMPORTING THE SQL PACKAGE

This is for making the JDBC API classes immediately available to the application program.

```
import java.sql.*;
```

STEP 2: LOADING THE JDBC DRIVER

This is for establishing a communication between the JDBC program and the Oracle database.

The `forName()` method of the `java.lang.Class` class can be used to load and register the JDBC driver:

```
Class.forName("oracle.jdbc.driver.OracleDriver");
```



STEP 3: CREATING A CONNECTION STRING

Connection String is nothing but database URL (address). In case of Oracle database the connection String will be as follows example:

```
String cs="jdbc:oracle:thin:@localhost:1521:XE"
```

STEP 4: REQUESTING FOR CONNECTION

This is done by using the getConnection() method of the DriverManager class which takes three parameter as URL i.e. connection string , username, and password od database user.

The following lines of code illustrate using getConnection() method:

```
Connection conn = DriverManager.getConnection(cs, "ajay","1234");
```

NEXT: YOU CAN PERFORM SOME DATABASE OPERATION

Database Operations: Create, Insert, Delete, Update etc...

STEP 5: CLOSING THE CONNECTION

The last step is to close the database connection opened in the beginning after importing the packages and loading the JDBC drivers. This is done by a call to the `close()` method of the `Connection` class.

`Conn.close();`



EXAMPLE : CONNECT THE JAVA APPLICATION WITH ORACLE DATABASE.

//Step 1: Import sql package

```
import java.sql.*;  
class JDBCdemo  
{  
    public static void main(String args[]) {  
        try {
```

//Step 2: Load the driver

```
Class.forName("oracle.jdbc.driver.OracleDriver");
```

//Step 3: Creating a connection String

```
String cs="jdbc:oracle:thin:@localhost:1521:XE"
```

//Step 4: Requesting for connection

```
Connection con = DriverManager.getConnection("cs,"scott", "tiger");
```

//Step 5: Closing the connection

```
        con.close(); }
```

```
catch(Exception e)
```

```
    {  
        System.out.println(e); }  
    } }
```



**GOVERNMENT POLYTECHNIC, GONDIA
INFORMATION TECHNOLOGY**

SUBJECT : ADVANCE JAVA (IT207G)

LESSON NO :2

UNIT V : INTRACTING WITH DATABASE

TEACHER: A.G. BARSAGADE



LESSON NO :2

TOPIC & SUB TOPIC

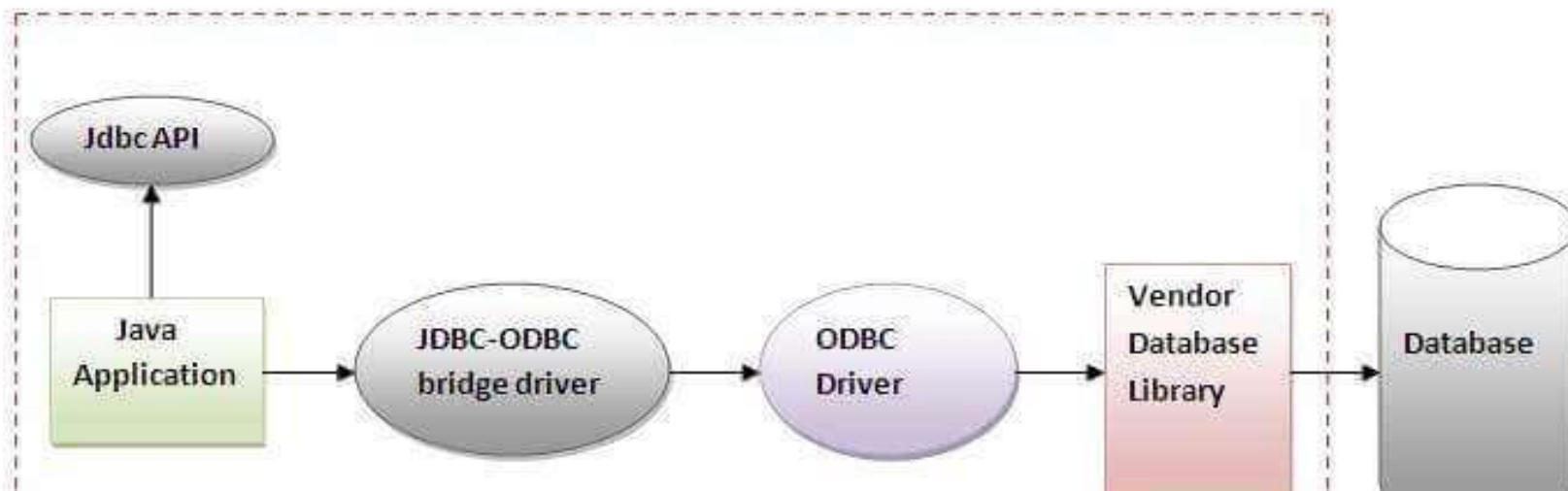
- Types of JDBC Drivers.
- JDBC Classes & Interfaces
 - 1. DriverManager class
 - 2. Driver Interface
 - 3. Connection Interface

OUTCOMES (UO's)

- Student should able to use relevant type of JDBC driver for Specified environment.
- Student should able understand and used specific classes & Interfaces of JDBC.
- Student should able to write

1) JDBC-ODBC BRIDGE DRIVER

- The JDBC-ODBC bridge driver uses ODBC driver to connect to the database. The JDBC-ODBC bridge driver converts JDBC method calls into the ODBC function calls. This is now discouraged because of thin driver.



CONTINUE.....

- Oracle does not support the JDBC-ODBC Bridge from Java 8. Oracle recommends that you use JDBC drivers provided by the vendor of your database instead of the JDBC-ODBC Bridge.

- **Advantages:**

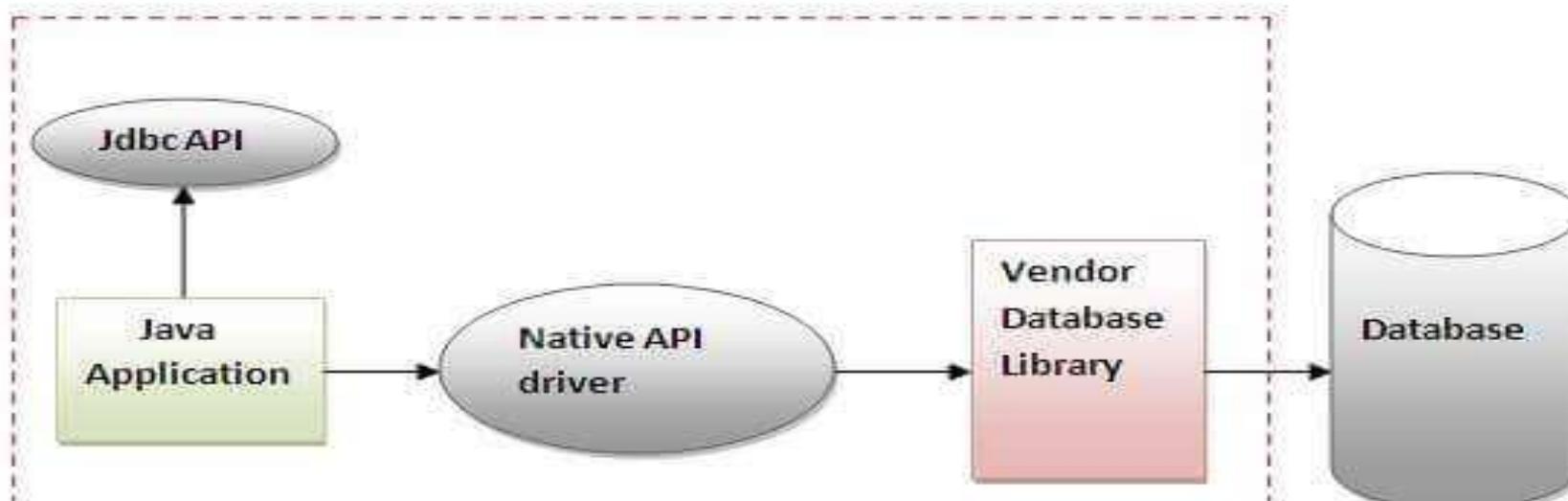
1. easy to use.
2. can be easily connected to any database.

- **Disadvantages:**

1. Performance degraded because JDBC method call is converted into the ODBC function calls.

2) NATIVE-API DRIVER

- The Native API driver uses the client-side libraries of the database. The driver converts JDBC method calls into native calls of the database API. It is not written entirely in java.



CONTINUE.....

- **Advantages:**

1. performance upgraded than JDBC-ODBC bridge driver.

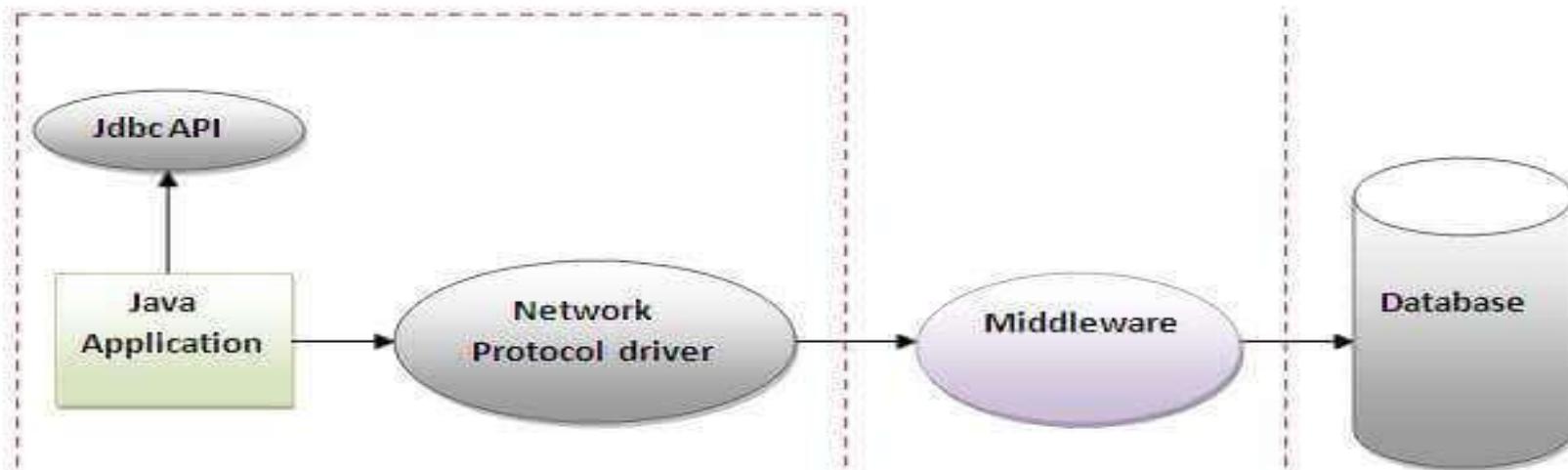
- **Disadvantages:**

1. The Native driver needs to be installed on the each client machine.
2. The Vendor client library needs to be installed on client machine.



3) NETWORK PROTOCOL DRIVER

- The Network Protocol driver uses middleware (application server) that converts JDBC calls directly or indirectly into the vendor-specific database protocol.
- It is fully written in java.



CONTINUE.....

- **Advantages:**

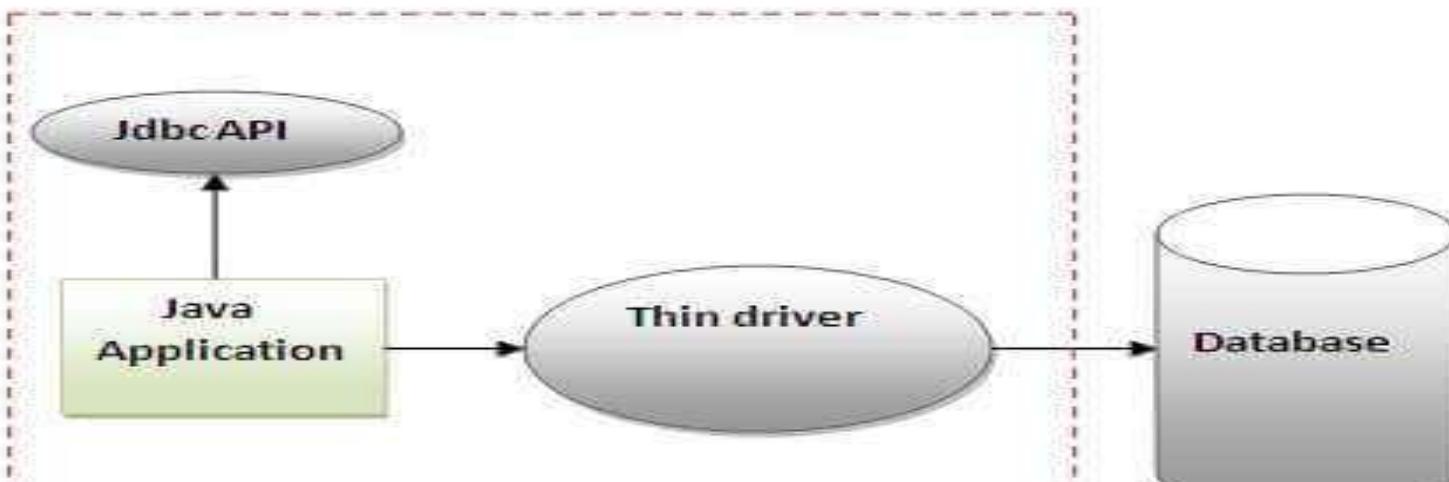
1. No client side library is required because of application server that can perform many tasks like auditing, load balancing, logging etc.

- **Disadvantages:**

1. Network support is required on client machine.
2. Requires database-specific coding to be done in the middle tier.
3. Maintenance of Network Protocol driver becomes costly because it requires database-specific coding to be done in

4) THIN DRIVER

- The thin driver converts JDBC calls directly into the vendor-specific database protocol. That is why it is known as thin driver.
- It is fully written in Java language.



CONTINUE.....

- **Advantages:**

1. Better performance than all other drivers.
2. No software is required at client side or server side.

- **Disadvantages:**

1. Drivers depend on the Database.



JDBC CLASSES & INTERFACES

1. DriverManager class:

The DriverManager class acts as an interface between user and drivers. It keeps track of the drivers that are available and handles establishing a connection between a database and the appropriate driver.

Useful methods of DriverManager class:

Method	Description
1) public static void registerDriver(Driver driver):	is used to register the given driver with DriverManager.
2) public static void deregisterDriver(Driver driver):	is used to deregister the given driver (drop the driver from the list) with DriverManager.
3) public static Connection	is used to establish the connection with

2. CONNECTION INTERFACE

A Connection is the session between java application and database. The Connection interface is a factory of Statement, PreparedStatement, and DatabaseMetaData i.e. object of Connection can be used to get the object of Statement and DatabaseMetaData.

Useful methods of DriverManager class:

1) public Statement createStatement(): creates a statement object that can be used to execute SQL queries.

2) public Statement createStatement(int resultSetType, int resultSetConcurrency): Creates a Statement object that will generate ResultSet objects with the given type and concurrency.

3. STATEMENT INTERFACE

The **Statement interface** provides methods to execute queries with the database. The statement interface is a factory of **ResultSet** i.e. it provides factory method to get the object of **ResultSet**.

Commonly used methods of Statement interface:

1) public ResultSet executeQuery(String sql): is used to execute SELECT query. It returns the object of **ResultSet**.

2) public int executeUpdate(String sql): is used to execute specified query, it may be create, drop, insert, update, delete etc.

3) public boolean execute(String sql): is used to execute queries that



**STAY AT HOME
STAY SAFE
STAY INSPIRED!**

**THANK
YOU**

**GOVERNMENT POLYTECHNIC, GONDIA
INFORMATION TECHNOLOGY**

SUBJECT : ADVANCE JAVA (IT207G)

LESSON NO :3

UNIT V : INTRACTING WITH DATABASE

TEACHER: A.G. BARSAGADE



LESSON NO :3

TOPIC & SUB TOPIC

- Create Table in Oracle DB.
- Statement Interface
- 1. Example Program 1
- 2. Example program 2
- ResultSet Class

OUTCOMES (UO's)

- Student should able to create table in oracle DB.
- Student should able understand and used Statement Interface & ResultSet Class of JDBC.

CREATE A TABLE

- Before establishing connection, let's first create a table in oracle database. Following is the SQL query to create a table.
- `create table emp(id number(10),name varchar2(40), age number(3));`

ID	NAME	AGE

STATEMENT INTERFACE

```
import java.sql.*;
class InsertRecord{
public static void main(String args[])throws Exception{
Class.forName("oracle.jdbc.driver.OracleDriver");
Connection con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:XE","system","oracle");
Statement stmt=con.createStatement();
int result=stmt.executeUpdate("insert into emp values(1,'Ajay',35)");
System.out.println(" Records Inserted: "+result);
con.close();
}}
```

CONTINUE.....

- Following is the SQL query to show the inserted record in a table.
- Sql> select * from emp;

ID	NAME	AGE
01	Ajay	35

- Again execute java program then data of table will be:

ID	NAME	AGE

ANOTHER EXAMPLE OF STATEMENT INTERFACE

```
import java.sql.*;
class InsertRecord{
public static void main(String args[])throws Exception{
Class.forName("oracle.jdbc.driver.OracleDriver");
Connection con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:XE","system","oracle");
Statement stmt=con.createStatement();
int result=stmt.executeUpdate("delete from emp where id=03");
System.out.println(" Records Deleted: "+result);
con.close();
}}
```

CONTINUE.....

- Following is the SQL query to show the deleted record in a table.
- Sql> select * from emp;

ID	NAME	AGE
01	Ajay	35
02	Sanju	32

RESULTSET INTERFACE

- 1) The object of ResultSet maintains a cursor pointing to a row of a table.
- 2) Initially, cursor points to before the first row.
- 3) By default, ResultSet object can be moved forward.

<u>Methods</u>	<u>Descriptions</u>
1) public int getInt(int columnIndex):	is used to return the data of specified column index of the current row as int.
2) public int getInt(String columnName):	is used to return the data of specified column name of the current row as int.
3) public String getString(int	is used to return the data of specified

CREATING A RESULTSET

- You create a ResultSet by executing a Statement or PreparedStatement, like this:

```
Statement statement = connection.createStatement();
```

```
ResultSet result = statement.executeQuery("select * from emp");
```

This ResultSet has 3 different columns (Id,Name,Age), and 3 records with different values for each column.

ID	NAME	AGE
1	A	20
2	B	21
3	C	22

ITERATING THE RESULTSET

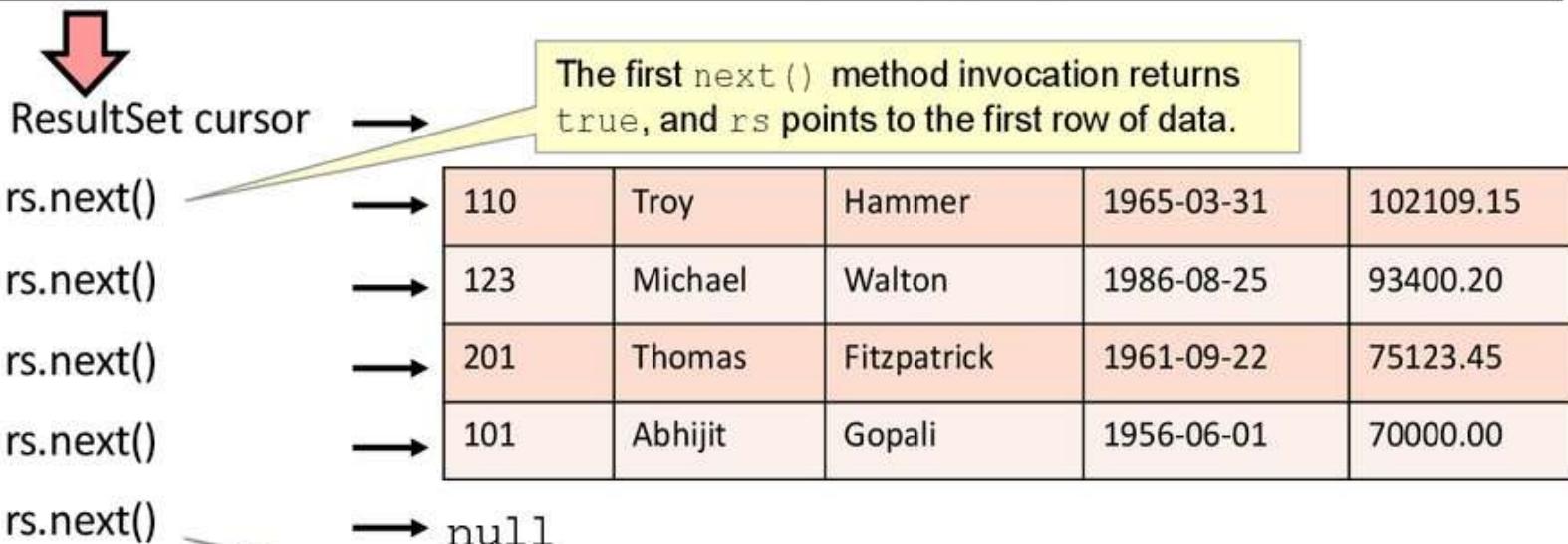
- To iterate the ResultSet you use its next() method.
- The next() method returns true if the ResultSet has a next record, and moves the ResultSet to point to the next record.
- If there were no more records, next() returns false, and you can no longer.
- Once the next() method has returned false, you should not call it anymore.

Here is an example of iterating a ResultSet using the next() method:

```
while(result.next()) { // ... get column values from this record }
```

Using a ResultSet Object

```
String query = "SELECT * FROM Employee";  
ResultSet rs = stmt.executeQuery(query);
```



The last next() method invocation returns

ACCESSING COLUMN VALUES

- When iterating the ResultSet you want to access the column values of each record.
- You do so by calling one or more of the many get***() methods.
- You pass the name of the column to get the value of, to the many get***() methods.

```
while(result.next())
{
    result.getInt ("id")
    result.getString ("name");
```

CONTINUE..... ANOTHER VERSION OF METHOD

- The getXXX() methods also come in versions that take a column index instead of a column name. For instance:

```
while(result.next())
{
    int id=result.getInt (1);
    String name=result.getString (2);
    int age=result.getInt (3);
}
```

NAVIGATION METHODS

- The ResultSet interface contains the following navigation methods.

Method	Description
absolute()	Moves the ResultSet to point at an absolute position. The position is a row number passed as parameter to the absolute() method.
afterLast()	Moves the ResultSet to point after the last row in the ResultSet.
beforeFirst()	Moves the ResultSet to point before the first row in the ResultSet.
first()	Moves the ResultSet to point at the first row in the ResultSet.
last()	Moves the ResultSet to point at the last row in the ResultSet.

4.4.1: RESULTSET INTERFACE EXAMPLE

```
import java.sql.*;
class FetchRecord{
public static void main(String args[])throws Exception{

Class.forName("oracle.jdbc.driver.OracleDriver");
Connection con=DriverManager.getConnection("jdbc:oracle:thin:@localhost
:1521:xe","system","oracle");
Statement stmt=con.createStatement();
ResultSet rs=stmt.executeQuery("select * from emp");
//getting the record
while(result.next())
{
int id=result.getInt (1);
String name=result.getString (2);
int age=result.getInt (3);
```



THANK
YOU

**GOVERNMENT POLYTECHNIC, GONDIA
INFORMATION TECHNOLOGY**

SUBJECT : ADVANCE JAVA (IT207G)

LESSON NO :4

UNIT V : INTRACTING WITH DATABASE

TEACHER: A.G. BARSAGADE



LESSON NO :4

TOPIC & SUB TOPIC

- *ResultSetMetaData*
Interface

1. Example Program 1

- *ResultSet* Class

1. Example Program 1

OUTCOMES (UO's)

- Student should able understand and used *ResultSetMetaData* Interface & all it's methods.

- Student should able understand and used *PreparedStatement* Interface & all it's methods.
- Student should able to write

RESULTSETMETADATA IN JDBC :

- ResultSetMetaData is an interface, which is coming from java.sql package.
- If we want to read the data from database we need to use executeQuery() method. Obviously executeQuery() method returns ResultSet object, which consists the data rows.
- If we want to read the data from ResultSet object, then we should have the knowledge about the data stored in ResultSet.
- Because based on the type of data only, we need to use appropriate getXxx() method like below:

CONTINUE.....

- If we don't know the exact data about the ResultSet, what type of get???() method we use?
- No problem, here is the solution. ResultSetMetaData is actually comes to help us regarding this.
- ResultSetMetaData provides the data about the ResultSet object.
- It will provide all necessary information about the data available in ResultSet.
- We can get the ResultSetMetaData object by calling **getMetadata()** method on ResultSet object.

COMMONLY USED METHODS OF RESULTSETMETADATA INTERFACE

Method	Description
<code>public int getColumnCount()throws SQLException</code>	it returns the total number of columns in the ResultSet object.
<code>public String getColumn Name(int index) throws SQLException</code>	it returns the column name of the specified column index.
<code>public String getColumnType Name(int index) throws SQLException</code>	it returns the column type name for the specified index.

RESULTSETMETADATA INTERFACE EXAMPLE

```
import java.sql.*;
class FetchRecord{
public static void main(String args[])throws Exception{

Class.forName("oracle.jdbc.driver.OracleDriver");
Connection con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:xe
","system","oracle");
Statement stmt=con.createStatement();
ResultSet rs=stmt.executeQuery("select * from emp");
ResultSetMetaData rsmd=rs.getMetaData();
int count=rsmd.getColumnCount();
for(int i=1;i<=count;i++)
{   System.out.print(" "+rsmd.getColumnName(i));    }
System.out.println();
while(result.next())
{
```

JDBC - PREPAREDSTATEMENT

- The PreparedStatement interface is a subinterface of Statement. It is used to execute parameterized query.
- Let's see the example of parameterized query:

```
String sql="insert into emp values(?, ?, ?);
```

As you can see, we are passing parameter (?) for the values. Its value will be set by calling the setter methods of PreparedStatement.

WHY USE PREPAREDSTATEMENT?

- **Improves performance:** The performance of the application will be faster if you use PreparedStatement interface because query is compiled only once.
- How to get the instance of PreparedStatement?
- The prepareStatement() method of Connection interface is used to return the object of PreparedStatement.

Example of PreparedStatement interface that inserts the record

- First of all create table as given below:

PreparedStatement stmt=con.prepareStatement("update emp set

name=? where id=?");

stmt.setString(1,"Sonoo");



THANK
YOU

Chapter 06

SERVLET

Lesson No: 1

Introduction:

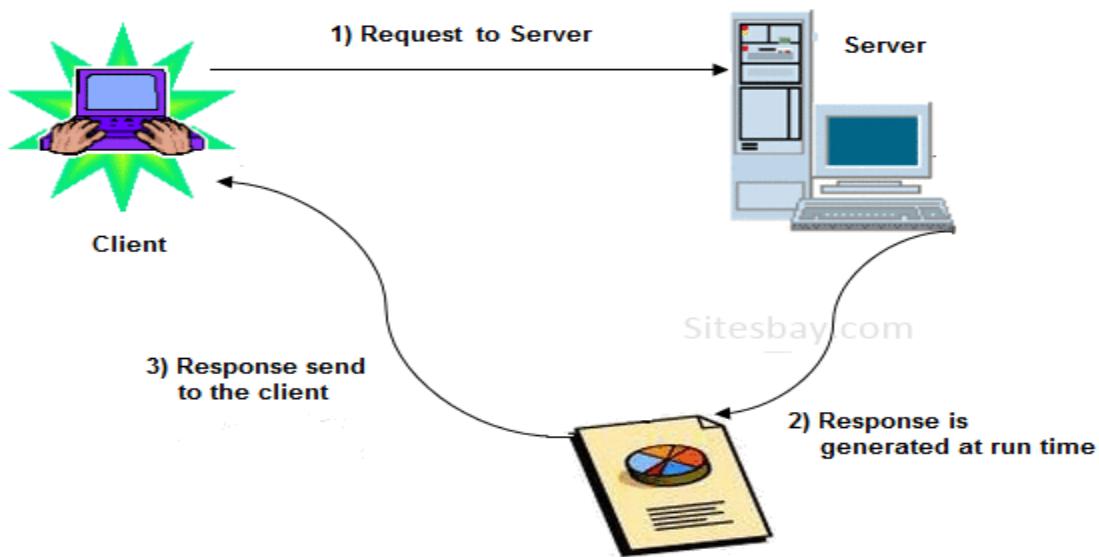
Servlet technology is used to create a web application (resides at server side and generates a dynamic web page).

Servlet technology is robust and scalable because of java language. Before Servlet, CGI (Common Gateway Interface) scripting language was common as a server-side programming language. There are many interfaces and classes in the Servlet API such as Servlet, GenericServlet, HttpServlet, HttpServletRequest, HttpServletResponse, etc.

What is a Servlet?

Servlet can be described in many ways, depending on the context.

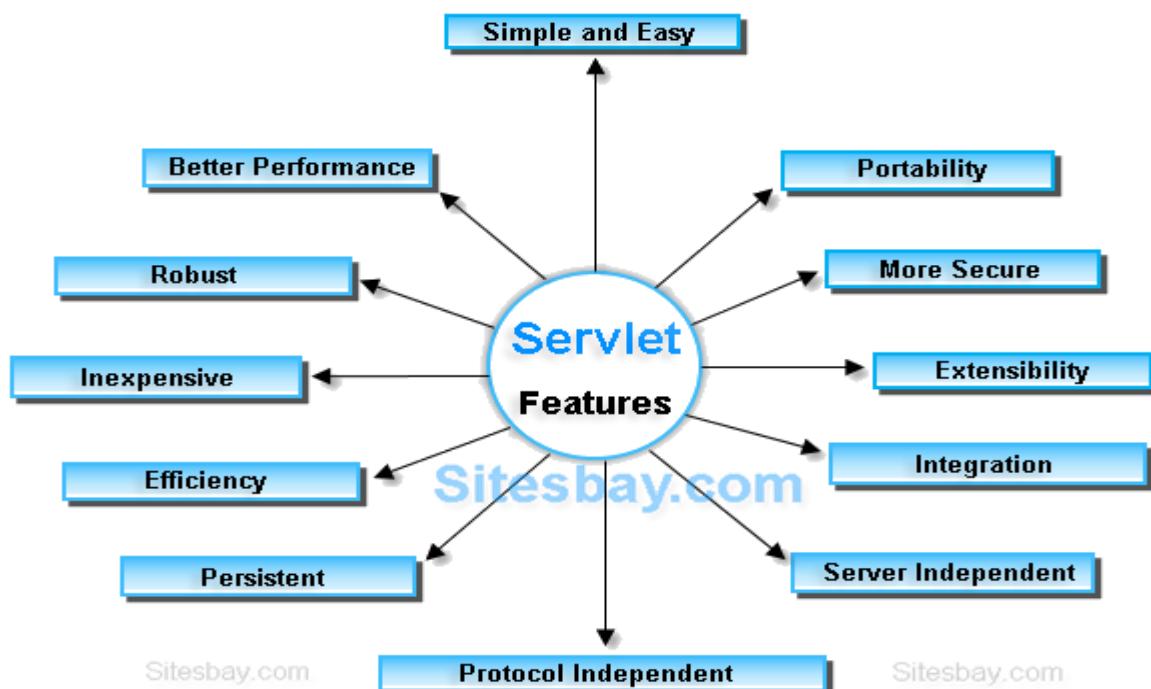
- Servlet is a technology which is used to create a web application.
- Servlet is an API that provides many interfaces and classes including documentation.
- Servlet is an interface that must be implemented for creating any Servlet.
- Servlet is a class that extends the capabilities of the servers and responds to the incoming requests. It can respond to any requests.
- Servlet is a web component that is deployed on the server to create a dynamic web page.



What is a web application?

A web application is an application accessible from the web. A web application is composed of web components like Servlet, JSP, Filter, etc. and other elements such as HTML, CSS, and JavaScript. The web components typically execute in Web Server and respond to the HTTP request.

Features of Servlet



Better performance: Because it creates a thread for each request not process (like CGI).

Portability: Because it uses java language and java is robust language.

Robust: Servlet are managed by JVM so no need to worry about memory leak, garbage collection etc.

Secure: Because it uses java language and java is a secure language. Java have automatic garbage collection mechanism and a lack of pointers protect the servlets from memory management problems.

Inexpensive There are number of free web servers available for personal use or for commercial purpose. Mostly web server are very costly. So by using free web server you can reduce project development price.

Extensibility The servlet API is designed in such a way that it can be easily extensible.

Efficiency Servlets invocation is highly efficient as compared to any CGI programs.

Integration Servlets are tightly integrated with the server. Servlet can use the server to translate the file paths, check authorization, perform logging and MIME type mapping etc.

Persistent: Servlets remain in memory until explicitly destroyed. This helps in serving several incoming requests. Servlets establishes connection only once with the database and can handle several requests on the same database.

Server Independent: Servlets are compatible with any web server available today.

Protocol Independent: Servlets can be created to support any protocols like FTP commands, Telnet sessions, NNTP newsgroups, etc. It also provides extended support for the functionality of HTTP protocol.

Fast: Since servlets are compiled into bytecodes, they can execute more quickly as compared to other scripting languages.

What is Servlet API

Servlet API provides Classes and Interface to develop web based applications.

Package

Servlet API contains two java packages are used to develop the servlet programs, they are:

- javax.servlet
- javax.servlet.http

javax.servlet

javax.servlet package contains list of interfaces and classes that are used by the servlet or web container. These classes and interface are not specific to any protocol.

javax.servlet.http

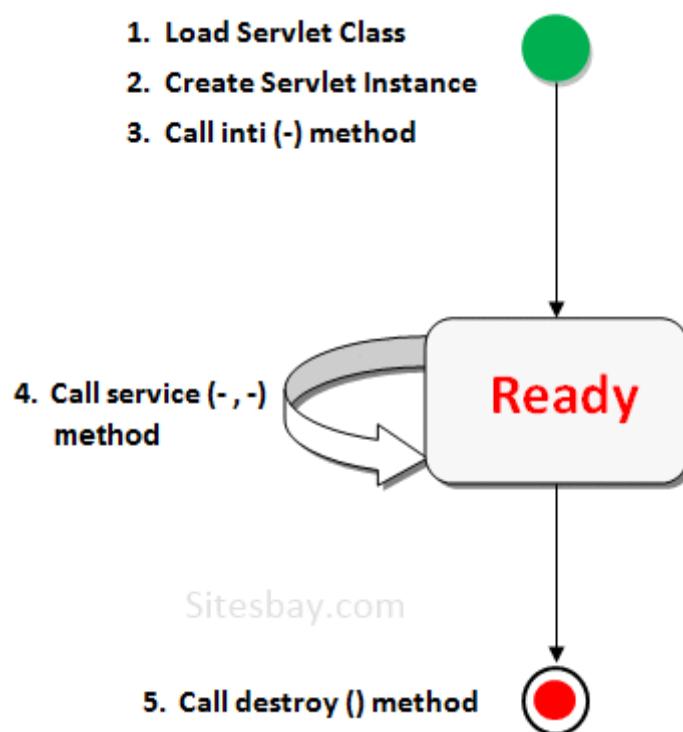
javax.servlet.http package contains list of classes and interfaces to define http servlet programs. This package is used to interact with browser using http protocol. It is only responsible for http requests.

Life Cycle of Servlet

The web container maintains the life cycle of a Servlet instance or object.

Life cycle of Servlet

1. Loading (Servlet class is loaded)
2. Instantiation (Servlet instance is created)
3. Initialization (init method is invoked)
4. Service providing (service method is invoked)
5. Destroying (destroy method is invoked)



As displayed in the above diagram, there are three states of a Servlet: new, ready and end. The Servlet is in new state if Servlet instance is created. After invoking the init() method, Servlet comes in the ready state. In the ready state, Servlet performs all the tasks. When the web container invokes the destroy() method, it shifts to the end state.

1. Servlet class is loaded

The classloader is responsible to load the Servlet class. The Servlet class is loaded when the first request for the Servlet is received by the web container.

2. Servlet instance is created

The web container creates the instance of a Servlet after loading the Servlet class. The Servlet instance is created only once in the Servlet life cycle.

3. init method is invoked

The web container calls the init method only once after creating the Servlet instance. The init method is used to initialize the Servlet. It is the life cycle method of the javax.servlet.Servlet interface. Syntax of the init method is given below:

Syntax

```
public void init(ServletConfig config) throws ServletException
```

4. service method is invoked

The web container calls the service method each time when request for the Servlet is received. If Servlet is not initialized, it follows the first three steps as described above then calls the service method. If Servlet is initialized, it calls the service method. Notice that Servlet is initialized only once. The syntax of the service method of the Servlet interface is given below:

Syntax

```
public void service(ServletRequest request, ServletResponse response)
throws ServletException, IOException
```

5. destroy method is invoked

The web container calls the destroy method before removing the Servlet instance from the service. It gives the Servlet an opportunity to clean up any resource for

example memory, thread etc. The syntax of the destroy method of the Servlet interface is given below:

Syntax

```
public void destroy()
```

Chapter 06

SERVLET

Lesson No: 2

Container in Servlet:

Container provides runtime environment for Java2ee (j2ee) applications. A web container is a predefined application provided by a server, its takes care of Servlet and JSP.

Operations of Servlet Container.

- Life Cycle Management
- Communication Support
- Multithreaded support
- Security etc.

1. Life cycle management:

The Servlet or JSP will run on a server and at server side. A container will take care about life and death of a Servlet or JSP. A container will instantiate, Initialize, Service and destroy of a Servlet or JSP. It means life cycle will be managed by a container.

2. Communication Support: If Servlet or JSP wants to communicate with server than its need some communication logic like socket programming. Designing communication logic is increase the burden on programmers, but container act as a mediator between a server and a Servlet or JSP and provides communication between them.

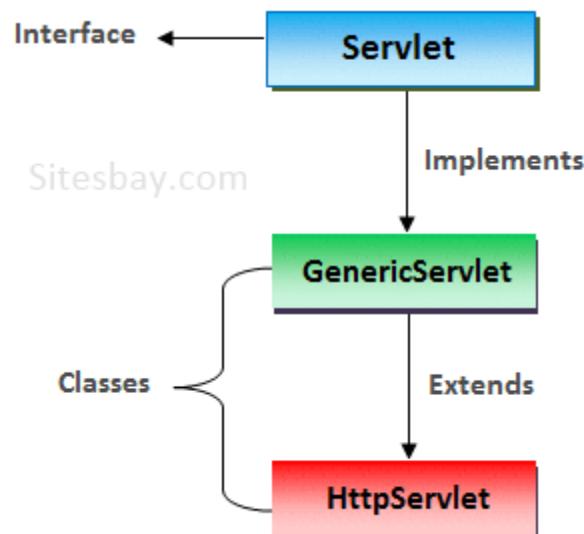
3. Multithreading: A container creates a thread for each request, maintains the thread and finally destroys it whenever its work is finished.

4. Security: A programmer is not required to write security code in a Servlet/JSP. A container will automatically provide security for a Servlet/JSP.

How to Create Servlet

According to servlet API we have three ways to creating a servlet class.

- By implementing servlet interface
- By extending GenericServlet class
- By extending HttpServlet class



By implementing servlet interface

Example

```
public class myServlet implements Servlet  
.....  
.....  
}
```

By extending GenericServlet class

Example

```
public class myServlet extends GenericServlet  
.....  
.....  
}
```

By extending HttpServlet class

Example

```
public class myServlet extends HttpServlet  
.....  
.....  
}
```

1. GenericServlet class

implements **Servlet**, **ServletConfig** and **Serializable** interfaces. It provides the implementation of all the methods of these interfaces except the service method.

GenericServlet class can handle any type of request so it is protocol-independent.

You may create a generic servlet by inheriting the GenericServlet class and providing the implementation of the service method.

Methods of GenericServlet class

There are many methods in GenericServlet class. They are as follows:

1. **public void init(ServletConfig config)** is used to initialize the servlet.
2. **public abstract void service(ServletRequest request, ServletResponse response)** provides service for the incoming request. It is invoked at each time when user requests for a servlet.
3. **public void destroy()** is invoked only once throughout the life cycle and indicates that servlet is being destroyed.
4. **public ServletConfig getServletConfig()** returns the object of ServletConfig.

a) ServletRequest class

True job of a Servlet is to handle client request. Servlet API provides an important interfaces **javax.servlet.ServletRequest** to encapsulate client request.

b) ServletResponse class

Servlet API provides an important interfaces **ServletResponse** to assist in sending response to client.

Methods of ServletResponse :

PrintWriter <code>getWriter()</code>	returns a PrintWriter object that can send character text to the client.
void <code>setContentType(String type)</code>	sets the content type of the response being sent to the client before sending the respond.

c) ServletConfig interface

When the **Web Container** initializes a servlet, it creates a **ServletConfig** object for the servlet. ServletConfig object is used to **pass information to a servlet during initialization by getting configuration information from web.xml(Deployment Descriptor)**.

Sample Code

Following is the sample source code structure of a servlet example to show Hello World –

```
// Import required java libraries
import java.io.*;
import javax.servlet.*;

// Extend GenericServlet class
public class HelloWorld extends GenericServlet {

    ;

    public void init() throws ServletException {
        // Do required initialization
    }

    public void service(ServletRequest request, ServletResponse response)
        throws ServletException, IOException {

        // Set response content type
        response.setContentType("text/html");

        // Actual logic goes here.
        PrintWriter out = response.getWriter();
        out.println("<h1>" + "Hello World" + "</h1>");
    }

    public void destroy() {
        // do nothing.
    }
}
```

Steps to create a servlet Application

There are given 6 steps to create a **Servlet example**. These steps are required for all the servers.

The servlet example can be created by three ways:

1. By implementing Servlet interface,
2. By inheriting GenericServlet class, (or)
3. By inheriting HttpServlet class

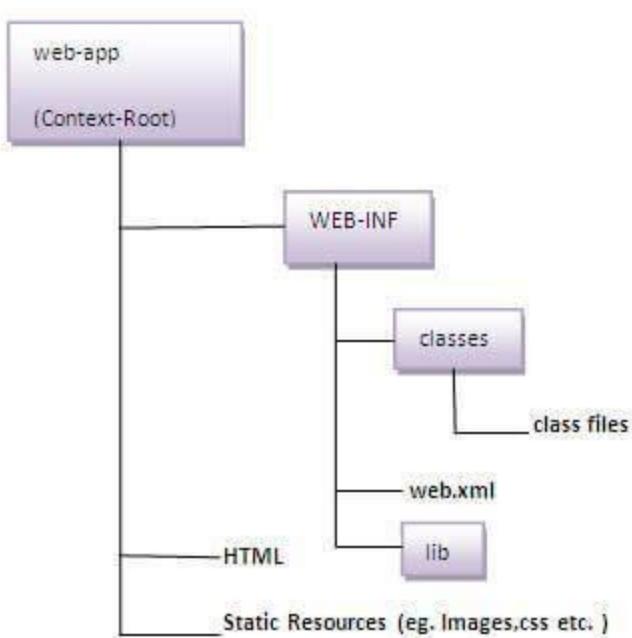
Here, we are going to use **apache tomcat server** in this example. The steps are as follows:

- 1. Create a directory structure**
- 2. Create a Servlet**
- 3. Compile the Servlet**
- 4. Create a deployment descriptor**
- 5. Start the server and deploy the project**
- 6. Access the servlet**

1) Create a directory structures

The **directory structure** defines that where to put the different types of files so that web container may get the information and responds to the client.

The Sun Microsystem defines a unique standard to be followed by all the server vendors. Let's see the directory structure that must be followed to create the servlet.



As you can see that the servlet class file must be in the classes folder. The web.xml file must be under the WEB-INF folder

2) Create a Servlet

In this example we are going to create a servlet that extends the GenericServlet class.

```

import javax.servlet.*; import java.io.*;

public class DemoServlet extends GenericServlet{

    public void service(ServletRequest req, ServletResponse res)
    throws ServletException, IOException
    {
        res.setContentType("text/html");//setting the content type
        PrintWriter pw=res.getWriter();//get the stream to write the data

        //writing html in the stream
        pw.println("<html><body>");
    }
}

```

```

pw.println("Welcome to servlet");
pw.println("</body></html>");

pw.close(); //closing the stream
}}
```

3) Compile the servlet

For compiling the Servlet, jar file is required to be loaded. Different Servers provide different jar files:

Jar file	Server
1) servlet-api.jar	Apache Tomcat
2) weblogic.jar	Weblogic

Two ways to load the jar file

1. set classpath
2. paste the jar file in JRE/lib/ext folder

Put the java file in any folder. After compiling the java file, paste the class file of servlet in **WEB-INF/classes** directory.

4) Create the deployment descriptor (web.xml file)

The **deployment descriptor** is an xml file, from which Web Container gets the information about the servet to be invoked.

The web container uses the Parser to get the information from the web.xml file. There are many xml parsers such as SAX, DOM and Pull.

There are many elements in the web.xml file. Here is given some necessary elements to run the simple servlet program.

web.xml file

1. **<web-app>**
- 2.
3. **<servlet>**
4. **<servlet-name>First</servlet-name>**
5. **<servlet-class>DemoServlet</servlet-class>**
6. **</servlet>**
- 7.
8. **<servlet-mapping>**
9. **<servlet-name>First</servlet-name>**
10. **<url-pattern>/welcome</url-pattern>**
11. **</servlet-mapping>**
- 12.
13. **</web-app>**

Description of the elements of web.xml file

There are too many elements in the web.xml file. Here is the illustration of some elements that is used in the above web.xml file. The elements are as follows:

<web-app> represents the whole application.

<servlet> is sub element of <web-app> and represents the servlet.

<servlet-name> is sub element of <servlet> represents the name of the servlet.

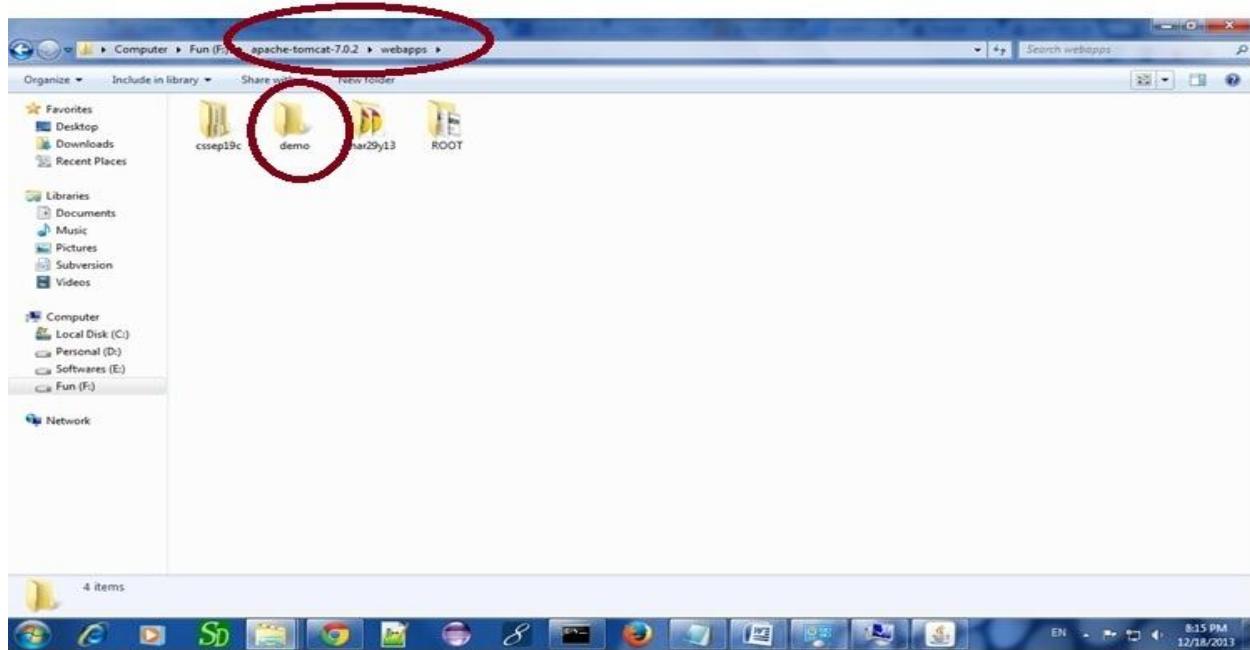
<servlet-class> is sub element of <servlet> represents the class of the servlet.

<servlet-mapping> is sub element of <web-app>. It is used to map the servlet.

<url-pattern> is sub element of <servlet-mapping>. This pattern is used at client side to invoke the servlet.

5) How to deploy the servlet project

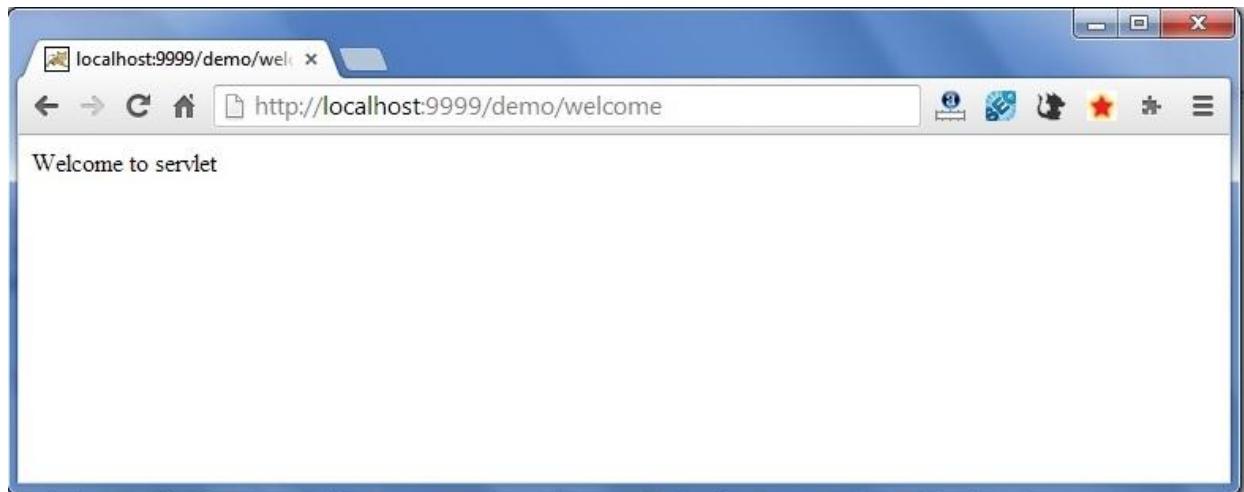
Copy the project and paste it in the webapps folder under apache tomcat.



6) How to access the servlet

Open broser and write <http://hostname:portno/contextroot/url> pattern of servlet. For example:

<http://localhost:9999/demo/welcome>



UNIT - 5 (Servlet)

1

```

// Web application 2
1 <HTML>
2 <BODY BGCOLOR="yellow">
3 <CENTER>
4 <H1> USER NAME ENTRY SCREEN</H1>
5 <FORM ACTION="http://localhost:8081/greetingapplication/greet">
6 USER NAME<INPUT TYPE="text" NAME="t1"><BR><BR>
7 <INPUT TYPE="submit" VALUE="get message"> Request type parameter
8 </FORM>
9 </CENTER>
10 </BODY>
11 </HTML>
12 -----
13 -----
14 import javax.servlet.*;
15 import java.io.*;
16 public class GreetingServlet extends GenericServlet
17 {
18     public void service(ServletRequest request, ServletResponse response)
19         throws ServletException, IOException
20     {
21         String name = request.getParameter("t1");
22         response.setContentType("text/html");
23         PrintWriter pw = response.getWriter();
24         pw.println("<HTML>");
25         pw.println("<BODY BGCOLOR=wheat>");
26         pw.println("<H1>HELLO !"+name+" WELCOME TO OUR WEBSITE</H1>");
27         pw.println("</BODY>");
28         pw.println("</HTML>");
29         pw.close();
30     } // service
31 }
32 -----
33 <web-app>
34     <servlet>
35         <servlet-name>two</servlet-name>
36         <servlet-class>GreetingServlet</servlet-class>
37     </servlet>
38     <servlet-mapping>
39         <servlet-name>two</servlet-name>
40         <url-pattern>/greet</url-pattern>
41     </servlet-mapping>
42 </web-app>
43 ****

```

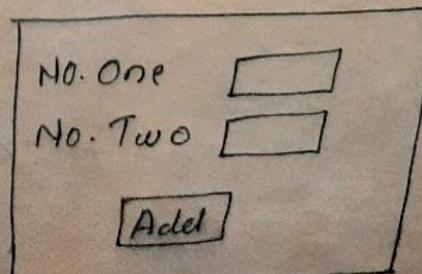
// Web application 3

```

44 <HTML>
45 <BODY BGCOLOR="yellow">
46 <CENTER>
47 <H1> NUMBERS INPUT SCREEN</H1>
48 <FORM ACTION=".add">
49 NUMBER ONE<INPUT TYPE="text" NAME="t1"><BR><BR>
50 NUMBER TWO<INPUT TYPE="text" NAME="t2"><BR><BR>
51 <INPUT TYPE="submit" VALUE="add">
52 </FORM>
53 </CENTER>
54 </BODY>
55 </HTML>
56 -----
57 -----
58 import javax.servlet.*;
59 import java.io.*;
60 public class AddingServlet extends GenericServlet
61 {

```

⇒ adding applicatⁿ
 numbers.html
WEB-INF
 web.xml
 classes-
 AddingServ.6
lib



2

```
62     public void service(ServletRequest request,ServletResponse response)
63         throws ServletException,IOException
64     {
65         String a=request.getParameter("t1");
66         String b=request.getParameter("t2");
67         int n1=Integer.parseInt(a);
68         int n2=Integer.parseInt(b);
69         int sum=n1+n2;
70         response.setContentType("text/html");
71         PrintWriter pw=response.getWriter();
72         pw.println("<HTML>");
73         pw.println("<BODY BGCOLOR=wheat>");
74         pw.println("<H1>THE SUM OF TWO NUMBERS:"+sum+"</H1>");
75         pw.println("</BODY>");
76         pw.println("</HTML>");
77         pw.close();
78     }//service
79 }
80 -----
81 <web-app>
82   <servlet>
83     <servlet-name>three</servlet-name>
84     <servlet-class>AddingServlet</servlet-class>
85   </servlet>
86   <servlet-mapping>
87     <servlet-name>three</servlet-name>
88     <url-pattern>/add</url-pattern>
89   </servlet-mapping>
90 </web-app>
```

✓ Performing database operations in servlets

While a servlet is communicating with the databases the following observations are made.

- ✓ 1. Driver class, database URL, user name and password don't hard code in the servlet. Get them from the web.xml as initialization parameters. As a result our servlet code will not change even when the database server, the driver or the authentication information is changed.
2. Almost all the times make use of the PreparedStatement object to perform database operations.]
- 3. Override zero argument init method. In the init method, establish the database connection and build the PreparedStatement object.
- 4. If the web form is submitted for retrieving data from the database, i.e. if the client request is GET request, override the doGet method in the servlet.
- 5. If the web form is submitted for making changes in the database, i.e. if the client request is POST request, override the doPost method in the servlet.
- 6. Within the doGet or doPost method perform database operations. I.e. executing the PreparedStatement for submitting the SQL statement to the database.
- 7. Close the PreparedStatement and database connection in the destroy method.
- 8. Implement exception handling explicitly in init(), doGet() or doPost() and destroy() methods. ClassNotFoundException and SQLException can't be added to the throws clause of these methods as it violates method overriding rule.

★ Supplying Initialization Parameters

In the web.xml the following elements are used to supply initialization parameters to the servlets.

```
<init-param>
    <param-name>driver</param-name>
    <param-value>sun.jdbc.odbc.JdbcOdbcDriver</param-value>
</init-param>
```

During servlet initialization phase the servlet engine encapsulates these name value pairs of strings in the ServletConfig object and passes them to the servlet. Within the servlet we retrieve the initialization parameter value by calling getInitParameter method.

```

//Web application 4
<HTML><BODY BGCOLOR="cyan">
<CENTER>
<H1> EMPLOYEE DETAILS INPUT SCREEN</H1>
<FORM ACTION=".store">
EMPLOYEE NUMBER<INPUT TYPE="text" NAME="t1"><BR><BR>
EMPLOYEE NAME<INPUT TYPE="text" NAME="t2"><BR><BR>
EMPLOYEE SALARY<INPUT TYPE="text" NAME="t3"><BR><BR>
<INPUT TYPE="submit" VALUE="store">
</FORM></CENTER></BODY></HTML>

import javax.servlet.*;import java.io.*;import java.sql.*;
public class StoringServlet extends GenericServlet
{
    Connection con;
    public void init(ServletConfig config) throws ServletException
    {
        try
        {
            Class.forName("oracle.jdbc.driver.OracleDriver");
            String cs="jdbc:oracle:thin:@localhost:1521:server";
            con=DriverManager.getConnection(cs,"scott","tiger");
        }
        catch(Exception e){ e.printStackTrace(); }
    }

    public void service(ServletRequest request,ServletResponse response)
    throws ServletException,IOException
    {
        response.setContentType("text/html");
        PrintWriter pw=response.getWriter();
        pw.println("<HTML>");
        pw.println("<BODY BGCOLOR=wheat>");
        int eno=Integer.parseInt(request.getParameter("t1"));
        String nm=request.getParameter("t2"); ↑(String) if name is char in DB
        float sal=Float.parseFloat(request.getParameter("t3"));
        String sql="INSERT INTO EMPLOYEE VALUES("+eno+","+nm+","+sal+ ")";
        try
        {
            Statement st=con.createStatement();
            st.executeUpdate(sql);
            pw.println("<H1> EMPLOYEE DETAILS STORED SUCCESSFULLY</H1>");
            st.close();
        }
        catch(SQLException e)
        {
            e.printStackTrace();
        }
        pw.println("</BODY>");
        pw.println("</HTML>");
        pw.close();
    }
    public void destroy()
    {
        try { con.close(); }
        catch(SQLException e){ e.printStackTrace(); }
    }
}

<web-app>
<servlet>
    <servlet-name>four</servlet-name>
    <servlet-class>StoringServlet</servlet-class>
</servlet>
<servlet-mapping>
    <servlet-name>four</servlet-name>
    <url-pattern>/store</url-pattern>
</servlet-mapping>
</web-app>
*****  

//Web application 5
<HTML><BODY BGCOLOR="cyan">
```

3

Emp No.	<input type="text"/>
Emp Name	<input type="text"/>
Emp Sal	<input type="text"/>
<input type="button" value="Submit"/>	

```

72 <CENTER>
73 <H1> EMPLOYEE DETAILS RETRIEVING SCREEN</H1>
74 <FORM ACTION="/retrieve">
75 EMPLOYEE NUMBER<INPUT TYPE="text" NAME="t1"><BR><BR>
76 <INPUT TYPE="submit" VALUE="get details">
77 </FORM></CENTER></BODY></HTML>
78 -----
79 import javax.servlet.*;import java.io.*;import java.sql.*;
80 public class RetrievingServlet extends GenericServlet
81 {
82     Connection con;
83     public void init(ServletConfig config) throws ServletException
84     {
85         try
86         {
87             Class.forName("oracle.jdbc.driver.OracleDriver");
88             String cs="jdbc:oracle:thin:@localhost:1521:server";
89             con=DriverManager.getConnection(cs,"scott","tiger");
90         }
91         catch(Exception e){ e.printStackTrace(); }
92     }
93     public void service(ServletRequest request,ServletResponse response)
94         throws ServletException,IOException
95     {
96         response.setContentType("text/html");
97         PrintWriter pw=response.getWriter();
98         pw.println("<HTML>");
99         pw.println("<BODY BGCOLOR=wheat>");
100        int eno=Integer.parseInt(request.getParameter("t1"));
101        try
102        {
103            Statement st=con.createStatement();
104            ResultSet rs=st.executeQuery("SELECT * FROM EMPLOYEE WHERE EMPNO="+eno);
105            if(rs.next())
106            {
107                pw.println("<H1> EMPLOYEE DETAILS</H1>");
108                pw.println("<H1>EMPNO:"+eno+"</H1>");
109                pw.println("<H1>NAME:"+rs.getString(2)+"</H1>");
110                pw.println("<H1>SALARY:$ "+rs.getDouble(3)+"</H1>");
111            }
112            else
113                pw.println("<H1> EMPLOYEE NOT FOUND</H1>");
114            rs.close();
115            st.close();
116        }
117        catch(SQLException e)
118        {
119            e.printStackTrace();
120        }
121        pw.println("</BODY>");
122        pw.println("</HTML>");
123        pw.close();
124    }//service
125    public void destroy()
126    {
127        try { con.close(); }
128        catch(SQLException e){ e.printStackTrace(); }
129    }
130 }
131 -----
132 <web-app>
133     <servlet>
134         <servlet-name>five</servlet-name>
135         <servlet-class>RetrievingServlet</servlet-class>
136     </servlet>
137     <servlet-mapping>
138         <servlet-name>five</servlet-name>
139         <url-pattern>/retrieve</url-pattern>
140     </servlet-mapping>
141 </web-app>

```

Emp No.

GreetingServlet.class

In order to deploy the web application, copy the root directory along with all the resources and helper files into the "webapps" directory of Tomcat installation directory and start the Tomcat.

Launch the browser and type the following URL.

<http://localhost:8080/greetingapp/user.html>

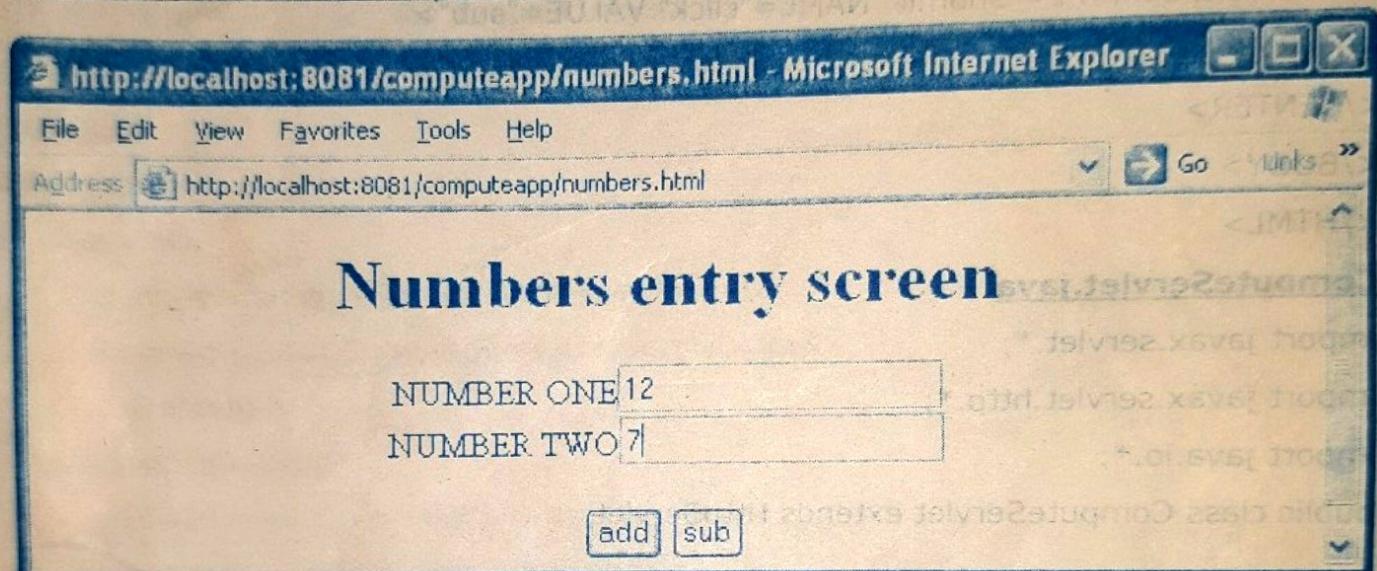


Q) Web application in which end-user should be able to enter 2 numbers into the web form. If add button is clicked, servlet should send the sum of two numbers to the client. If the subtract button is clicked the difference should be sent to the client.

Directory structure along with files

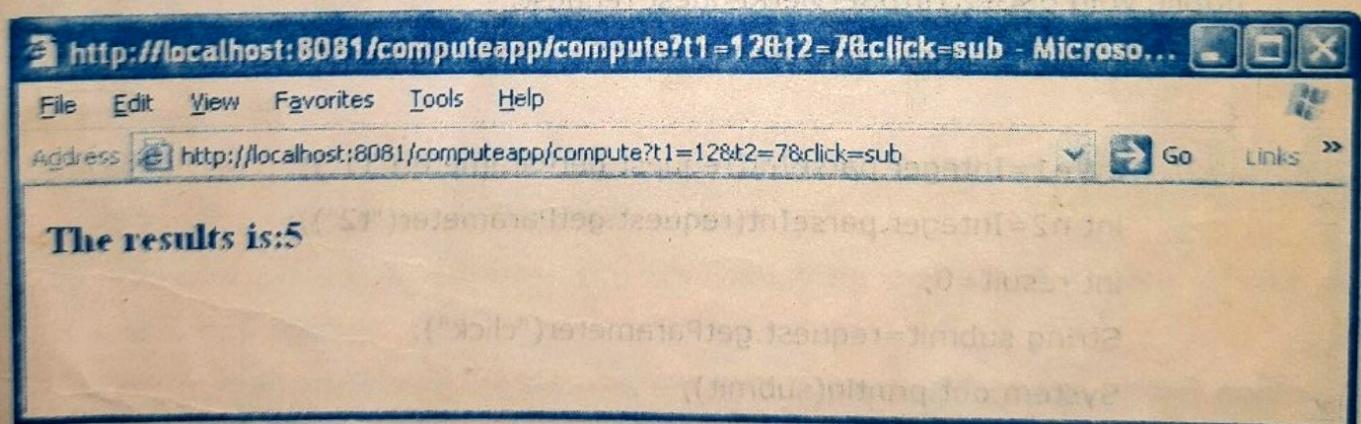
```
computeapp  
    numbers.html  
    WEB-INF  
        web.xml  
        classes  
            ComputeServlet.class
```

After deployment we have to type the following URL in the browser.
<http://localhost:8080/computeapp/numbers.html>



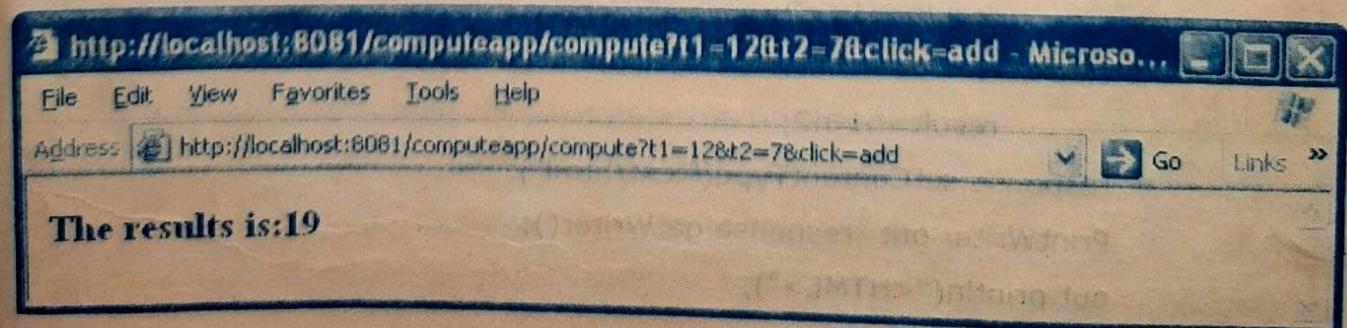
The screenshot shows a Microsoft Internet Explorer window with the title bar "http://localhost:8081/computeapp/numbers.html - Microsoft Internet Explorer". The menu bar includes File, Edit, View, Favorites, Tools, and Help. The address bar shows the URL "http://localhost:8081/computeapp/numbers.html". The main content area displays the text "Numbers entry screen". Below it, there are two input fields: "NUMBER ONE" containing "12" and "NUMBER TWO" containing "7". At the bottom are two buttons: "add" and "sub".

If the end user clicks on "sub" button, the following result appears.



The screenshot shows a Microsoft Internet Explorer window with the title bar "http://localhost:8081/computeapp/compute?t1=12&t2=7&click=sub - Microsoft Internet Explorer". The menu bar includes File, Edit, View, Favorites, Tools, and Help. The address bar shows the URL "http://localhost:8081/computeapp/compute?t1=12&t2=7&click=sub". The main content area displays the text "The results is:5".

If the clicked button is "add" the result is as follows.



The screenshot shows a Microsoft Internet Explorer window with the title bar "http://localhost:8081/computeapp/compute?t1=12&t2=7&click=add - Microsoft Internet Explorer". The menu bar includes File, Edit, View, Favorites, Tools, and Help. The address bar shows the URL "http://localhost:8081/computeapp/compute?t1=12&t2=7&click=add". The main content area displays the text "The results is:19".

After deployment <http://localhost:8080/postapp/emp.html> is typed into the browser.

The screenshot shows a Microsoft Internet Explorer window with the title bar "http://localhost:8080/postapp/emp.html - Microsoft Internet Explorer". The menu bar includes File, Edit, View, Favorites, Tools, and Help. The address bar shows the URL "http://localhost:8080/postapp/emp.html". The main content area displays a form titled "EMPLOYEE DETAILS". The form has three input fields: "EMPNO" with a value of "1000", "NAME" with a value of "John Doe", and "SALARY" with a value of "5000". Below the form is a button labeled "INSEPT".

EMPLOYEE DETAILS	
EMPNO	1000
NAME	John Doe
SALARY	5000

INSEPT

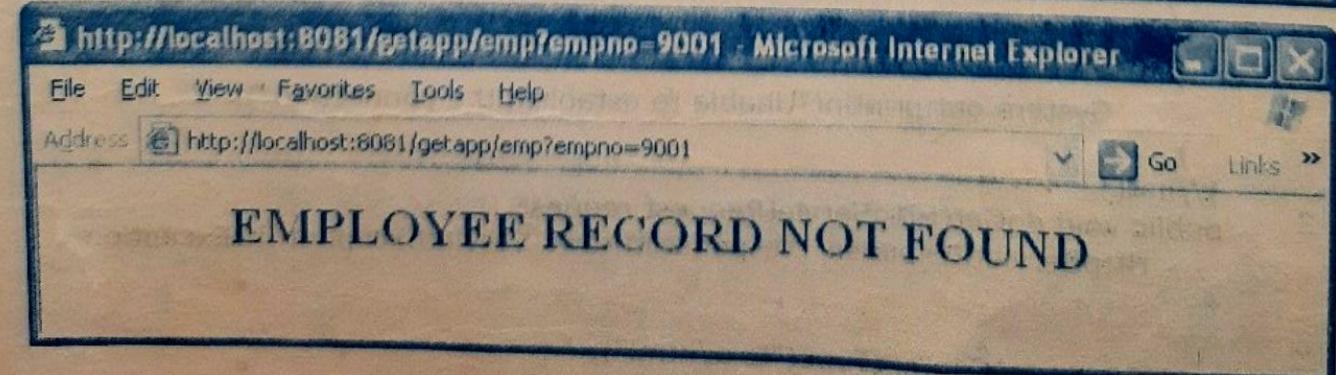
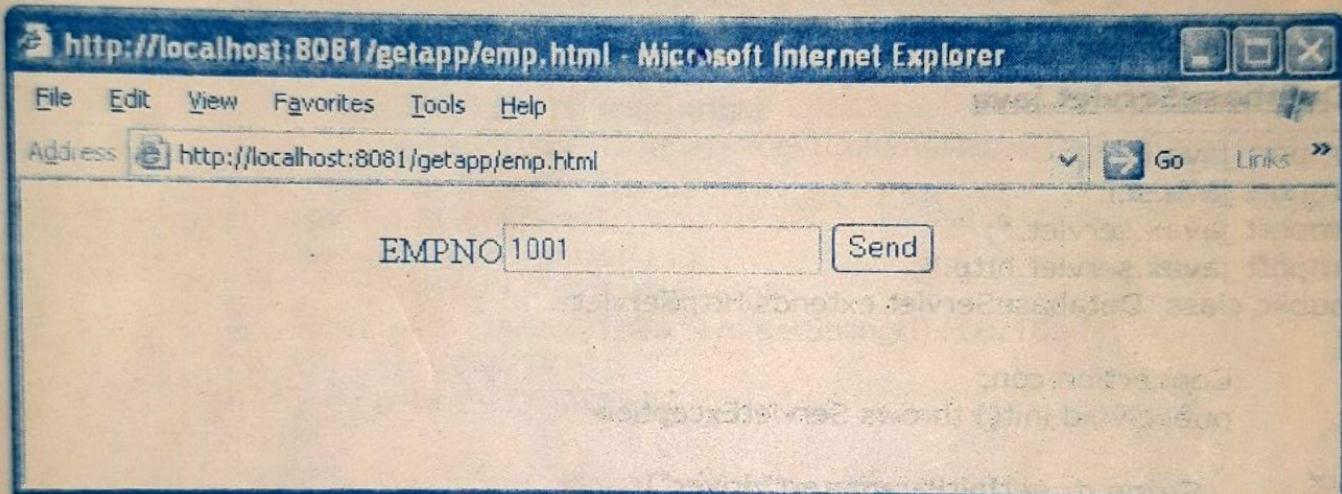
Q) Web application in which end-user enters the employee number into the web form. Servlet sends the employee details in HTML tabular format.

Directory structure

```

getapp
    emp.html
    WEB-INF
        web.xml
    classes
        DatabaseServlet.class
  
```

After deployment type the URL <http://localhost:8080/getapp/emp.html>



```

1 //Web application 6
2 <HTML>
3 <BODY BGCOLOR="cyan">
4 <CENTER>
5 <H1> ACCOUNT DETAILS RETRIEVING SCREEN</H1>
6 <FORM ACTION=".retrieve">
7 A/C NUMBER<INPUT TYPE="text" NAME="t1"><BR><BR>
8 <INPUT TYPE="submit" VALUE="get details">
9 </FORM>
10 </CENTER>
11 </BODY></HTML>
12 -----
13 import javax.servlet.*;import java.io.*;import java.sql.*;
14 public class RetrievingServlet extends GenericServlet
15 {
16     Connection con;
17     PreparedStatement ps;
18     public void init(ServletConfig config) throws ServletException
19     {
20         String driver=config.getInitParameter("p1");
21         String cs=config.getInitParameter("p2");
22         String user=config.getInitParameter("p3");
23         String pwd=config.getInitParameter("p4");
24         try
25         {
26             Class.forName(driver);
27             con=DriverManager.getConnection(cs,user,pwd);
28             ps=con.prepareStatement("SELECT * FROM ACCOUNT WHERE ACCNO=?");
29         }
30         catch(Exception e)
31         {
32             e.printStackTrace();
33         }
34     }
35     public void service(ServletRequest request,ServletResponse response)
36         throws ServletException,IOException
37     {
38         response.setContentType("text/html");
39         PrintWriter pw=response.getWriter();
40         pw.println("<HTML>");
41         pw.println("<BODY BGCOLOR=wheat>");
42         int ano=Integer.parseInt(request.getParameter("t1"));
43         try
44         {
45             ps.setInt(1,ano);
46             ResultSet rs=ps.executeQuery();
47             if(rs.next())
48             {
49                 pw.println("<H1> ACCOUNT DETAILS</H1>");
50                 pw.println("<H1>ACCOUNT NO:"+ano+"</H1>");
51                 pw.println("<H1>NAME:"+rs.getString(2)+" </H1>");
52                 pw.println("<H1>BALANCE:Rs." +rs.getFloat(3)+" </H1>");
53                 pw.println("<A HREF=account.html>ONE MORE ACCOUNT DETAILS ?</A>");
```

[]

```

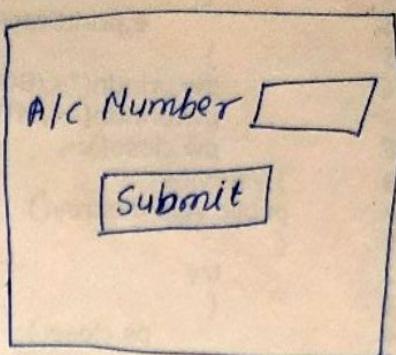
54             }
55             else
56             {
57                 pw.println("<H1> ACCOUNT NOT FOUND</H1>");
58                 pw.println("<A HREF=account.html>TRY AGAIN</A>");
```

]

```

59             }
60             rs.close();
61         }
62     }

```



22-8-08

6
62 catch(SQLException e)
63 {
64 e.printStackTrace();
65 }
66 pw.println("</BODY>");
67 pw.println("</HTML>");
68 pw.close();
69 }//service
70 public void destroy()
71 {
72 try
73 {
74 ps.close();
75 con.close();
76 }
77 catch(SQLException e)
78 {
79 e.printStackTrace();
80 }
81 }
82 }
83 -----
84 <web-app>
85 → <servlet>
86 <servlet-name>six</servlet-name>
87 <servlet-class>RetrievingServlet</servlet-class>
88 <init-param>
89 <param-name>p1</param-name>
90 <param-value>oracle.jdbc.driver.OracleDriver</param-value>
91 </init-param>
92 <init-param>
93 <param-name>p2</param-name>
94 <param-value>jdbc:oracle:thin:@localhost:1521:server</param-value>
95 </init-param>
96 <init-param>
97 <param-name>p3</param-name>
98 <param-value>scott</param-value>
99 </init-param>
100 <init-param>
101 <param-name>p4</param-name>
102 <param-value>tiger</param-value>
103 </init-param>
104 → </servlet>
105 <servlet-mapping>
106 <servlet-name>six</servlet-name>
→ 107 <url-pattern>/retrieve</url-pattern>
108 </servlet-mapping>
109 </web-app>
110 *****

* Login Application *

9

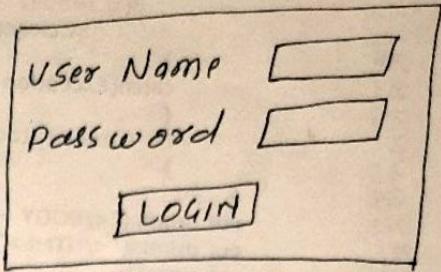
```

139 //login.html
140 <HTML><BODY BGCOLOR=cyan><CENTER> <H2> LOG IN TO OUR WEBSITE </H2>
141     <FORM ACTION="/authenticate" METHOD="post">
142         USER NAME<INPUT TYPE="text" NAME="user"> <BR>
143         PASSWORD<INPUT TYPE="password" NAME="pwd"> <BR>
144         <INPUT TYPE="submit" VALUE=" click here to login " >
145     </FORM>
146 </CENTER></BODY></HTML>
*****  

147 //web.xml
148 <web-app>
149     <servlet>
150         <servlet-name>login</servlet-name>
151         <servlet-class>AuthenticationServlet</servlet-class>
152     </servlet>
153     <servlet-mapping>
154         <servlet-name>login</servlet-name>
155         <url-pattern>/authenticate </url-pattern>
156     </servlet-mapping>
157 </web-app>
*****  

158 import java.io.*;import java.sql.*;import javax.servlet.*;import javax.servlet.http.*;  

159 public class AuthenticationServlet extends HttpServlet
160 {
161     Connection con;
162     public void init(ServletConfig config) throws ServletException
163     {
164         try
165         {
166             Class.forName("oracle.jdbc.driver.OracleDriver");
167             String url="jdbc:oracle:thin:@localhost:1521:ordl";
168             con=DriverManager.getConnection(url,"scott","tiger");
169         }
170         catch(Exception e)
171         {
172             e.printStackTrace();
173         }
174     } //init
175     public void doPost(HttpServletRequest req, HttpServletResponse res)
176         throws ServletException, IOException
177     {
178         Statement st=null;
179         ResultSet rs=null;
180         String user=req.getParameter("user");
181         String pwd=req.getParameter("pwd");
182         res.setContentType("text/html");
183         PrintWriter out=res.getWriter();
184         out.println("<HTML>");
185         out.println("<BODY BGCOLOR=wheat>");
186         try
187         {
188             st=con.createStatement();
189             String sql="SELECT * FROM OURUSERS WHERE usr='"+user+"' and password='"+pwd+"'";
190             rs=st.executeQuery(sql);
191             if(rs.next())
192                 out.println("<H2>WELCOME "+user+" TO OUR WEB SITE</H2>");
193             else
194                 {
195                     out.println("<H2>INVALID USER OR PASSWORD</H2>");
196                     out.println("<A HREF=/login.html> LOGIN AGAIN</A>");
197                 }
198         }
199         catch(Exception e)
200         {
201             e.printStackTrace();
202         }
203     }
204     finally
205     {
206     }
207 }
```



10

```
205     try
206     {
207         if(rs !=null)
208             rs.close();
209         if(st !=null)
210             st.close();
211     }
212     catch(Exception e)
213     {
214         e.printStackTrace();
215     }
216     out.println("</BODY >");
217     out.println("</HTML>");
218     out.close();
219 } //doPost()
220 public void destroy()
221 {
222     try
223     {
224         if(con !=null)
225             con.close();
226     }
227     catch(Exception e) { e.printStackTrace(); }
228 }
```

→ **Session Tracking**

Http is a stateless protocol. A web client opens a connection with the http server and requests some resource. The server responds with the requested resource and closes the connection with client. After closing the connection, the http server does not remember any information about the client. The server considers the next request from the same client as a fresh request, with no relation to the previous request. This is what the stateless nature of the Http protocol.

In enterprise web application it is mandatory that client and its associated data must be tracked at server side across multiple requests. There are four approaches to achieve this.

- Hidden form fields
- Cookies
- Session tracking with SERVLET API
- ↗ URL rewriting

Note: For Hidden form fields mechanism there is no SERVLET API support available.

→ **Hidden form fields:** - A hidden field is similar to an ordinary input field in HTML. The only difference is that the hidden field doesn't have an associated user-interface element. (When the form that contains these hidden fields is submitted, the values of these fields are sent with the request. On the server side, these values are received as request parameters. This mechanism works only when form is submitted, not when you click the hyperlink. The key point is that client selected data travels invisibly from client to server and server to client in this mechanism. That is how state is managed across multiple requests.) In J2EE environment this mechanism is not so widely used, as there is no SERVLETS API support available.

↗ **Cookies Mechanism**

This is a widely used mechanism for both state management and session management. A cookie is a simple place of information stored on the client, on behalf of the server. This information is returned to the server with every request, in addition to the requested document, if may choose to return some state information to the browser. This information includes a URL range within which the information should be returned to the server. The URL range comprises the domain name and some path within the domain. Whenever the browser requests a resource. It checks the URL against the URL range of all available cookies. If a match is found, the cookie is also returned with the request. This helps the server overcome the stateless nature of the Http protocol.

Using the cookie mechanism in a Servlet involves the following steps.

✓ Step 1: - creating the instance of `javax.Servlet.http.Cookie` class.

```
Cookie c=new Cookie ("user", "rama");
```

✓ Step 2: - Making the cookie persistent

```
c.setMaxAge (3600);
```

✓ Step 3: - Writing the cookie to the response header.

```
response.addCookie(c);
```

Step 4: To retrieve the cookie we call the following method.

```
Cookie c[] = request.getCookies();
```

List of important methods in cookie class:

- `public String getName():-` Returns the name of the cookie.
- `public String getValue():-` Returns the value of the cookie.
- `public void setMaxAge(int expiry):-` Sets the maximum age of the cookie in seconds. A positive value indicates that the cookie will expire after that many seconds have passed, note that the value is the maximum age when the cookie will expire, not the cookie's current age. A negative value means that the cookie is not stored persistently and will be deleted when the web browser exits. A zero value causes the cookie to be deleted.
- `public void setDomain(String pattern):-` Specifies the domain within which this cookies should be presented. By default, cookies are only returned to the server that sent them.

Q) Web Application cookies implementation

Directory Structure

```

Cookieapp
  Cookieexample.html
  WEB-INF
    web.xml
    classes
      CreateCookie.class
      CheckCookie.class

```

After the web application is deployed, the following URL has to be typed into browser.

`http://localhost: 8080/cookieapp/cookieexample.html`



- ✓ When the end-user enters the name and clicks on the button, the first servlet retrieves the name, convert it into a cookie, and send to the client. When the end-user clicks on the hyper link , the cookie is sent to the web server again.

Cookieexample.html

```
<HTML>
<BODY BGCOLOR="cyan">
<CENTER>
<H2> WELCOME TO SHOPPING MALL </H2>
<FORM ACTION="/create" METHOD="post">
<B>UserName</B> <INPUT TYPE="text NAME="user"><br><br>
<INPUT TYPE="submit" VALUE="WELCOME"><br><br>
</FORM>
</CENTER>
</BODY>
</HTML>
```

CreateCookie.java

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class CreateCookie extends HttpServlet
{
```

```

public void doPost(HttpServletRequest req, HttpServletResponse res)
    throws ServletException, IOException
{
    String name=req.getParameter("user");
    res.setContentType("text/html");
    PrintWriter pw=res.getWriter();
    →Cookie c=new Cookie("user",name);
    →res.addCookie(c);      t1   n
    pw.println("<HTML>");
    pw.println(" <BODY BGCOLOR=wheat><CENTER>");
    pw.println("<H2><A HREF=./check>SHOPPING GOES HERE</A></H2>");
    pw.println("</CENTER></BODY><HTML>");
    pw.close();
}

```

CheckCookie.java

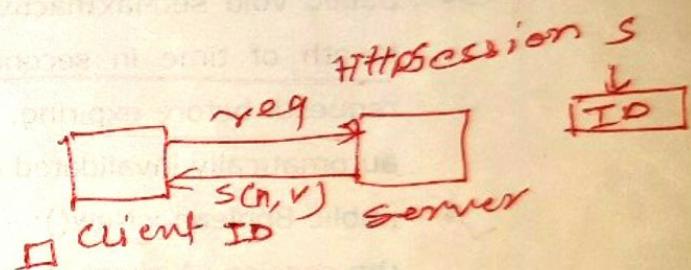
```

import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class CheckCookie extends HttpServlet
{
    public void doGet(HttpServletRequest req, HttpServletResponse res)
        throws ServletException, IOException
    {
        res.setContentType("text/html");
        PrintWriter pw=res.getWriter();
        →Cookie c[]=req.getCookies();
        pw.println("<HTML><BODY BGCOLOR=wheat><H2>");
        pw.println(" Hai "+c[0].getValue()+" ! hope enjoying
                  shopping here </H2>");
        pw.println("</BODY>");
        pw.println("</HTML>");
        pw.close();
    }//method
}//class

```

web.xml

```
<web-app>
  <servlet>
    <servlet-name>create</servlet-name>
    <servlet-class>CreateCookie</servlet-class>
  </servlet>
  <servlet>
    <servlet-name>check</servlet-name>
    <servlet-class>CheckCookie</servlet-class>
  </servlet>
  <servlet-mapping>
    <servlet-name>create</servlet-name>
    <url-pattern>/create</url-pattern>
  </servlet-mapping>
  <servlet-mapping>
    <servlet-name>check</servlet-name>
    <url-pattern>/check</url-pattern>
  </servlet-mapping>
</web-app>
```



```
//1. JDBC program for insertion of record into Student Table

import java.sql.*;

public class MyDatabase {

    public static void main(String[] args) {

        try
        {

            Class.forName("oracle.jdbc.driver.OracleDriver");

            System.out.println("Driver Loaded....");

            String cs="jdbc:oracle:thin:@localhost:1521:XE";

            Connection con=DriverManager.getConnection(cs,"system","student1234");

            System.out.println("Connection Established....");

            String s1="INSERT INTO EMP VALUES('aJAY2','EMP2',215000)";

            Statement st=con.createStatement();

            int c=st.executeUpdate(s1);

            System.out.println("Row inserted...."+c);

        }

        catch(Exception e)

        {

            e.printStackTrace();

        }

    }

}
```

//2. JDBC program for Retrieval of records from Student Table (ResultSet)

```
import java.sql.*;  
  
public class SqlResult  
{  
    public static void main(String[] args)  
    {  
        try {  
            Class.forName("oracle.jdbc.driver.OracleDriver");  
            System.out.println("Driver Loaded...");  
            String cs="jdbc:oracle:thin:@localhost:1521:XE";  
            Connection con=DriverManager.getConnection(cs,"system","student1234");  
            System.out.println("Connection Done...");  
            Statement stmt = con.createStatement();  
            String sql= "select * from Student";  
            ResultSet rs = stmt.executeQuery(sql);  
            while (rs.next())  
            {  
                String rno=rs.getInt(1);  
                name = rs.getString(2);  
                per = rs.getInt(3);  
                System.out.println("Roll No = " + rno + " Name = " + name + " Perc = " + per);  
            }  
            stmt.close();  
            con.close();  
        } catch(Exception e)  
        { System.out.println(e); }  
    }  
}
```

//3. JDBC program for insertion of records into Student Table using PreparedStatement.

```
public class MyPreparedStatement {  
    public static void main(String a[]){  
        try {  
            Class.forName("oracle.jdbc.driver.OracleDriver");  
            System.out.println("Driver Loaded...");  
            String cs="jdbc:oracle:thin:@localhost:1521:XE";  
            Connection con=DriverManager.getConnection(cs,"system","student1234");  
            System.out.println("Connection Done...");  
            String query = "insert into Student values(?, ?, ?)";  
            PreparedStatement prSt = con.prepareStatement(query);  
            prSt.setInt(1, 20);  
            prSt.setString(2, "Vimal");  
            prSt.setInt(3, 75);  
            //count will give you how many records got updated  
            int count = prSt.executeUpdate();  
            System.out.println("Record inserted: " +count)  
            //Run the same query with different values  
            prSt.setInt(1, 25);  
            prSt.setString(2, "Rajesh");  
            prSt.setInt(3, 66);  
            count = prSt.executeUpdate();  
            System.out.println("Record inserted: " +count)  
        } catch(Exception e)  
        {  
            System.out.println(e);  
        }  
    }  
}
```

```
//4. JDBC program for Retrieval of records from Student Table (ResultSetMetaData)

import java.sql.*;
public class SqlResult
{
    public static void main(String[] args)
    {
        try {
            Class.forName("oracle.jdbc.driver.OracleDriver");
            System.out.println("Driver Loaded...");
            String cs="jdbc:oracle:thin:@localhost:1521:XE";
            Connection con=DriverManager.getConnection(cs,"system","student1234");
            System.out.println("Connection Done...");
            Statement stmt = con.createStatement();
            String sql= "select * from Student";
            ResultSet rs = stmt.executeQuery(sql);
            ResultSetMetaData rsmd = rs.getMetaData();
            int columnCount = rsmd.getColumnCount();
            for(int i=0;i<=columnCount;i++){
                System.out.print(rsmd.getColumnName(i));
            }
            System.out.println();
            while (rs.next())
            {
                String rno=rs.getInt(1);
                name = rs.getString(2);
                per = rs.getInt(3);
                System.out.println(+ rno + " " + name + " " + per);
            }
            stmt.close();
        }
    }
}
```

```
con.close();  
} catch(Exception e)  
{ System.out.println(e); }  
}  
}
```


Short Note

GenericServlet:-

1. General for all protocol.
2. Implements Servlet Interface.
3. Use **Service** method.

HttpServlet:-

1. Only for HTTP Protocol.
2. Inherit GenericServlet class.
3. Use **doPost**, **doGet** method instead of service method.

Steps to write HttpServlet instead of Generic Servlet:-

1. Import one more package **javax.servlet.http**
2. Write class which extends **HttpServlet class**
3. Use **doGet()** or **doPost()** method instead of **service()**
4. In **<form>** tag write **method="post"** if you are using **doPost()**
5. In **<form>** tag no need to write **method="get"** if you are using **doGet()**

What is the difference between Difference between doGet() and doPost()

DoGet	DoPost
In doGet Method the parameters are appended to the URL and sent along with header information	In doPost, parameters are sent in separate line in the body
Maximum size of data that can be sent using doget is 240 bytes	There is no maximum size for data
Parameters are not encrypted	Parameters are encrypted
DoGet method generally is used to query or to get some information from the server	Dopost is generally used to update or post some information to the server
DoGet is faster if we set the response content length since the same connection is used. Thus increasing the performance	DoPost is slower compared to doGet since doPost does not write the content length

