

HISTORY

Java is the general purpose, true object oriented programming language and is highly suitable for modeling the real world and solving the real world problems. In the company Sun Microsystem, James Goslings(an employee there), started a project named 'GREEN'. He begun his work by attempting to extend C++ features for developing embedded software for electronic companies to automate the electronic devices such as microwave. But this could not be accomplished by C++.

So James Goslings started to develop a new language known as "OAK". In may 1995, Sun officially announced this language at Sunworld 95. Due to some reasons, the name was changed to "JAVA".

The prime motive of Java was the need for a platform independent language. The second motive was to design a language which is 'Internet Enable'. To run the programs on internet, Green project team come up with the idea of developing "Web Applet". The team developed a web browser called "Hot Java" to locate and run applet programs on internet.

The most striking feature of Java is 'Platform neutralness'. Java is the first language that is not tied to any particular hardware or O.S.

JAVA MILESTONES

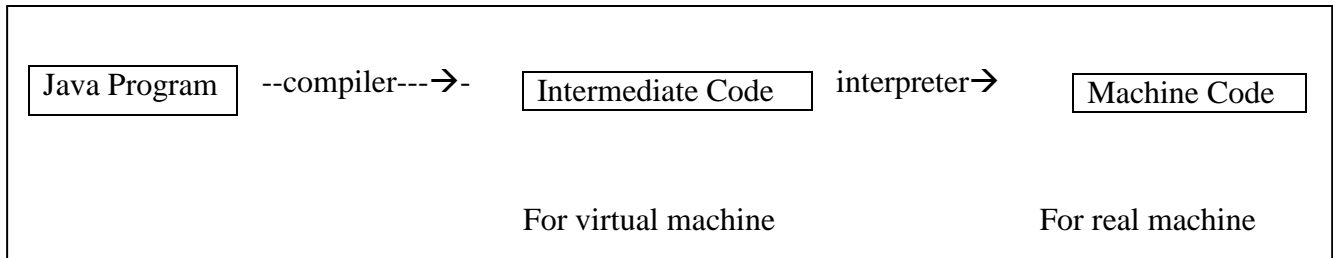
YEAR	DEVELOPMENT
1990	Sun Microsystem started to develop special software for electronic devices.
1991	James Goslings developed a new language "OAK".
1992	New language Oak was demonstrated in hand held devices.
1993	World Wide Web(WWW)appeared on internet and transformed the text based internet to graphical environment. Sun developed Applets.
1994	Green project team developed a web browser "Hot Java" to run applets on internet.
1995	Oak was renamed to JAVA.
1996	Java established itself as the leader of internet programming. Sun released Java development kit 1.0.
1997	Sun released Java development kit 1.1(JDK 1.1).
1998	Sun released Java2 with version 1.2(SDK 1.2).
1999	Sun released Java2 standard edition (J2SE) and Enterprise Edition (J2EE).
2000	J2SE with SDK 1.3 was released.
2002	J2SE with SDK 1.4 was released.
2004	J2SE with JDK 5.0 was released.

CHARACTERISTICS OF JAVA

1. **SIMPLE:** Java was designed to be easy for the professional programmers to learn and use effectively. Java uses C/C++ syntax. So, it is very simple to work in Java.
2. **OBJECT ORIENTED:** Java is true object oriented language. Almost everything in Java is an object. All program code and data reside within class and object. Java has a lot of classes arranged in packages that we can directly use in our programs.
3. **ROBUST AND SECURE:** Java is a robust language. Here robust means, it provides many safeguards to ensure reliable code. It has strict compile-time and run-time checking for data types. It also has the concept of exception handling to capture errors and remove the fear of crashing the system. Security is the major issue, when you work on internet. That's why, it not only verify all memory access but also ensure that no viruses are entering through the applet. That's why it doesn't use pointers.
4. **PLATFORM INDEPENDENT AND PORTABLE:** Java is portable that means Java programs can be easily moved from one system to another, anywhere, anytime. Java ensures portability in two ways. First, Java compiler generates bytecode instructions that can be implemented on any machine. Second, the size of primitive data types are machine dependent. Java is platform independent that means changes and upgrades in operating system, processors and system resources will not force any change in java programs. This is why Java is famous for programming on internet which interconnects different kinds of system worldwide.
5. **COMPILED AND INTERPRETED:** Usually, a language is either compiled or interpreted. Java combines both the approaches. First, java compiler translates the source code into bytecode. Bytecode is intermediate form, not the machine instruction. In the second stage, Java interpreter translate this bytecode to the machine code, which can be directly executed by the machine that is running the Java program.
6. **MULTITHREADED AND INTERACTIVE:** Multithreaded means handling multiple tasks simultaneously. Java supports multithreading programming. For example: we can listen an audio while typing or scrolling a page. Java comes with tools that support multiprocess synchronization and construct smooth running interactive system.
7. **DISTRIBUTED:** Java applications can open and access remote objects on internet as they easily do in a local system. This enables multiple programmers at multiple remote locations to collaborate and work together on a single project.

JAVA VIRTUAL MACHINE(JVM)

Java compiler translates the source code into intermediate code, known as byte code for a machine, that is called “Java virtual machine” and it exists only inside the computer memory. Then the computer interpreter convert the intermediate code into red machine code and it makes the java “Platform Independent” or “Architecture Neutral”.



All interpreters are different for different machines.

JAVA DEVELOPMENT KIT:

Java development kit comes with a collection of tools used for developing and running java programs. They include-

1. **APPLET VIEWER:** To view Java applets.
2. **JAVAC:** To compile the java programs.
3. **JAVA:** To run java programs (interpreter).
4. **JAVAP:** It is java deassembler, which converts bytecode files into program description.
5. **JAVAH:** It produces header files for use.
6. **JAVADOC:** It is used for HTML documents.
7. **JDB:** It is java program debugger.

API (APPLICATION PROGRAMMING INTERFACE)

API (Application programming interface) is a document that contains description of all the features of a product or software. It represents classes and interfaces that software programs can follow to communicate with each other. An API can be created for applications, libraries, operating systems, etc.

It is a java standard library. API includes hundreds of classes and methods grouped into several packages. Most common packages are:

1. **INPUT/OUTPUT PACKAGE(java.io.*):** It contains collection of classes required for input/output.
2. **AWT PACKAGE(java.awt.*):** Abstract window toolkit contains classes required for GUI.
3. **APPLET PACKAGE(java.applet.*):** It contains classes for applets.
4. **NETWORKING PACKAGE(java.net.*):** It contains classes for networking.
5. **UTILITY PACKAGE(java.util.*):** It contains classes for different utilities. Ex; date/time.

A PROGRAM TO DEMONSTRATE A SIMPLE PROGRAM TO PRINT SOMETHING ON THE SCREEN:

```
class example1  
  
{  
  
public static void main(String args[])  
  
    {  
  
        System.out.print("Hello");  
  
        System.out.println("Good morning");  
  
    }  
  
}
```

A PROGRAM TO DEMONSTRATE THE USE OF VARIABLE:

```
class Example2  
  
{  
  
public static void main(String args[])  
  
    { int num;  
  
        num=10;  
  
        System.out.println("The value is:" + num);  
  
    }  
  
}
```

HOW TO SAVE AND RUN THE PROGRAM:

- Save this program with the name Example2.java.
- Click on start menu and go to run option.
- Write cmd.
- Now command prompt will be opened.
- Go to the location where program is saved.
- Set the path of that location in command prompt.
- Give command to compile the program: **javac example2.java**
- Give the command to run the program. : **java example2**

Here compilation is compulsory before running the program. When you compile a program, if there is a mistake in program, it shows on the screen with line and location also.

JAVA KEYWORDS

There are 50 keywords defined in language java. The keywords const and goto are reserved but not used.

Abstract	Continue	for	new	Switch
Assert	Default	Goto	package	synchronize
boolean	Do	if	private	This
break	Double	implements	protected	throw
byte	Else	import	public	transient
case	Enum	Instance of	return	try
catch	Extends	int	short	void
char	Final	interface	static	throws
class	Finally	long	strictfp	volatile
const	Float	native	super	while

JAVA DATATYPES

java datatypes are divided into four categories:

1. Integer:

	NAME	WIDTH	RANGE
1.	Byte	8	-128 to 127
2.	Short	16	-32768 to 32767
3.	Int	32	-2147483648 to 2147483648
4.	Long	64	-9223372036854775808 to +ve

2. FLOATING POINT:

	NAME	WIDTH	RANGE
1.	Double	64	4.9e-324 to 1.8e+308
2.	Float	32	1.4e-45 to 3.4e+38

3. CHARACTER:

Char ----16 bit----0 to 65536

4. BOOLEAN: It provides “true” or ”false” result. It is used with all relational operators.

CLASS: class is the combination of data and its associated functions together. The data means the variables. These functions and variables are collectively called “class members” i.e data members and function members. A class is defined by class keyword. Class contains all the features of OOPS.

Ex:

```
class classname
{
    datatype var_name;
    returntype function_name( )
    {
        .....
        .....
    }
}
```

OBJECT: An object is essentially a block of memory that contains space to store all the instance variables. Creating an object is called “instantiating” a object.

Objects are created using the keyword “new”. The new keyword creates an object of the specified class and allocates memory and returns a reference to that object.

Ex:

```
Suppose Box is a class-
Box b=new Box( )
```

Here b is an object of class Box.

PACKAGE:

When java interfaces and classes are grouped together in to single unit that is called “Package”. Programs are organized as set of packages. Each package has its own names. The naming structure for packages is hierarchical.

Packages also help us to avoid class name collision when we use the same class as that of others.

To create a package, simply include a package command as the first statement in the java source file. Any class declared within that file will belong to the specified package.

Syntax:

Package <package name>

To use a package in your program, you must use following syntax-

Syntax:

```
import java.<package_name>.*;
```

ex: import java.util.*;

The classes and interface of util package will be accessible to the program.

INHERITANCE:

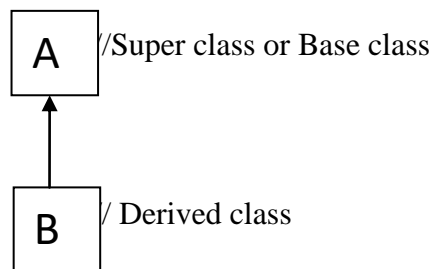
The mechanism of deriving a new class from an old name is called “Inheritance”. The old class is known as base class or super class or parent class and the new one is called derived class or sub class or child class.

The inheritance allows subclasses to inherit (reuse) all the variables and methods of their parent class.

Inheritance can be of following types in java-

1. **SINGLE INHERITANCE:** (Only one super class)

In single inheritance, only one super class is created that means there is only one super class and derived class.



The keyword “extends” is used to call a super class in derived class.

e.g class room

```
{
    int length, breadth;
    void show( )
    {
        System.out.println("L:" + length + "B:" + breadth);
    }
}
```

Class bedroom extends room

```
{
    int height;
    void display( )
    {
        System.out.println("Height:" + h);
    }
    void volume( )
    {
        System.out.println("L * B * H:" + (length * breadth * height));
    }
}
```

class simpleinheritance

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

```

{
    room t=new room( );
    bedroom m=new bedroom( );
    t.length=10;
    t.breadth=20;
    System.out.println("Controls of Super class");
    t.show( );
    m.height=50;
    System.out.println("Controls of Derive class");
    m.volume( );
}
}

```

2. MULTIPLE INHERITANCE:

Java does not implements multiple inheritance directly. This concept is implemented using a secondary inheritance path in the form of interfaces.

INTERFACE: An interface is actually a kind of class. Like classes, interface contains methods and variables but they define only abstract mehods and final fields.

Abstract methods are those methods which do not specify any code to implement these methods and data fields(variables) contain only constants.

In this way interface contains methods which do not have the code to implement them and variables having only constant value. Therefore, it is the responsibility of that class which implements them to define the code for implementation of these methods.

Syntax: To define an interface-

```

interface interface_name
{
    Variable declaration;
    Methods declaration;
}

```

Interface is used as “super class” whose properties are inherited by other classes. “implements” keyword is used to inherit an inherit an a class.

e.g

```

interface Area
{
    float pi=3.14;
    float compute(float x, float y);
}

```

class Rectangle implements Area

```

{
    float compute(float x, float y)
    {

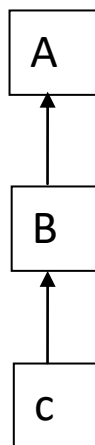
```

```
    return (x*y);  
  }  
}
```

```
class Circle implements Area  
{  
    float compute(float x, float y)  
    {  
        return(pi*x*y);  
    }  
}
```

```
class interface_test  
{  
    public static void main(String args[])  
    {  
        Rectangle r=new Rectangle( );  
        Circle    c=new Circle( );  
        Area a;  
        a=r;  
        System.out.println("AREA OF RECTANGLE:" + a.compute(10,20) );  
        a=c;  
        System.out.println("AREA OF RECTANGLE:" + a.compute(10,20) );  
    }  
}
```

3. MULTILEVEL INHERITANCE:



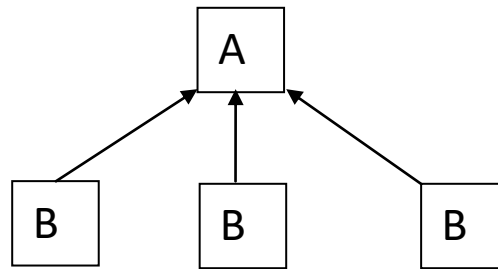
When a class A serve as a base class for the derive class B, which again serve as base class for the derive class C, this is called Multilevel Inheritance” and this chain A->B->C is called “Inheritance Path”.

Ex:

```
class Car{
    public Car()
    {
        System.out.println("Class Car");
    }
    public void vehicleType()
    {
        System.out.println("Vehicle Type: Car");
    }
}
class Maruti extends Car{
    public Maruti()
    {
        System.out.println("Class Maruti");
    }
    public void brand()
    {
        System.out.println("Brand: Maruti");
    }
    public void speed()
    {
        System.out.println("Max: 90Kmph");
    }
}
public class Maruti800 extends Maruti{

    public Maruti800()
    {
        System.out.println("Maruti Model: 800");
    }
    public void speed()
    {
        System.out.println("Max: 80Kmph");
    }
    public static void main(String args[])
    {
        Maruti800 obj=new Maruti800();
        obj.vehicleType();
        obj.brand();
        obj.speed();
    }
}
```

4. HIERARICAL INHERITANCE:



As you can see in the above diagram that when a class has more than one child classes (sub classes) or in other words more than one child classes have the same parent class then such kind of inheritance is known as hierarchical.

Example:

```
class A
{ public void methodA()
  {   System.out.println("method of Class A");
  }
}
class B extends A
{ public void methodB()
  {   System.out.println("method of Class B");
  }
}
class C extends A
{ public void methodC()
  {   System.out.println("method of Class C");
  }
}
class D extends A
{ public void methodD()
  {   System.out.println("method of Class D");
  }
}
class MyClass
{ public void methodB()
  {   System.out.println("method of Class B");
  }
  public static void main(String args[])
  { B obj1 = new B();
    C obj2 = new C();
    D obj3 = new D();
    obj1.methodA();
    obj2.methodA();
    obj3.methodA();
  }
}
```

CONSTRUCTOR:

Constructor is a special type of method that enables an object to initialize itself when it is created.

Constructors have the same name as the class itself. They do not specify a return type not given void because they are used for creating objects (instances) only.

Ex:

```
class Rectangle
{
    int length, width;
    rectangle(int x, int y)
    {
        length=x;
        width=y;
    }
    void area( )
    {
        System.out.println("Area is:"+(x*y));
    }
}
```

```
class RectangleArea
{
    public static void main(String ars[])
    {
        Rectangle r=new Rectangle(15,10);
        r.area( );
    }
}
```

TYPES OF CONSTRUCTORS:

1. **Default Constructor-** The constructor which has no parameters.
2. **Parameterized constructor-** The constructor which has some parameters.
3. **Multiple constructor-** A class can contain multiple constructors with different parameters.
4. **Copy constructor-** A constructors which copies the value of one object to another.

COPY CONSTRUCTOR:

A copy constructor is a constructor that takes only one parameter that is the same type as the class in which the copy constructor is defined.

A copy constructor is used to create an object that is the copy of its parameter but it is an actually independent copy.

Example:

```
class Complex
{
    private double re, im;

    public Complex(double re, double im)
    {
        this.re = re;
        this.im = im;
    }

    Complex(Complex c)
    {
        System.out.println("Copy constructor called");
        re = c.re;
        im = c.im;
    }

    public String toString()
    {
        return "(" + re + " + " + im + "i)";
    }
}

public class Main
{
    public static void main(String[] args)
    {
        Complex c1 = new Complex(10, 15);

        Complex c2 = new Complex(c1);

        Complex c3 = c2;

        System.out.println(c2); // toString() of c2 is called here
    }
}
```

GARBAGE COLLECTION:

Java deallocates the objects automatically, this technique is called “Garbage collection”. It is similar to C++ delete operator option but it is done manually in C++ while garbage collection is automatic.

When no reference to an object exists that object is assumed to be no longer needed and memory occupied is freed by garbage collector automatically.

THIS KEYWORD:

This keyword is used to resolve the ambiguity between instance variables(class variables) and parameters, when the names of instance variables and parameters are same. This can be also used inside any method to refer to the current object. This is always a reference to the object on which the method was invoked.

Example:

```
class Rectangle
{
    int length, width;
    void calculate(int length, int width)
    {
        this.length=x;
        this.width=y;
    }
    void area( )
    {
        System.out.println("Area is:"+(x*y));
    }
}
```

```
class RectangleArea
{
    public static void main(String args[])
    {
        Rectangle r=new Rectangle(15,10);
        r.calculate( );
        r.area( );
    }
}
```

ABSTRACT CLASS:

A class that contains one or more abstract methods (methods which have no definition), is called “Abstract Class”. In other words, it is a superclass that declares the structure of a given abstraction without providing a complete implementation of every method.

To declare a class abstract, you just need use simply “abstract” keyword in front on class keyword.

There can be no objects of abstract class. An abstract class can not be directly instantiated with the new operator, you can not declare abstract constructors.

Example:

```
abstract class A
{
    abstract void callme( );
    void callmetoo( )
    {
        System.out.println(“THIS IS NOT ABSTRACT METHOD”);
    }
}
class B extends A
{
    void callme( )
    {
        System.out.println(“ CALLME ABSTRACT METHOD IMPLEMENTATION IN B”);
    }
}

class abstarctdemo
{
    Public static void main(String args[])
    {
        B b=new B( );
        b.callme( );
        b.callmetoo( );
    }
}
```

FINALIZE METHOD ():

Finalization is just the opposite process of initialization. Constructor is used to initialize an object when it is declared; this process is called “Initialization”.

Java automatically frees up the memory resources used by the objects but objects may hold other non object resources like file descriptors or window system fonts. The java run-time (Garbage Collector) cannot free these resources. In order to free these resources we must use “finalize method ()”. This is similar to destructor in C++.

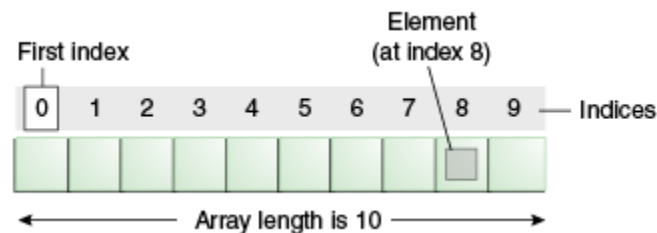
The finalize method () is simply finalize () and can be added to any class. Java calls that method whenever it is required.

ARRAY

An array is a group of like-typed variables that are referred by a common name a specific element in an array is accessed by its index. Array offers a convenient way of grouping related information.

Java array is an object the contains elements of similar data type. It is a data structure where we store similar elements. We can store only fixed set of elements in a java array.

Array in java is index based, first element of the array is stored at 0 index.



Advantage of Java Array

- **Code Optimization:** It makes the code optimized; we can retrieve or sort the data easily.
- **Random access:** We can get any data located at any index position.

TYPES OF ARRAY IN JAVA

There are two types of array.

- Single Dimensional Array
- Multidimensional Array

One-dimensional array

A one dimensional array is essentially a list of like typed variable to create an array, you must use following syntax-

Type variable-name [];

Ex- **int month_days [];**

This means, all the month_days which are about to insert in this month_days array must be integer. Still actually no array exists. The value of month_days is set to NULL, which means an array with no value. To make an array actual, physical array of integers, you must use new operator .New is a special operator that allocates memory.

Ex- month_days= new int [12];

On execution month_days will refer to an array of 12 integers and all elements will be initialized to zero.

Ex-

```
class array
{
public static void main (String args [ ])
{
    int month_days [ ];
    month_days= new int [12];
    month_days [0] =1;
    month_days [1] =2;
    month_days [2] =3;
    System.out.println ("Days:" + month_days [2]);
}
}
```

MULTIDIMENSIONAL ARRAY

Data is stored in row and column based index (also known as matrix form).

Syntax

1. dataType[][] arrayRefVar; (or)
2. dataType [][]arrayRefVar; (or)
3. dataType arrayRefVar[][]; (or)
4. dataType []arrayRefVar[];

Example:

```
class doublearray
{
public static void main(String args[])
{
    int arr[][]={{ 1,2,3},{2,4,5},{4,4,5 }};
    for(int i=0;i<3;i++)
    {
        for(int j=0;j<3;j++)
        {
            System.out.print(arr[i][j]+" ");
        }
        System.out.println();
    }
}
}
```

ACCESS-MODIFIERS

The visibility modifiers are also known as Access-modifiers. Java provides 3 types of modifiers:

- **Public**: A variable or method declared as public is accessible everywhere, in the class or out of the class .it has the widest possible visibility.

Ex:

```
public int x;  
public void sum()  
{  
=====
```

- **Private**: A method declared as private behaves like final. It prevents the method from being subclassed, they are accessible only with their own class, they can't be used out of that class. They can't be inherited by subclasses. They have highest degree of protection.

Ex:

```
private int x;  
private void sum()  
{  
=====
```

- **Protected**: It is between private & public .it makes the variable or methods visible to all classes and subclasses in the same package and also to subclasses in other package. Non-subclasses in other package cannot access the protected method .

Ex:

```
protected int x;  
protected void sum()  
{  
=====
```

VISIBILITY

LOCATION	PUBLIC	PRIVATE	PROTECTED
Same class sub class in	YES	YES	YES
Same package	YES	NO	YES
Other class in same package	YES	NO	YES
Sub class in other package	YES	NO	YES
Non-subclass in other package	YES	NO	NO

DIFFERENCE BETWEEN ABSTRACT CLASS AND INTERFACE

Abstract class and interface both are used to achieve abstraction where we can declare the abstract methods. Abstract class and interface both can't be instantiated.

But there are many differences between abstract class and interface that are given below.

Abstract class	Interface
1) Abstract class can have abstract and non-abstract methods.	Interface can have only abstract methods. Since Java 8, it can have default and static methods also.
2) Abstract class doesn't support multiple inheritance.	Interface supports multiple inheritance.
3) Abstract class can have final, non-final, static and non-static variables.	Interface has only static and final variables.
4) Abstract class can provide the implementation of interface.	Interface can't provide the implementation of abstract class.
5) The abstract keyword is used to declare abstract class.	The interface keyword is used to declare interface.
6) Example: public abstract class Shape{ public abstract void draw(); }	Example: public interface Drawable{ void draw(); }

```
interface A{  
void a();//bydefault, public and abstract  
void b();  
void c();  
void d();  
} //Creating abstract class that provides the imlemen  
tation of one method of A interface  
abstract class B implements A{  
public void c(){System.out.println("I am C");}  
} class M extends B{  
public void a(){System.out.println("I am a");}  
public void b(){System.out.println("I am b");}  
public void d(){System.out.println("I am d");}  
}
```

```
//Creating a test class that calls the methods of A  
interface  
class Test5{  
public static void main(String args[]){  
A a=new M();  
a.a();  
a.b();  
a.c();  
a.d();  
}}}
```

AWT (ABSTRACT WINDOW TOOLS)

The abstract window toolkit contains several classes and methods that allow you to create and manage window.

The awt classes are contained in the java.awt package. The awt define windows according to a class hierarchy that adss functionality and specify with each level.

COMPONENTS

The top of the AWT hierarchy is the component class.

Component is an abstract class that contains all the attributes of a visual components. All user interfaces implements that are displayed on the screen is the subclass of component class. It defines hundred of ublic methods to manage events, such as mouse, keyboard inputs etc. It is also responsible for current background color and font.

CONTAINERS

The container class is the subclass of component class. A container is responsible for laying out (Positioning) any component, it contains. It handles this with various layout managers.

PANEL:

Panel is the subclass of conatiner. It doesn't add any new methods: it simply implements container. Panel is the superclass of applet.

A panel is a window that doesn't contain a title bar, menu bar or border. When an output is directed to an applet, it is drawn on the surface of a Panel object.

WINDOW:

A window class creates a top-level window. Generally, it is not used directly, instead we use subclass of window called FRAME.

FRAME:

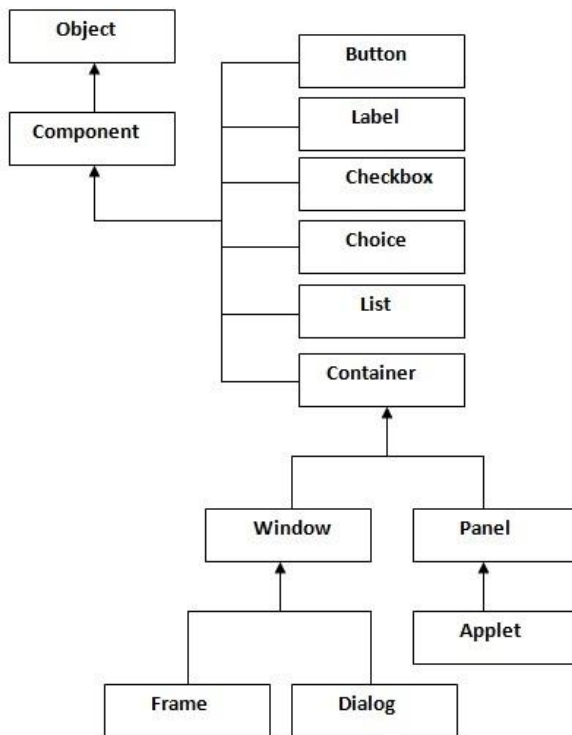
It is the surface of window and it has title bar, menu bar, border and rising corners.

When a frame window is created by a stand-alone application rather than a n applet, a normal window is created.

CANVAS:

Canvas creates a new blank window upon which you can draw.

JAVA AWT HIERARCHY



CLASS

DESCRIPTION

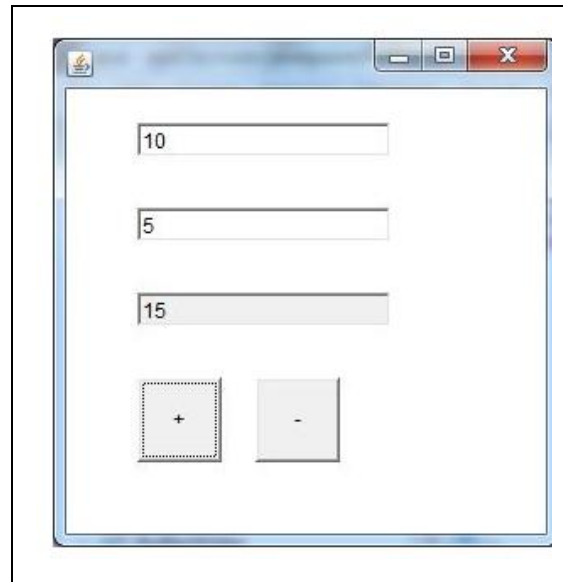
CLASS	DESCRIPTION
AWTEvent	It handles all events.
Button	It controls all the actions of button.
Checkbox	Creates a checkbox control.
Choice	Creates a pop-up list.
Dialog	Creates a dialog window.
Menu	Creates a pull-down menu.
Menubar	Creates a menu bar.
Window	Creates a window with no frame.
Scrollbar	Creates a scroll bar control.
Color	Manages color.
Canvas	A blank, semantics free window.
FileDialog	Creates a window from which file can be selected.
Cursor	Creates a bitmapped cursor.
Checkboxgroup	Creates a group of checkbox controls.
Textfield	Creates a single line edit control.
Textarea	Creates a multiline edit control.

TEXTFIELD

```
import java.awt.*;
class TextFieldExample{
public static void main(String args[]){
    Frame f= new Frame("TextField Example");
    TextField t1,t2;
    t1=new TextField("Welcome to Viva.");
    t1.setBounds(50,100, 200,30);
    t2=new TextField(" ");
    t2.setBounds(50,150, 200,30);
    f.add(t1); f.add(t2);
    f.setSize(400,400);
    f.setLayout(null);
    f.setVisible(true);
}
}
```

TEXTFIELD WITH EVENT

```
import java.awt.*;
import java.awt.event.*;
public class TextFieldExample extends Frame implements ActionListener{
    TextField tf1,tf2,tf3;
    Button b1,b2;
    TextFieldExample(){
        tf1=new TextField();
        tf1.setBounds(50,50,150,20);
        tf2=new TextField();
        tf2.setBounds(50,100,150,20);
        tf3=new TextField();
        tf3.setBounds(50,150,150,20);
        tf3.setEditable(false);
        b1=new Button("+");
        b1.setBounds(50,200,50,50);
        b2=new Button("-");
        b2.setBounds(120,200,50,50);
        b1.addActionListener(this);
        b2.addActionListener(this);
        add(tf1);add(tf2);add(tf3);add(b1);add(b2);
        setSize(300,300);
        setLayout(null);
        setVisible(true);
    }
    public void actionPerformed(ActionEvent e) {
```



```

String s1=tf1.getText();
String s2=tf2.getText();
int a=Integer.parseInt(s1);
int b=Integer.parseInt(s2);
int c=0;
if(e.getSource()==b1){
    c=a+b;
}else if(e.getSource()==b2){
    c=a-b;
}
String result=String.valueOf(c);
tf3.setText(result);
}
public static void main(String[] args) {
    new TextFieldExample();
}
}

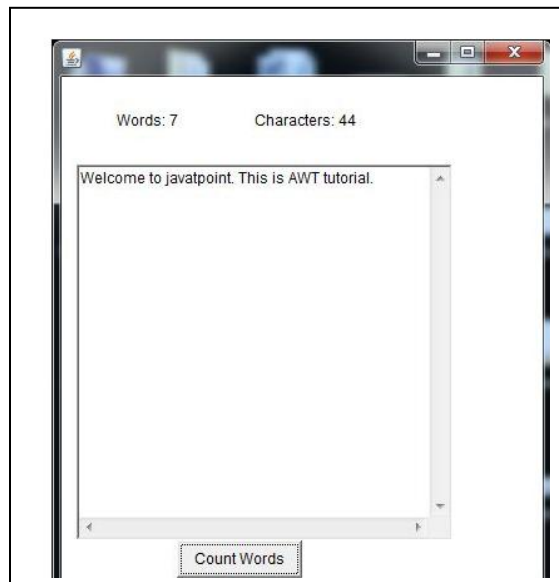
```

TEXTAREA

```

import java.awt.*;
import java.awt.event.*;
public class TextAreaExample extends Frame implements ActionListener{
    Label l1,l2;
    TextArea area;
    Button b;
    TextAreaExample(){
        l1=new Label();
        l1.setBounds(50,50,100,30);
        l2=new Label();
        l2.setBounds(160,50,100,30);
        area=new TextArea();
        area.setBounds(20,100,300,300);
        b=new Button("Count Words");
        b.setBounds(100,400,100,30);
        b.addActionListener(this);
        add(l1);add(l2);add(area);add(b);
        setSize(400,450);
        setLayout(null);
        setVisible(true);
    }
    public void actionPerformed(ActionEvent e){
        String text=area.getText();

```



```

String words[]=text.split("\\s");
l1.setText("Words: "+words.length);
l2.setText("Characters: "+text.length());
}
public static void main(String[] args) {
    new TextAreaExample();
}
}

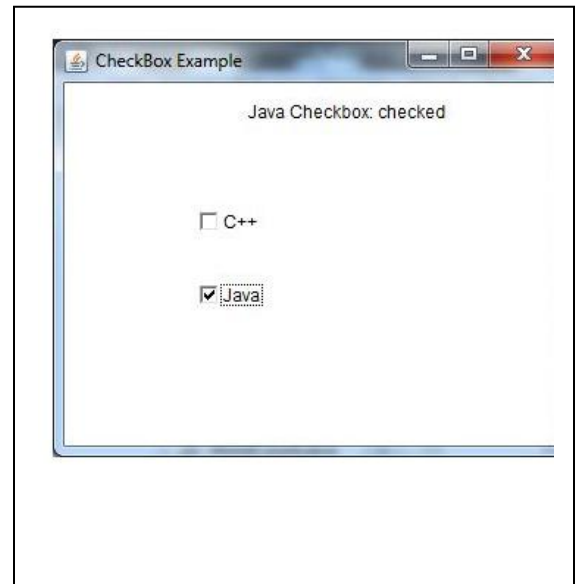
```

CHECKBOX

```

import java.awt.*;
import java.awt.event.*;
public class CheckboxExample
{
    CheckboxExample(){
        Frame f= new Frame("CheckBox Example");
        final Label label = new Label();
        label.setAlignment(Label.CENTER);
        label.setSize(400,100);
        Checkbox checkbox1 = new Checkbox("C++");
        checkbox1.setBounds(100,100, 50,50);
        Checkbox checkbox2 = new Checkbox("Java");
        checkbox2.setBounds(100,150, 50,50);
        f.add(checkbox1); f.add(checkbox2); f.add(label);
        checkbox1.addItemListener(new ItemListener() {
            public void itemStateChanged(ItemEvent e) {
                label.setText("C++ Checkbox: "
                    + (e.getStateChange()==1?"checked":"unchecked"));
            }
        });
        checkbox2.addItemListener(new ItemListener() {
            public void itemStateChanged(ItemEvent e) {
                label.setText("Java Checkbox: "
                    + (e.getStateChange()==1?"checked":"unchecked"));
            }
        });
        f.setSize(400,400);
        f.setLayout(null);
        f.setVisible(true);
    }
    public static void main(String args[])
    {
        new CheckboxExample();
    }
}

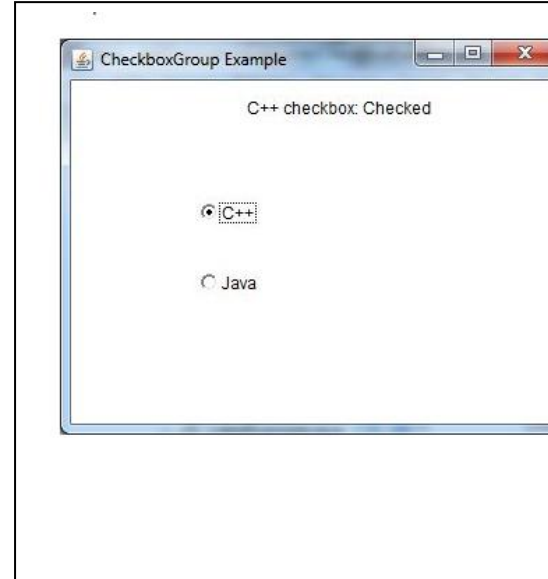
```



```
}
```

CHECKBOXGROUP

```
import java.awt.*;
import java.awt.event.*;
public class CheckboxGroupExample
{
    CheckboxGroupExample() {
        Frame f= new Frame("CheckboxGroup Example");
        final Label label = new Label();
        label.setAlignment(Label.CENTER);
        label.setSize(400,100);
        CheckboxGroup cbg = new CheckboxGroup();
        Checkbox checkBox1 = new Checkbox("C++", cbg, false);
        checkBox1.setBounds(100,100, 50,50);
        Checkbox checkBox2 = new Checkbox("Java", cbg, false);
        checkBox2.setBounds(100,150, 50,50);
        f.add(checkBox1); f.add(checkBox2); f.add(label);
        f.setSize(400,400);
        f.setLayout(null);
        f.setVisible(true);
        checkBox1.addItemListener(new ItemListener() {
            public void itemStateChanged(ItemEvent e) {
                label.setText("C++ checkbox: Checked");
            }
        });
        checkBox2.addItemListener(new ItemListener() {
            public void itemStateChanged(ItemEvent e) {
                label.setText("Java checkbox: Checked");
            }
        });
    }
    public static void main(String args[])
    {
        new CheckboxGroupExample();
    }
}
```



CHOICE

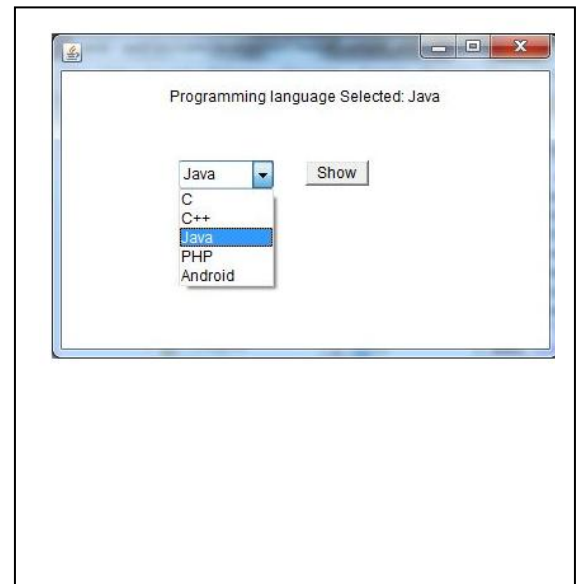
```
import java.awt.*;
import java.awt.event.*;
public class ChoiceExample
{
    ChoiceExample() {
        Frame f= new Frame();
```

```

final Label label = new Label();
label.setAlignment(Label.CENTER);
label.setSize(400,100);
Button b=new Button("Show");
b.setBounds(200,100,50,20);
final Choice c=new Choice();
c.setBounds(100,100, 75,75);
c.add("C");
c.add("C++");
c.add("Java");
c.add("PHP");
c.add("Android");
f.add(c);f.add(label); f.add(b);
f.setSize(400,400);
f.setLayout(null);
f.setVisible(true);
b.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        String data = "Programming language Selected: "+ c.getItem(c.getSelectedIndex());
        label.setText(data);
    }
});
}

public static void main(String args[])
{
    new ChoiceExample();
}
} .

```

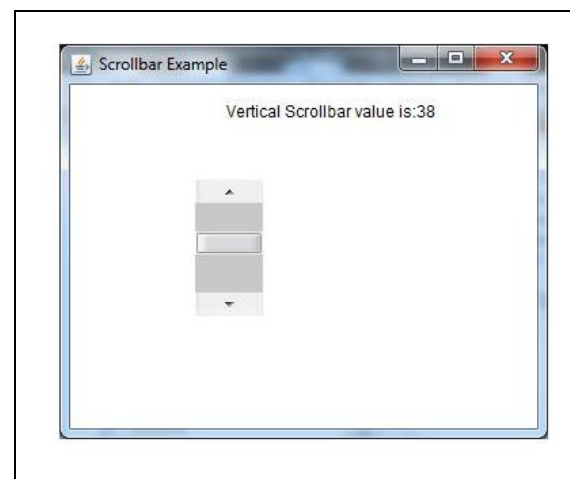


SCROLLBAR

```

import java.awt.*;
import java.awt.event.*;
class ScrollbarExample{
    ScrollbarExample(){
        Frame f= new Frame("Scrollbar Example");
        final Label label = new Label();
        label.setAlignment(Label.CENTER);
        label.setSize(400,100);
        final Scrollbar s=new Scrollbar();
        s.setBounds(100,100, 50,100);
        f.add(s);f.add(label);
        f.setSize(400,400);
        f.setLayout(null);
        f.setVisible(true);
    }
    public static void main(String args[]){

```



```

new ScrollbarExample();
}
}

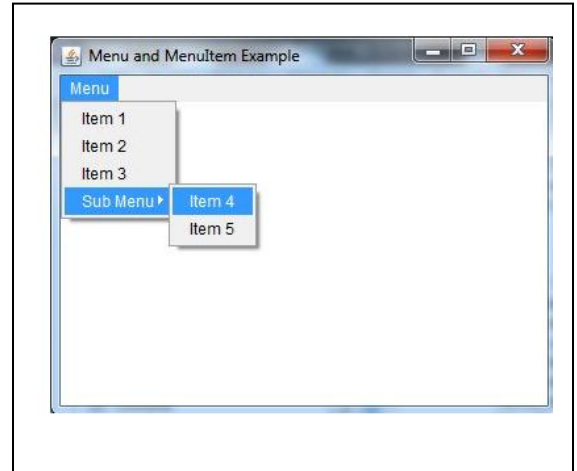
```

MENU

```

import java.awt.*;
class MenuExample
{
    MenuExample(){
        Frame f= new Frame("Menu and MenuItem Example");
        MenuBar mb=new MenuBar();
        Menu menu=new Menu("Menu");
        Menu submenu=new Menu("Sub Menu");
        MenuItem i1=new MenuItem("Item 1");
        MenuItem i2=new MenuItem("Item 2");
        MenuItem i3=new MenuItem("Item 3");
        MenuItem i4=new MenuItem("Item 4");
        MenuItem i5=new MenuItem("Item 5");
        menu.add(i1);
        menu.add(i2);
        menu.add(i3);
        submenu.add(i4);
        submenu.add(i5);
        menu.add(submenu);
        mb.add(menu);
        f.setMenuBar(mb);
        f.setSize(400,400);
        f.setLayout(null);
        f.setVisible(true);
    }
    public static void main(String args[])
    {
        new MenuExample();
    }
}

```

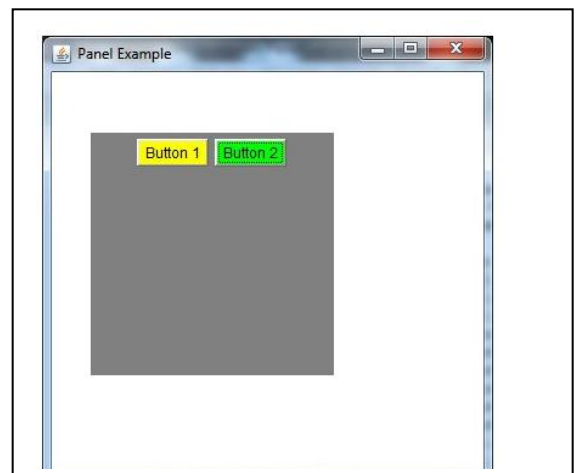


PANEL

```

import java.awt.*;
public class PanelExample {
    PanelExample()
    {
        Frame f= new Frame("Panel Example");
        Panel panel=new Panel();
        panel.setBounds(40,80,200,200);
        panel.setBackground(Color.gray);
        Button b1=new Button("Button 1");
        b1.setBounds(50,100,80,30);
    }
}

```



```

b1.setBackground(Color.yellow);
Button b2=new Button("Button 2");
b2.setBounds(100,100,80,30);
b2.setBackground(Color.green);
panel.add(b1); panel.add(b2);
f.add(panel);
f.setSize(400,400);
f.setLayout(null);
f.setVisible(true);
}      public static void main(String args[])
{      new PanelExample();
} }

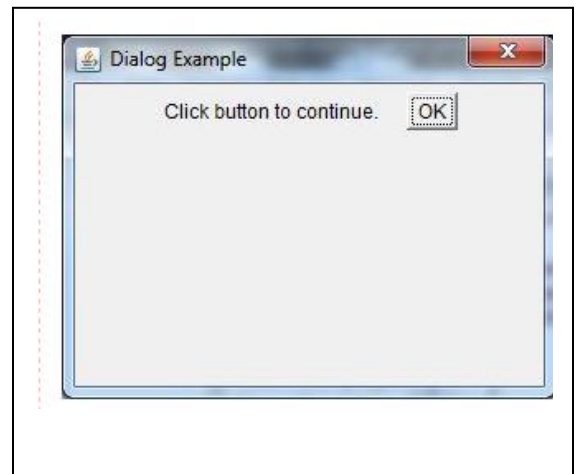
```

AWT DIALOG

```

import java.awt.*;
import java.awt.event.*;
public class DialogExample {
    private static Dialog d;
    DialogExample() {
        Frame f= new Frame();
        d = new Dialog(f , "Dialog Example", true);
        d.setLayout( new FlowLayout() );
        Button b = new Button ("OK");
        b.addActionListener ( new ActionListener()
        {
            public void actionPerformed((ActionEvent e)
            {
                DialogExample.d.setVisible(false);
            }
        });
        d.add( new Label ("Click button to continue."));
        d.add(b);
        d.setSize(300,300);
        d.setVisible(true);
    }
    public static void main(String args[])
    {
        new DialogExample();
    } }

```



LAYOUTMANAGER

A layout manager is an instance of any class that implements the layout manager interface. The layout manager is set by the setlayout method.

There are many types of layout manger:-

(A) FLOW LAYOUT

Flow layout is the default layout manger. Flow layout implements a simple layout style , which is similar to how words flow ion a text editor.

Example:

```
import java.awt.*;
import javax.swing.*;

public class MyFlowLayout{
    JFrame f;
    MyFlowLayout(){
        f=new JFrame();
        JButton b1=new JButton("1");
        JButton b2=new JButton("2");
        JButton b3=new JButton("3");
        JButton b4=new JButton("4");
        JButton b5=new JButton("5");
        f.add(b1);f.add(b2);f.add(b3);f.add(b4);f.add(b5);
        f.setLayout(new FlowLayout(FlowLayout.RIGHT));
        //setting flow layout of right alignment
        f.setSize(300,300);
        f.setVisible(true);
    }
    public static void main(String[] args) {
        new MyFlowLayout();
    }
}
```

(B) BORDER LAYOUT

The border layout outclass is implements a common layout style for top level windows.it has four narrow ,fixed width components at the edges and one large area in the centre .

The four sides are referred to as north ,south,east,and west.the middle area is called the center.

Example:

```
import java.awt.*;
import javax.swing.*;
public class Border {
    JFrame f;
    Border(){
        f=new JFrame();
        JButton b1=new JButton("NORTH");;
        JButton b2=new JButton("SOUTH");;
        JButton b3=new JButton("EAST");;
        JButton b4=new JButton("WEST");;
        JButton b5=new JButton("CENTER");;
        f.add(b1,BorderLayout.NORTH);
        f.add(b2,BorderLayout.SOUTH);
        f.add(b3,BorderLayout.EAST);
        f.add(b4,BorderLayout.WEST);
        f.add(b5,BorderLayout.CENTER);
        f.setSize(300,300);
        f.setVisible(true);
    }
    public static void main(String[] args) {
        new Border();
    }
}
```

(C) **GRIDLAYOUT**

Grid layout out components in a two-dimensional grid.when you instantiate a grid layout ,we define the number of rows and columns.

Example:

```
import java.awt.*;
import javax.swing.*;
public class MyGridLayout{
    JFrame f;
    MyGridLayout(){
        f=new JFrame();
        JButton b1=new JButton("1");
```

```

JButton b2=new JButton("2");
JButton b3=new JButton("3");
JButton b4=new JButton("4");
JButton b5=new JButton("5");
    JButton b6=new JButton("6");
    JButton b7=new JButton("7");
JButton b8=new JButton("8");
    JButton b9=new JButton("9");
    f.add(b1);f.add(b2);f.add(b3);f.add(b4);f.add(b5);
f.add(b6);f.add(b7);f.add(b8);f.add(b9);
    f.setLayout(new GridLayout(3,3));
//setting grid layout of 3 rows and 3 columns

f.setSize(300,300);
f.setVisible(true);
}
public static void main(String[] args) {
    new MyGridLayout();
}
}

```

(D) **CARDLAYOUT**

The card layout class is unique among the other layout managers in that it stores several different layouts each layout can be thought of as being on a separate index card in a deck that can be shuffled so that any.

JAVA EVENT DELEGATION MODEL

In this a source generates an event and sends it to one or more listeners. In this scheme, the listeners simply wait until they receive an event. Once an event is received, the listener processes the event and then returns.

The advantage of this design is that the application logic that processes events is clearly separated from the user interface logic that generates those events. A user interface element is able to “delegate” the processing of an event to a separate piece of code.

Listeners must register with a source in order to receive an event notification.

Before the advent of the event delegation model, an event was propagated up the containment hierarchy until it was handled by a component. It wasted time, this new model eliminates this overhead.

EVENT CLASS

At the root of the java event class hierarchy is event object ,which is in java.util. It is the superclass for all events.

Event object contains two methods:-

- (1) getSource():- It returns the source of the event.
- (2) toString():- It returns the string equivalent of the event.

The class “AWTEVENT”, defined withinb the java.awt package ,is a subclass of event object . It is superclass of all AWT based events used by the delegation event model . the getID() method can be used to determine the type of event.

Most of other classes ,which are given below are subclass of AWTEVENT.

ACTIONEVENTCLASS

An ActionEvent is general .when a button is pressed .The ActionEvent class defines four integer constant , used to identify any modifier associated with an action event.

- (A)ALT_MASK.
- (B)CTRL_MASK.
- (C)META_MASK
- (D)SHIFT_MASK.

ADJUSTMENTEVENT CLASS- When adjustment value is changed the this event is generated. an AdjustmentEvent is generated by a scrollbar.There are five types of adjustment event.The AdjustmentEvent class define integer constant that can be used to identify them.

- (a) Block_Decrement- Theuser clicked inside the scrollbar to decrease its value.
- (b) Block_Increment-The user clicked inside the scrollbar to increase its value.
- (c) Track- The slider was dragged.
- (d) Unit_Decrement-The butteon of the end of the scrollbar was clicked to decrease its value.
- (e) Unit_Increment-The button of the end of the scrollbar was clicked to increase its value.

CONTAINEREVENT CLASS-(Container contened changes)

AContainerEvent is generated when a component is added or removed from a container. There are two types of ContainerEvents.

The ContainerEvent class defines int constants that can be used two identify them.

- 1.COMPONENT_ADDED

2.COMPONENT_REMOVED

1.COMPONENT_EVENT-Indicates if object is moved changed and its state this class only perform notification about state of object. It perform as root class for other component level event.

2.PAINTEVENT-Insure that Paint () or Update() are serialized along with other events.

3.WINDOW_EVENT-If the window or the Frame of your application is changed(open closed achirated,deacrtrated),WindowEvent is generated.

4.DESTROY()-This method is called only one time in the whole life cycle of applet like init() method.

This method is called on that time when the browser needs to shut down.

EXAMPLE.

```
import java.applet.Applet.*;
import java.awt.Graphics;
public class simple extends Applet
{
    String Buffer buffer;
    public void init()
    {
        buffer=new StringBuffer();
        addItem("Initializing....");
    }
    public void start()
    {
        addItem("Starting...");
    }
    public void stop()
    {
        addItem("stopping..");
    }
    public void destroy()
    {
        addItem("Preparing for unloading...");
    }
}
```

SWING

Java Swing is a part of Java Foundation Classes (JFC) that is *used to create window-based applications*. It is built on the top of AWT (Abstract Windowing Toolkit) API and entirely written in java.

Unlike AWT, Java Swing provides platform-independent and lightweight components.

The javax.swing package provides classes for java swing API such as JButton, JTextField, JTextArea, JRadioButton, JCheckbox, JMenu, JColorChooser etc.

Difference between AWT and Swing

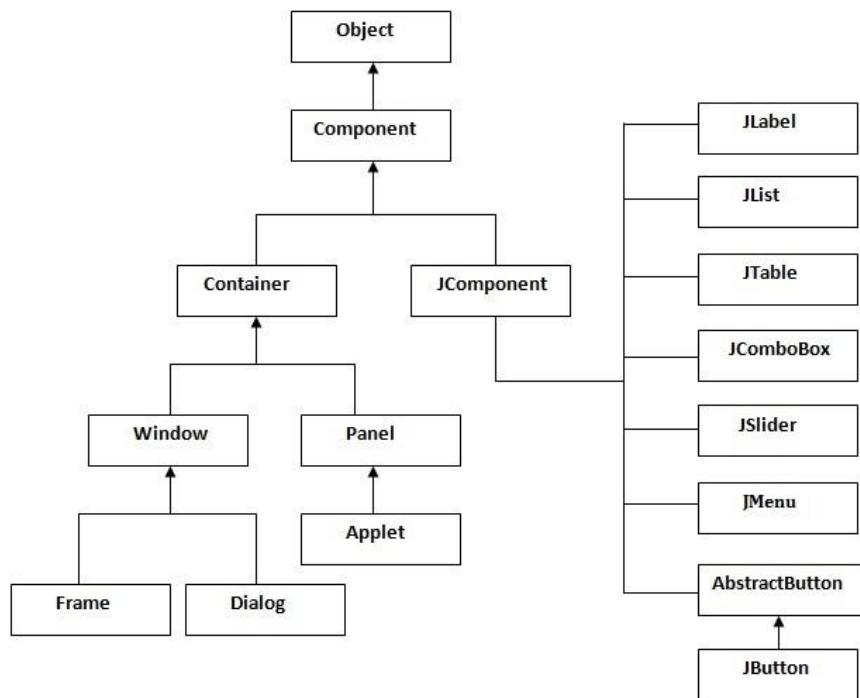
There are many differences between java awt and swing that are given below.

No.	Java AWT	Java Swing
1)	AWT components are platform-dependent.	Java swing components are platform-independent.
2)	AWT components are heavyweight.	Swing components are lightweight.
3)	AWT doesn't support pluggable look and feel.	Swing supports pluggable look and feel.
4)	AWT provides less components than Swing.	Swing provides more powerful components such as tables, lists, scrollpanes, colorchooser, tabbedpane etc.
5)	AWT doesn't follows MVC(Model View Controller) where model represents data, view represents presentation and controller acts as an interface between model and view.	Swing follows MVC.

JFC

The Java Foundation Classes (JFC) are a set of GUI components which simplify the development of desktop applications.

HIERARCHY OF JAVA SWING CLASSES



Example:

```

import javax.swing.*;

public class ButtonExample {
    public static void main(String[] args) {
        JFrame f=new JFrame("Button Example");
        JButton b=new JButton("Click Here");
        b.setBounds(50,100,95,30);
        f.add(b);
        f.setSize(400,400);
        f.setLayout(null);
        f.setVisible(true);
    }
}

```

Example: displaying image on the button

```

import javax.swing.*;

public class ButtonExample{
    ButtonExample(){
        JFrame f=new JFrame("Button Example");
        JButton b=new JButton(new ImageIcon("D:\\icon.png"));
        b.setBounds(100,100,100, 40);
    }
}

```

```

f.add(b);
f.setSize(300,400);
f.setLayout(null);
f.setVisible(true);
f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
}
public static void main(String[] args) {
    new ButtonExample();
}
}

```

JRadioButton

The JRadioButton class is used to create a radio button. It is used to choose one option from multiple options. It is widely used in exam systems or quiz. It should be added in ButtonGroup to select one radio button only.

Example:

```

import javax.swing.*.*;
import java.awt.event.*;
class RadioButtonExample extends JFrame implements ActionListener{
    JRadioButton rb1,rb2;
    JButton b;
    RadioButtonExample(){
        rb1=new JRadioButton("Male");
        rb1.setBounds(100,50,100,30);
        rb2=new JRadioButton("Female");
        rb2.setBounds(100,100,100,30);
        ButtonGroup bg=new ButtonGroup();
        bg.add(rb1);bg.add(rb2);
        b=new JButton("click");
        b.setBounds(100,150,80,30);
        b.addActionListener(this);
        add(rb1);add(rb2);add(b);
        setSize(300,300);
        setLayout(null);
        setVisible(true);
    }
}

```



```

public void actionPerformed(ActionEvent e){
    if(rb1.isSelected()){
        JOptionPane.showMessageDialog(this,"You are Male.");
    }
    if(rb2.isSelected()){
        JOptionPane.showMessageDialog(this,"You are Female.");
    }
}

public static void main(String args[]){
    new RadioButtonExample();
}}

```

JComboBox

The object of Choice class is used to show popup menu of choices. Choice selected by user is shown on the top of a menu. It inherits JComponent class.

Example:

```

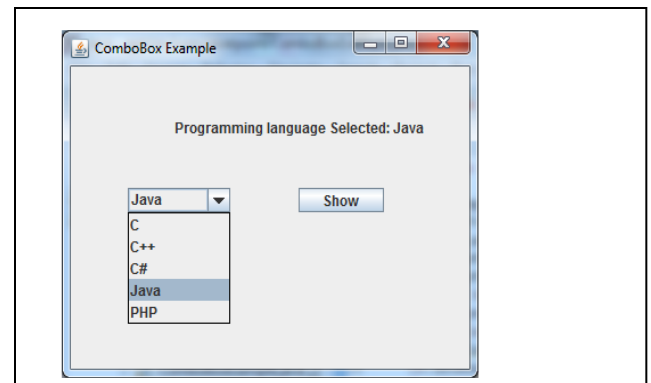
import javax.swing.*;
import java.awt.event.*;

public class ComboBoxExample {
    JFrame f;

    ComboBoxExample(){
        f=new JFrame("ComboBox Example");
        final JLabel label = new JLabel();
        label.setHorizontalAlignment(JLabel.CENTER);
        label.setSize(400,100);
        JButton b=new JButton("Show");
        b.setBounds(200,100,75,20);
        String languages[]={ "C","C++","C#","Java","PHP" };
        final JComboBox cb=new JComboBox(languages);
        cb.setBounds(50, 100,90,20);
        f.add(cb); f.add(label); f.add(b);
        f.setLayout(null);
        f.setSize(350,350);
        f.setVisible(true);

        b.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                String data = "Programming language Selected: "
                    + cb.getItemAt(cb.getSelectedIndex());
            }
        });
    }
}

```



```

label.setText(data);
}
});
}
public static void main(String[] args) {
    new ComboBoxExample();
}
}

```

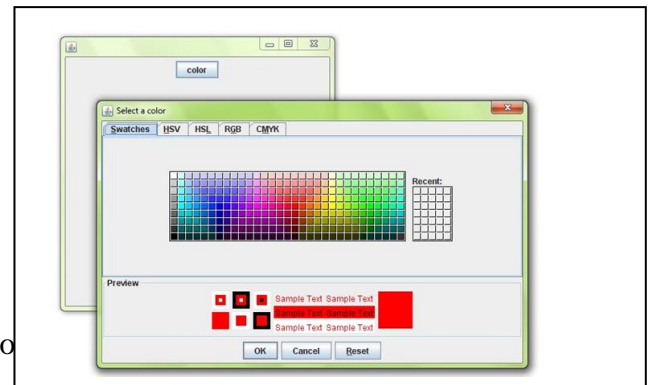
JColorChooser

The JColorChooser class is used to create a color chooser dialog box so that user can select any color. It inherits JComponent class.

```

import java.awt.event.*;
import java.awt.*;
import javax.swing.*;
public class ColorChooserExample extends JFrame implements ActionListener {
    JButton b;
    Container c;
    ColorChooserExample(){
        c=getContentPane();
        c.setLayout(new FlowLayout());
        b=new JButton("color");
        b.addActionListener(this);
        c.add(b);
    }
    public void actionPerformed(ActionEvent e) {
        Color initialcolor=Color.RED;
        Color color=JColorChooser.showDialog(this,"Select a color");
        c.setBackground(color);
    }
}

```



```

public static void main(String[] args) {
    ColorChooserExample ch=new ColorChooserExample();
    ch.setSize(400,400);
    ch.setVisible(true);
    ch.setDefaultCloseOperation(EXIT_ON_CLOSE);
}
}

```

APPLET

Applet is a java program that can be embedded in to HTML pages. Java applets runs on the enables web browsers such as Internet explorer, Mozilla firefox etc.

Applet is designed to run remotely on the client browser, so there are some rustications on it .applet can't access system resources on the local computer. Applet are used to make the website more dynamic and entertaining.

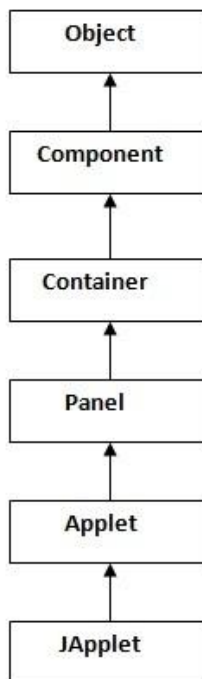
ADVANTAGES

1. Applet are cross platform and can run on windows ,Linux etc.
2. Applet can work all the versions of java plugin.
3. Applets are supported by most web browsers.
4. User can also have fall access to the machine if user allows.

DISADVANTAGES

- 1.Java plugin required to run applet.
- 2.Java applet required JVM(Java Virtual Machine).
- 3.It is difficult to design and build good .User interface in applets compared to HTML Technologys.

HIERARCHY OF APPLET



LIFECYCLE OF JAVA APPLET

1. Applet is initialized.
2. Applet is started.
3. Applet is painted.
4. Applet is stopped.
5. Applet is destroyed.

LIFECYCLE METHODS FOR APPLET:

The java.applet.Applet class 4 life cycle methods and java.awt.Component class provides 1 life cycle methods for an applet.

java.applet.Applet class

For creating any applet java.applet.Applet class must be inherited. It provides 4 life cycle methods of applet.

1. **public void init():** is used to initialize the Applet. It is invoked only once.
2. **public void start():** is invoked after the init() method or browser is maximized. It is used to start the Applet.
3. **public void stop():** is used to stop the Applet. It is invoked when Applet is stop or browser is minimized.
4. **public void destroy():** is used to destroy the Applet. It is invoked only once.

JAVA.AWT.COMPONENT CLASS

The Component class provides 1 life cycle method of applet.

1. **public void paint(Graphics g):** is used to paint the Applet. It provides Graphics class object that can be used for drawing oval, rectangle, arc etc.

Example: **First create a java program.**

```
import java.applet.Applet;
import java.awt.Graphics;
public class First extends Applet{
public void paint(Graphics g){
g.drawString("welcome",150,150);
}
}
```

Now design a HTML page to run the above java program.

```
<html>
<body>
<applet code="First.class" width="300" height="300">
</applet>
</body>
</html>
```

APPLETVIEWER TOOL

To execute the applet by appletviewer tool, create an applet that contains applet tag in comment and compile it. After that run it by: appletviewer First.java. Now Html file is not required but it is for testing purpose only.

Example:

```
import java.applet.Applet;
import java.awt.Graphics;
public class First extends Applet
{
    public void paint(Graphics g)
    {
        g.drawString("welcome to applet",150,150);
    }
}
/*
<applet code="First.class" width="300" height="300">
</applet>
*/
```

To execute the applet by appletviewer tool, write in command prompt:

```
c:\>javac First.java
c:\>appletviewer First.java
```

DISPLAYING GRAPHICS IN APPLETS

java.awt.Graphics class provides many methods for graphics programming.

Example:

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

```

public class GraphicsDemo extends Applet
{
public void paint(Graphics g)
{
g.setColor(Color.red);
g.drawString("Welcome",50, 50);
g.drawLine(20,30,20,300);
g.drawRect(70,100,30,30);
g.fillRect(170,100,30,30);
g.drawOval(70,200,30,30);
g.setColor(Color.pink);
g.fillOval(170,200,30,30); g.drawArc(90,150,30,30,30,270);
g.fillArc(270,150,30,30,0,180);
}
}

```

COMMONLY USED METHODS OF GRAPHICS CLASS:

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 boolean drawImage(Image img, int x, int y, ImageObserver observer):** is used draw the specified image.
8. **public abstract void drawArc(int x, int y, int width, int height, int startAngle, int arcAngle):** is used draw a circular or elliptical arc.
9. **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.
10. **public abstract void setColor(Color c):** is used to set the graphics current color to the specified color.
11. **public abstract void setFont(Font font):** is used to set the graphics current font to the specified font.

THREAD

In java, a single task is called a “thread”. The “thread of execution” meaning a sequence of instructions that are executed one after another, connecting each instruction to the next.

Each java program has at least one thread. The purpose of thread object is to execute a single method. The method is executed in its own thread of control, which can run in parallel with other threads. The thread is represented by an object of thread class which can be created in two ways:

- (a) Extending the java.lang.Thread class.
- (b) Implementing java.lang.Runnable interface.

If we extend java.lang.thread class, thread class is used to create a new class from which the threads could be created and run method could be created and run method could be defined. Thread class is extended to create a thread.

In the case of runnable interfaces the runnable interface is implemented in any class and thread can be created by using runnable interface.

MULTITHREADING

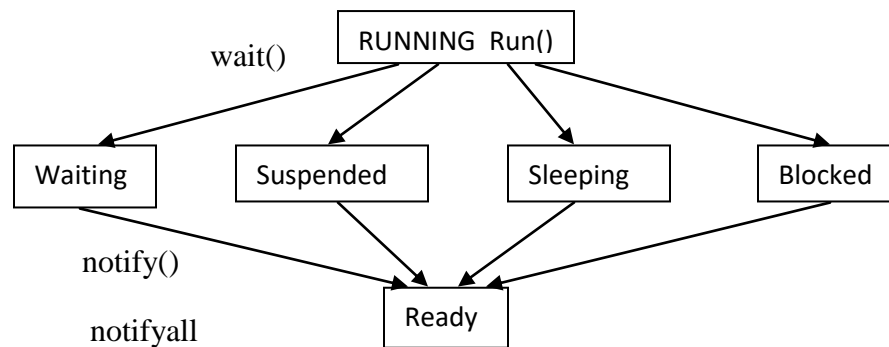
A multi threaded program contains two or more parts that can run concurrently. Each part of such a program is called “thread” and each of thread defines a separated path of execution. Multithreading is the specialized form of multitasking. Unlike other programming language , java provides built-in support for multithreaded programming.

The major advantages of multithreading:-

1. Multithreading require less overhead than multitasking processes. Because processes are heavy weight tasks that require their own separate address space. Inter process communication is also expensive and limited. Context switching from one process to another is also costly.
2. Threads are light-weight processes. They share same address space and cooperatively share the same heavy weight processes. Inter-threaded communication is inexpensive and context switching cost is also low.
3. Multithreading enables you to write very efficient programs that make maximum use of the CPU.

LIFE CYCLE OF THREAD

1. **BORN**:- The thread when created is called “Born” stage.
2. **START**:- The thread remains born until the thread starts. The start state cause the thread to enter in to the ready state.
3. **RUNNING**:- The thread start its execution when it is in the running stage. The execution of thread begins.
4. **STOP**:- The running of thread can be stopped in the stop state.
5. **WAITING**:- The running thread can acquire wait state and will be waiting for given time for further operation.
6. **SUSPEND**:- The running thread can be suspended, during this state the thread becomes non-operational.
7. **SLEEP**:- The sleep state is when the thread is in sleeping state. It becomes ready when sleep time is completed.
8. **BLOCKED**:- When thread goes to blocked state, when thread issues an I/O request. The blocked thread becomes ready when the I/O is waiting for completion.
9. **DEAD**:- After the stop state, thread enters to dead state and eventually removed from the system.



Thread Demonstration Program :-

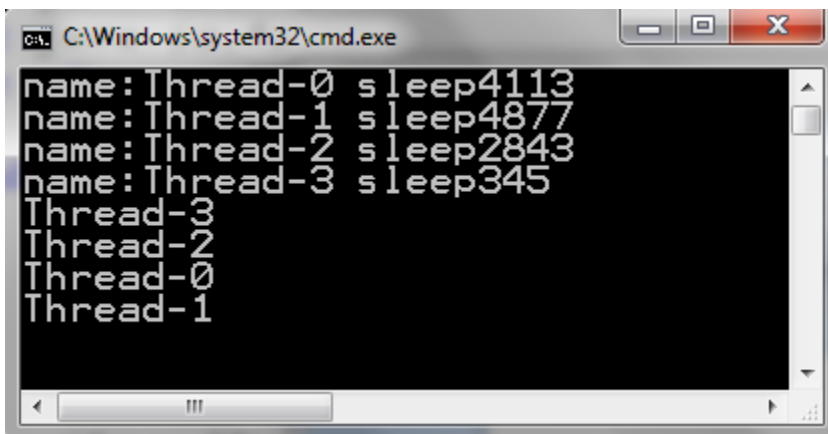
```
import java.awt.*;
import java.io.*;
public class exTh
{
    public static void main(String arg[])throws IOException
    {
        pThread thread1,thread2,thread3,thread4;
```

```

        thread1=new pThread();
        thread2=new pThread();
        thread3=new pThread();
        thread4=new pThread();
        thread1.start();
        thread2.start();
        thread3.start();
        thread4.start();
        System.in.read();
    }
}
class pThread extends Thread
{
    int sTime;
    public pThread()
    {
        sTime=(int)(Math.random()*5000);
        System.out.println("name:"+getName()+" sleep"+sTime);
    }
    public void run()
    {
        try
        {
            Thread.sleep(sTime);
        }
        catch(InterruptedException e)
        {
            System.out.println(e.toString());
        }
        System.out.println(getName());
    }
}

```

Output:-



```

C:\Windows\system32\cmd.exe
name:Thread-0 sleep4113
name:Thread-1 sleep4877
name:Thread-2 sleep2843
name:Thread-3 sleep345
Thread-3
Thread-2
Thread-0
Thread-1

```

COMMONLY USED METHODS OF THREAD CLASS:

1. **public void run():** is used to perform action for a thread.
2. **public void start():** starts the execution of the thread. JVM calls the run() method on the thread.
3. **public void sleep(long milliseconds):** Causes the currently executing thread to sleep (temporarily cease execution) for the specified number of milliseconds.
4. **public void join():** waits for a thread to die.
5. **public void join(long milliseconds):** waits for a thread to die for the specified milliseconds.
6. **public int getPriority():** returns the priority of the thread.
7. **public int setPriority(int priority):** changes the priority of the thread.
8. **public String getName():** returns the name of the thread.
9. **public void setName(String name):** changes the name of the thread.
10. **public Thread currentThread():** returns the reference of currently executing thread.
11. **public int getId():** returns the id of the thread.
12. **public Thread.State getState():** returns the state of the thread.
13. **public boolean isAlive():** tests if the thread is alive.
14. **public void yield():** causes the currently executing thread object to temporarily pause and allow other threads to execute.
15. **public void suspend():** is used to suspend the thread(deprecated).
16. **public void resume():** is used to resume the suspended thread(deprecated).
17. **public void stop():** is used to stop the thread(deprecated).
18. **public boolean isDaemon():** tests if the thread is a daemon thread.
19. **public void setDaemon(boolean b):** marks the thread as daemon or user thread.
20. **public void interrupt():** interrupts the thread.
21. **public boolean isInterrupted():** tests if the thread has been interrupted.
22. **public static boolean interrupted():** tests if the current thread has been interrupted.

THREAD PRIORITIES

Thread priorities are used by the thread scheduler to decide when each thread should be allowed to run. Generally, higher priority threads get more CPU time than lower priority threads but the amount of time of CPU that a thread gets often depend on several other factors besides its priority. A higher priority thread can be preempted and higher priority thread can preempt the lower priority thread.

In theory, thread of equal priorities should get equal access to the CPU. To set a thread's priority, use the set priority() method, which is a member of thread.

Syntax :- final void setPriority(int level);

The value of level must be within MIN-PRIORITY and MAX-PRIORITY. To obtain the current priority of any thread, use getPriority() method.

Syntax :- final int getPriority();

To return a thread to default priority use NORM_PRIORITY.

Example:

```
class TestMultiPriority1 extends Thread
{
    public void run( )
    {
        System.out.println("running thread name is:"+Thread.currentThread().getName());
        System.out.println("running thread priority is:"+Thread.currentThread().getPriority());
    }
    public static void main(String args[])
    {
        TestMultiPriority1 m1=new TestMultiPriority1();
        TestMultiPriority1 m2=new TestMultiPriority1();
        m1.setPriority(Thread.MIN_PRIORITY);
        m2.setPriority(Thread.MAX_PRIORITY);
        m1.start();
        m2.start();
    }
}
```

DAEMON THREAD

Daemon thread in java is a service provider thread that provides services to the user thread. Its life depend on the mercy of user threads i.e. when all the user threads dies, JVM terminates this thread automatically.

There are many java daemon threads running automatically e.g. gc, finalizer etc. User can see all the detail by typing the jconsole in the command prompt. The jconsole tool provides information about the loaded classes, memory usage, running threads etc.

Points to remember for Daemon Thread in Java

- It provides services to user threads for background supporting tasks. It has no role in life than to serve user threads.
- Its life depends on user threads.
- It is a low priority thread.

The sole purpose of the daemon thread is that it provides services to user thread for background supporting task. If there is no user thread, why should JVM keep running this thread. That is why JVM terminates the daemon thread if there is no user thread.

Example:

```
public class TestDaemonThread1 extends Thread
{
public void run(){
if(Thread.currentThread().isDaemon()){//checking for daemon thread
    System.out.println("daemon thread work");
}
else{
    System.out.println("user thread work");
}
}
public static void main(String[] args){
    TestDaemonThread1 t1=new TestDaemonThread1();//creating thread
    TestDaemonThread1 t2=new TestDaemonThread1();
    TestDaemonThread1 t3=new TestDaemonThread1();
    t1.setDaemon(true);//now t1 is daemon thread
    t1.start();//starting threads
    t2.start();
    t3.start();
} }
```

SYNCHRONIZATION

When two or more threads need access to a shared resource, they need some way to ensure that the resource will be used by only one thread at a time, the process by which this is achieved is called “Synchronization”. This is supported by java key to synchronization is the concept of monitor (also called semaphore). A monitor is an object that is used as a mutually exclusive lock or mutex. Only one thread can use the monitor at a time. When a thread acquires a lock, it is said to have entered in to the monitor. All the other threads trying to enter into the monitor will be suspended until the first thread exits the monitor. Other threads are in waiting state now.

Generally, as with other languages C,C++ etc. synchronization is not supported by directly. Instead, to synchronize threads, your program needs to utilize operating system synchronization but java implements synchronization through language elements, so most of the complexity has been eliminated.

There are two types of thread synchronization mutual exclusive and inter-thread communication.

1. Mutual Exclusive

1. Synchronized method.
2. Synchronized block.
3. static synchronization.

2. Cooperation (Inter-thread communication in java)

Synchronization is built around an internal entity known as the lock or monitor. Every object has an lock associated with it. By convention, a thread that needs consistent access to an object's fields has to acquire the object's lock before accessing them, and then release the lock when it's done with them.

MUTUAL EXCLUSIVE

Mutual Exclusive helps keep threads from interfering with one another while sharing data. This can be done by three ways in java:

1. **BY SYNCHRONIZED METHOD**- If you declare any method as synchronized, it is known as synchronized method. Synchronized method is used to lock an object for any shared resource. When a thread invokes a synchronized method, it automatically acquires the lock for that object and releases it when the thread completes its task.

Example:

```
class Table{
synchronized void printTable(int n){//synchronized method
for(int i=1;i<=5;i++){
    System.out.println(n*i);
```

```

try{
    Thread.sleep(400);
}catch(Exception e){System.out.println(e);}
}
} }

class MyThread1 extends Thread{
Table t;
MyThread1(Table t){
    this.t=t;
}
public void run(){
    t.printTable(5);
}

}

class MyThread2 extends Thread{
Table t;
MyThread2(Table t){
    this.t=t;
}
public void run(){
    t.printTable(100);
}
}

public class TestSynchronization2{
public static void main(String args[]){
    Table obj = new Table();//only one object
    MyThread1 t1=new MyThread1(obj);
    MyThread2 t2=new MyThread2(obj);
    t1.start();
    t2.start();
}
}

```

BY SYNCHRONIZED BLOCK- Synchronized block can be used to perform synchronization on any specific resource of the method. Suppose you have 50 lines of code in your method, but you want to synchronize only 5 lines, you can use synchronized block. If you put all the codes of the method in the synchronized block, it will work same as the synchronized method.

Points to remember for Synchronized block

- Synchronized block is used to lock an object for any shared resource.
- Scope of synchronized block is smaller than the method.

Example:

```
class Table
{
    void printTable(int n)
    {
        synchronized(this){//synchronized block
            for(int i=1;i<=5;i++){
                System.out.println(n*i);
                try{
                    Thread.sleep(400);
                }catch(Exception e){System.out.println(e);}
            }
        }
    }
}

class MyThread1 extends Thread{
    Table t;
    MyThread1(Table t){
        this.t=t;
    }
    public void run(){
        t.printTable(5);
    }
}

class MyThread2 extends Thread{
    Table t;
    MyThread2(Table t){
        this.t=t;
    }
    public void run(){
        t.printTable(100);
    }
}

public class TestSynchronizedBlock1 {
    public static void main(String args[]){
        Table obj = new Table();//only one object
        MyThread1 t1=new MyThread1(obj);
        MyThread2 t2=new MyThread2(obj);
        t1.start();
        t2.start();
    }
}
```

EXCEPTION HANDLING

EXCEPTION

An exception is a problem that arises during the execution of a program. An exception can occur for many different reasons, including the following:

- A user has entered 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.

To implement exception handling java uses five things-

- (a) Try Block
- (b) Catch Block
- (c) Throw keyword
- (d) Throws method
- (e) Finally

CATEGORIES OF EXCEPTIONS

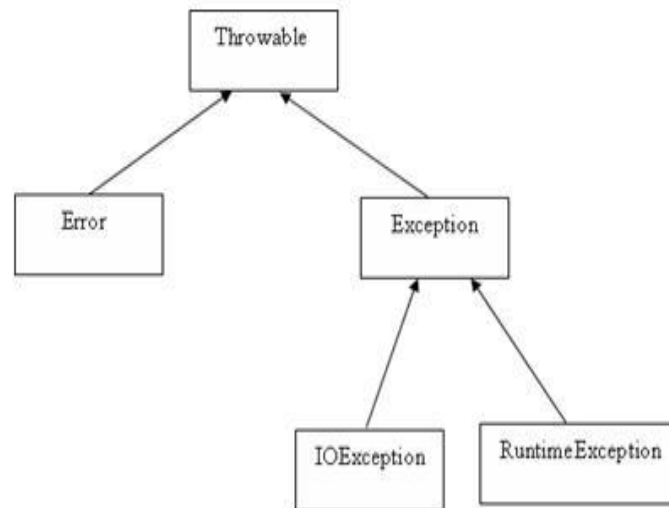
- **Checked exceptions:** A checked exception is an exception that is typically a user error or a problem that cannot be foreseen by the programmer. For example, if a file is to be opened, but the file cannot be found, an exception occurs. These exceptions cannot simply be ignored at the time of compilation.
- **Runtime exceptions:** A runtime exception is an exception that occurs that probably could have been avoided by the programmer. As opposed to checked exceptions, runtime exceptions are ignored at the time of compilation.
- **Errors:** These are not exceptions at all, but problems that arise beyond the control of the user or the programmer. Errors are typically ignored in your code because you can rarely do anything about an error. For example, if a stack overflow occurs, an error will arise. They are also ignored at the time of compilation.

EXCEPTION HIERARCHY

All exception classes are subtypes of the java.lang.Exception class. The exception class is a subclass of the Throwable class. Other than the exception class there is another subclass called Error which is derived from the Throwable class.

- Errors are not normally trapped from the Java programs. These conditions normally happen in case of severe failures, which are not handled by the java programs. Errors are generated to indicate errors generated by the runtime environment. Example : JVM is out of Memory. Normally programs cannot recover from errors.

- The Exception class has two main subclasses : IOException class and RuntimeException Class.



EXCEPTIONS METHODS

Following is the list of important methods available in the Throwable class.

SN	Methods with Description
1	public String getMessage() Returns a detailed message about the exception that has occurred. This message is initialized in the Throwable constructor.
2	public Throwable getCause() Returns the cause of the exception as represented by a Throwable object.
3	public String toString() Returns the name of the class concatenated with the result of getMessage()
4	public void printStackTrace() Prints the result of toString() along with the stack trace to System.err, the error output stream.
5	public StackTraceElement [] getStackTrace() Returns an array containing each element on the stack trace. The element at index 0 represents the top of the call stack, and the last element in the array represents the method at the bottom of the call stack.
6	public Throwable fillInStackTrace() Fills the stack trace of this Throwable object with the current stack trace, adding to any previous information in the stack trace.

TRY BLOCK:

This block is used to trap the exception. We put that code in this block whose exception we want or need to trap. It is followed by one or more catch blocks. This block always run whether some exception is in that code or not.

Syntax:

```
try
{
    code...
    ....
    ....
}
```

CATCH BLOCK:

This block is used to catch and handle the exceptions occurred in try block. The catch keyword indicates the catching of an exception. This block runs only when some exception occurs in try block.

TRY AND CATCH EXAMPLE:

```
import java.io.*;
public class exceptionHandle
{
    public static void main(String[] args) throws Exception
    {
        try
        {
            int a,b;
            BufferedReader in =
            new BufferedReader(new InputStreamReader(System.in));
            a = Integer.parseInt(in.readLine());
            b = Integer.parseInt(in.readLine());
        }
        catch(NumberFormatException ex)
        {
            System.out.println(ex.getMessage() + " is not a numeric value.");
            System.exit(0);
        }
    }
}
```

MULTIPLE CATCH BLOCKS:

```
class MultipleCatch
{
public static void main(String args[])
{
try
{
int den = Integer.parseInt(args[0]);
System.out.println(3/den);
}
catch (ArithmeticException exc)
{
System.out.println("Divisor was 0.");
}
catch (ArrayIndexOutOfBoundsException exc2)
{
System.out.println("Missing argument.");
}
System.out.println("After exception.");
}
}
```

NESTED TRY BLOCK

The try block within a try block is known as nested try block in java.

Syntax:

```
try
{
    statement 1;
    statement 2;
    try
    {
        statement 1;
        statement 2;
    }
    catch(Exception e)
    {
    }
}
catch(Exception e)
```

```
{  
}
```

....

Example:

```
class nestedtry{  
    public static void main(String args[]){  
        try{  
            try{  
                System.out.println("going to divide");  
                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("handeled");}  
        System.out.println("normal flow..");  
    }  
}
```

THROW:

The Java throw keyword is used to explicitly throw an exception. We can throw either checked or unchecked exception in java by throw keyword. The throw keyword is mainly used to throw custom exception.

Example:

```
public class TestThrow1{  
    static void validate(int age){  
        if(age<18)  
            throw new ArithmeticException("not valid");  
        else  
            System.out.println("welcome to vote");  
    }  
    public static void main(String args[]){  
        validate(13);  
        System.out.println("rest of the code...");  
    } }  
}
```

THROWS:

The Java throws keyword is used to declare an exception. It gives an information to the programmer that there may occur an exception so it is better for the programmer to provide the exception handling code so that normal flow can be maintained.

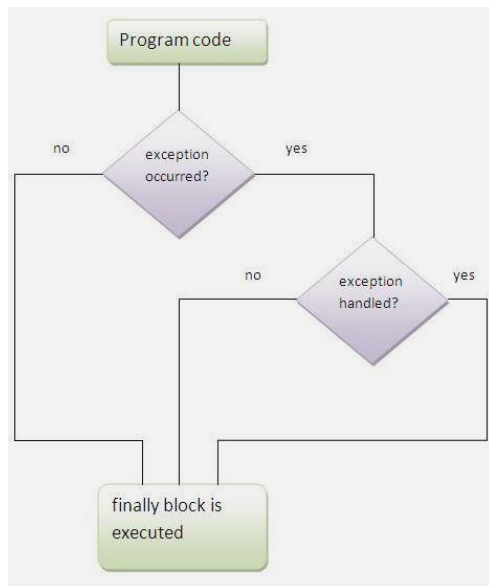
Exception Handling is mainly used to handle the checked exceptions. If there occurs any unchecked exception such as NullPointerException, it is programmers fault that he is not performing check up before the code being used.

Example:

```
import java.io.IOException;
class Testthrows1{
    void m()throws IOException{
        throw new IOException("device error");//checked exception
    }
    void n()throws IOException{
        m();
    }
    void p(){
        try{
            n();
        }catch(Exception e){System.out.println("exception handled");}
    }
    public static void main(String args[]){
        Testthrows1 obj=new Testthrows1();
        obj.p();
        System.out.println("normal flow...");
    }
}
```

FINALLY:

Java finally block is a block that is used *to execute important code* such as closing connection, stream etc. Java finally block is always executed whether exception is handled or not. Java finally block follows try or catch block.



Example:

```

class TestFinallyBlock{
    public static void main(String args[]){
        try{
            int data=25/5;
            System.out.println(data);
        }
        catch(NullPointerException e){System.out.println(e);}
        finally{System.out.println("finally block is always executed");}
        System.out.println("rest of the code...");
    }
}
  
```

DIFFERENCE BETWEEN THROW AND THROWS

There are many differences between throw and throws keywords. A list of differences between throw and throws are given below:

No.	throw	throws
1)	Java throw keyword is used to explicitly throw an exception.	Java throws keyword is used to declare an exception.
2)	Checked exception cannot be propagated using throw only.	Checked exception can be propagated with throws.
3)	Throw is followed by an instance.	Throws is followed by class.
4)	Throw is used within the method.	Throws is used with the method signature.
5)	You cannot throw multiple exceptions.	You can declare multiple exceptions e.g. public void method()throws IOException,SQLException.

JDBC

JDBC is Java application programming interface that allows the Java programmers to access database management system from Java code. It was developed by JavaSoft, a subsidiary of Sun Microsystems.

Java Database Connectivity in short called as JDBC. It is a java API which enables the java programs to execute SQL statements. It is an application programming interface that defines how a java programmer can access the database in tabular format from Java code using a set of standard interfaces and classes written in the Java programming language.

JDBC has been developed under the Java Community Process that allows multiple implementations to exist and be used by the same application. JDBC provides methods for querying and updating the data in Relational Database Management system such as SQL, Oracle etc.

The Java application programming interface provides a mechanism for dynamically loading the correct Java packages and drivers and registering them with the JDBC Driver Manager that is used as a connection factory for creating JDBC connections which supports creating and executing statements such as SQL INSERT, UPDATE and DELETE. Driver Manager is the backbone of the jdbc architecture.

Generally all Relational Database Management System supports SQL and we all know that Java is platform independent, so JDBC makes it possible to write a single database application that can run on different platforms and interact with different Database Management Systems.

Java Database Connectivity is similar to Open Database Connectivity (ODBC) which is used for accessing and managing database, but the difference is that JDBC is designed specifically for Java programs, whereas ODBC is not depended upon any language.

In short JDBC helps the programmers to write java applications that manage these three programming activities:

1. It helps us to connect to a data source, like a database.
2. It helps us in sending queries and updating statements to the database and
3. Retrieving and processing the results received from the database in terms of answering to your query.

JDBC COMPONENTS

JDBC has four Components:

1. The JDBC API.
2. The JDBC Driver Manager.
3. The JDBC Test Suite.
4. The JDBC-ODBC Bridge.

1. The JDBC API.

The JDBC application programming interface provides the facility for accessing the relational database from the Java programming language. The API technology provides the industrial standard for independently connecting Java programming language and a wide range of databases. The user not only execute the SQL statements, retrieve results, and update the data but can also access it anywhere within a network because of its "Write Once, Run Anywhere" (WORA) capabilities.

Due to JDBC API technology, user can also access other tabular data sources like spreadsheets or flat files even in the a heterogeneous environment. JDBC application programming interface is a part of the Java platform that have included Java Standard Edition (Java SE) and the Java Enterprise Edition (Java EE) in itself.

The JDBC API has four main interface:

The latest version of JDBC 4.0 application programming interface is divided into two packages

i-) java.sql

ii-) javax.sql.

Java SE and Java EE platforms are included in both the packages.

2. The JDBC Driver Manager.

The JDBC Driver Manager is a very important class that defines objects which connect Java applications to a JDBC driver. Usually Driver Manager is the backbone of the JDBC architecture. It's very simple and small that is used to provide a means of managing the different types of JDBC database driver running on an application. The main responsibility of JDBC database driver is to load all the drivers found in the system properly as well as to select the most appropriate driver from opening a connection to a database. The Driver Manager also helps to select the most appropriate driver from the previously loaded drivers when a new open database is connected.

3. THE JDBC TEST SUITE.

The function of JDBC driver test suite is to make ensure that the JDBC drivers will run user's program or not . The test suite of JDBC application program interface is very useful for testing a driver based on JDBC technology during testing period. It ensures the requirement of Java Platform Enterprise Edition (J2EE).

4. THE JDBC-ODBC BRIDGE.

The JDBC-ODBC bridge, also known as JDBC type 1 driver is a database driver that utilize the ODBC driver to connect the database. This driver translates JDBC method calls into ODBC function calls. The Bridge implements Jdbc for any database for which an Odbc driver is available. The Bridge is always implemented as the sun.jdbc.odbc Java package and it contains a native library used to access ODBC.

Now we can conclude this topic: This first two component of JDBC, the JDBC API and the JDBC Driver Manager manages to connect to the database and then build a java program that utilizes SQL commands to communicate with any RDBMS. On the other hand, the last two components are used to communicate with ODBC or to test web application in the specialized environment.

JDBC DRIVERS

JDBC Driver is a software component that enables java application to interact with the database. There are 4 types of JDBC drivers:

1. JDBC-ODBC bridge driver
2. Native-API driver (partially java driver)
3. Network Protocol driver (fully java driver)
4. Thin driver (fully java driver)

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.

It is easy to use and it can be easily connected to any database. But the Performance degraded because JDBC method call is converted into the ODBC function calls. The ODBC driver needs to be installed on the client machine.

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.

Its performance is upgraded than JDBC-ODBC bridge driver. But The Native driver needs to be installed on each client machine. 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.

No client side library is required because of application server that can perform many tasks like auditing, load balancing, logging etc. But Network support is required on client machine.

Requires database-specific coding to be done in the middle tier. Maintenance of Network Protocol driver becomes costly because it requires database-specific coding to be done in the middle tier.

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.

Better performance than all other drivers. No software is required at client side or server side. But Drivers depends on the Database.

5 STEPS TO CONNECT TO THE DATABASE

There are 5 steps to connect any java application with the database in java using JDBC. They are as follows:

- 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.
- Creating connection- The `getConnection()` method of `DriverManager` class is used to establish connection with the database.
- Creating statement- The `createStatement()` method of `Connection` interface is used to create statement. The object of statement is responsible to execute queries with the database.
- Executing queries- The `executeQuery()` method of `Statement` interface is used to execute queries to the database. This method returns the object of `ResultSet` that can be used to get all the records of a table.
- Closing connection- By closing connection object statement and `ResultSet` will be closed automatically. The `close()` method of `Connection` interface is used to close the connection.

CONNECTION WITH MS ACCESS

```
import java.sql.*;
class Test{
public static void main(String ar[]){
try{
String database="student.mdb";//Here database exists in the current directory
String url="jdbc:odbc:Driver={Microsoft Access Driver (*.mdb)};
        DBQ="+ database + ";DriverID=22;READONLY=true";
```

```

    Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
    Connection c=DriverManager.getConnection(url);
    Statement st=c.createStatement();
    ResultSet rs=st.executeQuery("select * from login");
    while(rs.next()){
        System.out.println(rs.getString(1));
    }
} catch (Exception ee) { System.out.println(ee); }
}
}

```

PREPAREDSTATEMENT INTERFACE

```

import java.sql.*;
class InsertPrepared{
    public static void main(String args[]){
        try{
            Class.forName("oracle.jdbc.driver.OracleDriver");
            Connection con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:xe",system",
            oracle");
            preparedStatement stmt=con.prepareStatement("insert into Emp values(?,?)");
            stmt.setInt(1,101);//1 specifies the first parameter in the query
            stmt.setString(2,"Ratan");
            int i=stmt.executeUpdate();
            System.out.println(i+" records inserted");
            con.close();
        } catch (Exception e) { System.out.println(e); }
        }
    }
}

```

IO PACKAGE

Java I/O (Input and Output) is used *to process the input and produce the output*. Java uses the concept of stream to make I/O operation fast. The java.io package contains all the classes required for input and output operations. We can perform **file handling in java** by Java I/O API.

STREAM

A stream is a sequence of data. In Java a stream is composed of bytes. It's called a stream because it is like a stream of water that continues to flow.

In java, 3 streams are created for us automatically. All these streams are attached with console.

1) **System.out**: standard output stream

2) **System.in**: standard input stream

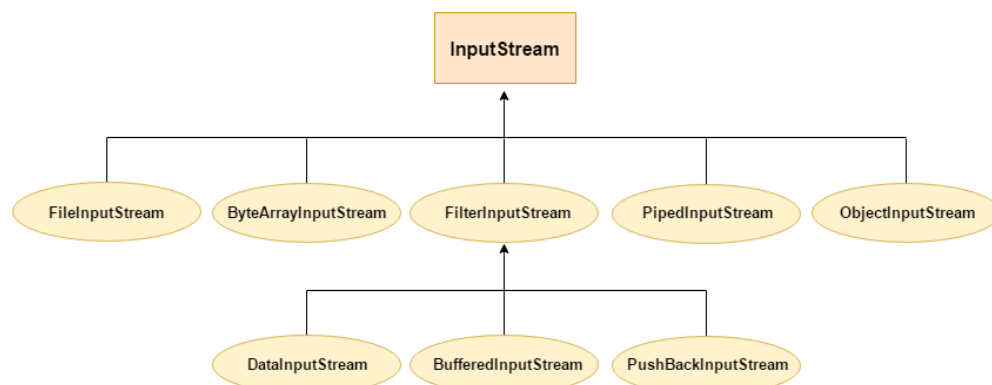
3) **System.err**: standard error stream

INPUT STREAM

The Input Stream and Output Stream classes are the base classes for byte oriented I/O in java. The Input Stream is an abstract class that defines java's model of streaming byte input. All the methods of this class will throw an I/o Exception.

Methods Of Input Stream Class :-

	Methods	Work
1.	Int available()	It gets the number of bytes that can be read from the input stream.
2.	void close()	It closes the input stream.
3.	void mark(int read limit)	It marks the current location in this input stream.
4.	abstract(int read())	It reads the next byte of data from the input stream.
5.	void reset()	It repositions the stream to the position where the mark method was last called.



OUTPUT STREAM

The counterpart of input stream class is output stream class, on which you base output streams.

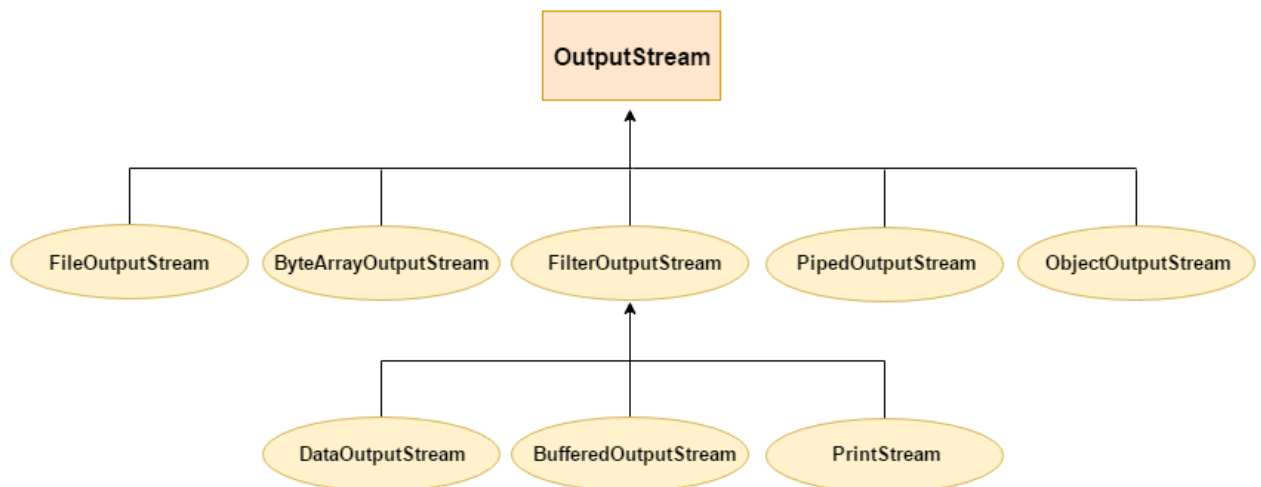
java.lang.Object

└─ java.io.OutputStream

The output stream is an abstract class that defines streaming byte output. All the methods of this class return void and throw IOException. While Input stream implements the closeable interface, Output stream implements the closeable and Flushable interface.

Methods Of Output Stream Class :-

	Methods	Work
1.	void close()	It closes the Output stream.
2.	void flush()	It flushes the output stream and writes any waiting buffered output bytes.
3.	void write(byte[]b)	It writes b.length bytes from the given byte array to this output stream.
4.	Abstract void write(int b)	It writes the given byte to this output stream.



FILE OUTPUT STREAM

This class is used to write data, byte by byte, to a file. They can throw a FileNotFoundException or a SecurityException. Creation of a FileOutputStream is not dependent on the file that already exists. Infact File Output Stream will create the file before opening it for output. If you attempt to open a read only file, it will throw an IOException. All File Output Stream methods have a return type of void.

Methods Of File Output Stream Class :-

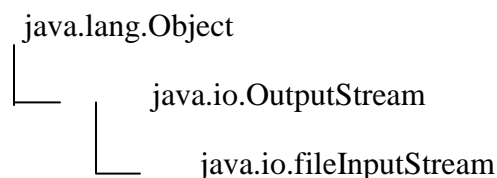
	Methods	Work
1.	void close()	It closes this File Output stream.
2.	protected void finalize ()	It make sure that the close method of this file output stream is called where there are no more reference.
3.	FileDescriptor getFD()	It gets the file descriptor associated with this stream.
4.	void write(int b)	It writes the given byte to this file output stream.
5.	File Channel getChannel()	It gets the unique File Channel Object associated with this file Output stream.

Example:

```
import java.io.FileOutputStream;
public class FileOutputStreamExample {
    public static void main(String args[]){
        try{
            FileOutputStream fout=new FileOutputStream("D:\\testout.txt");
            fout.write(65);
            fout.close();
            System.out.println("success...");
        }catch(Exception e){System.out.println(e);}
    }
}
```

FILE INPUT STREAM

The class is specially designed to work with byte oriented input from fields and it is derived from Input stream.



All the construtors can throw a FileNotFoundException. To create a file input stream, you use the File Input stream class's construtor.

Methods Of File Input Stream Class :-

	Methods	Work
1.	int available()	It gets the number of bytes that can be read from the file input stream.
2.	void close()	It closes this File Output stream.

3.	protected void finalize ()	It make sure that the close method of the file input stream is called where there are no more references to it.
4.	FileDescriptor getFD()	It gets the file descriptor object.
5.	long skip(long n)	It skips over and discard n bytes.
6.	int read()	It reads a byte of data from this input stream.

Example:

```
import java.io.FileInputStream;
public class DataStreamExample {
    public static void main(String args[]){
        try{
            FileInputStream fin=new FileInputStream("D:\\testout.txt");
            int i=fin.read();
            System.out.print((char)i);

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

RANDOM ACCESS FILE

This class is used to move around in a file using the seek method, it means being able to move around in a file at random.

java.lang.Object

java.io.Random Access File

Random Access File is not derived from Input stream or Output stream rather it encapsulates a random-access file. It also implements the closeable interface. Random-Access File supports positioning requests.

Types of Access:-

- **r:-** File is read not written.
- **w:-** File opened in read-write mode.
- **rcos:-** File opened for write-read operation and every change to the content written synchronously to the underlying storage device.

Methods:-

	Methods	Work
1.	void write(byte []b)	It writes b.length bytes from the given byte array to this file, starting at file pointer.
2.	void write(int b)	It writes the given byte to this file.
3.	void writeByte(int v)	It writes a byte to the file as a 1-byte value.
4.	void writeChar(int v)	It writes a char to the file as a 2-byte value.
5.	void writeByte(int v)	It writes a integer to the file as a 4-byte value.

BUFFEREDOUTPUTSTREAM CLASS

Java BufferedOutputStream class is used for buffering an output stream. It internally uses buffer to store data. It adds more efficiency than to write data directly into a stream. So, it makes the performance fast.

For adding the buffer in an OutputStream, use the BufferedOutputStream class.

BufferedOutputStream class methods

Method	Description
void write(int b)	It writes the specified byte to the buffered output stream.
void write(byte[] b, int off, int len)	It write the bytes from the specified byte-input stream into a specified byte array, starting with the given offset
void flush()	It flushes the buffered output stream.

Example:

```
package com.javatpoint;
import java.io.*;
public class BufferedOutputStreamExample{
public static void main(String args[])throws Exception{
    FileOutputStream fout=new FileOutputStream("D:\\testout.txt");
    BufferedOutputStream bout=new BufferedOutputStream(fout);
    String s="Welcome to javaTpoint.";
    byte b[]=s.getBytes();
    bout.write(b);
    bout.flush();
}
```

```

    bout.close();
    fout.close();
    System.out.println("success");
}
}

```

BUFFEREDINPUTSTREAM CLASS

Java BufferedInputStream class is used to read information from stream. It internally uses buffer mechanism to make the performance fast.

The important points about BufferedInputStream are:

- When the bytes from the stream are skipped or read, the internal buffer automatically refilled from the contained input stream, many bytes at a time.
- When a BufferedInputStream is created, an internal buffer array is created.

Example:

```

import java.io.*;
public class BufferedInputStreamExample{
    public static void main(String args[]){
        try{
            FileInputStream fin=new FileInputStream("D:\\testout.txt");
            BufferedInputStream bin=new BufferedInputStream(fin);
            int i;
            while((i=bin.read())!=-1){
                System.out.print((char)i);
            }
            bin.close();
            fin.close();
        }catch(Exception e){System.out.println(e);}
    }
}

```

DATAINPUTSTREAM CLASS

Java DataInputStream class allows an application to read primitive data from the input stream in a machine-independent way. Java application generally uses the data output stream to write data that can later be read by a data input stream.

Example:

```

import java.io.*;
public class DataStreamExample {
    public static void main(String[] args) throws IOException {
        InputStream input = new FileInputStream("D:\\testout.txt");
        DataInputStream inst = new DataInputStream(input);
        int count = input.available();
        byte[] ary = new byte[count];
        inst.read(ary);
        for (byte bt : ary) {
            char k = (char) bt;
            System.out.print(k+"-");
        }
    }
}

```

DATAOUTPUTSTREAM CLASS

Java DataOutputStream class allows an application to write primitive Java data types to the output stream in a machine-independent way. Java application generally uses the data output stream to write data that can later be read by a data input stream.

Example;

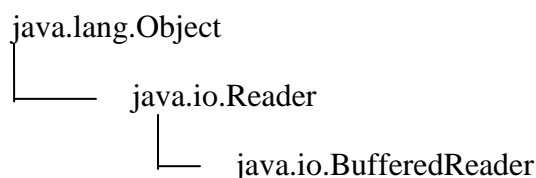
```

import java.io.*;
public class OutputExample {
    public static void main(String[] args) throws IOException {
        FileOutputStream file = new FileOutputStream(D:\\testout.txt);
        DataOutputStream data = new DataOutputStream(file);
        data.writeInt(65);
        data.flush();
        data.close();
        System.out.println("Success...");
    }
}

```

BUFFERED READER

This class provides a buffered character reader class. One of its advantage is that Buffered Reader provides the readline method.



Buffered Reader has two constructors. In the first form, a buffered character stream uses a default buffer size. In the second, the size of the buffer is passed in sz.

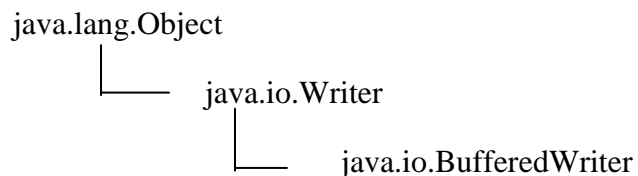
	Methods	Work
1.	void close()	It closes the stream.
2.	int read()	It reads a single character.
3.	int read(char[] cbuf, int off, int len)	They read characters in to a position of an array.
4.	String readLine()	It reads a line of text.
5.	void reset()	It resets the stream to the most recent mark.
6.	long skip(long n)	It skips characters.

Example:-

```
import java.io.*;
class BufferedReaderDemo
{
    public static void main(String arg[])throws Exception
    {
        FileReader fileredreader=new FileReader("File4.txt");
        BufferedReader bufferReader=new BufferedReader(fileredreader);
        String instring;
        while((instring=bufferedReader.readLine())!=null)
        {
            System.out.println(instring);
        }
        filereader.close();
    }
}
```

BUFFERED WRITER

This is the counter part class of BufferedReader class.



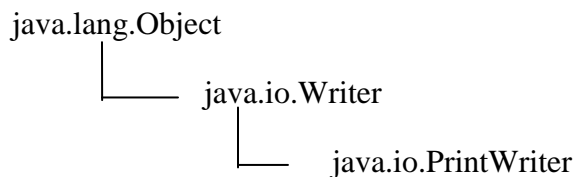
	Methods	Work
1.	void close()	It closes the stream.
2.	void Flush()	It flushes the stream.
3.	void newLine()	It writes a line separator.
4.	void write(char[] cbuf,int off,int len)	It writes a section of an array of characters.
5.	void write(int c)	It writes the single character.
6.	void write(String s,int off,int len)	It writes the portion of a string.

Example:

```
import java.io.*;
public class BufferedWriterExample {
public static void main(String[] args) throws Exception {
    FileWriter writer = new FileWriter("D:\\testout.txt");
    BufferedWriter buffer = new BufferedWriter(writer);
    buffer.write("Welcome to javaTpoint.");
    buffer.close();
    System.out.println("Success");
}
}
```

PRINT WRITER

The methods of this class are used to print formatted representation of objects to a text output stream. Methods of this class never throw `IOException`. If automatic flushing is enabled, that will take place only when one of the `println()`, `printf` or `format()` method is invoked. We can check some errors using `checkError()` method.



	Methods	Work
1.	<code>PrintWriter</code>	It appends the given character sequence to the writer.
2.	<code>boolean checkError()</code>	It flushes the stream if not closed & check its error states.
3.	<code>void clearError()</code>	It clears error state of the stream.
4.	<code>void close()</code>	It closes stream.
5.	<code>void Flush()</code>	It flushes the stream.
6.	<code>void print(boolean b)</code>	Prints a boolean value.
7.	<code>void print(char c)</code>	It prints a character.
8.	<code>void println()</code>	Terminates the current line by writing the line separator string.
9.	<code>void write(String s)</code>	It writes the a string.
10.	<code>void write(int c)</code>	It writes the single character.

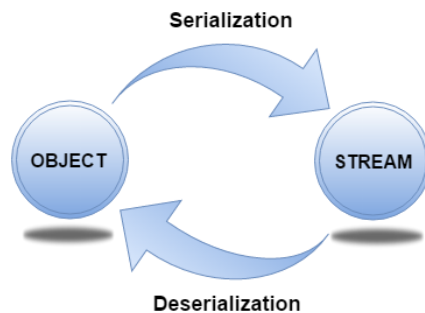
Example:-

```
import java.io.*;
import java.io.PrintWriter;
public class PrintWriterDemo
{
    public static void main(String arg[])
    {
        {
            PrintWriter out=new PrintWriter(System.out,true);
            out.println("Hello");
        }
    }
}
```

SERIALIZATION

Serialization is the process of writing objects to a stream and reading them back. This is particularly useful when you want to save the state of your program to a persistent storage area, such as a file. Later on, though, you may restore these objects by using the process of deserialization. For serialization, serializable interface is implemented.

Serialization in java is a mechanism of *writing the state of an object into a byte stream*. It is mainly used in Hibernate, RMI, JPA, EJB and JMS technologies. The reverse operation of serialization is called *deserialization*. It is mainly used to travel object's state on the network (known as marshaling).



java.io.Serializable interface

Serializable is a marker interface (has no data member and method). It is used to "mark" java classes so that objects of these classes may get certain capability. The Cloneable and Remote are also marker interfaces. It must be implemented by the class whose object you want to persist. The String class and all the wrapper classes implement the *java.io.Serializable* interface by default.

Example:

```
import java.io.Serializable;
public class Student implements Serializable{
    int id;
    String name;
    public Student(int id, String name) {
        this.id = id;
        this.name = name;
    }
}
```

ObjectOutputStream class

The ObjectOutputStream class is used to write primitive data types and Java objects to an OutputStream. Only objects that support the java.io.Serializable interface can be written to streams.

Important Methods

Method	Description
1) public final void writeObject(Object obj) throws IOException { }	writes the specified object to the ObjectOutputStream.
2) public void flush() throws IOException { }	flushes the current output stream.
3) public void close() throws IOException { }	closes the current output stream.

Example:

```
import java.io.*;
class Persist{
    public static void main(String args[])throws Exception{
        Student s1 =new Student(211,"ravi");
        FileOutputStream fout=new FileOutputStream("f.txt");
        ObjectOutputStream out=new ObjectOutputStream(fout);
        out.writeObject(s1);
        out.flush();
        System.out.println("success");
    }
}
```

DESERIALIZATION

Deserialization is the process of reconstructing the object from the serialized state.It is the reverse operation of serialization.

ObjectInputStream class

An ObjectInputStream deserializes objects and primitive data written using an ObjectOutputStream.

Example:

```
import java.io.*;
class Depersist{
    public static void main(String args[])throws Exception{
        ObjectInputStream in=new ObjectInputStream(new FileInputStream("f.txt"));
        Student s=(Student)in.readObject();
        System.out.println(s.id+" "+s.name);
        in.close();
    }
}
```

JAVA NETWORKING AND RMI

JAVA NETWORKING PACKAGE(.net.*)

The Java Networking package is one of the package include in the Java Development Kit (JDK) . It contains collection of related classes and interfaces that are required for networking . Actually these classes help us to built application that can communicate with eachother as well as transfer files or information to each other.

Java networking is performed using using TCP/IP Protocol i.e Transmission Control Protocol /Internet Protocol or UDP (User Datagram Protocol) .

An important advantage of Java networking is that it enables to establish a connection which can be customised according to requirements.

The TCP/IP connections are established using sockets and sever sockets. These connections allow to send and receive data to and receive data to a website. On the other hand the UDP connections are established using datagram sockets. These connections alooow to send data to other hosts over the network and that to establishing a connection with the host.

Following are the various classes included in the Networking Package:

- 1) Authenticator – This class represents an object that knows how to obtain authentication for a network connection.
- 2) DatagramPacket – This class represents a socket for sending and recieving datagram packets.
- 3) InetSocketAddress – This class represents an IP Socket Address.
- 4) Inet4Address - This class represents an internet protocol version 4(IPV4).
- 5) Inet6Address – This class represents an internet protocol version 6(IPV6)
- 6) URL – This class represents a uniform resource locator.
- 7) URLDecoder – This helps in HTML form decoding.
- 8) URLEncoder - This class helps in HTML form encoding .
- 9) SocketImpl – This is the superclass for all those classes which actually implement socket.
- 10)DatagramSocket – This class represents a socket for sending and recieving datagram packets

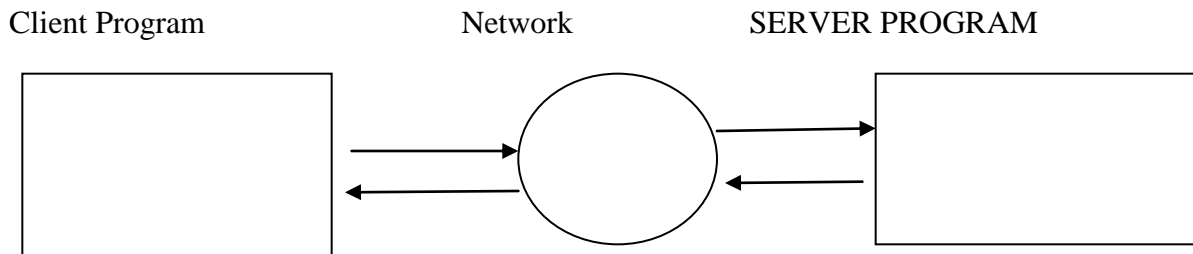
SOCKETS IN JAVA

The term Socket is often confused with the term port .Though both seem similar but there is a difference between the two.

The socket is strictly a software construct that is attached to a port on some host. Also, a port can have several sockets. Sockets and ports together enable devices to communicate with each other .Hence it is a sort of virtual communication device(not strictly).

Thus socket can also be defined as end point of a two way communication link, between two programs running on the network.

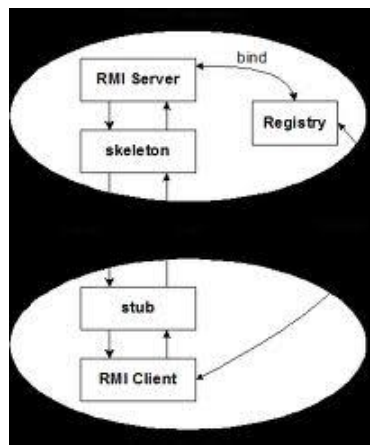
This can be understood with the help of diagram below:



JAVA RMI

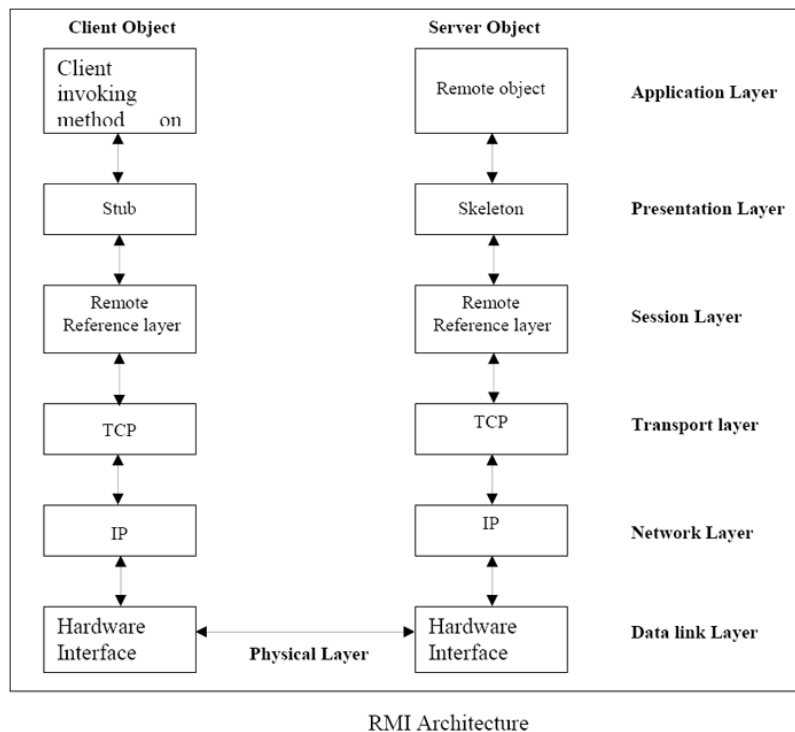
Its full form is Remote Method Invocation. It is an alternative to low level sockets. It is a form of RPC (Remote Protocol Call) .

It is a Java technology that allows us to create remote objects or some machine .Remote objects are the objects whose methods can be accessed from some other machine as if they are local objects of that machine. This technology is implemented with the help of rmi package included in Java development Kit .



GENERAL RMI ARCHITECTURE

RMI CLIENT SERVER ARCHITECTURE



RMI uses stub and skeleton object for communication with the remote object. A **remote object** is an object whose method can be invoked from another JVM. Let's understand the stub and skeleton objects:

STUB

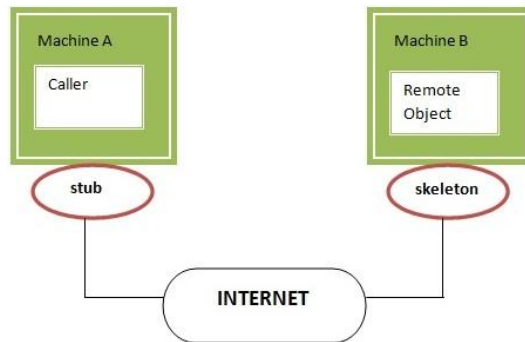
The stub is an object, acts as a gateway for the client side. All the outgoing requests are routed through it. It resides at the client side and represents the remote object. When the caller invokes method on the stub object, it does the following tasks:

1. It initiates a connection with remote Virtual Machine (JVM),
2. It writes and transmits (marshals) the parameters to the remote Virtual Machine (JVM),
3. It waits for the result
4. It reads (unmarshals) the return value or exception, and
5. It finally, returns the value to the caller.

SKELETON

The skeleton is an object, acts as a gateway for the server side object. All the incoming requests are routed through it. When the skeleton receives the incoming request, it does the following tasks:

1. It reads the parameter for the remote method
2. It invokes the method on the actual remote object, and
3. It writes and transmits (marshals) the result to the caller.

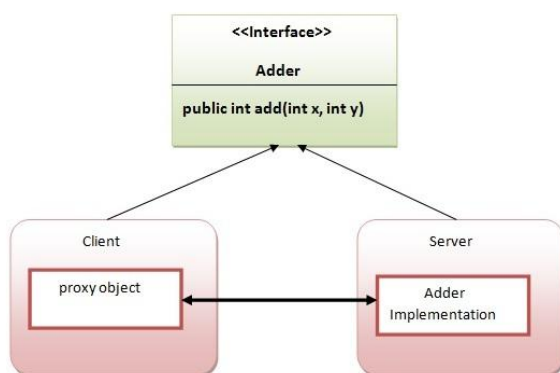


HOW TO CREATE A RMI APPLICATION

6 steps to write the RMI program.

1. Create the remote interface
2. Provide the implementation of the remote interface
3. Compile the implementation class and create the stub and skeleton objects using the rmic tool
4. Start the registry service by rmiregistry tool
5. Create and start the remote application
6. Create and start the client application

The client application need only two files, remote interface and client application. In the rmi application, both client and server interacts with the remote interface. The client application invokes methods on the proxy object, RMI sends the request to the remote JVM. The return value is sent back to the proxy object and then to the client application.



1) create the remote interface

For creating the remote interface, extend the Remote interface and declare the RemoteException with all the methods of the remote interface. Here, we are creating a remote interface that extends the Remote interface. There is only one method named add() and it declares RemoteException.

```
import java.rmi.*;
public interface Adder extends Remote{
public int add(int x,int y)throws RemoteException;
}
```

2) Provide the implementation of the remote interface

Now provide the implementation of the remote interface. For providing the implementation of the Remote interface, we need to

- Either extend the UnicastRemoteObject class,
- or use the exportObject() method of the UnicastRemoteObject class

In case, you extend the UnicastRemoteObject class, you must define a constructor that declares RemoteException.

```
import java.rmi.*;
import java.rmi.server.*;
public class AdderRemote extends UnicastRemoteObject implements Adder{
AdderRemote()throws RemoteException{
super();
}
public int add(int x,int y){return x+y;}
}
```

3) create the stub and skeleton objects using the rmic tool.

Next step is to create stub and skeleton objects using the rmi compiler. The rmic tool invokes the RMI compiler and creates stub and skeleton objects.

```
rmic AdderRemote
```

4) Start the registry service by the rmiregistry tool

Now start the registry service by using the rmiregistry tool. If you don't specify the port number, it uses a default port number. In this example, we are using the port number 5000.

```
rmiregistry 5000
```

5) Create and run the server application

Now rmi services need to be hosted in a server process. The Naming class provides methods to get and store the remote object. The Naming class provides 5 methods.

<pre> public static java.rmi.Remote lookup(java.lang.String) throws java.rmi.NotBoundException, java.net.MalformedURLException, java.rmi.RemoteException; </pre>	<p>It returns the reference of the remote object.</p>
<pre> public static void bind(java.lang.String, java.rmi.Remote) throws java.rmi.AlreadyBoundException, java.net.MalformedURLException, java.rmi.RemoteException; </pre>	<p>It binds the remote object with the given name.</p>
<pre> public static void unbind(java.lang.String) throws java.rmi.RemoteException, java.rmi.NotBoundException, java.net.MalformedURLException; </pre>	<p>It destroys the remote object which is bound with the given name.</p>
<pre> public static void rebind(java.lang.String, java.rmi.Remote) throws java.rmi.RemoteException, java.net.MalformedURLException; </pre>	<p>It binds the remote object to the new name.</p>
<pre> public static java.lang.String[] list(java.lang.String) throws java.rmi.RemoteException, java.net.MalformedURLException; </pre>	<p>It returns an array of the names of the remote objects bound in the registry.</p>

Example : we are binding the remote object by the name sonoo.

```

import java.rmi.*;
import java.rmi.registry.*;
public class MyServer{
public static void main(String args[]){
try{
Adder stub=new AdderRemote();
Naming.rebind("rmi://localhost:5000/sonoo",stub);
}catch(Exception e){System.out.println(e);}
}
}

```

6) Create and run the client application

At the client we are getting the stub object by the lookup() method of the Naming class and invoking the method on this object. In this example, we are running the server and client applications, in the same machine so we are using localhost. If you want to access the remote object from another machine, change the localhost to the host name (or IP address) where the remote object is located.

```
import java.rmi.*;
public class MyClient{
public static void main(String args[]){
try{
Adder stub=(Adder)Naming.lookup("rmi://localhost:5000/sonoo");
System.out.println(stub.add(34,4));
}catch(Exception e){ }
}
}
```

FOR RUNNING **THIS** RMI EXAMPLE,

1) compile all the java files

```
javac *.java
```

2)create stub and skeleton object by rmic tool

```
rmic AdderRemote
```

3)start rmi registry in one command prompt

```
rmiregistry 5000
```

4)start the server in another command prompt

```
java MyServer
```

5)start the client application in another command prompt

```
java MyClient
```