

# JAVA

## Introduction

Java programming language was originally developed by Sun Microsystems which was initiated by James Gosling and released in 1995 as core component of Sun Microsystems' Java platform (Java 1.0 [J2SE]).

Java is guaranteed to be **Write Once, Run Anywhere**.

Java is:

- **Object Oriented:** In Java, everything is an Object. Java can be easily extended since it is based on the Object model.
- **Platform independent:** Unlike many other programming languages including C and C++, when Java is compiled, it is not compiled into platform specific machine, rather into platform independent byte code. This byte code is distributed over the web and interpreted by virtual Machine (JVM) on whichever platform it is being run.
- **Simple:** Java is designed to be easy to learn. If you understand the basic concept of OOP Java would be easy to master.
- **Secure:** With Java's secure feature it enables to develop virus-free, tamper-free systems. Authentication techniques are based on public-key encryption.
- **Architectural-neutral:** Java compiler generates an architecture-neutral object file format which makes the compiled code to be executable on many processors, with the presence of Java runtime system.
- **Portable:** Being architectural-neutral and having no implementation dependent aspects of the specification makes Java portable. Compiler in Java is written in ANSI C with a clean portability boundary which is a POSIX subset.
- **Robust:** Java makes an effort to eliminate error prone situations by emphasizing mainly on compile time error checking and runtime checking.
- **Multithreaded:** With Java's multithreaded feature it is possible to write programs that can do many tasks simultaneously. This design feature allows developers to construct smoothly running interactive applications.
- **Interpreted:** Java byte code is translated on the fly to native machine instructions and is not stored anywhere. The development process is more rapid and analytical since the linking is an incremental and light weight process.
- **High Performance:** With the use of Just-In-Time compilers, Java enables high performance.
- **Distributed:** Java is designed for the distributed environment of the internet.
- **Dynamic:** Java is considered to be more dynamic than C or C++ since it is designed to adapt to an evolving environment. Java programs can carry extensive amount of run-time information that can be used to verify and resolve accesses to objects on run-time.

### Creating a simple Java Program

Hello World example:

```
class HelloWorld {  
public static void main (String args[]) {  
    System.out.println("Hello World! ");  
}  
}
```

This program has two main parts:

- All the program is enclosed in a class definition—here, a class calledHello World.
- The body of the program (here, just the one line) is contained in a method (function)called main(). In Java applications, as in a C or C++ program, main() is the first method(function) that is run when the program is executed.

### Compiling the above program :

- In Sun's JDK, the Java compiler is called javac.  
`javac HelloWorld.java`
- When the program compiles without errors, a file called HelloWorld.class is created,in the same directory as the source file. This is the Java bytecode file.
- Then run that bytecode file using the Java interpreter. In the JDK, the Java interpreteris called simply java.

```
java HelloWorld
```

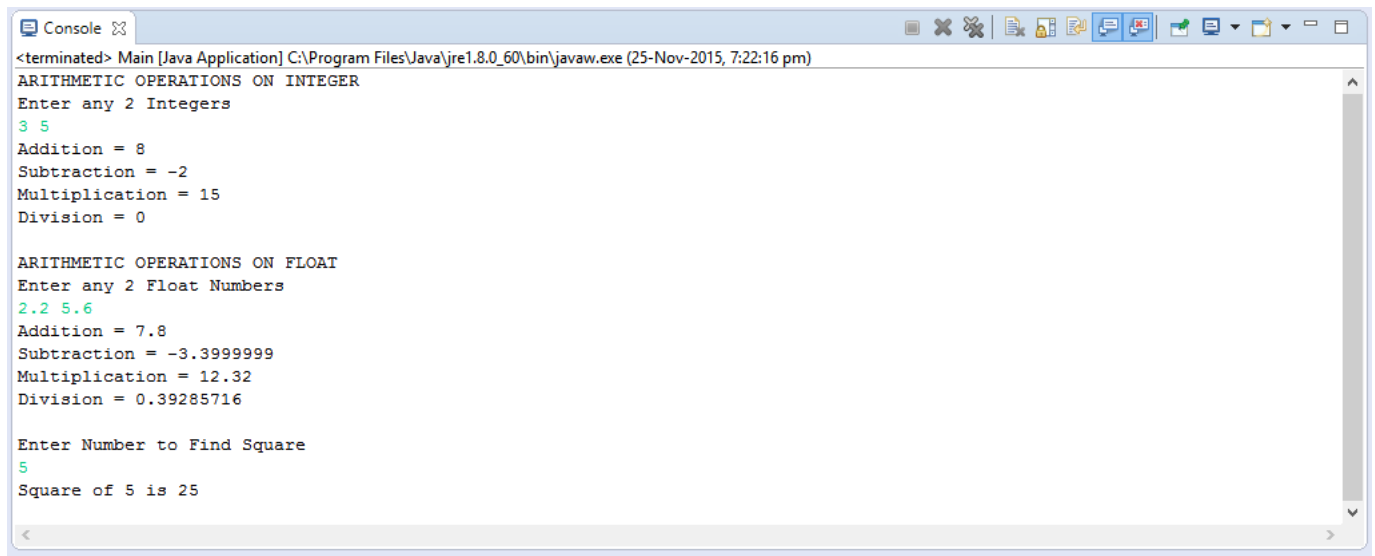
If the program was typed and compiled correctly, the output will be:

```
"Hello World!"
```

### 1a) Design and implement a JAVA Program to demonstrate Constructor Overloading and Method overloading.

```
import java.util.*;
class arithmetic
{
    int a,b;
    Scanner s1=new Scanner(System.in);
    arithmetic()
    {
        System.out.println("Enter any 2 Integers");
        a=s1.nextInt();
        b=s1.nextInt();
    }
    void display()
    {
        System.out.println("Addition = "+(a+b));
        System.out.println("Subtraction = "+(a-b));
        System.out.println("Multiplication = "+(a*b));
        System.out.println("Division = "+(a/b));
    }
    arithmetic(float a1, float b1)
    {
        System.out.println("Addition = "+(a1+b1));
        System.out.println("Subtraction = "+(a1-b1));
        System.out.println("Multiplication = "+(a1*b1));
        System.out.println("Division = "+(a1/b1));
    }
    void display(int x)
    {
        System.out.println("Square of "+x+" is "+(x*x));
    }
}
class Main
{
    public static void main(String args[])
    {
        Scanner s1=new Scanner(System.in);
        System.out.println("ARITHMETIC OPERATIONS ON INTEGER");
        arithmetic a=new arithmetic();
        a.display();
        System.out.println("\nARITHMETIC OPERATIONS ON FLOAT");
        System.out.println("Enter any 2 Float Numbers");
        float a1=s1.nextFloat();
        float a2=s1.nextFloat();
        arithmetic arth1=new arithmetic(a1,a2);
        System.out.println("\nEnter Number to Find Square");
        int x=s1.nextInt();
        a.display(x);
    }
}
```

## Output



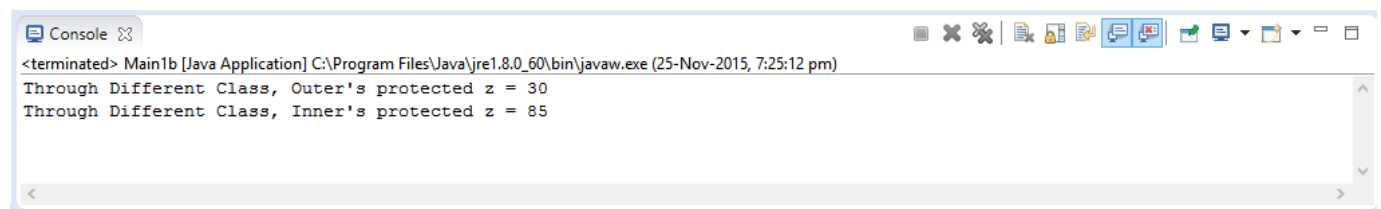
```
<terminated> Main [Java Application] C:\Program Files\Java\jre1.8.0_60\bin\javaw.exe (25-Nov-2015, 7:22:16 pm)
ARITHMETIC OPERATIONS ON INTEGER
Enter any 2 Integers
3 5
Addition = 8
Subtraction = -2
Multiplication = 15
Division = 0

ARITHMETIC OPERATIONS ON FLOAT
Enter any 2 Float Numbers
2.2 5.6
Addition = 7.8
Subtraction = -3.3999999
Multiplication = 12.32
Division = 0.39285716

Enter Number to Find Square
5
Square of 5 is 25
```

**1b) Write a JAVA program to implement Inner class and demonstrate its Access protections.**

```
class outer
{
    private int x=10;
    protected int z=30;
    class inner
    {
        private int x=20;
        protected int z=85;
    }
    public static void main(String args[])
    {
        outer obj1=new outer();
        inner obj2=new outer().new inner();
        System.out.println("Through Outer Class, x = "+obj1.x);
        System.out.println("Through Inner Class, x = "+obj2.x);
    }
}
class Main1b
{
    public static void main(String args[])
    {
        outer ob1=new outer();
        outer.inner ob2=new outer().new inner();
        System.out.println("Through Different Class, Outer's protected z = "+ob1.z);
        System.out.println("Through Different Class, Inner's protected z = "+ob2.z);
    }
}
```

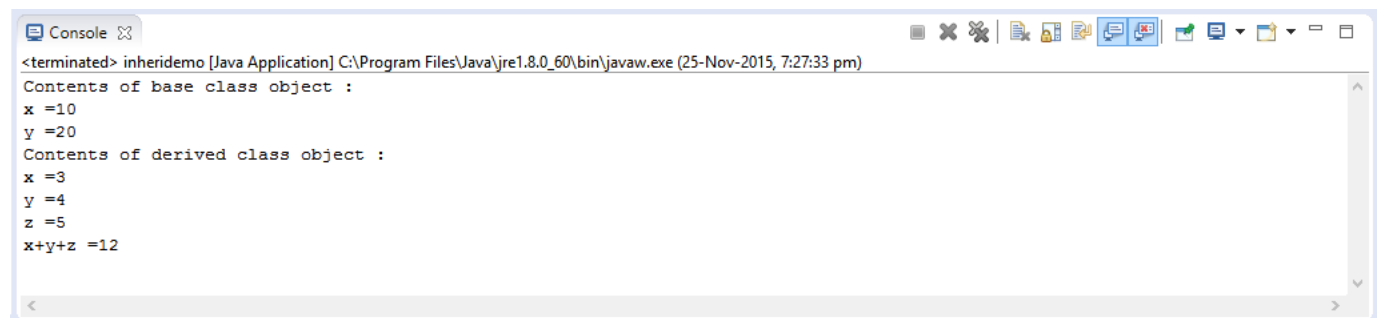
**Output**A screenshot of a Java console window titled "Console". The window shows the output of a Java application. The first line is "<terminated> Main1b [Java Application] C:\Program Files\Java\jre1.8.0\_60\bin\javaw.exe (25-Nov-2015, 7:25:12 pm)". The second line is "Through Different Class, Outer's protected z = 30". The third line is "Through Different Class, Inner's protected z = 85". The console window has a standard Windows-style title bar and a scroll bar on the right.

```
<terminated> Main1b [Java Application] C:\Program Files\Java\jre1.8.0_60\bin\javaw.exe (25-Nov-2015, 7:25:12 pm)
Through Different Class, Outer's protected z = 30
Through Different Class, Inner's protected z = 85
```

**2a) Write a JAVA program to demonstrate reusability using Inheritance.**

```
class A
{
    int x,y;
    void showxy()
    {
        System.out.println("x =" + x + "\ny =" + y );
    }
}
class B extends A {
    int z;

    void showz() {
        System.out.println("z =" + z);
        System.out.println("x+y+z =" + (x + y + z));
    }
}
class inheridemo
{
    public static void main(String a[])
    {
        A baseob=new A();
        B derob=new B();
        baseob.x=10;
        baseob.y=20;
        System.out.println("Contents of base class object :");
        baseob.showxy();
        derob.x=3;
        derob.y= 4;
        derob.z=5;
        System.out.println("Contents of derived class object :");
        derob.showxy();
        derob.showz();
    }
}
```

**Output**

The screenshot shows a Java console window titled "Console" with the following output:

```
<terminated> inheridemo [Java Application] C:\Program Files\Java\jre1.8.0_60\bin\javaw.exe (25-Nov-2015, 7:27:33 pm)
Contents of base class object :
x =10
y =20
Contents of derived class object :
x =3
y =4
z =5
x+y+z =12
```

**2b) Write a JAVA program to handle run-time errors using Exception Handling (Using Nested try catch and finally) mechanism.**

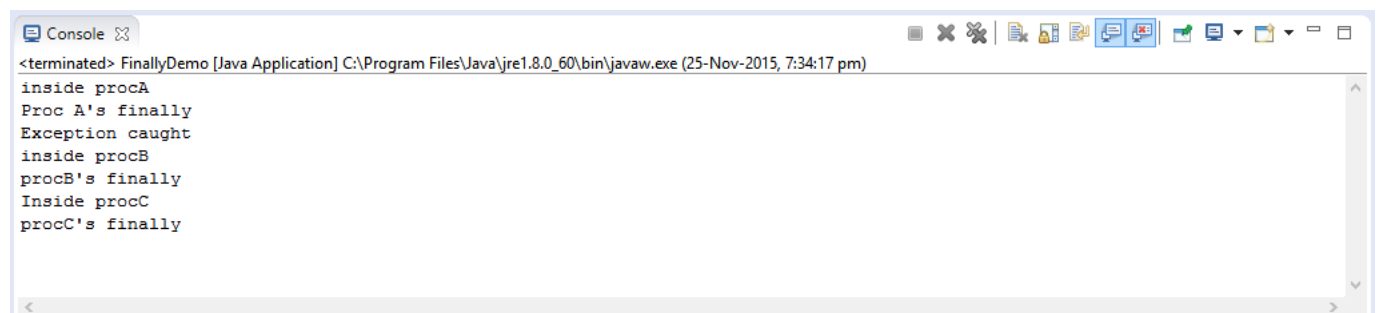
```
class FinallyDemo {
    //throw an exception out of the method
    static void procA() {
        try {
            System.out.println("inside procA");
            throw new RuntimeException("demo");
        } finally {
            System.out.println("Proc A's finally");
        }
    }

    // return from within a try block
    static void procB() {
        try {
            System.out.println("inside procB");
            return;
        } finally {
            System.out.println("procB's finally");
        }
    }

    //execute a try block normally
    static void procC() {
        try {
            System.out.println("Inside procC");
        } finally {
            System.out.println("procC's finally");
        }
    }

    public static void main(String args[]) {
        try {
            procA();
        } catch (Exception e) {
            System.out.println("Exception caught");
        }
        procB();
        procC();
    }
}
```

### Output



```
<terminated> FinallyDemo [Java Application] C:\Program Files\Java\jre1.8.0_60\bin\javaw.exe (25-Nov-2015, 7:34:17 pm)
inside procA
Proc A's finally
Exception caught
inside procB
procB's finally
Inside procC
procC's finally
```

**3a) Write a JAVA program to create five threads with different priorities. Send two threads of the highest priority to sleep state. Check the aliveness of the threads and mark which is long lasting.**

ThreadClass.java

```
class ThreadClass implements Runnable
{
    long click=0;
    Thread t;
    private volatile boolean running =true;
    public ThreadClass(int p)
    {
        t=new Thread(this);
        t.setPriority(p);
    }
    public void run()
    {
        while(running)
        {
            click++;
        }
    }

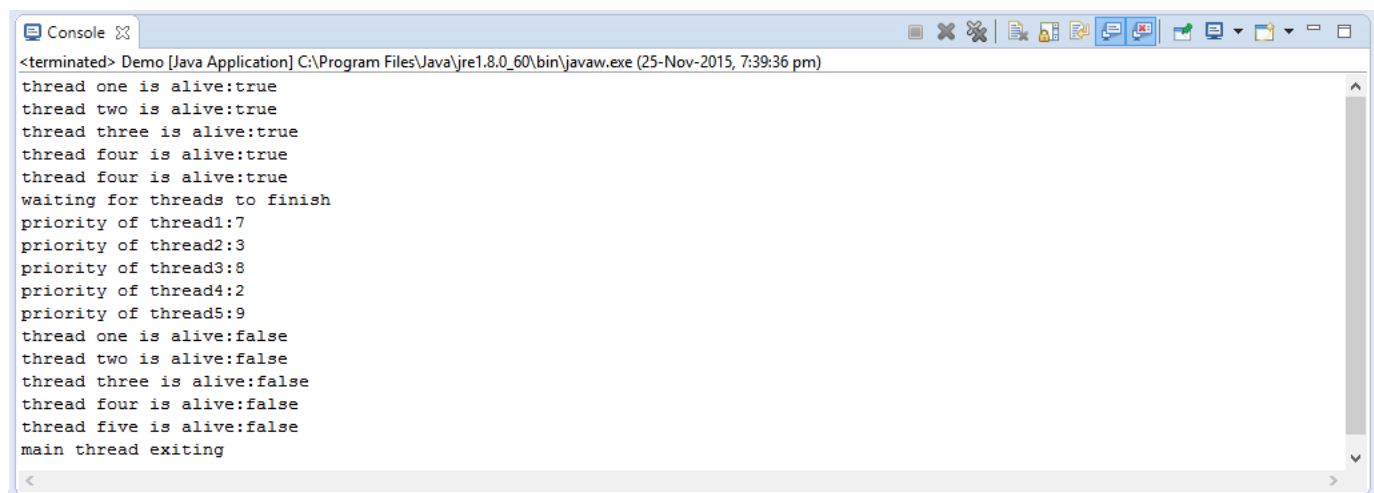
    public void stop()
    {
        running =false;
    }
    public void start()
    {
        t.start();
    }
}
```



Demo.java

```
public class Demo {
    public static void main(String args[])
    {
        Thread.currentThread().setPriority(Thread.MAX_PRIORITY);
        Thread hi1=new ThreadClass(Thread.NORM_PRIORITY + 2);
        Thread hi2=new ThreadClass(Thread.NORM_PRIORITY -2);
        Thread hi3=new ThreadClass(Thread.NORM_PRIORITY + 3);
        Thread hi4=new ThreadClass(Thread.NORM_PRIORITY - 3);
        Thread hi5=new ThreadClass(Thread.NORM_PRIORITY +4);
        hi1.start();
        hi2.start();
        hi3.start();
        hi4.start();
        hi5.start();
        System.out.println("thread one is alive:" +hi1.t.isAlive());
        System.out.println("thread two is alive:" +hi2.t.isAlive());
        System.out.println("thread three is alive:" +hi3.t.isAlive());
        System.out.println("thread four is alive:" +hi4.t.isAlive());
        System.out.println("thread four is alive:" +hi5.t.isAlive());
        try
        {
            hi5.t.sleep(1000);
            hi3.t.sleep(1000);
        }
        catch(InterruptedException e){
            System.out.println("main thread interrupted");
        }
        hi1.stop();
        hi2.stop();
        hi3.stop();
        hi4.stop();
        hi5.stop();
        try
        {
            System.out.println("waiting for threads to finish");
            hi1.t.join();
            hi2.t.join();
            hi3.t.join();
            hi4.t.join();
            hi5.t.join();
        }
        catch(InterruptedException e)
        {
            System.out.println("main thread interrupted");
        }
        System.out.println("priority of thread1:" +hi1.t.getPriority());
        System.out.println("priority of thread2:" +hi2.t.getPriority());
        System.out.println("priority of thread3:" +hi3.t.getPriority());
        System.out.println("priority of thread4:" +hi4.t.getPriority());
        System.out.println("priority of thread5:" +hi5.t.getPriority());
        System.out.println("thread one is alive:" +hi1.t.isAlive());
        System.out.println("thread two is alive:" +hi2.t.isAlive());
        System.out.println("thread three is alive:" +hi3.t.isAlive());
        System.out.println("thread four is alive:" +hi4.t.isAlive());
        System.out.println("thread five is alive:" +hi5.t.isAlive());
        System.out.println("main thread exiting");
    }
}
```

## Output



The screenshot shows a Java console window titled "Console" with a standard Windows-style title bar. The window contains the following text output from a Java application:

```
<terminated> Demo [Java Application] C:\Program Files\Java\jre1.8.0_60\bin\javaw.exe (25-Nov-2015, 7:39:36 pm)
thread one is alive:true
thread two is alive:true
thread three is alive:true
thread four is alive:true
thread four is alive:true
waiting for threads to finish
priority of thread1:7
priority of thread2:3
priority of thread3:8
priority of thread4:2
priority of thread5:9
thread one is alive:false
thread two is alive:false
thread three is alive:false
thread four is alive:false
thread five is alive:false
main thread exiting
```

The output indicates that five threads were created, their priorities were set, and then they were all terminated. The main thread then exits.

**3b) Write a Java program using synchronized threads which demonstrate producer-consumer concepts.**

```
class Q {
    int n;
    boolean valueset = false;

    synchronized int get() {
        while (!valueset)
            try {
                wait();
            } catch (InterruptedException e) {
                System.out.println("Thread Interrupted");
            }
        System.out.println("Got :" + n);
        valueset = false;
        notify();
        return n;
    }

    synchronized void put(int n) {
        while (valueset)
            try {
                wait();
            } catch (InterruptedException e) {
                System.out.println("Thread interrupted");
            }
        this.n = n;
        valueset = true;
        System.out.println("put " + n);
        notify();
    }
}

class Producer implements Runnable {
    Q q;

    Producer(Q q) {
        this.q = q;
        new Thread(this, "Producer").start();
    }

    public void run() {
        int i = 0;
        while (true) {
            q.put(i++);
        }
    }
}

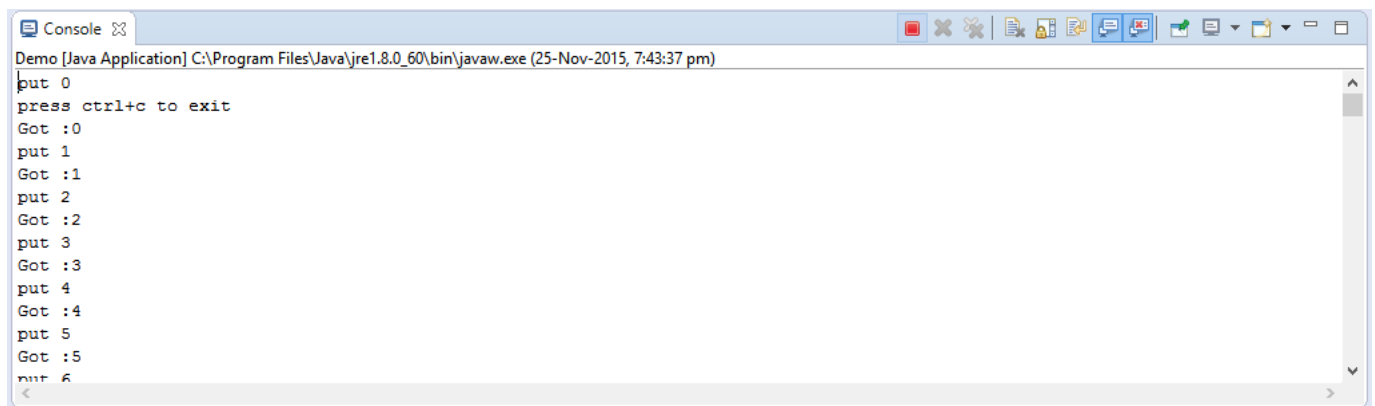
class Consumer implements Runnable {
    Q q;

    Consumer(Q q) {
        this.q = q;
        new Thread(this, "Consumer").start();
    }
}
```

```
        public void run() {
            int i = 0;
            while (true) {
                q.get();
            }
        }
    }

    class Demo {
        public static void main(String args[]) {
            Q q = new Q();
            new Producer(q);
            new Consumer(q);
            System.out.println("press ctrl+c to exit");
        }
    }
}
```

### Output



```
Console
Demo [Java Application] C:\Program Files\Java\jre1.8.0_60\bin\javaw.exe (25-Nov-2015, 7:43:37 pm)
put 0
press ctrl+c to exit
Got :0
put 1
Got :1
put 2
Got :2
put 3
Got :3
put 4
Got :4
put 5
Got :5
put 6
<
```

### 4a) Create an interface and implement it in a class in JAVA.

#### Callback.java

```
package callback;

public interface Callback {
    void Callback(int param);
}
```

#### Client.java

```
package callback;

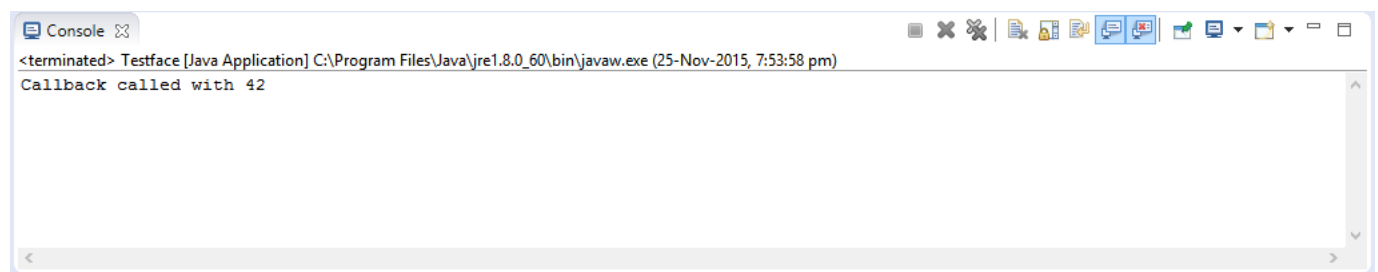
public class Client implements Callback{
    public void Callback(int param) {
        System.out.println("Callback called with "+param);
    }
}
```

#### Testface.java

```
package callback;

public class Testface {
    public static void main(String[] args) {
        Callback c = new Client();
        c.Callback(42);
    }
}
```

### Output



**4b) Write a program to make a package balance which has account class with display\_balance method in it. Import balance package in another program to access display\_balance method of account class.**

Account.java

```
package Balance;
import java.util.Scanner;

public class Account {
    int curBalance, amt;

    public Account() {
        curBalance = 500;
    }

    void deposit() {
        Scanner s = new Scanner(System.in);
        System.out.println("Enter the amount :");
        amt = s.nextInt();
        curBalance += amt;
        System.out.println("Current balance is :"+ curBalance);
    }

    void withdraw() {
        Scanner s = new Scanner(System.in);
        System.out.println("Enter the amount :");
        amt = s.nextInt();
        try {
            if ((curBalance - amt) < 500)
                throw new LessBalanceException(amt);
            curBalance -= amt;
            System.out.println("\nBalance left :"+ curBalance);
        } catch (LessBalanceException e) {
            System.out.println(e);
        }
    }

    void display_balance() {
        System.out.println("Balance in your a/c :"+ curBalance);
    }
}

class LessBalanceException extends Exception {
    int amt;

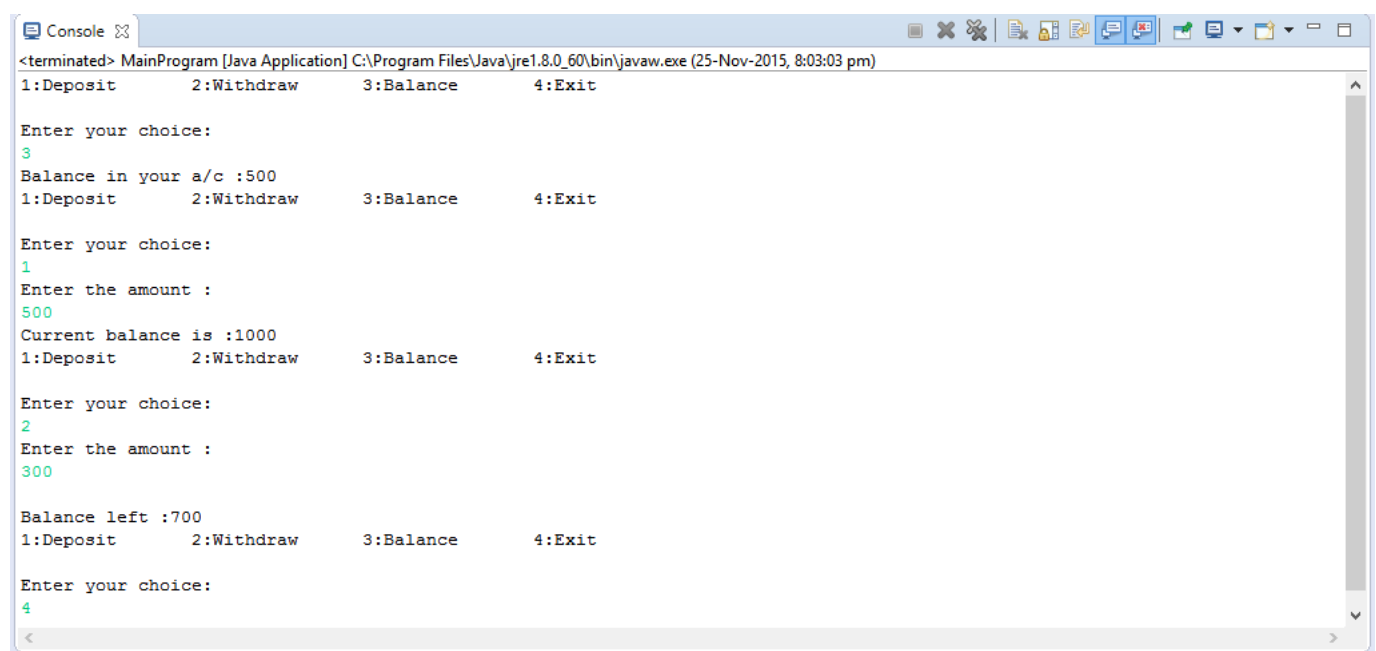
    LessBalanceException(int x) {
        System.out.println("Balance is less :"+ amt);
    }
}
```

## MainProgram.java

```
package Balance;
import java.util.Scanner;

public class MainProgram {
    public static void main(String[] args) {
        int ch;
        Scanner s = new Scanner(System.in);
        Account a = new Account();
        while (true) {
            System.out.println("1:Deposit\t2:Withdraw\t3:Balance\t4:Exit\n");
            System.out.println("Enter your choice:");
            ch = s.nextInt();
            switch (ch) {
                case 1:
                    a.deposit();
                    break;
                case 2:
                    a.withdraw();
                    break;
                case 3:
                    a.display_balance();
                    break;
                case 4:
                    return;
                default:
                    System.out.println("Invalid choice\n");
                    return;
            }
        }
    }
}
```

## Output



```
<terminated> MainProgram [Java Application] C:\Program Files\Java\jre1.8.0_60\bin\javaw.exe (25-Nov-2015, 8:03:03 pm)
1:Deposit      2:Withdraw      3:Balance      4:Exit

Enter your choice:
3
Balance in your a/c :500
1:Deposit      2:Withdraw      3:Balance      4:Exit

Enter your choice:
1
Enter the amount :
500
Current balance is :1000
1:Deposit      2:Withdraw      3:Balance      4:Exit

Enter your choice:
2
Enter the amount :
300
Balance left :700
1:Deposit      2:Withdraw      3:Balance      4:Exit

Enter your choice:
4
```

## JAVA APPLET

Applet is a special type of program that is embedded in the webpage to generate the dynamic content. It runs inside the browser and works at client side.

### **Features of Applets**

- An applet is a Java class that extends the *java.applet.Applet* class.
- A main() method is not invoked on an applet, and an applet class will not define main().
- Applets are designed to be embedded within an HTML page.
- When a user views an HTML page that contains an applet, the code for the applet is downloaded to the user's machine.
- A JVM is required to view an applet.
- The JVM on the user's machine creates an instance of the applet class and invokes various methods during the applet's lifetime.



### The Applet (java.applet.Applet) class

Every applet is an extension of the *java.applet.Applet* class. The base Applet class provides methods that a derived Applet class may call to obtain information and services from the browser context.

The Applet class provides an interface by which the viewer or browser obtains information about the applet and controls the applet's execution. The viewer may:

- request information about the author, version and copyright of the applet
- request a description of the parameters the applet recognizes
- initialize the applet
- destroy the applet
- start the applet's execution
- stop the applet's execution

The Applet class provides default implementations of each of these methods. Those implementations may be overridden as necessary.

### Life Cycle of an Applet

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

- **public void init():** This method is used to initialize the applet. It is invoked only once.
- **public void start():** This method is automatically called after the browser calls the **init()** method. It is also called whenever the user returns to the page containing the applet after having gone off to other pages. It is used to start the applet.
- **public void stop():** This method is used to stop the applet. It is automatically invoked when the applet is stopped or the browser is minimised.
- **public void destroy():** This method is used to destroy the applet. It is invoked only once.

The *java.awt.Component* class provides 1 life cycle method of applet.

- **public void paint(Graphics g):** Invoked immediately after the **start()** method, and also any time the applet needs to repaint itself in the browser. The **paint()** method is actually inherited from the *java.awt* class.

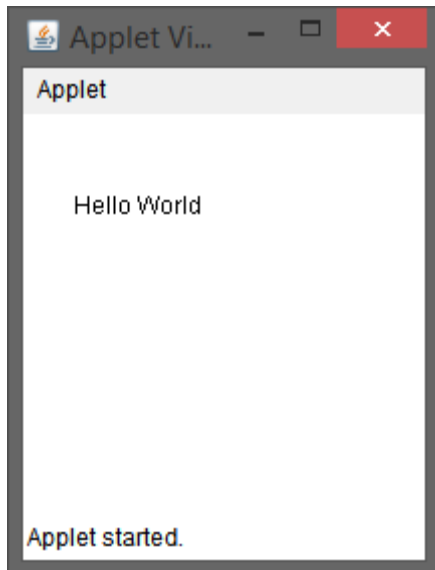
### A "Hello, World" Applet

The following is a simple applet named HelloWorldApplet.java:

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

public class HelloWorldApplet extends Applet
{
```

```
public void paint (Graphics g)
{
    g.drawString ("Hello World", 25, 50);
}
}
```



### **Getting Applet parameters**

The `Applet.getParameter()` method fetches a parameter from the applet tag given the parameter's name (the value of a parameter is always a string). If the value is numeric or other non-character data, the string must be parsed.

## **EVENT HANDLING**

### **The Delegation Event Model**

The modern approach to handling events is based on the *delegation event model*, which defines standard and consistent mechanisms to generate and process events. Its concept is quite simple: a *source* generates an event and sends it to one or more *listeners*. In this scheme, the listener simply waits until it receives 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 cleanly 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.

In the delegation event model, listeners must register with a source in order to receive an event notification. This provides an important benefit: notifications are sent only to listeners that want to receive them.

### **Events**

In the delegation model, an *event* is an object that describes a state change in a source. It can be generated as a consequence of a person interacting with the elements in a graphical user interface. Some of the activities that cause events to be generated are pressing a button, entering a character via the keyboard, selecting an item in a list, and clicking the mouse.

### Event Sources

A *source* is an object that generates an event. This occurs when the internal state of that object changes in some way. Sources may generate more than one type of event. A source must register listeners in order for the listeners to receive notifications about a specific type of event. Each type of event has its own registration method.

The general form is shown below:

```
public void addTypeListener(TypeListener el)
```

Here, *Type* is the name of the event, and *el* is a reference to the event listener. For example, the method that registers a keyboard event listener is called **addKeyListener( )**. The method that registers a mouse motion listener is called **addMouseMotionListener( )**. When an event occurs, all registered listeners are notified and receive a copy of the event object. This is known as *multicasting* the event. In all cases, notifications are sent only to listeners that register to receive them.

### Event Listener Interface

The delegation event model has two parts: sources and listeners. Listeners are created by implementing one or more of the interfaces defined by the **java.awt.event** package. When an event occurs, the event source invokes the appropriate method defined by the listener and provides an event object as its argument.

#### The KeyListener Interface

This interface defines three methods. The **keyPressed( )** and **keyReleased( )** methods are invoked when a key is pressed and released, respectively. The **keyTyped( )** method is invoked when a character has been entered.

The general forms of these methods are shown here:

```
void keyPressed(KeyEvent ke)  
void keyReleased(KeyEvent ke)  
void keyTyped(KeyEvent ke)
```

#### The MouseListener Interface

This interface defines five methods. If the mouse is pressed and released at the same point, **mouseClicked( )** is invoked. When the mouse enters a component, the **mouseEntered( )** method is called. When it leaves, **mouseExited( )** is called. The **mousePressed( )** and **mouseReleased( )** methods are invoked when the mouse is pressed and released, respectively.

The general forms of these methods are shown here:

```
void mouseClicked(MouseEvent me)  
void mouseEntered(MouseEvent me)  
void mouseExited(MouseEvent me)  
void mousePressed(MouseEvent me)  
void mouseReleased(MouseEvent me)
```

#### The MouseMotionListener Interface

This interface defines two methods. The **mouseDragged( )** method is called multiple times as the mouse is dragged. The **mouseMoved( )** method is called multiple times as the mouse is moved.

Their general forms are shown here:

**void mouseDragged(MouseEvent *me*)**

**void mouseMoved(MouseEvent *me*)**

**5a) Write JAVA Applet program which handles Mouse Event.**

```
/*
<applet code = "MouseEvents" width = 300 height = 300>
</applet>
*/

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

```
public class MouseEvents extends Applet implements MouseListener,
MouseMotionListener {
    int mousex = 0, mousey = 0;
    String msg = "";

    public void init() {
        addMouseListener(this);
        addMouseMotionListener(this);
    }

    public void mouseClicked(MouseEvent e) {
        mousex = 0;
        mousey = 10;
        msg = "Mouse Clicked";
        repaint();
    }

    public void mousePressed(MouseEvent e) {
        mousex = e.getX();
        mousey = e.getY();
        msg = "Mouse Pressed";
        repaint();
    }

    public void mouseMoved(MouseEvent e) {
        showStatus("Mouse moved at :" + e.getX() + "," + e.getY());
    }

    public void mouseReleased(MouseEvent e) {
        mousex = e.getX();
        mousey = e.getY();
        msg = "Mouse Released";
        repaint();
    }

    public void mouseEntered(MouseEvent e) {
        mousex = 0;
        mousey = 10;
        msg = "Mouse Entered";
        repaint();
    }

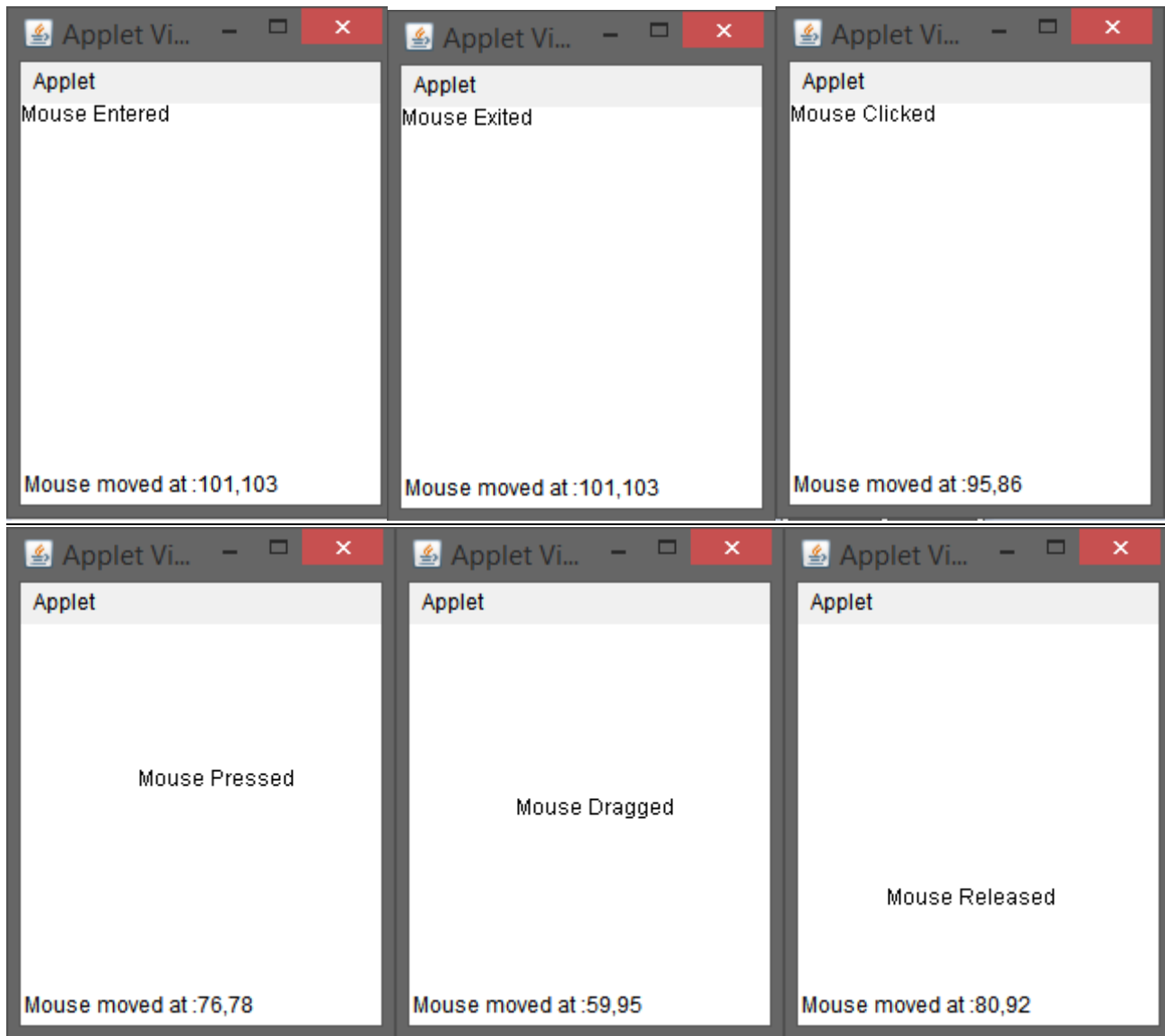
    public void mouseDragged(MouseEvent e) {
        mousex = e.getX();
        mousey = e.getY();
        msg = "Mouse Dragged";
        repaint();
    }

    public void mouseExited(MouseEvent e) {
        mousex = 0;
        mousey = 10;
        msg = "Mouse Exited";
        repaint();
    }

    public void paint(Graphics g) {
        g.drawString(msg, mousex, mousey);
    }
}
```

```
}  
}
```

## Output

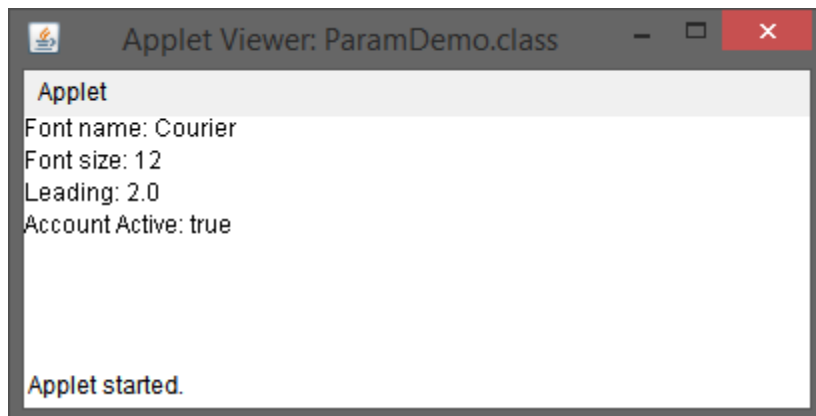


**5b) Write JAVA Applet program to Pass parameters and display the same.**

```
import java.awt.*;  
import java.applet.*;  
/*  
<applet code="ParamDemo" width=300 height=80>  
<param name=fontName value=Courier>  
<param name=fontSize value=14>  
<param name=leading value=2>  
<param name=accountEnabled value=true>
```

```
</applet>
*/
public class ParamDemo extends Applet{
    String fontName;
    int fontSize;
    float leading;
    boolean active;
    // Initialize the string to be displayed.
    public void start() {
        String param;
        fontName = getParameter("fontName");
        if(fontName == null)
            fontName = "Not 222";
        param = getParameter("fontSize");
        try {
            if(param != null) // if not found
                fontSize = Integer.parseInt(param);
            else
                fontSize = 0;
        } catch(NumberFormatException e) {
            fontSize = -1;
        }
        param = getParameter("leading");
        try {
            if(param != null) // if not found
                leading = Float.valueOf(param).floatValue();
            else
                leading = 0;
        } catch(NumberFormatException e) {
            leading = -1;
        }
        param = getParameter("accountEnabled");
        if(param != null)
            active = Boolean.valueOf(param).booleanValue();
    }
    // Display parameters.
    public void paint(Graphics g) {
        g.drawString("Font name: " + fontName, 0, 10);
        g.drawString("Font size: " + fontSize, 0, 26);
        g.drawString("Leading: " + leading, 0, 42);
        g.drawString("Account Active: " + active, 0, 58);
    }
}
```

### Output



## JAVA SWING

**Swing API** is set of extensible GUI Components to ease developer's life to create JAVA based Front End/ GUI Applications. It is built upon top of **AWT API** and acts as replacement of AWT API as it has almost



every control corresponding to AWT controls. It is a part of **Java Foundation Classes (JFC)** that is *used to create window-based applications*

### **MVC Architecture**

Swing API architecture follows loosely based MVC architecture in the following manner.

- A Model represents component's data.
- View represents visual representation of the component's data.
- Controller takes the input from the user on the view and reflects the changes in Component's data.
- Swing component have Model as a separate element and View and Controller part are clubbed in User Interface elements. Using this way, Swing has pluggable look-and-feel architecture.

### **Swing features**

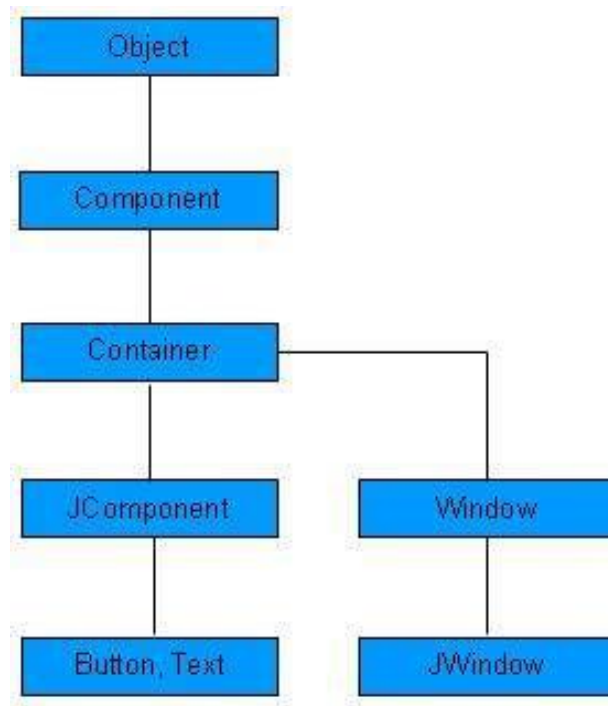
- **Light Weight** - Swing component are independent of native Operating System's API as Swing API controls are rendered mostly using pure JAVA code instead of underlying operating system calls.
- **Rich controls** - Swing provides a rich set of advanced controls like Tree, TabbedPane, slider, colorpicker, table controls
- **Highly Customizable** - Swing controls can be customized in very easy way as visual appearance is independent of internal representation.
- **Pluggable look-and-feel**- SWING based GUI Application look and feel can be changed at run time based on available values.

Every user interface considers the following three main aspects:

- **UI elements**:These are the core visual elements the user eventually sees and interacts with.
- **Layouts**: They define how UI elements should be organized on the screen and provide a final look and feel to the GUI (Graphical User Interface). This part will be covered in Layout chapter.
- **Behaviour**: These are events which occur when the user interacts with UI elements.

Every SWING controls inherits properties from following Component class hierarchy.

- **Component**:A Container is the abstract base class for the non-menu user-interface controls of SWING. Component represents an object with graphical representation.
- **Container**:A Container is a component that can contain other SWING components.
- **JComponent**:A JComponent is a base class for all swing UI components. In order to use a swing component that inherits from JComponent, component must be in a containment hierarchy whose root is a top-level Swing container.



### **SWING UI Elements:**

Following is the list of commonly used controls while designed GUI using SWING.

SL. No.	Control & Description
1	<b>JLabel:</b> A JLabel object is a component for placing text in a container.
2	<b>JButton:</b> This class creates a labelled button.
4	<b>JCheck Box:</b> A JCheckBox is a graphical component that can be in either an on (true) or off (false) state.
5	<b>JRadioButton:</b> The JRadioButton class is a graphical component that can be in either an on (true) or off (false) state. in a group.
6	<b>JList:</b> A JList component presents the user with a scrolling list of text items.
7	<b>JComboBox:</b> A JComboBox component presents the user with a show up menu of choices.
8	<b>JTextField:</b> A JTextField object is a text component that allows for the editing of a single line of text.
9	<b>JPasswordField:</b> A JPasswordField object is a text component specialized for password entry.
10	<b>JTextArea:</b> A JTextArea object is a text component that allows for the editing of a multiple lines of text.

**6. Write a Swing application which uses****a) JTabbed Pane****b) Each tab should JPanel which include any one component given below in each JPanel****c) ComboBox/List/Tree/RadioButton**

```
import javax.swing.*;

/*
<applet code="JTabbedPaneDemo" width=400 height=100>
</applet>
*/
public class JTabbedPaneDemo extends JApplet {
    public void init() {
        try {
            SwingUtilities.invokeLater(
                new Runnable() {
                    public void run() {
                        makeGUI();
                    }
                }
            );
        } catch (Exception exc) {
            System.out.println("Can't create because of " + exc);
        }
    }

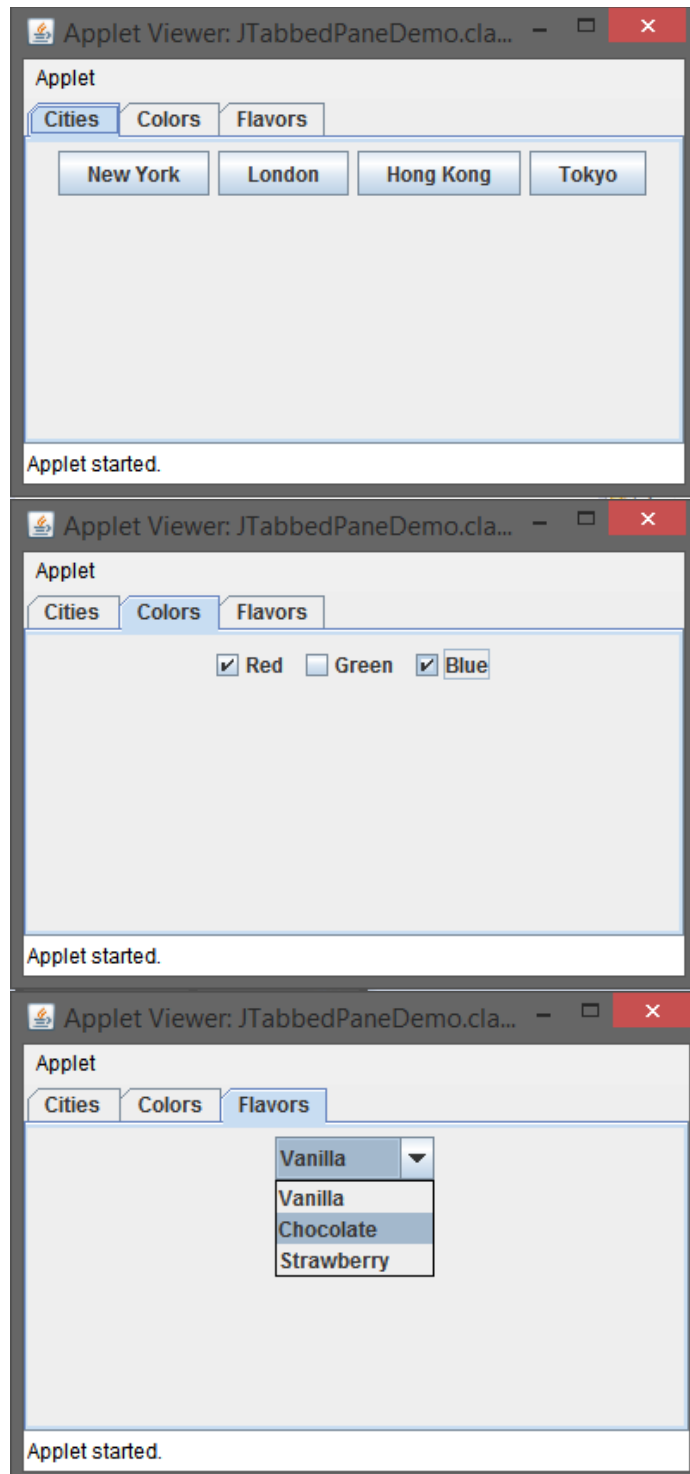
    private void makeGUI() {
        JTabbedPane jtp = new JTabbedPane();
        jtp.addTab("Cities", new CitiesPanel());
        jtp.addTab("Colors", new ColorsPanel());
        jtp.addTab("Flavors", new FlavorsPanel());
        add(jtp);
    }
}

// Make the panels that will be added to the tabbed pane.
class CitiesPanel extends JPanel {
    public CitiesPanel() {
        JButton b1 = new JButton("New York");
        add(b1);
        JButton b2 = new JButton("London");
        add(b2);
        JButton b3 = new JButton("Hong Kong");
        add(b3);
        JButton b4 = new JButton("Tokyo");
        add(b4);
    }
}

class ColorsPanel extends JPanel {
    public ColorsPanel() {
        JCheckBox cb1 = new JCheckBox("Red");
        add(cb1);
        JCheckBox cb2 = new JCheckBox("Green");
        add(cb2);
        JCheckBox cb3 = new JCheckBox("Blue");
        add(cb3);
    }
}
```

```
class FlavorsPanel extends JPanel {  
    public FlavorsPanel() {  
        JComboBox jcb = new JComboBox();  
        jcb.addItem("Vanilla");  
        jcb.addItem("Chocolate");  
        jcb.addItem("Strawberry");  
        add(jcb);  
    }  
}
```

### Output



### SERVLET PROGRAMMING

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

They are modules of Java code that run in a server application.

The advantages of using Servlets over traditional CGI programs are:

1. **Better performance:** because it creates a thread for each request not process.
2. **Portability:** because it uses java language.
3. **Robust:** Servlets are managed by JVM so no need to worry about memory leak, garbage collection etc.
4. **Secure:** because it uses java language.

#### Life cycle of a servlet

The life cycle of a servlet is controlled by the container in which the servlet has been deployed.

When a request is mapped to a servlet, the container performs the following steps:

1. If an instance of the servlet does not exist, the web container:
  - a. Loads the servlet class
  - b. Creates an instance of the servlet class
  - c. Initializes the servlet instance by calling the init method. Initialization is covered in Initializing a Servlet
2. Invokes the service method, passing a request and response object.
3. If the container needs to remove the servlet, it finalizes the servlet by calling the servlet's destroy method.

#### Servlet API

- The **javax.servlet** and **javax.servlet.http** packages represent interfaces and classes for servlet api.
- The **javax.servlet** package contains many interfaces and classes that are used by the servlet or web container. These are not specific to any protocol.
- The **javax.servlet.http** package contains interfaces and classes that are responsible for http requests only

#### javax.servlet package

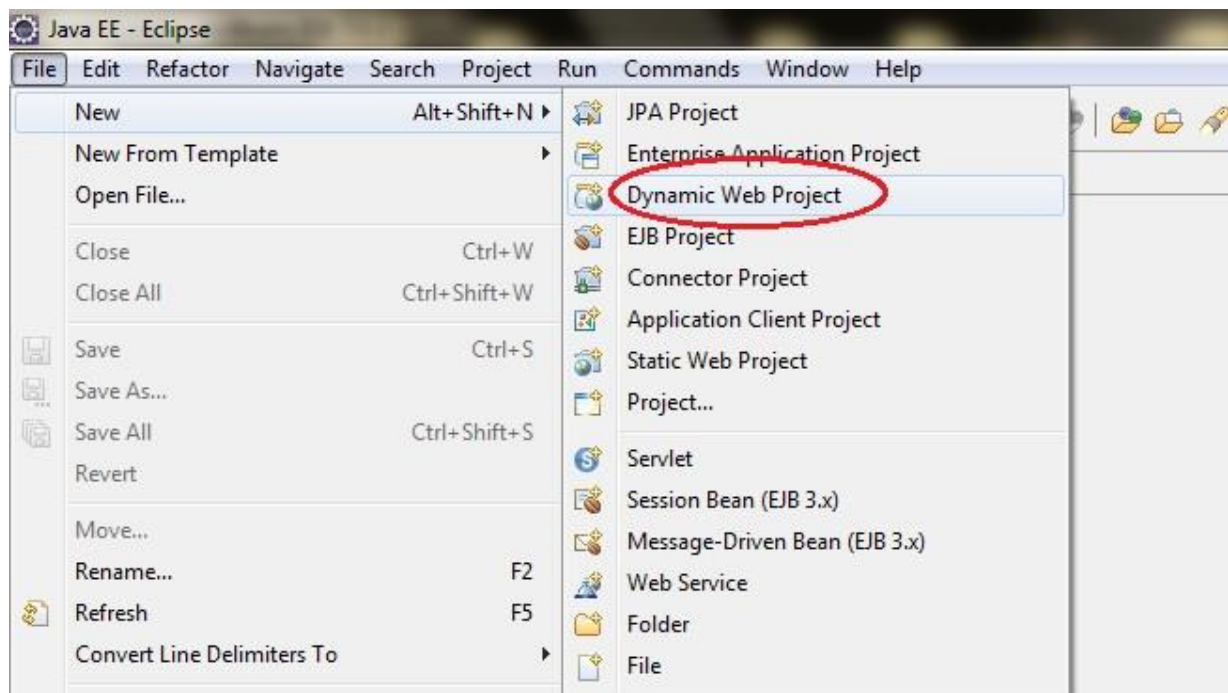
- The *javax.servlet package* contains a number of classes and interfaces that describe and define the contracts between a servlet class and the runtime environment provided for an instance of such a class by a conforming servlet container.
- The *Servlet* interface is the central abstraction of the servlet API.
- All servlets implement this interface either directly, or more commonly, by extending a class that implements the interface.
- The two classes in the servlet API that implement the *Servlet* interface are *GenericServlet* and *HttpServlet*.
- For most purposes, developers will extend *HttpServlet* to implement their servlets while implementing web applications employing the HTTP protocol.

- The basic *Servlet* interface defines a *service* method for handling client requests. This method is called for each request that the servlet container routes to an instance of a servlet.

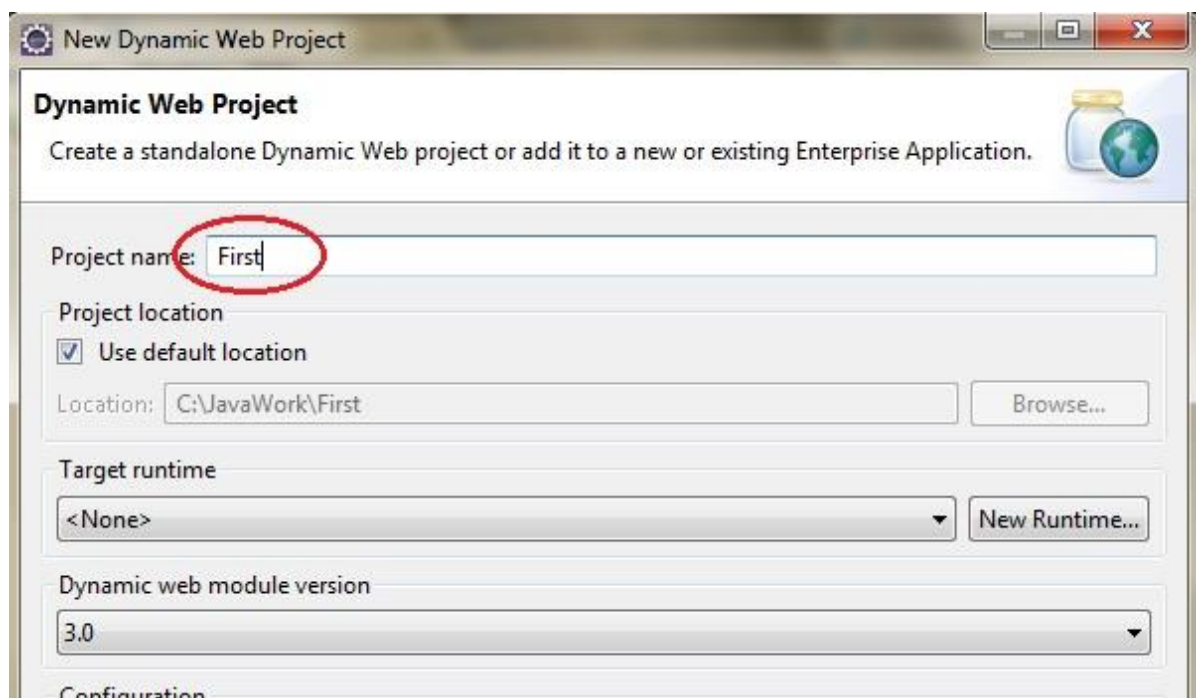
### Running Servlet Programs in Eclipse EE

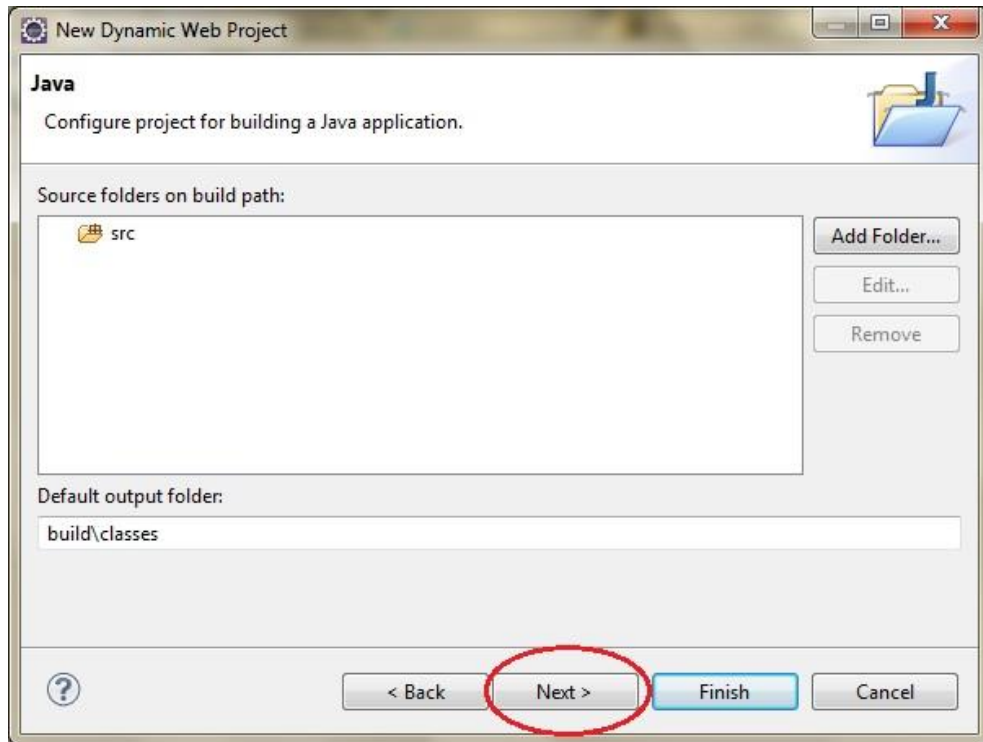
To create a Servlet application in Eclipse IDE you will need to follow the following steps:

Step 1. Goto **File ->New ->Dynamic Web Project**

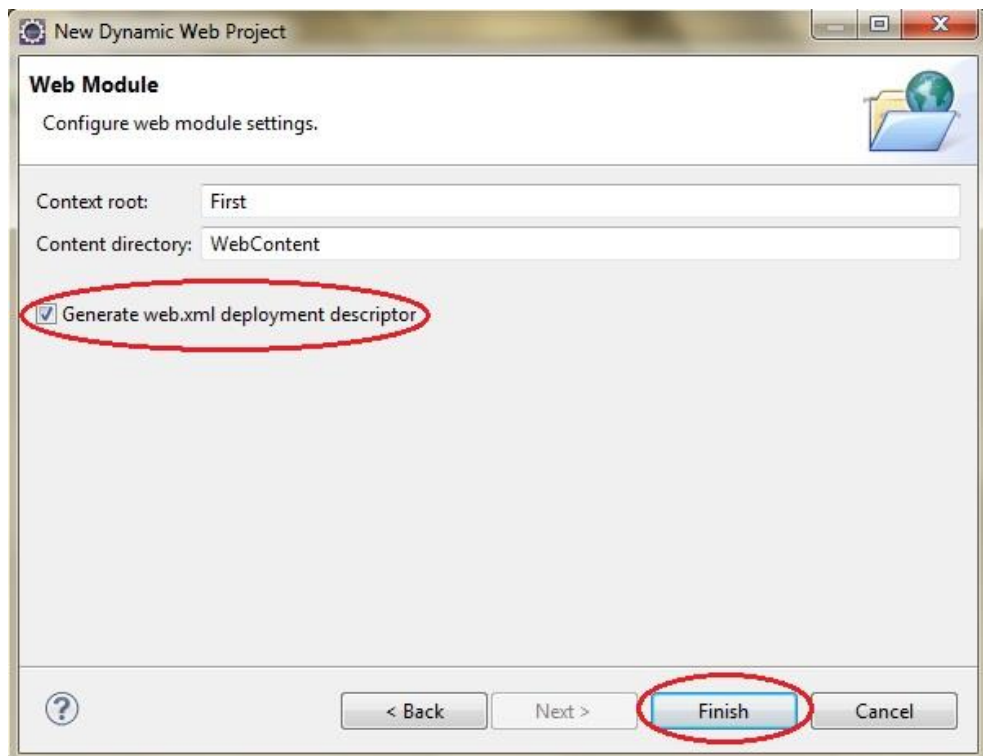


Step 2. Give a Name to your Project and click **Next**



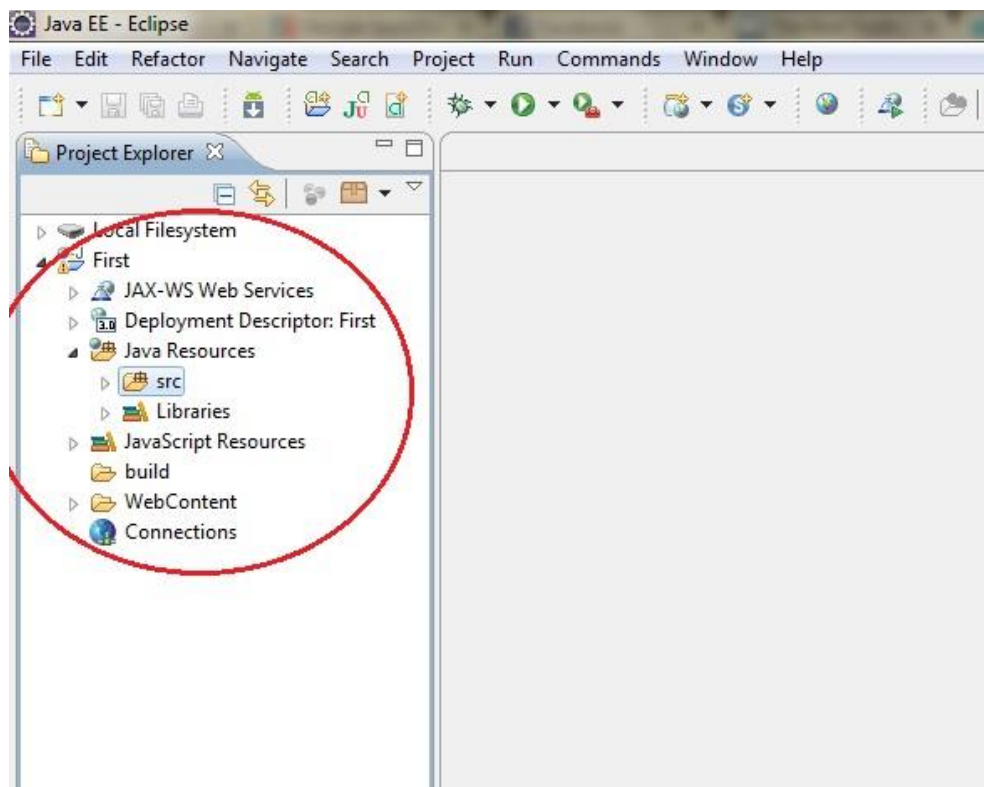


Step 3. Check **Generate web.xml Deployment Descriptor** and click **Finish**

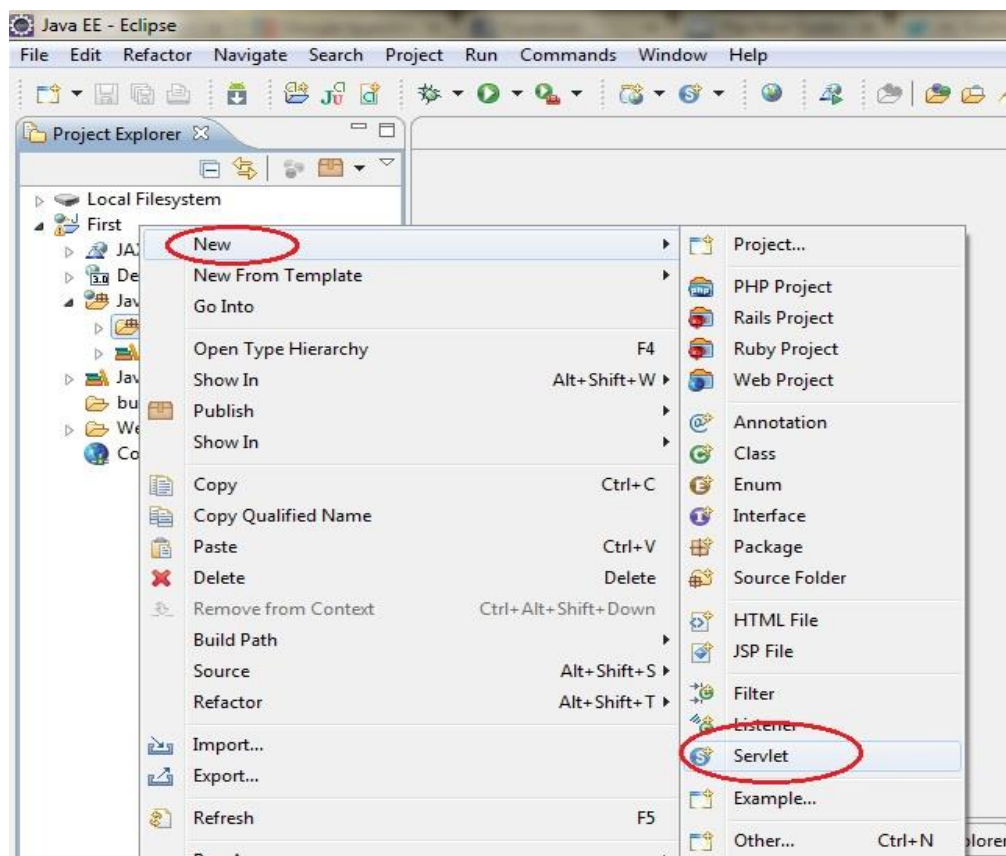




Step 4. Now, the complete directory structure of your Project will be automatically created by Eclipse IDE.

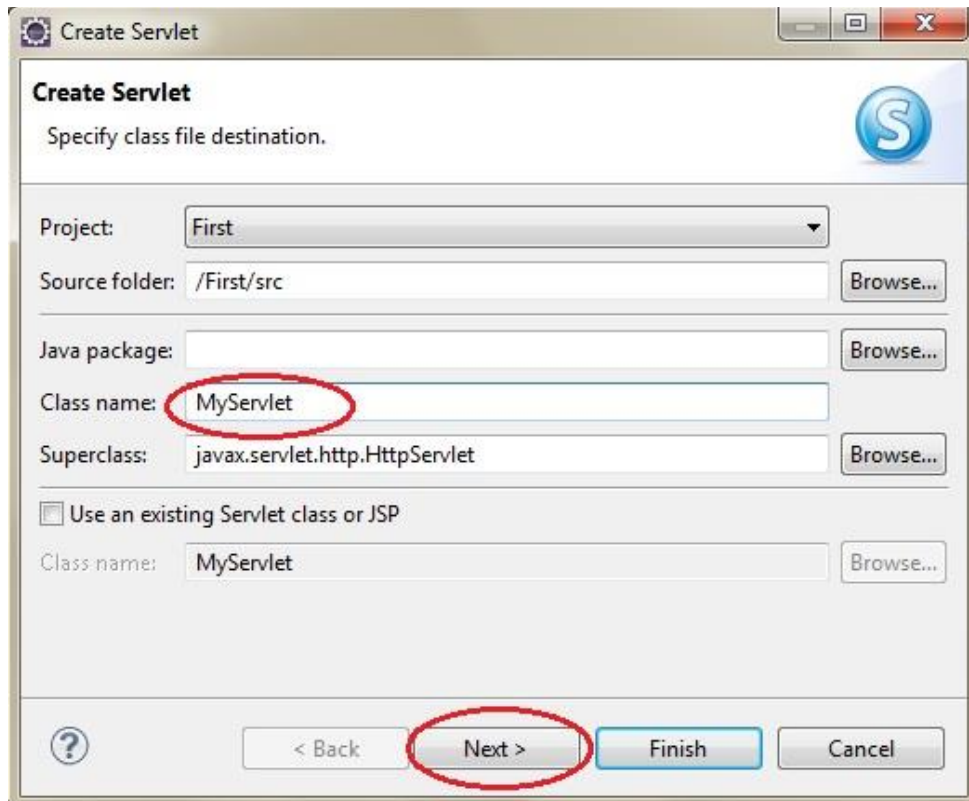


Step 5. Click on **First** project, go to **Java Resources** ->**src**. Right click on **src** select **New** ->**Servlet**





Step 6. Give Servlet class name and click **Next**



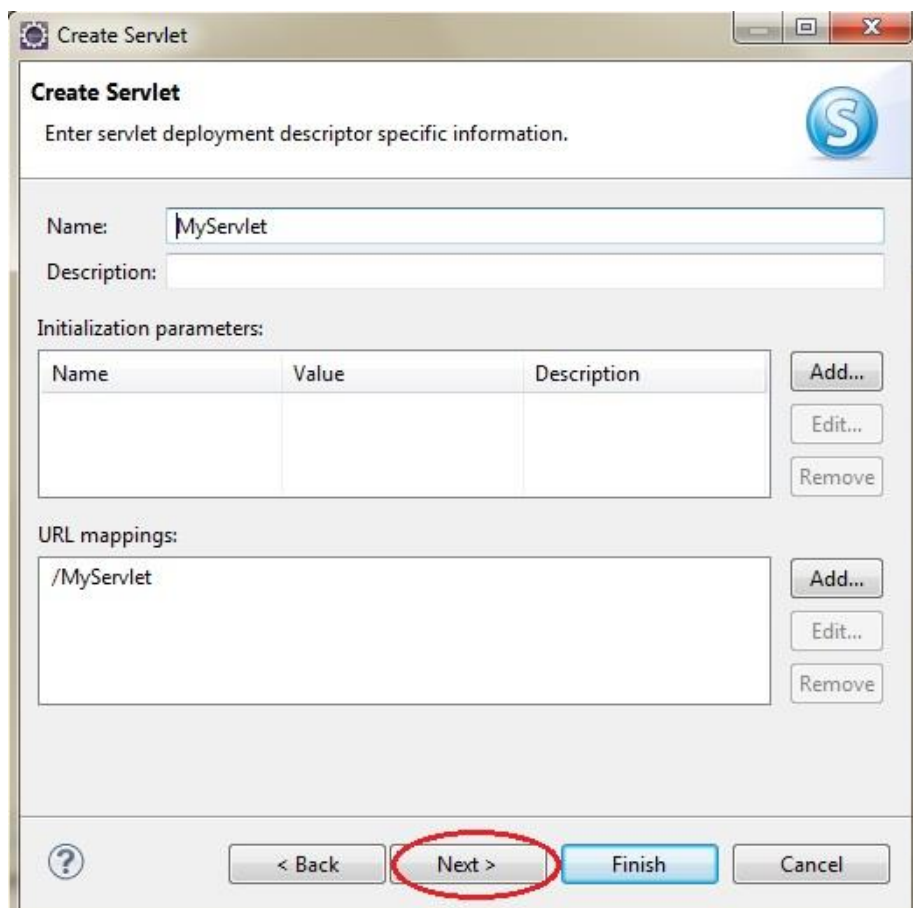
The 'Create Servlet' dialog box is shown. The 'Class name' field is highlighted with a red circle and contains the text 'MyServlet'. The 'Next >' button at the bottom is also highlighted with a red circle. Other fields include 'Project' (First), 'Source folder' (/First/src), 'Java package' (empty), 'Superclass' (javax.servlet.http.HttpServlet), and a checkbox for 'Use an existing Servlet class or JSP'.

**Create Servlet**  
Specify class file destination.

Project: First  
Source folder: /First/src  
Java package:  
Class name: MyServlet  
Superclass: javax.servlet.http.HttpServlet  
☐ Use an existing Servlet class or JSP  
Class name: MyServlet

< Back **Next >** Finish Cancel

Step 7. Give your Servlet class a Name of your choice.



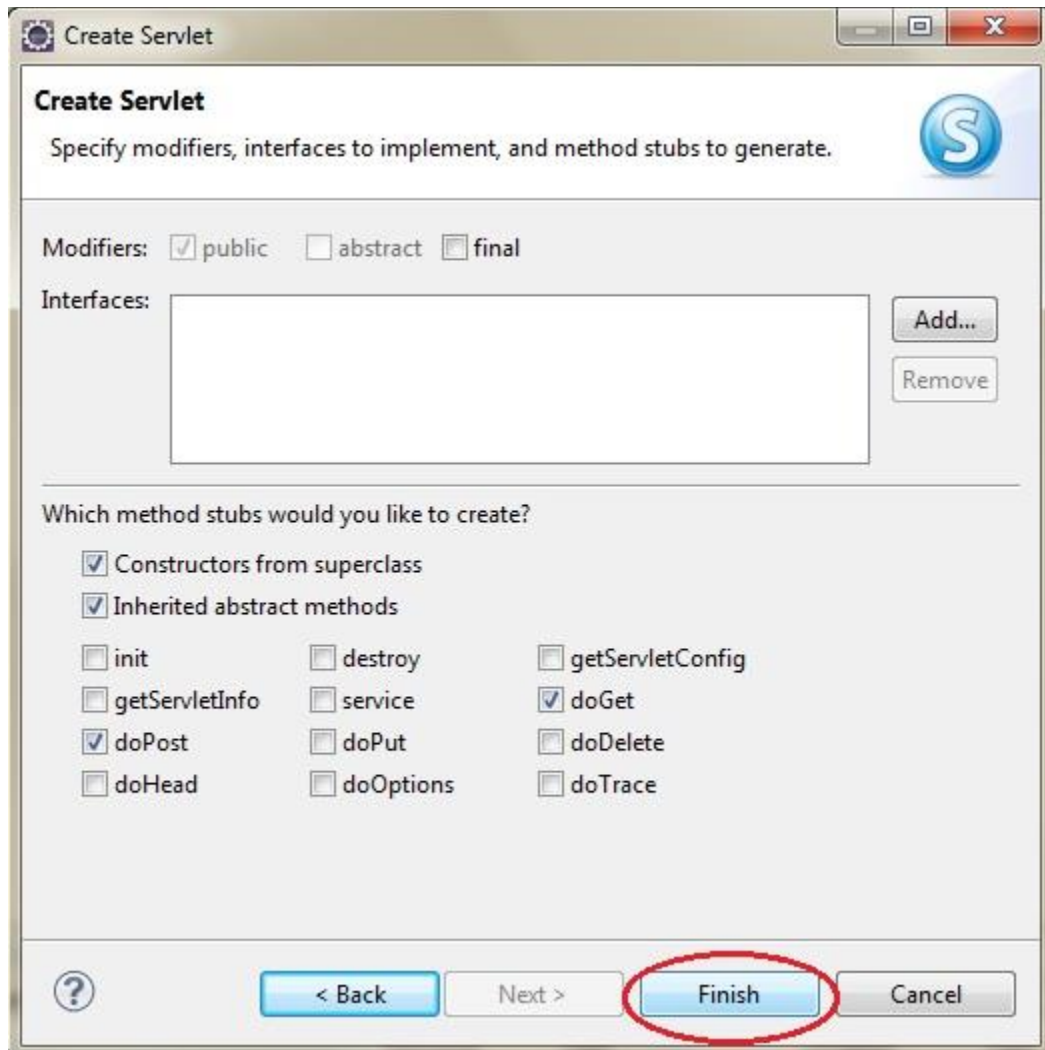
The 'Create Servlet' dialog box is shown. The 'Name' field is highlighted with a red circle and contains the text 'MyServlet'. The 'Next >' button at the bottom is also highlighted with a red circle. Other fields include 'Description' (empty), 'Initialization parameters' (empty table), and 'URL mappings' (containing /MyServlet).

**Create Servlet**  
Enter servlet deployment descriptor specific information.

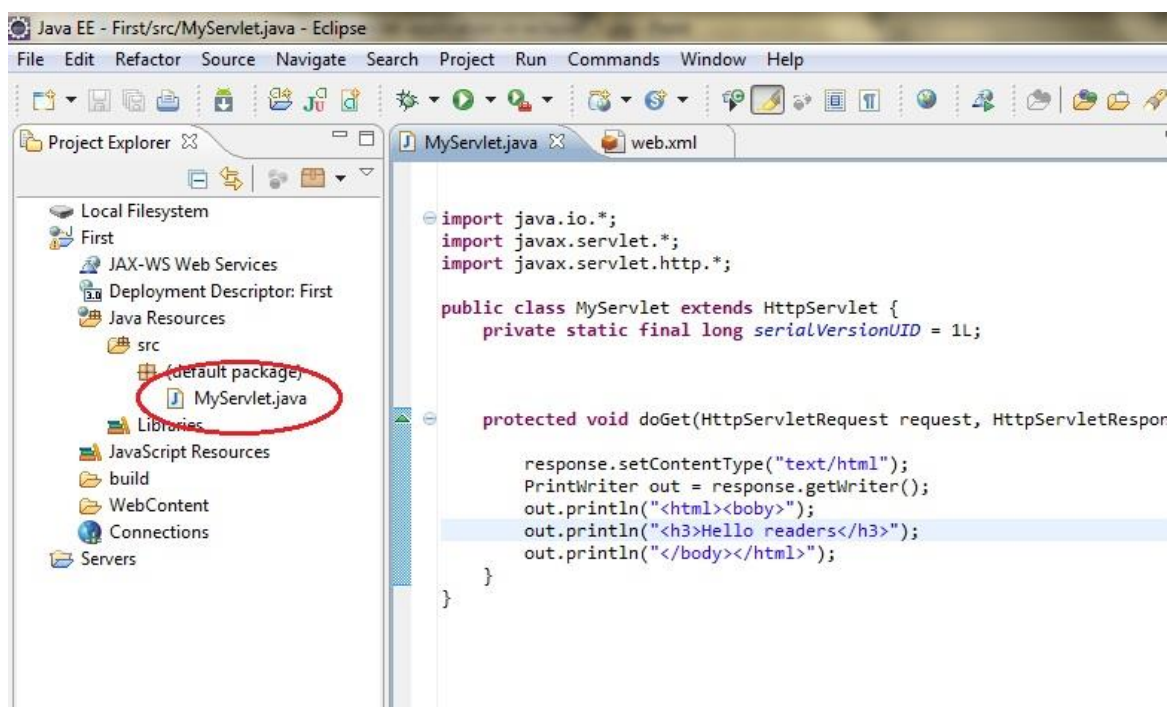
Name: MyServlet  
Description:  
Initialization parameters:  
URL mappings: /MyServlet

< Back **Next >** Finish Cancel

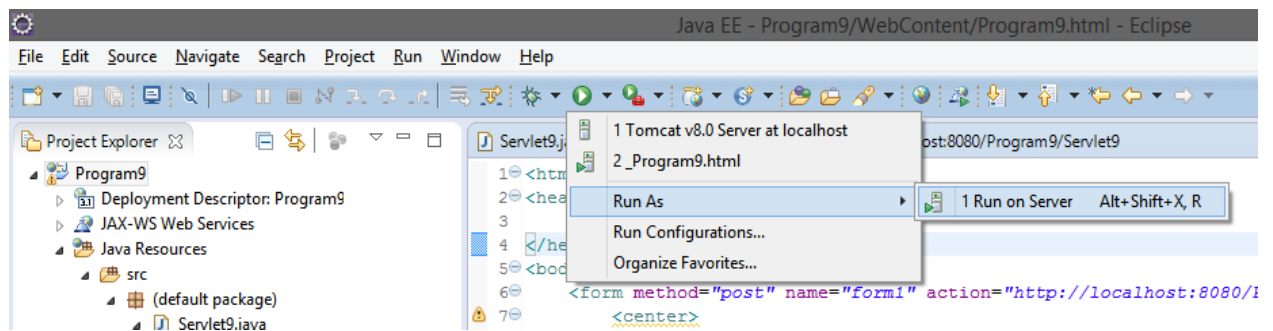
Step 8. Leave everything else to default and click **Finish**



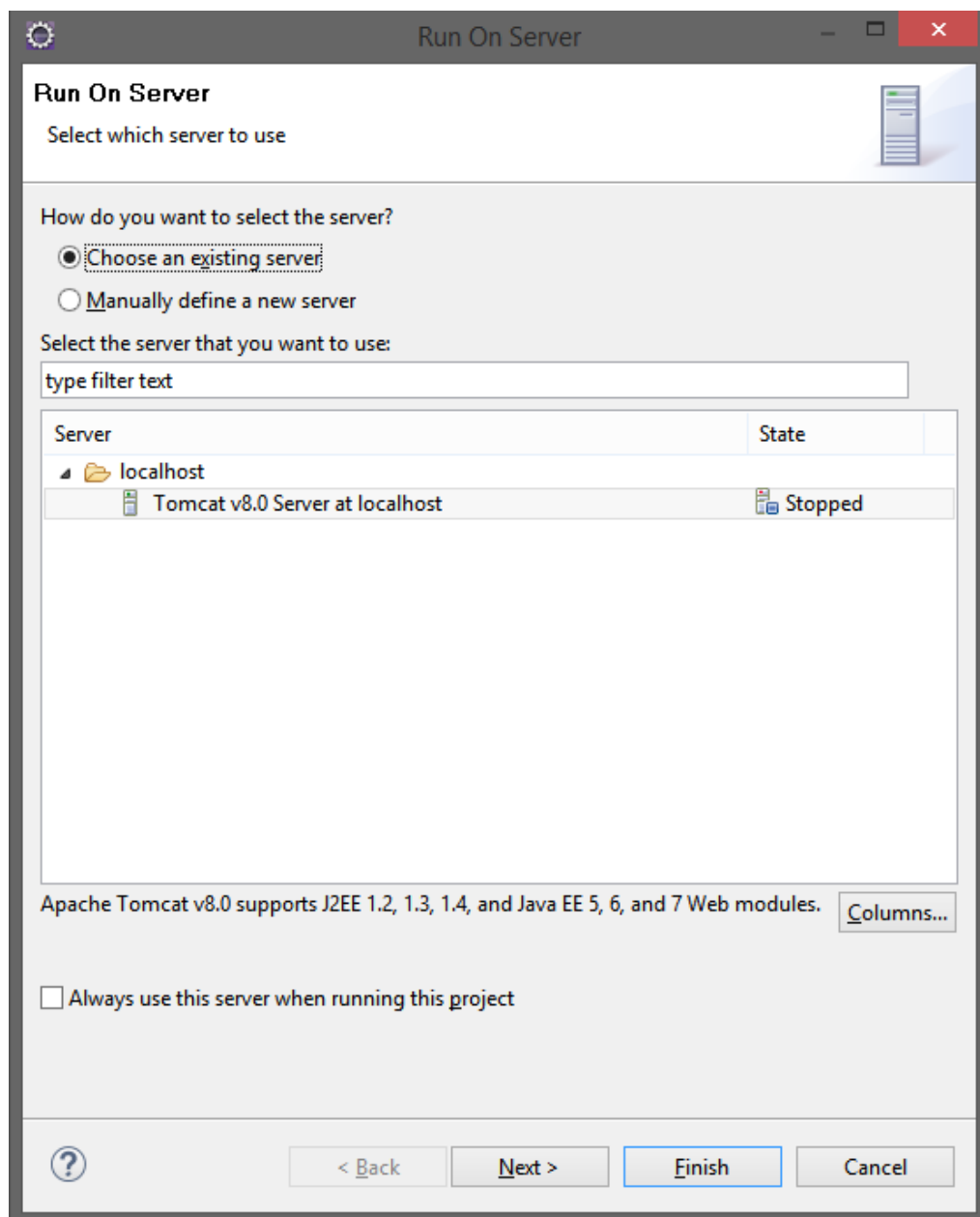
Step 9. Now your Servlet is created, write the code inside it.



Step 10. Now all you have to do is Start the server and run the application.



Step 11. Select the existing Tomcat server and click finish



**7.a) Implement a JAVA Servlet Program to implement a dynamic HTML using Servlet (user name and password should be accepted using HTML and displayed using a Servlet).**

Create a new servlet named Servlet9 in the project (as shown in the steps above from Page 37) and then type the following code in it

Servlet9.java

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.*;

@WebServlet("/Servlet9")
public class Servlet9 extends HttpServlet {

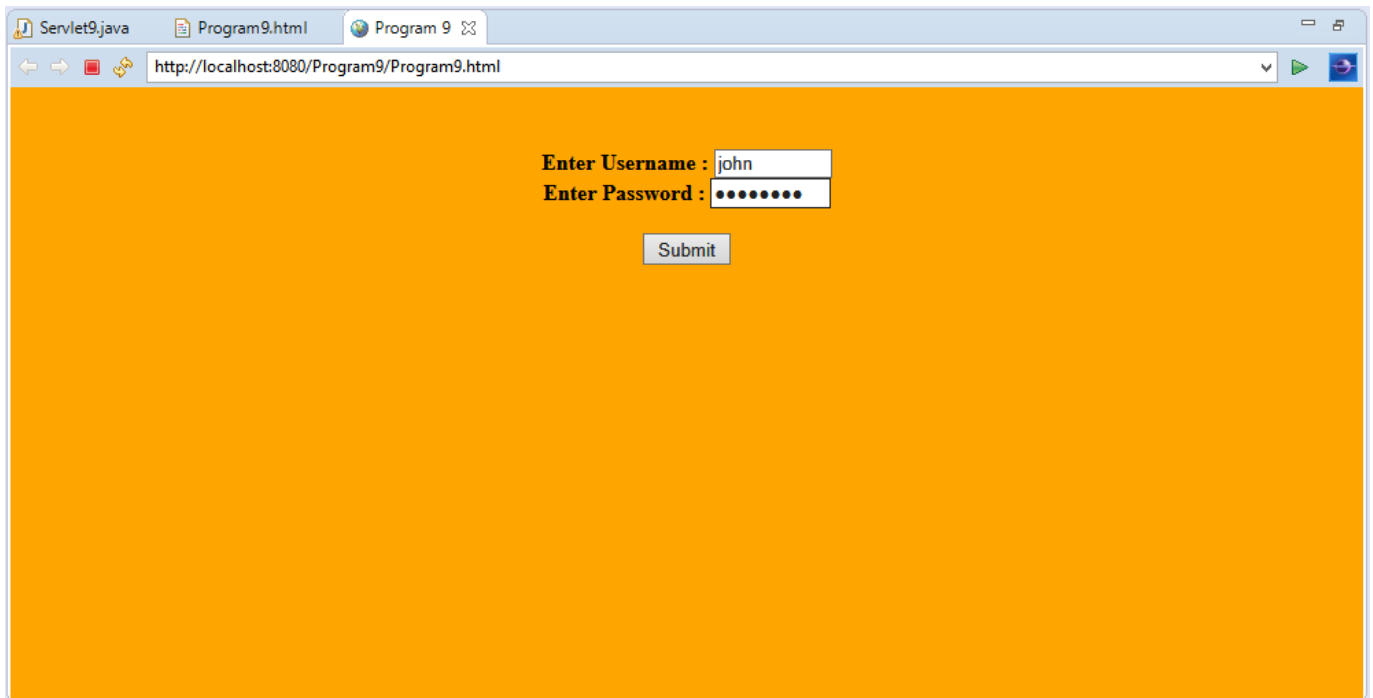
    protected void doPost(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
        response.setContentType("text/html");
        PrintWriter out = response.getWriter();
        String str = request.getParameter("uname");
        String str1 = request.getParameter("pname");
        out.println("<html>");
        out.println("<body>");
        out.println("Username is : " + str + "<br/>");
        out.println("Password is : " + str1);
        out.println("</body>");
        out.println("</html>"); }
}
```

Under WebContent, create a new html file, Program9.html

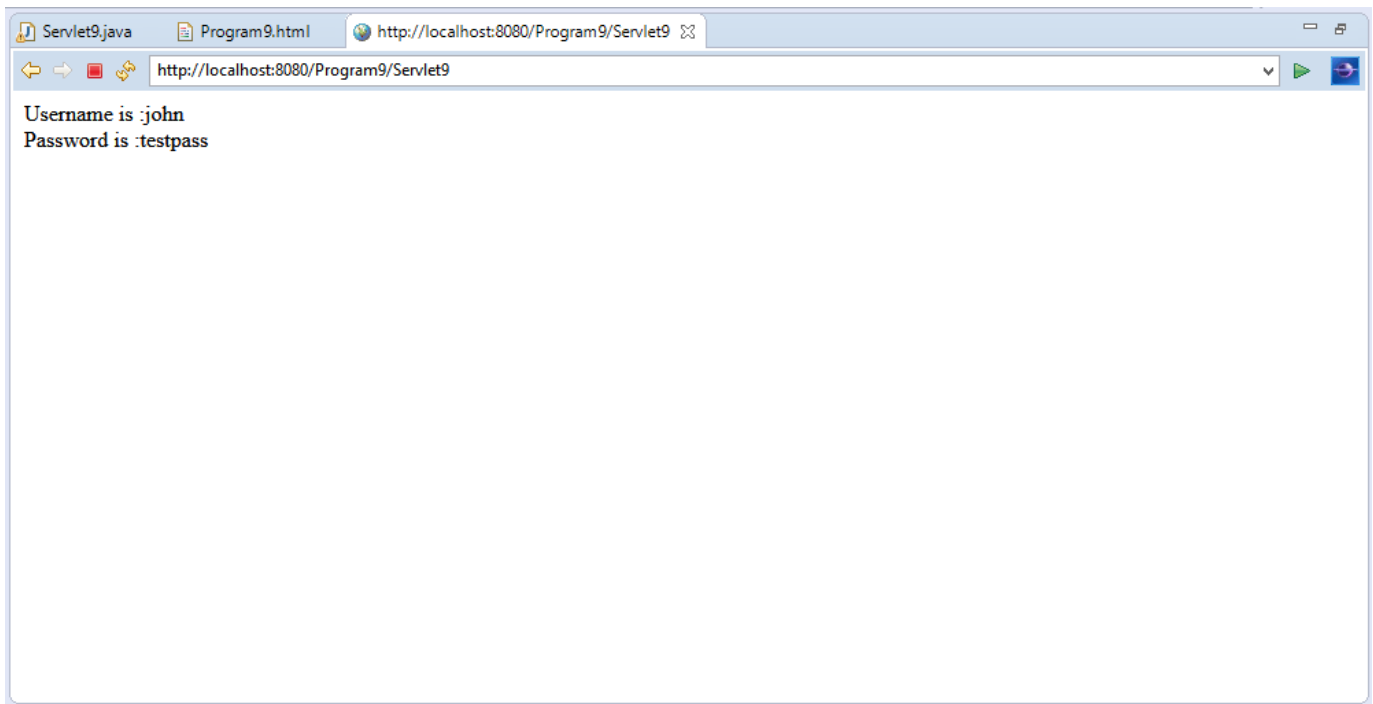
```
<html>
<head>
<title>Program 9</title>
</head>
<body bgcolor=orange>
<form method="post" name="form1"
action="http://localhost:8080/ProjectName/ServletClassName">
<center>
<b><br/><br/>
Enter Username : <input type="text" name="uname" size="10"/>
<br/>
Enter Password : <input type="password" name="pname" size="10"/>
<br/><br/>
<input type="button" value="Submit" onclick="submit()" />
</center>
<script type="text/javascript">
function validate(){
if(document.form1.uname.value =="" || document.form1.pname.value == ""){
alert("Fields cannot be blank");
return;
}
}
</script>
</form>
</body>
</html>
```

In the above html file, replace **ProjectName** and **ServletClassName** with your respective project and filename

## Output



A screenshot of a web browser window. The address bar shows the URL `http://localhost:8080/Program9/Program9.html`. The page has an orange background. In the center, there is a login form with two input fields. The first field is labeled "Enter Username :" and contains the text "john". The second field is labeled "Enter Password :" and contains ten dots. Below these fields is a button labeled "Submit".



A screenshot of a web browser window. The address bar shows the URL `http://localhost:8080/Program9/Servlet9`. The page is white. At the top left, there is text that reads "Username is :john" and "Password is :testpass".

**7.b) Design a JAVA Servlet Program to Download a file and display it on the screen (A link has to be provided in HTML, when the link is clicked corresponding file has to be displayed on Screen).**

Create a new servlet named Servlet10 in the project (as shown in the steps in Page 37) and then type the following code in it

Servlet10.java

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.*;

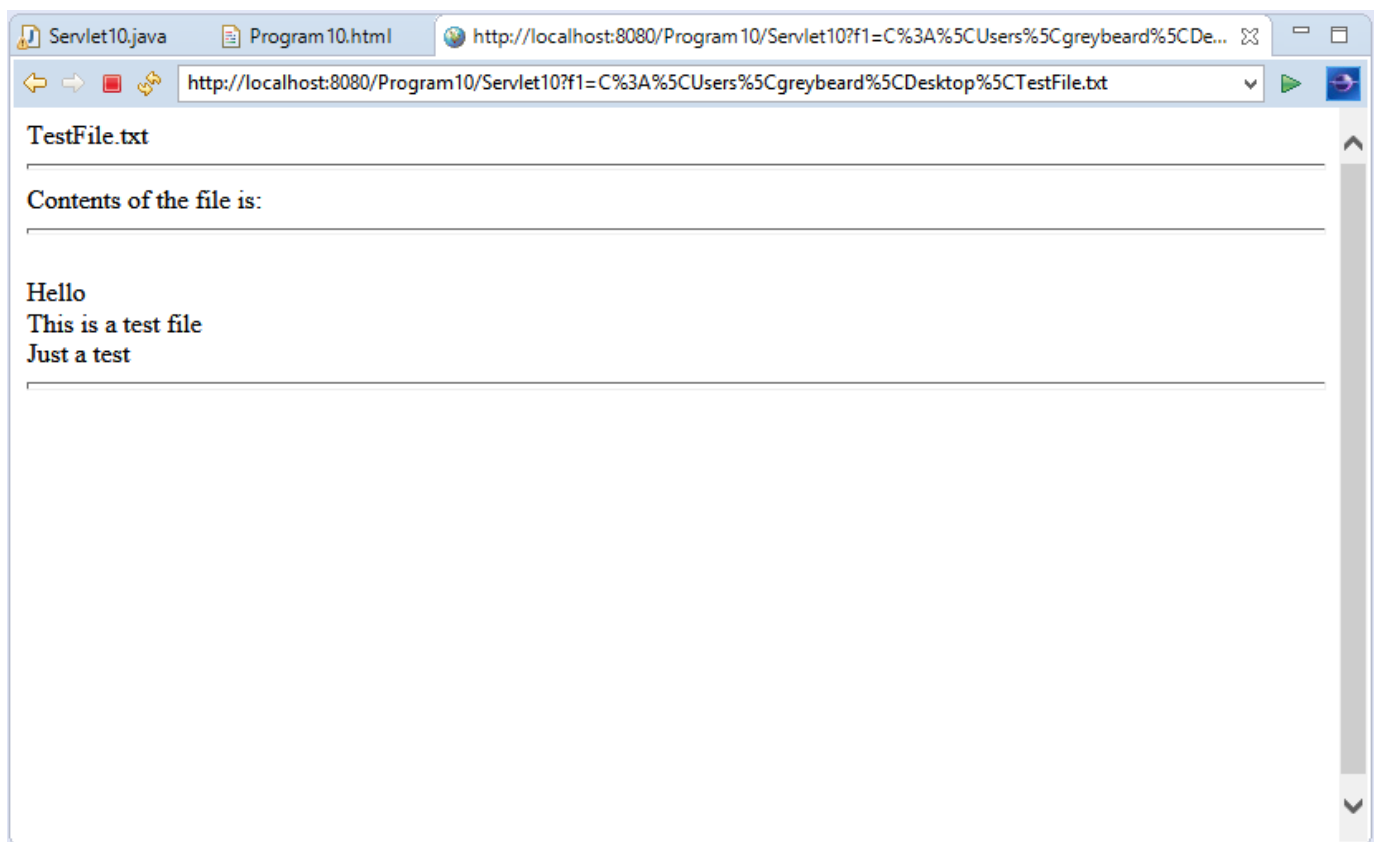
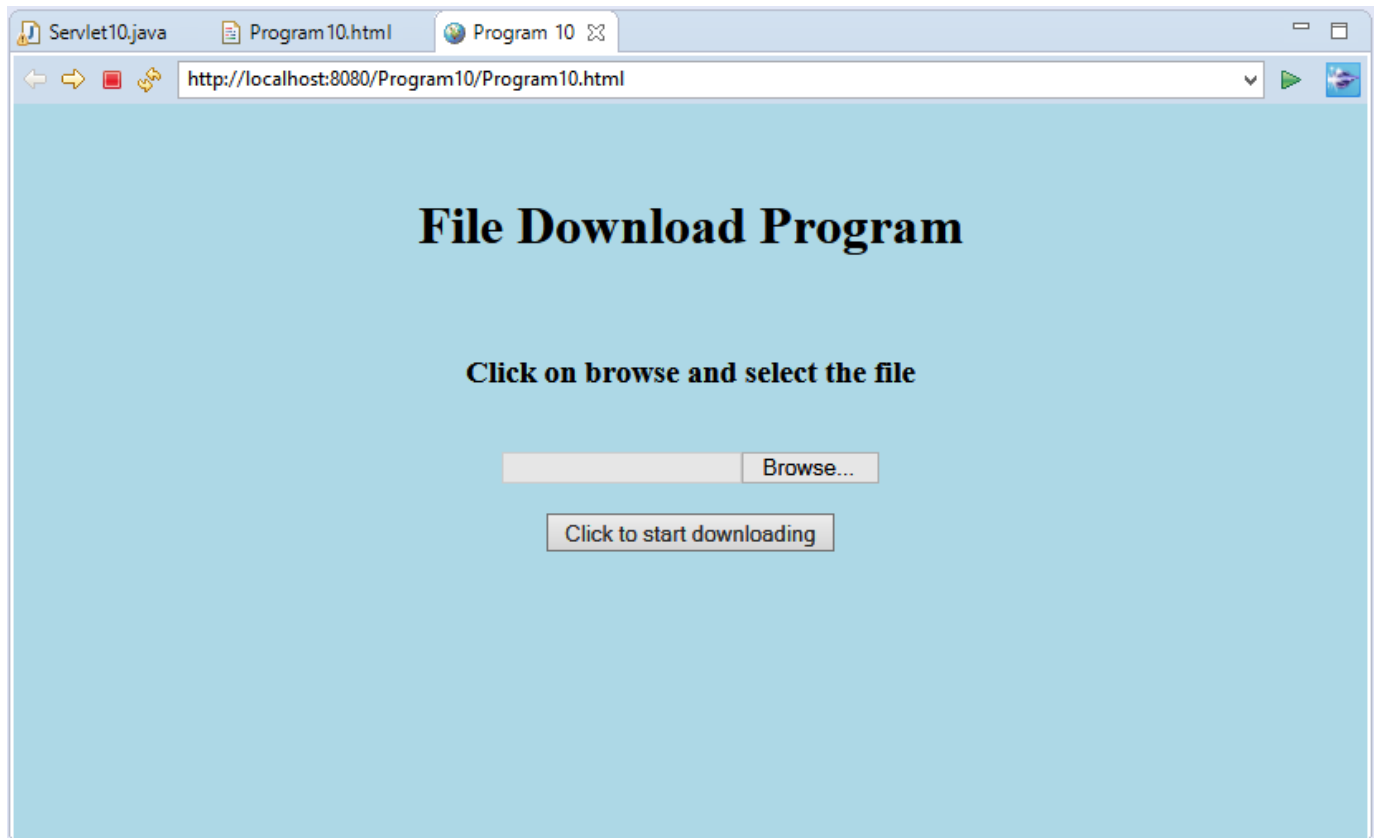
@WebServlet("/Servlet10")
public class Servlet10 extends HttpServlet {

    protected void doGet(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
        response.setContentType("text/html");
        PrintWriter out = response.getWriter();
        String fname = request.getParameter("f1");
        System.out.println(fname);
        File f = new File(fname);
        if (f.exists())
        {
            out.println(f.getName());
            out.println("<hr size='2' style='color:green'>");
            out.println("Contents of the file is:<br>");
            out.println("<hr size='2' style='color:green' /><br>");
            BufferedReader in = new BufferedReader(new FileReader(f));
            String buf = "";
            while ((buf = in.readLine()) != null)
            {
                out.write(buf);
                out.flush();
                out.println("<br>");
            }
            in.close();
            out.println("<hr size='3' style='color:red'></font></p></body>\n</html>");
        }
        else
        {
            out.println("Filename:" + fname);
            out.println("<h1>File doesn't exist</h1>\n");
        }
    }
}
```

Under WebContent, create a new html file, Program10.html

```
<!DOCTYPE html>
<html>
<head>
<title>Program 10</title>
</head>
<script type="text/javascript">
    function validate() {
        if (document.form1.f1.value == "")
            alert("First click on browse and select the file");
        else
            document.from1.submit();
    }
</script>
<body bgcolor="lightblue">
    <form name="form1" method="get"
        action="http://localhost:8080/ProjectName/ServletClassName">
        <p>
            <center>
                <br />
                <h1>File Download Program</h1>
                <br />
                <h3>Click on browse and select the file</h3>
                <br /><input type="file" name="f1"><br />
                <br /><input type="submit" value="Click to start downloading"
                    onclick="validate()" />
            </center>
        </p>
    </form>
</body>
</html>
```

In the above html file, replace **ProjectName** and **ServletClassName** with your respective project and filename



## REMOTE METHOD INVOCATION (RMI)



The **RMI** (Remote Method Invocation) is an API that provides a mechanism to create distributed application in java. The RMI allows an object to invoke methods on an object running in another JVM.

The RMI provides remote communication between the applications using two objects *stub* and *skeleton*.

### Understanding stub and skeleton

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.

#### stub

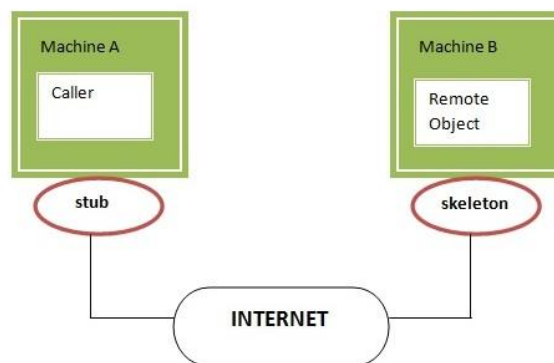
The stub is an object, acts as a gateway for the client side. All the outgoing requests are routed through it. It resides at the client side and represents the remote object. When the caller invokes method on the stub object, it does the following tasks:

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 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.



### 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

### 8.a) Design and implement a simple Client Server Application using RMI.

#### AddServerIntf.java

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

#### AddServerImpl.java

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

#### AddServer.java

```
import java.rmi.*;
public class AddServer {
    public static void main(String[] args) {
        try{
            AddServerImpl server = new AddServerImpl();
            Naming.rebind("registerme",server);
            System.out.println("Server is running...");
        } catch (Exception e) {
            System.out.println(e);
        }
    }
}
```

#### AddClient.java

```
import java.rmi.*;
public class AddClient {
    public static void main(String[] args) {
        try{
            AddServerIntf client =
            (AddServerIntf)Naming.lookup("registerme");
            System.out.println("First number is :" + args[0]);
            int x = Integer.parseInt(args[0]);
            System.out.println("Second number is :" + args[1]);
            int y = Integer.parseInt(args[1]);
            System.out.println("Sum =" + client.add(x,y));
        } catch (Exception e){
            System.out.println(e);
        }
    }
}
```

### Output:

Open a terminal

Navigate to the src folder of your project

```
Terminal
+ C:\Users\greybeard\IdeaProjects\Test\src>javac AddServerImpl.java
X C:\Users\greybeard\IdeaProjects\Test\src>javac AddServer.java

C:\Users\greybeard\IdeaProjects\Test\src>javac AddClient.java

C:\Users\greybeard\IdeaProjects\Test\src>rmic AddServerImpl
Warning: generation and use of skeletons and static stubs for JRMP
is deprecated. Skeletons are unnecessary, and static stubs have
been superseded by dynamically generated stubs. Users are
encouraged to migrate away from using rmic to generate skeletons and static
stubs. See the documentation for java.rmi.server.UnicastRemoteObject.

C:\Users\greybeard\IdeaProjects\Test\src>start rmiregistry

C:\Users\greybeard\IdeaProjects\Test\src>
```

In another terminal (while previous one is still running)

Navigate to the src folder of your project

```
C:\Users\greybeard\IdeaProjects\Test\src>java AddServer
Server is running...
```

In third terminal (while previous both are still open)

Navigate to the src folder of your project

```
C:\Users\greybeard\IdeaProjects\Test\src>java AddClient 2 3
First number is :2
Second number is :3
Sum =5

C:\Users\greybeard\IdeaProjects\Test\src>
```

## SOCKET PROGRAMMING

Sockets provide the communication mechanism between two computers using TCP. A client program creates a socket on its end of the communication and attempts to connect that socket to a server.

When the connection is made, the server creates a socket object on its end of the communication. The client and server can now communicate by writing to and reading from the socket.

The **java.net.Socket** class represents a socket, and the **java.net.ServerSocket** class provides a mechanism for the server program to listen for clients and establish connections with them.

The following steps occur when establishing a TCP connection between two computers using sockets:

- The server instantiates a **ServerSocket** object, denoting which port number communication is to occur on.
- The server invokes the **accept()** method of the **ServerSocket** class. This method waits until a client connects to the server on the given port.
- After the server is waiting, a client instantiates a **Socket** object, specifying the server name and port number to connect to.
- The constructor of the **Socket** class attempts to connect the client to the specified server and port number. If communication is established, the client now has a **Socket** object capable of communicating with the server.
- On the server side, the **accept()** method returns a reference to a new socket on the server that is connected to the client's socket.

After the connections are established, communication can occur using I/O streams. Each socket has both an **OutputStream** and an **InputStream**. The client's **OutputStream** is connected to the server's **InputStream**, and the client's **InputStream** is connected to the server's **OutputStream**.

TCP is a two-way communication protocol, so data can be sent across both streams at the same time. There are following useful classes providing complete set of methods to implement sockets.

### ServerSocket Class Methods:

The **java.net.ServerSocket** class is used by server applications to obtain a port and listen for client requests

One of the four **ServerSocket** constructors are shown below:

**public ServerSocket(int port) throws IOException**

Attempts to create a server socket bound to the specified port. An exception occurs if the port is already bound by another application.

If the **ServerSocket** constructor does not throw an exception, it means that your application has successfully bound to the specified port and is ready for client requests.

Here are some of the common methods of the **ServerSocket** class:

Sl.No	Methods with Description
1	<b>public int getLocalPort()</b>

	Returns the port that the server socket is listening on. This method is useful if you passed in 0 as the port number in a constructor and let the server find a port for you.
2	<b>public Socket accept() throws IOException</b> Waits for an incoming client. This method blocks until either a client connects to the server on the specified port or the socket times out, assuming that the time-out value has been set using the <code>setSoTimeout()</code> method. Otherwise, this method blocks indefinitely
3	<b>public void setSoTimeout(int timeout)</b> Sets the time-out value for how long the server socket waits for a client during the <code>accept()</code> .
4	<b>public void bind(SocketAddress host, int backlog)</b> Binds the socket to the specified server and port in the <code>SocketAddress</code> object. Use this method if you instantiated the <code>ServerSocket</code> using the no-argument constructor.

When the `ServerSocket` invokes `accept()`, the method does not return until a client connects. After a client does connect, the `ServerSocket` creates a new `Socket` on an unspecified port and returns a reference to this new `Socket`. A TCP connection now exists between the client and server, and communication can begin.

### Socket Class Methods:

The **java.net.Socket** class represents the socket that both the client and server use to communicate with each other. The client obtains a `Socket` object by instantiating one, whereas the server obtains a `Socket` object from the return value of the **accept()** method.

The `Socket` class has five constructors that a client uses to connect to a server. One of them is shown below:

**public Socket(String host, int port) throws UnknownHostException, IOException.**

This method attempts to connect to the specified server at the specified port. If this constructor does not throw an exception, the connection is successful and the client is connected to the server.

When the `Socket` constructor returns, it does not simply instantiate a `Socket` object but it actually attempts to connect to the specified server and port.

Some methods of interest in the `Socket` class are listed here. Notice that both the client and server have a `Socket` object, so these methods can be invoked by both the client and server.

Sl.No.	Methods with Description
1	<b>public int getPort()</b> Returns the port the socket is bound to on the remote machine.
2	<b>public SocketAddress getRemoteSocketAddress()</b> Returns the address of the remote socket.
3	<b>public InputStream getInputStream() throws IOException</b> Returns the input stream of the socket. The input stream is connected to the output stream of the remote socket.
4	<b>public OutputStream getOutputStream() throws IOException</b> Returns the output stream of the socket. The output stream is connected to the input stream of the remote socket
5	<b>public void close() throws IOException</b> Closes the socket, which makes this <code>Socket</code> object no longer capable of connecting again to any server

\

**8.b) Design and implement Client Server communication using socket programming (Client requests a file, Server responds to client with contents of that file which is then display on the screen by Client).**Client.java

```
import java.net.*;
import java.io.*;
public class Client {
    public static void main(String[] args) {
        Socket client = null;
        BufferedReader br = null;
        try {
            System.out.println(args[0] + " " + args[1]);
            client = new Socket(args[0], Integer.parseInt(args[1]));
        } catch (Exception e){}
        DataInputStream input = null;
        PrintStream output = null;
        try {
            input = new DataInputStream(client.getInputStream());
            output = new PrintStream(client.getOutputStream());
            br = new BufferedReader(new InputStreamReader(System.in));
            String str = input.readLine(); //get the prompt from the server
            System.out.println(str);
            String filename = br.readLine();
            if (filename!=null){
                output.println(filename);
            }
            String data;
            while ((data=input.readLine())!=null) {
                System.out.println(data);
            }
            client.close();
        } catch (Exception e){
            System.out.println(e);
        }
    }
}
```

### Server.java

```
import java.net.*;
import java.io.*;

public class Server {
    public static void main(String[] args) {
        ServerSocket server = null;
        try {
            server = new ServerSocket(Integer.parseInt(args[0]));
        } catch (Exception e) {
        }
        while (true) {
            Socket client = null;
            PrintStream output = null;
            DataInputStream input = null;
            try {
                client = server.accept();
            } catch (Exception e) {
                System.out.println(e);
            }
            try {
                output = new PrintStream(client.getOutputStream());
                input = new DataInputStream(client.getInputStream());
            } catch (Exception e) {
                System.out.println(e);
            }
            //Send the command prompt to client
            output.println("Enter the filename :>");
            try {
                //get the filename from client
                String filename = input.readLine();
                System.out.println("Client requested file :" + filename);
                try {
                    File f = new File(filename);
                    BufferedReader br = new BufferedReader(new
FileReader(f));

                    String data;
                    while ((data = br.readLine()) != null) {
                        output.println(data);
                    }
                } catch (FileNotFoundException e) {
                    output.println("File not found");
                }
                client.close();
            } catch (Exception e) {
                System.out.println(e);
            }
        }
    }
}
```

### Output

Create a file called testfile.txt in the folder where Client.java and Server.java is located. Add some content.

Open two terminals

Navigate to the src folder of your project

```
C:\Users\greybeard\IdeaProjects\Test\src>javac Server.java
Note: Server.java uses or overrides a deprecated API.
Note: Recompile with -Xlint:deprecation for details.

C:\Users\greybeard\IdeaProjects\Test\src>java Server 4000
Client requested file :testfile.txt
□
```

```
C:\Users\greybeard\IdeaProjects\Test\src>javac Client.java
Note: Client.java uses or overrides a deprecated API.
Note: Recompile with -Xlint:deprecation for details.

C:\Users\greybeard\IdeaProjects\Test\src>java Client localhost 4000
localhost 4000
Enter the filename :>
testfile.txt
Hello
How are you?

C:\Users\greybeard\IdeaProjects\Test\src>
```



**9.a) Design and implement a simple JDBC application program.**

```
import java.sql.*;
class MysqlCon{
public static void main(String args[]){ try{
Class.forName("com.mysql.jdbc.Driver");
Connection con=DriverManager.getConnection(
"jdbc:mysql://localhost:3306/sonoo","root","root");
//here sonoo is database name, root is username and password
Statement stmt=con.createStatement();
ResultSet rs=stmt.executeQuery("select * from emp");
while(rs.next())
System.out.println(rs.getInt(1)+" "+rs.getString(2)+" "+rs.getString(3));
con.close();
}catch(Exception e){ System.out.println(e);}
}
}
```

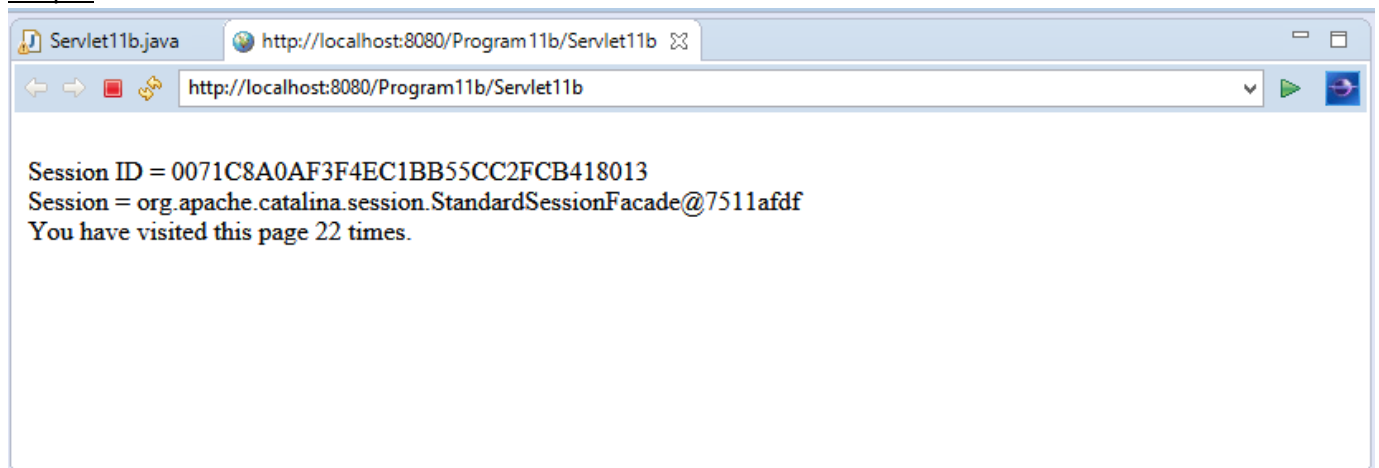
**9.b) Implement a JAVA Servlet Program to implement sessions using HTTP Session Interface.**

Create a new servlet named Servlet11b in the project (as shown in the steps in Page 37) and then type the following code in it

Servlet11b.java

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.*;

@WebServlet("/Servlet11b")
public class Servlet11b extends HttpServlet {
    protected void doGet(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
        response.setContentType("text/html");
        PrintWriter out = response.getWriter();
        HttpSession session = request.getSession(true);
        String id = session.getId();
        out.println("<html>");
        out.println("<body>");
        out.println("<br>");
        out.println("Session ID = " + id);
        out.println("<br>");
        out.println("Session = " + session);
        out.println("<br>");
        Integer val = (Integer) session.getAttribute("sessiontest.counter");
        if(val == null)
            val = new Integer(1);
        else
            val = new Integer(val.intValue()+1);
        session.setAttribute("sessiontest.counter", val);
        out.println("You have visited this page " + val + " times.");
        out.println("</body>");
        out.println("</html>");
    }
}
```

Output

## JavaServer Pages (JSP)

JavaServer Pages (JSP) is a server-side programming technology that enables the creation of dynamic, platform-independent method for building Web-based applicationsby making use of special JSP tags, most of which start with <% and end with %>.

A JavaServer Pages component is a type of Java servlet that is designed to fulfill the role of a user interface for a Java web application. Web developers write JSPs as text files that combine HTML or XHTML code, XML elements, and embedded JSP actions and commands.

Using JSP, you can collect input from users through web page forms, present records from a database or another source, and create web pages dynamically.

JSP tags can be used for a variety of purposes, such as retrieving information from a database or registering user preferences, accessing JavaBeans components, passing control between pages and sharing information between requests, pages etc.

### **Advantages of using JSP**

JavaServer Pages often serve the same purpose as programs implemented using the Common Gateway Interface (CGI). But JSP offer several advantages in comparison with the CGI.

- Performance is significantly better because JSP allows embedding Dynamic Elements in HTML Pages itself instead of having a separate CGI files.
- JSP are always compiled before it's processed by the server unlike CGI/Perl which requires the server to load an interpreter and the target script each time the page is requested.
- JavaServer Pages are built on top of the Java Servlets API, so like Servlets, JSP also has access to all the powerful Enterprise Java APIs, including JDBC, JNDI, EJB, JAXP etc.
- JSP pages can be used in combination with servlets that handle the business logic, the model supported by Java servlet template engines.

Finally, JSP is an integral part of Java EE, a complete platform for enterprise class applications. This means that JSP can play a part in the simplest applications to the most complex and demanding.

**The general syntax and tags used for JSP development is shown below:**

### **The Scriptlet:**

A scriptlet can contain any number of JAVA language statements, variable or method declarations, or expressions that are valid in the page scripting language.

Following is the syntax of Scriptlet:

```
<%  
code fragment  
%>
```

### **JSP Declarations:**

A declaration declares one or more variables or methods that you can use in Java code later in the JSP file. You must declare the variable or method before you use it in the JSP file.

Following is the syntax of JSP Declarations:

```
<%!declaration;[ declaration;]+... %>
```

Following is the simple example for JSP Declarations:

```
<%!int i =0; %>  
<%!int a, b, c; %>
```

### **JSP Expression:**

A JSP expression element contains a scripting language expression that is evaluated, converted to a String, and inserted where the expression appears in the JSP file.

Because the value of an expression is converted to a String, you can use an expression within a line of text, whether or not it is tagged with HTML, in a JSP file.

The expression element can contain any expression that is valid according to the Java Language Specification but you cannot use a semicolon to end an expression.

Following is the syntax of JSP Expression:

```
<%= expression %>
```

Following is the simple example for JSP Expression:

```
<html>  
<head><title>A Comment Test</title></head>  
<body>  
<p>  
    Today's date: <%=newjava.util.Date()).toLocaleString()%>  
</p>  
</body>  
</html>
```

This would generate following result:

```
Today's date: 11-Sep-2010 21:24:25
```

### **JSP Comments:**

JSP comment marks text or statements that the JSP container should ignore. A JSP comment is useful when you want to hide or "comment out" part of your JSP page.

Following is the syntax of JSP comments:

```
<!--This is JSP comment -->
```

### 10.a) Design a JAVA JSP Program to implement verification of a particular user login and display a welcome page.

Create a new **Dynamic Web Project**

Under WebContent, create a new html file, Program12.html

```
<!DOCTYPE html>
<html>
<head>
<title>Program 12</title>
</head>
<body bgcolor=lightblue>
    <form method="post"
        action="http://localhost:8080/ProjectName/Verification.jsp">
        <p>
            <center>
                <br>
                <br>
                <h1>Verification of a particular User Login</h1>
                <br>
                <br> Username:<input type=text name="uname" size=10><br>
                Password:<input type=password name="pwd" size=10><br>
                <br><input type=submit value=submit>
            </center>
        </p>
    </form>
</body>
</html>
```

In the above html file, replace **ProjectName** with your respective project and filename.

Under WebContent, create a new jsp file, Verification.jsp

#### Verification.jsp

```
<%!String username=null,password=null;%>
<%
    username=request.getParameter("uname");
    password=request.getParameter("pwd");
%>
<%
    if(username.equals("john")&& password.equals("testpass"))
        response.sendRedirect("Welcome.jsp");
    else
        out.println("<center><h4>Invalid username or password</h2></center>");
%>
```

Under WebContent, create another jsp file, Welcome.jsp

#### Welcome.jsp

```
<html>
<head>
    <title>Welcome Page</title>
</head>
<body bgcolor=yellow>
    <%
        out.println("<center><h4>Welcome user<br>");
        out.println("You are now logged in!</h4></center>");
    %>
</body>
</html>
```

## Output

Program 12.html | Verification.jsp | Welcome.jsp | Program 12

http://localhost:8080/Program12/Program12.html

### Verification of a particular User Login

Username:

Password:

Program 12.html | Verification.jsp | Welcome.jsp | http://localhost:8080/Program12/Verification.jsp

http://localhost:8080/Program12/Verification.jsp

**Invalid username or password**

Program 12.html | Verification.jsp | Welcome.jsp | Program 12

http://localhost:8080/Program12/Program12.html

### Verification of a particular User Login

Username:

Password:

Program 12.html | Verification.jsp | Welcome.jsp | Welcome Page

http://localhost:8080/Program12/Welcome.jsp

**Welcome user**  
**You are now logged in!**

## JSP - JavaBeans

A JavaBean is a specially constructed Java class written in the Java and coded according to the JavaBeans API specifications.

Following are the unique characteristics that distinguish a JavaBean from other Java classes:

- It provides a default, no-argument constructor.
- It should be serializable and implement the **Serializable** interface.
- It may have a number of properties which can be read or written.
- It may have a number of "getter" and "setter" methods for the properties.

### JavaBeans Properties:

A JavaBean property is a named attribute that can be accessed by the user of the object. The attribute can be of any Java data type, including classes that you define.

A JavaBean property may be read, write, read only, or write only. JavaBean properties are accessed through two methods in the JavaBean's implementation class:

Method	Description
<b>getPropertyName()</b>	For example, if property name is <i>firstName</i> , your method name would be <code>getFirstName()</code> to read that property. This method is called accessor.
<b>setPropertyName()</b>	For example, if property name is <i>firstName</i> , your method name would be <code>setFirstName()</code> to write that property. This method is called mutator.

A read-only attribute will have only a **getPropertyName()** method, and a write-only attribute will have only a **setPropertyName()** method.

### Accessing JavaBeans:

The **useBean** action declares a JavaBean for use in a JSP. Once declared, the bean becomes a scripting variable that can be accessed by both scripting elements and other custom tags used in the JSP. The full syntax for the useBean tag is as follows:

```
<jsp:useBean id="bean's name" scope="bean's scope" typeSpec/>
```

Here values for the scope attribute could be page, request, session or application based on your requirement. The value of the **id** attribute may be any value as long as it is a unique name among other useBean declarations in the same JSP.

### Accessing JavaBeans Properties:

Along with `<jsp:useBean...>`, you can use `<jsp:getProperty/>` action to access get methods and `<jsp:setProperty/>` action to access set methods. Here is the full syntax:

```
<jsp:useBeanid="id"class="bean's class"scope="bean's scope">
<jsp:setPropertyname="bean's id"property="property name"
value="value"/>
<jsp:getPropertyname="bean's id"property="property name"/>
.....
</jsp:useBean>
```

The name attribute references the id of a JavaBean previously introduced to the JSP by the useBean action. The property attribute is the name of the get or set methods that should be invoked.

### 10.b) Design and implement a JAVA JSP Program to get student information through a HTML and create a JAVA Bean Class, populate it and display the same information through another JSP.

Create a new **Dynamic Web Project**

Under WebContent, create a new html file, Program13.html

```
<!DOCTYPE html>
<html>
<head>
<title>Student Information</title>
</head>
<body bgcolor=orange>
    <form action="http://localhost:8080/ProjectName/First.jsp" method="post">
        <center>
            <h1>student information</h1>
            <h3>
                USN          :<input type="text" name="usn" size=20 /><br>
                Student Name :<input type="text" name="sname" size=20/><br>
                Total Marks  :<input type="text" name="smarks" size=20/><br>
                <br><input type="submit" value="DISPLAY" />
            </h3>
        </center>
    </form>
</body>
</html>
```

In the above html file, replace **ProjectName** with your respective project and filename.

Under WebContent, create a new jsp file, Display.jsp

#### Display.jsp

```
<html>
<head>
<title>Student Information</title>
</head>
<body bgcolor=pink>
    <jsp:useBean id="student" scope="request" class="beans.Student" />
    <h2>Entered Student Information</h2>
    <br>
    <br>
    <h3>
        Student Name :<jsp:getProperty name="student" property="sname" /><br>
        USN :<jsp:getProperty name="student" property="usn" /><br>
        Total Marks :<jsp:getProperty name="student" property="smarks" />
    </h3>
</body>
</html>
```

Under WebContent, create another jsp file, First.jsp



### First.jsp

```
<html>
<head>
<title>Student Information</title>
</head>
<body>
    <jsp:useBean id="student" scope="request" class="beans.Student" />
    <jsp:setProperty name="student" property="*" />
    <jsp:forward page="Display.jsp" />
</body>
</html>
```

Create a new java class inside a package (ex: package beans;)

### Student.java

```
package beans;

public class Student implements java.io.Serializable {
    public String sname;
    public String usn;
    public int smarks;

    public Student() {
    }

    public void setsname(String e) {
        sname = e;
    }

    public String getsname() {
        return sname;
    }

    public void setusn(String en) {
        usn = en;
    }

    public String getusn() {
        return usn;
    }

    public void setsmarks(int m) {
        smarks = m;
    }

    public int getsmarks() {
        return smarks;
    }
}
```

Output

A screenshot of a web browser window. The address bar shows 'http://localhost:8080/Program13/Program13.html'. The page has an orange background and the title 'student information' in bold black text. Below the title, there are three input fields: 'USN : 1DA12CS000', 'Student Name : John', and 'Total Marks : 90'. A 'DISPLAY' button is centered below the fields. The browser's tab bar shows 'Display.jsp', 'First.jsp', 'Program13.html', 'Student.java', and 'Student Information'.

A screenshot of a web browser window. The address bar shows 'http://localhost:8080/Program13/First.jsp'. The page has a pink background and the title 'Entered Student Information' in bold black text. Below the title, the entered information is displayed: 'Student Name :John', 'USN :1DA12CS000', and 'Total Marks :90'. The browser's tab bar shows 'Display.jsp', 'First.jsp', 'Program13.html', 'Student.java', and 'Student Information'.