# Experiment 7

***Aim:*** Write a Java Program to implement Inheritance.

***Theory:*** Inheritance is an important pillar of OOP(Object-Oriented Programming). It is the mechanism in Java by which one class is allowed to inherit the features(fields and methods) of another class. In Java, Inheritance means creating new classes based on existing ones. A class that inherits from another class can reuse the methods and fields of that class. In addition, you can add new fields and methods to your current class as well.

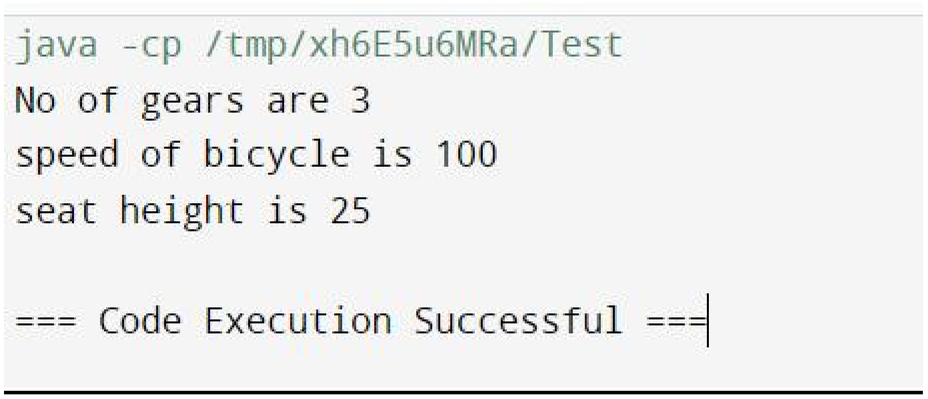
Syntax :

```
class DerivedClass extends BaseClass    –
{
    //methods and fields
}
```

***Code:***

```
class Bicycle {
        public int gear;
        public int speed;
        public Bicycle(int gear, int speed)
        {
                this.gear = gear;
                this.speed = speed;
        }

        public void applyBrake(int decrement)
        {
                speed -= decrement;
        }

        public void speedUp(int increment)
        {
                speed += increment;
        }

        public String toString()
        {
                return ("No of gears are " + gear + "\n"
                                + "speed of bicycle is " + speed);
        }
}

class MountainBike extends Bicycle {
        public int seatHeight;
        public MountainBike(int gear, int speed,int startHeight)
        {
                super(gear, speed);
                seatHeight = startHeight;
        }
        public void setHeight(int newValue)
        {
                seatHeight = newValue;
        }
        @Override public String toString()
        {
                return (super.toString() + "\nseat height is "
                                + seatHeight);
        }
}
public class Test {
        public static void main(String args[])
        {
                MountainBike mb = new MountainBike(3, 100, 25);
                System.out.println(mb.toString());
        }
}
```

***<u>Output:</u>***



```
java -cp /tmp/xh6E5u6MRa/Test
No of gears are 3
speed of bicycle is 100
seat height is 25

=== Code Execution Successful ===
```

## Experiment 8

***Aim:*** Write a Java Program to implement multiple inheritance using interface.

***Theory:*** Multiple Inheritance is a feature of an object-oriented concept, where a class can inherit properties of more than one parent class. The problem occurs when methods with the same signature exist in both the superclasses and subclass. On calling the method, the compiler cannot determine which class method to be called and even on calling which class method gets the priority.

Interface is just like a class, which contains only abstract methods. In this article, we will discuss How to Implement Multiple Inheritance by Using Interfaces in Java.

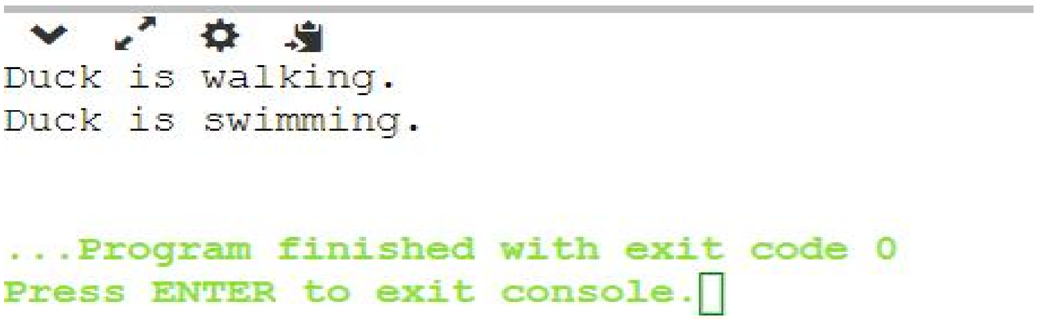
Syntax:

```
Class super
{
---
}
class sub1 Extends super
{
----
----
}
class sub2 Extend sub1
{
-----
-----
}
```

***Code:***

```
interface Walkable {
        void walk(); }
interface Swimmable {
        void swim();
}
class Duck implements Walkable, Swimmable {
        public void walk()
        {
                System.out.println("Duck is walking.");
        }
        public void swim()
        {
                System.out.println("Duck is swimming.");
        }
}
class Main {
        public static void main(String[] args)
        {
                Duck duck = new Duck();
                duck.walk();
                duck.swim();
        }
}
```

**Output:**



```
Duck is walking.
Duck is swimming.


...Program finished with exit code 0
Press ENTER to exit console.
```

# Experiment 9

***Aim:*** Write a Java Program to implement Exception Handling.

***Theory:*** The Exception Handling in Java is one of the powerful mechanism to handle the runtime errors so that the normal flow of the application can be maintained.
In Java, an exception is an event that disrupts the normal flow of the program. It is an object which is thrown at runtime.
Exception Handling is a mechanism to handle runtime errors such as ClassNotFoundException, IOException, SQLException, RemoteException, etc.

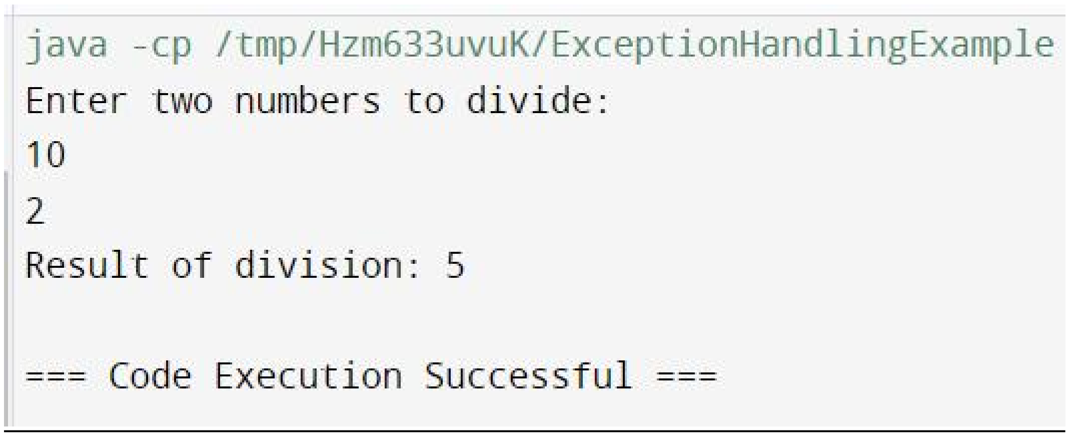
The Exception Handling in Java is one of the powerful mechanism to handle the runtime errors so that the normal flow of the application can be maintained.
In Java, an exception is an event that disrupts the normal flow of the program. It is an object which is thrown at runtime.
Exception Handling is a mechanism to handle runtime errors such as ClassNotFoundException, IOException, SQLException, RemoteException, etc.

**Code:**

```
import java.util.Scanner;
public class ExceptionHandlingExample {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.println("Enter two numbers to divide:");
        int numerator = scanner.nextInt();
        int denominator = scanner.nextInt();
        try {
            int result = divideNumbers(numerator, denominator);
            System.out.println("Result of division: " + result);
        } catch (ArithmeticException e) {
            System.out.println("Error: Division by zero is not allowed.");
        } catch (Exception e) {
            System.out.println("An unexpected error occurred: " + 
e.getMessage());
        } finally {
            scanner.close();
        }
    }
    public static int divideNumbers(int numerator, int denominator) {
        if (denominator == 0) {
```

```
                        throw new ArithmeticException("Cannot divide by zero");
                    }
                    return numerator / denominator;
            }
    }
```

**Output:**



```
java -cp /tmp/Hzm633uvuK/ExceptionHandlingExample
Enter two numbers to divide:
10
2
Result of division: 5

=== Code Execution Successful ===
```

## Experiment 10

***Aim:*** Write a Java Program to implement MultiThreading.

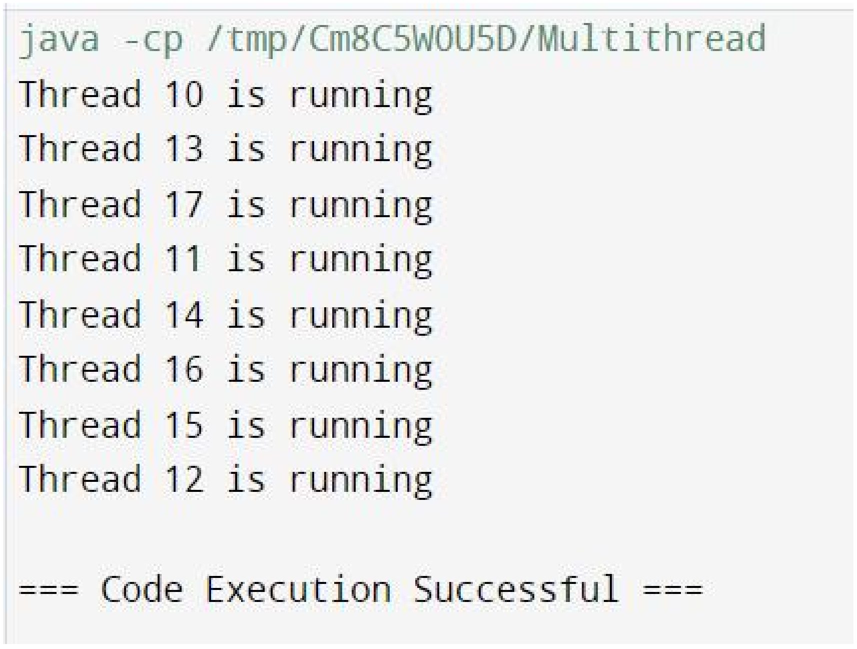
***Theory:*** Multithreading is a Java feature that allows concurrent execution of two or more parts of a program for maximum utilization of CPU. Each part of such program is called a thread. So, threads are light-weight processes within a process. Threads can be created by using two mechanisms:

- Extending the Thread class
- Implementing the Runnable Interface

**Code:**

```
class MultithreadingDemo extends Thread {
	public void run()
	{
		try {
			System.out.println(
				"Thread " + Thread.currentThread().getId()
				+ " is running");
		}
		catch (Exception e) {
			System.out.println("Exception is caught");
		}
	}
}

public class Multithread {
	public static void main(String[] args)
	{
		int n = 8; // Number of threads
		for (int i = 0; i < n; i++) {
			MultithreadingDemo object
				= new MultithreadingDemo();
			object.start();
		}
	}
}
```

**Output:**



```
java -cp /tmp/Cm8C5WOU5D/Multithread
Thread 10 is running
Thread 13 is running
Thread 17 is running
Thread 11 is running
Thread 14 is running
Thread 16 is running
Thread 15 is running
Thread 12 is running

=== Code Execution Successful ===
```

# Experiment 11

***Aim:*** Write a Java applet to display the Application Program screen i.e. calculator and other.

***Theory:***

- Create a text field to accept the expression and display the output also.
- Create buttons for digits and a decimal point.
- Create a button to clear the complete expression.
- Create the buttons for operations, that is for addition, subtraction, multiplication and division and an equals button to compute the result.
- Add ActionListener to all the buttons.
- Compute the result and display in the text field.

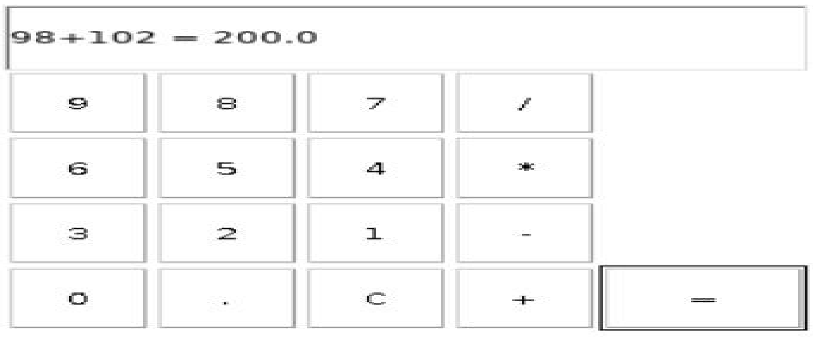
**Code:**

```
import java.awt.*;
import java.applet.*;
import java.awt.event.*;
public class Calculator extends Applet implements ActionListener
{
	TextField inp;
	public void init()
	{
	  setBackground(Color.white);
	  setLayout(null);
	  int i;
	  inp = new TextField();
	  inp.setBounds(150,100,270,50);
	  this.add(inp);
	  Button button[] = new Button[10];
	  for(i=0;i<10;i++)
	  {
		button[i] = new Button(String.valueOf(9-i));
		button[i].setBounds(150+((i%3)*50),150+((i/3)*50),50,50);
		this.add(button[i]);
		button[i].addActionListener(this);
	  }
	  Button dec=new Button(".");
	  dec.setBounds(200,300,50,50);
```

```
        this.add(dec);
        dec.addActionListener(this);

        Button clr=new Button("C");
        clr.setBounds(250,300,50,50);
        this.add(clr);
        clr.addActionListener(this);

        Button operator[] = new Button[5];
        operator[0]=new Button("/");
        operator[1]=new Button("*");
        operator[2]=new Button("-");
        operator[3]=new Button("+");
        operator[4]=new Button("=");
        for(i=0;i<4;i++)
        {
                operator[i].setBounds(300,150+(i*50),50,50);
                this.add(operator[i]);
                operator[i].addActionListener(this);
        }
        operator[4].setBounds(350,300,70,50);
        this.add(operator[4]);
        operator[4].addActionListener(this);
      }
      String num1="";
      String op="";
      String num2="";
      public void actionPerformed(ActionEvent e)
      {
        String button = e.getActionCommand();
              char ch = button.charAt(0);
        if(ch>='0' && ch<='9'|| ch=='.')
        {
                if (!op.equals(""))
                  num2 = num2 + button;
                else
                  num1 = num1 + button;
                inp.setText(num1+op+num2);
        }
        else if(ch=='C')
```

```
{
        num1 = op = num2 = "";
        inp.setText("");
}
else if (ch =='=')
{
        if(!num1.equals("") && !num2.equals(""))
        {
          double temp;
          double n1=Double.parseDouble(num1);
          double n2=Double.parseDouble(num2);
          if(n2==0 && op.equals("/"))
          {
                  inp.setText(num1+op+num2+" = Zero Division Error");
                  num1 = op = num2 = "";
          }
          else
          {
                  if (op.equals("+"))
                          temp = n1 + n2;
                  else if (op.equals("-"))
                          temp = n1 - n2;
                  else if (op.equals("/"))
                          temp = n1/n2;
                  else
                          temp = n1*n2;
                  inp.setText(num1+op+num2+" = "+temp);
                  num1 = Double.toString(temp);
                  op = num2 = "";
                }
              }
        else
        {
          num1 = op = num2 = "";
          inp.setText("");
        }
      }
else
{
        if (op.equals("") || num2.equals(""))
```

```
                    op = button;
                  else
                  {
                    double temp;
                    double n1=Double.parseDouble(num1);
                    double n2=Double.parseDouble(num2);
                    if(n2==0 && op.equals("/"))
                    {
                           inp.setText(num1+op+num2+" = Zero Division Error");
                           num1 = op = num2 = "";
                    }
                    else
                    {
                           if (op.equals("+"))
                                  temp = n1 + n2;
                           else if (op.equals("-"))
                                  temp = n1 - n2;
                           else if (op.equals("/"))
                                  temp = n1/n2;
                           else
                                  temp = n1*n2;
                           num1 = Double.toString(temp);
                           op = button;
                           num2 = "";
                         }
                       }
                  inp.setText(num1+op+num2);
                }
        }
}
```
**<u>Output:</u>**


98+102 = 200.0

# Experiment 12

***Aim:*** Program to illustrate JDBC connectivity.

***Theory:***

**Creating JDBC Application**

There are following six steps involved in building a JDBC application −

- **Import the packages:** Requires that you include the packages containing the JDBC classes needed for database programming. Most often, using *import java.sql.** will suffice.
- **Register the JDBC driver:** Requires that you initialize a driver so you can open acommunication channel with the database.
- **Open a connection:** Requires using the *DriverManager.getConnection()* method to create a Connection object, which represents a physical connection with the database.
- **Execute a query:** Requires using an object of type Statement for building and submittingan SQL statement to the database.
- **Extract data from result set:** Requires that you use the appropriate *ResultSet.getXXX()* method to retrieve the data from the result set.
- **Clean up the environment:** Requires explicitly closing all database resources versusrelying on the JVM's garbage collection.

***Code:***

```
//STEP 1. Import required packagesimport java.sql.*;
public class FirstExample {
// JDBC driver name and database URL
static final String JDBC_DRIVER = "com.mysql.jdbc.Driver"; static final String
DB_URL = "jdbc:mysql://localhost/EMP";

//  Database credentials
static final String USER = "username";static final String PASS = "password";
```

```
public static void main(String[] args) {Connection conn = null;
Statement stmt = null;try{
//STEP 2: Register JDBC driver

Class.forName("com.mysql.jdbc.Driver");

//STEP 3: Open a connection System.out.println("Connecting to database...");
conn = DriverManager.getConnection(DB_URL,USER,PASS);

//STEP 4: Execute a query System.out.println("Creating statement..."); stmt =
conn.createStatement();
String sql;
sql = "SELECT id, first, last, age FROM Employees"; ResultSet rs =
stmt.executeQuery(sql);

//STEP 5: Extract data from result setwhile(rs.next()){
//Retrieve by column nameint id  = rs.getInt("id");
int age = rs.getInt("age");
String first = rs.getString("first");String last = rs.getString("last");

//Display values System.out.print("ID: " + id); System.out.print(", Age: " + age);
System.out.print(", First: " + first); System.out.println(", Last: " + last);
}
//STEP 6: Clean-up environmentrs.close();
stmt.close();
conn.close();
}catch(SQLException se){
//Handle errors for JDBCse.printStackTrace();
}catch(Exception e){
//Handle errors for Class.forNamee.printStackTrace();
}finally{
//finally block used to close resourcestry{
if(stmt!=null) stmt.close();
}catch(SQLException se2){
}// nothing we can dotry{
if(conn!=null)
conn.close();
}catch(SQLException se){se.printStackTrace();
}//end finally try
}//end try System.out.println("Goodbye!");
}//end main
}//end FirstExample
```

**Output:**

```
Connecting to database...
Creating statement...
ID: 100, Age: 30, First: Zara, Last: Ali
ID: 101, Age: 25, First: Mahnaz, Last: Fatma
ID: 102, Age: 20, First: Zaid, Last: Khan
ID: 103, Age: 25, First: Sumit, Last: Mittal
Goodbye!
```

## Experiment 5

***Aim:*** Write a Java Program to implement Method Overloading.

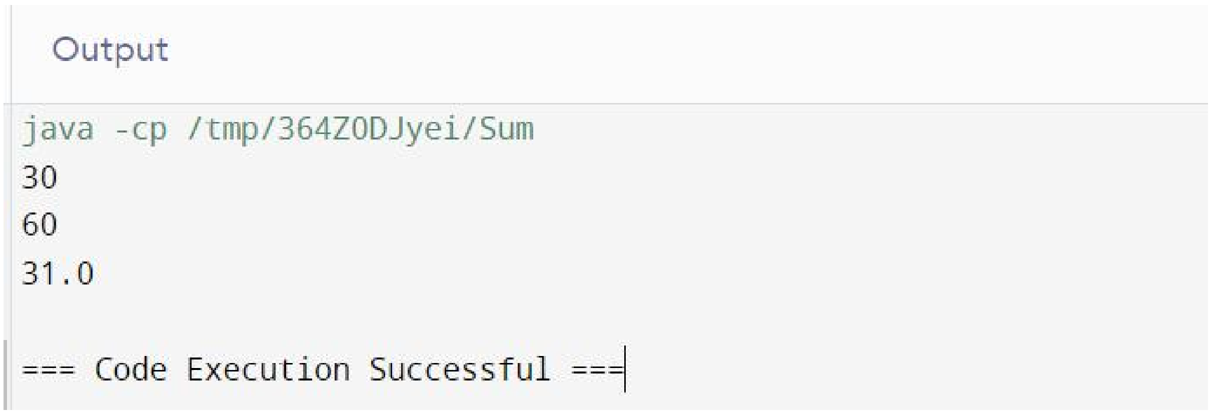
***Theory:*** In Java, Method Overloading allows different methods to have the same name, but different signatures where the signature can differ by the number of input parameters or type of input parameters, or a mixture of both.

Method overloading in Java is also known as Compile-time Polymorphism, Static Polymorphism, or Early binding. In Method overloading compared to the parent argument, the child argument will get the highest priority.

***Code:***

```
public class Sum {
        public int sum(int x, int y) { return (x + y); }
        public int sum(int x, int y, int z)
        {
                return (x + y + z);
        }       public double sum(double x, double y)
        {
                return (x + y);
        }
        public static void main(String args[])
        {
                Sum s = new Sum();
                System.out.println(s.sum(10, 20));
                System.out.println(s.sum(10, 20, 30));
                System.out.println(s.sum(10.5, 20.5));
        }
}
```

**Output:**



```
java -cp /tmp/364ZODJyei/Sum
30
60
31.0

=== Code Execution Successful ===
```

# Experiment 6

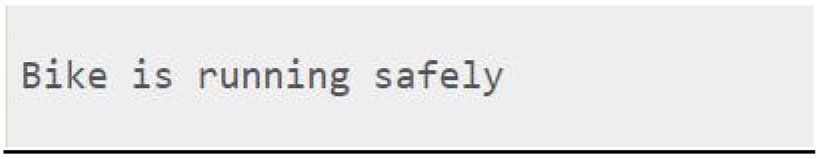
***Aim:*** Write a Java Program to implement Method Overriding.

***Theory:*** Java Overriding Overview
• Allows subclasses or child classes to override methods provided by their super-classes or parent classes.
• A method in a subclass overrides a method in its super-class if it has the same name, parameters, signature, and return type.
• Achieves Run Time Polymorphism through method overriding.
• The method's execution version is determined by the object invoked.
• If a parent class object is used, the parent class version is executed.
• The type of the object, not the reference variable type, determines the execution of an overridden method.

***Code:***

```
 class Vehicle{

 void run(){System.out.println("Vehicle is running");}

}

class Bike2 extends Vehicle{

 void run(){System.out.println("Bike is running safely");}

 public static void main(String args[]){

 Bike2 obj = new Bike2();

 obj.run();

 }

}
```

**Output:**



```
Bike is running safely
```

# PROGRAM: 13

**OBJECTIVE:** Install TOMCAT web server and APACHE. Access the above developed static web pages for books web site, using these servers by putting the web pages developed

## THEORY

### SettheJAVA_HOMEVariable

You must set the JAVA_HOME environment variable to tell Tomcat where to find Java. Failingto properly set this variable prevents Tomcat from handling JSP pages. This variable should listthe base JDK installation directory, not the bin subdirectory.

On Windows XP, you could also go to the Start menu, select Control Panel, chooseSystem, clickon the Advanced tab, press the Environment Variables button at the bottom, and enter the JAVA_HOME variable and value directly as:

Name: JAVA_HOME

Value: C:\jdk

### SettheCLASSPATH

Since servlets and JSP are not part of the Java 2 platform, standard edition, youhave to identify the servlet classes to the compiler. The server already knows about theservlet classes, but the compiler (i.e., javac ) you use for development probably doesn't.So, if you don't set your CLASSPATH, attempts to compile servlets, tag libraries, or other classes that use the servlet andJSP APIs will fail with error messages about unknownclasses.

Name: JAVA_HOME

Value: *install_dir/common/lib/servlet-api.jar*

### Turn on Servlet Reloading

The next step is to tell Tomcat to check the modification dates of the class files of requested servlets and reload ones that have changed since they were loaded into theserver's memory. Thisslightly degrades performance in deployment situations, so isturned off by default. However, if you fail to turn it on for your development server,you'll have to restart the server every time yourecompile a servlet that has already beenloaded into the server's memory.

To turn on servlet reloading, edit install_dir/conf/server.xml and add a DefaultContext subelement to the main Host element and supply true for the reloadable attribute. For example, inTomcat 5.0.27, search for this entry:

```
<Host name="localhost" debug="0" appBase="webapps" ...>and
```

then insert the following immediately below it:

```
<DefaultContext reloadable="true"/>
```

Be sure to make a backup copy of server.xm before making the above change.

## Enable the Invoker Servlet

The invoker servlet lets you run servlets without first making changes to yourWeb application's deployment descriptor. Instead, you just drop your servlet into WEB-INF/classes and use the URL http://host/servlet/ServletName . The invoker servlet isextremely convenient when you arelearning and even when you are doing your initialdevelopment.

To enable the invoker servlet, uncomment the following servlet and servlet-mapping elements in install_dir/conf/web.xml. Finally, remember to make a backup copyof the original version of thisfile before you make the changes.

```
<servlet>

        <servlet-name>invoker</servlet-name>

        <servlet-class>

        org.apache.catalina.servlets.InvokerServlet

        </servlet-class>

        ...

</servlet>

...

<servlet-mapping>

        <servlet-name>invoker</servlet-name>

        <url-pattern>/servlet/*</url-pattern>

</servlet-mapping.
```

# PROGRAM: 14

**OBJECTIVE:** Assume four users user1, user2, user3 and user4 having the passwords pwd1, pwd2, pwd3 and pwd4 respectively. Write a servlet for doing the following.

1. Create a Cookie and add these four user id’s and passwords to this Cookie.
2. Read the user id and passwords entered in the Login form (week1) and authenticate with the values (user id and passwords) available in the cookies.

## THEORY

**Servlet Life cycle:**

1. Servlet class loading
2. Servlet Instantiation
3. call the init method
4. call the service method
5. call destroy method

- **Class loading and instantiation**

If you consider a servlet to be just like any other Java program, except that it runs within a servlet container, there has to be a process of loading the class and making it ready for requests. Servlets do not have the exact equivalent of a main method that causes them to start execution. When a web container starts it searches for the deployment descriptor (WEB.XML) for each of its web applications. When it finds a servlet in the descriptor it will create an instance of the servlet class. At this point the class is considered to be loaded (but not initialized).

- **The init method**

The HttpServlet class inherits the init method from GenericServlet. The init method performs a role slightly similar to a constructor in an “ordinary” Java program in that it allows initialization of an instance at start up. It is called automatically by the servlet container and as it causes the application context (WEB.XML) to be parsed and any initialization will be performed. It comes in two versions, one with a zero parameter constructor and one that takes a ServletConfig parameter.

The servlet engine creates a request object and a response object. The servlet engine invokes the servlet service() method, passing the request and response objects. Once the init method returns the servlet is said to be placed into service. The process of using init to initialize servlets means that it is possible to change configuration details by modifying the deployment descriptor without having them hard coded in with your Java source and needing a re-compilation.

void init(ServletConfig sc)

**Calling the service method**

The service() method gets information about the request from the request object, processes the request, and uses methods of the response object to create the client response. The service method can invoke other methods to process the request, such as doGet(), doPost(), or methods you write. The service method is called for each request processed and is not normally overridden by the programmer.

The code that makes a servlet "go" is the. servlet void service(ServletRequest req,ServletResponse res

**The destroy Method**

Two typical reasons for the destroy method being called are if the container is shutting down or if the container is low on resources. This can happen when the container keeps a pool of instances of servlets to ensure adequate performance. If no requests have come in for a particular servlet for a while it may destroy it to ensure resources are available for the servlets that are being requested. The destroy method is called only once, before a servlet is unloaded and thus you cannot be certain when and if it is called.

void destroy()

**ServletConfig Class**

ServletConfig object is used by the Servlet Container to pass information to the Servlet during it's initialization. Servlet can obtain information regarding initialization parameters and their values using different methods of ServletConfig class initialization parameters are name/value pairs used to provide basic information to the Servlet during it's initialization like JDBC driver name, path to database, username, password etc.

**Methods of ServletConfig class**

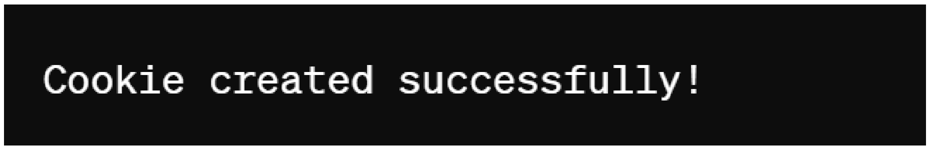
Following are the four methods of this class :

1. **getInitParameter(String paramName)** Returns value of the given parameter. If value of parameter could not be found in web.xml file then a null value is returned.
2. **GetInitParameterNames()** Returns an Enumeration object containing all the names of initialization parameters provided for this Servlet.
3. **GetServletContext()** Returns reference to the ServletContext object for this Servlet. It is similar to getServletContext() method provided by HttpServlet class.
4. **GetServletName()** Returns name of the Servlet as provided in the web.xml file or if none is provided then returns complete class path to the Servlet.

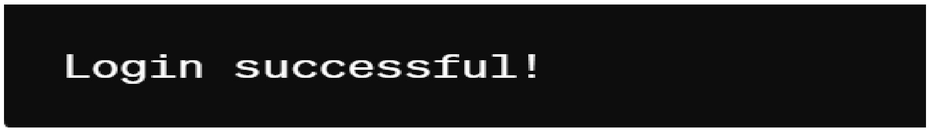
**<u>SOURCE CODE:</u>**

```
import java.io.IOException;

import javax.servlet.ServletException;

import javax.servlet.annotation.WebServlet;

import javax.servlet.http.Cookie;

import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;

import javax.servlet.http.HttpServletResponse;


@WebServlet("/LoginServlet")

public class LoginServlet extends HttpServlet {

    private static final long serialVersionUID = 1L;


    // Method to handle GET requests

    protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException
{

        // Creating a Cookie

        Cookie cookie = new Cookie("userCredentials", "user1:pwd1,user2:pwd2,user3:pwd3,user4:pwd4");

        response.addCookie(cookie);

        response.getWriter().println("Cookie created successfully!");}

    // Method to handle POST requests (Login authentication)

    protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException,
IOException {

        // Retrieving user id and password from login form

        String userId = request.getParameter("userId");

        String password = request.getParameter("password");


        // Retrieving the Cookie containing user credentials

        Cookie[] cookies = request.getCookies();

        if (cookies != null) {

            for (Cookie cookie : cookies) {

                if (cookie.getName().equals("userCredentials")) {

                    String[] userCredentials = cookie.getValue().split(",");

                    for (String userCredential : userCredentials) {

                        String[] parts = userCredential.split(":");
                        String storedUserId = parts[0];

                        String storedPassword = parts[1];

                        if (userId.equals(storedUserId) && password.equals(storedPassword)) {

                            response.getWriter().println("Login successful!");

                            return;}}

        // If user id and/or password do not match

        response.getWriter().println("Invalid user id or password!");}}
```

**OUTPUT:**



```
Cookie created successfully!
```



```
Login successful!
```

## PROGRAM: 15

**OBJECTIVE:** Install a database (Mysql or Oracle). Create a table which should contain at least the following fields: name, password, email-id, phone number Write a java program/servlet/JSP to connect to that database and extract data from the tables and display them. Insert the details of the users who register with the web site, whenever a new user clicks the submit button in the registration page.

Program to implement cookies in java Servlet.

**THEORY:**

**SOURCE CODE:** **Registration.html:**

```
<html>
<head>
<title>Registration page</title>
</head>
<body bgcolor="#00FFFf">
<form METHOD="POST" ACTION="register">
<CENTER>
<table>
<center>
<tr> <td> Username </td>
<td><input type="text" name="usr"> </td> </tr>
<tr><td> Password </td>
<td><input type="password" name="pwd"> </td> </tr>
<tr> <td>email</td>
<td> <input type="text" name="mail"> </td> </tr>
<tr> <td>Phone</td>
<td> <input type="text" name="phone"> </td> </tr>
<tr> <td colspan=2 align=center> <input type="submit" value="submit"> </td> </tr>
</center>
</table>
</form>
</body>
```

**Login.html**

```
<html>
<head>
<title>Display Information</title>
</head>
<body bgcolor=pink> <center> <table>
```

```
<form METHOD="POST" ACTION="display">
<tr> <td> Username </td>
<td><input type="text" name="usr"></td> </tr>
<tr> <td> Password </td>
<td> <input type="password" name="pwd"> </td> </tr>
<tr> <td align=center colspan="2"><input type="submit" value="submit"></td> </tr>
</table> </center>
</form>
</body>
</html>
```

**Ini.java:**

```
import javax.servlet.*;
import java.sql.*;
import java.io.*;
public class Ini extends GenericServlet
{
private String user1,pwd1,email1;

public void service(ServletRequest req,ServletResponse res) throws ServletException,IOException
{
user1=req.getParameter("user");
pwd1=req.getParameter("pwd");
email1=req.getParameter("email");
res.setContentType("text/html");
PrintWriter out=res.getWriter();
try
   {
    Class.forName("oracle.jdbc.driver.OracleDriver");
    Connection con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:XE","system","admin");
   PreparedStatement st=con.prepareStatement("insert into personal values(?,?,?,?)");
 st.setString(1,user1);
st.setString(2,pwd1);
st.setString(5,email1);
st.setString(6,"21234");
int r= st.executeUpdate();
out,println(r+ “row inserted”);
con.close();
  }
catch(SQLException s)
{  out.println("not found "+s);
}
catch(ClassNotFoundException c)
{
   out.println("not found "+c);
}
} }
```

**Display.java:**

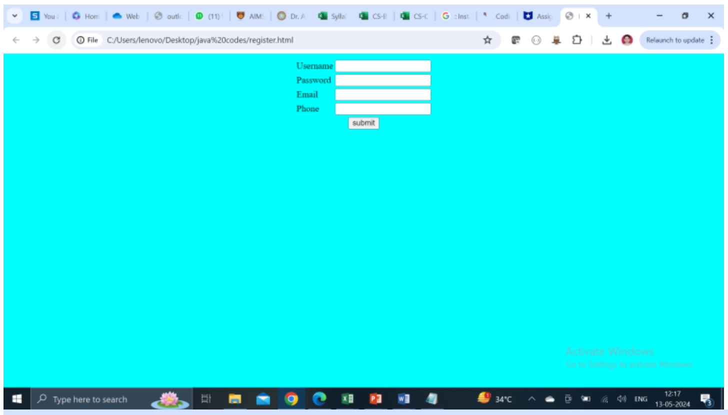
```
import javax.servlet.*;
import java.sql.*;
import java.io.*;
public class Display extends GenericServlet
{
private String user1,pwd1,email1;
public void service(ServletRequest req,ServletResponse res) throws ServletException,IOException
{
user1=req.getParameter("user");
pwd1=req.getParameter("pwd");
res.setContentType("text/html");
PrintWriter out=res.getWriter();
try
   {
    Class.forName("oracle.jdbc.driver.OracleDriver");
    Connection con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:XE","system","admin");
   Statement st=con.createStatement();
ResultSet rs=st.executeQuery("select * from emp where name="+user1);
 while(rs.next()){
out.println(rs.getString(1)+ rs.getString(2)+ rs.getString(3)+ rs.getString(4));
}
con.close();
  }
catch(SQLException s)
{  out.println("not found "+s);
}
catch(ClassNotFoundException c)
{
   out.println("not found "+c);
}
} }
```

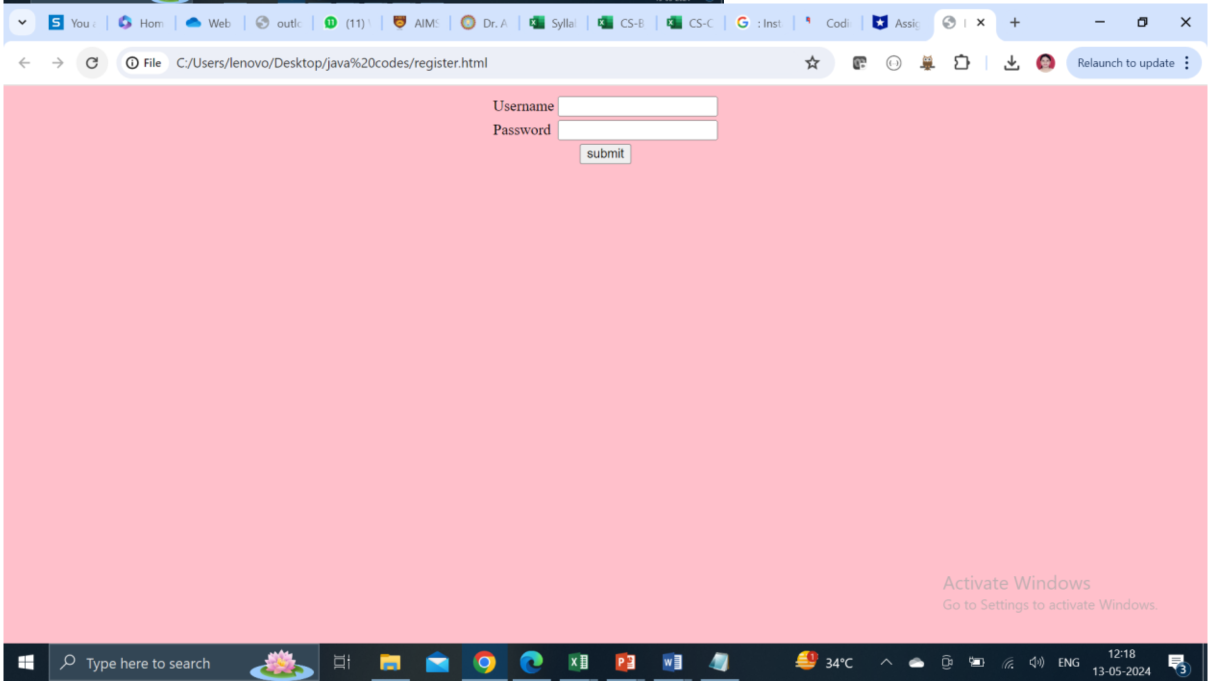
**web.xml:**

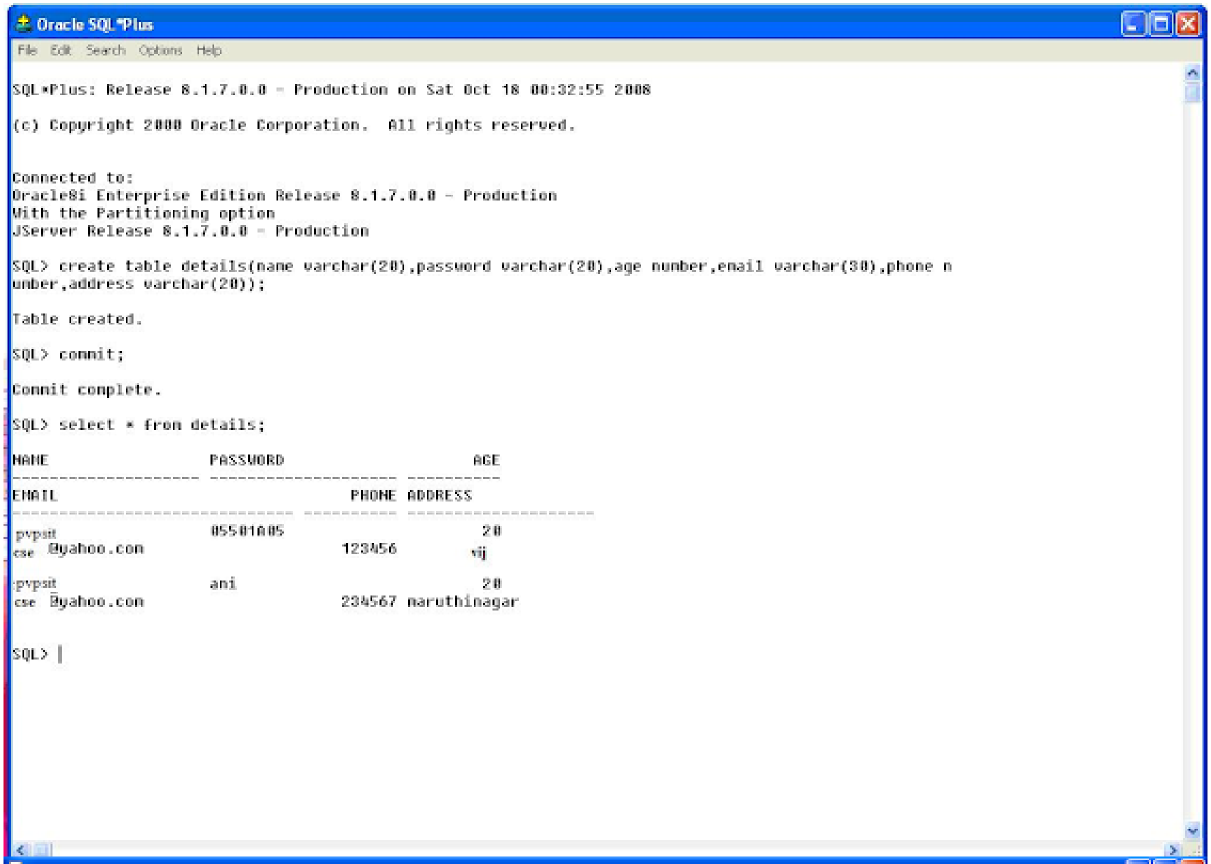
```
<web-app>
<servlet>
<servlet-name>init1</servlet-name>
<servlet-class>Ini</servlet-class>
</servlet>
```

```
<servlet-mapping>
<servlet-name>init1</servlet-name>
<url-pattern>/register</url-pattern>
</servlet-mapping>
<servlet>
<servlet-name>display</servlet-name>
<servlet-class>Display</servlet-class>
</servlet>
<servlet-mapping>
<servlet-name>display</servlet-name>
<url-pattern>/display</url-pattern>
</servlet-mapping>
</web-app>
```

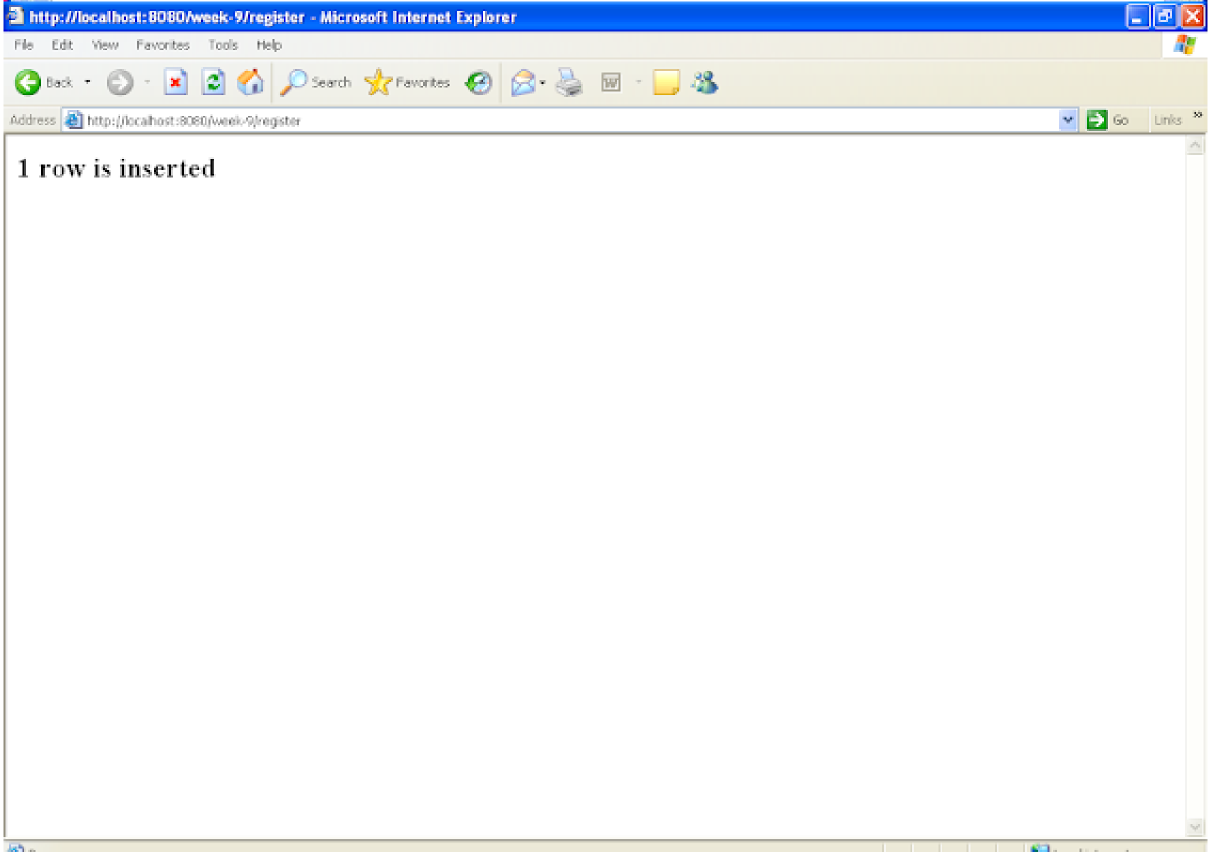
**OUTPUT:**







```
SQL*Plus: Release 8.1.7.0.0 - Production on Sat Oct 18 00:32:55 2008

(c) Copyright 2000 Oracle Corporation.  All rights reserved.


Connected to:
Oracle8i Enterprise Edition Release 8.1.7.0.0 - Production
With the Partitioning option
JServer Release 8.1.7.0.0 - Production

SQL> create table details(name varchar(20),password varchar(20),age number,email varchar(30),phone n
umber,address varchar(20));

Table created.

SQL> commit;

Commit complete.

SQL> select * from details;

NAME                 PASSWORD                    AGE
-------------------- -------------------- ----------
EMAIL                              PHONE ADDRESS
------------------------------ ---------- --------------------
pvpsit               05501A05                     20
cse  @yahoo.com                   123456         vij

pvpsit               ani                          20
cse  @yahoo.com                   234567 maruthinagar


SQL> |
```



http://localhost:8080/week-9/register - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back
Search
Favorites
Address
http://localhost:8080/week-9/register
Go
Links

1 row is inserted

## PROGRAM: 16

**OBJECTIVE:** Write a JSP which insert the details of the 3 or 4 users who register with the web site by using registration form. Authenticate the user when he submits the login form using the user name and password from the database

**THEORY:** **JSP** technology is used to create web application just like Servlet technology. It can be thought of as an extension to Servlet because it provides more functionality than servlet such as expression language, JSTL, etc.

A JSP page consists of HTML tags and JSP tags. The JSP pages are easier to maintain than Servlet because we can separate designing and development. It provides some additional features such as Expression Language, Custom Tags, etc.

### SOURCE CODE:

**UserPass.html**

```
<body>
<form method="get" action="http://localhost:8888/india/Validation.jsp">
<h3>
  Enter User Name <input type="text" name="t1"> <br>
  Enter Password   <input type="password" name="t2"> <br>
  <input type="submit" value="Please Validate">
  <input type="reset" value="Clear Please">
</h3>
</body>
```

**Validation.jsp**

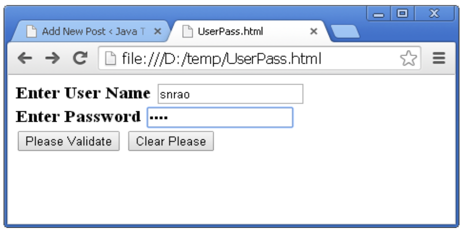
```
<body>
<h2 align="center"> Validating User Name and Password </h2>

<%
 String str1=request.getParameter("t1");
 String str2=request.getParameter("t2");

 if(str1.equalsIgnoreCase("snrao") && str2.equals("java"))
 {
  out.println("<h3>Thankyou, you are VALID</h3>");
 }
 else
 {
  out.println("<h3>Sorry, you are INVALID</h3>");
 }
%>

</body>
```

**Validation.java**

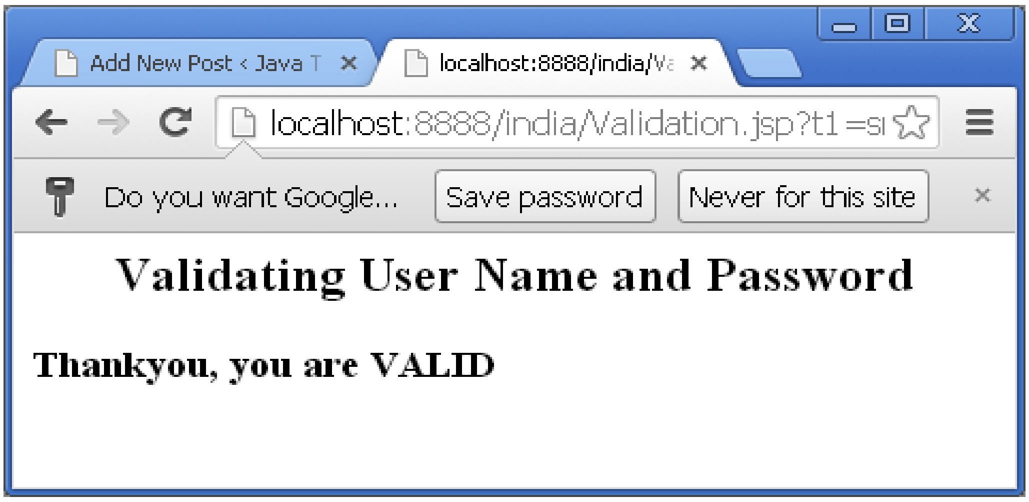
```
import javax.servlet.http.*;
import java.io.*;

public class Validation extends HttpServlet
{
  public void service(HttpServletRequest req, HttpServletResponse res) throws
ServletException, IOException
  {
    res.setContentType("text/html");
    PrintWriter pw = res.getWriter( );

    String str1 = req.getParameter("t1");
    String str2 = req.getParameter("t2");

    if(str1.equalsIgnoreCase("snrao") && str2.equals("java"))
    {
      pw.println("<h3>Thankyou, you are VALID</h3>");
    }
    else
    {
      pw.println("<h3>Sorry, you are INVALID</h3>");
    }
```

```
   pw.close( );}}
```

**OUTPUT:**

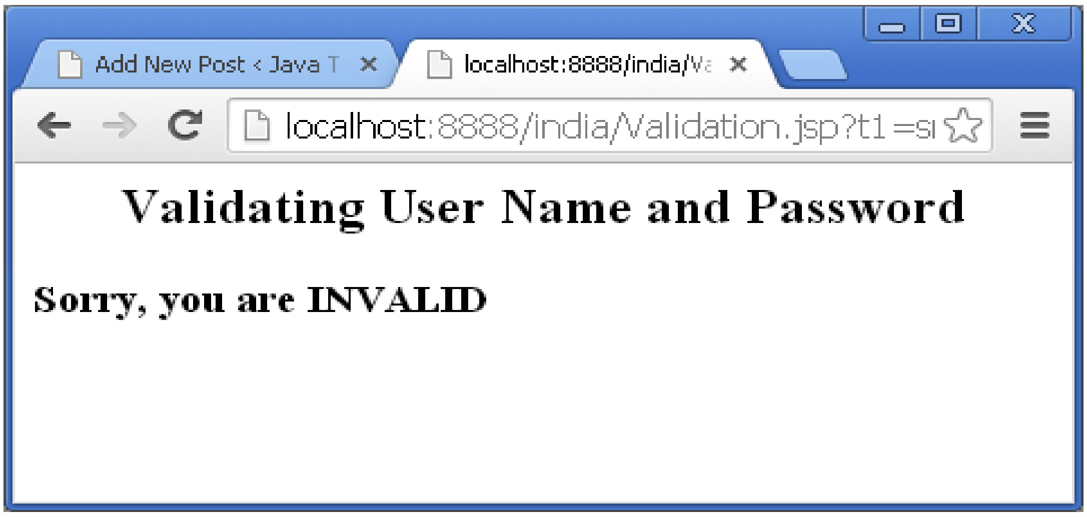


Enter User Name snrao
Enter Password ····
Please Validate  Clear Please



Validating User Name and Password

Thankyou, you are VALID



Validating User Name and Password

Sorry, you are INVALID

## PROGRAM: 17

**OBJECTIVE:** Design and implement a simple shopping cart example with session tracking API.

**THEORY:** **Session Tracking** is a way to maintain state (data) of an user. It is also known as **session** management in**servlet**. Http protocol is a stateless so we need to maintain state using **session tracking** techniques. Each time user requests to the server, server treats the request as the new request.

## SOURCE CODE:

```
<h3>Cookie Example through Shopping Cart</h3>
<body>
<form method="get" action="http://localhost:8888/india/SC">
  Enter Item Name <input type="text" name="item"><br>
  Enter Item Quantity <input type="text" name="qty"><br>
  <input type="submit" value="Add Cookie" name="add">
  <input type="submit" value="List Cookies" name="list"></form></body>
```

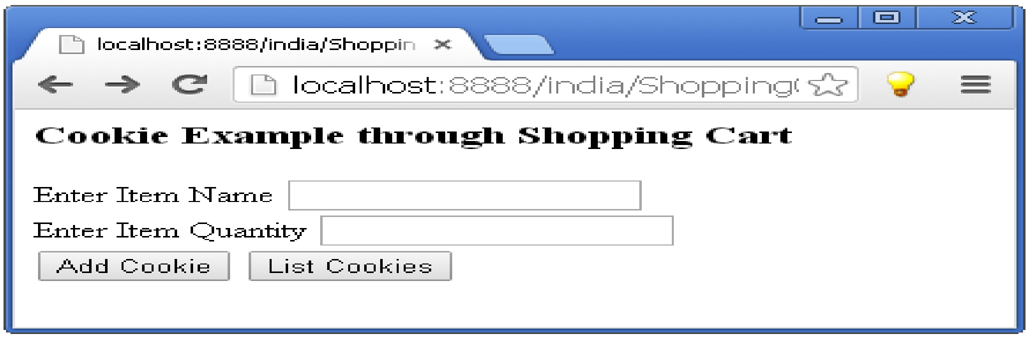
**web.xml entry for ShoppingCart servlet**

```
<servlet>
  <servlet-name>snrao1</servlet-name>
  <servlet-class>ShoppingCart</servlet-class>
</servlet>
<servlet-mapping>
  <servlet-name>snrao1</servlet-name>
  <url-pattern>/SC</url-pattern>
</servlet-mapping>
```

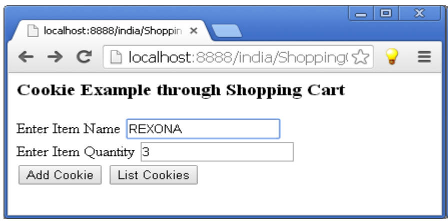
**ShoppingCart.java**

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class ShoppingCart extends HttpServlet   {
  public void service(HttpServletRequest req,HttpServletResponse res) throws ServletException,
IOException   {
    String str1 = req.getParameter("item");      // item name
    String str2 = req.getParameter("qty");       // item quantity
    String str3 = req.getParameter("add");       // submit button by name add
    String str4 = req.getParameter("list");      // submit button by name list
    res.setContentType("text/html");
    PrintWriter out = res.getWriter();
    if(str3 != null)   {
      Cookie c1 = new Cookie(str1, str2);
      res.addCookie(c1);
      res.sendRedirect("ShoppingCart.html");
    else if(str4 != null)  {
      Cookie clientCookies[] = req.getCookies();
      for( int i = 0; i < clientCookies.length; i++)    {
        out.print("<B>" + clientCookies[i].getName() + " : " + clientCookies[i].getValue() +
"</B><BR>");}}
    out.close( ) ;}}
```

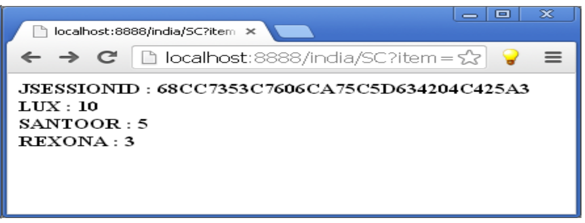
**OUTPUT:**


**Cookie Example through Shopping Cart**

Enter Item Name
Enter Item Quantity


**Cookie Example through Shopping Cart**

Enter Item Name REXONA
Enter Item Quantity 3


```
JSESSIONID : 68CC7353C7606CA75C5D634204C425A3
LUX : 10
SANTOOR : 5
REXONA : 3
```