

**Swami Keshvanand Institute of Technology
Management & Gramothan, Jaipur**

Lab Manual

5CS4-24 Advance Java Lab

Semester: V

Branch: Computer Science & Engineering



Session 2021-22

Faculty:

Kajal Mathur

(Assistant Professor-II)

Department of Computer Science and Engineering

Swami Keshvanand Institute of Technology Management & Gramothan,

Ramnagar, Jaipur-302017

Lab Manual
Advance Java LAB
5CS4-24

SESSION 2020-21

	AUTHOR/ OWNER	REVIEWED BY	APPROVED BY
NAME	Kajal Mathur	Dr. Rajat Goel	Prof (Dr.) Mukesh Gupta
DESIGNATION	Assistant Professor-II	Associate Professor	HOD(CSE)
SIGNATURE			

INDEX

Sr. No.	Topic
1	Lab Ethics & Instructions
2	Marking/Assessment System
3	List of Experiments Prescribed by RTU
4	Lab Plan
5	Lab Objectives and Requirements
6	Introduction
7	Experiment 1: Create a hello world applet
8	Experiment 2: Set the color of applet and draw a fill oval
9	Experiment 3: Design GUI in Java
10	Experiment 4: Make calculator using Java swing
11	Experiment 5: Implement the SQL login ID commands using JDBC
12	Experiment 6: Implement the SQL commands using JDBC
13	Experiment 7: Implement RMI
14	Experiment 8: Demonstrate basic Servlets
15	Experiment 9: Basic JSP program
16	Experiment 10: Database operations in JSP
17	Multiple Choice Questions
18	Practical Exam Sample Paper
19	Viva Voce(Interview based) Question and answers
20	Text & Reference Books

LAB ETHICS

Responsibilities of Users

Users are expected to follow some fairly obvious rules of conduct:



Always:

- Enter the lab on time and leave at proper time.
- Wait for the previous class to leave before the next class enters.
- Keep the bag outside in the respective racks.
- Utilize lab hours in the corresponding.
- Turn off the machine before leaving the lab unless a member of lab staff has specifically told you not to do so.
- Leave the labs at least as nice as you found them.
- If you notice a problem with a piece of equipment (e.g. a computer doesn't respond) or the room in general (e.g. cooling, heating, lighting) please report it to lab staff immediately. Do not attempt to fix the problem yourself.

Never :



- Don't abuse the equipment.
- Do not adjust the heat or air conditioners. If you feel the temperature is not properly set, inform lab staff; we will attempt to maintain a balance that is healthy for people and machines.
- Do not attempt to reboot a computer. Report problems to lab staff.
- Do not remove or modify any software or file without permission.

- Do not remove printers and machines from the network without being explicitly told to do so by lab staff.
- Don't monopolize equipment. If you're going to be away from your machine for more than 10 or 15 minutes, log out before leaving. This is both for the security of your account, and to ensure that others are able to use the lab resources while you are not.
- Don't use internet, internet chat of any kind in your regular lab schedule.
- Do not download or upload of MP3, JPG or MPEG files.
- No games are allowed in the lab sessions.
- No hardware including USB drives can be connected or disconnected in the labs without prior permission of the lab in-charge.
- No food or drink is allowed in the lab or near any of the equipment. Aside from the fact that it leaves a mess and attracts pests, spilling anything on a keyboard or other piece of computer equipment could cause permanent, irreparable, and costly damage. (and in fact *has*) If you need to eat or drink, take a break and do so in the canteen.
- Don't bring any external material in the lab, except your lab record, copy and books.
- Don't bring the mobile phones in the lab. If necessary then keep them in silence mode.
- Please be considerate of those around you, especially in terms of noise level. While labs are a natural place for conversations of all types, kindly keep the volume turned down.

Note: If you are having problems or questions, please go to either the faculty, lab in-charge or the lab supporting staff. They will help you. We need your full support and cooperation for smooth functioning of the lab.

INSTRUCTIONS

Before Entering in lab

- All the students are supposed to prepare the theory regarding the next experiment/ Program.
- Students are supposed to bring their lab records as per their lab schedule.
- Previous experiment/program should be written in the lab record.
- If applicable trace paper/graph paper must be pasted in lab record with proper labeling.
- All the students must follow the instructions, failing which he/she may not be allowed in the lab.

While working in lab

- Adhere to experimental schedule as instructed by the lab in-charge/faculty.
- Get the previously performed experiment/ program signed by the faculty/ lab in charge.
- Get the output of current experiment/program checked by the faculty/ lab in charge in the lab copy.
- Each student should work on his/her assigned computer at each turn of the lab.
- Take responsibility of valuable accessories.
- Concentrate on the assigned practical and don't play games.
- If anyone is caught red-handed carrying any equipment of the lab, then he/she will have to face serious consequences.

Marking/Assessment System

Total Marks -50

Distribution of Marks - 30 (Sessional)

Attendance	File Work	Performance	Viva	Total
05	05	10	10	10

Distribution of Marks - 20 (End Term)

File Work	Performance	Viva	Total
05	10	05	20

Rajasthan Technical University, Kota

Syllabus

III year V Semester B.Tech. Computer Science and Engineering

5CS4-24: Advance Java Lab

Credit: 1
L + 0T + 2P

Max. Marks: 50 (IA: 30, ETE:20)
End Term Exam: 2 Hours

SN	List of Experiments
1	Introduction To Swing, MVC Architecture, Applets, Application Look and Feel, Basic swing components : Text Fields, Buttons, Checkboxes, and Radio Buttons
2	Java database Programming, java.sql Package, JDBC Programming With java.net Package, Client and Server Program Protocol Handlers
3	RMI architecture, RMI registry, Writing distributed applications, Naming services, Naming And Directory Services, Overview of serialization and Internationalization
4	J2EE architecture, Enterprise application concepts, n-tier concepts, J2EE platform, HTTP protocol, web application, Web Application servers
5	Server side programming with Java Servlet, HTTP and Servlet, cycle, configuration and context, Request and Response handling and event handling, Introduction to filters with writing application

LAB PLAN

Total number of experiment 10

Total number of turns required 10

Number of turns required for

Experiment Number	Turns	Scheduled Day
Exp. 1	1	Day 1
Exp. 2	1	Day 2
Exp. 3	1	Day 3
Exp. 4	1	Day 4
Exp. 5	1	Day 5
Exp. 6	1	Day 6
Exp. 7	1	Day 7
Exp. 8	1	Day 8
Exp. 9	1	Day 9
Exp. 10	1	Day 10

Lab Objective & Requirements

Java Programming is intended for software engineers, systems analysts, program managers and user support personnel who wish to learn the Python programming language.

Prerequisites

Experience with a high level language (C/C++) is suggested. Prior knowledge of a concepts of core java programming and Object-Oriented concepts are mandatory.

Learning Objectives

The learning objectives of this course are:

- To understand why Java is a useful development language for developers.
- To learn how to design and program java applications.
- To learn how to use Applets, Swings, Java Data Base Programming, RMI, Sever Side Programming with Servlets, JSP.
- To learn how to use Naming and Directory Services.
- To learn how to use J2EE Architecture, HTTP Protocols, Web Applications.
- To learn how to use Servlet API, Request and Response Objects, Session and Event handling.
- To learn how to use JSP elements, fragments, Tag Extensions, SQL Tag and Function Library.

Software / Hardware Required

a. Software Requirement:

- Operating systems: Windows 7 or later OR Linux
- Eclipse IDE with Server setting

b. Hardware Requirement:

- Processors: or Intel® Core™ i3 processor
- 4 GB RAM or Higher
- Optical Mouse
- Keyboard

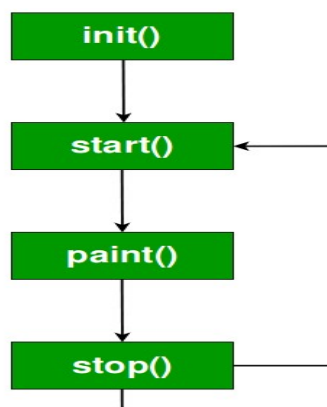
Java Applets

Applet is a Java program that can be embedded into a web page. It runs inside the web browser and works at client side. Applet is embedded in a HTML page using the APPLET or OBJECT tag and hosted on a web server. Applets are used to make the web site more dynamic and entertaining. (A platform dependent GUI toolkit)

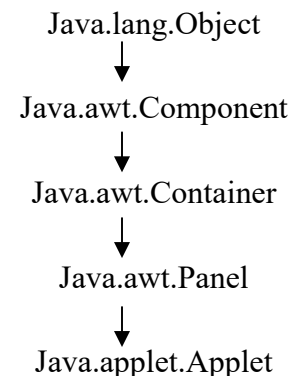
Some important points :

1. All applets are sub-classes (either directly or indirectly) of *java.applet.Applet* class.
2. Applets are not stand-alone programs. Instead, they run within either a web browser or an applet viewer. JDK provides a standard applet viewer tool called applet viewer.
3. In general, execution of an applet does not begin at main() method.
4. Output of an applet window is not performed by *System.out.println()*. Rather it is handled with various AWT methods, such as *drawString()*.

Life cycle of an applet :



Hierarchy of Applet Class :



It is important to understand the order in which the various methods shown in the above image are called. When an applet begins, the following methods are called, in this sequence:

1. `init()`
2. `start()`
3. `paint()`

When an applet is terminated, the following sequence of method calls takes place:

1. `stop()`
2. `destroy()`

Let's look more closely at these methods.

1. **init()** : The **init()** method is the first method to be called. This is where you should initialize variables. This method is called **only once** during the run time of your applet.
2. **start()** : The **start()** method is called after **init()**. It is also called to restart an applet after it has been stopped. Note that **init()** is called once i.e. when the first time an applet is loaded whereas **start()** is called each time an applet's HTML document is displayed onscreen. So, if a user leaves a web page and comes back, the applet resumes execution at **start()**.
3. **paint()** : The **paint()** method is called each time an AWT-based applet's output must be redrawn. This situation can occur for several reasons. For example, the window in which the applet is running may be overwritten by another window and then uncovered. Or the applet window may be minimized and then restored.
paint() is also called when the applet begins execution. Whatever the cause, whenever the applet must redraw its output, **paint()** is called.
The **paint()** method has one parameter of type **Graphics**. This parameter will contain the graphics context, which describes the graphics environment in which the applet is running. This context is used whenever output to the applet is required.
4. **stop()** : The **stop()** method is called when a web browser leaves the HTML document containing the applet—when it goes to another page, for example. When **stop()** is called, the applet is probably running. You should use **stop()** to suspend threads that don't need to run when the applet is not visible. You can restart them when **start()** is called if the user returns to the page.
5. **destroy()** : The **destroy()** method is called when the environment determines that your applet needs to be removed completely from memory. At this point, you should free up any resources the applet may be using. The **stop()** method is always called before **destroy()**.

Experiment 1 – Write a Program to create a Hello World applet.

Let's begin with the HelloWorld applet :

```
// A Hello World Applet
// Save file as HelloWorld.java

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

// HelloWorld class extends Applet
public class HelloWorld extends Applet
{
    // Overriding paint() method
    @Override
    public void paint(Graphics g)
    {
        g.drawString("Hello World", 20, 20);
    }
}
```

Experiment 2 – Write a Program to Set the color of the applet and draws a fill oval

```
import java.applet.Applet;
import java.awt.Color;
import java.awt.Graphics;
/*
    <applet code = "FillOval" width = 200 height = 200></applet>
*/
public class FillOval extends Applet {
    public void paint(Graphics g) {
        g.setColor(Color.red);
        g.fillOval(20,20,60,60);
    }
}
```

Java Swing

Java Swing is a lightweight Graphical User Interface (GUI) toolkit that includes a rich set of widgets. It includes package lets you make GUI components for your Java applications, and It is platform independent.

The Swing library is built on top of the Java Abstract Widget Toolkit (AWT), an older, platform dependent GUI toolkit. You can use the Java GUI components like button, textbox, etc. from the library and do not have to create the components from scratch.

Extensible: Swing is a highly modular-based architecture. Users can provide customized implementation(s) of the components to override the default implementations using Java's inheritance mechanism.

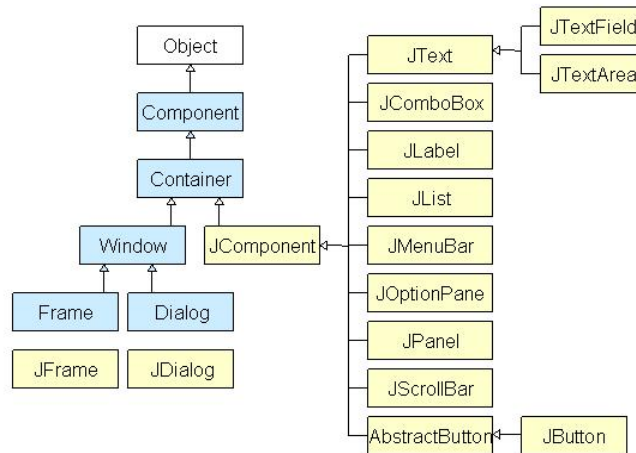
Component-based framework: The components are derived from the javax.swing. JComponent class. Swing components compliant with the Java Beans Component Architecture specifications. Swing objects asynchronously fire events, have bound properties, and respond to a documented set of methods specific to the component.

Configurable: Swing can respond to fundamental changes in its settings at run time.

Lightweight User Interface: Swing's high level of flexibility is reflected in its inherent ability to override the native host [operating system](#) (OS)'s GUI controls for displaying itself. A Swing component does not have a corresponding native OS GUI component, and is free to render itself in any way that is possible with the underlying graphics GUIs.

Loosely coupled and [Model/View/Controller](#) (MVC): The Swing library makes heavy use of the software [design pattern](#), which conceptually decouples the data being viewed from the user interface controls through which it is viewed. Therefore, Swing components have associated *models* that are specified in terms of Java [interfaces](#) and the programmers can use various default implementations or provide their own.

Java Swing class Hierarchy Diagram



All components in swing are JComponent which can be added to container classes.

What is a container class?

Container classes are classes that can have other components on it. So for creating a GUI, we need at least one container object. There are 3 types of containers.

1. **Panel:** It is a pure container and is not a window in itself. The sole purpose of a Panel is to organize the components on to a window.
2. **Frame:** It is a fully functioning window with its title and icons.
3. **Dialog:** It can be thought of like a pop-up window that pops out when a message has to be displayed. It is not a fully functioning window like the Frame.

Experiment 3 – Write a Program to design GUI in Java

Step 1) Copy the following code into an editor

```
import javax.swing.*;
4. class gui{
5.     public static void main(String args[]){
6.         JFrame frame = new JFrame("My First GUI");
7.         frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
8.         frame.setSize(300,300);
9.         JButton button = new JButton("Press");
10.        frame.getContentPane().add(button); // Adds Button to content pane of frame
11.        frame.setVisible(true);
12.    }
13. }
```

Step 2) Save, Compile, and Run the code.

Step 3) Now let's Add a Button to our frame. Copy following code into an editor

Step 4) Execute the code. You will get a big button



Java Layout Manger

The Layout manager is used to layout (or arrange) the GUI java components inside a container. There are many layout managers, but the most frequently used are-

JavaBorderLayout

A **BorderLayout** places components in up to five areas: top, bottom, left, right, and center. It is the default layout manager for every java JFrame.



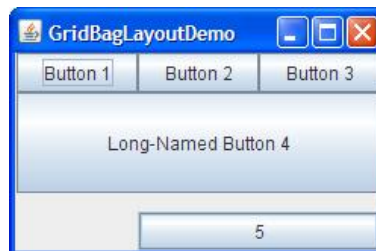
Java FlowLayout

FlowLayout is the default layout manager for every **JPanel**. It simply lays out components in a single row one after the other.



Java GridBagLayout

It is the more sophisticated of all layouts. It aligns components by placing them within a grid of cells, allowing components to span more than one cell.



Experiment 4 : WAP to make a calculator using Java Swing.

```
package calculator;

public class form1 extends javax.swing.JFrame {

    int a;

    int count=0;

    public form1() {
        initComponents();
    }

    private void jButton1ActionPerformed(java.awt.event.ActionEvent evt) {
        jTextField1.setText(jTextField1.getText()+"1");
    }

    private void jButton2ActionPerformed(java.awt.event.ActionEvent evt) {
        jTextField1.setText(jTextField1.getText()+"2");
    }

    private void jButton3ActionPerformed(java.awt.event.ActionEvent evt) {
        jTextField1.setText(jTextField1.getText()+"3");
    }

    private void jButton4ActionPerformed(java.awt.event.ActionEvent evt) {
        jTextField1.setText(jTextField1.getText()+"4");
    }

    private void jButton5ActionPerformed(java.awt.event.ActionEvent evt) {
        jTextField1.setText(jTextField1.getText()+"5");
    }

    private void jButton6ActionPerformed(java.awt.event.ActionEvent evt) {
        jTextField1.setText(jTextField1.getText()+"6");
    }

    private void jButton7ActionPerformed(java.awt.event.ActionEvent evt) {
        jTextField1.setText(jTextField1.getText()+"7");
    }
}
```

```

private void jButton8ActionPerformed(java.awt.event.ActionEvent evt) {
jTextField1.setText(jTextField1.getText()+"8");
}
private void jButton9ActionPerformed(java.awt.event.ActionEvent evt) {
jTextField1.setText(jTextField1.getText()+"9");
}
private void jButton18ActionPerformed(java.awt.event.ActionEvent evt) {
jTextField1.setText(jTextField1.getText()+"0");
}
private void jButton10ActionPerformed(java.awt.event.ActionEvent evt) {
    a=Integer.parseInt(jTextField1.getText());
    jTextField1.setText("");
    count=1;
}
private void jButton11ActionPerformed(java.awt.event.ActionEvent evt) {
    a=Integer.parseInt(jTextField1.getText());
    jTextField1.setText("");
    count=2;
}
private void jButton12ActionPerformed(java.awt.event.ActionEvent evt) {
    a=Integer.parseInt(jTextField1.getText());
    jTextField1.setText("");
    count=3;
}
private void jButton13ActionPerformed(java.awt.event.ActionEvent evt) {
    a=Integer.parseInt(jTextField1.getText());
    jTextField1.setText("");
    count=4;
}
private void jButton14ActionPerformed(java.awt.event.ActionEvent evt) {

```

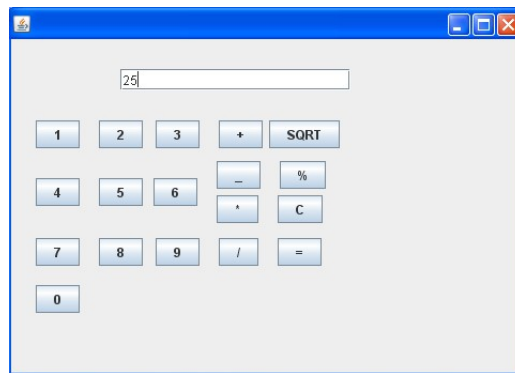
```

        a=Integer.parseInt(jTextField1.getText());
        jTextField1.setText("");
        count=5;
    }
    private void jButton15ActionPerformed(java.awt.event.ActionEvent evt) {
        a=Integer.parseInt(jTextField1.getText());
        jTextField1.setText("");
        count=6;
    }
    private void jButton17ActionPerformed(java.awt.event.ActionEvent evt) {
        int b=Integer.parseInt(jTextField1.getText());
        int c=0;
        if(count==1)
            c=a+b;
        else if(count==2)
            c=a-b;
        else if(count==3)
            c=a*b;
        else if(count==4)
            c=a/b;
        else if(count==5)
            c=a%b;
        else if(count==6)
            c=a*b;
        jTextField1.setText(Integer.toString(c));
    }
    private void jButton16ActionPerformed(java.awt.event.ActionEvent evt) {
        jTextField1.setText("");
    }

```

```
public static void main(String args[]) {  
    java.awt.EventQueue.invokeLater(new Runnable() {  
        public void run() {  
            new form1().setVisible(true); }  
    });  
}
```

OUTPUT



Java Database Programming

Objectives

- To understand the concept of database and database management systems
- To understand the relational data model: relational data structures constraints, and languages
- To use SQL to create and drop tables, and to retrieve and modify data
- To learn how to load a driver, connect to a database, execute statements, and process result sets using JDBC
- To use prepared statements to execute precompiled SQL statements
- To use callable statements to execute stored SQL procedures and functions
- To explore database metadata using the DatabaseMetaData and ResultSetMetaData interfaces

Why Java for Database Programming?

First, Java is platform independent. You can develop platform-independent database applications using SQL and Java for any relational database systems.

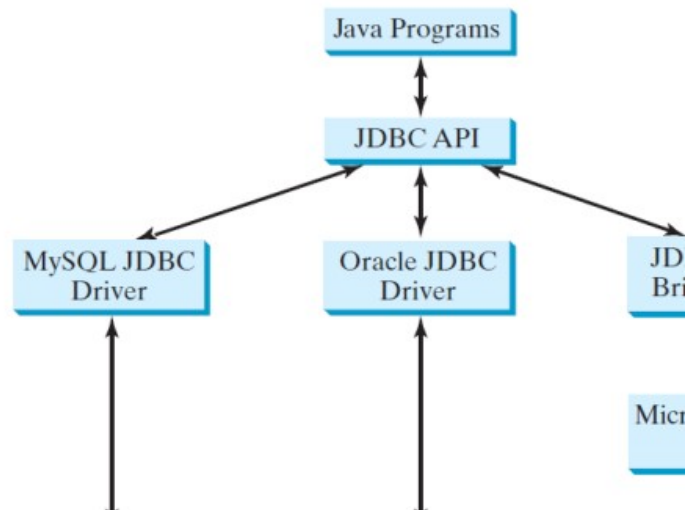
Second, the support for accessing database systems from Java is built into Java API, so you can create database applications using all Java code with a common interface.

Third, Java is taught in almost every university either as the first programming language or as the second programming language.

Database Applications Using Java:

- GUI
- Client/Server
- Server-Side programming

The Architecture of JE



The execute, executeQuery, and executeUpdate Methods

The methods for executing SQL statements are `execute`, `executeQuery`, and `executeUpdate`, each of which accepts a string containing a SQL statement as an argument. This string is passed to the database for execution. The `execute` method should be used if the execution produces multiple result sets, multiple update counts, or a combination of result sets and update counts. The `executeQuery` method should be used if the execution produces a single result set, such as the SQL select statement. The `executeUpdate` method should be used if the statement results in a single update count or no update count, such as a SQL INSERT, DELETE, UPDATE, or DDL statement.

JDBC Drivers

Java Database Connectivity (JDBC) is an application programming interface (API) for the programming language Java, which defines how a client may access any kind of tabular data, especially relational database. It is part of Java Standard Edition platform, from Oracle Corporation. It acts as a middle layer interface between java applications and database.

The JDBC classes are contained in the Java Package **java.sql** and **javax.sql**. JDBC helps you to write Java applications that manage these three programming activities:

1. Connect to a data source, like a database.
2. Send queries and update statements to the database
3. Retrieve and process the results received from the database in answer to your query

JDBC drivers are client-side adapters (installed on the client machine, not on the server) that convert requests from Java programs to a protocol that the DBMS can understand. There are 4 types of JDBC drivers:

1. Type-1 driver or JDBC-ODBC bridge driver
2. Type-2 driver or Native-API driver
3. Type-3 driver or Network Protocol driver
4. Type-4 driver or Thin driver

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 the 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 the port number to connect to.
- The constructor of the `Socket` class attempts to connect the client to the specified server and the 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, hence data can be sent across both streams at the same time. Following are the 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. The ServerSocket class has four constructors –

Method & Description

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.

public ServerSocket(int port, int backlog) throws IOException

Similar to the previous constructor, the backlog parameter specifies how many incoming clients to wait in a queue.

public ServerSocket(int port, int backlog, InetAddress address) throws IOException

Similar to the previous constructor, the InetAddress parameter specifies the local IP address to bind to. The InetAddress is used for servers that may have multiple IP addresses, allowing the server to specify which of its IP addresses to accept client requests on.

public ServerSocket() throws IOException

Creates an unbound server socket. When using this constructor, use the bind() method when you are ready to bind the server socket.

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.

Following are some of the common methods of the ServerSocket class –

Method & Description

public int getLocalPort()

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.

public Socket accept() throws IOException

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 setTimeout() method. Otherwise, this method blocks indefinitely.

public void setSoTimeout(int timeout)

Sets the time-out value for how long the server socket waits for a client during the accept().

public void bind(SocketAddress host, int backlog)

Binds the socket to the specified server and port in the SocketAddress object. Use this method if you instantiated the ServerSocket 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 the server, and communication can begin.

Socket Class Methods

The **java.net.Socket** class represents the socket that both the client and the 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 –

Method & Description**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.

public Socket(InetAddress host, int port) throws IOException

This method is identical to the previous constructor, except that the host is denoted by an InetAddress object.

public Socket(String host, int port, InetAddress localAddress, int localPort) throws IOException

Connects to the specified host and port, creating a socket on the local host at the specified address and port.

public Socket(InetAddress host, int port, InetAddress localAddress, int localPort) throws IOException.

This method is identical to the previous constructor, except that the host is denoted by an `InetAddress` object instead of a `String`.

`public Socket()`

Creates an unconnected socket. Use the `connect()` method to connect this socket to a 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 the server have a `Socket` object, so these methods can be invoked by both the client and the server.

Method & Description

`public void connect(SocketAddress host, int timeout) throws IOException`

This method connects the socket to the specified host. This method is needed only when you instantiate `Socket` using the no-argument constructor.

`public InetAddress getInetAddress()`

This method returns the address of the other computer that this socket is connected to.

`public int getPort()`

Returns the port the socket is bound to on the remote machine.

`public int getLocalPort()`

Returns the port the socket is bound to on the local machine.

`public SocketAddress getRemoteSocketAddress()`

Returns the address of the remote socket.

`public InputStream getInputStream() throws IOException`

Returns the input stream of the socket. The input stream is connected to the output stream of the remote socket.

`public OutputStream getOutputStream() throws IOException`

Returns the output stream of the socket. The output stream is connected to the input stream of the remote socket.

socket.

public void close() throws IOException

Closes the socket, which makes this Socket object no longer capable of connecting again to any server.

InetAddress Class Methods

This class represents an Internet Protocol (IP) address. Here are following usefull methods which you would need while doing socket programming –

Method & Description

static InetAddress getByAddress(byte[] addr)

Returns an InetAddress object given the raw IP address.

static InetAddress getByAddress(String host, byte[] addr)

Creates an InetAddress based on the provided host name and IP address.

static InetAddress getByName(String host)

Determines the IP address of a host, given the host's name.

String getHostAddress()

Returns the IP address string in textual presentation.

String getHostName()

Gets the host name for this IP address.

static InetAddress InetAddress getLocalHost()

Returns the local host.

String toString()

Converts this IP address to a String.

Experiment 5 – Write a Program to implement the SQL –login ID commands using JDBC

```
package javaapplication5;
import java.sql.*;
import java.awt.*;
import javax.swing.*;
public class NewJFrame extends javax.swing.JFrame {
    public NewJFrame() {
        initComponents();
    }
    private void initComponents() {
        jTextField1 = new javax.swing.JTextField();
        jTextField2 = new javax.swing.JTextField();
        jButton1 = new javax.swing.JButton();
        jLabel1 = new javax.swing.JLabel();
        jLabel2 = new javax.swing.JLabel();
        setDefaultCloseOperation(javax.swing.WindowConstants.EXIT_ON_CLOSE);
        jButton1.setText("Login");
        jButton1.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                jButton1ActionPerformed(evt);
            }
        });
    }
    private void jButton1ActionPerformed(java.awt.event.ActionEvent evt) {
        try
        {
            //      Connection conn;
            Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
            String url = "Jdbc:Odbc:g2";
            Connection conn = DriverManager.getConnection(url);
            ResultSet rs;
            Statement stmt=conn.createStatement();
            rs=stmt.executeQuery("select * from mytab");

            while(rs.next())
            {
                if(((jTextField1.getText()).equals(rs.getString(1)))&& ((jTextField2.getText()).equals(rs.getString(2)))){
                    JOptionPane.showMessageDialog(this, "login successfull");
                    System.exit(0);
                }
            }
        }
        catch (Exception e)
        {
            JOptionPane.showMessageDialog(this, e.getMessage());
        }
    }
}
```

```

}}
    JOptionPane.showMessageDialog(this, "login unsuccessful");
    System.exit(0);
    conn.close();
}
catch (Exception e) {
    System.err.println("Got an exception! ");
    System.err.println(e.getMessage());
}
}
}

```

OUTPUT:



The image shows a Java Swing window titled "Login". It has a light gray background and a blue border. Inside the window, there are two labels: "User Name" and "Password". To the right of each label is a text input field. Below these fields is a button labeled "Login". The window is centered on the screen.

Experiment 6 – Write a Program to implement the SQL commands using JDBC.

```
package opertiondemo;
import java.sql.*;
import javax.swing.*;
public class operationdemo1 extends javax.swing.JFrame {
    ResultSet rs1;
    /** Creates new form operationdemo1 */
    public operationdemo1() {
        initComponents();

        try
        {
            Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
            String url = "Jdbc:Odbc:g4";
            Connection conn = DriverManager.getConnection(url);

            Statement
            stmt=conn.createStatement(ResultSet.TYPE_SCROLL_SENSITIVE,ResultSet.CONCUR_UPDATABLE
            );

            rs1=stmt.executeQuery("select * from student");

        }
        catch (Exception e) {
            System.err.println("Got an exception! ");
            System.err.println(e.getMessage());
        }
    }
    private void jButton1ActionPerformed(java.awt.event.ActionEvent evt) {
        try
        {System.out.println("Outside rs1");
            if(rs1.next()){
                System.out.println("Inside rs1");
                String r1=rs1.getString(1);
                String n1=rs1.getString(2);
                String a1=rs1.getString(3);
```



```

        jTextField1.setText(r1);
        jTextField2.setText(n1);
        jTextField3.setText(a1);
    }
    else
    {
        JOptionPane.showMessageDialog(this, "eND OF THE RECORD");
    }
} catch (Exception e)
{}
}

private void jButton3ActionPerformed(java.awt.event.ActionEvent evt) {
    try
    {
        Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
        String url = "Jdbc:Odbc:g4";
        Connection conn = DriverManager.getConnection(url);
        ResultSet rs;
        Statement stmt=conn.createStatement();
        String qry=("Insert into student
values(""+jTextField1.getText()+"','"+jTextField2.getText()+"','"+jTextField3.getText()+"");
        stmt.executeUpdate(qry);
        JOptionPane.showMessageDialog(this, "Record inserted");

        conn.close();
    }
    catch (Exception e) {
        System.err.println("Got an exception! ");
        System.err.println(e.getMessage());
    }
}

private void jButton7ActionPerformed(java.awt.event.ActionEvent evt) {
    try
    {System.out.println("Outside rs1");
        if(rs1.previous()){
            System.out.println("Inside rs1");
            String r1=rs1.getString(1);
            String n1=rs1.getString(2);
            String a1=rs1.getString(3);
            jTextField1.setText(r1);

```

```

        jTextField2.setText(n1);
        jTextField3.setText(a1);
    }
} catch (Exception e)
{}
}

private void jButton5ActionPerformed(java.awt.event.ActionEvent evt) {
// TODO add your handling code here:
    try
    {
//        Connection conn;
        Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
        String url = "Jdbc:Odbc:g4";
        Connection conn = DriverManager.getConnection(url);
        ResultSet rs;
        Statement
stmt=conn.createStatement(ResultSet.TYPE_SCROLL_SENSITIVE,ResultSet.CONCUR_UPDATABLE
);
        rs=stmt.executeQuery("select * from student");
        rs.first();
        String r=rs.getString(1);
        String n=rs.getString(2);
        String a=rs.getString(3);
        jTextField1.setText(r);
        jTextField2.setText(n);
        jTextField3.setText(a);
        conn.close();
    }
    catch (Exception e) {
        System.err.println("Got an exception! ");
        System.err.println(e.getMessage());
    }

}

private void jButton6ActionPerformed(java.awt.event.ActionEvent evt) {
// TODO add your handling code here:

    try
    {
//        Connection conn;

```

```

        Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
        String url = "Jdbc:Odbc:g4";
        Connection conn = DriverManager.getConnection(url);

        ResultSet rs;

        Statement
stmt=conn.createStatement(ResultSet.TYPE_SCROLL_SENSITIVE,ResultSet.CONCUR_UPDATABLE
);
        rs=stmt.executeQuery("select * from student");

        rs.last();
        String r=rs.getString(1);
        String n=rs.getString(2);
        String a=rs.getString(3);
        jTextField1.setText(r);
        jTextField2.setText(n);
        jTextField3.setText(a);
        conn.close();
    }
    catch (Exception e) {
        System.err.println("Got an exception! ");
        System.err.println(e.getMessage());
    }
}

private void jButton4ActionPerformed(java.awt.event.ActionEvent evt) {
// TODO add your handling code here:

    try
    {
//        Connection conn;
        Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
        String url = "Jdbc:Odbc:g4";
        Connection conn = DriverManager.getConnection(url);

        ResultSet rs;

```

```

        String qry="update student set rollnum="+jTextField1.getText()+"age="+jTextField3.getText()+"
where name='"+jTextField2.getText()+"'";
        Statement stmt=conn.createStatement();
        stmt.executeUpdate(qry);
        JOptionPane.showMessageDialog(this, "Record Updated");

        conn.close();
    }
    catch (Exception e) {
        System.err.println("Got an exception! ");
        System.err.println(e.getMessage());
    }
}

private void jButton2ActionPerformed(java.awt.event.ActionEvent evt) {
// TODO add your handling code here:

    try
    {
//        Connection conn;
        Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
        String url = "Jdbc:Odbc:g4";
        Connection conn = DriverManager.getConnection(url);

        Statement stmt=conn.createStatement();
        stmt.executeUpdate("delete * from student where rollnum="+jTextField1.getText()+"");
        JOptionPane.showMessageDialog(this, "Record deleted");

        jTextField1.setText(" ");
        jTextField2.setText(" ");
        jTextField3.setText(" ");
        conn.close();
    }
    catch (Exception e) {
        System.err.println("Got an exception! ");
        System.err.println(e.getMessage());
    }
}

```

```

private void jButton8ActionPerformed(java.awt.event.ActionEvent evt) {
    System.exit(0);
    // TODO add your handling code here:
}

private void jButton9ActionPerformed(java.awt.event.ActionEvent evt) {
    jTextField1.setText(" ");
    jTextField2.setText(" ");
    jTextField3.setText(" ");
    // TODO add your handling code here:
}

private void jButton10ActionPerformed(java.awt.event.ActionEvent evt) {
    // TODO add your handling code here:

    try
    {
        //      Connection conn;
        Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
        String url = "Jdbc:Odbc:g4";
        Connection conn = DriverManager.getConnection(url);

        ResultSet rs;

        Statement stmt=conn.createStatement();
        rs=stmt.executeQuery("Select * from student");
        while(rs.next()){
            String r=rs.getString(1);
            String n=rs.getString(2);
            String a=rs.getString(3);
            jTextField1.setText(r);
            jTextField2.setText(n);
            jTextField3.setText(a);
        }
        conn.close();
    }
    catch (Exception e) {
        System.err.println("Got an exception! ");
        System.err.println(e.getMessage());
    }
}

```

}}

OUTPUT:

rollnum 4

name kirti

age 45

first next Delete Insert Update

last previous Select Exit Reset

Java 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. Let's understand the stub and skeleton objects:

Stub

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

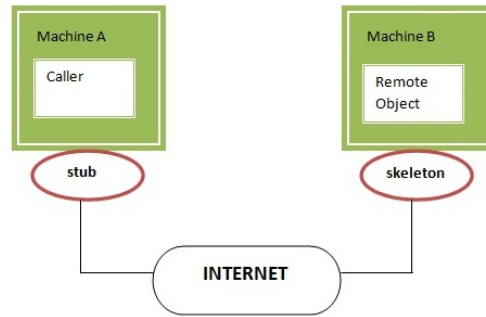
1. It initiates a connection with remote Virtual Machine (JVM),
2. It writes and transmits (marshals) the parameters to the remote Virtual Machine (JVM),
3. It waits for the result
4. It reads (unmarshals) the return value or exception, and
5. It finally, returns the value to the caller.

Skeleton

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

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

In the Java 2 SDK, an stub protocol was introduced that eliminates the need for skeletons.



Understanding requirements for the distributed applications

If any application performs these tasks, it can be distributed application.

The application need to locate the remote method

1. It need to provide the communication with the remote objects, and
2. The application need to load the class definitions for the objects.

The RMI application have all these features, so it is called the distributed application.

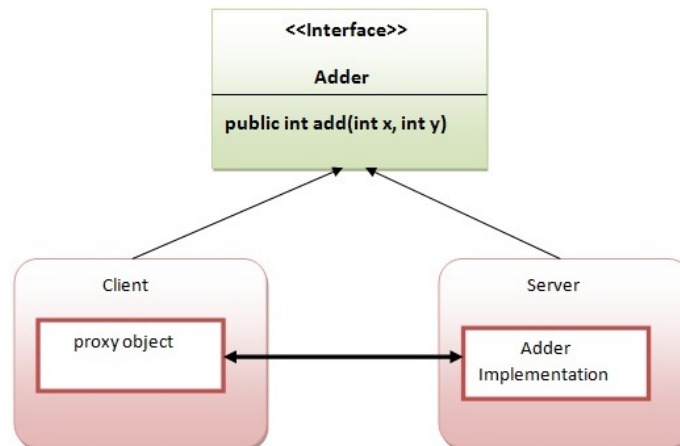
Java RMI Example

The is given the 6 steps to write the RMI program.

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

RMI Example

In this example, we have followed all the 6 steps to create and run the rmi application. The client application need only two files, remote interface and client application. In the rmi application, both client and server interacts with the remote interface. The client application invokes methods on the proxy object, RMI sends the request to the remote JVM. The return value is sent back to the proxy object and then to the client application.



1. create the remote interface

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

1. import java.rmi.*;
2. public interface Adder extends Remote{
3. public int add(int x,int y)throws RemoteException;
4. }

2) Provide the implementation of the remote interface

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

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

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

1. import java.rmi.*;
2. import java.rmi.server.*;
3. public class AdderRemote extends UnicastRemoteObject implements Adder{
4. AdderRemote()throws RemoteException{
5. super();
6. }
7. public int add(int x,int y){return x+y;}
8. }

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

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

```
rmic AdderRemote
```

4) Start the registry service by the rmiregistry tool

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

```
rmiregistry 5000
```

5) Create and run the server application

In this example, we are binding the remote object by the name sonoo.

```
1. import java.rmi.*;
2. import java.rmi.registry.*;
3. public class MyServer{
4.     public static void main(String args[]){
5.         try{
6.             Adder stub=new AdderRemote();
7.             Naming.rebind("rmi://localhost:5000/sonoo",stub);
8.         }catch(Exception e){System.out.println(e);}
9.     }
10. }
```

6) Create and run the client application

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

```
11. import java.rmi.*;
12. public class MyClient{
13.     public static void main(String args[]){
14.         try{
15.             Adder stub=(Adder)Naming.lookup("rmi://localhost:5000/sonoo");
16.             System.out.println(stub.add(34,4));
17.         }catch(Exception e){}
18.     }
```

19. }

For running this rmi example,

1) compile all the java files

```
javac *.java
```

2) create stub and skeleton object by rmic tool

```
rmic AdderRemote
```

3) start rmi registry in one command prompt

```
rmiregistry 5000
```

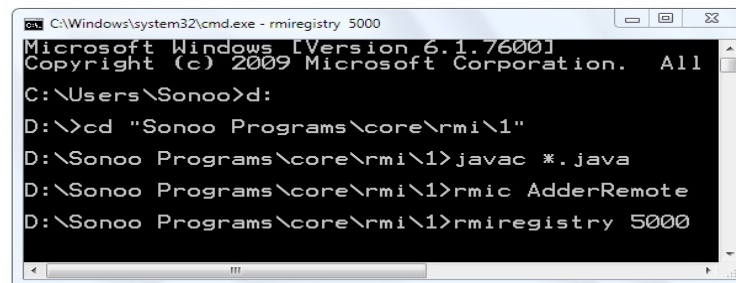
4) start the server in another command prompt

```
java MyServer
```

5) start the client application in another command prompt

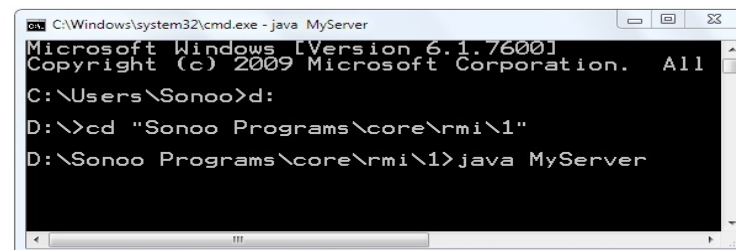
```
java MyClient
```

Output of this RMI example



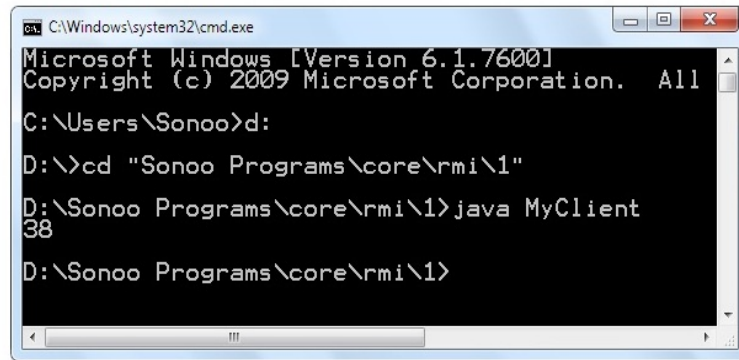
```
C:\Windows\system32\cmd.exe - rmiregistry 5000
Microsoft Windows [Version 6.1.7600]
Copyright (c) 2009 Microsoft Corporation. All rights reserved.

C:\Users\Sonoo>d:
D:\>cd "Sonoo Programs\core\rmi\1"
D:\Sonoo Programs\core\rmi\1>javac *.java
D:\Sonoo Programs\core\rmi\1>rmic AdderRemote
D:\Sonoo Programs\core\rmi\1>rmiregistry 5000
```



```
C:\Windows\system32\cmd.exe - java MyServer
Microsoft Windows [Version 6.1.7600]
Copyright (c) 2009 Microsoft Corporation. All rights reserved.

C:\Users\Sonoo>d:
D:\>cd "Sonoo Programs\core\rmi\1"
D:\Sonoo Programs\core\rmi\1>java MyServer
```



```
C:\Windows\system32\cmd.exe
Microsoft Windows [Version 6.1.7600]
Copyright (c) 2009 Microsoft Corporation. All rights reserved.

C:\Users\Sonoo>d:
D:\>cd "Sonoo Programs\core\rmi\1"
D:\Sonoo Programs\core\rmi\1>java MyClient
38
D:\Sonoo Programs\core\rmi\1>
```

Experiment 7 – Write a Program to implement the Remote Method Invocation.

INTERFACE:

```
import java.rmi.*;
interface Bank extends Remote
{
double getAmount(double p,double t) throws RemoteException;
}
```

BANK SEVER:

```
import java.rmi.*;
import java.rmi.server;
public class BankImpl extends UnicastRemoteObject implements Bank
{
public BankImpl throws RemoteException
{

}
double getAmount(double p,double t) throws RemoteException
{
return p*Math.pow(1.41,t);
}
}
```

RMI REGISTRY:

```
import java.rmi.*;
import javax.naming.*;
public class BankServer
{
public static void main(String args[])
{
BankImpl centralbank=new BankImpl();
Naming.rebind("uti",centralbank);
}
}
```

BANKCLIENT:

```
import java.rmi.*;
import javax.naming.*;
public class Bankclient
{
public static void main(String args[]) throws RemoteException
{
String url="rmi://localhost//uti";
Bank b=(Bank) Naming.lookup(url);
System.out.println(b.getAmount(4000,3));
}
}
```

JSP

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.

Advantages of JSP over Servlet:

There are many advantages of JSP over the Servlet. They are as follows:

- JSP technology is the extension to Servlet technology. We can use all the features of the Servlet in JSP. In addition to, we can use implicit objects, predefined tags, expression language and Custom tags in JSP, that makes JSP development easy.
- JSP can be easily managed because we can easily separate our business logic with presentation logic. In Servlet technology, we mix our business logic with the presentation logic.
- If JSP page is modified, we don't need to recompile and redeploy the project. The Servlet code needs to be updated and recompiled if we have to change the look and feel of the application.
- In JSP, we can use many tags such as action tags, JSTL, custom tags, etc. that reduces the code. Moreover, we can use EL, implicit objects, etc.

The Lifecycle of a JSP Page

The JSP pages follow these phases:

- Translation of JSP Page
- Compilation of JSP Page
- Classloading (the classloader loads class file)
- Instantiation (Object of the Generated Servlet is created).
- Initialization (the container invokes `jspInit()` method).
- Request processing (the container invokes `_jspService()` method).
- Destroy (the container invokes `jspDestroy()` method).

Experiment 8 – Write a Program to demonstrate Basic Servlet.

Program

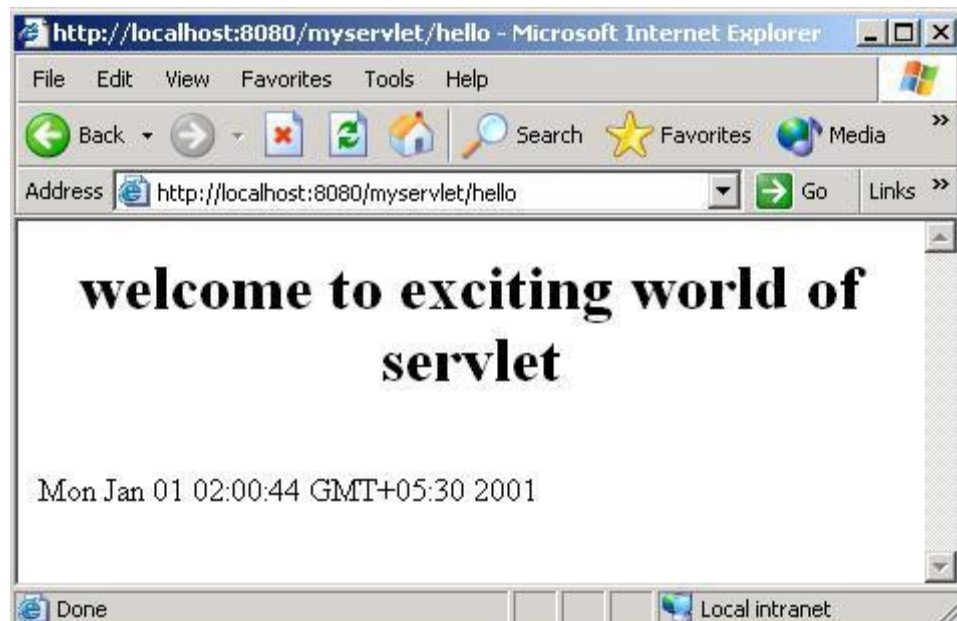
```
Import java.io.IOException;
import java.io.PrintWriter;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
public class hello extends HttpServlet {
private static final long serialVersionUID = 1L;
public void doGet(HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException {
PrintWriter out=response.getWriter(); java.util.Date today=new
java.util.Date(); out.println("<html>"+<body>"+<h1
align=center>welcome to
exciting world of servlet</h1>"+<br>"+today+"</body>"+</html>");
}
}
```

Step to Run the Program

- 1) Remove all the errors in the program.
- 2) Start the web server.
- 3) Deploy your program to the web server
- 4) Open the web browser and execute the program as

http://localhost:8080/myservlet/hello

Output:



Experiment 9 – Write a *Basic JSP program*

Pr ogr am :

```
<%@ page language="java" contentType="text/html; charset=ISO88591"
pageEncoding="ISO88591"%>
```

```
<%@page language="java" %>
```

```
<!DOCTYPE html PUBLIC "-//
```

```
W3C//DTD HTML 4.01 Transitional//EN"
```

```
"http://www.w3.org/TR/html4/loose.dtd">
```

```
<html>
```

```
<head>
```

```
<meta httpequiv="
```

```
ContentType"
```

```
content="text/html; charset=ISO88591">
```

```
<title>this is jsp1</title>
```

```
</head>
```

```
<body bgcolor="red">
```

```
<font size="10">
```

```
<%
```

```
String name="roseindia.net";
```

```
out.println("hello" + name + "!");
```

```
%>
```

```
</font>
```

```
</body>
```

```
</html>
```

Step to R un the Pr ogr am :

- 1) Remove all the errors in the program.
- 2) Start the web server.

3) Deploy your program to the web server

20

4) Open the web browser and execute the program as

`http://localhost:8080/secondjsp/jsp1.jsp`

Output:



Experiment 10 – Write a Program *for Database Operation in jsp*

Create a database: First create a database named 'student' in mysql and table named "stu_info" in same database by sql query given below:

Create database student

```
create table stu_info (  
ID int not null auto_increment,  
Name varchar(20),  
City varchar(20),  
Phone varchar(15),  
primary key(ID)  
);
```

Create a new directory named "user" in the tomcat6.0.16/

webapps and

WEBINF

directory in same directory. Before running this java code you

need to paste a .jar file named mysql connector.jar in the Tomcat6.0.16/

webapps/user/WEBINF/

lib.

prepared_statement_query.jsp

Save this code as a .jsp file named "**prepared_statement_query.jsp**" in the directory Tomcat6.0.16/

webapps/user/ and you can run this jsp page with

url **http://localhost:8080/user/prepared_statement_query.jsp** in address bar of the browser

```
<!DOCTYPE HTML PUBLIC "-//  
W3C//DTD HTML 4.01  
Transitional//EN"
```

```

"http://www.w3.org/TR/html4/loose.dtd" >
<%@ page import="java.sql.*" %>
<%@ page import="java.io.*" %>
<HTML>
<HEAD>
<TITLE>insert data using prepared statement </TITLE>
</HEAD>
<BODY bgcolor="#ffffcc">
<font size="+3" color="green"><br>Welcome in www.roseindia.net
!</font>
<FORM action="prepared_statement_query.jsp" method="get">
<TABLE style="backgroundcolor:
#ECE5B6;" WIDTH="30%" >
<TR>
<TH width="50%"> Name</TH>
<TD width="50%"><INPUT TYPE="text" NAME="name"></TD>
</tr>
<TR>
<TH width="50%">City</TH>
<TD width="50%"><INPUT TYPE="text" NAME="city"></TD>
</tr>
<TR>
<TH width="50%">Phone</TH>
<TD width="50%"><INPUT TYPE="text" NAME="phone"></TD>
</tr>
<TR>
<TH></TH>
<TD width="50%">< INPUT TYPE="submit" VALUE="submit"></TD>
</tr>
</TABLE>

```

```

<%
String name = request.getParameter ("name");
String city = request.getParameter ("city");
String phone = request.getParameter ("phone");
String connectionURL = "jdbc: mysql:
Connection connection = null;
PreparedStatement pstatement = null;
Class.forName("com.mysql.jdbc.Driver").newInstance();
int updateQuery = 0;
if(name!=null && city!=null && phone!=null){
if(name!="" && city!="" && phone!="") {
try {
connection = DriverManager.getConnection
(connectionURL, "root", "root");
String queryString = "INSERT INTO stu_info(Name,
Address, Phone) VALUES (?, ?, ?)";
pstatement = connection.prepareStatement(queryString);
pstatement.setString(1, name);
pstatement.setString(2, city);
pstatement.setString(3, phone);
updateQuery = pstatement.executeUpdate();
if(updateQuery != 0) { %>
<br>
<TABLE style="backgroundcolor:
#E3E4FA;"
WIDTH="30%" border="1">
<tr><th>Data is inserted successfully
in database.</th></tr>
</table>
<%

```

```

    }
    }
    catch (Exception ex) {
        out.println("Unable to connect to database.");
    }
    finally {
        .
        pstatement.close();
        connection.close();
    }
    }
    }
    %>
</FORM>
</body>
</html>

```

Output:

Fill all the fields and click on submit button, that shows a response message. If any field is blank or only blank spaces are there page will remain same after clicking on submit button.

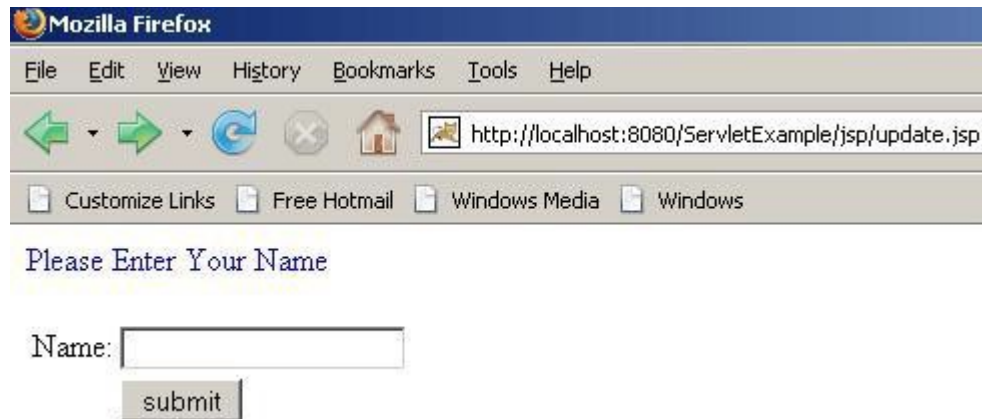


Update Data:

This example shows how to update the existing record of mysql table using jdbc connectivity in the jsp page. In this example we have created two jsp pages **update.jsp** and **updatingDatabase.jsp**. In the **update.jsp** page, we are using a Text box where user can give his/her name and submit the page. After submitting the page, **updatingDatabase.jsp** will be called and the sql query ("**update servlet set name = ? where id = ?**") is executed which will modify the table record.

```
<%@page language="java" session="true"
contentType="text/html;charset= %">
<font color="blue"> Please Enter Your Name </font><br><br>
<form name="frm" method="post" action="updatingDatabase.jsp">
<table border = "0">
<tr align="left" valign="top">
<td>Name:</td>
<td><input type="text" name = "name" /></td>
</tr>
<tr align="left" valign="top">
<td></td>
<td><input type="submit" name="submit" value="submit"/></td>
</tr>
</table>
</form>
```

Type the url: **http://localhost:8080/ServletExample/jsp/update.jsp** on your browser. Following output will be displayed:



updatingDatabase.jsp

```
<%@ page language = "java" contentType =  
"text/html; charset = ISO88591"  
import = "java.io.*"  
import = "java.sql.*"  
import = "java.util.*"  
import = "javax.sql.*"  
import = "java.sql.ResultSet"  
import = "java.sql.Statement"  
import = "java.sql.Connection"  
import = "java.sql.DriverManager"  
import = "java.sql.SQLException"  
%>  
<%  
  
Connection con = null;  
PreparedStatement ps = null;  
ResultSet rs = null;  
Statement stmt = null;
```

```

String name = request.getParameter("name");
Integer id = 5;
%>
<html>
<head>
<title>Updating Database</title>
</head>
<body>
<%
try {
Class.forName("com.mysql.jdbc.Driver");
con =DriverManager.getConnection
("jdbc:mysql://192.168.10.59:3306/example",
"root", "root");
ps = con.prepareStatement("update servlet set
name = ? where id = ?");
ps.setInt(2, id);
ps.setString(1, name);
ps.executeUpdate();
%>
Database successfully Updated!<br>
<%
if(ps.executeUpdate())>=1){
28
stmt=con.createStatement();
rs = stmt.executeQuery("SELECT * FROM servlet");
while(rs.next()){
%>
<%=rs.getObject(1).toString()%>
<%=("\t\t\t")%>

```

```

<%=rs.getObject(2).toString()%>
<%=("<br>")%>
<%
}
}
} catch (IOException e) {
throw new IOException("Can not display records.", e);
} catch (ClassNotFoundException e) {
throw new SQLException("JDBC Driver not found.", e);
} finally {
try {
if(stmt != null){
stmt.close();
stmt = null;
}
if(ps != null) {
ps.close();
ps = null;
}
if(con != null) {
con.close();
con = null;
}
} catch (SQLException e) {}
}
%>
</body>
</html>

```

After submitting the name, it will be updated in the database and the records will be displayed on your browser.

Please Enter Your Name

Name:

Database successfully Updated!

1 sandeep
2 amit
3 anusmita
4 vineet
5 roseindia

Save this code as a .jsp file named "**prepared_statement_query.jsp**" in the directory Tomcat6.0.16/webapps/user/ and you can run this jsp page with url **http://localhost:8080/user/prepared_statement_query.jsp** in address bar of the browser.

Conclusion:

Hence we studied how to perform different database operation in jsp.

Multiple Choice Questions

1) What technique is used for the authentication mechanism in the servlet specification?

- a. Role Based Authentication
- b. Form Based Authentication
- c. Both A & B
- d. None of the above

2) Which attribute specifies a JSP page that should process any exceptions thrown but not caught in the current page?

- a. The `ErrorMessage` Attribute
- b. The `IsErrorMessage` Attribute
- c. Both A & B
- d. None of the above

3) Which Error Handling in Java handles runtime errors with exceptions, If an exception is not caught in your JSP or Servlet, Resin will use a special error page to send results back to the browser, Resin uses a default error page unless you explicitly provide an error page yourself?

- a. Client Request Time Processing Errors
- b. Compilation Time Processing Errors
- c. JSP Translation Time Processing Errors
- d. None of the above

4) The ASP and JSP technologies are quite similar in the way they support the creation of Dynamic pages, using HTML templates, scripting code and components for business logic.

- a. True
- b. False

5) Which can generate HTML dynamically on the client but can hardly interact with the web server to perform complex tasks like database access and image processing etc. in JSP?

a. vs.Static HTML

b. vs.Server-Side Includes

c. vs.Pure Servlets

d. vs.JavaScript

6) In JSP Action tags which tags are used for bean development?

a. `jsp:useBean`

b. `jsp:setProperty`

c. `jsp:getProperty`

d. All mentioned above

7) Which two interfaces does the `javax.servlet.jsp` package have?

a. `JspPage`

b. `HttpJspPage`

c. `JspWriter`

d. `PageContext`

e. Both A & B

8) Which of the following is an advantage of the statement – Separation of business logic from JSP ?

a. Custom Tags in JSP

b. JSP Standard Tag Library

c. All the above

d. None of the above

9) JSPs eventually are compiled into Java servlets, you can do as much with JSPs as you can do with Java servlets.

a. True

b. False

10) How many jsp implicit objects are there and these objects are created by the web container that are available to all the jsp pages?

a. 8

b. 9

c. 10

d. 7

11) JavaServer Pages often serve the same purpose as programs implemented using the Common Gateway Interface (CGI)

a. True

b. False

12) Which action tags are used in JSP for developing web application with Java Bean?

a. `jsp:useBean`

b. `jsp:setProperty`

c. `jsp:getProperty`

d. Both B & C

13) Which technology do we mix our business logic with the presentation logic?

a. Servlet

- b. JSP
- c. Both A & B
- d. None of the above

14) Which is the Microsoft solution for providing dynamic Web content?

- a. ASP
- b. JSP
- c. Both A & B
- d. None of the above

15) A bean encapsulates many objects into one object, so we can access this object from multiple places.

- a. True
- b. False

16) Which tag is used to execute java source code in JSP?

- a. Declaration Tag
- b. Scriptlet tag
- c. Expression tag
- d. None of the above

17) Which JSP Action tags is used to include the content of another resource, it may be jsp, html or servlet?

- a. jsp:include
- b. jsp:forward
- c. jsp:plugin
- d. jsp:papam

18) In JSP how many ways are there to perform exception handling?

- a. 3
- b. 2
- c. 4
- d. 5

19) In JSP page directive which attribute defines the MIME(Multipurpose Internet Mail Extension) type of the HTTP response?

- a. import
- b. Content Type
- c. Extends
- d. Info

20) A JSP page consists of which tags?

- a. HTML tags
- b. JSP tags
- c. Both A & B
- d. None of the above

21) Which packages does a JSP API consist of?

- a. javax.servlet.jsp
- b. java.servlet
- c. javax.servlet.jsp.tagext
- d. Both A & C

22) JSP's provide better facilities for separation of page code and template data by mean of Java beans, EJBs and custom tag libraries

a. True

b. False

Practical Exam Sample Paper

Set-A

Q1. Write a program to calculate difference of two numbers using Remote Method Invocation.

Q2. Write a note on Model View Controller

Set-B

Q1. Write a program to implement serialization and deserialization in Java.

Q2. Describe Java Servlet.

Viva Voce (Interview based)

What is a transient variable?

A transient variable is a variable that may not be serialized.

Which containers use a border Layout as their default layout?

The **Window**, **Frame** and **Dialog** classes use a border layout as their default layout.

Why do threads block on I/O?

Threads block on I/O (that is enters the waiting state) so that other threads may execute while the I/O Operation is performed.

How are Observer and Observable used?

Objects that subclass the **Observable** class maintain a list of observers. When an **Observable** object is updated it invokes the **update()** method of each of its observers to notify the observers that it has changed state. The Observer interface is implemented by objects that observe **Observable** objects.

What is synchronization and why is it important?

With respect to multithreading, synchronization is the capability to control the access of multiple threads to shared resources. Without synchronization, it is possible for one thread to modify a shared object while another thread is in the process of using or updating that object's value. This often leads to significant errors.

Can a lock be acquired on a class?

Yes, a lock can be acquired on a class. This lock is acquired on the class's **Class** object..

What's new with the stop(), suspend() and resume() methods in JDK 1.2?

The **stop()**, **suspend()** and **resume()** methods have been deprecated in JDK 1.2.

Is null a keyword?

The **null** is not a keyword.

What is the preferred size of a component?

The preferred size of a component is the minimum component size that will allow the component to display normally.

What method is used to specify a container's layout?

The **setLayout()** method is used to specify a container's layout.

Which containers use a FlowLayout as their default layout?

The **Panel** and **Applet** classes use the **FlowLayout** as their default layout.

What state does a thread enter when it terminates its processing?

When a thread terminates its processing, it enters the dead state.

What is the Collections API?

The Collections API is a set of classes and interfaces that support operations on collections of objects.

Which characters may be used as the second character of an identifier, but not as the first character of an identifier?

The digits 0 through 9 may not be used as the first character of an identifier but they may be used after the first character of an identifier.

What is the List interface?

The **List** interface provides support for ordered collections of objects.

How does Java handle integer overflows and underflows?

It uses those low order bytes of the result that can fit into the size of the type allowed by the operation.

What is the Vector class?

The **Vector** class provides the capability to implement a growable array of objects

What modifiers may be used with an inner class that is a member of an outer class?

A (non-local) inner class may be declared as **public**, **protected**, **private**, **static**, **final**, or **abstract**.

What is an Iterator interface?

The **Iterator** interface is used to step through the elements of a Collection.

What is the difference between the >> and >>> operators?

The >> operator carries the sign bit when shifting right. The >>> zero-fills bits that have been shifted out.

Which method of the Component class is used to set the position and size of a component?

setBounds() method is used to set the position and size of a component.

What is the difference between yielding and sleeping?

When a task invokes its **yield()** method, it returns to the ready state. When a task invokes its **sleep()** method, it returns to the waiting state.

Which java.util classes and interfaces support event handling?

The **EventObject** class and the **EventListener** interface support event processing.

Is sizeof a keyword?

The **sizeof** operator is not a keyword.

What is the difference between a Scrollbar and a ScrollPane?

A **Scrollbar** is a Component, but not a Container. A **ScrollPane** is a Container.
A **ScrollPane** handles its own events and performs its own scrolling.

What is the difference between a public and a non-public class?

A public class may be accessed outside of its package.

A non-public class may not be accessed outside of its package.

To what value is a variable of the boolean type automatically initialized?

The default value of the **boolean** type is **false**.

Can try statements be nested?

Try statements may be tested.

What is the difference between the prefix and postfix forms of the ++ operator?

The prefix form performs the increment operation and returns the value of the increment operation.

The postfix form returns the current value all of the expression and then performs the increment operation on that value.

What is the purpose of a statement block?

A statement block is used to organize a sequence of statements as a single statement group.

Reference Books

1. Java The Complete Reference, Herbert Schildt
2. Programming with Java, E. Balagurusamy