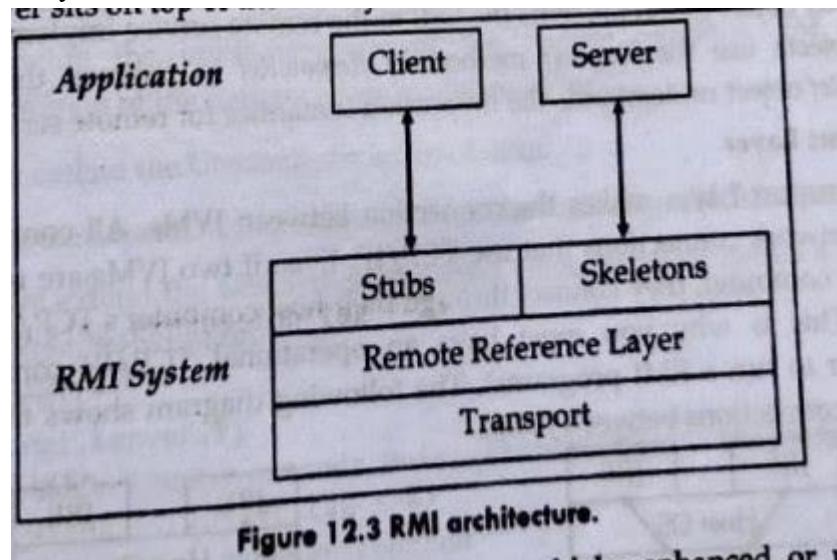


1. Describe RMI architecture with suitable block diagram.

The RMI system consists of three layers:

- a) The stub/skeleton layer
- b) The remote reference layer
- c) The transport layer



By using a layered architecture each of the layers could be enhanced or replaced without affecting the rest of the system. For example, the transport layer could be replaced by a UDP/IP layer without affecting the upper layers.

a) Stub and Skeleton Layer

The stub/skeleton layer is the interface between the application layer and the rest of the RMI system. This layer does not deal with specifics of any transport, but transmits data to the remote reference layer by using marshaling. It also takes data from remote reference layer by using unmarshalling or demarshalling.

During communication between two machines through RPC or RMI, parameters are packed into a message and then sent over the network. This packing of parameters into a message is called marshalling. On the other side these packed parameters are unpacked from the message which is called unmarshalling.

A stub for a remote object is the client-side proxy for the remote object. Such a stub implements all the interfaces that are supported by the remote object implementation.

b) Remote Reference Layer

The remote reference layer defines and supports the invocation semantics of the RMI connection. This layer maintains the session during the method call. This layer provides a Remote Ref object that represents the link to the remote service implementation object. The stub objects use the invoke() method in RemoteRef to forward the method call. The RemoteRef object understands the invocation semantics for remote services.

c) Transport Layer

The Transport Layer makes the connection between JVMs. All connections are stream based network connections that use TCP/IP. Even if two JVMs are running on the same physical computer, they connect through their host computer's TCP/IP network protocol

stack. (This is why you must have an operational TCP/IP configuration on your computer to run a RMI programs).

2.How to access database in JSP? Explain with suitable example.

To perform database access through JSP, you can use JDBC (Java Database Connectivity) to interact with a database.

Accessing a database in JSP (JavaServer Pages) involves using JDBC (Java Database Connectivity). JDBC provides a set of classes and interfaces to connect and interact with databases from Java applications, including JSP pages. Here's a detailed explanation along with a simple example.

Steps to Access a Database in JSP

a.Load the JDBC Driver

- o You need to load the driver class for your database. For example, for MySQL:
`Class.forName("com.mysql.cj.jdbc.Driver");`

b.Establish a Connection

- o Create a connection to the database using `DriverManager.getConnection()`:
`Connection con = DriverManager.getConnection("jdbc:mysql://localhost:3306/db_name", "username", "password");`

c.Create a Statement

- o Use Statement or PreparedStatement to execute SQL queries:
`Statement stmt = con.createStatement();`

d.Execute SQL Queries

- o Execute queries using `executeQuery()` for SELECT and `executeUpdate()` for INSERT, UPDATE, DELETE:
`ResultSet rs = stmt.executeQuery("SELECT * FROM students");`

e.Process the Result

- o Loop through the ResultSet to fetch data:
`while(rs.next()) {
 out.println("ID: " + rs.getInt("id") + " Name: " + rs.getString("name"));
}`

f.Close the Connection

- o Always close the database connection to free resources:
`rs.close();
stmt.close();
con.close();`

Example JSP Page to Access Database

Suppose you have a MySQL database named college with a table students:

| id | name | age | course |
|----|--------|-----|---------|
| 1 | Bibash | 21 | IT |
| 2 | Sita | 22 | Science |

```
Students.jsp
<%@ page import="java.sql.*" %>
<html>
<head>
    <title>Student List</title>
</head>
<body>
<h2>Student List from Database</h2>
<%
    String url = "jdbc:mysql://localhost:3306/college";
    String user = "root";
    String password = "1234";

    Connection con = null;
    Statement stmt = null;
    ResultSet rs = null;

    try {
        // Load JDBC driver
        Class.forName("com.mysql.cj.jdbc.Driver");

        // Connect to database
        con = DriverManager.getConnection(url, user, password);

        // Create statement
        stmt = con.createStatement();

        // Execute query
        rs = stmt.executeQuery("SELECT * FROM students");

        // Display results
        out.println("<table border='1'>");

        out.println("<tr><th>ID</th><th>Name</th><th>Age</th><th>Course</th></tr>");
        while(rs.next()) {
            out.println("<tr>");
            out.println("<td>" + rs.getInt("id") + "</td>");
            out.println("<td>" + rs.getString("name") + "</td>");
            out.println("<td>" + rs.getInt("age") + "</td>");
            out.println("<td>" + rs.getString("course") + "</td>");
            out.println("</tr>");
        }
        out.println("</table>");

    }
%>
```

```

    } catch(Exception e) {
        out.println("Error: " + e.getMessage());
    } finally {
        try { if(rs != null) rs.close(); } catch(Exception e) {}
        try { if(stmt != null) stmt.close(); } catch(Exception e) {}
        try { if(con != null) con.close(); } catch(Exception e) {}
    }
%>
</body>
</html>

```

3. Describe JDBC driver types in Java with suitable example.

JDBC (Java Database Connectivity) drivers are used to connect Java applications (including JSP/Servlets) with a database. There are **four types of JDBC drivers** in Java, each with its own architecture, advantages, and limitations. Here's a detailed explanation with examples.

1. JDBC-ODBC Bridge Driver (Type 1)

- Description:
This driver uses the ODBC driver to connect to the database. Java application communicates with the ODBC driver, which then communicates with the database.
- Advantages: Simple to use for small projects.
- Disadvantages: Platform-dependent, slower, and requires ODBC setup.

Example:

```

Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
Connection con = DriverManager.getConnection("jdbc:odbc:DSNName",
    "username", "password");

```

2. Native-API Driver (Type 2)

- Description:
This driver uses native code library provided by the database vendor. Java calls native database API to communicate with the database.
- Advantages: Faster than Type 1.
- Disadvantages: Platform-dependent (requires client library installed on the system).

Example (Oracle Database):

```

Class.forName("oracle.jdbc.driver.OracleDriver");
Connection con = DriverManager.getConnection(
    "jdbc:oracle:thin:@localhost:1521:XE", "username", "password");

```

3. Network Protocol Driver / Middleware Driver (Type 3)

- Description:
This driver communicates with a middleware server using database-independent network protocol. The middleware then connects to the database.

- Advantages: Database-independent, suitable for Internet applications.
- Disadvantages: Needs middleware, slightly slower than Type 4.

Example:

```
Class.forName("com.somevendor.jdbc.Driver");
Connection con = DriverManager.getConnection(
    "jdbc:someprotocol://server:port/database", "username", "password");
```

4. Thin Driver / Pure Java Driver (Type 4)

- Description:

This is a 100% Java driver that converts JDBC calls directly into the network protocol of the database. No native library or middleware is required.

- Advantages: Platform-independent, fast, most commonly used.
- Disadvantages: Specific to the database vendor.

Example (MySQL):

```
Class.forName("com.mysql.cj.jdbc.Driver");
Connection con = DriverManager.getConnection(
    "jdbc:mysql://localhost:3306/college", "root", "1234");
```

4.What is Java Beans? Explain their advantages.

A JavaBean is a reusable, platform-independent, and serializable Java class that follows certain conventions. JavaBeans are mainly used to encapsulate multiple objects into a single object (the bean), so they can be easily passed around in applications, especially in JSP, Servlets, and GUI development.

Characteristics / Conventions of JavaBeans

1. Private Properties (Fields):
 - The data (variables) in a JavaBean must be private to ensure encapsulation.
2. Public Getter and Setter Methods:
 - Provide access to private properties using standard naming conventions:
 - public void setPropertyName(Type property) { ... }
 - public Type getPropertyName() { ... }
3. No-Argument Constructor:
 - A JavaBean must have a public default constructor to allow easy instantiation.
4. Serializable:
 - Implements java.io.Serializable so that the bean can be persisted or transferred.

Example of a JavaBean

```
import java.io.Serializable;

public class Student implements Serializable {
    private int id;
    private String name;
    private int age;
```

```

// No-argument constructor
public Student() { }

// Getter for id
public int getId() {
    return id;
}

// Setter for id
public void setId(int id) {
    this.id = id;
}

// Getter for name
public String getName() {
    return name;
}

// Setter for name
public void setName(String name) {
    this.name = name;
}

// Getter for age
public int getAge() {
    return age;
}

// Setter for age
public void setAge(int age) {
    this.age = age;
}

```

Advantages of JavaBeans

Reusability:

- Beans can be reused across multiple applications.

Encapsulation:

- Private fields with getter/setter methods ensure data hiding.

Easy to Use in JSP/Servlets:

- Directly usable in JSP using <jsp:useBean>, <jsp:setProperty>, and <jsp:getProperty>.

Platform Independent:

- JavaBeans are pure Java classes, so they work on any platform.

Supports Persistence:

- Being serializable allows beans to be saved and restored, e.g., in sessions.

Simplifies GUI Development:

- Often used in GUI builders (like NetBeans) as components.

5. Describe mouse event handling with suitable Java program.

Mouse event handling in Java allows a program to respond to mouse actions such as clicking, pressing, releasing, entering, exiting, dragging, or moving the mouse on a GUI component. It is part of the AWT or Swing event-handling mechanism.

Mouse Events in Java

Java provides the `java.awt.event` package which defines several mouse-related events:

| Event Method | Description |
|--|--|
| <code>mouseClicked(MouseEvent e)</code> | Triggered when the mouse is clicked (pressed and released). |
| <code>mousePressed(MouseEvent e)</code> | Triggered when the mouse button is pressed. |
| <code>mouseReleased(MouseEvent e)</code> | Triggered when the mouse button is released. |
| <code>mouseEntered(MouseEvent e)</code> | Triggered when the mouse enters a component. |
| <code>mouseExited(MouseEvent e)</code> | Triggered when the mouse exits a component. |
| <code>mouseDragged(MouseEvent e)</code> | Triggered when the mouse is dragged while a button is pressed. |
| <code>mouseMoved(MouseEvent e)</code> | Triggered when the mouse is moved without any button pressed. |

Steps to Handle Mouse Events

Import Required Packages

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

Implement MouseListener and/or MouseMotionListener Interface

- MouseListener handles click, press, release, enter, exit events.
- MouseMotionListener handles drag and move events.

Override Required Methods

Add Mouse Listener to Component

Example Program: Mouse Event Handling

This example creates a Frame that displays messages when mouse events occur:

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

```
public class MouseEventDemo extends Frame implements MouseListener,
MouseMotionListener {
    Label label;

    public MouseEventDemo() {
        setLayout(new FlowLayout());
        setSize(400, 300);
        setTitle("Mouse Event Demo");

        label = new Label("Move or Click the Mouse");
        add(label);

        // Register mouse listeners
        addMouseListener(this);
        addMouseMotionListener(this);

        setVisible(true);

        // Close window on click of close button
        addWindowListener(new WindowAdapter() {
            public void windowClosing(WindowEvent we) {
                System.exit(0);
            }
        });
    }

    // MouseListener methods
    public void mouseClicked(MouseEvent e) {
        label.setText("Mouse Clicked at: " + e.getX() + ", " + e.getY());
    }

    public void mousePressed(MouseEvent e) {
        label.setText("Mouse Pressed");
    }

    public void mouseReleased(MouseEvent e) {
        label.setText("Mouse Released");
    }
}
```

```

public void mouseEntered(MouseEvent e) {
    label.setText("Mouse Entered Frame");
}

public void mouseExited(MouseEvent e) {
    label.setText("Mouse Exited Frame");
}

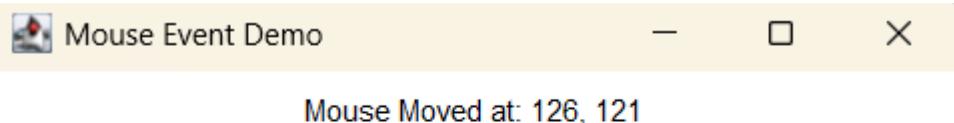
// MouseMotionListener methods
public void mouseDragged(MouseEvent e) {
    label.setText("Mouse Dragged at: " + e.getX() + ", " + e.getY());
}

public void mouseMoved(MouseEvent e) {
    label.setText("Mouse Moved at: " + e.getX() + ", " + e.getY());
}

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

```

Output:



6. Why do we need top level container like JFrame to write Java programs with GUI? How can we display 2D objects in Java?

A top-level container in Java is a window with decorations (like title bar, close/minimize buttons) that can hold other GUI components. Common top-level containers are:

- JFrame (used for main application window)
- JDialog (popup dialog)
- JApplet (used in older Java applets)

Reasons for using JFrame (or any top-level container):

1. Provides a Window for Components
 - Components like buttons, labels, text fields need a container to appear on the screen.

2. Manages Layout and Display
 - o JFrame can hold multiple components using layout managers.
3. Handles Window Events
 - o Supports closing, minimizing, maximizing, resizing, etc.
4. Serves as the Main GUI Window
 - o All GUI elements must be added to a container that can be displayed.

Example: Simple JFrame

```
import javax.swing.*;
```

```
public class SimpleFrame {
    public static void main(String[] args) {
        JFrame frame = new JFrame("My First GUI"); // Top-level container
        frame.setSize(400, 300); // Set size of window
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE); // Close
        window properly
        frame.setVisible(true); // Make it visible
    }
}
```

Display 2D objects in Java

Java provides java.awt.Graphics and java.awt.Graphics2D classes to draw 2D objects (lines, rectangles, ovals, etc.) on components.

Steps to Draw 2D Objects:

1. Extend a Component (e.g., JPanel)
 - o Create a class that extends JPanel or another container.
2. Override paintComponent(Graphics g) Method
 - o Use the Graphics object to draw shapes.
3. Use Graphics2D for Advanced Drawing
 - o Convert Graphics to Graphics2D for better control (rotation, stroke, transformations).

Example: Drawing 2D Shapes

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

```
class MyPanel extends JPanel {
    @Override
    protected void paintComponent(Graphics g) {
        super.paintComponent(g);
        Graphics2D g2d = (Graphics2D) g;

        // Draw a rectangle
        g2d.setColor(Color.BLUE);
```

```

        g2d.fillRect(50, 50, 100, 80);

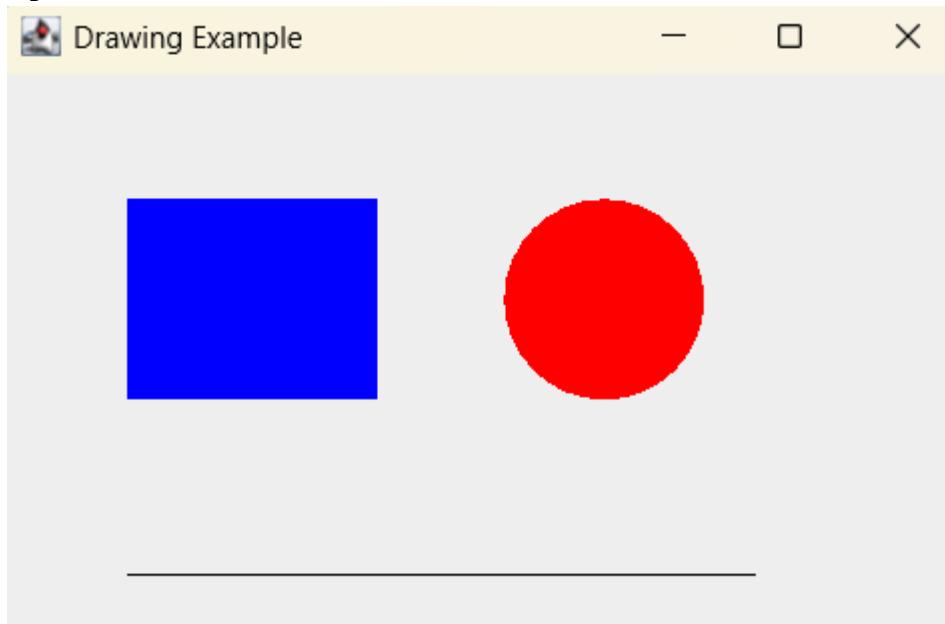
        // Draw a circle
        g2d.setColor(Color.RED);
        g2d.fillOval(200, 50, 80, 80);

        // Draw a line
        g2d.setColor(Color.BLACK);
        g2d.drawLine(50, 200, 300, 200);
    }
}

public class Draw2DExample {
    public static void main(String[] args) {
        JFrame frame = new JFrame("2D Drawing Example");
        frame.setSize(400, 300);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.add(new MyPanel()); // Add custom panel to JFrame
        frame.setVisible(true);
    }
}

```

Output:



7.Explain JTextField and JTextArea components of Java swing library.

JTextField and JTextArea in Java Swing

Java Swing provides two essential components for text input and editing: JTextField and JTextArea. Here's a concise explanation of each:

1. JTextField

- Purpose: Used for single-line text input.
- Features:
 - Allows users to enter or edit a single line of text.
 - Inherits from JTextField and implements SwingConstants.
 - Commonly used for fields like username, password, or search bars.
- Key Methods:
 - getText(): Retrieves the text entered.
 - setText(String text): Sets the text in the field.
 - setColumns(int columns): Sets the width of the text field in terms of columns.

Example:

Java

Copy code

```
import javax.swing.*;
```

```
public class JTextFieldExample {
    public static void main(String[] args) {
        JFrame frame = new JFrame("JTextField Example");
        JTextField textField = new JTextField(20); // 20 columns wide
        frame.add(textField);
        frame.setSize(300, 100);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setVisible(true);
    }
}
```

2. JTextArea

- Purpose: Used for multi-line text input.
- Features:
 - Allows users to enter or edit multiple lines of text.
 - Inherits from JTextField.
 - Commonly used for comments, descriptions, or larger text inputs.
- Key Methods:
 - getText(): Retrieves the text entered.
 - setText(String text): Sets the text in the area.
 - setRows(int rows): Sets the number of visible rows.
 - setColumns(int columns): Sets the width in terms of columns.
 - append(String text): Appends text to the existing content.

Example:

```
import javax.swing.*;
```

```
public class JTextAreaExample {
    public static void main(String[] args) {
        JFrame frame = new JFrame("JTextArea Example");
```

```

JTextArea textArea = new JTextArea(5, 20); // 5 rows, 20 columns
frame.add(new JScrollPane(textArea)); // Add scroll for better usability
frame.setSize(300, 200);
frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
frame.setVisible(true);
}
}

Output:

```



8.How do you execute SQL Statements using JDBC? Explain with example.

Executing SQL Statements Using JDBC

JDBC (Java Database Connectivity) is an API in Java that allows you to interact with databases. To execute SQL statements using JDBC, you follow these steps:

Steps to Execute SQL Statements Using JDBC

1. Load the JDBC Driver: Register the driver class using Class.forName() or DriverManager.
2. Establish a Connection: Use DriverManager.getConnection() to connect to the database.
3. Create a Statement:
Use Connection.createStatement() or Connection.prepareStatement() for executing SQL queries.
4. Execute the Query: Use methods like executeQuery() for SELECT statements or executeUpdate() for INSERT, UPDATE, DELETE.
5. Process the Results: For SELECT queries, process the ResultSet object.
6. Close Resources: Always close the ResultSet, Statement, and Connection to free resources.

Example: Executing SQL Statements

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

```

public class JDBCExample {
    public static void main(String[] args) {

```

```
// Database URL, username, and password

String url = "jdbc:mysql://localhost:3306/mydatabase";
String username = "root";
String password = "password";

// SQL query

String query = "SELECT * FROM employees";

try {

    // 1. Load the JDBC Driver

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

    // 2. Establish a Connection

    Connection connection = DriverManager.getConnection(url, username,
password);

    // 3. Create a Statement

    Statement statement = connection.createStatement();

    // 4. Execute the Query

    ResultSet resultSet = statement.executeQuery(query);

    // 5. Process the Results

    while (resultSet.next()) {

        System.out.println("ID: " + resultSet.getInt("id"));

        System.out.println("Name: " + resultSet.getString("name"));

        System.out.println("Department: " + resultSet.getString("department"));

    }

    // 6. Close Resources

    resultSet.close();

}
```

```

        statement.close();

        connection.close();

    } catch (ClassNotFoundException e) {
        System.out.println("JDBC Driver not found: " + e.getMessage());
    } catch (SQLException e) {
        System.out.println("Database error: " + e.getMessage());
    }
}
}
}

```

9.Compare result set with row set. Explain prepared statement with example.

A ResultSet maintains a live connection to the database, while a RowSet is a disconnected, serializable version of it. A PreparedStatement is a precompiled SQL statement that improves performance and security.

Comparison: ResultSet vs RowSet

| Feature | Resultset | Rowset |
|------------------------|--------------------------------------|-----------------------------------|
| Connection | Connected to the database | Can be disconnected |
| Serialization | Not serializable | Serializable |
| javaBean compatibility | Not a JavaBean | Is a JavaBean |
| Network Transfer | Cannot be passed over network | Can be passed over network |
| Usage | Used for direct database interaction | Used for cached data manipulation |

Prepared Statement

A PreparedStatement is a subclass of Statement in JDBC that allows you to execute parameterized SQL queries. It helps prevent SQL injection and improves performance by precompiling the SQL statement.

Benefits:

- Security: Protects against SQL injection.
- Performance: SQL is compiled once and reused.
- Convenience: Parameters can be set dynamically.

Example:

```

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

```

```

String url = "jdbc:mysql://localhost:3306/mydb";
String user = "root";
String password = "password";

try (Connection conn = DriverManager.getConnection(url, user, password)) {
    String query = "SELECT * FROM users WHERE age > ?";
    PreparedStatement pstmt = conn.prepareStatement(query);
    pstmt.setInt(1, 25); // Set age parameter

    ResultSet rs = pstmt.executeQuery();
    while (rs.next()) {
        System.out.println("User: " + rs.getString("name") + ", Age: " +
        rs.getInt("age"));
    }
} catch (SQLException e) {
    e.printStackTrace();
}
}
}
}

```

10. How do you set and get Cookie in Servlet? Explain using suitable Java program.

In a Java Servlet, you set a cookie using `response.addCookie()` and retrieve it using `request.getCookies()`. Below is a complete example demonstrating both.

Setting and Getting Cookies in Java Servlet

Step 1: Set a Cookie

To set a cookie, create a `Cookie` object and add it to the response:

```

import java.io.*;
import jakarta.servlet.*;
import jakarta.servlet.http.*;

```

```

public class SetCookieServlet extends HttpServlet {
    public void doGet(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
        response.setContentType("text/html");

        // Create a cookie
        Cookie userCookie = new Cookie("username", "JohnDoe");

        // Set cookie expiration to 1 day (in seconds)
        userCookie.setMaxAge(24 * 60 * 60);

        // Add cookie to response
        response.addCookie(userCookie);
    }
}

```

```

        PrintWriter out = response.getWriter();
        out.println("<h2>Cookie has been set!</h2>");
    }
}

```

Step 2: Get a Cookie

To retrieve cookies, use `request.getCookies()` and loop through them:

```

import java.io.*;
import jakarta.servlet.*;
import jakarta.servlet.http.*;

```

```

public class GetCookieServlet extends HttpServlet {
    public void doGet(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
        response.setContentType("text/html");
        PrintWriter out = response.getWriter();

        Cookie[] cookies = request.getCookies();
        if (cookies != null) {
            for (Cookie cookie : cookies) {
                out.println("<p>" + cookie.getName() + " = " + cookie.getValue() + "</p>");
            }
        } else {
            out.println("<p>No cookies found</p>");
        }
    }
}

```

11.Explain different scripting elements of JSP with example.

JSP (JavaServer Pages) provides three main scripting elements: *declarations*, *scriptlets*, and *expressions*. Each allows embedding Java code into HTML to create dynamic web content.

Types of JSP Scripting Elements

JSP scripting elements enable Java code to be inserted directly into a JSP page. Here's a breakdown of each:

1. Declaration (<%! ... %>)

- Purpose: Declare variables or methods that can be used throughout the JSP page.
- Scope: Class-level (shared across requests).

Example:

```

<%! int counter = 0; %>
<%! String greetUser(String name) {
    return "Hello, " + name;
}

```

} %>

2. Scriptlet (<% ... %>)

- Purpose: Embed Java code that executes each time the page is requested.
- Scope: Inside the `_jspService()` method.
- Syntax: <% Java code %>

Example:

```
<%  
    counter++;  
    String user = "Alice";  
%>  
<p>Welcome, <%= user %>! You are visitor number <%= counter %>.</p>  
3. Expression (<%= ... %>)  
• Purpose: Output the result of a Java expression directly into the HTML.  
• Scope: Evaluated and converted to a string.  
• Syntax: <%= expression %>  
Example:  
<p>Current time: <%= new java.util.Date() %></p>
```

s

12. Write short notes on Java web framework, CORBA, and Bean bound property.

Java Web Framework

A Java web framework is a software platform that provides tools and libraries to build web applications efficiently. These frameworks abstract away low-level details like HTTP handling, session management, and database connectivity.

Popular Java web frameworks include:

- Spring MVC: Offers powerful dependency injection and RESTful web services.
- Struts: Based on the MVC architecture, good for form-based applications.
- JSF (JavaServer Faces): Component-based UI framework integrated with Java EE.
- Grails: Built on Groovy, simplifies Java web development.

Benefits:

- Reduces boilerplate code
- Enhances scalability and maintainability
- Supports MVC architecture and REST APIs

CORBA (Common Object Request Broker Architecture)

CORBA is a standard defined by the Object Management Group (OMG) for enabling communication between software components written in different languages and running on different platforms.

Key features:

- Language independence: Supports Java, C++, Python, etc.
- Platform neutrality: Works across operating systems.

- Object Request Broker (ORB): Mediates method calls between clients and servers.
How it works:
 - A client invokes a method on a remote object.
 - The ORB locates the object, passes parameters, executes the method, and returns results.
 - Uses IDL (Interface Definition Language) to define object interfaces.
 CORBA is useful in large-scale distributed systems but has been largely replaced by modern technologies like REST and gRPC.

Bean Bound Property

A bound property in a Java Bean is a property that notifies registered listeners when its value changes.

Mechanism:

- The bean fires a `PropertyChangeEvent` when the property changes.
- Listeners implement `PropertyChangeListener` and register with the bean.
- This allows other components to react to changes dynamically.

Example:

```
private String name;
private PropertyChangeSupport support = new PropertyChangeSupport(this);

public void setName(String newName) {
    String oldName = this.name;
    this.name = newName;
    support.firePropertyChange("name", oldName, newName);
}

public void addPropertyChangeListener(PropertyChangeListener pcl) {
    support.addPropertyChangeListener(pcl);
}
```

13. Discuss AWT components **TextField**, **TextArea**, **Label**, **CheckBox**, and **CheckBoxGroup** briefly.

1. **TextField**

- Purpose: Allows single-line text input.
- Class: `java.awt.TextField`
- Key Methods:
 - `getText()` – retrieves the entered text
 - `setText(String)` – sets default text
- Example:
`TextField tf = new TextField("Enter name");`

2. **TextArea**

- Purpose: Allows multi-line text input.

- Class: `java.awt.TextArea`
- Key Methods:
 - `append(String)` – adds text at the end
 - `setText(String)` – sets content
- Example:

```
TextArea ta = new TextArea("Write your message here", 5, 40);
```

3. Label

- Purpose: Displays static text.
- Class: `java.awt.Label`
- Key Methods:
 - `setText(String)` – updates label text
 - `getText()` – retrieves label content

- Example:

```
Label lbl = new Label("Username:");
```

4. CheckBox

- Purpose: Allows selection of multiple options independently.
- Class: `java.awt.Checkbox`
- Key Methods:
 - `getState()` – returns true if selected
 - `setState(boolean)` – sets selection

- Example:

```
Checkbox cb1 = new Checkbox("Java");
Checkbox cb2 = new Checkbox("Python");
```

5. CheckBoxGroup

- Purpose: Groups checkboxes to behave like radio buttons (only one can be selected).
- Class: `java.awt.CheckboxGroup`
- Usage: Used with Checkbox constructor to assign group.
- Example:

```
CheckboxGroup genderGroup = new CheckboxGroup();
Checkbox male = new Checkbox("Male", genderGroup, false);
Checkbox female = new Checkbox("Female", genderGroup, true);
```

14. Differentiate between Choice class and List class in AWT with suitable description and example.

Difference between Choices and List class in AWT

| Feature | Choice class | List class |
|-------------------|--------------------------------------|------------------------------|
| Type of component | Drop-down menu (single visible item) | Scrollable list box |
| Selection Mode | Single selection only | Single or multiple selection |

| | | |
|---------------|--------------------------|--|
| UI behavior | Compact, expand on click | Shows multiple items at once |
| Scrolling | No scroll bar | Has scroll bar if items exceed visible count |
| Class package | Java.awt.Choice | Java.awt.List |

Example: Using Choice

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

public class ChoiceExample extends Applet {
    public void init() {
        Choice language = new Choice();
        language.add("Java");
        language.add("Python");
        language.add("C++");
        add(language);
    }
}
```

Example: Using List

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

public class ListExample extends Applet {
    public void init() {
        List languageList = new List(3, true); // 3 visible rows, multiple selection
        languageList.add("Java");
        languageList.add("Python");
        languageList.add("C++");
        languageList.add("JavaScript");
        add(languageList);
    }
}
```

15.Explain gridbag layout manager with suitable constructors and also demonstrate gridbag constraints briefly.

GridBagLayout is one of the most flexible and complex layout managers in Java AWT. It arranges components in a grid of cells, allowing varying sizes and alignments. Each component can span multiple rows or columns and be aligned in different ways using GridBagConstraints

GridBagLayout Overview

- Package: java.awt
- Class: GridBagLayout
- Purpose: To place components in a grid with fine control over size, position, and alignment.

Constructors

```
GridBagLayout layout = new GridBagLayout();
```

GridBagConstraints

To control the placement and behavior of each component, you use GridBagConstraints.

Key Fields:

| Field | Description |
|-----------------------|--|
| Gridx, gridy | Position in grid(row, column) |
| Gridwidth, gridheight | Number of columns/rows the component spans |
| weightx, weighty | How extra space is distributed |
| Fill | How component fills space (NONE, HORIZONTAL, VERTICAL, BOTH) |
| Anchor | Alignment (CENTER, NORTH, SOUTH, etc.) |
| Insets | External padding |
| ipadx, ipady | Internal padding |

Example: Using GridBagLayout

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

public class GridBagExample {
    public static void main(String[] args) {
        JFrame frame = new JFrame("GridBagLayout Demo");
        frame.setLayout(new GridBagLayout());
        GridBagConstraints gbc = new GridBagConstraints();

        JButton btn1 = new JButton("Button 1");
        gbc.gridx = 0;
        gbc.gridy = 0;
        gbc.insets = new Insets(5, 5, 5, 5);
        frame.add(btn1, gbc);

        JButton btn2 = new JButton("Button 2");
        gbc.gridx = 1;
        gbc.gridy = 0;
        frame.add(btn2, gbc);

        JButton btn3 = new JButton("Button 3");
    }
}
```

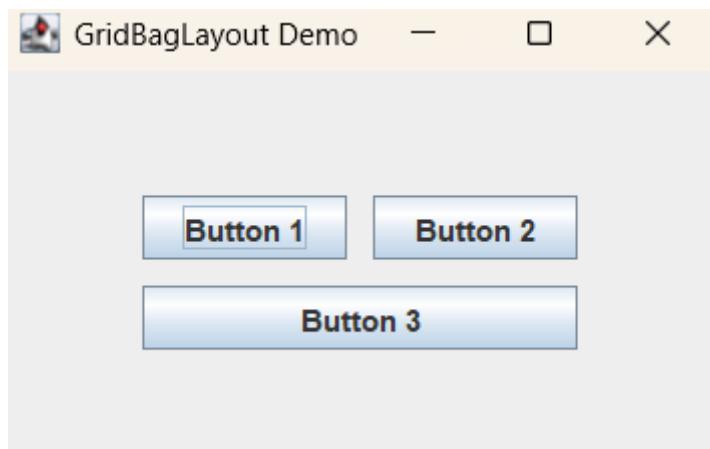
```

gbc.gridx = 0;
gbc.gridy = 1;
gbc.gridwidth = 2;
gbc.fill = GridBagConstraints.HORIZONTAL;
frame.add(btn3, gbc);

frame.setSize(300, 200);
frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
frame.setVisible(true);
}
}

```

Output:



16. Write a program to prepare one small form containing fields; employee id, name and salary with buttons view, insert, and delete. And perform the operations as indicated by button names.

Program:

```

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

class Employee {
    String id, name;
    double salary;

    Employee(String id, String name, double salary) {
        this.id = id;
        this.name = name;
        this.salary = salary;
    }
}

```

```
public class EmployeeForm extends JFrame implements ActionListener {  
    JTextField txtId, txtName, txtSalary;  
    JButton btnInsert, btnView, btnDelete;  
    ArrayList<Employee> employees = new ArrayList<>();  
  
    EmployeeForm() {  
        setTitle("Employee Form");  
        setSize(400, 300);  
        setLayout(new GridLayout(5, 2, 10, 10));  
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);  
        setLocationRelativeTo(null);  
  
        // Labels and Fields  
        add(new JLabel("Employee ID:"));  
        txtId = new JTextField();  
        add(txtId);  
  
        add(new JLabel("Name:"));  
        txtName = new JTextField();  
        add(txtName);  
  
        add(new JLabel("Salary:"));  
        txtSalary = new JTextField();  
        add(txtSalary);  
  
        // Buttons  
        btnInsert = new JButton("Insert");  
        btnView = new JButton("View");  
        btnDelete = new JButton("Delete");  
  
        add(btnInsert);  
        add(btnView);  
        add(btnDelete);  
  
        // Event Listeners  
        btnInsert.addActionListener(this);  
        btnView.addActionListener(this);  
        btnDelete.addActionListener(this);  
  
        setVisible(true);  
    }  
}
```

```

@Override
public void actionPerformed(ActionEvent e) {
    String id = txtId.getText().trim();
    String name = txtName.getText().trim();
    String salaryText = txtSalary.getText().trim();

    if (e.getSource() == btnInsert) {
        if (id.isEmpty() || name.isEmpty() || salaryText.isEmpty()) {
            JOptionPane.showMessageDialog(this, "Please fill all fields!");
            return;
        }
        double salary = Double.parseDouble(salaryText);
        employees.add(new Employee(id, name, salary));
        JOptionPane.showMessageDialog(this, "Employee inserted successfully!");
        clearFields();
    }

    else if (e.getSource() == btnView) {
        if (employees.isEmpty()) {
            JOptionPane.showMessageDialog(this, "No employees to display!");
            return;
        }
        StringBuilder data = new StringBuilder("Employee List:\n");
        for (Employee emp : employees) {
            data.append("ID: ").append(emp.id)
                .append(", Name: ").append(emp.name)
                .append(", Salary: ").append(emp.salary).append("\n");
        }
        JOptionPane.showMessageDialog(this, data.toString());
    }

    else if (e.getSource() == btnDelete) {
        if (id.isEmpty()) {
            JOptionPane.showMessageDialog(this, "Enter Employee ID to delete!");
            return;
        }
        boolean removed = employees.removeIf(emp -> emp.id.equals(id));
        if (removed)
            JOptionPane.showMessageDialog(this, "Employee deleted successfully!");
        else
            JOptionPane.showMessageDialog(this, "Employee not found!");
        clearFields();
    }
}

```

```

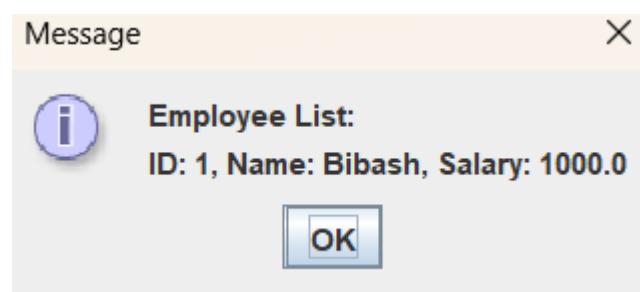
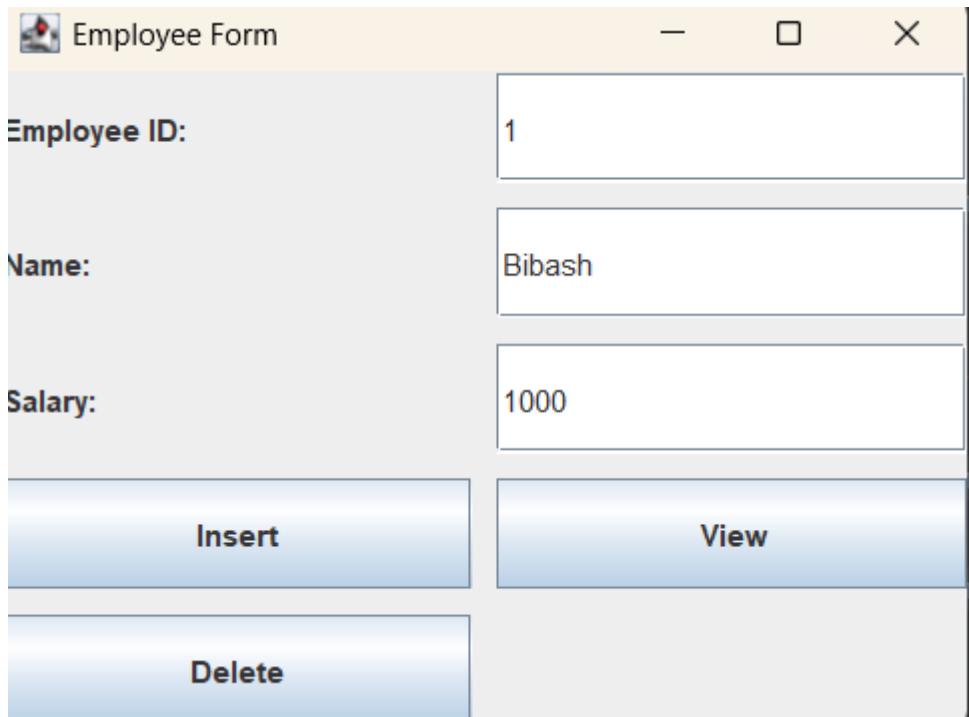
    }

private void clearFields() {
    txtId.setText("");
    txtName.setText("");
    txtSalary.setText("");
}

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

```

Output:



17. Define Bean property. What are its different types? Explain each property briefly.
A Bean property is an attribute of a JavaBean that can be read or written using standard naming conventions of accessor (getter) and mutator (setter) methods.

Example:

```
public class Employee {  
    private String name;  
  
    // Getter (Read method)  
    public String getName() {  
        return name;  
    }  
  
    // Setter (Write method)  
    public void setName(String name) {  
        this.name = name;  
    }  
}
```

Here, name is a bean property.

Types of Bean Properties:

1.Simple Property

Has a single value, can be read and written using getter/setter.

Example: getName(), setName(String name)

```
private int age;  
  
public int getAge() { return age; }  
public void setAge(int age) { this.age = age; }
```

2.Indexed Property

Holds multiple values (like an array or list), accessed using an index.

Example: getMarks(int index), setMarks(int index, int value)

```
private int marks[];  
  
public int[] getMarks() { return marks; }  
public void setMarks(int[] marks) { this.marks = marks; }  
  
public int getMarks(int index) { return marks[index]; }  
public void setMarks(int index, int value) { marks[index] = value; }
```

3.Bound Property

Notifies listeners when the property value changes. Used in GUI applications.

Example: Uses PropertyChangeSupport class

```
import java.beans.*;
```

```

public class Person {
    private String name;
    private PropertyChangeSupport support = new PropertyChangeSupport(this);

    public void addPropertyChangeListener(PropertyChangeListener listener) {
        support.addPropertyChangeListener(listener);
    }

    public void setName(String newName) {
        String oldName = this.name;
        this.name = newName;
        support.firePropertyChange("name", oldName, newName);
    }
}

```

18.What do you mean by action command? How it can be used? Explain with example and mention its advantages.

In Java (especially Swing and AWT), an Action Command is a string identifier assigned to a button or menu item to identify which action should be performed when that component is clicked.

It is mainly used when multiple buttons share the same ActionListener, so you can differentiate which button triggered the event.

Definition:

An Action Command is a string associated with an event source (like a JButton) that helps identify the source of the event in the actionPerformed() method.

How It Is Used:

1.Assign an action command to a button using:

```
button.setActionCommand("someCommand");
```

2.Inside the actionPerformed() method, use:

```
e.getActionCommand();
```

Example Program

```

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

```

```

public class ActionCommandExample extends JFrame implements ActionListener {
    JButton btnAdd, btnSub;

    public ActionCommandExample() {
        setTitle("Action Command Example");
        setSize(300, 150);
        setLayout(null);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }
}

```

```
btnAdd = new JButton("Add");
btnSub = new JButton("Subtract");

btnAdd.setBounds(50, 50, 80, 30);
btnSub.setBounds(150, 50, 100, 30);

// Set action commands
btnAdd.setActionCommand("add");
btnSub.setActionCommand("sub");

// Register same ActionListener for both buttons
btnAdd.addActionListener(this);
btnSub.addActionListener(this);

add(btnAdd);
add(btnSub);

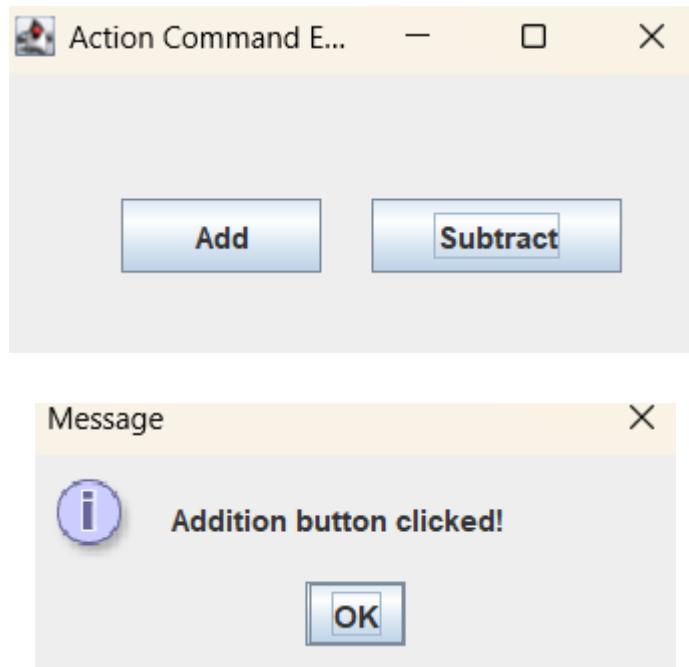
setVisible(true);
}

@Override
public void actionPerformed(ActionEvent e) {
    String command = e.getActionCommand();

    if (command.equals("add")) {
        JOptionPane.showMessageDialog(this, "Addition button clicked!");
    } else if (command.equals("sub")) {
        JOptionPane.showMessageDialog(this, "Subtraction button clicked!");
    }
}

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

Output:



Advantages of Using Action Commands

1. Code Reusability: Multiple buttons can share the same ActionListener.
2. Easier Maintenance: Avoids writing separate listeners for each button.
3. Better Readability: Each button action is identified by a meaningful string.
4. Flexibility: Easy to change button labels without affecting program logic (since the command is separate).

19.What is key event? How it is handled in Java? Explain with suitable Java source code.

A Key Event in Java occurs when the user presses, releases, or types a key on the keyboard while a component (like a text field or frame) has focus.

It is a part of Event Handling in the java.awt.event package.

Definition:

A KeyEvent is an event generated when a key on the keyboard is pressed, released, or typed.

It is handled using the KeyListener interface in Java.

How Key Events Are Handled in Java

To handle a key event:

1. Implement the KeyListener interface.
2. Register the component with addKeyListener().
3. Override the three methods:
 - o keyPressed()
 - o keyReleased()
 - o keyTyped()

```
Example Program: Key Event Handling
import javax.swing.*;
import java.awt.event.*;

public class KeyEventExample extends JFrame implements KeyListener {
    JTextField textField;
    JLabel label;

    public KeyEventExample() {
        setTitle("Key Event Example");
        setSize(400, 200);
        setLayout(null);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

        label = new JLabel("Type something:");
        label.setBounds(50, 40, 200, 30);

        textField = new JTextField();
        textField.setBounds(50, 80, 200, 30);

        // Register KeyListener to the text field
        textField.addKeyListener(this);

        add(label);
        add(textField);

        setVisible(true);
    }

    @Override
    public void keyTyped(KeyEvent e) {
        label.setText("Key Typed: " + e.getKeyChar());
    }

    @Override
    public void keyPressed(KeyEvent e) {
        label.setText("Key Pressed: " + e.getKeyChar());
    }

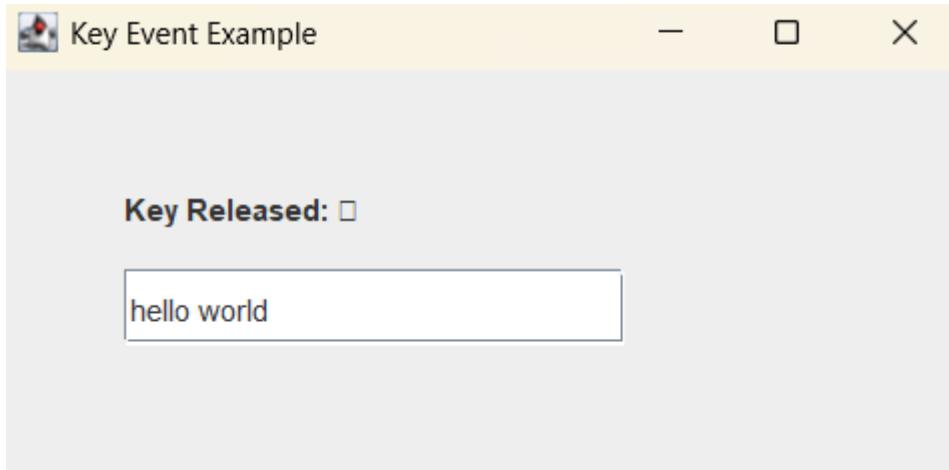
    @Override
    public void keyReleased(KeyEvent e) {
        label.setText("Key Released: " + e.getKeyChar());
    }
}
```

```

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

```

Output:



20. How modal dialog boxes are different from modeless dialog boxes? Explain creation of dialog boxes by using example.

Definition:

A Dialog Box in Java is a small window that communicates information to the user or gets input from them.

Dialogs are commonly created using the JDialog or JOptionPane class.

There are two types of dialog boxes:

- Modal Dialog Box
- Modeless Dialog Box

Comparison Table

| Basis | Model Dialog Box | Modeless Dialog Box |
|------------------|---|---|
| Definition | A dialog box that blocks interaction with other windows until it is closed. | A dialog box that allows interaction with other windows even when it is open. |
| User Interaction | The user must close it before returning to the main window. | The user can switch between the dialog and the main window freely. |
| Example Class | JDialog with true parameter or JOptionPane.showMessageDialog() | JDialog with false parameter |
| Use Case | Used for important tasks like confirmation or error messages. | Used for additional tools or small floating windows. |
| Example | Confirmation message ("Are you sure?") | Tool palette or help window |

```
Example Program Demonstrating Both

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

public class DialogExample extends JFrame implements ActionListener {
    JButton btnModal, btnModeless;

    public DialogExample() {
        setTitle("Dialog Box Example");
        setSize(400, 200);
        setLayout(null);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

        btnModal = new JButton("Open Modal Dialog");
        btnModeless = new JButton("Open Modeless Dialog");

        btnModal.setBounds(80, 50, 200, 30);
        btnModeless.setBounds(80, 100, 200, 30);

        btnModal.addActionListener(this);
        btnModeless.addActionListener(this);

        add(btnModal);
        add(btnModeless);

        setVisible(true);
    }

    @Override
    public void actionPerformed(ActionEvent e) {
        if (e.getSource() == btnModal) {
            // Create a Modal Dialog (true means modal)
            JDialog modalDialog = new JDialog(this, "Modal Dialog", true);
            modalDialog.setLayout(null);
            modalDialog.setSize(250, 150);

            JLabel lbl = new JLabel("This is a Modal Dialog");
            lbl.setBounds(40, 40, 200, 20);
            modalDialog.add(lbl);

            modalDialog.setVisible(true); // Blocks main window until closed
    }
}
```

```

if (e.getSource() == btnModeless) {
    // Create a Modeless Dialog (false means modeless)
    JDialog modelessDialog = new JDialog(this, "Modeless Dialog", false);
    modelessDialog.setLayout(null);
    modelessDialog.setSize(250, 150);

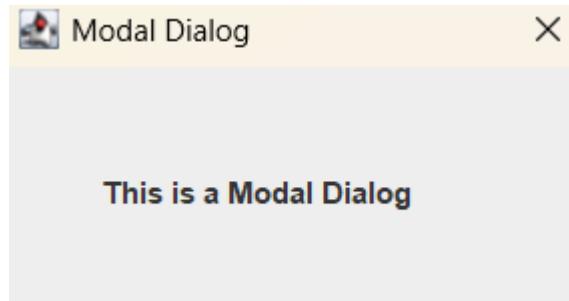
    JLabel lbl = new JLabel("This is a Modeless Dialog");
    lbl.setBounds(40, 40, 200, 20);
    modelessDialog.add(lbl);

    modelessDialog.setVisible(true); // Allows interaction with main window
}
}

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

```

Output:



21. How can we create open and save file dialog boxes? Explain with suitable example.

Definition:

Java provides a JFileChooser class (in javax.swing package) that allows the user to open or save files through a standard file dialog box.

You can use it to:

- Open (select) an existing file → Open Dialog
- Save (choose location and name) → Save Dialog

Class Used

`javax.swing.JFileChooser`

This class provides two main methods:

- `showOpenDialog(Component parent)` → Opens Open File dialog
- `showSaveDialog(Component parent)` → Opens Save File dialog

Common Methods of JFileChooser

| Method | Description |
|--|---|
| <code>showOpenDialog()</code> | Displays a dialog for selecting a file to open. |
| <code>showSaveDialog()</code> | Displays a dialog for selecting a file to save. |
| <code>getSelectedFile()</code> | Returns the file selected by the user. |
| <code>getCurrentDirectory()</code> | Returns the current directory. |
| <code>setCurrentDirectory(File dir)</code> | Sets the default directory. |
| <code>setFileSelectionMode()</code> | Restricts selection to files or directories. |

Example: Open and Save File Dialog Boxes

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

public class FileDialogExample extends JFrame implements ActionListener {
    JButton btnOpen, btnSave;
    JTextArea textArea;
    JFileChooser fileChooser;

    public FileDialogExample() {
        setTitle("Open and Save File Example");
        setSize(500, 400);
        setLayout(null);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

        btnOpen = new JButton("Open File");
        btnSave = new JButton("Save File");
        textArea = new JTextArea();
        fileChooser = new JFileChooser();

        btnOpen.setBounds(100, 20, 120, 30);
        btnSave.setBounds(250, 20, 120, 30);
        textArea.setBounds(50, 70, 380, 250);
```

```

        add(btnOpen);
        add(btnSave);
        add(textArea);

        btnOpen.addActionListener(this);
        btnSave.addActionListener(this);

        setVisible(true);
    }

    @Override
    public void actionPerformed(ActionEvent e) {
        if (e.getSource() == btnOpen) {
            int returnValue = fileChooser.showOpenDialog(this);

            if (returnValue == JFileChooser.APPROVE_OPTION) {
                File file = fileChooser.getSelectedFile();
                try (BufferedReader br = new BufferedReader(new FileReader(file))) {
                    textArea.setText("");
                    String line;
                    while ((line = br.readLine()) != null) {
                        textArea.append(line + "\n");
                    }
                } catch (IOException ex) {
                    JOptionPane.showMessageDialog(this, "Error reading file!");
                }
            }
        }

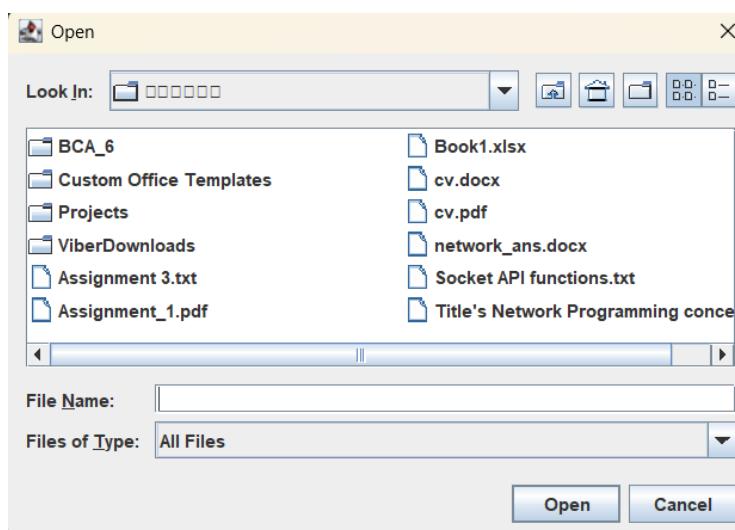
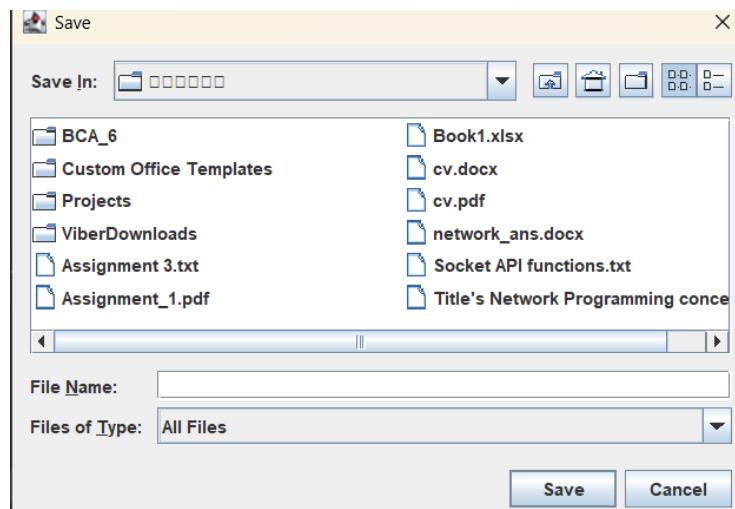
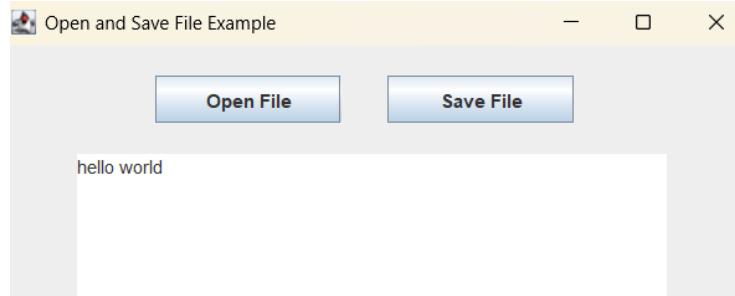
        if (e.getSource() == btnSave) {
            int returnValue = fileChooser.showSaveDialog(this);

            if (returnValue == JFileChooser.APPROVE_OPTION) {
                File file = fileChooser.getSelectedFile();
                try (BufferedWriter bw = new BufferedWriter(new FileWriter(file))) {
                    bw.write(textArea.getText());
                    JOptionPane.showMessageDialog(this, "File saved successfully!");
                } catch (IOException ex) {
                    JOptionPane.showMessageDialog(this, "Error saving file!");
                }
            }
        }
    }
}

```

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

Output:



21.What is the use of multiple results set? How it can be used? Explain with suitable Java code.

Definition:

A multiple result set occurs when a single SQL statement (often a stored procedure or batch query) returns more than one set of results.

For example:

- A stored procedure might return a list of employees and then a list of departments.
- In JDBC, this can be handled using the Statement or CallableStatement object and the getMoreResults() method.

Why Use Multiple Result Sets?

- To reduce database calls (run several queries in one go).
- To improve performance by retrieving multiple tables or reports with a single command.
- Commonly used in stored procedures that return multiple tables of data.

How to Handle Multiple Result Sets

When executing a query that returns multiple result sets, we can:

1. Execute the SQL using execute() method.
2. Retrieve the first result set using getResultSet().
3. Use a loop with getMoreResults() to process additional result sets.

Example Java Program

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

```
public class MultipleResultSetExample {  
    public static void main(String[] args) {  
        String url = "jdbc:mysql://localhost:3306/testdb";  
        String user = "root";  
        String password = "";  
  
        try (Connection con = DriverManager.getConnection(url, user, password);  
             Statement stmt = con.createStatement()) {  
  
            // Example SQL that returns two result sets  
            String sql = "SELECT * FROM employees; SELECT * FROM departments;";  
  
            boolean hasResults = stmt.execute(sql);  
  
            // Loop through multiple result sets  
            while (hasResults) {  
                ResultSet rs = stmt.getResultSet();  
                ResultSetMetaData rsmd = rs.getMetaData();  
  
                int columnCount = rsmd.getColumnCount();  
                System.out.println("---- New Result Set ----");  
            }  
        }  
    }  
}
```

```

// Print each row of current result set
while (rs.next()) {
    for (int i = 1; i <= columnCount; i++) {
        System.out.print(rs.getString(i) + "\t");
    }
    System.out.println();
}

// Move to the next result set
hasResults = stmt.getMoreResults();
}

} catch (SQLException e) {
    e.printStackTrace();
}
}
}
}

```

23.Explain the concept behind SQL escape, connection pooling and LOB's with brief description.

SQL Escape

Concept:

SQL Escape sequences are special syntax elements used in JDBC (Java Database Connectivity) to make SQL queries database-independent.

They allow Java programs to include special functions (like dates, times, and outer joins) in SQL statements in a uniform way.

Common SQL Escape Sequences:

| Escape Sequence | Description | Example |
|----------------------------|-----------------------|----------------------------|
| {d 'yyyy-mm-dd'} | Date literal | {d '2025-11-09'} |
| {t 'hh:mm:ss'} | Time literal | {t '10:30:00'} |
| {ts 'yyyy-mm-dd hh:mm:ss'} | Timestamp literal | {ts '2025-11-09 10:30:00'} |
| {fn function_name(args)} | Function call | {fn UCASE(name)} |
| {call procedure_name()} | Call stored procedure | {call getEmployeeList()} |

Example:

Statement stmt = con.createStatement();

ResultSet rs = stmt.executeQuery(

"SELECT * FROM employees WHERE hire_date > {d '2020-01-01'}");

Connection Pooling

Concept:

Connection pooling is a technique to reuse database connections instead of creating a new one each time a request is made.

- Creating a DB connection is time-consuming.
- A connection pool keeps several active connections ready.
- When an application needs one, it borrows it from the pool.
- After use, it returns it back to the pool for reuse.

Advantages:

- Faster database access.
- Reduces resource consumption.
- Increases scalability and performance.

Example (Using DataSource):

```
import javax.sql.DataSource;  
import org.apache.commons.dbcp2.BasicDataSource;
```

```
public class ConnectionPoolDemo {  
    public static void main(String[] args) throws Exception {  
        BasicDataSource ds = new BasicDataSource();  
        ds.setUrl("jdbc:mysql://localhost:3306/testdb");  
        ds.setUsername("root");  
        ds.setPassword("password");  
        ds.setMinIdle(2);  
        ds.setMaxIdle(5);  
  
        // Get connection from pool  
        java.sql.Connection con = ds.getConnection();  
        System.out.println("Connection obtained from pool!");  
  
        con.close(); // Returns connection to pool  
    }  
}
```

LOBs (Large Objects)

Concept:

LOBs are used to store large binary or character data such as:

- Images
- Audio files
- PDFs
- Long text documents

There are two main types:

- BLOB (Binary Large Object) → For binary data (images, videos)
- CLOB (Character Large Object) → For character data (text, documents)

Example (Inserting Image as BLOB):

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

```

import java.io.*;

public class InsertImage {
    public static void main(String[] args) throws Exception {
        Connection con = DriverManager.getConnection(
            "jdbc:mysql://localhost:3306/testdb", "root", "password");

        String query = "INSERT INTO employees (name, photo) VALUES (?, ?)";
        PreparedStatement ps = con.prepareStatement(query);
        ps.setString(1, "John");

        FileInputStream fis = new FileInputStream("john_photo.jpg");
        ps.setBinaryStream(2, fis, (int)new File("john_photo.jpg").length());
        ps.executeUpdate();

        System.out.println("Image inserted successfully!");
        con.close();
    }
}

```

24.What is bean writing process? Explain the steps used in writing a bean with example.

A JavaBean is a reusable Java class that follows certain conventions. The Bean writing process refers to the steps followed to create a JavaBean, so it can be used in other applications or GUI builders.

Steps to Write a JavaBean

Step 1: Create a Public Class

- The class should be public.
- The class name should be meaningful.

```
public class EmployeeBean {
```

```
    // Class body
```

```
}
```

Step 2: Make Properties Private

- Properties (fields) should be private to follow encapsulation.

```
private String name;
```

```
private int age;
```

```
private double salary;
```

Step 3: Provide Public Getter and Setter Methods

- These are accessor and mutator methods.
- Naming convention:
 - Getter: getPropertyName()
 - Setter: setPropertyName(value)

```
public String getName() {
```

```
    return name;
```

```
}
```

```
public void setName(String name) {  
    this.name = name;  
}
```

```
public int getAge() {  
    return age;  
}
```

```
public void setAge(int age) {  
    this.age = age;  
}
```

```
public double getSalary() {  
    return salary;  
}
```

```
public void setSalary(double salary) {  
    this.salary = salary;  
}
```

Step 4: Provide a No-Argument Constructor

- JavaBeans should have a public default constructor.

```
public EmployeeBean() {  
    // Default constructor  
}
```

Step 5: (Optional) Implement Serializable

- If the bean needs to be saved or transmitted, implement Serializable.

```
import java.io.Serializable;
```

```
public class EmployeeBean implements Serializable {  
    // properties, getter, setter  
}
```

Step 6: Use the Bean in Another Program

```
public class TestBean {  
    public static void main(String[] args) {  
        EmployeeBean emp = new EmployeeBean();  
        emp.setName("John");  
        emp.setAge(30);  
        emp.setSalary(50000.0);
```

```
        System.out.println("Name: " + emp.getName());  
        System.out.println("Age: " + emp.getAge());
```

```

        System.out.println("Salary: " + emp.getSalary());
    }
}

```

25.What is action event? How it is handled in Java? Explain with suitable Java source code.

Definition:

An ActionEvent occurs when a user performs an action on a GUI component, such as:

- Clicking a button (JButton)
- Pressing Enter in a text field (JTextField)
- Selecting a menu item (JMenuItem)

ActionEvent is part of the java.awt.event package.

Key Points:

- Generated by components that support user actions.
- Handled using the ActionListener interface.
- The event provides information about the source of the action.

How Action Events Are Handled

Steps:

1. Implement ActionListener Interface
 - Contains a single method:
void actionPerformed(ActionEvent e);
2. Register the Listener with Component
 - Use addActionListener() method.
3. Write Action Code
 - Inside actionPerformed(), define what happens when the action occurs.

Example: Handling ActionEvent with Buttons

```

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

public class ActionEventExample extends JFrame implements ActionListener {
    JButton btnHello, btnExit;

    public ActionEventExample() {
        setTitle("Action Event Example");
        setSize(300, 150);
        setLayout(null);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

        btnHello = new JButton("Say Hello");
        btnExit = new JButton("Exit");

        btnHello.setBounds(50, 50, 100, 30);
    }

    public void actionPerformed(ActionEvent e) {
        if (e.getSource() == btnHello) {
            System.out.println("Hello Clicked");
        } else if (e.getSource() == btnExit) {
            System.out.println("Exit Clicked");
            System.exit(0);
        }
    }
}

```

```

btnExit.setBounds(160, 50, 80, 30);

// Register ActionListener
btnHello.addActionListener(this);
btnExit.addActionListener(this);

add(btnHello);
add(btnExit);

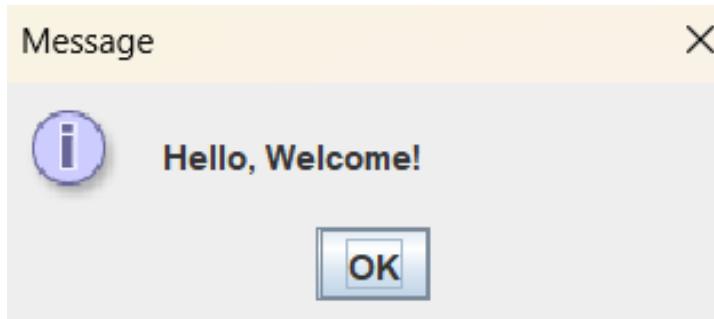
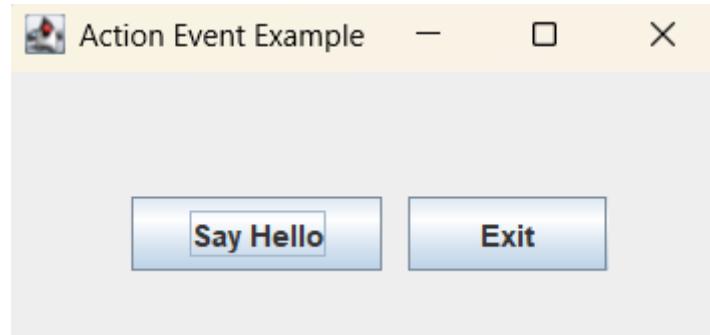
setVisible(true);
}

@Override
public void actionPerformed(ActionEvent e) {
    if (e.getSource() == btnHello) {
        JOptionPane.showMessageDialog(this, "Hello, Welcome!");
    } else if (e.getSource() == btnExit) {
        System.exit(0);
    }
}

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

```

Output:



26.Prepare a program with three text boxes. First number, second number and result and four buttons add, subtract, multiply and divide. Handle the events to perform the required operations and display result.

Java Program: Simple Calculator Using Swing

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

public class SimpleCalculator extends JFrame implements ActionListener {

    JTextField txtNum1, txtNum2, txtResult;
    JButton btnAdd, btnSub, btnMul, btnDiv;

    public SimpleCalculator() {
        // Frame title and layout
        setTitle("Simple Calculator");
        setLayout(new GridLayout(5, 2, 10, 10));

        // Labels
        add(new JLabel("First Number:"));
        txtNum1 = new JTextField();
        add(txtNum1);

        add(new JLabel("Second Number:"));
        txtNum2 = new JTextField();
        add(txtNum2);

        add(new JLabel("Result:"));
        txtResult = new JTextField();
        txtResult.setEditable(false);
        add(txtResult);

        // Buttons
        btnAdd = new JButton("Add");
        btnSub = new JButton("Subtract");
        btnMul = new JButton("Multiply");
        btnDiv = new JButton("Divide");

        // Add buttons to frame
        add(btnAdd);
        add(btnSub);
        add(btnMul);
        add(btnDiv);
    }

    public void actionPerformed(ActionEvent e) {
        if (e.getSource() == btnAdd) {
            int num1 = Integer.parseInt(txtNum1.getText());
            int num2 = Integer.parseInt(txtNum2.getText());
            txtResult.setText(String.valueOf(num1 + num2));
        }
        else if (e.getSource() == btnSub) {
            int num1 = Integer.parseInt(txtNum1.getText());
            int num2 = Integer.parseInt(txtNum2.getText());
            txtResult.setText(String.valueOf(num1 - num2));
        }
        else if (e.getSource() == btnMul) {
            int num1 = Integer.parseInt(txtNum1.getText());
            int num2 = Integer.parseInt(txtNum2.getText());
            txtResult.setText(String.valueOf(num1 * num2));
        }
        else if (e.getSource() == btnDiv) {
            int num1 = Integer.parseInt(txtNum1.getText());
            int num2 = Integer.parseInt(txtNum2.getText());
            txtResult.setText(String.valueOf(num1 / num2));
        }
    }
}
```

```

// Register event listeners
btnAdd.addActionListener(this);
btnSub.addActionListener(this);
btnMul.addActionListener(this);
btnDiv.addActionListener(this);

// Frame settings
setSize(350, 250);
setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
setVisible(true);
}

@Override
public void actionPerformed(ActionEvent e) {
    try {
        double num1 = Double.parseDouble(txtNum1.getText());
        double num2 = Double.parseDouble(txtNum2.getText());
        double result = 0;

        if (e.getSource() == btnAdd) {
            result = num1 + num2;
        } else if (e.getSource() == btnSub) {
            result = num1 - num2;
        } else if (e.getSource() == btnMul) {
            result = num1 * num2;
        } else if (e.getSource() == btnDiv) {
            if (num2 == 0) {
                JOptionPane.showMessageDialog(this, "Cannot divide by zero!");
                return;
            }
            result = num1 / num2;
        }

        txtResult.setText(String.valueOf(result));
    } catch (NumberFormatException ex) {
        JOptionPane.showMessageDialog(this, "Please enter valid numbers!");
    }
}

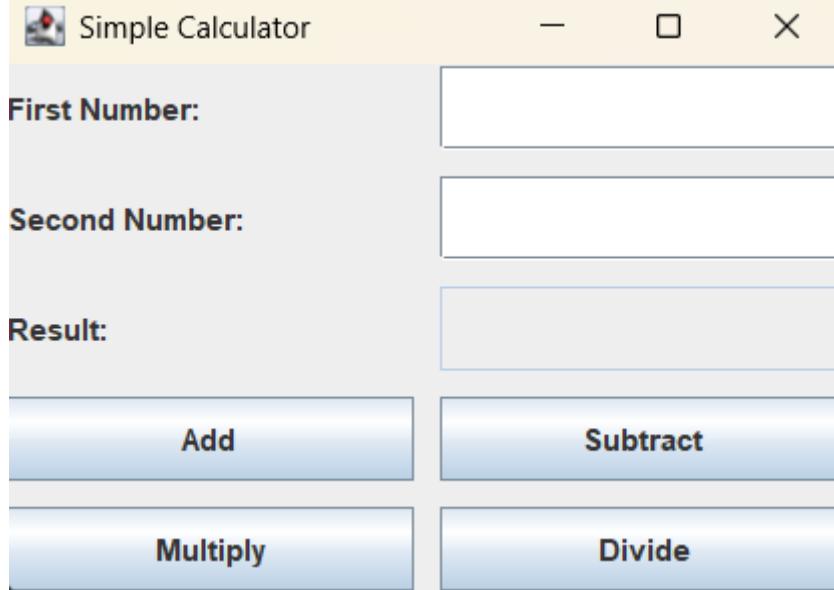
public static void main(String[] args) {
    new SimpleCalculator();
}

```

```
}
```

```
}
```

Output:



27. Write a program to create a form to enter employee data like employee id, name, DOB, salary, gender, country, remarks and two buttons add and cancel by using frame, panels and appropriate components.

Java Program: Employee Data Entry Form

```
import javax.swing.*;
```

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

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

```
public class EmployeeForm extends JFrame implements ActionListener {
```

```
    // Form components
```

```
    JTextField txtId, txtName, txtDOB, txtSalary;
```

```
    JRadioButton rbMale, rbFemale;
```

```
    JComboBox<String> cbCountry;
```

```
    JTextArea txtRemarks;
```

```
    JButton btnAdd, btnCancel;
```

```
    public EmployeeForm() {
```

```
        setTitle("Employee Data Entry Form");
```

```
        setSize(400, 400);
```

```
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
```

```
        setLayout(new BorderLayout(10, 10));
```

```
        // ----- PANEL 1: Form Fields -----
```

```
        JPanel formPanel = new JPanel(new GridLayout(7, 2, 8, 8));
```

```

// Employee ID
formPanel.add(new JLabel("Employee ID:"));
txtId = new JTextField();
formPanel.add(txtId);

// Employee Name
formPanel.add(new JLabel("Name:"));
txtName = new JTextField();
formPanel.add(txtName);

// Date of Birth
formPanel.add(new JLabel("Date of Birth:"));
txtDOB = new JTextField("YYYY-MM-DD");
formPanel.add(txtDOB);

// Salary
formPanel.add(new JLabel("Salary:"));
txtSalary = new JTextField();
formPanel.add(txtSalary);

// Gender
formPanel.add(new JLabel("Gender:"));
JPanel genderPanel = new JPanel();
rbMale = new JRadioButton("Male");
rbFemale = new JRadioButton("Female");
ButtonGroup bgGender = new ButtonGroup();
bgGender.add(rbMale);
bgGender.add(rbFemale);
genderPanel.add(rbMale);
genderPanel.add(rbFemale);
formPanel.add(genderPanel);

// Country
formPanel.add(new JLabel("Country:"));
String countries[] = {"Nepal", "India", "USA", "UK", "China"};
cbCountry = new JComboBox(countries);
formPanel.add(cbCountry);

// Remarks
formPanel.add(new JLabel("Remarks:"));
txtRemarks = new JTextArea(3, 20);
formPanel.add(new JScrollPane(txtRemarks));

```

```

        add(formPanel, BorderLayout.CENTER);

        // ----- PANEL 2: Buttons -----
        JPanel buttonPanel = new JPanel();
        btnAdd = new JButton("Add");
        btnCancel = new JButton("Cancel");
        buttonPanel.add(btnAdd);
        buttonPanel.add(btnCancel);

        add(buttonPanel, BorderLayout.SOUTH);

        // ----- Action Listeners -----
        btnAdd.addActionListener(this);
        btnCancel.addActionListener(this);

        setVisible(true);
    }

    @Override
    public void actionPerformed(ActionEvent e) {
        if (e.getSource() == btnAdd) {
            // Collect data from form
            String data = "Employee ID: " + txtId.getText() +
                "\nName: " + txtName.getText() +
                "\nDOB: " + txtDOB.getText() +
                "\nSalary: " + txtSalary.getText() +
                "\nGender: " + (rbMale.isSelected() ? "Male" : rbFemale.isSelected() ? "Female"
                : "Not Selected") +
                "\nCountry: " + cbCountry.getSelectedItem() +
                "\nRemarks: " + txtRemarks.getText();

            JOptionPane.showMessageDialog(this, "Employee Added Successfully!\n\n" + data);
        }
        else if (e.getSource() == btnCancel) {
            // Clear all fields
            txtId.setText("");
            txtName.setText("");
            txtDOB.setText("");
            txtSalary.setText("");
            txtRemarks.setText("");
            cbCountry.setSelectedIndex(0);
            rbMale.setSelected(false);
        }
    }
}

```

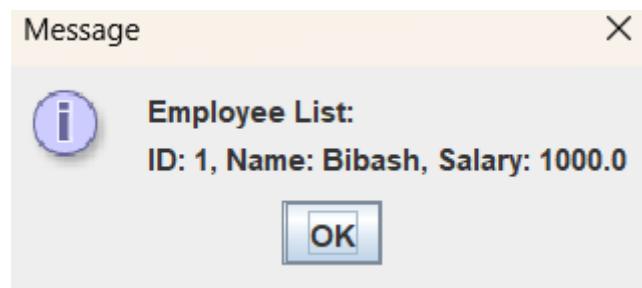
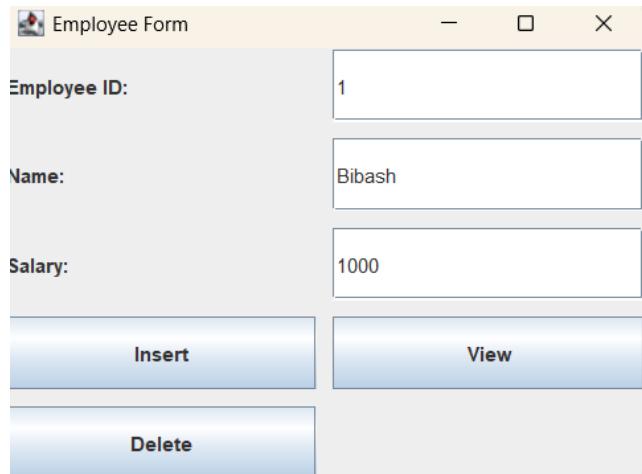
```

        rbFemale.setSelected(false);
    }
}

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

```

Output:



28.Explain creation of menu bar, menu and menu items with suitable Java code segments.

Steps to Create a Menu Bar

Step 1: Create a JMenuBar object.

```
JMenuBar menuBar = new JMenuBar();
```

Step 2: Create one or more JMenu objects (like File, Edit).

```
JMenu fileMenu = new JMenu("File");
```

```
JMenu editMenu = new JMenu("Edit");
```

Step 3: Create JMenuItem objects for each menu.

```
JMenuItem newItem = new JMenuItem("New");
```

```
JMenuItem openItem = new JMenuItem("Open");
```

```
JMenuItem exitItem = new JMenuItem("Exit");
```

Step 4: Add menu items to their corresponding menus.

```
fileMenu.add(newItem);
```

```
    fileMenu.add(openItem);
    fileMenu.addSeparator(); // adds a horizontal line
    fileMenu.add(exitItem);
```

Step 5: Add menus to the menu bar.

```
    menuBar.add(fileMenu);
    menuBar.add(editMenu);
```

Step 6: Attach the menu bar to the frame.

```
    setJMenuBar(menuBar);
```

Complete Example Code

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

```
public class MenuExample extends JFrame implements ActionListener {
```

```
    JMenuBar menuBar;
    JMenu fileMenu, editMenu;
    JMenuItem newItem, openItem, exitItem, copyItem, pasteItem;
```

```
    public MenuExample() {
        setTitle("Menu Bar Example");
        setSize(400, 300);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
```

```
        // Step 1: Create menu bar
```

```
        menuBar = new JMenuBar();
```

```
        // Step 2: Create menus
```

```
        fileMenu = new JMenu("File");
        editMenu = new JMenu("Edit");
```

```
        // Step 3: Create menu items
```

```
        newItem = new JMenuItem("New");
        openItem = new JMenuItem("Open");
        exitItem = new JMenuItem("Exit");
```

```
        copyItem = new JMenuItem("Copy");
        pasteItem = new JMenuItem("Paste");
```

```
        // Step 4: Add menu items to menus
```

```
        fileMenu.add(newItem);
        fileMenu.add(openItem);
        fileMenu.addSeparator(); // adds a divider line
        fileMenu.add(exitItem);
```

```

editMenu.add(copyItem);
editMenu.add(pasteItem);

// Step 5: Add menus to menu bar
menuBar.add(fileMenu);
menuBar.add(editMenu);

// Step 6: Attach the menu bar to frame
setJMenuBar(menuBar);

// Step 7: Add action listeners
exitItem.addActionListener(this);

setVisible(true);
}

@Override
public void actionPerformed(ActionEvent e) {
    if (e.getSource() == exitItem) {
        JOptionPane.showMessageDialog(this, "Exiting the program...");
        System.exit(0);
    }
}

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

Output:

```



29.Define row set and cached row set. Explain both of them with suitable example.

Definition:

A RowSet is an advanced version of a ResultSet that provides a scrollable, updatable, and disconnected mechanism for handling tabular data from a database.

Features of RowSet

- Can be scrollable (move forward & backward).
- Can be updatable.
- Can work even when not connected to the database.
- Supports JavaBeans properties and event listeners.

Types of RowSet

1. JdbcRowSet → connected RowSet (always connected to DB).
2. CachedRowSet → disconnected RowSet (works offline).
3. WebRowSet, FilteredRowSet, JoinRowSet → extended versions of CachedRowSet.

Example: JdbcRowSet

```
import javax.sql.rowset.JdbcRowSet;
import javax.sql.rowset.RowSetProvider;

public class JdbcRowSetExample {
    public static void main(String[] args) {
        try {
            // Create JdbcRowSet object
            JdbcRowSet rowSet = RowSetProvider.newFactory().createJdbcRowSet();

            // Set connection properties
            rowSet.setUrl("jdbc:mysql://localhost:3306/testdb");
            rowSet.setUsername("root");
            rowSet.setPassword("password");

            // Set SQL command
            rowSet.setCommand("SELECT * FROM employees");
            rowSet.execute();

            // Display data
            while (rowSet.next()) {
                System.out.println("ID: " + rowSet.getInt("id") +
                    ", Name: " + rowSet.getString("name"));
            }

            rowSet.close();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

```
    }  
}
```

30.What is transaction? How can it be used with JDBC? Explain with suitable Java program.

Definition:

A transaction is a set of one or more SQL operations that are executed as a single logical unit of work.

Either all operations succeed (commit) or none of them are applied (rollback).

Transaction Handling in JDBC

By default, JDBC runs in auto-commit mode, meaning each SQL statement is committed automatically.

To manage transactions manually:

1. Disable auto-commit → conn.setAutoCommit(false);
2. Execute SQL statements.
3. If successful → conn.commit();
4. If any error → conn.rollback();

Transaction Example: Bank Transfer

```
import java.sql.*;  
  
public class TransactionExample {  
    public static void main(String[] args) {  
        Connection conn = null;  
        PreparedStatement withdraw = null;  
        PreparedStatement deposit = null;  
  
        try {  
            // 1. Connect to database  
            conn = DriverManager.getConnection(  
                "jdbc:mysql://localhost:3306/bankdb", "root", "password");  
  
            // 2. Disable auto-commit for transaction control  
            conn.setAutoCommit(false);  
  
            // 3. SQL statements  
            String withdrawSQL = "UPDATE accounts SET balance = balance - ? WHERE  
acc_no = ?";  
            String depositSQL = "UPDATE accounts SET balance = balance + ? WHERE acc_no  
= ?";  
  
            withdraw = conn.prepareStatement(withdrawSQL);  
        }
```

```
deposit = conn.prepareStatement(depositSQL);

// Withdraw Rs.1000 from account 101
withdraw.setDouble(1, 1000);
withdraw.setInt(2, 101);
withdraw.executeUpdate();

// Deposit Rs.1000 into account 102
deposit.setDouble(1, 1000);
deposit.setInt(2, 102);
deposit.executeUpdate();

// 4. Commit if all successful
conn.commit();
System.out.println("Transaction successful! Amount transferred.");

} catch (Exception e) {
    try {
        if (conn != null) {
            // 5. Rollback if any error occurs
            conn.rollback();
            System.out.println("Transaction failed! Rolled back.");
        }
    } catch (SQLException se) {
        se.printStackTrace();
    }
    e.printStackTrace();
} finally {
    try {
        if (withdraw != null) withdraw.close();
        if (deposit != null) deposit.close();
        if (conn != null) conn.close();
    } catch (SQLException e) {
        e.printStackTrace();
    }
}
}
```

31.What is BDK and Bean Box? Explain the process of using Bean Box.

Definition:

BDK (Bean Development Kit) is a development environment provided by Sun Microsystems (now Oracle) to create, test, and configure JavaBeans.

It provides a set of tools and utilities to help developers understand how JavaBeans work.

Definition:

BeanBox is a test container included in the BDK that allows developers to:

- Load beans visually.
- Connect beans together.
- Modify their properties.
- Test event handling between beans.

Process of Using BeanBox

1. Create a Bean

- Write a simple Java class following JavaBean conventions:
 - Must have a public no-argument constructor
 - Must be serializable
 - Use getter/setter methods for properties

Example Bean (MessageBean.java):

```
import java.io.Serializable;
```

```
public class MessageBean implements Serializable {  
    private String message;  
  
    public MessageBean() {  
        message = "Hello from JavaBean!";  
    }  
  
    public String getMessage() {  
        return message;  
    }  
  
    public void setMessage(String message) {  
        this.message = message;  
    }  
  
    public void showMessage() {  
        System.out.println(message);  
    }  
}
```

2. Compile the Bean

```
javac MessageBean.java
```

3. Create a JAR File

Package the class into a JAR file:

```
jar cf MessageBean.jar MessageBean.class
```

4. Copy the JAR File

- Place the JAR file inside:

- BDK/beanbox/jars/

5.Run BeanBox

- Open command prompt and navigate to the BeanBox directory:
- cd BDK\beanbox
- run.bat
- The BeanBox GUI will appear.

6.Load the Bean

- Use File → Load Jar option to load your MessageBean.jar.
- Then from Toolbox → Add Bean, select your bean.

7.Test and Configure

- Drag your bean into the BeanBox workspace.
- Modify its properties in the Properties window.
- If your bean has events or methods, you can test them directly.

32.What are different paces for putting event handling code? Explain with suitable Java code. What is inner class? Explain its importance in event handling with example of your interest.

Event Handling in Java

In Java GUI programming (Swing or AWT), event handling means writing code that responds to user actions — such as:

- Button clicks
- Mouse movements
- Key presses

Events are handled by event listeners (interfaces like ActionListener, MouseListener, etc.) which define what should happen when an event occurs.

Different Places to Put Event Handling Code

There are three main ways (places) to put event-handling code in Java:

| Approach | Description |
|-------------------------------|--|
| 1. Separate Listener Class | Create a separate class that implements the listener interface. |
| 2. Anonymous Inner Class | Write the listener code directly inside component registration using an unnamed inner class. |
| 3. Inner Class (Member Class) | Create an inner class inside the main class to handle events. |

(a) Separate Listener Class Example

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

```
class MyListener implements ActionListener {
    public void actionPerformed(ActionEvent e) {
        System.out.println("Button clicked!");
    }
}
```

```

}

public class SeparateClassExample {
    public static void main(String[] args) {
        Frame f = new Frame("Separate Listener Example");
        Button b = new Button("Click Me");
        b.addActionListener(new MyListener()); // register listener

        f.add(b);
        f.setSize(200, 150);
        f.setLayout(new FlowLayout());
        f.setVisible(true);
    }
}

```

Output:



Advantage:

- Code separation (clean, reusable).

Disadvantage:

- Not compact (listener far from the component).

(b) Anonymous Inner Class Example

```

import java.awt.*;
import java.awt.event.*;

```

```

public class AnonymousClassExample {
    public static void main(String[] args) {
        Frame f = new Frame("Anonymous Inner Class Example");
        Button b = new Button("Click Here");

        // Listener written inline using anonymous class
        b.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                System.out.println("Button clicked using anonymous class!");
            }
        });

        f.add(b);
        f.setSize(250, 150);
        f.setLayout(new FlowLayout());
    }
}

```

```
    f.setVisible(true);
}
}

Output:
```



Advantage:

- Compact, keeps code close to where it's used.

Disadvantage:

- Harder to reuse and maintain.

(c) Inner Class Example

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

```
public class InnerClassExample extends Frame {
    Button b;

    public InnerClassExample() {
        b = new Button("Click Me");
        b.addActionListener(new MyHandler()); // Register inner class listener

        add(b);
        setSize(200, 150);
        setLayout(new FlowLayout());
        setVisible(true);
    }

    // Inner class for event handling
    class MyHandler implements ActionListener {
        public void actionPerformed(ActionEvent e) {
            System.out.println("Button clicked using inner class!");
        }
    }

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

Output:



Advantage:

- Inner class has direct access to outer class members.
- Cleaner and easier for medium-size GUI programs.

What is an Inner Class?

Definition:

An inner class is a class defined inside another class.

It can access all private and protected members of the outer class directly.

Syntax:

```
class Outer {  
    private int x = 10;  
  
    class Inner {  
        void display() {  
            System.out.println("x = " + x); // can access outer class variable  
        }  
    }  
}
```

Importance of Inner Class in Event Handling

| Benefit | Description |
|----------------------------------|--|
| 1. Access to outer class members | Inner class can directly access GUI components (like text fields, labels) declared in outer class. |
| 2. Logical grouping | Keeps event-handling code close to the component logic. |
| 3. Code organization | Makes the code easier to maintain and understand. |

Example: Inner Class for Button Event

```
import java.awt.*;  
import java.awt.event.*;  
  
public class CalculatorDemo extends Frame {  
    TextField t1, t2, result;  
    Button add;  
  
    public CalculatorDemo() {  
        setLayout(new FlowLayout());
```

```

t1 = new TextField(5);
t2 = new TextField(5);
result = new TextField(10);
add = new Button("Add");

add(t1); add(t2); add(add); add(result);

add.addActionListener(new AddHandler()); // Inner class

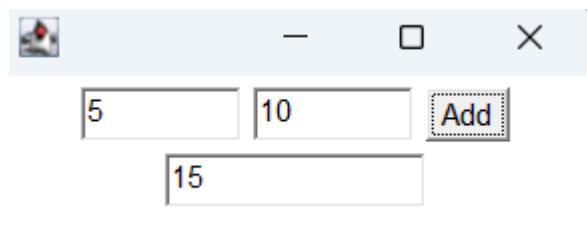
setSize(250, 150);
setVisible(true);
}

// Inner class for handling event
class AddHandler implements ActionListener {
    public void actionPerformed(ActionEvent e) {
        int a = Integer.parseInt(t1.getText());
        int b = Integer.parseInt(t2.getText());
        result.setText(String.valueOf(a + b));
    }
}

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

```

Output:



Why inner class is useful here:

The inner class directly accesses the text fields (t1, t2, result) of the outer class — no need to pass them as arguments.

33. Differentiate between frames and panels. Explain different SWT components briefly.

Difference Between Frames and Panels

| Basis | Frame | Panel |
|--------------------|---|---|
| Definition | A Frame is a top-level container (window) that can hold other components. | A Panel is a container used to group and organize components inside another container (like Frame). |
| Class | Defined in java.awt.Frame | Defined in java.awt.Panel |
| Parent class | Inherits from Window (which extends Container) | Directly inherits from Container |
| Title Bar / Border | Has a title bar, border, minimize/maximize/close buttons. | No title bar or border, just an area inside the window. |
| Usage | Used to create main application windows. | Used to organize layout or sections inside a frame. |

Example Demonstrating Frame and Panel

```
import java.awt.*;

public class FramePanelExample {

    public static void main(String[] args) {

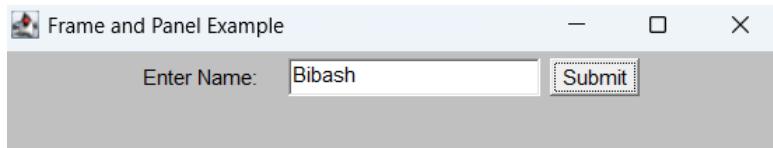
        // Create a Frame
        Frame f = new Frame("Frame and Panel Example");

        // Create a Panel
        Panel p = new Panel();
        p.setBackground(Color.LIGHT_GRAY);
        p.add(new Label("Enter Name:"));
        p.add(new TextField(15));
        p.add(new Button("Submit"));

        // Add panel to frame
        f.add(p);
        f.setSize(300, 200);
        f.setVisible(true);

    }
}
```

Output:



What is SWT (Standard Widget Toolkit)?

Definition:

SWT (Standard Widget Toolkit) is a GUI toolkit developed by Eclipse Foundation as an alternative to AWT/Swing.

It provides native OS look-and-feel, meaning it uses the operating system's GUI components directly instead of drawing them itself.

Key Features of SWT

| Feature | Description |
|----------------------|--|
| Native performance | Uses OS's GUI widgets for faster, native-looking interfaces. |
| Platform independent | Works across Windows, macOS, and Linux. |
| Used in Eclipse IDE | The Eclipse GUI is built using SWT. |
| Event-driven | Like AWT/Swing, it uses listeners and events. |

Common SWT Components

| Component | Description | Example Usage |
|-----------|---|-----------------------------------|
| Display | Manages the connection between SWT and the OS GUI system. | Acts as main event loop handler. |
| Shell | Equivalent to a Frame or Window in AWT/Swing. | Main window container. |
| Composite | Acts like a Panel; used to group widgets together. | Organize multiple controls. |
| Label | Displays non-editable text or images. | Showing captions or descriptions. |
| Text | Used for text input or display. | Entering user data. |
| Button | Push, checkbox, or radio button widget. | Trigger user actions. |

| | | |
|----------------------|---|-------------------------------------|
| List / Combo | Display list of items (selectable). | Drop-downs, selection lists. |
| Table / Tree | Displays structured or hierarchical data. | Showing data sets or folders. |
| Menu & MenuItem | Used to create application menus. | File, Edit, Help menus. |
| Canvas | Area for custom graphics or drawing. | Creating charts, shapes, etc. |
| TabFolder / TabItem | Used to create tabbed interfaces. | Multiple tabs like browser windows. |
| ProgressBar / Slider | Shows progress or allows selection of numeric values. | Upload progress, volume control. |

Example of SWT Program

```

import org.eclipse.swt.*;
import org.eclipse.swt.widgets.*;

public class SWTEexample {
    public static void main(String[] args) {
        Display display = new Display(); // Manages OS interaction
        Shell shell = new Shell(display); // Main window
        shell.setText("SWT Example");
        shell.setSize(300, 200);

        // Create widgets
        Label label = new Label(shell, SWT.NONE);
        label.setText("Enter Name:");
        label.setBounds(20, 30, 80, 20);

        Text text = new Text(shell, SWT.BORDER);
        text.setBounds(110, 30, 150, 20);
    }
}
```

```

Button button = new Button(shell, SWT.PUSH);
button.setText("Submit");
button.setBounds(100, 70, 80, 25);

shell.open();
while (!shell.isDisposed()) {
    if (!display.readAndDispatch())
        display.sleep();
}
display.dispose();
}
}

```

34.Explain grid layout manager with suitable constructors and demonstrate it through sample code.

What is GridLayout Manager?

Definition:

The GridLayout manager in Java arranges components in a rectangular grid (rows × columns).

Each cell in the grid is of equal size, and each component occupies exactly one cell.

Characteristics of GridLayout

| Feature | Description |
|------------------------|---|
| Equal cells | All cells have the same width and height. |
| Row-column arrangement | Components are placed row-wise (left to right, top to bottom). |
| Auto adjustment | Automatically resizes components when the container is resized. |
| Simplifies alignment | Ideal for creating forms, keypads, and tables. |

Constructors of GridLayout

| Constructor | Description |
|--------------|-----------------------------------|
| GridLayout() | Creates a single-cell grid (1×1). |

| | |
|--|---|
| GridLayout(int rows, int cols) | Creates a grid with specified number of rows and columns. |
| GridLayout(int rows, int cols, int hgap, int vgap) | Creates grid with horizontal and vertical gaps between cells. |

Parameters:

- rows → number of rows in the grid.
- cols → number of columns in the grid.
- hgap → horizontal spacing (pixels) between components.
- vgap → vertical spacing (pixels) between components.

Example: Demonstrating GridLayout

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

public class GridLayoutExample extends Frame {
    public GridLayoutExample() {
        setTitle("GridLayout Example");
        setSize(300, 200);
        setLayout(new GridLayout(3, 2, 10, 10));
        // 3 rows, 2 columns, 10px horizontal & vertical gap

        // Add components to the grid
        add(new Label("Name:"));
        add(new TextField(10));

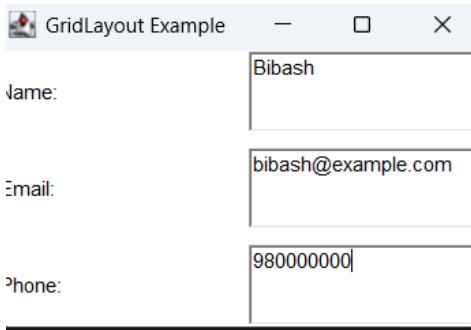
        add(new Label("Email:"));
        add(new TextField(10));

        add(new Label("Phone:"));
        add(new TextField(10));

        setVisible(true);
    }

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

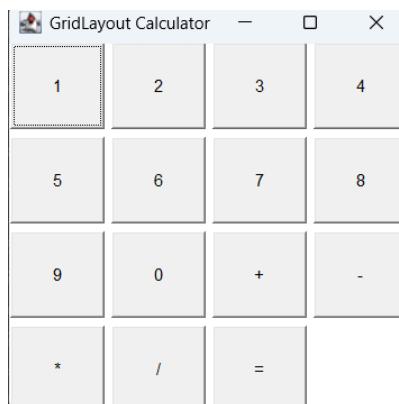
Output:



Example: Calculator Layout Using GridLayout

```
import java.awt.*;  
  
public class CalculatorGrid {  
    public static void main(String[] args) {  
        Frame f = new Frame("GridLayout Calculator");  
        f.setLayout(new GridLayout(4, 4, 5, 5)); // 4x4 grid  
  
        // Add calculator buttons  
        for (int i = 1; i <= 9; i++)  
            f.add(new Button(String.valueOf(i)));  
  
        f.add(new Button("0"));  
        f.add(new Button("+"));  
        f.add(new Button("-"));  
        f.add(new Button("*"));  
        f.add(new Button("/"));  
        f.add(new Button("="));  
  
        f.setSize(300, 300);  
        f.setVisible(true);  
    }  
}
```

Output:



Advantages of GridLayout

| Advantage | Description |
|----------------------|--|
| Uniform appearance | Ensures equal size for all components. |
| Automatic adjustment | Resizes all components proportionally. |
| Simple setup | Easy to arrange form-like layouts. |
| | |

Limitations

| Limitation | Description |
|----------------------|---|
| Equal cell size only | Can't have components of different sizes. |
| No alignment control | Components fill their cell entirely. |
| Fixed structure | Changing layout dynamically can be difficult. |

35.What is prepared statement? When it is useful? Explain its use with suitable Java code.

What is a PreparedStatement?

A PreparedStatement is a precompiled SQL statement in Java used for executing parameterized queries efficiently and securely.

A PreparedStatement is an object used to execute parameterized SQL queries that can be compiled once and executed multiple times with different input values.

Why Use PreparedStatement?

| Feature | Description |
|------------------------|---|
| Precompiled SQL | SQL statement is compiled only once by the database for faster performance. |
| Prevents SQL Injection | Automatically escapes input values — protects against SQL attacks. |
| Dynamic Parameters | Uses ? placeholders for dynamic data. |
| Easier Maintenance | Reduces code complexity for repeated SQL executions. |

Syntax:

```
PreparedStatement pstmt = con.prepareStatement("SQL QUERY");
```

- Use ? to represent parameters in SQL.
- Set parameter values using methods like:
 - setString(int index, String value)
 - setInt(int index, int value)
 - setDouble(int index, double value)
 - etc.

Example: Inserting Data Using PreparedStatement

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

```
public class PreparedStatementExample {
    public static void main(String[] args) {
        Connection con = null;
        PreparedStatement pstmt = null;
```

```

try {
    // Load the MySQL driver
    Class.forName("com.mysql.cj.jdbc.Driver");

    // Establish connection
    con = DriverManager.getConnection("jdbc:mysql://localhost:3306/testdb", "root",
"password");

    // SQL query with placeholders
    String query = "INSERT INTO students (id, name, age) VALUES (?, ?, ?)";

    // Create PreparedStatement object
    pstmt = con.prepareStatement(query);

    // Set values for placeholders
    pstmt.setInt(1, 101);
    pstmt.setString(2, "Bibash");
    pstmt.setInt(3, 21);

    // Execute the query
    int rows = pstmt.executeUpdate();

    System.out.println(rows + " record(s) inserted successfully!");

} catch (Exception e) {
    e.printStackTrace();
} finally {
    try {
        if (pstmt != null) pstmt.close();
        if (con != null) con.close();
    } catch (SQLException ex) {
        ex.printStackTrace();
    }
}
}

```

Example: Fetching Data Using PreparedStatement

```

String sql = "SELECT * FROM students WHERE age > ?";
PreparedStatement pstmt = con.prepareStatement(sql);
pstmt.setInt(1, 18);
ResultSet rs = pstmt.executeQuery();

```

```

while (rs.next()) {
    System.out.println(rs.getInt("id") + " " + rs.getString("name"));
}

```

When is PreparedStatement Useful?

| Situation | Why Useful |
|--|--------------------------------|
| When executing same SQL multiple times | Reduces compilation overhead |
| When accepting user input | Prevents SQL injection attacks |
| When working with dynamic parameters | Simplifies value assignment |
| When performance is important | Precompilation improves speed |
| | |

Difference Between Statement and PreparedStatement

| Feature | Statement | PreparedStatement |
|-------------|-----------------------------|-------------------------------|
| SQL Type | Static (fixed SQL) | Parameterized (uses ?) |
| Compilation | Compiled every time | Compiled once |
| Security | Vulnerable to SQL injection | Safe (auto-escapes input) |
| Performance | Slower for repeated queries | Faster for repeated execution |
| Use Case | Simple one-time queries | Repeated or dynamic queries |

36. Define result set. What are its variations? Explain each variation with suitable example.

A ResultSet is a Java object that stores the data returned by executing a SQL query (usually a SELECT statement) using JDBC.

ResultSet Class

- Belongs to package java.sql
- Created by executing a query using either Statement or PreparedStatement

```
ResultSet rs = stmt.executeQuery("SELECT * FROM employees");
```

Working Mechanism

- Initially, the cursor of the ResultSet is placed before the first row.
- You must call rs.next() to move it to the next record.
- Data is read using methods like:
 - getInt("columnName")
 - getString("columnName")
 - getDouble("columnName")
 - etc.

Example: Basic ResultSet Usage

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

```

public class ResultSetExample {
    public static void main(String[] args) {
        try {
            // Load driver

```

```

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

// Connect to DB
Connection con =
DriverManager.getConnection("jdbc:mysql://localhost:3306/testdb", "root", "password");

// Create statement
Statement stmt = con.createStatement();

// Execute query
ResultSet rs = stmt.executeQuery("SELECT id, name, salary FROM employees");

// Process result
while (rs.next()) {
    System.out.println("ID: " + rs.getInt("id") +
        ", Name: " + rs.getString("name") +
        ", Salary: " + rs.getDouble("salary"));
}

// Close connections
rs.close();
stmt.close();
con.close();
} catch (Exception e) {
    e.printStackTrace();
}
}
}
}

```

Variations (Types) of ResultSet

ResultSet variations are based on scrollability and updatability.

When creating a Statement, you can specify these types.

(A) By Scrollability

| Type | Description |
|-------------------------|--|
| TYPE_FORWARD_ONLY | Cursor moves only forward (default type). |
| TYPE_SCROLL_INSENSITIVE | Cursor can move both forward and backward but does <i>not</i> reflect database changes made by others. |
| TYPE_SCROLL_SENSITIVE | Cursor can move both directions and <i>reflects real-time changes</i> made in the database by others. |

Example: Scrollable ResultSet

Statement stmt = con.createStatement(

```

    ResultSet.TYPE_SCROLL_INSENSITIVE,
    ResultSet.CONCUR_READ_ONLY);

```

```

ResultSet rs = stmt.executeQuery("SELECT * FROM employees");

// Move cursor to last record
rs.last();
System.out.println("Last Employee: " + rs.getString("name"));

// Move cursor backward
rs.previous();
System.out.println("Previous Employee: " + rs.getString("name"));

// Move cursor to first record
rs.first();
System.out.println("First Employee: " + rs.getString("name"));

```

(B) By Updatability

| Type | Description |
|------------------|--|
| CONCUR_READ_ONLY | Default. You can only read data. |
| CONCUR_UPDATABLE | You can update, insert, or delete rows through the ResultSet directly. |

Example: Updatable ResultSet

```

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

```

```
ResultSet rs = stmt.executeQuery("SELECT id, name, salary FROM employees");
```

```

// Move to the first row
if (rs.next()) {
    // Update salary of first employee
    rs.updateDouble("salary", 50000);
    rs.updateRow();
    System.out.println("Salary updated successfully!");
}

```

```

// Insert new record through ResultSet
rs.moveToInsertRow();
rs.updateInt("id", 104);
rs.updateString("name", "New Employee");
rs.updateDouble("salary", 30000);
rs.insertRow();
System.out.println("New record inserted through ResultSet!");

```

37.What are the advantages of using Java Beans? How is it different from Java classes?

Explain?

A JavaBean is a reusable software component written in Java that follows specific conventions so it can be easily used, reused, and manipulated in visual development tools (like NetBeans, Eclipse GUI Builder, etc.).

JavaBean Conventions (Rules):

A class becomes a JavaBean if it follows these rules:

1. Must be public and have a public no-argument constructor
2. Must have private properties (fields)
3. Must provide public getter and setter methods to access those properties
4. Must be serializable (usually implements java.io.Serializable)

Example of a Simple Java Bean

```
import java.io.Serializable;
```

```
public class EmployeeBean implements Serializable {  
    private int id;  
    private String name;  
    private double salary;  
  
    // No-argument constructor  
    public EmployeeBean() {}  
  
    // Getters and Setters (Bean Properties)  
    public int getId() {  
        return id;  
    }  
  
    public void setId(int id) {  
        this.id = id;  
    }  
  
    public String getName() {  
        return name;  
    }  
  
    public void setName(String name) {  
        this.name = name;  
    }  
  
    public double getSalary() {  
        return salary;  
    }
```

```

public void setSalary(double salary) {
    this.salary = salary;
}
}

```

Advantages of Using JavaBeans

| Advantage | Explanation |
|-----------------------------|--|
| Reusability | JavaBeans are reusable components that can be used across different applications without modification. |
| Encapsulation | Properties are private and accessed only through getters/setters, ensuring data protection. |
| Easy to Use in GUI Builders | Beans can be visually manipulated (drag-and-drop) in tools like NetBeans or Eclipse. |
| Customization | Bean properties can be customized using property editors. |
| Persistence | Beans can be serialized — saved and restored later (object persistence). |
| Platform Independent | Written in Java, so portable across platforms. |
| Event Handling Support | Supports event delegation model to handle user interactions. |
| Modularity | Promotes modular design by dividing applications into reusable components. |

Difference Between JavaBean and Normal Java Class

| Feature | Java Class | JavaBean |
|---------------|---|--|
| Purpose | General-purpose class used for logic | Reusable component used for data and GUI handling |
| Constructor | Can have any type of constructors | Must have a public no-argument constructor |
| Properties | Not mandatory to follow naming convention | Follows specific naming convention (getX(), setX()) |
| Encapsulation | Optional | Mandatory (fields private, accessed via getters/setters) |
| Serialization | Not required | Recommended (implements Serializable) |
| Reusability | Limited | High — can be reused across applications |
| Used In | General Java programs | Component-based development, GUI tools, frameworks |
| Tool Support | Not automatically recognized | Recognized by IDEs and Bean containers (like BeanBox) |

Example of Java Class vs JavaBean

- ◊ Normal Java Class

```
public class Employee {
```

```

    public int id;
    public String name;
    public double salary;
}
 Public fields directly accessible
 No encapsulation
 Not reusable or recognized by GUI tools

```

JavaBean

```

public class EmployeeBean implements Serializable {
    private int id;
    private String name;
    private double salary;

    public EmployeeBean() {} // No-arg constructor

    public int getId() { return id; }
    public void setId(int id) { this.id = id; }

    public String getName() { return name; }
    public void setName(String name) { this.name = name; }

    public double getSalary() { return salary; }
    public void setSalary(double salary) { this.salary = salary; }
}

```

38. Define adapter class. Explain advantages of adapter class over listener interfaces with suitable example.

An Adapter Class in Java is an abstract class that provides empty implementations of all methods in a listener interface.

Package and Usage

Adapter classes are available in the `java.awt.event` package.

Some commonly used adapter classes:

| Listener Interface | Corresponding Adapter Class |
|--------------------|-----------------------------|
| MouseListener | MouseAdapter |
| KeyListener | KeyAdapter |
| WindowListener | WindowAdapter |
| FocusListener | FocusAdapter |
| ComponentListener | ComponentAdapter |

Why Adapter Classes Are Needed

Without adapter classes:

- If you implement a listener interface directly, you must write all the methods, even if you need only one.
- This makes the code longer and harder to maintain.

Adapter classes help by:

- Allowing you to override only the methods you need
- Reducing boilerplate code
- Making the code more readable and cleaner

Example Without Adapter Class

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

public class MouseListenerExample extends Frame implements MouseListener {

    Label l;

    MouseListenerExample() {
        l = new Label("Mouse not clicked yet");
        l.setBounds(50, 100, 200, 30);
        add(l);

        addMouseListener(this); // Register listener
        setSize(300, 200);
        setLayout(null);
        setVisible(true);
    }

    public void mouseClicked(MouseEvent e) {
        l.setText("Mouse Clicked!");
    }

    // We must override all these even if unused
    public void mousePressed(MouseEvent e) {}
    public void mouseReleased(MouseEvent e) {}
    public void mouseEntered(MouseEvent e) {}
    public void mouseExited(MouseEvent e) {}

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

Example Using Adapter Class

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

public class MouseAdapterExample extends Frame {
```

```
Label l;
```

```
MouseAdapterExample() {
    l = new Label("Click anywhere on the frame");
    l.setBounds(50, 100, 200, 30);
    add(l);

    // Using anonymous inner class extending MouseAdapter
    addMouseListener(new MouseAdapter() {
        public void mouseClicked(MouseEvent e) {
            l.setText("Mouse Clicked at (" + e.getX() + ", " + e.getY() + ")");
        }
    });
}

setSize(300, 200);
setLayout(null);
setVisible(true);
}
```

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

Output:



Mouse not clicked yet

Mouse Clicked!

Advantages of Adapter Classes over Listener Interfaces

| Feature | Listener Interface | Adapter Class |
|-----------------------|--|--------------------------------------|
| Method Implementation | Must implement all methods | Only implement the needed ones |
| Code Length | Longer and repetitive | Shorter and cleaner |
| Ease of Use | Tedious for interfaces with many methods | Very convenient and readable |
| Error-Prone | Easy to make empty/unintended methods | Safer and focused on relevant events |
| Usage | Used directly by implementing class | Used by extending adapter class |

When to Use Adapter Classes

Use an adapter class when:

- The listener interface has multiple abstract methods
- You need to handle only one or two events from that interface
- You want to simplify event-handling code in AWT/Swing applications

39. How can we use GenericServlet and HttpServlet class in writing servlets? Explain with example.

A Servlet is a Java program that runs on a web server and is used to create dynamic web content (like HTML pages generated from database data).

Servlet Class Hierarchy

```
java.lang.Object
  └── javax.servlet.GenericServlet
    └── javax.servlet.http.HttpServlet
```

GenericServlet Class

◊ Definition:

GenericServlet is an abstract class that implements the Servlet, ServletConfig, and Serializable interfaces.

It is protocol-independent, meaning it can handle any type of request (HTTP, FTP, etc.).

Features:

- Base class for all servlets.
- Provides default implementations of lifecycle methods.
- You only need to override the service() method.

Example: Using GenericServlet

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

public class MyGenericServlet extends GenericServlet {

    @Override
    public void service(ServletRequest req, ServletResponse res)
```

```

        throws ServletException, IOException {
    res.setContentType("text/html");
    PrintWriter out = res.getWriter();

    out.println("<html><body>");
    out.println("<h2>Hello from GenericServlet!</h2>");
    out.println("<p>This servlet does not depend on any protocol.</p>");
    out.println("</body></html>");
}
}

```

web.xml Configuration:

```

<web-app>
    <servlet>
        <servlet-name>genericExample</servlet-name>
        <servlet-class>MyGenericServlet</servlet-class>
    </servlet>

    <servlet-mapping>
        <servlet-name>genericExample</servlet-name>
        <url-pattern>/generic</url-pattern>
    </servlet-mapping>
</web-app>

```

URL Example:

<http://localhost:8080/MyApp/generic>

HttpServlet Class

Definition:

HttpServlet is a subclass of GenericServlet and is specifically designed for handling HTTP requests (GET, POST, PUT, DELETE, etc.).

Features:

- Protocol-dependent (only works with HTTP)
- Simplifies web development
- Widely used for web applications

Example: Using HttpServlet

```

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

```

```

public class MyHttpServlet extends HttpServlet {

```

@Override

```

protected void doGet(HttpServletRequest req, HttpServletResponse res)
    throws ServletException, IOException {

```

```

res.setContentType("text/html");
PrintWriter out = res.getWriter();

out.println("<html><body>");
out.println("<h2>Hello from HttpServlet!</h2>");
out.println("<p>This servlet handles HTTP GET requests.</p>");
out.println("</body></html>");

}

@Override
protected void doPost(HttpServletRequest req, HttpServletResponse res)
    throws ServletException, IOException {

```

```

    res.setContentType("text/html");
    PrintWriter out = res.getWriter();

    String name = req.getParameter("name");
    out.println("<html><body>");
    out.println("<h2>Welcome, " + name + "</h2>");
    out.println("</body></html>");

}
}

```

web.xml Configuration:

```

<web-app>
    <servlet>
        <servlet-name>httpExample</servlet-name>
        <servlet-class>MyHttpServlet</servlet-class>
    </servlet>

```

```

    <servlet-mapping>
        <servlet-name>httpExample</servlet-name>
        <url-pattern>/http</url-pattern>
    </servlet-mapping>
</web-app>

```

URL Example:

<http://localhost:8080/MyApp/http>

Difference Between GenericServlet and HttpServlet

| Feature | GenericServlet | HttpServlet |
|-------------|-------------------------|-----------------------------------|
| Protocol | Protocol-independent | Works only with HTTP |
| Methods | Must override service() | Overrides doGet(), doPost(), etc. |
| Ease of Use | Basic and general | Simplifies web development |

| | | |
|----------------------------------|------------------------------------|--|
| Common Use | Rare in web apps | Common in all modern web apps |
| Input Type | ServletRequest, ServletResponse | HttpServletRequest, HttpServletResponse |
| Support for Sessions, Cookies | No | Yes |

When to Use Which?

| Scenario | Recommended Class |
|----------|-------------------|
|----------|-------------------|

You need to handle any protocol (not just HTTP) GenericServlet

You are building a web application using HTTP HttpServlet

40. How can we create GUI by using AWT frames? Explain the techniques with suitable example.

Concept: Creating GUI using AWT Frames

In Java AWT (Abstract Window Toolkit), a Frame is a top-level window with a title and border.

It serves as the main window for AWT-based GUI applications.

A Frame can contain various components such as buttons, labels, text fields, checkboxes, etc.

We can create frames by:

1. Extending the Frame class, or
2. Creating an instance of Frame directly.

Steps to Create a GUI using AWT Frame

1. Import the AWT and event packages
 - o import java.awt.*;
 - o import java.awt.event.*;
2. Create a class extending Frame
 - o This allows direct use of AWT methods and components.
3. Add components such as Labels, TextFields, and Buttons.
4. Set layout manager (e.g., FlowLayout, GridLayout, etc.).
5. Register event listeners (e.g., ActionListener for buttons).
6. Set frame properties (size, visibility, closing operation).

Example: Simple AWT Frame GUI

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

```
public class SimpleAWTFrame extends Frame implements ActionListener {
```

```
    TextField tf;
    Button b1, b2;
    Label lbl;
```

```
// Constructor
```

```
SimpleAWTFrame() {
    // Create components
    lbl = new Label("Enter your name:");
    tf = new TextField(20);
    b1 = new Button("Greet");
    b2 = new Button("Clear");

    // Set layout
    setLayout(new FlowLayout());

    // Add components to frame
    add(lbl);
    add(tf);
    add(b1);
    add(b2);

    // Register event listeners
    b1.addActionListener(this);
    b2.addActionListener(this);

    // Set frame properties
    setTitle("Simple AWT GUI Example");
    setSize(350, 200);
    setVisible(true);

    // Close window
    addWindowListener(new WindowAdapter() {
        public void windowClosing(WindowEvent e) {
            dispose(); // closes the frame
        }
    });
}

// Handle button clicks
public void actionPerformed(ActionEvent e) {
    String str = e.getActionCommand();

    if (str.equals("Greet")) {
        String name = tf.getText();
        lbl.setText("Hello, " + name + "!");
    } else if (str.equals("Clear")) {
        tf.setText("");
        lbl.setText("Enter your name:");
    }
}
```

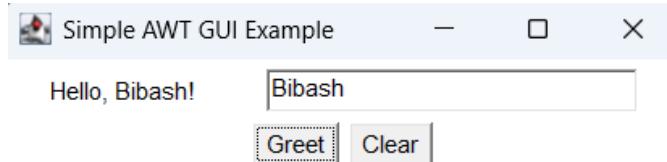
```

        }
    }

    public static void main(String[] args) {
        new SimpleAWTFrame(); // create frame instance
    }
}

```

Output:



41.Explain border layout manager with constructors and demonstrate it by using suitable program.

BorderLayout Manager in Java

BorderLayout is one of the most commonly used layout managers in Java AWT and Swing. It arranges components into five regions of a container:

Regions of BorderLayout

| Region | Constant Used | Description |
|--------|---------------------|----------------------------------|
| NORTH | BorderLayout.NORTH | Placed at the top |
| SOUTH | BorderLayout.SOUTH | Placed at the bottom |
| EAST | BorderLayout.EAST | Placed at the right |
| WEST | BorderLayout.WEST | Placed at the left |
| CENTER | BorderLayout.CENTER | Takes remaining space (required) |

Constructors

1. `BorderLayout()`
 - o Creates a layout with default gaps (0, 0) between components.
2. `BorderLayout(int hgap, int vgap)`
 - o Creates a layout with specified horizontal (hgap) and vertical (vgap) spacing between components.

Example Program: Demonstrating BorderLayout

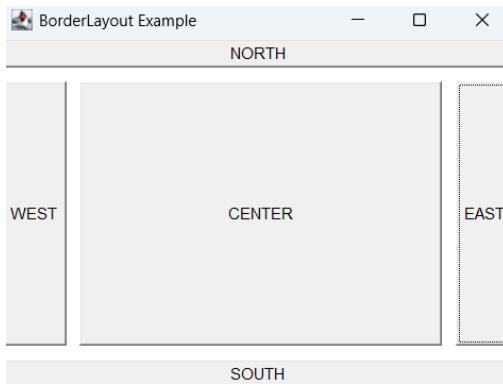
```

import java.awt.*;
import java.awt.event.*;

```

```
public class BorderLayoutExample extends Frame {  
  
    BorderLayoutExample() {  
        // Set title  
        setTitle("BorderLayout Example");  
  
        // Set layout manager with gaps  
        setLayout(new BorderLayout(10, 10));  
  
        // Create buttons for each region  
        Button b1 = new Button("NORTH");  
        Button b2 = new Button("SOUTH");  
        Button b3 = new Button("EAST");  
        Button b4 = new Button("WEST");  
        Button b5 = new Button("CENTER");  
  
        // Add buttons to respective regions  
        add(b1, BorderLayout.NORTH);  
        add(b2, BorderLayout.SOUTH);  
        add(b3, BorderLayout.EAST);  
        add(b4, BorderLayout.WEST);  
        add(b5, BorderLayout.CENTER);  
  
        // Set frame properties  
        setSize(400, 300);  
        setVisible(true);  
  
        // Close the frame properly  
        addWindowListener(new WindowAdapter() {  
            public void windowClosing(WindowEvent e) {  
                dispose();  
            }  
        });  
    }  
  
    public static void main(String[] args) {  
        new BorderLayoutExample();  
    }  
}
```

Output:



42.What are different steps used in JDBC? Write down a small program showing all its steps.

JDBC (Java Database Connectivity)

JDBC is a Java API that allows Java programs to connect and interact with a database (like MySQL, Oracle, PostgreSQL, etc.).

It provides methods to execute SQL queries, insert, update, and retrieve data from databases.

Steps in JDBC

There are 5 main steps to connect and perform operations in JDBC:

| Step | Description |
|-----------------------------|---|
| 1. Load the Driver | Load the database driver class into memory using Class.forName(). |
| 2. Establish the Connection | Create a connection to the database using DriverManager.getConnection(). |
| 3. Create a Statement | Create a Statement or PreparedStatement object to execute SQL queries. |
| 4. Execute Query | Run SQL commands using methods like executeQuery() or executeUpdate(). |
| 5. Process Results | Handle the result set returned by the query (if applicable). |
| 6. Close Connection | Always close the connection, statement, and result set to free resources. |

Example Program: JDBC Steps Demonstration

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

```
public class JDBCExample {  
    public static void main(String[] args) {  
        // Step 1: Declare variables  
        Connection conn = null;  
        Statement stmt = null;  
        ResultSet rs = null;  
  
        try {  
            // Step 2: Load and register the JDBC driver
```

```

Class.forName("com.mysql.cj.jdbc.Driver");
System.out.println("Driver loaded successfully.");

// Step 3: Establish the connection
conn = DriverManager.getConnection(
    "jdbc:mysql://localhost:3306/testdb", "root", "password");
System.out.println("Database connected successfully.");

// Step 4: Create statement
stmt = conn.createStatement();

// Step 5: Execute query
rs = stmt.executeQuery("SELECT * FROM employees");

// Step 6: Process results
System.out.println("Employee Data:");
while (rs.next()) {
    int id = rs.getInt("id");
    String name = rs.getString("name");
    double salary = rs.getDouble("salary");
    System.out.println(id + " | " + name + " | " + salary);
}

} catch (Exception e) {
    e.printStackTrace();
} finally {
    try {
        // Step 7: Close all resources
        if (rs != null) rs.close();
        if (stmt != null) stmt.close();
        if (conn != null) conn.close();
    } catch (SQLException ex) {
        ex.printStackTrace();
    }
}
}
}

```

43.How can we use different DDL and DML statements with Java? Explain each with sample example.

JDBC with DDL and DML Statements

In Java, you can use JDBC (Java Database Connectivity) to interact with databases using SQL commands.

There are two main categories of SQL commands:

| Type | Full Form | Purpose | Example Statements |
|------|----------------------------|--|--------------------------------|
| DDL | Data Definition Language | Define or modify structure of database | CREATE, ALTER, DROP |
| DML | Data Manipulation Language | Manage data stored in tables | INSERT, UPDATE, DELETE, SELECT |

Common JDBC Methods Used

| Method | Description |
|-----------------|--|
| executeUpdate() | Used for DDL and DML statements that change data or structure (CREATE, INSERT, UPDATE, DELETE) |
| executeQuery() | Used for SELECT statements that return data (ResultSet) |

Example 1: Using DDL Statements

This example demonstrates creating a table using CREATE TABLE (a DDL command).

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

```
public class DDLEExample {  
    public static void main(String[] args) {  
        Connection conn = null;  
        Statement stmt = null;  
  
        try {  
            // Step 1: Load JDBC driver  
            Class.forName("com.mysql.cj.jdbc.Driver");  
  
            // Step 2: Establish connection  
            conn = DriverManager.getConnection(  
                "jdbc:mysql://localhost:3306/testdb", "root", "password");  
  
            // Step 3: Create statement  
            stmt = conn.createStatement();  
  
            // Step 4: Execute DDL statement  
            String sql = "CREATE TABLE Employees (" +  
                "id INT PRIMARY KEY AUTO_INCREMENT, " +  
                "name VARCHAR(50), " +  
                "salary DOUBLE);  
            stmt.executeUpdate(sql);  
            System.out.println("Table 'Employees' created successfully!");  
  
        } catch (Exception e) {  
            e.printStackTrace();  
        } finally {
```

```
        try { if (stmt != null) stmt.close(); if (conn != null) conn.close(); } catch (Exception e)
    {}
}
}
```

Example 2: Using DML Statements

This example shows how to insert, update, delete, and select data from the database.

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

```
public class DMLExample {
    public static void main(String[] args) {
        Connection conn = null;
        Statement stmt = null;
        ResultSet rs = null;

        try {
            // Step 1: Load driver
            Class.forName("com.mysql.cj.jdbc.Driver");

            // Step 2: Establish connection
            conn = DriverManager.getConnection(
                "jdbc:mysql://localhost:3306/testdb", "root", "password");

            // Step 3: Create statement
            stmt = conn.createStatement();

            // INSERT
            String insertSQL = "INSERT INTO Employees (name, salary) VALUES ('Bibash',
45000)";
            stmt.executeUpdate(insertSQL);
            System.out.println("Record inserted successfully!");

            // UPDATE
            String updateSQL = "UPDATE Employees SET salary = 50000 WHERE name =
'Bibash'";
            stmt.executeUpdate(updateSQL);
            System.out.println("Record updated successfully!");

            // SELECT
            String selectSQL = "SELECT * FROM Employees";
            rs = stmt.executeQuery(selectSQL);
            System.out.println("Employee Data:");
            while (rs.next()) {
```

```

        System.out.println(rs.getInt("id") + " | " + rs.getString("name") + " | " +
rs.getDouble("salary"));
    }

// DELETE
String deleteSQL = "DELETE FROM Employees WHERE name = 'Bibash'";
stmt.executeUpdate(deleteSQL);
System.out.println("Record deleted successfully!");

} catch (Exception e) {
    e.printStackTrace();
} finally {
    try { if (rs != null) rs.close(); if (stmt != null) stmt.close(); if (conn != null)
conn.close(); } catch (Exception e) {}
}
}
}
}

```

44.What is Java Bean? Explain key features of Java Bean with brief description.

A Java Bean is a reusable software component written in Java that follows specific naming conventions and design rules so that it can be easily used in visual development tools (like NetBeans, Eclipse GUI Builder, etc.) or in enterprise applications.

Java Beans are commonly used to encapsulate data (like an Employee, Student, or Product) and provide getter/setter methods to access and modify that data safely.

Example: A Simple Java Bean Class

```
import java.io.Serializable;
```

```

// A simple Employee Bean
public class Employee implements Serializable {
    private int id;
    private String name;
    private double salary;

    // No-argument constructor (required for Java Beans)
    public Employee() {}

    // Getter and Setter methods
    public int getId() {
        return id;
    }
    public void setId(int id) {
        this.id = id;
    }
}
```

```

public String getName() {
    return name;
}
public void setName(String name) {
    this.name = name;
}

public double getSalary() {
    return salary;
}
public void setSalary(double salary) {
    this.salary = salary;
}
}

```

Key Features of Java Beans

| Feature | Description |
|----------------------------|--|
| 1. Introspection | Beans can be examined (introspected) by development tools to understand their properties, events, and methods. |
| 2. Properties | Private data fields (like name, salary) that can be accessed via public getter/setter methods. |
| 3. Persistence | Beans are serializable, meaning their state can be saved and restored (e.g., stored in a file or database). |
| 4. Reusability | Once created, a Bean can be reused in multiple applications without code modification. |
| 5. Event Handling | Beans can generate and handle events using the standard Java event model (ActionListener, PropertyChangeListener, etc.). |
| 6. Customization | Beans can be customized in visual development tools using property editors (for example, changing colors, labels, etc.). |
| 7. No-Argument Constructor | Every Java Bean must have a public no-arg constructor so that development tools can instantiate it easily. |

Why Use Java Beans?

- Encourages modular, reusable design
- Simplifies data encapsulation
- Ideal for enterprise and GUI development
- Easily integrated with frameworks like JSP, JSF, and Spring

Example of Using the Bean

```

public class TestBean {

    public static void main(String[] args) {
        Employee emp = new Employee();
        emp.setId(101);
    }
}

```

```

        emp.setName("Bibash");
        emp.setSalary(50000);

        System.out.println("Employee ID: " + emp.getId());
        System.out.println("Employee Name: " + emp.getName());
        System.out.println("Employee Salary: " + emp.getSalary());
    }
}

```

45.What are Listeners and Event sources? Explain at least five Listener interfaces and Event sources briefly.

Java uses an event delegation model for GUI programming.

It involves three main components:

| Component | Description |
|----------------|--|
| Event Source | The object that generates an event (like a Button, TextField, or MenuItem). |
| Event Object | The object that contains information about the event (like ActionEvent, KeyEvent). |
| Event Listener | The object that receives and handles the event (like ActionListener, KeyListener). |

Event Source

An event source is any GUI component that can generate events.

For example:

- Button → generates an ActionEvent
- TextField → generates TextEvent or ActionEvent
- Window → generates WindowEvent
- Mouse → generates MouseEvent

Example:

```

Button b = new Button("Click Me");
b.addActionListener(this);

```

Event Listener

A listener is an interface that defines one or more methods to handle specific types of events.

When the user performs an action (like a click or keypress), the listener method is automatically called.

Example:

```

public void actionPerformed(ActionEvent e) {
    System.out.println("Button clicked!");
}

```

Common Listener Interfaces and Their Event Sources

Below are five important listener interfaces and their typical event sources

| Listener Interface | Event Source | Event Class | Description |
|--------------------|--------------------------------|-------------|--|
| 1. ActionListener | Button, MenuItem, TextField | ActionEvent | Handles actions like button clicks or menu selections. |
| 2. ItemListener | Checkbox, Choice, List | ItemEvent | Handles item selection or deselection. |
| 3. KeyListener | Any component (like TextField) | KeyEvent | Handles keyboard actions (key press, release, type). |
| 4. MouseListener | Button, Label, Panel, etc. | MouseEvent | Handles mouse actions like click, press, enter, exit, release. |
| 5. WindowListener | Frame, Dialog | WindowEvent | Handles window events like opening, closing, minimizing. |

Example: Multiple Listeners in Action

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

public class EventExample extends Frame implements ActionListener, KeyListener,
MouseListener, WindowListener {
```

```
TextField tf;
Button b;

EventExample() {
    setLayout(new FlowLayout());

    tf = new TextField(20);
    b = new Button("Click Me");

    add(tf);
    add(b);

    // Register listeners
    b.addActionListener(this);      // Action Event
    tf.addKeyListener(this);       // Key Event
    addMouseListener(this);        // Mouse Event
    addWindowListener(this);       // Window Event
```

```
    setTitle("Event Handling Example");
    setSize(350, 250);
    setVisible(true);
}
```

```
// 1. ActionListener
```

```

public void actionPerformed(ActionEvent e) {
    tf.setText("Button Clicked!");
}

// 2. KeyListener
public void keyTyped(KeyEvent e) {}
public void keyPressed(KeyEvent e) {
    tf.setText("Key Pressed: " + e.getKeyChar());
}
public void keyReleased(KeyEvent e) {}

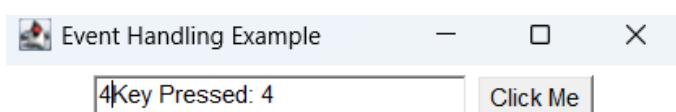
// 3. MouseListener
public void mouseClicked(MouseEvent e) {
    tf.setText("Mouse Clicked at (" + e.getX() + "," + e.getY() + ")");
}
public void mouseEntered(MouseEvent e) {}
public void mouseExited(MouseEvent e) {}
public void mousePressed(MouseEvent e) {}
public void mouseReleased(MouseEvent e) {}

// 4. WindowListener
public void windowClosing(WindowEvent e) {
    dispose();
}
public void windowOpened(WindowEvent e) {}
public void windowClosed(WindowEvent e) {}
public void windowIconified(WindowEvent e) {}
public void windowDeiconified(WindowEvent e) {}
public void windowActivated(WindowEvent e) {}
public void windowDeactivated(WindowEvent e) {}

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

```

Output:



Explanation of Listeners Used

| Listener | Triggered When | Event Source Example |
|----------------|--------------------------------|-----------------------------|
| ActionListener | Button is clicked | Button, MenuItem, TextField |
| KeyListener | Key pressed/released/typed | TextField, Frame |
| MouseListener | Mouse clicked or moved | Panel, Label, Frame |
| WindowListener | Window opened/closed/minimized | Frame, Dialog |
| ItemListener | Checkbox or list item selected | Checkbox, Choice |

46.What are the APIs used in creating servlet programs? How can Servlet interface be used in creating servlets? Explain with example.

Servlet APIs

Java provides two main packages for servlet development:

1. javax.servlet

- o This package contains classes and interfaces that define the basic structure of servlets.
- o It includes interfaces like:
 - Servlet
 - ServletRequest
 - ServletResponse
 - ServletConfig
 - ServletContext

2. javax.servlet.http

- o This package provides classes and interfaces specific to HTTP protocol-based servlets.
- o Common classes/interfaces:
 - HttpServlet
 - HttpServletRequest
 - HttpServletResponse
 - HttpSession
 - Cookie

Servlet Interface

The Servlet interface is the root interface for all servlets.

Every servlet must implement this interface either directly or indirectly (through GenericServlet or HttpServlet).

Methods of the Servlet Interface

| Method | Description |
|--|--|
| init(ServletConfig config) | Called once when the servlet is initialized. |
| service(ServletRequest req, ServletResponse res) | Called for every client request. Handles request and response. |
| destroy() | Called before servlet is destroyed. Used for cleanup. |
| getServletConfig() | Returns configuration info about servlet. |

| | |
|-------------------------------|---|
| <code>getServletInfo()</code> | Returns information about servlet (like version, author). |
|-------------------------------|---|

Example Using Servlet Interface

```

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

@WebServlet("/simpleServlet")
public class SimpleServlet implements Servlet {

    private ServletConfig config = null;

    // 1. Initialize the servlet
    public void init(ServletConfig config) throws ServletException {
        this.config = config;
        System.out.println("Servlet is initialized");
    }

    // 2. Service method – handles client request
    public void service(ServletRequest request, ServletResponse response)
        throws ServletException, IOException {

        response.setContentType("text/html");
        PrintWriter out = response.getWriter();
        out.println("<html><body>");
        out.println("<h2>Welcome to Simple Servlet Example!</h2>");
        out.println("<p>This servlet implements the Servlet interface directly.</p>");
        out.println("</body></html>");
    }

    // 3. Destroy method – cleanup before servlet is removed
    public void destroy() {
        System.out.println("Servlet is destroyed");
    }

    // 4. Return servlet config
    public ServletConfig getServletConfig() {
        return config;
    }

    // 5. Return servlet info
    public String getServletInfo() {
        return "SimpleServlet implementing Servlet interface";
    }
}

```

```
}
```

Advantages of Using Servlet Interface

- It defines the life cycle of a servlet clearly.
- Provides a standard API for any servlet container.
- Helps developers understand servlet internals before using higher-level classes like HttpServlet.

47. Differentiate between components and containers. Explain different AWT containers briefly.

Differentiate between Components and Containers

| Basis | Component | Container |
|----------------------------|--|--|
| Definition | A Component is an object that represents a visual element (like a button, text field, label) that can be displayed on the screen and can interact with the user. | A Container is a special type of component that can hold and organize other components (including other containers). |
| Purpose | Used to perform specific user interface tasks such as input or display. | Used to group and manage layout and behavior of multiple components. |
| Examples | Button, Label, TextField, Checkbox, etc. | Frame, Panel, Dialog, Applet, etc. |
| Can hold other components? | No, components cannot hold other components. | Yes, containers can hold components. |
| Inheritance | All components inherit from the Component class. | All containers inherit from the Container class (which itself extends Component). |

AWT Containers and Their Explanation

Java AWT provides several container classes that help in holding and organizing components.

1. Panel

- Definition: Simplest container used for grouping multiple components.
- Parent class: java.awt.Panel
- Commonly used as a section or sub-container inside a frame.

Example:

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

```
public class PanelExample {  
    public static void main(String[] args) {  
        Frame frame = new Frame("Panel Example");  
        Panel panel = new Panel();
```

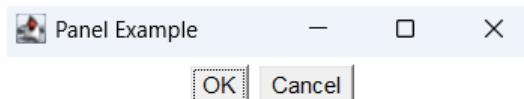
```

        panel.add(new Button("OK"));
        panel.add(new Button("Cancel"));

        frame.add(panel);
        frame.setSize(300, 200);
        frame.setVisible(true);
    }
}

```

Output:



2. Frame

- Definition: A top-level window with a title bar and border.
- Parent class: `java.awt.Frame`
- It can hold panels, buttons, menus, etc.
- Most commonly used container for standalone AWT applications.

Example:

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

```

public class FrameExample {
    public static void main(String[] args) {
        Frame f = new Frame("Frame Example");
        f.add(new Button("Click Me"));
        f.setSize(300, 200);
        f.setVisible(true);
    }
}

```

3. Dialog

- Definition: A pop-up window used to interact with the user.
- Parent class: `java.awt.Dialog`
- Usually used for displaying messages, confirmations, or data input.
- Can be modal (blocks parent window) or modeless (non-blocking).

Example:

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

```

public class DialogExample {
    public static void main(String[] args) {
        Frame frame = new Frame("Main Frame");

```

```

        Dialog dialog = new Dialog(frame, "Dialog Box", true);

        dialog.add(new Label("This is a modal dialog"));
        dialog.setSize(200, 100);
        dialog.setVisible(true);

        frame.setSize(300, 200);
        frame.setVisible(true);
    }
}

```

4. Applet

- Definition: A container used to run Java programs inside a web browser.
- Parent class: `java.applet.Applet`
- Mainly used in older Java web applications (now deprecated).

Example:

```

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

```

```

public class MyApplet extends Applet {
    public void paint(Graphics g) {
        g.drawString("Hello from Applet!", 20, 20);
    }
}

```

48.Explain flow layout manager with suitable constructors and demonstrate it by using suitable Java code.

The `FlowLayout` is the simplest layout manager in Java AWT and Swing.

It arranges the components in a single row, one after another — like words in a sentence.

When one row is filled, components automatically flow to the next line.

Key Features of `FlowLayout`

- Components are arranged left-to-right, top-to-bottom.
- It respects the component's preferred size.
- Automatically wraps components to the next line when space runs out.
- Commonly used in panels or simple frames.

Constructors of `FlowLayout`

| Constructor | Description |
|--|---|
| <code>FlowLayout()</code> | Creates a flow layout with center alignment and default gaps (5 pixels). |
| <code>FlowLayout(int align)</code> | Creates a flow layout with the specified alignment (<code>FlowLayout.LEFT</code> , <code>FlowLayout.RIGHT</code> , <code>FlowLayout.CENTER</code>). |
| <code>FlowLayout(int align, int hgap, int vgap)</code> | Creates a flow layout with specified alignment and horizontal (hgap) and vertical (vgap) gaps between components. |

Common Alignment Constants

- FlowLayout.LEFT
- FlowLayout.CENTER (default)
- FlowLayout.RIGHT
- FlowLayout.LEADING
- FlowLayout.TRAILING

Example Program: FlowLayout Demonstration

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

public class FlowLayoutExample {
    public static void main(String[] args) {
        // Create a frame
        Frame frame = new Frame("FlowLayout Example");

        // Set layout as FlowLayout (center alignment, gaps of 10px)
        frame.setLayout(new FlowLayout(FlowLayout.CENTER, 10, 10));

        // Create some buttons
        Button b1 = new Button("Button 1");
        Button b2 = new Button("Button 2");
        Button b3 = new Button("Button 3");
        Button b4 = new Button("Button 4");
        Button b5 = new Button("Button 5");

        // Add buttons to the frame
        frame.add(b1);
        frame.add(b2);
        frame.add(b3);
        frame.add(b4);
        frame.add(b5);

        // Set frame size and visibility
        frame.setSize(350, 150);
        frame.setVisible(true);

        // Add a window listener to close the frame
        frame.addWindowListener(new WindowAdapter() {
            public void windowClosing(WindowEvent e) {
                frame.dispose();
            }
        });
    }
}
```

```
}
```

```
}
```

Output:



49.What is jar file? Which jar file is required to connect to MySQL? How it can be used? Explain.

A JAR (Java ARchive) file is a compressed package that bundles:

- Java .class files (compiled code),
- Metadata (like MANIFEST.MF),
- Resources (like images, configuration files, etc.),
into a single file (with extension .jar).

It is similar to a ZIP file and is used for distributing Java applications or libraries.

Purpose of JAR Files

- To package multiple Java classes into one file.
- To distribute libraries or frameworks.
- To reuse third-party code easily.
- To execute Java programs with all dependencies in one file (Executable JAR).

Which JAR File is Required to Connect to MySQL?

The JAR file required is:

mysql-connector-j-x.x.xx.jar

(e.g., mysql-connector-j-8.3.0.jar)

This JAR file contains the JDBC (Java Database Connectivity) driver for MySQL — which allows Java programs to communicate with MySQL databases.

Steps to Use MySQL Connector JAR

1. Add the JAR file to your project

If using:

- Eclipse / IntelliJ:
Right-click on your project → Properties → Java Build Path → Libraries → Add External JARs → select the downloaded .jar file.
- Command line:
Use -cp (classpath) when compiling/running.
javac -cp .;mysql-connector-j-8.3.0.jar MyProgram.java
java -cp .;mysql-connector-j-8.3.0.jar MyProgram

2. Import the JDBC classes

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

3. Load MySQL Driver

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

4. Establish Connection

```
Connection con = DriverManager.getConnection(
    "jdbc:mysql://localhost:3306/testdb", "root", "your_password");
```

5. Perform SQL Operations

```
Statement stmt = con.createStatement();
ResultSet rs = stmt.executeQuery("SELECT * FROM employee");

while (rs.next()) {
    System.out.println(rs.getInt(1) + " " + rs.getString(2));
}
```

6. Close Connection

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

Complete Example: Java Program to Connect MySQL Using JAR

```
import java.sql.*;

public class MySQLConnectionExample {
    public static void main(String[] args) {
        try {
            // 1. Load the MySQL JDBC driver
            Class.forName("com.mysql.cj.jdbc.Driver");

            // 2. Establish the connection
            Connection con = DriverManager.getConnection(
                "jdbc:mysql://localhost:3306/testdb", "root", "your_password");

            // 3. Create a statement
            Statement stmt = con.createStatement();

            // 4. Execute a query
            ResultSet rs = stmt.executeQuery("SELECT * FROM employee");

            // 5. Process the result
            while (rs.next()) {
                System.out.println("ID: " + rs.getInt("id") +
                    ", Name: " + rs.getString("name"));
            }

            // 6. Close the connection
            con.close();
        } catch (Exception e) {
            System.out.println(e);
        }
    }
}
```

```
}
```

50.What are different form of execute statement? Explain each form with suitable syntax and example.

In JDBC, SQL statements can be executed using three main methods:

1. executeQuery()
2. executeUpdate()
3. execute()

These methods are available in the Statement, PreparedStatement, and CallableStatement interfaces.

1. executeQuery()

Purpose:

- Used to execute SELECT statements.
- Returns a ResultSet object containing the data retrieved from the database.

Syntax:

```
ResultSet rs = statement.executeQuery(String sql);
```

Example:

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

```
public class ExecuteQueryExample {  
    public static void main(String[] args) {  
        try {  
            Class.forName("com.mysql.cj.jdbc.Driver");  
            Connection con = DriverManager.getConnection(  
                "jdbc:mysql://localhost:3306/testdb", "root", "your_password");  
  
            Statement stmt = con.createStatement();  
            ResultSet rs = stmt.executeQuery("SELECT id, name FROM employee");  
  
            while (rs.next()) {  
                System.out.println("ID: " + rs.getInt("id") +  
                    ", Name: " + rs.getString("name"));  
            }  
  
            con.close();  
        } catch (Exception e) {  
            System.out.println(e);  
        }  
    }  
}
```

Key Point:

- Always used for retrieving data.
- Returns a ResultSet object.
- Throws an exception if used for non-SELECT SQL.

2. executeUpdate()

Purpose:

- Used for executing INSERT, UPDATE, DELETE, or DDL statements (like CREATE TABLE).
- Returns an integer representing the number of rows affected.

Syntax:

```
int count = statement.executeUpdate(String sql);
```

Example:

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

```
public class ExecuteUpdateExample {
    public static void main(String[] args) {
        try {
            Class.forName("com.mysql.cj.jdbc.Driver");
            Connection con = DriverManager.getConnection(
                "jdbc:mysql://localhost:3306/testdb", "root", "your_password");

            Statement stmt = con.createStatement();
            int result = stmt.executeUpdate("UPDATE employee SET salary = 50000 WHERE id
= 101");

            System.out.println(result + " record(s) updated");
        } catch (Exception e) {
            System.out.println(e);
        }
    }
}
```

Key Point:

- Returns number of rows affected for DML operations.
- Returns 0 for DDL statements (e.g., CREATE TABLE).
- Does not return a ResultSet.

3. execute()

Purpose:

- Used for executing any kind of SQL statement (SELECT, INSERT, UPDATE, DELETE, or stored procedures).
- Returns a boolean value:
 - true → if the result is a ResultSet (like SELECT)

- false → if the result is an update count or no result (like INSERT/UPDATE/DELETE)

Syntax:

```
boolean hasResultSet = statement.execute(String sql);
```

Example:

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

```
public class ExecuteExample {
    public static void main(String[] args) {
        try {
            Class.forName("com.mysql.cj.jdbc.Driver");
            Connection con = DriverManager.getConnection(
                "jdbc:mysql://localhost:3306/testdb", "root", "your_password");

            Statement stmt = con.createStatement();
            boolean result = stmt.execute("SELECT * FROM employee");

            if (result) {
                ResultSet rs = stmt.getResultSet();
                while (rs.next()) {
                    System.out.println("ID: " + rs.getInt("id") +
                        ", Name: " + rs.getString("name"));
                }
            } else {
                int updateCount = stmt.getUpdateCount();
                System.out.println("Rows affected: " + updateCount);
            }

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

Key Point:

- Can execute both query and update statements.
- Must check return value (true / false) to decide whether a ResultSet or update count is returned.

Summary Table

| Method | Used For | Return Type | Description |
|----------------|-------------------|-------------|-----------------------------|
| executeQuery() | SELECT statements | ResultSet | Returns data from database. |

| | | | |
|-----------------|-----------------------------|---------|---|
| executeUpdate() | INSERT, UPDATE, DELETE, DDL | int | Returns number of rows affected. |
| execute() | Any SQL statement | boolean | Returns true (ResultSet) or false (update count). |

51.Explain the process of reading and updating jar file with suitable example of each.

What is a JAR File?

A JAR (Java ARchive) file is a compressed file (similar to .zip) that bundles multiple Java .class files, metadata, and resources (like images or config files) together.

It is commonly used to:

- Package Java applications or libraries.
- Distribute compiled Java code.
- Reuse existing libraries (e.g., mysql-connector.jar).

A. Reading a JAR File

Purpose:

To inspect or extract information from a JAR file — for example, to list all files inside it.

Using Java Code

You can read the contents of a JAR file using the `java.util.jar` package — specifically the `JarFile` and `JarEntry` classes.

Example: Reading Contents of a JAR File

```
import java.io.*;
import java.util.jar.*;
import java.util.Enumeration;
```

```
public class ReadJarExample {
    public static void main(String[] args) {
        try {
            // Specify the JAR file to read
            JarFile jarFile = new JarFile("example.jar");

            // Get all entries (files) inside the JAR
            Enumeration<JarEntry> entries = jarFile.entries();

            System.out.println("Contents of example.jar:");
            while (entries.hasMoreElements()) {
                JarEntry entry = entries.nextElement();
                System.out.println(entry.getName());
            }

            jarFile.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

```
    }  
}
```

B. Updating a JAR File

Purpose:

To add, replace, or modify files in an existing JAR.

Since JAR files are ZIP-based, you can't directly "edit" them.

Instead, the typical process is:

1. Extract the contents.
2. Add or update files.
3. Recreate the JAR file.

Method 1: Using Command Line (Easiest)

```
jar uf example.jar newfile.txt
```

Explanation:

- u → update the JAR.
- f → specifies the JAR file name.
- newfile.txt → file to add inside.

Method 2: Using Java Code

You can use JarOutputStream and JarInputStream to programmatically copy entries and add new ones.

Example: Updating a JAR File (Programmatically)

```
import java.io.*;  
import java.util.jar.*;  
import java.util.Enumeration;
```

```
public class UpdateJarExample {  
    public static void main(String[] args) {  
        String sourceJar = "example.jar";  
        String updatedJar = "updated-example.jar";  
        String newFile = "newfile.txt";  
  
        try {  
            // Open the existing jar  
            JarFile jarFile = new JarFile(sourceJar);  
            JarOutputStream jos = new JarOutputStream(new FileOutputStream(updatedJar));  
  
            // Copy all old entries  
            Enumeration<JarEntry> entries = jarFile.entries();  
            byte[] buffer = new byte[1024];  
            int bytesRead;
```

```

        while (entries.hasMoreElements()) {
            JarEntry entry = entries.nextElement();
            InputStream is = jarFile.getInputStream(entry);
            jos.putNextEntry(new JarEntry(entry.getName()));

            while ((bytesRead = is.read(buffer)) != -1) {
                jos.write(buffer, bytesRead);
            }
            jos.closeEntry();
            is.close();
        }

        // Add a new file entry
        FileInputStream fis = new FileInputStream(newFile);
        jos.putNextEntry(new JarEntry("newfile.txt"));
        while ((bytesRead = fis.read(buffer)) != -1) {
            jos.write(buffer, 0, bytesRead);
        }
        jos.closeEntry();
        fis.close();

        jos.close();
        jarFile.close();

        System.out.println("JAR file updated successfully!");

    } catch (IOException e) {
        e.printStackTrace();
    }
}
}

```

52.What steps are required to handle event by using Delegation Model? Explain event objects.

Event Handling in Java (Delegation Event Model)

In Java's event delegation model, an event generated by a GUI component (like a button) is handled by a separate object called an event listener.

This separates the event source (the component that generates the event) from the event handler (the object that processes it).

Key Concept

- Event Source → The component that generates the event (e.g., Button, TextField).
- Event Listener → The object that wants to be notified when the event occurs.

- Event Object → Contains detailed information about the event (like which button was pressed, mouse coordinates, etc.).

Steps to Handle an Event using Delegation Model

Let's go step-by-step

Step 1: Import Event Packages

You must import the packages that contain event and listener classes:

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

Step 2: Implement a Listener Interface

Create or use a class that implements one of the listener interfaces such as:

- ActionListener
- MouseListener
- KeyListener, etc.

Example:

```
class MyHandler implements ActionListener {
    public void actionPerformed(ActionEvent e) {
        System.out.println("Button clicked!");
    }
}
```

Step 3: Register the Listener with the Source

Tell the event source which listener will handle its events using a registration method like:

```
button.addActionListener(handlerObject);
```

Step 4: Generate the Event

When the user performs an action (like clicking the button), the source creates an Event Object and sends it to the registered listener.

Step 5: Listener Handles the Event

The listener's method (like actionPerformed()) is automatically called, and the event is handled.

Complete Example

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

public class EventDemo extends Frame implements ActionListener {
    Button b;

    EventDemo() {
        b = new Button("Click Me");
        b.setBounds(100, 100, 100, 50);

        add(b);
        b.addActionListener(this); // Step 3: Register listener
        setSize(300, 200);
        setLayout(null);
    }
}
```

```

        setVisible(true);
    }

// Step 2 + 5: Implement listener method
public void actionPerformed(ActionEvent e) {
    System.out.println("Button was clicked!");
}

public static void main(String[] args) {
    new EventDemo(); // Step 4: Event generated when button clicked
}
}

```

Output:



53.Explain the steps used in writing servlets with suitable source code and brief description of each step.

What is a Servlet?

A Servlet is a Java program that runs on a web server and generates dynamic web content (like HTML pages) in response to client requests — typically from a web browser.

Servlets are part of Java EE (Jakarta EE) and are executed inside a Servlet Container (like Apache Tomcat).

Steps for Writing and Running a Servlet

Here are the main steps involved in writing a servlet:

Step 1: Import Required Packages

You need to import the javax.servlet and javax.servlet.http packages.

```

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

```

Step 2: Extend HttpServlet Class

A servlet must extend the HttpServlet class to handle HTTP requests.

```

public class MyServlet extends HttpServlet {
    // code goes here
}

```

Step 3: Override Service Methods

You can override one or more of the following methods based on your needs:

- `doGet()` → Handles HTTP GET requests
- `doPost()` → Handles HTTP POST requests
- `init()` → Initializes the servlet (runs once)
- `destroy()` → Cleans up resources before servlet is destroyed

Example:

```
public void doGet(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {
    // Processing code
}
```

Step 4: Set the Response Content Type

This tells the browser what kind of output to expect (HTML, plain text, etc.)

```
    response.setContentType("text/html");
```

Step 5: Obtain Output Stream

Use a PrintWriter object to send data back to the client (browser):

```
    PrintWriter out = response.getWriter();
```

Step 6: Write the Response

Write the HTML or text response:

```
out.println("<html><body>");
out.println("<h2>Hello, Welcome to My First Servlet!</h2>");
out.println("</body></html>");
```

Step 7: Configure Servlet in web.xml (Deployment Descriptor)

In older servlet versions (before annotations), you must configure the servlet inside the WEB-INF/web.xml file.

```
<web-app>
    <servlet>
        <servlet-name>MyServlet</servlet-name>
        <servlet-class>MyServlet</servlet-class>
    </servlet>
```

```
    <servlet-mapping>
        <servlet-name>MyServlet</servlet-name>
        <url-pattern>/hello</url-pattern>
    </servlet-mapping>
</web-app>
```

Step 8: Compile and Deploy

1. Compile your servlet file using:
2. `javac -classpath "C:\tomcat\lib\servlet-api.jar" MyServlet.java`
3. Place the compiled .class file inside:
4. `webapps\YourApp\WEB-INF\classes\`
5. Restart the Tomcat server.
6. Access in browser:

`http://localhost:8080/YourApp/hello`

Complete Example

File: MyServlet.java

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

```
public class MyServlet extends HttpServlet {  
    public void doGet(HttpServletRequest request, HttpServletResponse response)  
        throws ServletException, IOException {  
  
        // Step 4: Set content type  
        response.setContentType("text/html");  
  
        // Step 5: Get writer  
        PrintWriter out = response.getWriter();  
  
        // Step 6: Write output  
        out.println("<html>");  
        out.println("<head><title>Servlet Example</title></head>");  
        out.println("<body>");  
        out.println("<h2>Hello, this is my first servlet!</h2>");  
        out.println("<p>Request served using GET method.</p>");  
        out.println("</body></html>");  
    }  
}
```

web.xml (Deployment Descriptor)

```
<web-app>  
    <servlet>  
        <servlet-name>MyServlet</servlet-name>  
        <servlet-class>MyServlet</servlet-class>  
    </servlet>
```

```
    <servlet-mapping>  
        <servlet-name>MyServlet</servlet-name>  
        <url-pattern>/hello</url-pattern>  
    </servlet-mapping>  
</web-app>
```

Step 9: Run the Servlet

Open browser and type:

<http://localhost:8080/MyApp/hello>

54.What is Abstract Window Toolkit? Present AWT class hierarchy showing all major components and containers.

AWT (Abstract Window Toolkit) is a Java GUI (Graphical User Interface) framework provided in the `java.awt` package.

It allows developers to create window-based, platform-independent GUI applications.

AWT is called *Abstract* because it depends on the native windowing system (like Windows, Linux, macOS) for rendering GUI components.

Key Features of AWT:

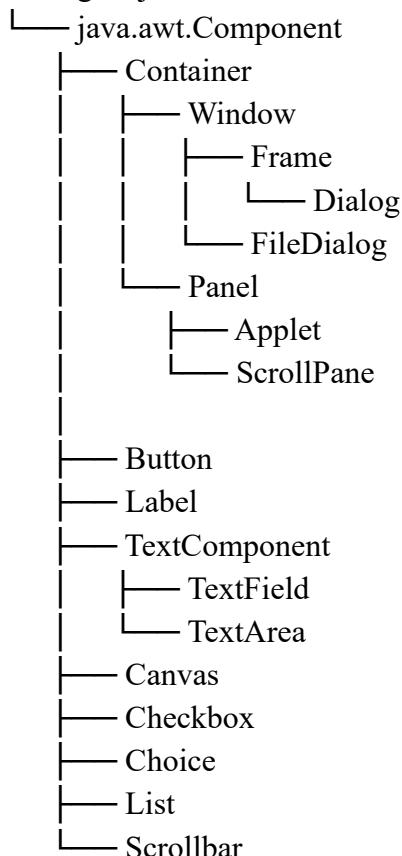
| Feature | Description |
|---------------------------------------|--|
| Platform-dependent | Uses native OS components (called “peers”) for GUI rendering. |
| Component-based | GUI elements like buttons, labels, text fields, etc. |
| Event-driven | Supports Java’s <i>Delegation Event Model</i> for handling user actions. |
| Container-based | Components are placed inside containers like Frame, Panel, etc. |
| Part of <code>java.awt</code> package | All AWT classes are defined inside the <code>java.awt</code> and <code>java.awt.event</code> packages. |

AWT Class Hierarchy

The AWT hierarchy starts from the top-most class `Object` and goes down to specific GUI components.

Here’s the simplified AWT hierarchy showing the major containers and components:

`java.lang.Object`



Explanation of Major Classes:

1. Component (Base Class)

- Superclass for all GUI elements.
- Provides common properties like size, color, font, and event handling.

Example subclasses:

- Button, Label, TextField, TextArea, etc.

2. Container

- Can hold other components (like a layout manager or panel).
- Provides methods like add(), remove(), etc.

Example subclasses:

- Panel, Window, Frame, Dialog.

3. Panel

- Generic container for grouping components.
- Often used inside Frames or Applets.

4. Window

- Top-level window with no borders or menu bar.

Subclasses:

- Frame
- Dialog
- FileDialog

5. Frame

- Main top-level window for most AWT applications.
- Can hold panels, buttons, menus, etc.

Example:

```
Frame f = new Frame("My Frame");
```

6. Dialog

- Popup window used for short-term user interaction.
- Example: confirmation boxes, messages.

7. Button, Label, TextField

- Common UI components for user interaction.

| Component | Description |
|-----------|-------------------------------|
| Button | Clickable element for actions |
| Label | Displays non-editable text |
| TextField | Allows single-line text input |
| TextArea | Multi-line editable text area |

Simple Example: AWT GUI Program

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

```
public class AWTExample {
    public static void main(String[] args) {
        Frame f = new Frame("AWT Example"); // Top-level container

        Label l = new Label("Enter your name:");
        l.setBounds(50, 50, 150, 30);
```

```

TextField t = new TextField();
t.setBounds(200, 50, 150, 30);

Button b = new Button("Submit");
b.setBounds(150, 100, 80, 30);

f.add(l);
f.add(t);
f.add(b);

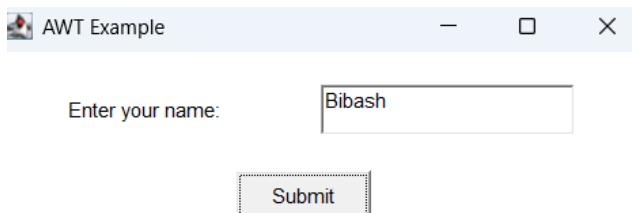
f.setSize(400, 200);
f.setLayout(null);
f.setVisible(true);

}

}

Output:

```



55.What is meant by layout managers? Explain the importance of layout management and discuss the method used in setting layout managers briefly.

A Layout Manager in Java is an object that controls the position and size of components inside a container (like a Frame, Panel, or Applet).

It automatically arranges GUI components (buttons, text fields, labels, etc.) according to a specific layout policy — without manually setting pixel positions.

Why Use Layout Managers? (Importance)

| Importance | Description |
|------------------------------------|---|
| 1. Automatic component arrangement | Layout managers automatically arrange components, reducing manual work. |
| 2. Platform independence | GUI looks consistent on all platforms (Windows, Linux, macOS). |
| 3. Dynamic resizing | Components adjust automatically when the window is resized. |
| 4. Readable and maintainable code | Avoids the complexity of using setBounds() for every component. |
| 5. Flexible layouts | Easy to change the look of GUI just by switching the layout manager. |

Commonly Used Layout Managers in AWT

| Layout Manager | Description | Constructor Example |
|----------------|---|---------------------|
| FlowLayout | Arranges components in a row (left to right). | new FlowLayout() |
| BorderLayout | Divides container into 5 regions: NORTH, SOUTH, EAST, WEST, CENTER. | new BorderLayout() |
| GridLayout | Arranges components in a grid of rows and columns. | new GridLayout(2,3) |
| CardLayout | Allows multiple components to share the same space (like flipping cards). | new CardLayout() |
| GridBagLayout | Flexible and complex layout for professional UI designs. | new GridBagLayout() |
| null layout | No layout manager; you manually set positions using setBounds(). | setLayout(null) |

Methods Used in Setting Layout Managers

1. Setting a Layout

You can assign a layout manager to any container (like Frame or Panel) using:

```
setLayout(new FlowLayout());
```

2. Adding Components

You can add components to the container normally:

```
add(new Button("OK"));
add(new Button("Cancel"));
```

3. Changing Layout Dynamically

You can change the layout at runtime using:

```
setLayout(new GridLayout(2, 2));
validate();
```

Example 1: Using FlowLayout

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

```
public class FlowLayoutExample {
    public static void main(String[] args) {
        Frame f = new Frame("FlowLayout Example");

        f.setLayout(new FlowLayout()); // Step 1: Set Layout

        f.add(new Button("Button 1"));
        f.add(new Button("Button 2"));
        f.add(new Button("Button 3"));

        f.setSize(300, 150);
        f.setVisible(true);
    }
}
```

Output:

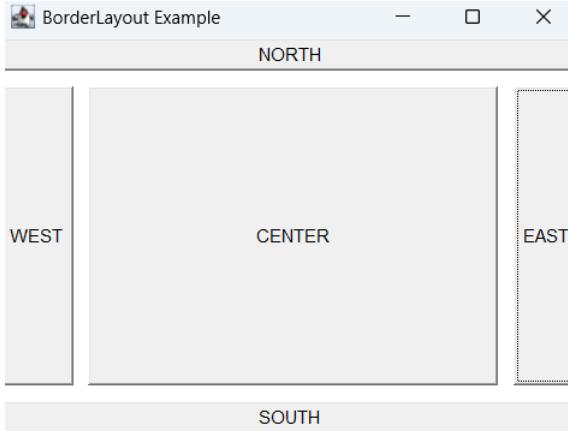


Example 2: Using BorderLayout

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

```
public class BorderLayoutExample {  
    public static void main(String[] args) {  
        Frame f = new Frame("BorderLayout Example");  
  
        f.setLayout(new BorderLayout());  
  
        f.add(new Button("North"), BorderLayout.NORTH);  
        f.add(new Button("South"), BorderLayout.SOUTH);  
        f.add(new Button("East"), BorderLayout.EAST);  
        f.add(new Button("West"), BorderLayout.WEST);  
        f.add(new Button("Center"), BorderLayout.CENTER);  
  
        f.setSize(400, 200);  
        f.setVisible(true);  
    }  
}
```

Output:



Example 3: Using GridLayout

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

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

```

Frame f = new Frame("GridLayout Example");

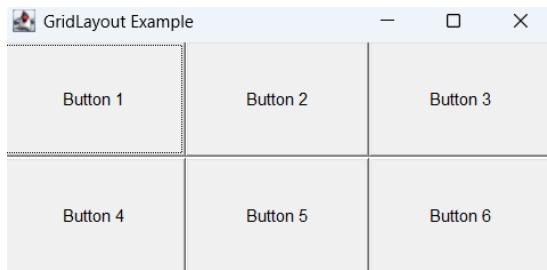
f.setLayout(new GridLayout(2, 3)); // 2 rows, 3 columns

for (int i = 1; i <= 6; i++) {
    f.add(new Button("Button " + i));
}

f.setSize(300, 200);
f.setVisible(true);
}
}

```

Output:



56.What is JDBC? Explain JDBC architecture and components with suitable and brief description.

JDBC (Java Database Connectivity) is an API (Application Programming Interface) provided by Java that allows Java applications to interact with relational databases in a platform-independent manner.

It acts as a bridge between Java applications and databases like MySQL, Oracle, PostgreSQL, etc.

JDBC Architecture

The JDBC architecture consists of two layers:

1. JDBC API Layer
2. JDBC Driver Layer

JDBC API Layer

- This layer provides the application-to-JDBC Manager connection.
- It defines a set of interfaces and classes for connecting to the database, executing SQL statements, and processing results.
- Commonly used interfaces include:
 - DriverManager
 - Connection
 - Statement
 - PreparedStatement
 - ResultSet

JDBC Driver Layer

- This layer handles the actual communication with the database.

- It translates the JDBC API calls into database-specific calls.
- Each database requires its own JDBC driver.

JDBC Components

| Component | Description |
|-------------------|---|
| DriverManager | Manages a list of database drivers. It establishes a connection to the database. |
| Connection | Represents a connection (session) with a specific database. |
| Statement | Used to execute SQL queries like SELECT, INSERT, UPDATE, etc. |
| PreparedStatement | A precompiled version of Statement used for executing parameterized queries efficiently. |
| ResultSet | Represents a table of data returned from executing a query. Allows navigation through the data. |

JDBC Example Program

```

import java.sql.*;

public class JdbcExample {
    public static void main(String[] args) {
        String url = "jdbc:mysql://localhost:3306/testdb";
        String username = "root";
        String password = "";

        try {
            // Step 1: Load the driver
            Class.forName("com.mysql.cj.jdbc.Driver");

            // Step 2: Establish the connection
            Connection con = DriverManager.getConnection(url, username, password);

            // Step 3: Create a statement
            Statement stmt = con.createStatement();

            // Step 4: Execute query
            ResultSet rs = stmt.executeQuery("SELECT * FROM employees");

            // Step 5: Process results
            while (rs.next()) {
                System.out.println("ID: " + rs.getInt("id"));
                System.out.println("Name: " + rs.getString("name"));
            }

            // Step 6: Close connections
            rs.close();
        }
    }
}

```

```

        stmt.close();
        con.close();

    } catch (Exception e) {
        e.printStackTrace();
    }
}
}
}

```

57.What are the different types of drivers used in JDBC? Which is the most used driver? Explain.

In JDBC, drivers are used to connect a Java application with a database.

JDBC defines four types of drivers, each differing in how they communicate between Java applications and the database.

Type 1: JDBC–ODBC Bridge Driver

Description:

- Uses the ODBC (Open Database Connectivity) driver to connect to the database.
- Acts as a bridge between JDBC calls and the ODBC driver.
- Requires ODBC installation on the client machine.

Architecture:

Java Application → JDBC API → JDBC-ODBC Bridge → ODBC Driver → Database

Advantages:

- Easy to use and good for prototyping.

Disadvantages:

- Slow performance.
- Not platform-independent (requires ODBC setup).
- Deprecated from Java 8 onwards.

Example:

```

Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
Connection con = DriverManager.getConnection("jdbc:odbc:myDB");

```

Type 2: Native-API / Partly Java Driver

Description:

- Converts JDBC calls into database-specific native API calls using C/C++ libraries.
- Requires native client libraries to be installed on the system.

Architecture:

Java Application → JDBC API → Native API → Database

Advantages:

- Faster than Type 1 (direct call to native libraries).

Disadvantages:

- Not fully portable (depends on native libraries).
- Requires client-side configuration.

Example:

```

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

```

```
Connection con = DriverManager.getConnection("jdbc:oracle:oci:@myDB", "user",  
"password");
```

Type 3: Network Protocol / Middleware Driver

Description:

- Uses a middleware server between Java application and database.
- The driver sends JDBC calls to the middleware server, which then communicates with the database.

Architecture:

Java Application → JDBC API → Middleware Server → Database

Advantages:

- No client-side native library required.
- Easier to maintain and secure (centralized).

Disadvantages:

- Slower due to the extra middleware layer.

Example:

Used in enterprise applications with middleware servers like WebLogic or WebSphere.

Type 4: Pure Java / Thin Driver

Description:

- Fully written in Java and directly converts JDBC calls into database-specific network protocol calls (like MySQL, Oracle, etc.).
- No need for native libraries or middleware.

Architecture:

Java Application → JDBC API → Database (Directly)

Advantages:

- Platform-independent.
- Fastest performance.
- Easy to deploy (no extra setup).

Disadvantages:

- Each database requires its own Type 4 driver.

Example:

```
Class.forName("com.mysql.cj.jdbc.Driver");  
Connection con =  
DriverManager.getConnection("jdbc:mysql://localhost:3306/testdb", "root",  
"password");
```

58.What is jar file? Explain the process of creating jar file with example.

A JAR (Java ARchive) file is a package file format used to combine multiple Java class files, metadata, and resources (like images, text files, etc.) into a single compressed file.

It is similar to a ZIP file but specifically designed for Java applications and libraries.

Key Uses of JAR Files:

1. Packaging Java classes for distribution.
2. Library storage — reusable code modules.
3. Executable JAR — run Java applications directly using java -jar.

4. Resource management — images, audio, and configuration files can be bundled.

Advantages of JAR Files:

| Advantage | Description |
|-------------|---|
| Compression | Reduces file size by combining multiple files into one. |
| Portability | Easy to distribute and run on any platform. |
| Security | Can be digitally signed to ensure integrity. |
| Convenience | Easier to manage multiple files as one archive. |
| Executable | Can include a Main-Class in MANIFEST.MF to run as an application. |

Steps to Create a JAR File

Step 1: Compile Java Source Files

First, compile your .java files to .class files using javac:

```
javac HelloWorld.java
```

Step 2: Create a Manifest File (Optional)

A manifest file defines metadata for the JAR, like which class contains main() for executable JARs.

Example: manifest.txt

Main-Class: HelloWorld

Step 3: Create the JAR File

Use the jar command to bundle .class files (and optionally the manifest) into a JAR.

Basic JAR command:

```
jar cf myjarfile.jar HelloWorld.class
```

- c → create a new JAR
- f → specify the filename

Executable JAR with manifest:

```
jar cfm HelloWorld.jar manifest.txt HelloWorld.class
```

- m → include manifest file
- f → specify JAR file name
- c → create JAR

Step 4: Run the JAR File (if executable)

```
java -jar HelloWorld.jar
```

Example

1. Java Program: HelloWorld.java

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

2. Compile

```
javac HelloWorld.java
```

3. Create Manifest (manifest.txt)

Main-Class: HelloWorld

4. Create Executable JAR

```
jar cfm HelloWorld.jar manifest.txt HelloWorld.class
```

5. Run JAR

```
java -jar HelloWorld.jar
```

58.What is event handling? Explain two ways of handling events briefly with suitable example.

Event handling in Java is the process of detecting and responding to user actions on GUI components, such as:

- Clicking a button
- Selecting a menu item
- Typing in a text field
- Closing a window

An event is an object that describes a state change or user interaction, and event handling allows the program to respond appropriately.

Event Handling in Java GUI

- Java uses the Delegation Event Model, which separates the event source (component) from the event listener (handler).
- Every event has:
 1. Event Source: Component that generates the event (e.g., Button)
 2. Event Object: Contains information about the event (ActionEvent, MouseEvent)
 3. Event Listener: Object that handles the event (ActionListener, MouseListener)

Two Ways of Handling Events

Using Event Listener Interface

- Method: Create a class that implements a listener interface like ActionListener, then register it with the component.
- Steps:
 1. Implement listener interface
 2. Override its method(s)
 3. Register listener with the component

Example: Using ActionListener for a Button

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

```
public class EventDemo1 extends Frame implements ActionListener {  
    Button b;  
  
    EventDemo1() {  
        b = new Button("Click Me");  
        b.setBounds(100, 100, 100, 50);  
        add(b);  
  
        b.addActionListener(this); // Register listener  
  
        setSize(300, 300);
```

```

        setLayout(null);
        setVisible(true);
    }

    public void actionPerformed(ActionEvent e) {
        System.out.println("Button clicked!");
    }

    public static void main(String[] args) {
        new EventDemo1();
    }
}

```

Output:



Using Anonymous Inner Class

- Method: Create an anonymous class inside the component registration. No need to implement the interface separately.
- Useful for short event handling code.

Example:

```

import java.awt.*;
import java.awt.event.*;

```

```

public class EventDemo2 extends Frame {
    EventDemo2() {
        Button b = new Button("Click Me");
        b.setBounds(100, 100, 100, 50);
        add(b);

        // Anonymous inner class
        b.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                System.out.println("Button clicked using anonymous class!");
            }
        });
    }

    setSize(300, 300);
    setLayout(null);
}

```

```

        setVisible(true);
    }

    public static void main(String[] args) {
        new EventDemo2();
    }
}

```

59.What is Servlet? Explain the position of servlets in web applications with suitable diagram.

A Servlet is a Java program that runs on a web server or servlet container (like Apache Tomcat) to handle client requests and generate dynamic web content.

- Servlets are part of Java EE / Jakarta EE.
- They extend the capabilities of servers and are platform-independent.
- Commonly used to process HTML forms, database queries, and session management.

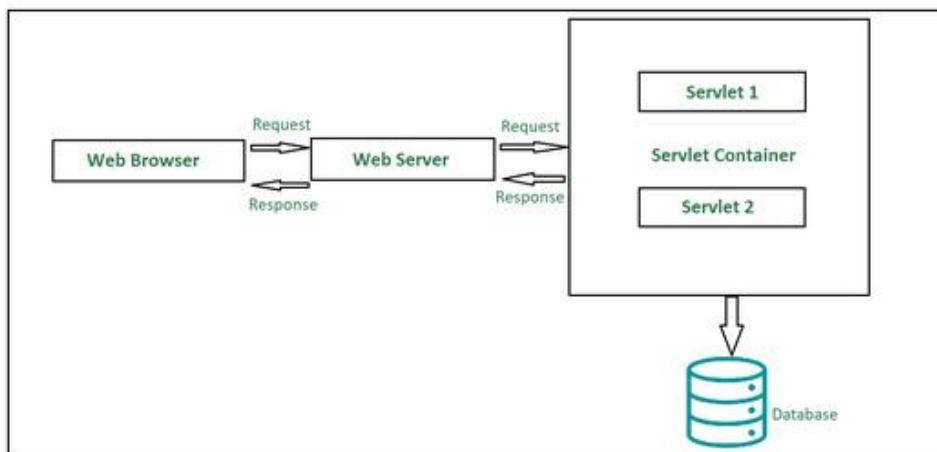
Key Features of Servlets:

| Feature | Description |
|--------------------------|--|
| Platform-independent | Works on any OS with a servlet container |
| Server-side execution | Runs on web server, not client |
| Handles multiple clients | Supports multithreading |
| Dynamic content | Generates HTML, JSON, XML dynamically |
| Lifecycle management | Init → Service → Destroy |

Position of Servlets in Web Applications

Servlets act as the middle layer between the client (browser) and the backend resources (databases, files, APIs).

Servlet Architecture Diagram



Explanation of Servlet Position

1. Client Layer:

- Users interact via browser or mobile app.
- Sends HTTP requests (GET, POST) to the server.

2. Web Server / Servlet Container:
 - Receives client requests.
 - Loads the servlet and manages its lifecycle.
 - Provides services like multithreading, session management, and security.
3. Servlet Layer:
 - Processes request data.
 - Interacts with business logic or database.
 - Generates a dynamic response.
4. Backend Layer:
 - Databases, files, APIs, or business services.
 - Servlets communicate with backend to retrieve or update data.
5. Response to Client:
 - Servlet sends processed HTML, JSON, XML, or other data back to client.

60.Explain JSP access model with suitable diagram.

JSP (JavaServer Pages) is a server-side technology that allows embedding Java code inside HTML pages to create dynamic web content.

The JSP Access Model explains how a JSP page interacts with the web client, server, and Java components to process requests and generate responses.

Key Concepts in JSP Access Model

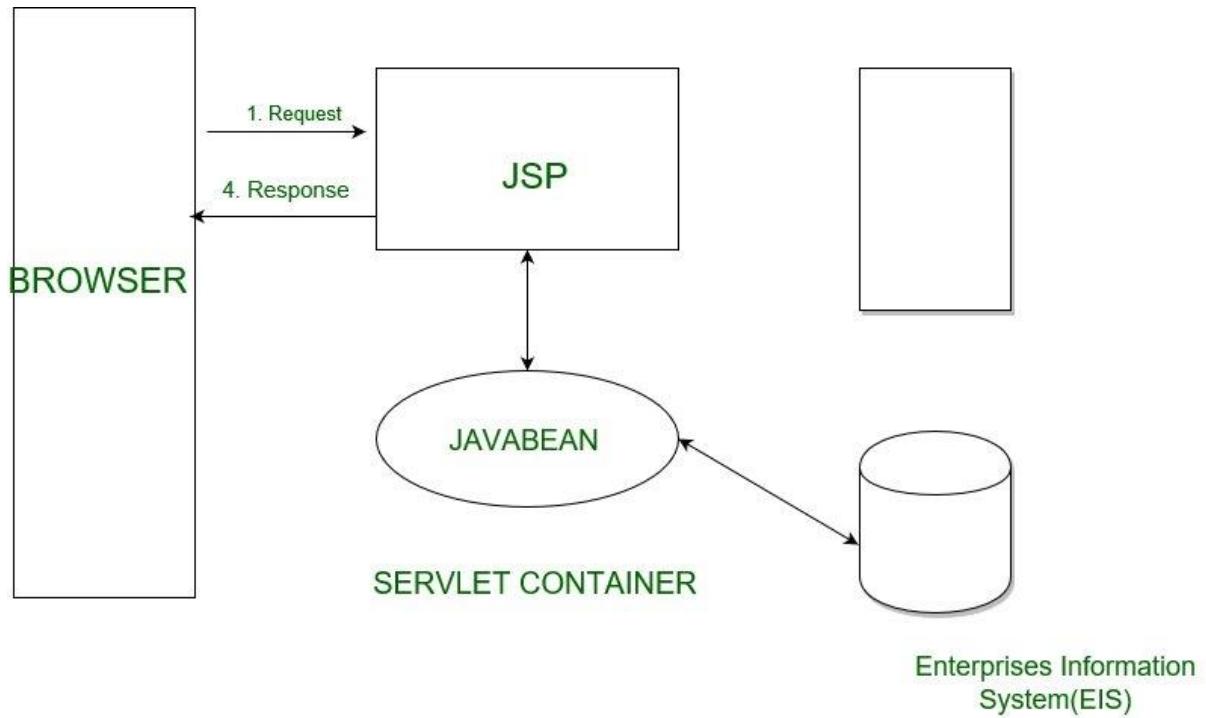
1. Client / Browser
 - Sends an HTTP request to access a JSP page.
 - Request can include form data, URL parameters, or cookies.
2. Web Server / Servlet Container
 - Receives the client request.
 - If the requested page is a JSP, the container translates it into a Servlet.
3. JSP Page as Servlet
 - JSP pages are compiled into Servlets by the container.
 - The servlet contains the Java code extracted from the JSP tags.
4. Servlet Processing
 - Servlet executes Java code embedded in JSP.
 - Interacts with backend resources (databases, files, EJBs).
5. Response Generation
 - Servlet generates dynamic content (HTML, XML, JSON) as a response.
 - Response is sent back to the client browser via HTTP.

JSP Access Model Diagram

JSP Model 1 Architecture: JSP Model1 Architecture or JSP Centric architecture contains JavaBeans or EJB Model Object, View JSP pages and Action JSP Pages. In model 1 architecture, the incoming request is directly sent to the JSP page from a web browser and JSP page is responsible for processing it and sending back to the client. All the data access is performed using beans so there is still a separation of presentation from content. Advantages of JSP Centric Model:

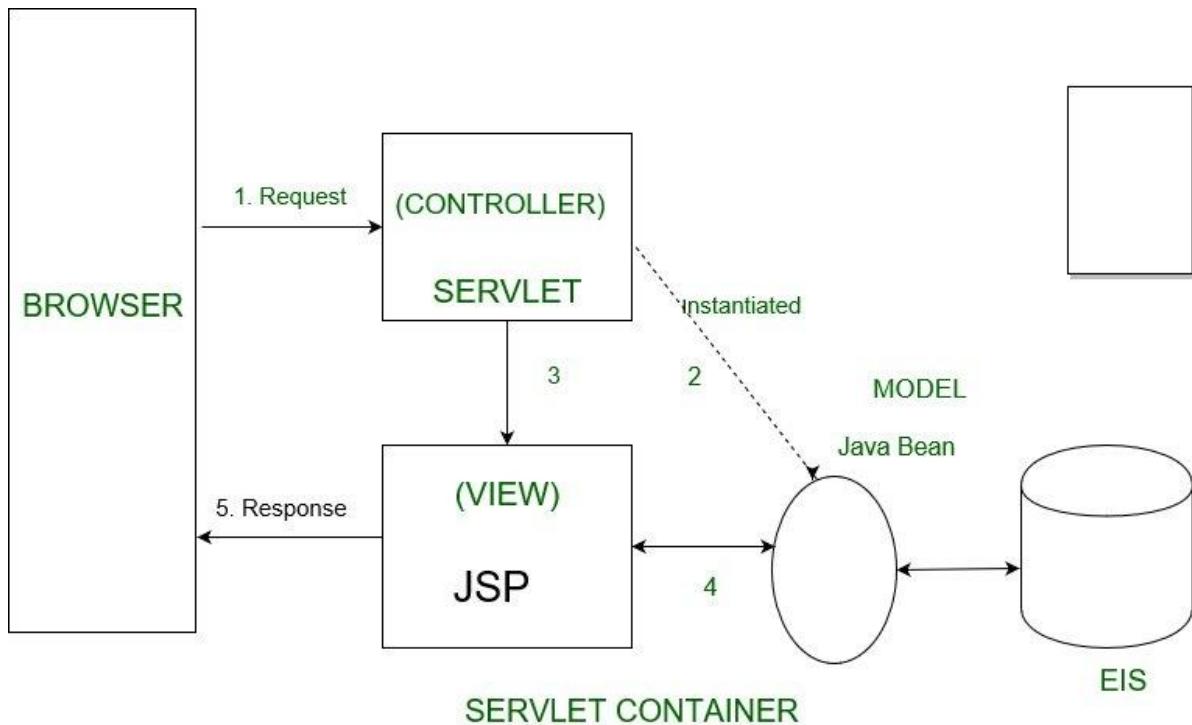
1. Small number of components are required to build the applications.

2. A small number of technologies, reducing the learning curve for inexperienced resources.



Model 1 Architecture

JSP Model 2 Architecture: JSP Model 2 Architecture or Servlet-Centric Architecture contains Java beans or EJB Model Objects, view JSP pages and servlet or command classes. It is basically a Model View Controller approach involved notification/event models, direct manipulation of model objects. MVC basically interposes controller component between View and Model Components where the controller is responsible for navigation, presentation-tier logic, validation and emphasizes the separation of presentation logic and model objects.



Model 2 Architecture

61.What are implicit objects? Explain different implicit objects provided by JSP briefly.

Implicit objects are predefined objects created by the JSP container.

They are automatically available in every JSP page without needing to declare or instantiate them.

These objects help developers access request data, application data, session data, output stream, configuration, and server information easily.

Different Implicit Objects in JSP (with brief explanation)

JSP provides 9 implicit objects, each with a specific purpose:

1. request

- Type: HttpServletRequest
- Used to access:
 - form data (getParameter())
 - headers
 - cookies
 - attributes
 - client information

2. response

- Type: HttpServletResponse
- Used to:
 - set response headers
 - send redirects

- o control output sent to the browser

3. out

- Type: JspWriter
- Used to write data directly to the browser using out.println().

4. session

- Type: HttpSession
- Stores user-specific data across multiple requests.
- Used for login and user tracking.

5. application

- Type: ServletContext
- Represents the entire web application.
- Used to share data across all users and all sessions.

6. config

- Type: ServletConfig
- Provides initialization parameters for a JSP/Servlet.

7. pageContext

- Type: PageContext
- Provides access to all JSP-level objects.
- Used for:
 - o managing attributes
 - o controlling page execution
 - o accessing namespaces

8. page

- Type: Object (this)
- Refers to the current JSP page instance (similar to this in Java).

9. exception

- Type: Throwable
- Available only in error pages (isErrorPage="true").
- Used to display error details.

62.How does Swing differ from AWT? Draw Swing class hierarchy diagram showing containers and components.

Difference between Swing and AWT

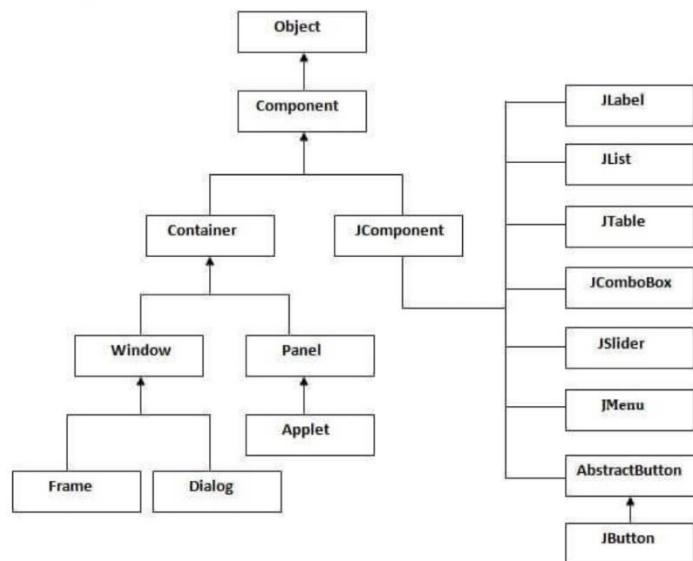
| AWT (Abstract Window Toolkit) | Swing |
|--|--|
| Platform-dependent (uses native OS components) | Platform-independent (written entirely in Java) |
| Heavyweight components | Lightweight components |
| Limited set of components | Rich set of advanced components (JTable, JTree, JTabbedPane, etc.) |
| Slower performance due to native calls | Faster and more flexible |
| UI look depends on OS | Consistent look across all platforms |

| | |
|--|--|
| Less customizable | Highly customizable using Pluggable Look and Feel |
| Components like Button, Label, TextField | Components start with "J": JButton, JLabel, JTextField |

Swing Class Hierarchy Diagram (Simplified)

Hierarchy of Java Swing classes

The hierarchy of java swing API is given below.



Swing is a GUI widget toolkit for Java. It is built on top of the AWT API. Also, it is a part of Oracle's Java Foundation Classes (JFC). Furthermore, Swing provides basic components such as labels, textboxes, buttons, etc. as well as advanced components such as tabbed panes, table, and, trees. Therefore, Swing provides more sophisticated components than AWT. Here, the programmer has to import javax.swing package to write a Swing application. This package provides a number of classes such as JButton, JTable, JList, JTextArea, and, JCheckBox.

Swing is platform-independent and its components are lightweight. Furthermore, the components require minimum memory space. Therefore, Swing applications execute much faster. One common design pattern in development is the Model, View, Controller (MVC) pattern. Swing follows this pattern. It helps to maintain the code easily.

63. Write down steps for writing CORBA programs with suitable example.

Steps for Writing CORBA Programs

Writing a CORBA program involves these major steps:

Step 1: Write the IDL (Interface Definition Language) File

- IDL describes methods/services that the server provides.
- It is language-independent.
- Save the file with .idl extension.

Example: Hello.idl

Step 2: Compile IDL using IDL Compiler

- The IDL compiler generates:
 - Client stub
 - Server skeleton
 - Interfaces for object implementation

Command example:

```
idlj -fall Hello.idl
```

Step 3: Write the Server Program

- Implement the generated skeleton class.
- Initialize ORB.
- Create the servant object.
- Register it with the *Naming Service*.
- Start waiting for client requests.

Step 4: Write the Client Program

- Initialize ORB.
- Use Naming Service to obtain object reference.
- Call methods on the remote object using the stub.

Step 5: Start the ORB / Naming Service

Example:

```
tnameserv -ORBInitialPort 1050
```

Step 6: Run the Server

Start the server so it registers the object with ORB.

Step 7: Run the Client

Client connects to the server and invokes remote methods.

CORBA Example Program

1. IDL File (Hello.idl)

```
module HelloApp {
    interface Hello {
        string sayHello();
    };
};
```

2. Compile the IDL

```
idlj -fall Hello.idl
```

This generates Java classes:

- Hello.java
- HelloHelper.java
- HelloHolder.java
- HelloPOA.java
- _HelloStub.java

3. Server Program (HelloServer.java)

```
import HelloApp.*;
import org.omg.CORBA.*;
import org.omg.CosNaming.*;
import org.omg.PortableServer.*;

public class HelloServer {
    public static void main(String[] args) {
        try {
            ORB orb = ORB.init(args, null);

            POA rootpoa = POAHelper.narrow(orb.resolve_initial_references("RootPOA"));
            rootpoa.the_POAManager().activate();

            HelloImpl helloImpl = new HelloImpl();
            org.omg.CORBA.Object ref = rootpoa.servant_to_reference(helloImpl);
            Hello href = HelloHelper.narrow(ref);

            org.omg.CORBA.Object objRef = orb.resolve_initial_references("NameService");
            NamingContextExt ncRef = NamingContextExtHelper.narrow(objRef);

            NameComponent path[] = ncRef.to_name("Hello");
            ncRef.rebind(path, href);

            System.out.println("Server ready...");
            orb.run();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

Hello Implementation (HelloImpl.java)

```
import HelloApp.*;

public class HelloImpl extends HelloPOA {
    public String sayHello() {
        return "Hello from CORBA Server!";
    }
}
```

4. Client Program (HelloClient.java)

```
import HelloApp.*;
import org.omg.CORBA.*;
import org.omg.CosNaming.*;
```

```

public class HelloClient {
    public static void main(String[] args) {
        try {
            ORB orb = ORB.init(args, null);

            org.omg.CORBA.Object objRef =
                orb.resolve_initial_references("NameService");
            NamingContextExt ncRef = NamingContextExtHelper.narrow(objRef);

            Hello hello = HelloHelper.narrow(ncRef.resolve_str("Hello"));

            System.out.println("Server Response: " + hello.sayHello());

        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}

```

64.What is XML? How it is different from HTML? Explain the structure of XML documents and its components.

XML (eXtensible Markup Language) is a markup language designed to store, describe, and transport data in a structured and readable format.

- It does not define how data looks, but defines what data is.
- It is self-descriptive and platform-independent.
- Data is stored using user-defined tags.

Difference Between XML and HTML

| HTML | XML |
|--|---|
| Used to display data on web pages. | Used to store and transport data. |
| Has predefined tags like <p>, <table>, <h1> | Tags are user-defined, e.g., <student>, <book> |
| Focus: How data looks (presentation) | Focus: What data is (structure) |
| Not strict with syntax | Very strict syntax (case-sensitive, tags must close) |
| Allows formatting | No formatting; only data representation |
| Browser renders HTML | XML is processed by programs/software |

Structure of an XML Document

A well-formed XML document has the following components:

1. XML Declaration (optional but recommended)

Specifies XML version and encoding.

<?xml version="1.0" encoding="UTF-8"?>

2. Root Element (Mandatory)

Every XML file must contain one and only one root element.

Example:

```
<studentRecord> ... </studentRecord>
```

3. Child Elements

Elements inside the root element that store data.

```
<name>John</name>
```

```
<age>21</age>
```

4. Attributes

Used to provide additional information inside tags.

```
<student id="101">
```

5. Text Content

Data stored inside tags.

```
<course>Computer Science</course>
```

6. Comments

Used to describe or document the XML content.

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

7. Empty Elements

Elements that have no content.

```
<email />
```

or

```
<email></email>
```

Example of a Well-Structured XML Document

```
<?xml version="1.0" encoding="UTF-8"?>
```

```
<studentRecord>
  <student id="101">
    <name>John Doe</name>
    <age>21</age>
    <course>Computer Science</course>
    <marks>
      <math>85</math>
      <science>90</science>
    </marks>
  </student>
</studentRecord>
```

65.What is web application? Explain the structure of web application with suitable diagram.

A web application is a software application that runs on a web server and is accessed by users through a web browser using the Internet or an intranet.

Examples:

- Online shopping sites (Amazon)
- Social media (Facebook)

- Online banking
- Email services (Gmail)

A web application typically involves client-server architecture, where the browser (client) sends requests and the server processes them and sends responses.

Structure of a Web Application

A web application is usually divided into three main layers:

1. Client Layer (Presentation Layer)

- Runs in the web browser.
- Contains HTML, CSS, JavaScript.
- Responsible for UI/UX, forms, display, and user interaction.

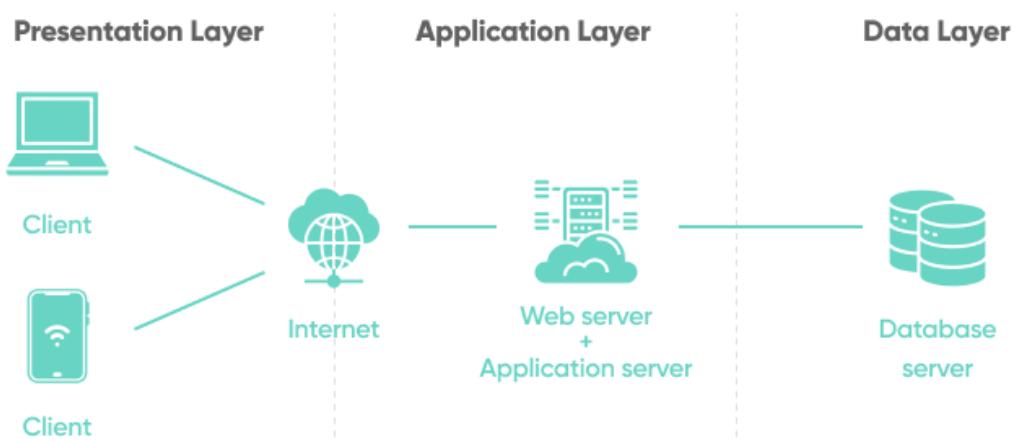
2. Server Layer (Business Logic Layer)

- Contains code that processes user requests.
- Technologies: Java (Servlet, JSP), PHP, Python, Node.js, .NET, etc.
- Validates input, performs business logic, manages sessions, etc.

3. Database Layer (Data Layer)

- Stores and retrieves data.
- Databases: MySQL, Oracle, PostgreSQL, MongoDB, etc.

Diagram: Structure of a Web Application



66. Discuss Swing and MVC Design Pattern.

Swing

Swing is a GUI (Graphical User Interface) toolkit in Java used to create window-based applications.

Features of Swing

- Pure Java, platform independent
- Lightweight components
- Highly customizable using pluggable Look and Feel
- Rich set of components: JButton, JFrame, JTable, JTree, etc.
- Event-driven programming model

Swing Architecture

Swing follows a Model–View–Controller (MVC)–based architecture, but in a modified form.

How Swing Uses MVC

Swing uses a modified MVC architecture called "Separable Model Architecture":

- Each component (like JTable, JList) has a separate Model class.
- The component acts as both View and Controller.
- Example:
 - JTable → View
 - TableModel → Model
 - JTable + renderer/listener → Controller

Thus, Swing achieves a clean separation between data, UI, and event handling.

67. Write short notes on Java Web Framework and Servlets.

Java Web Framework

A Java Web Framework is a software framework that simplifies the development of web applications by providing:

- Ready-made libraries
- MVC architecture
- Database integration
- Routing, security, validation
- Simplified request/response handling

Popular Java Web Frameworks

1. Spring MVC / Spring Boot
 - Most widely used
 - Inbuilt MVC, security, REST API, ORM support
2. Struts
 - Action-based MVC framework
3. JSF (JavaServer Faces)
 - Component-based UI framework
4. Hibernate
 - ORM framework for database operations (not a web framework but used with web apps)

Advantages

- Less boilerplate code
- Faster development
- Better structure (MVC)
- More secure and scalable

Servlets

A Servlet is a Java class that runs on a web server and handles client (browser) requests and generates responses.

Features

- Part of Java EE
- Platform independent
- Uses HTTP protocol
- Efficient and scalable

Servlet Life Cycle

1. init() → runs once when servlet loads
2. service() → handles each request
3. destroy() → runs before servlet is removed from memory

Basic Servlet Example

```
public class HelloServlet extends HttpServlet {
    protected void doGet(HttpServletRequest req, HttpServletResponse res)
        throws IOException {
        res.getWriter().println("Hello from Servlet");
    }
}
```

Advantages of Servlets

- Faster than CGI
- Multithreading supported
- Easy to integrate with JSP, JDBC
- Used as the base for all Java web frameworks

Question for 10 Marks:

1.How swing differ from AWT? Write a program in swing to take any two numbers in two different text fields then find their sum by pressing add button and display sum to third text field.

How Swing Differs from AWT

| AWT (Abstract Window Toolkit) | Swing |
|---|--|
| Heavyweight components (depend on OS) | Lightweight components (pure Java) |
| Limited GUI components | Rich GUI components (JTable, JTree, JTabbedPane) |
| Look and feel depends on OS | Platform-independent, consistent look & feel |
| Less customizable | Highly customizable (pluggable look & feel) |
| Slower in rendering because of native calls | Faster and flexible |
| Uses Frame, Button, TextField | Uses JFrame, JButton, JTextField |

Swing Program: Sum of Two Numbers

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

public class SumCalculator extends JFrame implements ActionListener {

    // Components
    private JTextField num1Field, num2Field, sumField;
    private JButton addButton;
```

```

// Constructor
public SumCalculator() {
    // Frame properties
    setTitle("Sum Calculator");
    setSize(400, 200);
    setLayout(new GridLayout(4, 2, 10, 10));
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

    // Create labels and text fields
    JLabel num1Label = new JLabel("Number 1:");
    JLabel num2Label = new JLabel("Number 2:");
    JLabel sumLabel = new JLabel("Sum:");

    num1Field = new JTextField();
    num2Field = new JTextField();
    sumField = new JTextField();
    sumField.setEditable(false);

    // Create button
    addButton = new JButton("Add");
    addButton.addActionListener(this);

    // Add components to frame
    add(num1Label); add(num1Field);
    add(num2Label); add(num2Field);
    add(sumLabel); add(sumField);
    add(new JLabel()); add(addButton); // empty label to align button

    setVisible(true);
}

// Handle button click
@Override
public void actionPerformed(ActionEvent e) {
    try {
        double num1 = Double.parseDouble(num1Field.getText());
        double num2 = Double.parseDouble(num2Field.getText());
        double sum = num1 + num2;
        sumField.setText(String.valueOf(sum));
    } catch (NumberFormatException ex) {
        JOptionPane.showMessageDialog(this, "Please enter valid numbers",
                "Error", JOptionPane.ERROR_MESSAGE);
    }
}

```

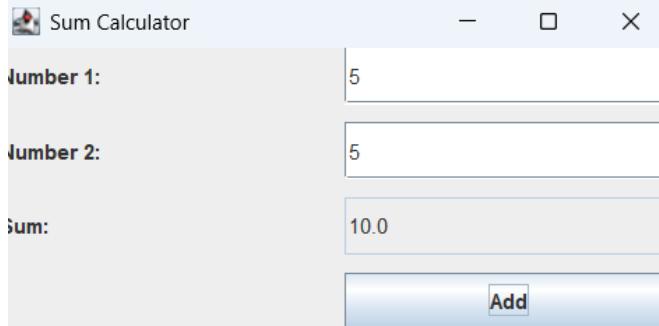
```

        }
    }

// Main method
public static void main(String[] args) {
    new SumCalculator();
}
}

```

Output:



2. How do you track session in Servlets? Explain.

A session is a series of requests from the same user during a single visit to a web application. Since HTTP is stateless, Servlets provide mechanisms to track sessions.

Ways to Track Session in Servlets

1. Using Cookies

- A cookie is a small piece of data stored on the client browser.
- Servlet can create a cookie with user/session info and send it to the browser.
- Browser sends the cookie back with every subsequent request.
- Example:

```
Cookie c = new Cookie("username", "Bibash");
response.addCookie(c);
```

Advantages:

- Simple
- Works across requests

Disadvantages:

- Users may disable cookies
- Limited size (4KB per cookie)

2. Using URL Rewriting

- Session ID is appended to the URL when cookies are disabled.
- Example:

```
String url = response.encodeURL("welcome.jsp");
```

- encodeURL() appends session ID to URL automatically if cookies are disabled.

Advantages:

- Works even if cookies are disabled

Disadvantages:

- Makes URLs messy
- Less secure (session ID visible in URL)

3. Using Hidden Form Fields

- Session ID or user info is sent as a hidden field in HTML forms.
- Example:

```
<form action="nextServlet" method="post">
  <input type="hidden" name="sessionID" value="12345">
  <input type="submit" value="Next">
</form>
```

Advantages:

- Works without cookies
- Simple for forms

Disadvantages:

- Only works with forms
- Less secure (visible in page source)

4. Using HttpSession Object (Most Common)

- HttpSession is a built-in object in Servlets.
- Tracks user session on the server.
- Automatically generates a unique session ID.

Example:

```
// Getting or creating a session
HttpSession session = request.getSession();

// Store attributes
session.setAttribute("username", "Bibash");

// Retrieve attributes
String user = (String) session.getAttribute("username");

// Set session timeout (optional)
session.setMaxInactiveInterval(10*60); // 10 minutes

// Invalidate session
// session.invalidate();
```

Advantages:

- Most secure
- Server-side storage
- Can store multiple objects (attributes)
- Supports timeout and session invalidation

3. Write a GUI application to find sum and difference of two integer numbers. Use two text fields for input and third text field for output. Your program should display sum if the user presses the mouse and difference if the user releases the mouse.

Java Swing Program: Sum and Difference

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

public class SumDifferenceCalculator extends JFrame implements MouseListener {

    // Components
    private JTextField num1Field, num2Field, resultField;
    private JLabel num1Label, num2Label, resultLabel;

    public SumDifferenceCalculator() {
        // Frame properties
        setTitle("Sum & Difference Calculator");
        setSize(400, 200);
        setLayout(new GridLayout(4, 2, 10, 10));
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

        // Labels
        num1Label = new JLabel("Number 1:");
        num2Label = new JLabel("Number 2:");
        resultLabel = new JLabel("Result:");

        // Text Fields
        num1Field = new JTextField();
        num2Field = new JTextField();
        resultField = new JTextField();
        resultField.setEditable(false);

        // Add components to frame
        add(num1Label); add(num1Field);
        add(num2Label); add(num2Field);
        add(resultLabel); add(resultField);

        // Empty label to align mouse area
        JLabel mouseLabel = new JLabel("Press/Release here");
        mouseLabel.setHorizontalAlignment(SwingConstants.CENTER);
        mouseLabel.setBorder(BorderFactory.createLineBorder(Color.BLACK));
        mouseLabel.addMouseListener(this);
        add(new JLabel()); // placeholder
    }

    public void mouseClicked(MouseEvent e) {
        int num1 = Integer.parseInt(num1Field.getText());
        int num2 = Integer.parseInt(num2Field.getText());
        int result = num1 - num2;
        resultField.setText(String.valueOf(result));
    }

    public void mouseEntered(MouseEvent e) {}

    public void mouseExited(MouseEvent e) {}

    public void mousePressed(MouseEvent e) {}

    public void mouseReleased(MouseEvent e) {}
}
```

```

        add(mouseLabel);

        setVisible(true);
    }

// MouseListener methods
@Override
public void mousePressed(MouseEvent e) {
    try {
        int num1 = Integer.parseInt(num1Field.getText());
        int num2 = Integer.parseInt(num2Field.getText());
        int sum = num1 + num2;
        resultField.setText(String.valueOf(sum));
    } catch (NumberFormatException ex) {
        JOptionPane.showMessageDialog(this, "Please enter valid integers",
                "Error", JOptionPane.ERROR_MESSAGE);
    }
}

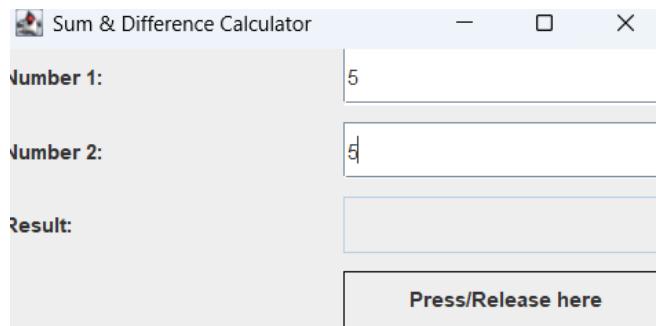
@Override
public void mouseReleased(MouseEvent e) {
    try {
        int num1 = Integer.parseInt(num1Field.getText());
        int num2 = Integer.parseInt(num2Field.getText());
        int diff = num1 - num2;
        resultField.setText(String.valueOf(diff));
    } catch (NumberFormatException ex) {
        JOptionPane.showMessageDialog(this, "Please enter valid integers",
                "Error", JOptionPane.ERROR_MESSAGE);
    }
}

// Unused mouse events
@Override public void mouseClicked(MouseEvent e) {}
@Override public void mouseEntered(MouseEvent e) {}
@Override public void mouseExited(MouseEvent e) {}

// Main method
public static void main(String[] args) {
    new SumDifferenceCalculator();
}
}

```

Output:



4.What is Bean design pattern? Explain simple, Boolean and indexed property design pattern with suitable example in detail.

Bean Design Pattern

JavaBeans are reusable Java classes that follow certain conventions. They are used to create modular, reusable, and maintainable components.

The Bean Design Pattern refers to designing classes in such a way that they can be used as JavaBeans, following these conventions:

1. Private fields (properties)
2. Public getter and setter methods for accessing properties
3. A no-argument constructor
4. Serializable (optional, for persistence)

Types of Properties in JavaBeans

1. Simple Property

- Stores a single value.
- Accessed through getter and setter methods.
- Follows naming convention:
 - `getPropertyName()` → getter
 - `setPropertyName(Type value)` → setter

Example: Simple Property

```
import java.io.Serializable;
```

```
public class Student implements Serializable {  
    private String name; // simple property  
  
    // No-argument constructor  
    public Student() {}  
  
    // Getter method  
    public String getName() {  
        return name;  
    }
```

```
// Setter method
public void setName(String name) {
    this.name = name;
}
```

Usage:

```
Student s = new Student();
s.setName("Bibash");      // Set value
System.out.println(s.getName()); // Get value
```

2. Boolean Property

- Special type of simple property storing true/false.
- Getter method uses isPropertyName() instead of getPropertyName().

Example: Boolean Property

```
import java.io.Serializable;
```

```
public class Light implements Serializable {
    private boolean on; // boolean property

    public Light() {}
```

```
// Getter for boolean property
public boolean isOn() {
    return on;
}
```

```
// Setter
public void setOn(boolean on) {
    this.on = on;
}
```

Usage:

```
Light l = new Light();
l.setOn(true);      // Turn on light
System.out.println(l.isOn()); // Check light status
```

3. Indexed Property

- Represents a list or array of values.
- Getter and setter access specific index.
- Example: Array of numbers, students, etc.

Example: Indexed Property

```
import java.io.Serializable;
```

```
public class Marks implements Serializable {
    private int[] scores = new int[5]; // indexed property
```

```

public Marks() {}

// Getter for single index
public int getScore(int index) {
    return scores[index];
}

// Setter for single index
public void setScore(int index, int value) {
    scores[index] = value;
}

// Getter for entire array
public int[] getScores() {
    return scores;
}

// Setter for entire array
public void setScores(int[] scores) {
    this.scores = scores;
}
}

```

Usage:

```

Marks m = new Marks();
m.setScore(0, 85);      // Set score at index 0
m.setScore(1, 90);      // Set score at index 1
System.out.println(m.getScore(0)); // Get score at index 0

```

5.Define RMI. What is stub and parameter marshalling? Write a client/server application using RMI to find the product of two numbers.

RMI (Remote Method Invocation) is a Java API that allows an object running in one Java virtual machine (JVM) to invoke methods on an object running in another JVM, possibly on a different machine.

- Enables distributed applications in Java.
- Works on client-server architecture.

Stub:

- Acts as a client-side proxy for the remote object.
- Provides the same methods as the remote object.
- Forwards the client request to the remote server object.

Skeleton:

- (Server-side proxy)
- Receives requests from stub and forwards them to the actual remote object.

- Note: From Java 2 SDK v1.2 onwards, skeleton is optional.

Parameter Marshalling

- Marshalling: Converting parameters/objects into a byte stream to send over the network.
- Unmarshalling: Converting byte stream back to objects at the receiver end.
- RMI automatically performs marshalling and unmarshalling for remote method calls.

RMI Client/Server Application Example

We will write an application to compute the product of two numbers using RMI.

Step 1: Remote Interface (ProductInterface.java)

```
import java.rmi.*;
```

```
public interface ProductInterface extends Remote {
    int multiply(int a, int b) throws RemoteException;
}
```

Step 2: Server Implementation (ProductServer.java)

```
import java.rmi.*;
import java.rmi.server.*;
```

```
public class ProductServer extends UnicastRemoteObject implements ProductInterface {
```

```
// Constructor
public ProductServer() throws RemoteException {
    super();
}
```

```
// Implement remote method
public int multiply(int a, int b) throws RemoteException {
    return a * b;
}
```

```
public static void main(String[] args) {
    try {
        // Create server object
        ProductServer obj = new ProductServer();
```

```
        // Bind to RMI registry
        Naming.rebind("rmi://localhost:1099/ProductService", obj);
```

```
        System.out.println("Product Server is ready... ");
    } catch (Exception e) {
        e.printStackTrace();
    }
}
```

```

        }
    }

Step 3: Client Program (ProductClient.java)
import java.rmi.*;

public class ProductClient {
    public static void main(String[] args) {
        try {
            // Lookup remote object from RMI registry
            ProductInterface obj = (ProductInterface)
                Naming.lookup("rmi://localhost:1099/ProductService");

            // Call remote method
            int a = 5, b = 10;
            int result = obj.multiply(a, b);

            System.out.println("Product of " + a + " and " + b + " is: " + result);
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}

```

Step 4: Steps to Run the RMI Program

1. Compile all classes:

javac *.java

2. Start RMI registry:

rmiregistry

3. Run the Server:

java ProductServer

4. Run the Client:

java ProductClient

Output on client:

Product of 5 and 10 is: 50

6.Create a servlet that displays two text boxes in web browser, reads number entered in first text box, and calculates factorial and displays it in second text box.

Factorial Servlet Program

File: FactorialServlet.java

```

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 FactorialServlet extends HttpServlet {

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

        response.setContentType("text/html");
        PrintWriter out = response.getWriter();

        // Read number from text field
        String numStr = request.getParameter("num");
        long fact = 0;

        if (numStr != null && !numStr.equals("")) {
            int n = Integer.parseInt(numStr);
            fact = factorial(n);
        }

        out.println("<html><body>");
        out.println("<h2>Factorial Calculator</h2>");

        out.println("<form method='GET'>");

        out.println("Enter Number: <input type='text' name='num' value='" +
                   (numStr != null ? numStr : "") + "' /><br><br>");

        out.println("Factorial: <input type='text' value='" +
                   (numStr != null ? fact : "") + "' readonly /><br><br>");

        out.println("<input type='submit' value='Calculate' />");
        out.println("</form>");

        out.println("</body></html>");
    }

    // Function to calculate factorial
    private long factorial(int n) {
        long f = 1;
        for (int i = 1; i <= n; i++) {
            f *= i;
        }
    }
}
```

```

    }
    return f;
}
}

web.xml Configuration
<servlet>
    <servlet-name>FactorialServlet</servlet-name>
    <servlet-class>FactorialServlet</servlet-class>
</servlet>

<servlet-mapping>
    <servlet-name>FactorialServlet</servlet-name>
    <url-pattern>/factorial</url-pattern>
</servlet-mapping>

```

7.What is CORBA? Compare it with RMI. Discuss the steps used in creating CORBA programs.

CORBA (Common Object Request Broker Architecture)

- A standard defined by the OMG (Object Management Group).
- Allows objects written in different programming languages to communicate over a network.
- Uses IDL (Interface Definition Language) for defining remote interfaces.
- Platform-independent and language-independent.

CORBA uses an ORB (Object Request Broker) to locate objects, send requests, and return responses.

Comparison: CORBA vs RMI

| Aspect | CORBA | RMI |
|----------------------|--|------------------------------------|
| Language Support | Language-independent (Java, C++, Python, etc.) | Java-only |
| Interface Definition | Uses IDL | Uses Java Interfaces |
| Communication Model | Uses ORB (Object Request Broker) | Uses RMI registry |
| Data Representation | CDR (Common Data Representation) | Java serialization |
| Speed | Fast for multi-language systems | Fast for pure Java |
| Complexity | More complex | Easier to implement |
| Usage | Heterogeneous distributed systems | Pure Java distributed applications |
| Protocol | IIOP (Internet Inter-Orb Protocol) | JRMP (Java Remote Method Protocol) |

Steps for Creating CORBA Programs

To develop a CORBA application, follow these five standard steps:

Step 1: Define the Remote Interface (IDL File)

Write an IDL file describing methods to be called remotely.

Example: product.idl

```
module MyModule {  
    interface Product {  
        long multiply(in long a, in long b);  
    };  
};
```

Step 2: Compile IDL File

Use IDL compiler to generate:

- Client stubs
- Server skeletons
- Language bindings

Example:

```
idlj -fall product.idl
```

Output folders:

```
MyModule/  
    Product.java  
    _ProductStub.java  
    ProductOperations.java  
    ProductPOA.java // server skeleton
```

Step 3: Write the Server Program

Implement the generated skeleton.

```
import MyModule.*;  
import org.omg.CosNaming.*;  
import org.omg.CORBA.*;  
  
public class ProductServer extends ProductPOA {  
  
    ORB orb;  
  
    public void setORB(ORB orb_val) {  
        orb = orb_val;  
    }  
  
    public int multiply(int a, int b) {  
        return a * b;  
    }  
  
    public static void main(String args[]) {  
        try {  
            ORB orb = ORB.init(args, null);  
  
            ProductServer serverObj = new ProductServer();
```

```

serverObj.setORB(orb);

org.omg.CORBA.Object ref = serverObj._this(orb);

NamingContextExt nc = NamingContextExtHelper.narrow(
    orb.resolve_initial_references("NameService"));

nc.rebind(nc.to_name("ProductService"), ref);

System.out.println("CORBA Server ready...");

orb.run();

} catch (Exception e) { e.printStackTrace(); }
}
}

```

Step 4: Write the Client Program

```

import MyModule.*;
import org.omg.CosNaming.*;
import org.omg.CORBA.*;

public class ProductClient {
    public static void main(String args[]) {
        try {
            ORB orb = ORB.init(args, null);

            NamingContextExt nc = NamingContextExtHelper.narrow(
                orb.resolve_initial_references("NameService"));

            Product obj = ProductHelper.narrow(nc.resolve_str("ProductService"));

            int result = obj.multiply(5, 10);

            System.out.println("Product = " + result);

        } catch (Exception e) { e.printStackTrace(); }
    }
}

```

Step 5: Run the CORBA System

1. Compile all Java files:

```
javac *.java MyModule/*.java
```

2. Start ORB Naming Service:

```
tnameserv
```

3. Run server:

```
java ProductServer
```

4. Run client:

```
java ProductClient
```

8.How dialog boxes can be created in Java by using JDialog class? Explain with Java code.

In Java Swing, a dialog box is a small popup window used to display messages or take input from the user.

To create dialog boxes manually, we use the JDialog class.

Key Features of JDialog

- It is a top-level window, but smaller than a JFrame.
- It can be modal or non-modal.
 - Modal dialog blocks the parent window until closed.
 - Non-modal dialog allows interacting with parent window even when shown.
- You can add components like buttons, labels, text fields inside the dialog.

Steps to Create a Dialog Using JDialog

1. Create an instance of JDialog
2. Set its title, size, and modality
3. Add necessary components (labels, text fields, buttons, etc.)
4. Make it visible using setVisible(true)

Example: Creating a JDialog in Swing

Main Frame + Custom Dialog

```
import javax.swing.*;  
import java.awt.*;  
import java.awt.event.*;  
  
public class DialogExample extends JFrame {  
  
    public DialogExample() {  
        setTitle("Main Window");  
        setSize(400, 300);  
        setLayout(new FlowLayout());  
  
        JButton btn = new JButton("Open Dialog");  
        add(btn);  
  
        btn.addActionListener(new ActionListener() {  
            public void actionPerformed(ActionEvent e) {  
                showMyDialog();  
            }  
        });  
    }  
}
```

```

        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setVisible(true);
    }

    // Method to create dialog
    private void showMyDialog() {
        JDialog dialog = new JDialog(this, "My Custom Dialog", true);
        // true = modal dialog

        dialog.setSize(300, 200);
        dialog.setLayout(new FlowLayout());

        JLabel label = new JLabel("This is a dialog box!");
        JButton closeBtn = new JButton("Close");

        dialog.add(label);
        dialog.add(closeBtn);

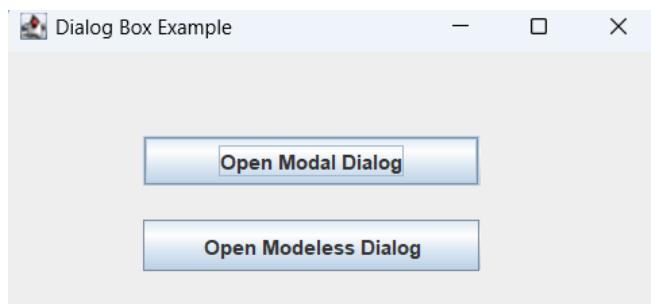
        closeBtn.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                dialog.dispose(); // close dialog
            }
        });
    }

    dialog.setLocationRelativeTo(this); // center relative to parent
    dialog.setVisible(true);
}

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

```

Output:



9. Write server and client program by using RMI such that the program finds factorial of n.

Remote Interface – FactorialInterface.java

```
import java.rmi.*;
```

```
public interface FactorialInterface extends Remote {  
    long factorial(int n) throws RemoteException;  
}
```

Server Program – FactorialServer.java

```
import java.rmi.*;  
import java.rmi.server.*;
```

```
public class FactorialServer extends UnicastRemoteObject implements FactorialInterface {
```

```
// Constructor
```

```
public FactorialServer() throws RemoteException {  
    super();  
}
```

```
// Remote method: factorial logic
```

```
public long factorial(int n) throws RemoteException {  
    long fact = 1;  
    for (int i = 1; i <= n; i++) {  
        fact *= i;  
    }  
    return fact;  
}
```

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

```
    try {  
        // Create object  
        FactorialServer server = new FactorialServer();
```

```
        // Bind remote object in registry
```

```
        Naming.rebind("rmi://localhost:1099/FactorialService", server);
```

```
        System.out.println("Factorial Server is running...");
```

```
    } catch (Exception e) {  
        e.printStackTrace();  
    }  
}
```

```
}
```

```

Client Program – FactorialClient.java
import java.rmi.*;

public class FactorialClient {
    public static void main(String[] args) {

        try {
            // Lookup remote object
            FactorialInterface obj =
                (FactorialInterface) Naming.lookup("rmi://localhost:1099/FactorialService");

            int n = 5; // example input
            long result = obj.factorial(n);

            System.out.println("Factorial of " + n + " = " + result);

        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}

```

10.What do you mean by tooltip text and accelerator keys? Explain their application with proper java code.

Tooltip Text

A tooltip is a small pop-up message that appears when the mouse pointer hovers over a component (button, text field, menu item, etc.).

Purpose

- To provide extra information or help text to the user.
- Helps explain what a button or component does.

How to set tooltip

```
component.setToolTipText("This is tooltip text");
```

Accelerator Keys

Accelerator keys are keyboard shortcuts used to activate menu items without using the mouse.

Examples:

- Ctrl + S → Save
- Ctrl + C → Copy
- Ctrl + X → Cut

How to set accelerator

```
menuItem.setAccelerator(KeyStroke.getKeyStroke(KeyEvent.VK_S,
InputEvent.CTRL_DOWN_MASK));
```

Accelerators work even if the menu is not open.

```
Java Program Showing Tooltip Text & Accelerator Keys
import javax.swing.*;
import java.awt.event.*;

public class ToolTipAndAcceleratorExample extends JFrame {

    public ToolTipAndAcceleratorExample() {

        // Frame Properties
        setTitle("Tooltip & Accelerator Example");
        setSize(400, 300);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

        // Button with Tooltip
        JButton btn = new JButton("Click Me");
        btn.setToolTipText("This button shows a tooltip on hover");

        add(btn);

        // Menu Bar
        JMenuBar menuBar = new JMenuBar();
        JMenu fileMenu = new JMenu("File");

        // Menu Item with Accelerator Key
        JMenuItem saveItem = new JMenuItem("Save");
        saveItem.setToolTipText("Save the document (Ctrl + S)");

        // Setting accelerator key: Ctrl + S
        saveItem.setAccelerator(KeyStroke.getKeyStroke(KeyEvent.VK_S,
            InputEvent.CTRL_DOWN_MASK));

        // Add Menu Item
        fileMenu.add(saveItem);
        menuBar.add(fileMenu);
        setJMenuBar(menuBar);

        // Action listener for Save
        saveItem.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                JOptionPane.showMessageDialog(null, "Save option selected!");
            }
        });
    }
}
```

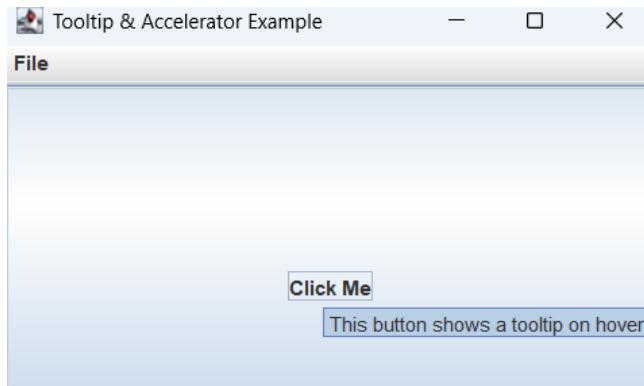
```

        setVisible(true);
    }

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

```

Output:



11. How JSP is different from Servlets? Explain advantages of JSP over other server-side scripting languages.

Difference Between JSP and Servlet

| Servlet | JSP (JavaServer Pages) |
|---|---|
| Servlet is a Java class used to handle HTTP requests and responses. | JSP is an HTML-based page with Java code embedded inside. |
| Designed mainly for processing (controller logic). | Designed mainly for presentation (view/UI layer). |
| Writing HTML inside servlet becomes complex and messy (using out.println() statements). | Writing HTML in JSP is easier because it is similar to writing normal webpages. |
| Modifying UI requires recompilation of servlet. | JSP changes are automatically recompiled by the server. |
| Follows Servlet API strictly. | Internally converted to Servlet by the server (JSP → Servlet → Class). |
| Better for business logic and request handling. | Better for UI development and template-based web pages. |

Advantages of JSP Over Other Server-Side Scripting Languages (Like PHP, ASP, Perl)

1. Pure Java Technology

- JSP is built on Java, so it uses Java's platform independence, security, and robustness.
- Code can run on any OS with a Java-enabled web server.

2. Separation of Business Logic and Presentation

- JSP allows mixing HTML + Java using JSTL, EL, and custom tags.
- Supports MVC architecture where JSP = View, Servlet = Controller.

3. Automatic Servlet Generation

- JSP pages are automatically converted to servlets by the server.
- Reduces developer effort and improves maintainability.

4. Built-in Custom Tag Libraries (JSTL)

- Provides <c:if>, <c:forEach>, and other powerful tags.
- Reduces Java scriptlet code inside HTML.

5. Faster Development

- No need to recompile after every change.
- Just save the JSP file, and server automatically recompiles it.

6. Reusable Components

- JSP supports Custom Tags, JSTL, JavaBeans, and EL.
- Encourages modularity and reusability.

7. Strong Integration with Java Technologies

- Can use JDBC, RMI, EJB, Spring, Hibernate, and many Java APIs.
- Makes JSP suitable for large enterprise applications.

8. High Performance

- Compiled into Servlets → Executes faster than interpreted scripts like PHP or Perl.

9. Built-in Exception Handling

- JSP provides <error-page> mechanism.
- More powerful than typical scripting language error handling.

10. Scalability and Security

- Java's memory management and security model make JSP suitable for big enterprise applications.
- Supports HTTPS, authentication, authorization, filters, etc.

12.Create an application where an HTML file displays a form containing field like company name, city and established date and a save button and when we click on save button it must save records in database.

Step 1: Create Database Table

Use MySQL:

```
CREATE DATABASE companydb;
```

```
USE companydb;
```

```
CREATE TABLE companies (
    id INT AUTO_INCREMENT PRIMARY KEY,
    company_name VARCHAR(100),
    city VARCHAR(100),
    established_date DATE
);
```

Step 2: HTML Form (company_form.html)

```
<!DOCTYPE html>
```

```
<html>
```

```

<head>
    <title>Company Registration</title>
</head>
<body>

<h2>Company Information Form</h2>

<form action="SaveCompany" method="post">
    Company Name: <input type="text" name="company_name" required><br><br>
    City: <input type="text" name="city" required><br><br>
    Established Date: <input type="date" name="established_date" required><br><br>

    <input type="submit" value="Save">
</form>

```

</body>
</html>

Step 3: Servlet Code (SaveCompany.java)

```

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

public class SaveCompany extends HttpServlet {

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

        response.setContentType("text/html");
        PrintWriter out = response.getWriter();

        String cname = request.getParameter("company_name");
        String city = request.getParameter("city");
        String estDate = request.getParameter("established_date");

        Connection con = null;
        PreparedStatement pst = null;

        try {
            Class.forName("com.mysql.cj.jdbc.Driver");
            con = DriverManager.getConnection(
                "jdbc:mysql://localhost:3306/companydb", "root", ""
            );

```

```

String sql = "INSERT INTO companies (company_name, city, established_date)
VALUES (?, ?, ?)";
pst = con.prepareStatement(sql);

pst.setString(1, cname);
pst.setString(2, city);
pst.setString(3, estDate);

int status = pst.executeUpdate();

if (status > 0) {
    out.println("<h2>Record Saved Successfully!</h2>");
} else {
    out.println("<h2>Error Saving Record!</h2>");
}

} catch (Exception e) {
    out.println("Error: " + e);
}
}
}
}

```

Step 4: web.xml Configuration

Add inside WEB-INF/web.xml:

```

<servlet>
    <servlet-name>SaveCompany</servlet-name>
    <servlet-class>SaveCompany</servlet-class>
</servlet>

<servlet-mapping>
    <servlet-name>SaveCompany</servlet-name>
    <url-pattern>/SaveCompany</url-pattern>
</servlet-mapping>

```

13.Explain the steps involved in creating RMI applications with suitable Java code.

Steps in Creating an RMI Application

Step 1: Create Remote Interface

- Must extend java.rmi.Remote
- Each remote method must declare throws RemoteException

Step 2: Create Remote Implementation Class

- Must extend UnicastRemoteObject
- Must implement the remote interface
- Provide method definitions

Step 3: Create Server Program

- Create instance of the implementation class
- Register object in RMI Registry

Step 4: Start RMI Registry

- Use command:
- rmiregistry

Step 5: Create Client Program

- Look up remote object from RMI registry
- Call remote methods

Step 6: Compile All Files

```
javac *.java
```

Step 7: Run Server then Client

Complete RMI Example: Program to Add Two Numbers

1. Remote Interface (Add.java)

```
import java.rmi.*;
```

```
public interface Add extends Remote {  
    public int addNumbers(int a, int b) throws RemoteException;  
}
```

2. Implementation Class (AddImpl.java)

```
import java.rmi.*;  
import java.rmi.server.*;
```

```
public class AddImpl extends UnicastRemoteObject implements Add {
```

```
    public AddImpl() throws RemoteException {  
        super();  
    }
```

```
    public int addNumbers(int a, int b) throws RemoteException {  
        return a + b;  
    }  
}
```

3. Server Program (AddServer.java)

```
import java.rmi.*;
```

```
public class AddServer {  
    public static void main(String[] args) {  
        try {  
            AddImpl obj = new AddImpl();  
            Naming.rebind("AddService", obj);  
            System.out.println("Server Ready...");  
        } catch (Exception e) {
```

```

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

4. Client Program (AddClient.java)
import java.rmi.*;

public class AddClient {
    public static void main(String[] args) {
        try {
            Add obj = (Add) Naming.lookup("rmi://localhost/AddService");
            int result = obj.addNumbers(10, 20);
            System.out.println("Sum = " + result);
        }
        catch(Exception e) {
            System.out.println(e);
        }
    }
}

```

14.What is toolbar? Explain the way of creating toolbar using Swing with suitable example.

A toolbar in Java Swing is a component that contains a row (or column) of buttons or controls that provide quick access to commonly used actions (like New, Open, Save, Cut, Copy, Paste, etc.).

Swing provides a toolbar using the class:

`javax.swing.JToolBar`

A toolbar can contain:

- Buttons (JButton)
- Separators
- Combo boxes
- Labels or any Swing component

It is usually placed at the top of a frame.

How to Create a Toolbar in Swing

Steps:

1. Create a JToolBar object
2. Create buttons (or other components)
3. Add buttons to the toolbar
4. Add the toolbar to the frame (usually using BorderLayout.NORTH)

Complete Example Program (Swing Toolbar Example)

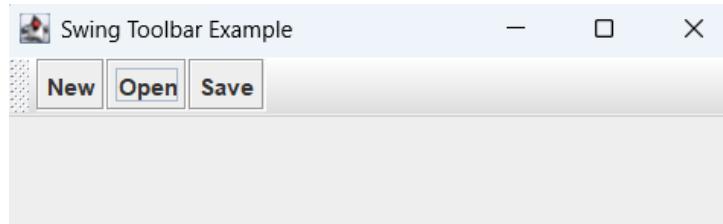
```

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

```

```
public class ToolbarExample extends JFrame {  
  
    public ToolbarExample() {  
  
        // Create toolbar  
        JToolBar toolbar = new JToolBar();  
  
        // Create buttons  
        JButton btnNew = new JButton("New");  
        JButton btnOpen = new JButton("Open");  
        JButton btnSave = new JButton("Save");  
  
        // Add buttons to toolbar  
        toolbar.add(btnNew);  
        toolbar.add(btnOpen);  
        toolbar.add(btnSave);  
  
        // Simple actions for buttons  
        btnNew.addActionListener(e -> JOptionPane.showMessageDialog(this, "New File  
Created"));  
        btnOpen.addActionListener(e -> JOptionPane.showMessageDialog(this, "Open File  
Dialog"));  
        btnSave.addActionListener(e -> JOptionPane.showMessageDialog(this, "File Saved"));  
  
        // Add toolbar to frame  
        add(toolbar, BorderLayout.NORTH);  
  
        // Frame settings  
        setTitle("Swing Toolbar Example");  
        setSize(400, 300);  
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);  
        setVisible(true);  
    }  
  
    public static void main(String[] args) {  
        new ToolbarExample();  
    }  
}
```

Output:



15. Define cookie. How can you store and read cookies by using servlet? Explain with example.

A cookie is a small piece of information that a server stores on the client's browser.

Cookies help identify users across multiple requests.

Examples:

- Remembering username
- Tracking user sessions
- Storing preferences

Cookies are sent by server → stored by browser → returned back to server on every request.

Servlets use the class:

`javax.servlet.http.Cookie`

How to Create (Store) Cookies in Servlet

Steps:

1. Create a cookie object
2. Set its value & optional expiry time
3. Add cookie to the response

`Cookie c = new Cookie("username", "Bibash");`

`response.addCookie(c);`

How to Read Cookies in Servlet

Steps:

1. Use `request.getCookies()` to read all cookies
2. Loop through them
3. Match cookie name and get the value

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

Complete Example: Store and Read Cookies in Servlet

A. Servlet to Store Cookie (`SetCookieServlet.java`)

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

```
public class SetCookieServlet extends HttpServlet {
```

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

```
        response.setContentType("text/html");
```

```

PrintWriter out = response.getWriter();

// Create cookie
Cookie cookie = new Cookie("username", "Bibash");
cookie.setMaxAge(60 * 60); // 1 hour

// Add cookie to response
response.addCookie(cookie);

out.println("<h3>Cookie Saved Successfully!</h3>");
}
}

```

B. Servlet to Read Cookie (ReadCookieServlet.java)

```

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

public class ReadCookieServlet extends HttpServlet {

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

response.setContentType("text/html");
PrintWriter out = response.getWriter();

Cookie[] cookies = request.getCookies();
String username = "";

if (cookies != null) {
for (Cookie c : cookies) {
if (c.getName().equals("username")) {
username = c.getValue();
}
}
}

if (!username.equals("")) {
out.println("<h3>Welcome Back, " + username + "!</h3>");
} else {
out.println("<h3>No cookie found!</h3>");
}
}
}
```

web.xml Mapping

```
<servlet>
  < servlet-name>SetCookie</servlet-name>
  < servlet-class>SetCookieServlet</servlet-class>
</servlet>

<servlet-mapping>
  < servlet-name>SetCookie</servlet-name>
  < url-pattern>/setcookie</url-pattern>
</servlet-mapping>

<servlet>
  < servlet-name>ReadCookie</servlet-name>
  < servlet-class>ReadCookieServlet</servlet-class>
</servlet>

<servlet-mapping>
  < servlet-name>ReadCookie</servlet-name>
  < url-pattern>/readcookie</url-pattern>
</servlet-mapping>
```

16.Discuss RMI architecture with suitable diagram and discuss each layer in the architecture briefly.

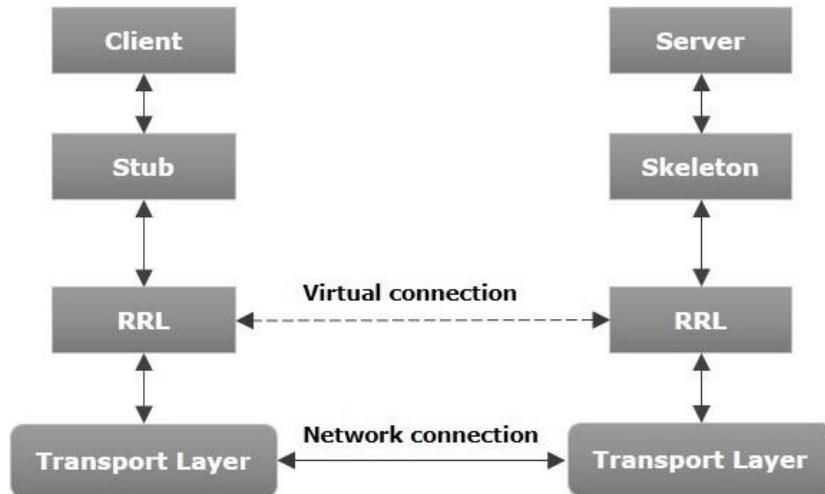
RMI Architecture

RMI (Remote Method Invocation) allows a Java program on one JVM to invoke methods on objects running on another JVM, possibly on a remote machine.

In an RMI application, we write two programs, a server program (resides on the server) and a client program (resides on the client).

- Inside the server program, a remote object is created and reference of that object is made available for the client (using the registry).
- The client program requests the remote objects on the server and tries to invoke its methods.

RMI Architecture Diagram



Let us now discuss the components of this architecture.

- Transport Layer – This layer connects the client and the server. It manages the existing connection and also sets up new connections.
- Stub – A stub is a representation (proxy) of the remote object at client. It resides in the client system; it acts as a gateway for the client program.
- Skeleton – This is the object which resides on the server side. stub communicates with this skeleton to pass request to the remote object.
- RRL(Remote Reference Layer) – It is the layer which manages the references made by the client to the remote object.

Flow of a Remote Method Call

1. Client calls method on stub.
2. Stub marshals parameters → sends request over network.
3. Transport layer delivers request to server.
4. Skeleton/unmarshalling layer receives request → invokes actual remote object.
5. Remote object executes method → returns result.
6. Skeleton marshals response → sent back via transport.
7. Client stub receives → unmarshals → returns result to client program.

17. How can you create menus by using Swing? Explain different classes used in designing menus with example.

In Java Swing, menus allow the user to perform actions by selecting items from a list. Menus are created using the Swing menu classes.

Classes Used in Creating Menus

| Class | Description |
|----------------------|---|
| JMenuBar | Represents the menu bar attached to the frame (usually at the top). |
| JMenu | Represents a menu (e.g., File, Edit, View). Can contain JMenuItem objects. |
| JMenuItem | Represents a menu item inside a JMenu. Clicking performs an action. |
| JCheckBoxMenuItem | A menu item with a checkbox (used for options that can be selected/deselected). |
| JRadioButtonMenuItem | A menu item with radio button (used for mutually exclusive choices). |
| JSeparator | Adds a separator line between menu items. |

Steps to Create Menu in Swing

1. Create a JMenuBar object.
2. Create JMenu objects and add them to the JMenuBar.
3. Create JMenuItem objects and add them to the JMenu.
4. Add ActionListener to menu items for functionality.
5. Attach JMenuBar to the JFrame using setJMenuBar().

Complete Example Program

```

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

public class MenuExample1 extends JFrame {

    public MenuExample1() {

        // Create MenuBar
        JMenuBar menuBar = new JMenuBar();

        // Create Menus
        JMenu fileMenu = new JMenu("File");
        JMenu editMenu = new JMenu("Edit");

        // Create Menu Items for File Menu
        JMenuItem newItem = new JMenuItem("New");
        JMenuItem openItem = new JMenuItem("Open");
        JMenuItem exitItem = new JMenuItem("Exit");

        // Add menu items to File menu
        fileMenu.add(newItem);
        fileMenu.add(openItem);
    }
}
```

```

fileMenu.addSeparator(); // adds a line separator
fileMenu.add(exitItem);

// Create Menu Items for Edit Menu
JMenuItem cutItem = new JMenuItem("Cut");
JMenuItem copyItem = new JMenuItem("Copy");
JMenuItem pasteItem = new JMenuItem("Paste");

editMenu.add(cutItem);
editMenu.add(copyItem);
editMenu.add(pasteItem);

// Add menus to MenuBar
menuBar.add(fileMenu);
menuBar.add(editMenu);

// Set MenuBar to JFrame
setJMenuBar(menuBar);

// Action Listeners
newItem.addActionListener(e -> JOptionPane.showMessageDialog(this, "New File
Created"));
openItem.addActionListener(e -> JOptionPane.showMessageDialog(this, "Open File
Dialog"));
exitItem.addActionListener(e -> System.exit(0));

cutItem.addActionListener(e -> JOptionPane.showMessageDialog(this, "Cut Action"));
copyItem.addActionListener(e -> JOptionPane.showMessageDialog(this, "Copy
Action"));
pasteItem.addActionListener(e -> JOptionPane.showMessageDialog(this, "Paste
Action"));

// Frame settings
setTitle("Swing Menu Example");
setSize(400, 300);
setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
setVisible(true);
}

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

```

Explanation of Code

1. JMenuBar

```
JMenuBar menuBar = new JMenuBar();
```

- Container for menus (File, Edit)

2. JMenu

```
JMenu fileMenu = new JMenu("File");
```

- Represents a menu on the menu bar
- 3. JMenuItem

```
JMenuItem newItem = new JMenuItem("New");
```

- Represents an item in the menu
- 4. JSeparator

```
fileMenu.addSeparator();
```

- Adds a horizontal line between menu items
- 5. ActionListener
- Menu items perform actions when clicked

18.What is session? What are different ways of tracking sessions? Write down suitable servlet for tracking session.

A session represents a series of requests from the same user (client) during a time period in a web application.

HTTP is a stateless protocol, so sessions are used to maintain user-specific information across multiple requests.

Examples of session usage:

- Login information (username, roles)
- Shopping cart in e-commerce
- Preferences or settings

Different Ways of Tracking Sessions in Servlets

| Method | Description |
|-----------------------|--|
| 1. Cookies | Server stores a unique session ID in a cookie on the client. Client sends the cookie in each request. |
| 2. URL Rewriting | Append session ID in the URL (e.g., page.jsp?jsessionid=12345). Useful if cookies are disabled. |
| 3. Hidden Form Fields | Session ID or user data is stored in hidden fields of forms. Useful for multi-page forms without cookies. |
| 4. HttpSession API | Servlet provides HttpSession object to store and retrieve user data easily. Most common and reliable method. |

Tracking Session Using HttpSession

HttpSession allows you to create a session, store attributes, and retrieve them in subsequent requests.

Servlet Example: Tracking Session

Servlet Name: SessionExample.java

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

```
import javax.servlet.*;
```

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

```
public class SessionExample extends HttpServlet {
```

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

```
        response.setContentType("text/html");
```

```
        PrintWriter out = response.getWriter();
```

```
        // Create or get existing session
```

```
        HttpSession session = request.getSession(true); // true = create if doesn't exist
```

```
        // Set an attribute in session
```

```
        session.setAttribute("username", "Bibash");
```

```
        // Get the attribute from session
```

```
        String name = (String) session.getAttribute("username");
```

```
        // Display session info
```

```
        out.println("<h2>Welcome, " + name + "!"</h2>");
```

```
        out.println("<p>Session ID: " + session.getId() + "</p>");
```

```
        out.println("<p>Creation Time: " + session.getCreationTime() + "</p>");
```

```
        out.println("<p>Last Accessed Time: " + session.getLastAccessedTime() + "</p>");
```

```
}
```

```
}
```

web.xml Mapping

```
<servlet>
```

```
    <servlet-name>SessionExample</servlet-name>
```

```
    <servlet-class>SessionExample</servlet-class>
```

```
</servlet>
```

```
<servlet-mapping>
```

```
    <servlet-name>SessionExample</servlet-name>
```

```
    <url-pattern>/session</url-pattern>
```

```
</servlet-mapping>
```

19.What is RMI? Discuss stub and skeleton. Explain its role in creating distributed applications.

RMI (Remote Method Invocation) is a Java API that allows an object running in one Java Virtual Machine (JVM) to invoke methods on an object running in another JVM, possibly on a remote machine.

- It enables distributed applications in Java.
- Provides transparency, meaning the client can call a remote object just like a local object.

Key Features of RMI:

- Object-oriented RPC (Remote Procedure Call)
- Passes objects as arguments and return values
- Automatic serialization of objects
- Works only with Java objects

Stub and Skeleton in RMI

RMI uses stub and skeleton to provide communication between client and server.

| Component | Role |
|---|---|
| Stub (Client-side Proxy) | Acts as a proxy for the remote object on the client. Handles: <ol style="list-style-type: none">1. Marshalling: Converts method arguments into a byte stream.2. Sending requests to the server.3. Receiving responses and unmarshalling results back to client. |
| Skeleton (Server-side Proxy / Dispatcher) | Exists on the server (in older Java versions) and handles: <ol style="list-style-type: none">1. Receiving requests from stub.2. Unmarshalling parameters.3. Invoking the actual method on the remote object.4. Marshalling return values and sending back to stub. Note: Skeleton is optional in Java 2 onwards (Java automatically handles it). |

Role of Stub and Skeleton in Distributed Applications

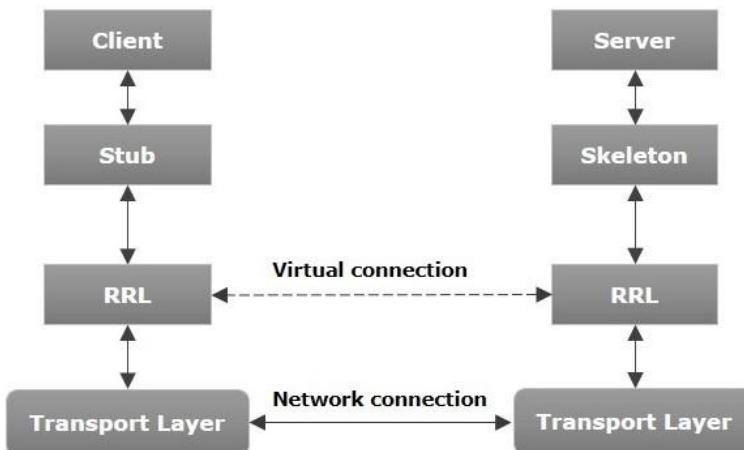
1. Client Transparency
 - Client calls methods on stub as if calling a local object.
 - Hides network details, IP addresses, and TCP communication.
2. Marshalling / Unmarshalling
 - Stub converts arguments into a byte stream (marshalling).
 - Skeleton converts bytes back into objects (unmarshalling) and vice versa for return values.
3. Communication Layer

- Stub and skeleton handle network communication between client and server.
- Developers only write the business logic; RMI handles the remote invocation.

4. Distributed Object Support

- Multiple clients can call the same remote object concurrently.
- Stub and skeleton coordinate requests to the correct server object.

RMI Architecture Diagram (Simplified)



20.What are different types of borders used in Swing? Explain all the borders with suitable example.

Different Types of Borders in Swing

In Java Swing, borders are used to decorate components like panels, buttons, text fields, etc. Swing provides the javax.swing.border.Border interface and the BorderFactory class to create various borders.

1. Line Border

- Description: Draws a simple line border around the component.
- Usage: Highlight a component with a colored line.

```
JPanel panel = new JPanel();
```

```
panel.setBorder(BorderFactory.createLineBorder(Color.BLUE, 3));
```

Parameters:

- Color of border
- Thickness of border

2. Etched Border

- Description: Creates a 3D etched look (lowered or raised effect).
- Usage: Gives a sunken or raised effect to components.

```
JPanel panel = new JPanel();
```

```
panel.setBorder(BorderFactory.createEtchedBorder(EtchedBorder.RAISED, Color.GRAY, Color.WHITE));
```

Parameters:

- EtchedBorder.RAISED or EtchedBorder.LOWERED
- Highlight color, shadow color

3. Bevel Border

- Description: Creates a beveled edge around a component (raised or lowered).

- Usage: Gives a 3D appearance.

```
 JPanel panel = new JPanel();
panel.setBorder(BorderFactory.createBevelBorder(BevelBorder.RAISED, Color.WHITE,
Color.GRAY));
```

Parameters:

- BevelBorder.RAISED or BevelBorder.LOWERED
- Highlight and shadow colors

4. Titled Border

- Description: Displays a title text around the border of a component.
- Usage: Often used for grouping panels with labels.

```
 JPanel panel = new JPanel();
panel.setBorder(BorderFactory.createTitledBorder("User Info"));
```

Advanced: You can combine with etched or line border:

```
 panel.setBorder(BorderFactory.createTitledBorder(
    BorderFactory.createLineBorder(Color.BLUE), "User Info"));
```

5. Empty Border

- Description: Provides empty space padding around a component.
- Usage: For spacing and layout management.

```
 JPanel panel = new JPanel();
panel.setBorder(BorderFactory.createEmptyBorder(10, 10, 10, 10)); // top, left, bottom, right
padding
```

6. Compound Border

- Description: Combines two borders into one.
- Usage: Useful for nested effects.

```
 JPanel panel = new JPanel();
Border outer = BorderFactory.createLineBorder(Color.RED, 2);
Border inner = BorderFactory.createEmptyBorder(5, 5, 5, 5);
panel.setBorder(BorderFactory.createCompoundBorder(outer, inner));
```

7. Matte Border

- Description: Allows a border with a solid color or image.
- Usage: Customizing component borders with colors or images.

```
 JPanel panel = new JPanel();
panel.setBorder(BorderFactory.createMatteBorder(5, 5, 5, 5, Color.GREEN)); // top, left,
bottom, rig
```

Example Program Using Different Borders

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

public class BorderExample extends JFrame {
    public BorderExample() {
        setTitle("Swing Borders Example");
```

```
setLayout(new GridLayout(3, 2, 10, 10));

JPanel panel1 = new JPanel();
panel1.setBorder(BorderFactory.createLineBorder(Color.BLUE, 3));
panel1.add(new JLabel("Line Border"));

JPanel panel2 = new JPanel();
panel2.setBorder(BorderFactory.createEtchedBorder(EtchedBorder.RAISED));
panel2.add(new JLabel("Etched Border"));

JPanel panel3 = new JPanel();
panel3.setBorder(BorderFactory.createBevelBorder(BevelBorder.RAISED));
panel3.add(new JLabel("Bevel Border"));

JPanel panel4 = new JPanel();
panel4.setBorder(BorderFactory.createTitledBorder("Title Border"));
panel4.add(new JLabel("Titled Border"));

JPanel panel5 = new JPanel();
panel5.setBorder(BorderFactory.createEmptyBorder(10,10,10,10));
panel5.add(new JLabel("Empty Border"));

JPanel panel6 = new JPanel();
Border outer = BorderFactory.createLineBorder(Color.RED, 2);
Border inner = BorderFactory.createEmptyBorder(5,5,5,5);
panel6.setBorder(BorderFactory.createCompoundBorder(outer, inner));
panel6.add(new JLabel("Compound Border"));

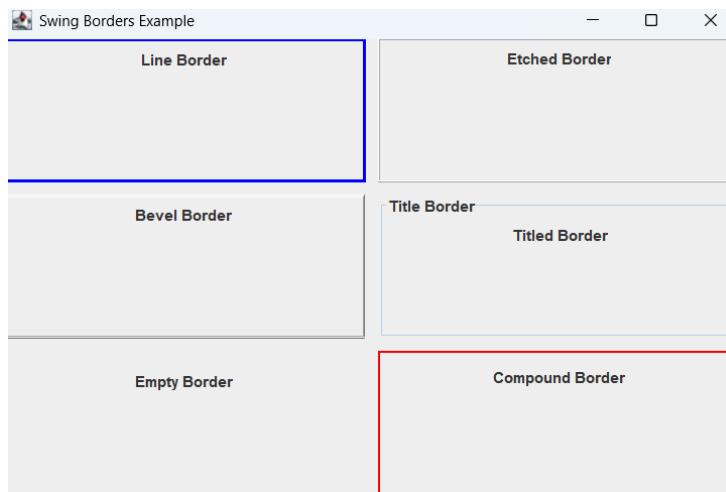
add(panel1);
add(panel2);
add(panel3);
add(panel4);
add(panel5);
add(panel6);

setSize(600, 400);
setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
setVisible(true);
}

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

}

Output:



21. How exceptions can be handled in JSP scripts? Explain with suitable JSP script.

Using try-catch Block inside JSP Scriptlets

You can handle exceptions in a scriptlet using standard Java try-catch:

```
<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>
<html>
<head><title>Exception Handling in JSP</title></head>
<body>
<%
try {
    int num1 = 10;
    int num2 = 0; // This will cause ArithmeticException
    int result = num1 / num2;
    out.println("Result: " + result);
} catch (ArithmaticException e) {
    out.println("Error: Division by zero is not allowed!");
}
%>
</body>
</html>
```

Explanation:

- Java exceptions like ArithmaticException are caught and handled inside JSP.
- Prevents the page from crashing.

Using errorPage and isErrorPage Directives

- JSP allows you to forward exceptions to a dedicated error page.

Step 1: Main JSP Page (divide.jsp)

```
<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8" errorPage="error.jsp"%>
```

```

<html>
<body>
<%
    int num1 = 10;
    int num2 = 0; // This will cause exception
    int result = num1 / num2;
    out.println("Result: " + result);
%
</body>
</html>

```

Step 2: Error Page (error.jsp)

```

<%@ page isErrorPage="true" %>
<html>
<body>
<h2>An error occurred in JSP page!</h2>
<p>Error Details: <%= exception %></p>
</body>
</html>

```

Explanation:

- `errorPage="error.jsp"` → Specifies the page to handle exceptions.
- `isErrorPage="true"` → Allows access to the implicit exception object in error.jsp.
- Cleaner approach for global error handling.

Using Implicit Object exception

- JSP provides an implicit object exception that is available only in error pages.
- It contains the Exception object thrown in the JSP page.

22.What is Java Bean? How can it be used in JSP scripts? Explain with suitable example.

JavaBean is a reusable software component written in Java that follows certain conventions and can be manipulated visually in a builder tool.

Characteristics of a JavaBean

1. Public default constructor – must have a no-argument constructor.
2. Private properties – use getter and setter methods to access.
3. Serializable – implements `java.io.Serializable` for persistence.
4. Reusability – can be used across multiple applications.
5. Encapsulation – data is hidden and accessed via methods.

Using JavaBeans in JSP

JSP provides standard actions to use JavaBeans:

1. `<jsp:useBean>` – Creates an instance of the bean.
2. `<jsp:setProperty>` – Sets property values of the bean.

3. <jsp:getProperty> – Retrieves property values from the bean.

Step 1: Create JavaBean Class

File: Student.java

```
package beans;
```

```
import java.io.Serializable;
```

```
public class Student implements Serializable {
```

```
    private String name;
```

```
    private int age;
```

```
    // Default constructor
```

```
    public Student() {}
```

```
    // Getter and Setter methods
```

```
    public String getName() {
```

```
        return name;
```

```
}
```

```
    public void setName(String name) {
```

```
        this.name = name;
```

```
}
```

```
    public int getAge() {
```

```
        return age;
```

```
}
```

```
    public void setAge(int age) {
```

```
        this.age = age;
```

```
}
```

Step 2: Use Bean in JSP Page

File: student.jsp

```
<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>
```

```
<html>
```

```
<head><title>JavaBean Example</title></head>
```

```
<body>
```

```
<jsp:useBean id="stu" class="beans.Student" scope="session" />
```

```
<jsp:setProperty name="stu" property="name" value="Bibash" />
```

```
<jsp:setProperty name="stu" property="age" value="22" />
```

```

<h2>Student Details Using JavaBean</h2>
<p>Name: <jsp:getProperty name="stu" property="name" /></p>
<p>Age: <jsp:getProperty name="stu" property="age" /></p>

</body>
</html>

```

Explanation of JSP Code

1. <jsp:useBean id="stu" class="beans.Student" scope="session" />
 - o Creates an instance of Student bean with session scope.
2. <jsp:setProperty name="stu" property="name" value="Bibash" />
 - o Sets the name property of the bean.
3. <jsp:getProperty name="stu" property="name" />
 - o Retrieves the name property and displays it on the JSP page.
4. scope can be: page, request, session, or application.

Advantages of Using JavaBeans in JSP

- Promotes reusability of components.
- Encapsulates data and reduces scriptlet code.
- Provides clean and readable JSP pages.
- Can be used with MVC architecture.

23. How can JSP be used for enabling and disabling session? Explain with suitable JSP script.

In JSP, sessions are enabled by default. Each JSP page maintains an implicit session object of type HttpSession.

Enabling Session in JSP

- By default, session is enabled (session="true").
- You can explicitly enable it using the page directive:

```

<%@ page session="true" %>
    • You can use the implicit session object to store and retrieve attributes:
<%>
    // Storing data in session
    session.setAttribute("username", "Bibash");

    // Retrieving data from session
    String user = (String) session.getAttribute("username");
    out.println("Hello, " + user);
%>

```

Disabling Session in JSP

- Sometimes, you may want to disable sessions to improve performance or avoid creating unnecessary session objects.
- Use page directive:

```
<%@ page session="false" %>
```

- Important: If session is disabled, the implicit session object is not available. Attempting to use it will cause an error.

Example JSP Script: Enabling and Disabling Session

A. Enabling Session (enableSession.jsp)

```
<%@ page session="true" %>
<html>
<head><title>Enable Session Example</title></head>
<body>
<%
    // Check if session is new
    if (session.isNew()) {
        out.println("<p>New session created.</p>");
    } else {
        out.println("<p>Existing session found.</p>");
    }

    // Store a session attribute
    session.setAttribute("username", "Bibash");

    // Retrieve session attribute
    String name = (String) session.getAttribute("username");
    out.println("<p>Username from session: " + name + "</p>");
%>
</body>
</html>
```

B. Disabling Session (disableSession.jsp)

```
<%@ page session="false" %>
<html>
<head><title>Disable Session Example</title></head>
<body>
<%
    out.println("<p>Session is disabled for this page.</p>");
    // Trying to access session object will throw error
    // String name = (String) session.getAttribute("username"); // Not allowed
%>
</body>
</html>
```

24. Write a program to display a login form containing user id, password, account type and a "OK" button. Account type must be admin or user. Use layout managers properly.

Java Swing Login Form Example

This program creates a login form with User ID, Password, Account Type (Admin/User), and OK button using layout managers.

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

public class LoginForm extends JFrame implements ActionListener {

    // Components
    private JLabel lblUserId, lblPassword, lblAccountType;
    private JTextField txtUserId;
    private JPasswordField txtPassword;
    private JComboBox<String> cmbAccountType;
    private JButton btnOk;

    public LoginForm() {
        // Frame title
        setTitle("Login Form");

        // Set layout manager
        setLayout(new BorderLayout());

        // Create Panels
        JPanel panelFields = new JPanel(new GridLayout(3, 2, 10, 10)); // 3 rows, 2 columns
        JPanel panelButton = new JPanel(); // For button

        // Initialize Components
        lblUserId = new JLabel("User ID:");
        lblPassword = new JLabel("Password:");
        lblAccountType = new JLabel("Account Type:");

        txtUserId = new JTextField();
        txtPassword = new JPasswordField();

        String[] accountTypes = {"Admin", "User"};
        cmbAccountType = new JComboBox<>(accountTypes);

        btnOk = new JButton("OK");
    }
}
```

```

btnOk.addActionListener(this);

// Add components to panelFields
panelFields.add(lblUserId);
panelFields.add(txtUserId);
panelFields.add(lblPassword);
panelFields.add(txtPassword);
panelFields.add(lblAccountType);
panelFields.add(cmbAccountType);

// Add button to panelButton
panelButton.add(btnOk);

// Add panels to frame
add(panelFields, BorderLayout.CENTER);
add(panelButton, BorderLayout.SOUTH);

// Frame settings
setSize(400, 200);
setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
 setLocationRelativeTo(null); // Center the frame
setVisible(true);
}

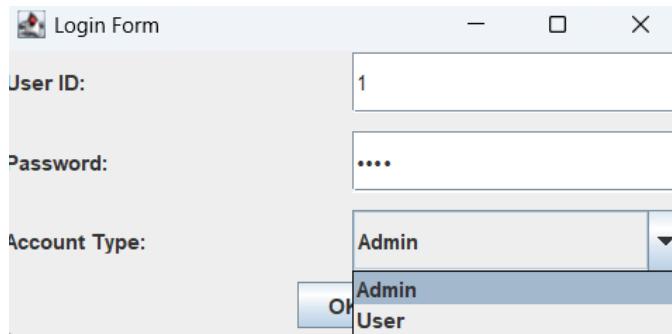
// Action performed when OK button is clicked
@Override
public void actionPerformed(ActionEvent e) {
    String userId = txtUserId.getText();
    String password = new String(txtPassword.getPassword());
    String accountType = (String) cmbAccountType.getSelectedItem();

    JOptionPane.showMessageDialog(this,
        "User ID: " + userId + "\nPassword: " + password + "\nAccount Type: " +
        accountType,
        "Login Details",
        JOptionPane.INFORMATION_MESSAGE);
}

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

```

Output:



25. Write a JSP script that demonstrates use of select and insert operations with suitable example.

JSP Script for SELECT and INSERT

Assumptions

- Database: companydb
- Table: employees

CREATE TABLE employees (

id INT AUTO_INCREMENT PRIMARY KEY,
name VARCHAR(50),
email VARCHAR(50),
position VARCHAR(50)

);

- JDBC Driver: com.mysql.cj.jdbc.Driver
- Database URL: jdbc:mysql://localhost:3306/companydb
- User: root
- Password: password

JSP Script: employees.jsp

```
<%@ page import="java.sql.*" %>
<%@ page contentType="text/html;charset=UTF-8" language="java" %>
<html>
<head>
    <title>Employee Management</title>
</head>
<body>
    <h2>Employee Management</h2>

    <%
        // JDBC parameters
        String url = "jdbc:mysql://localhost:3306/companydb";
        String user = "root";
        String pass = "password";

        Connection con = null;
```

```

Statement stmt = null;
ResultSet rs = null;

try {
    Class.forName("com.mysql.cj.jdbc.Driver");
    con = DriverManager.getConnection(url, user, pass);
    stmt = con.createStatement();

    // Insert operation
    if (request.getParameter("submit") != null) {
        String name = request.getParameter("name");
        String email = request.getParameter("email");
        String position = request.getParameter("position");

        String insertQuery = "INSERT INTO employees(name, email, position) VALUES("
            + name + "','" + email + "','" + position + ")";
        int i = stmt.executeUpdate(insertQuery);

        if(i > 0){
            out.println("<p style='color:green;'>Employee added successfully!</p>");
        } else {
            out.println("<p style='color:red;'>Error adding employee.</p>");
        }
    }

    // Select operation
    String selectQuery = "SELECT * FROM employees";
    rs = stmt.executeQuery(selectQuery);

%>

<!-- Form for adding employee -->
<h3>Add Employee</h3>
<form method="post" action="employees.jsp">
    Name: <input type="text" name="name" required><br><br>
    Email: <input type="email" name="email" required><br><br>
    Position: <input type="text" name="position" required><br><br>
    <input type="submit" name="submit" value="Add Employee">
</form>

<!-- Display all employees -->
<h3>Employee List</h3>
<table border="1" cellpadding="5" cellspacing="0">

```

```

<tr>
    <th>ID</th>
    <th>Name</th>
    <th>Email</th>
    <th>Position</th>
</tr>
<%
    while(rs.next()) {
%>
<tr>
    <td><%= rs.getInt("id") %></td>
    <td><%= rs.getString("name") %></td>
    <td><%= rs.getString("email") %></td>
    <td><%= rs.getString("position") %></td>
</tr>
<%
}
}

} catch(Exception e){
    out.println("<p style='color:red;'>Error: " + e.getMessage() + "</p>");
} finally {
    try { if(rs != null) rs.close(); } catch(Exception e) {}
    try { if(stmt != null) stmt.close(); } catch(Exception e) {}
    try { if(con != null) con.close(); } catch(Exception e) {}
}
%>
</table>

</body>
</html>

```

Explanation

1. Database Connection

```

Class.forName("com.mysql.cj.jdbc.Driver");
Connection con = DriverManager.getConnection(url, user, pass);

```

- Connects to MySQL database.

2. Insert Operation

```

String insertQuery = "INSERT INTO employees(name,email,position) VALUES(\""
    + name + "\",\"" + email + "\",\"" + position + "\")";
stmt.executeUpdate(insertQuery);

```

- Inserts data from the HTML form into the database.

3. Select Operation

```

ResultSet rs = stmt.executeQuery("SELECT * FROM employees");

```

- Fetches all employee records and displays them in a table.
4. Form
 - Allows user to enter name, email, and position.
 - Submitting the form triggers insert operation.
 5. Displaying Records
 - Uses while(rs.next()) to iterate and show employee details in HTML table.

26.What are two key features of Swing? Explain how GUI can be developed by using JFrame.

Two Key Features of Swing

Swing is a part of Java Foundation Classes (JFC) used for building GUI applications. It provides a richer set of components than AWT.

Key Features of Swing

1. Lightweight Components
 - Swing components are pure Java (lightweight), unlike AWT components which rely on native system GUI components (heavyweight).
 - Examples: JButton, JTextField, JLabel, JPanel.
 - Advantage: Consistent look and behavior across platforms.
2. Pluggable Look and Feel (PLAF)
 - Swing supports customizable look and feel, allowing the application to appear:
 - Metal (default Java style)
 - System native style
 - Custom themes
 - Example:


```
UIManager.setLookAndFeel("javax.swing.plaf.nimbus.NimbusLookAndFeel");
});
```

Developing GUI Using JFrame

JFrame is the top-level container in Swing. It represents the main window of a GUI application.

Steps to Create GUI Using JFrame

1. Import Required Packages

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

2. Create a Class Extending JFrame

```
public class MyGUI extends JFrame {
    public MyGUI() {
        setTitle("My First GUI");
        setSize(400, 300);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
       .setLayout(new FlowLayout()); // Layout manager
    }
}
```

3. Add Components

```
JLabel lbl = new JLabel("Enter Name:");
```

```

JTextField txt = new JTextField(15);
JButton btn = new JButton("Submit");

add(lbl);
add(txt);
add(btn);

4. Add Event Handling (Optional)
btn.addActionListener(e -> {
    JOptionPane.showMessageDialog(this, "Hello, " + txt.getText());
});

5. Set Frame Visible
setVisible(true);
}

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

```

27.Explain CORBA architecture briefly. What is IDL? Explain its structure briefly.

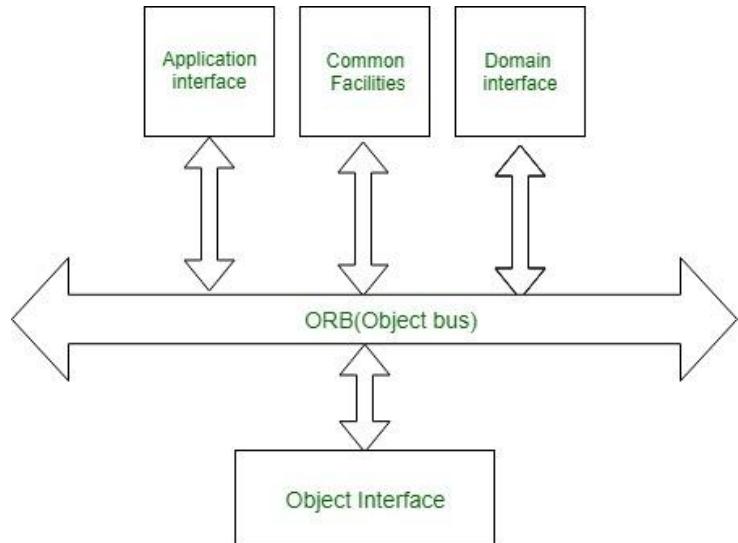
CORBA Architecture and IDL

CORBA (Common Object Request Broker Architecture) is a standard defined by the Object Management Group (OMG) for enabling communication between distributed objects across different programming languages, platforms, and networks.

CORBA Architecture

CORBA architecture consists of several layers and components that allow clients and servers to communicate seamlessly.

CORBA Architecture Diagram (Simplified)



Object Management Architecture(OMA)

The CORBA reference model known as Object Management design (OMA) is shown below figure. The OMA is itself a specification (actually, a group of connected specifications) that defines a broad variety of services for building distributed client-server applications. Several services one may expect to search out in a very middleware product like CORBA (e.g., naming, dealings, and asynchronous event management services) are literally fixed as services within the OMA. Different parts communicate via the ORB. ORB is additionally referred to as the item bus. An associate example of the application interface is a distributed document facility. In a very domain interface, it will have domain dependent services, for instance, producing domain.

IDL (Interface Definition Language)

- IDL is a language-independent way to define the interface of CORBA objects.
- Enables communication between different languages (e.g., Java client → C++ server).
- Defines:
 - Methods
 - Attributes
 - Data types
 - Exceptions

IDL Structure

```

module ExampleModule {
    interface Employee {
        // Method declaration
        string getName();
        void setName(in string name);
    }
}
    
```

```
// Another method
long getSalary();
void setSalary(in long salary);
};

};
```

Explanation:

- module ExampleModule → Namespace grouping interfaces
- interface Employee → Defines remote object interface
- string getName() → Method returning a string
- void setName(in string name) → Method accepting input
- long getSalary() → Method returning long
- in keyword → Input parameter

Question (2025)

1.Explain event handling model in Java in detail.

Event handling is the mechanism that controls what happens when a user interacts with a GUI component.

Examples of events:

- Clicking a button
- Pressing a key
- Moving the mouse
- Selecting a checkbox
- Closing a window

Java follows the Delegation Event Model to handle these actions.

Components of Event Handling Model

There are three main components:

Event Source

- The object that generates the event
- Examples:
 - Button
 - TextField
 - JButton
 - Checkbox

Example: A button that is clicked

Event Object

- An object that represents the event
- Contains information like:
 - Source of event
 - Type of event
 - Time of event

All event classes inherit from:

java.util.EventObject

Common event classes:

- ActionEvent
- MouseEvent
- KeyEvent
- WindowEvent
- ItemEvent

Event Listener

- An interface that receives and handles events
- Contains event-handling methods

Common listener interfaces:

- ActionListener
- MouseListener
- KeyListener
- WindowListener

- ItemListener

Example:

```
public void actionPerformed(ActionEvent e)
```

How Event Handling Works (Flow)

Here's the step-by-step flow:

1. User performs an action (clicks button)
2. Event source creates an event object
3. Event object is sent to registered listener
4. Listener's method executes
5. Required action is performed

Types of Events in Java

◊ Action Event

Triggered when:

- Button clicked
- Menu item selected
- Enter key pressed in text field

Class:

```
ActionEvent
```

Listener:

```
ActionListener
```

◊ Mouse Event

Triggered when:

- Mouse clicked
- Mouse entered/exited
- Mouse pressed/released

Class:

```
MouseEvent
```

Listener:

```
MouseListener
```

◊ Key Event

Triggered when:

- Key pressed
- Key released
- Key typed

Class:

```
KeyEvent
```

Listener:

```
KeyListener
```

◊ Window Event

Triggered when:

- Window opened
- Window closed

- Window activated

Class:

WindowEvent

Listener:

WindowListener

- ◊ Item Event

Triggered when:

- Checkbox selected/deselected
- Choice changed

Class:

ItemEvent

Listener:

ItemListener

Steps to Handle an Event in Java

To handle any event, follow these standard steps:

Step 1: Import required packages

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

Step 2: Create the GUI component (Event Source)

```
Button btn = new Button("Click Me");
```

Step 3: Implement Listener Interface

```
class MyClass implements ActionListener {
    public void actionPerformed(ActionEvent e) {
        System.out.println("Button clicked!");
    }
}
```

Step 4: Register Listener with Source

```
btn.addActionListener(this);
```

7. Simple Example (Button Click)

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

```
class EventDemo extends Frame implements ActionListener {
```

```
    Button btn;
```

```
    EventDemo() {
        btn = new Button("Click Me");
        btn.addActionListener(this);
```

```
        add(btn);
        setSize(300, 200);
        setVisible(true);
```

```

    }

    public void actionPerformed(ActionEvent e) {
        System.out.println("Button Clicked");
    }

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

```

Adapter Classes (Important Concept)

Some listener interfaces have many methods (e.g., MouseListener, WindowListener).

To avoid implementing all methods, Java provides Adapter Classes.

Example:

MouseAdapter

WindowAdapter

KeyAdapter

Usage:

```

addWindowListener(new WindowAdapter() {
    public void windowClosing(WindowEvent e) {
        System.exit(0);
    }
});

```

Advantages of Delegation Event Model

- Clean separation of logic
- Easy to maintain
- Multiple listeners can be added
- Improved performance
- Less coupling between GUI and logic

2.What is the use of Layout managers in GUI applications in Java? Explain with proper examples.

A Layout Manager in Java is used to automatically arrange and control the position and size of GUI components (buttons, labels, text fields, etc.) inside a container.

Instead of manually setting coordinates (setBounds()), layout managers handle:

- Component alignment
- Spacing
- Resizing when the window size changes

Why Layout Managers are Needed?

Without layout managers:

- Components overlap
- GUI breaks on different screen sizes
- Resizing window causes layout issues
- Platform dependency problems

Common Layout Managers in Java

| Layout Manager | Package | Use |
|----------------|-------------|---------------------------------|
| FlowLayout | java.awt | Simple row-wise layout |
| BorderLayout | java.awt | Divide container into 5 regions |
| GridLayout | java.awt | Grid (rows × columns) |
| GridBagLayout | java.awt | Flexible grid |
| CardLayout | java.awt | Multiple screens/cards |
| BoxLayout | javax.swing | Vertical / Horizontal layout |

FlowLayout

Description:

- Default layout for Panel
- Components placed left to right
- Wraps to next line if space is insufficient

Example:

```
import java.awt.*;

class FlowLayoutExample extends Frame {
    FlowLayoutExample() {
        setLayout(new FlowLayout());

        add(new Button("Button 1"));
        add(new Button("Button 2"));
        add(new Button("Button 3"));

        setSize(300, 200);
        setVisible(true);
    }
}
```

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

BorderLayout

Description:

- Default layout for Frame
- Divides container into 5 regions:

- North
- South
- East
- West
- Center

Example:

```
import java.awt.*;

class BorderLayoutExample extends Frame {
    BorderLayoutExample() {
        setLayout(new BorderLayout());

        add(new Button("North"), BorderLayout.NORTH);
        add(new Button("South"), BorderLayout.SOUTH);
        add(new Button("East"), BorderLayout.EAST);
        add(new Button("West"), BorderLayout.WEST);
        add(new Button("Center"), BorderLayout.CENTER);

        setSize(400, 300);
        setVisible(true);
    }

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

GridLayout

Description:

- Arranges components in rows and columns
- All components have equal size

Example:

```
import java.awt.*;

class GridLayoutExample extends Frame {
    GridLayoutExample() {
        setLayout(new GridLayout(2, 2));

        add(new Button("1"));
        add(new Button("2"));
        add(new Button("3"));
        add(new Button("4"));

        setSize(300, 200);
    }
}
```

```
    setVisible(true);
}

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

GridBagLayout

Description:

- Most flexible and complex layout
- Components can:
 - Span rows/columns
 - Have different sizes
 - Be aligned differently

Example:

```
import java.awt.*;

class GridBagLayoutExample extends Frame {
    GridBagLayoutExample() {
        GridBagLayout gbl = new GridBagLayout();
        GridBagConstraints gbc = new GridBagConstraints();

        setLayout(gbl);

        gbc.gridx = 0;
        gbc.gridy = 0;
        add(new Button("Button 1"), gbc);

        gbc.gridx = 1;
        gbc.gridy = 0;
        add(new Button("Button 2"), gbc);

        gbc.gridx = 0;
        gbc.gridy = 1;
        gbc.gridwidth = 2;
        add(new Button("Button 3"), gbc);

        setSize(400, 300);
        setVisible(true);
    }

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

```
    }
}
```

CardLayout

Description:

- Displays one component at a time
- Like flipping cards

Example:

```
import java.awt.*;

class CardLayoutExample extends Frame {
    CardLayout card;
    Panel panel;

    CardLayoutExample() {
        card = new CardLayout();
        panel = new Panel(card);

        panel.add("First", new Button("First Screen"));
        panel.add("Second", new Button("Second Screen"));

        add(panel);

        card.next(panel);

        setSize(300, 200);
        setVisible(true);
    }

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

3.What is the use of Resultset class in JDBC? Explain the types of Resultset in JDBC.

In JDBC, the ResultSet class is used to store and process the data retrieved from a database after executing a SQL SELECT query.

ResultSet is an interface in JDBC that holds the data returned by executing a SQL query and allows the program to read and manipulate the data row by row.

Why ResultSet is Used?

- To read data from database
- To navigate through rows
- To retrieve column values
- To update database records (for updatable ResultSets)

Basic Example of ResultSet

```
Statement stmt = con.createStatement();
ResultSet rs = stmt.executeQuery("SELECT * FROM students");

while (rs.next()) {
    System.out.println(rs.getInt("id") + " " + rs.getString("name"));
}
```

Cursor in ResultSet

- A cursor is a pointer that points to a row in the ResultSet
- Initially, cursor is before the first row
- Methods to move cursor:
 - next()
 - previous()
 - first()
 - last()

Types of ResultSet in JDBC

JDBC defines three types of ResultSet, based on cursor movement and sensitivity to database changes.

1. TYPE_FORWARD_ONLY

Description:

- Cursor moves only forward
- Default type
- Fastest and least memory usage

Characteristics:

- Only next() method allowed
- Cannot move backward
- Cannot detect database changes

Syntax:

```
Statement stmt = con.createStatement(
    ResultSet.TYPE_FORWARD_ONLY,
    ResultSet.CONCUR_READ_ONLY
);
```

Example:

```
while (rs.next()) {
    System.out.println(rs.getString("name"));
}
```

2. TYPE_SCROLL_INSENSITIVE

Description:

- Cursor can move forward and backward
- Does not reflect changes made by other users after ResultSet creation

Characteristics:

- Supports next(), previous(), first(), last()
- Data remains static

Syntax:

```
Statement stmt = con.createStatement(
    ResultSet.TYPE_SCROLL_INSENSITIVE,
    ResultSet.CONCUR_READ_ONLY
);
```

Example:

```
rs.last();
System.out.println(rs.getString("name"));
rs.first();
```

3. TYPE_SCROLL_SENSITIVE

Description:

- Cursor can move both directions
- Reflects real-time changes in the database

Characteristics:

- Sensitive to insert, update, delete operations
- Slower than insensitive type

Syntax:

```
Statement stmt = con.createStatement(
    ResultSet.TYPE_SCROLL_SENSITIVE,
    ResultSet.CONCUR_READ_ONLY
);
```

ResultSet Concurrency Types (Extra Important)

Apart from types, ResultSet also has concurrency modes:

1. CONCUR_READ_ONLY

- Data can only be read
- Default mode

2. CONCUR_UPDATABLE

- Data can be updated directly

Example:

```
Statement stmt = con.createStatement(
    ResultSet.TYPE_SCROLL_INSENSITIVE,
    ResultSet.CONCUR_UPDATABLE
);

rs.next();
rs.updateString("name", "Bibash");
rs.updateRow();
```

4.Define Javabean. How persistence is achieved in Javabeans?

A JavaBean is a reusable software component written in Java that follows a specific set of conventions, making it easy to use, reuse, and manage in applications.

A JavaBean is a reusable Java class that follows standard conventions such as having a no-argument constructor, private properties, and public getter and setter methods.

Features / Characteristics of JavaBeans

A JavaBean must follow these rules:

1. Public no-argument constructor
2. Private properties (variables)
3. Public getter and setter methods
4. Implements Serializable interface (for persistence)
5. Supports:
 - Properties
 - Events
 - Persistence

Example of a JavaBean

```
import java.io.Serializable;

public class StudentBean implements Serializable {

    private int id;
    private String name;

    // No-argument constructor
    public StudentBean() {}

    // Getter and Setter
    public int getId() {
        return id;
    }

    public void setId(int id) {
        this.id = id;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }
}
```

What is Persistence in JavaBeans?

Persistence means the ability of a JavaBean to save its state (property values) to a storage medium and restore it later.

How Persistence is Achieved in JavaBeans

JavaBeans achieve persistence mainly through serialization.

1. Persistence Using Serialization (Default Mechanism)

JavaBeans implement the Serializable interface.

How it works:

- The JVM converts the object into a byte stream
- The byte stream is stored (file, database, network)
- Later, it is restored back into an object

Example: Saving a JavaBean

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

```
public class SaveBean {  
    public static void main(String[] args) throws Exception {  
        StudentBean sb = new StudentBean();  
        sb.setId(1);  
        sb.setName("Bibash");  
  
        ObjectOutputStream oos =  
            new ObjectOutputStream(new FileOutputStream("student.ser"));  
        oos.writeObject(sb);  
        oos.close();  
    }  
}
```

Example: Loading a JavaBean

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

```
public class LoadBean {  
    public static void main(String[] args) throws Exception {  
        ObjectInputStream ois =  
            new ObjectInputStream(new FileInputStream("student.ser"));  
        StudentBean sb = (StudentBean) ois.readObject();  
        ois.close();  
  
        System.out.println(sb.getName());  
    }  
}
```

2. Persistence Using Externalization (Advanced)

JavaBeans can also implement Externalizable interface.

Characteristics:

- Developer controls how data is saved and restored

- Faster and more flexible
- Requires implementing:
 - writeExternal()
 - readExternal()

Example:

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

```
public class StudentBean implements Externalizable {

    private int id;
    private String name;

    public StudentBean() {}

    public void writeExternal(ObjectOutput out) throws IOException {
        out.writeInt(id);
        out.writeObject(name);
    }

    public void readExternal(ObjectInput in)
        throws IOException, ClassNotFoundException {
        id = in.readInt();
        name = (String) in.readObject();
    }
}
```

Advantages of Persistence in JavaBeans

- Maintains object state
- Supports component reuse
- Easy data storage and retrieval
- Useful in distributed applications

5.How JSP is different from servlet? Explain JSP scripting tag with an example.

Difference between JSP and Servlet

What is Servlet?

A Servlet is a Java class that runs on the server and handles client requests and responses using Java code.

What is JSP?

JSP (JavaServer Pages) is a server-side technology used to create dynamic web pages using HTML with embedded Java code.

JSP vs Servlet (Comparison Table)

| JSP | Servlet |
|---------------------------------------|---|
| JSP is HTML-based | Servlet is Java-based |
| Easier to write and maintain | More complex to write |
| Used mainly for presentation (view) | Used mainly for business logic (controller) |
| Java code is embedded inside HTML | HTML is written inside Java code |
| Automatically compiled into a servlet | Written directly as a servlet |
| Faster development | Better performance |
| Best for UI pages | Best for request handling |

Example Difference

Servlet Example:

```
out.println("<html><body>");
out.println("<h1>Hello World</h1>");
out.println("</body></html>");
```

JSP Example:

```
<h1>Hello World</h1>
```

JSP Scripting Tags

JSP Scripting Tags allow Java code to be embedded inside a JSP page.

There are three types of scripting tags:

1. Scriptlet Tag <% %>

Use:

- Write Java statements
- Executed every time the JSP page is requested

Syntax:

```
<% Java code %>
```

Example:

```
<%
    int a = 10;
    int b = 20;
    out.println("Sum = " + (a + b));
%>
```

2. Expression Tag <%= %>

Use:

- Display output directly on the webpage
- No semicolon required

Syntax:

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

Example:

```
<%= "Welcome to JSP" %>
```

3. Declaration Tag <%! %>

Use:

- Declare variables or methods
- Code placed outside the `_jspService()` method

Syntax:

```
<%! declaration %>
```

Example:

```
<%!
```

```
    int count = 0;
```

```
    int square(int x) {  
        return x * x;  
    }
```

```
%>
```

Square of 5 is: <%= square(5) %>

6.What are stub and skeleton? Also explain the use of RMI Registry.

Java RMI (Remote Method Invocation) allows an object running on one JVM to invoke methods of an object running on another JVM.

To make this happen, Java uses Stub and Skeleton.

- ◊ Stub (Client Side)

Definition:

A Stub is a client-side proxy object that represents the remote object.

Role of Stub:

- Acts as a local representative of the remote object
- Receives method calls from client
- Converts method calls into network requests
- Sends request to remote JVM
- Receives result and returns it to client

Simple Definition (Exam-Friendly):

Stub is a client-side object that forwards method calls to a remote object.

Stub Workflow:

1. Client calls a method on stub
2. Stub serializes arguments
3. Stub sends request to server
4. Stub receives result
5. Stub returns result to client

- ◊ Skeleton (Server Side)

Definition:

A Skeleton is a server-side helper object that receives requests from the stub and invokes the actual method on the remote object.

Role of Skeleton:

- Receives request from stub
- Deserializes arguments
- Calls actual remote method
- Sends result back to stub

Stub vs Skeleton (Quick Comparison)

| Stub | Skeleton |
|----------------------|---------------------------|
| Client-side | Server-side |
| Sends request | Receives request |
| Acts as proxy | Acts as dispatcher |
| Exists in modern RMI | Obsolete (handled by JVM) |

2.What is RMI Registry?

Definition:

The RMI Registry is a naming service that allows clients to locate remote objects by name.

It works like a directory or phonebook for remote objects.

Why RMI Registry is Needed?

- Client needs to find remote object
- Client doesn't know:
 - Object location
 - JVM address
 - Port number
- Registry provides this mapping

How RMI Registry Works

On Server Side:

1. Server creates remote object
2. Registers it with RMI Registry using a name

```
Naming.rebind("CalculatorService", remoteObject);
```

On Client Side:

1. Client looks up object by name
2. Receives stub reference

Calculator stub =

```
(Calculator) Naming.lookup("rmi://localhost/CalculatorService");
```

Registry Functions:

- bind()
- rebind()
- lookup()
- unbind()

Role of RMI Registry (Exam Points)

- Acts as central lookup service
- Stores name–object mapping
- Helps client locate remote object
- Runs on default port 1099

Simple Real-Life Analogy

- Stub → Receptionist who forwards your request
- Skeleton → Manager who executes the task
- RMI Registry → Office directory

7. Write short notes on (Any Two):

a) Declaration Tag in JSP

The Declaration tag in JSP is used to declare variables and methods that belong to the JSP page.

- Code written inside declaration tag is placed outside the `_jspService()` method
- Variables declared here are instance variables
- Methods declared can be used anywhere in the JSP page

Syntax:

```
<%! declaration %>
```

Example:

```
<%!
    int count = 0;
    int square(int x) {
        return x * x;
    }
%>
```

Square of 4 is: `<%= square(4) %>`

Key Points:

- Executed once per JSP instance
- Used for reusable logic
- Different from scriptlet tag

b) Graphics Classes

The Graphics class in Java is used to draw shapes, text, and images in GUI applications.

- Part of `java.awt` package
- Used in components like Frame, Panel, Applet
- Graphics object is obtained via `paint()` method

Common Graphics Methods:

- `drawLine()`

- drawRect()
- fillRect()
- drawOval()
- drawString()

Example:

```
import java.awt.*;
public class GraphicsExample extends Frame {
    public void paint(Graphics g) {
        g.drawRect(50, 50, 100, 80);
        g.drawString("Hello Graphics", 60, 150);
    }
    public static void main(String[] args) {
        new GraphicsExample().setVisible(true);
    }
}
```

Use:

- Drawing charts
- Custom UI components
- Games and animations

c) Join in JDBC

A Join in JDBC is used to retrieve data from multiple database tables using a single SQL query.

- JDBC itself does not define joins
- Joins are written in SQL, executed using JDBC
- Helps combine related data

Types of Joins:

- INNER JOIN
- LEFT JOIN

- RIGHT JOIN

Example:

```
String sql =
"SELECT s.name, c.course " +
"FROM student s INNER JOIN course c " +
"ON s.id = c.sid";
Statement stmt = con.createStatement();
ResultSet rs = stmt.executeQuery(sql);
```

```
while (rs.next()) {
    System.out.println(rs.getString("name") +
        " " + rs.getString("course"));
}
```

Use:

- Fetching relational data
- Reports
- Database-driven applications

9.What is window event? List and explain WindowListener interface methods in detail.

What is a Window Event?

A Window Event occurs when a user interacts with a window (Frame / Dialog) in a Java GUI application.

These events are generated when actions like:

- Opening a window
- Closing a window
- Activating or deactivating a window
- Minimizing or restoring a window

WindowListener Interface

The WindowListener interface is used to handle window events.

- It contains 7 abstract methods
- A class must implement all methods of this interface
- Used with components like Frame

List of WindowListener Methods

| Method | Event Trigger |
|---------------------|-------------------------------|
| windowOpened() | When window is opened |
| windowClosing() | When user clicks close button |
| windowClosed() | After window is closed |
| windowActivated() | When window becomes active |
| windowDeactivated() | When window loses focus |
| windowIconified() | When window is minimized |
| windowDeiconified() | When window is restored |

Explanation of Each Method

1. windowOpened(WindowEvent e)

- Called when the window is opened for the first time
- Used for initialization tasks

```
public void windowOpened(WindowEvent e) {
```

```
    System.out.println("Window opened");
```

```
}
```

2. windowClosing(WindowEvent e)

- Called when the user clicks the close (X) button
- Commonly used to terminate the application

```
public void windowClosing(WindowEvent e) {
```

```
    System.exit(0);
```

```
}
```

3. windowClosed(WindowEvent e)

- Called after the window has been closed
- Used for cleanup operations

```
public void windowClosed(WindowEvent e) {
```

```
    System.out.println("Window closed");
```

```
}
```

4. windowActivated(WindowEvent e)

- Called when the window becomes active
- Window gains focus

```
public void windowActivated(WindowEvent e) {
```

```
    System.out.println("Window activated");
```

```
}
```

windowDeactivated(WindowEvent e)

- Called when the window loses focus
- Another window becomes active

```
public void windowDeactivated(WindowEvent e) {
```

```
    System.out.println("Window deactivated");
```

```
}
```

6. windowIconified(WindowEvent e)

- Called when the window is minimized
- (Iconified = minimized)

```

public void windowIconified(WindowEvent e) {
    System.out.println("Window minimized");
}
7.windowDeiconified(WindowEvent e)
    • Called when the window is restored from minimized state
public void windowDeiconified(WindowEvent e) {
    System.out.println("Window restored");
}
Simple Example Using WindowListener
import java.awt.*;
import java.awt.event.*;

public class WindowEventExample extends Frame
    implements WindowListener {

    WindowEventExample() {
        addWindowListener(this);
        setSize(300, 200);
        setVisible(true);
    }

    public void windowClosing(WindowEvent e) {
        System.exit(0);
    }

    public void windowOpened(WindowEvent e) {}
    public void windowClosed(WindowEvent e) {}
    public void windowActivated(WindowEvent e) {}
    public void windowDeactivated(WindowEvent e) {}
    public void windowIconified(WindowEvent e) {}
    public void windowDeiconified(WindowEvent e) {}

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

```

10.What are HttpServletRequest and HttpServletResponse? Create a servlet application that takes student's information including their roll number, name, address and level and store these information in the database.

What is HttpServletRequest?

HttpServletRequest is an interface that represents the client's request sent to the servlet.

It is used to read data sent by the client (form data, parameters, headers, etc.).

Common Uses:

- Get form input values
- Read request headers
- Get client information

Important Methods:

```
request.getParameter("name");
request.getParameterValues("hobby");
request.getMethod();
request.getSession();
```

Example:

```
String name = request.getParameter("name");
```

2.What is HttpServletResponse?

HttpServletResponse is an interface that represents the response sent by the servlet to the client.

It is used to send output (HTML, text, redirect, status codes) back to the browser.

Common Uses:

- Send HTML response
- Redirect to another page
- Set response type

Important Methods:

```
response.getWriter();
response.setContentType("text/html");
response.sendRedirect("success.jsp");
```

Example:

```
PrintWriter out = response.getWriter();
out.println("Data saved successfully");
```

Short Definition (Exam-Friendly)

HttpServletRequest is used to read client request data, while HttpServletResponse is used to send response data from servlet to client.

3.Servlet Application: Store Student Information in Database

Student Information:

- Roll Number
- Name
- Address
- Level

Step 1 Database Table (MySQL)

```
CREATE TABLE student (
    roll INT PRIMARY KEY,
    name VARCHAR(100),
    address VARCHAR(150),
    level VARCHAR(50)
);
```

Step 2 HTML Form (student.html)

```

<!DOCTYPE html>
<html>
<body>
<form action="StudentServlet" method="post">
    Roll No: <input type="text" name="roll"><br><br>
    Name: <input type="text" name="name"><br><br>
    Address: <input type="text" name="address"><br><br>
    Level: <input type="text" name="level"><br><br>
    <input type="submit" value="Save">
</form>
</body>
</html>

```

Step 3 Servlet Code (StudentServlet.java)

```

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

public class StudentServlet extends HttpServlet {

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

        response.setContentType("text/html");
        PrintWriter out = response.getWriter();

        // Read form data using HttpServletRequest
        int roll = Integer.parseInt(request.getParameter("roll"));
        String name = request.getParameter("name");
        String address = request.getParameter("address");
        String level = request.getParameter("level");

        try {
            Class.forName("com.mysql.cj.jdbc.Driver");
            Connection con = DriverManager.getConnection(
                "jdbc:mysql://localhost:3306/college",
                "root", ""
            );

            PreparedStatement ps = con.prepareStatement(
                "INSERT INTO student VALUES (?, ?, ?, ?)"
            );

```

```

        ps.setInt(1, roll);
        ps.setString(2, name);
        ps.setString(3, address);
        ps.setString(4, level);

        ps.executeUpdate();

        out.println("<h3>Student record saved successfully!</h3>");

        con.close();
    } catch (Exception e) {
        out.println("Error: " + e);
    }
}
}

```

Step 4 web.xml Mapping (Optional but Exam-Useful)

```

<web-app>
    <servlet>
        <servlet-name>StudentServlet</servlet-name>
        <servlet-class>StudentServlet</servlet-class>
    </servlet>

    <servlet-mapping>
        <servlet-name>StudentServlet</servlet-name>
        <url-pattern>/StudentServlet</url-pattern>
    </servlet-mapping>
</web-app>

```

11.Differentiate between RMI and CORBA. Write a client and server program to accept two numbers and display the greatest one.

Difference between RMI and CORBA

| RMI | CORBA |
|-------------------------------------|--|
| Stands for Remote Method Invocation | Stands for Common Object Request Broker Architecture |
| Java-specific | Language-independent |
| Uses Java objects | Uses objects written in multiple languages |
| Easy to implement | More complex |
| Uses Java serialization | Uses IDL (Interface Definition Language) |
| Platform dependent | Platform independent |
| Faster in Java-only apps | Suitable for heterogeneous systems |

RMI is suitable for Java-to-Java communication, while CORBA is used for communication between objects written in different programming languages.

RMI Program: Find Greatest of Two Numbers

We'll create:

1. Remote Interface
2. Server Program
3. Client Program

Step 1 Remote Interface (Greatest.java)

```
import java.rmi.*;
```

```
public interface Greatest extends Remote {  
    int findGreatest(int a, int b) throws RemoteException;  
}
```

Step 2 Server Implementation (GreatestImpl.java)

```
import java.rmi.*;  
import java.rmi.server.*;
```

```
public class GreatestImpl extends UnicastRemoteObject  
    implements Greatest {  
  
    public GreatestImpl() throws RemoteException {  
        super();  
    }
```

```
    public int findGreatest(int a, int b) {  
        return (a > b) ? a : b;  
    }  
}
```

Step 3 RMI Server (Server.java)

```
import java.rmi.*;
```

```
public class Server {  
    public static void main(String[] args) {  
        try {  
            Greatest obj = new GreatestImpl();  
            Naming.rebind("GreatestService", obj);  
            System.out.println("Server is running...");  
        } catch (Exception e) {  
            System.out.println(e);  
        }  
    }  
}
```

Step 4 RMI Client (Client.java)

```
import java.rmi.*;
import java.util.Scanner;

public class Client {
    public static void main(String[] args) {
        try {
            Greatest stub = (Greatest)
                Naming.lookup("rmi://localhost/GreatestService");

            Scanner sc = new Scanner(System.in);
            System.out.print("Enter first number: ");
            int a = sc.nextInt();
            System.out.print("Enter second number: ");
            int b = sc.nextInt();

            int result = stub.findGreatest(a, b);
            System.out.println("Greatest number is: " + result);

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

Question (2024)

2.What is the use of WindowEvent? List different methods of WindowListener interface with its uses.

What is WindowEvent in Java?

In Java AWT/Swing GUI programming, a window event is an event that occurs when an action happens to a window, such as opening, closing, minimizing, or activating it.

WindowEvent is a class in `java.awt.event` package that represents events related to a window (like a Frame or Dialog).

It is triggered when a window state changes, and it is handled by implementing the `WindowListener` interface.

Common uses of WindowEvent

- Detect when a user closes a window.
- Detect when a window is minimized, restored, or activated.
- Perform cleanup or save data before closing a window.

2. WindowListener Interface

The `WindowListener` interface is used to listen to window events. Any class that wants to respond to window events must implement this interface.

Methods of `WindowListener`

`WindowListener` has 7 abstract methods, each handling a specific type of window event:

| Method | Description / Use |
|--|--|
| <code>void windowOpened(WindowEvent e)</code> | Called when a window is first opened. Useful to initialize resources. |
| <code>void windowClosing(WindowEvent e)</code> | Called when the user clicks the close button. Useful to confirm exit or save data. |
| <code>void windowClosed(WindowEvent e)</code> | Called after the window is actually closed. |
| <code>void windowIconified(WindowEvent e)</code> | Called when the window is minimized. |
| <code>void windowDeiconified(WindowEvent e)</code> | Called when the window is restored from minimized state. |
| <code>void windowActivated(WindowEvent e)</code> | Called when the window becomes the active window (gains focus). |
| <code>void windowDeactivated(WindowEvent e)</code> | Called when the window loses focus (another window becomes active). |

Example: Handling Window Events using `WindowListener`

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

```
public class WindowEventExample extends Frame implements WindowListener {
```

```
    WindowEventExample() {
        setTitle("Window Event Example");
        setSize(400, 300);
```

```

setVisible(true);

// Registering WindowListener
addWindowListener(this);
}

// Implementing all WindowListener methods

public void windowOpened(WindowEvent e) {
    System.out.println("Window Opened");
}

public void windowClosing(WindowEvent e) {
    System.out.println("Window Closing");
    System.exit(0); // Close the application
}

public void windowClosed(WindowEvent e) {
    System.out.println("Window Closed");
}

public void windowIconified(WindowEvent e) {
    System.out.println("Window Minimized");
}

public void windowDeiconified(WindowEvent e) {
    System.out.println("Window Restored");
}

public void windowActivated(WindowEvent e) {
    System.out.println("Window Activated");
}

public void windowDeactivated(WindowEvent e) {
    System.out.println("Window Deactivated");
}

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

```

Explanation of the Example

1. We extend Frame to create a GUI window.

2. We implement WindowListener to handle window events.
3. addWindowListener(this) registers the listener with the window.
4. Each method prints a message when the corresponding event occurs.
5. windowClosing method uses System.exit(0) to terminate the program when the user closes the window.

Using WindowAdapter

If you don't want to implement all 7 methods, you can extend WindowAdapter:

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

```
public class WindowAdapterExample extends Frame {
```

```
    WindowAdapterExample() {
        setTitle("Window Adapter Example");
        setSize(400, 300);
        setVisible(true);

        addWindowListener(new WindowAdapter() {
            public void windowClosing(WindowEvent e) {
                System.out.println("Closing the window...");
                System.exit(0);
            }
        });
    }

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

3.How can you display images in GUI using Java? Explain.

Introduction

In Java GUI programming (AWT/Swing), you can display images in your applications to make them interactive and visually appealing.

There are two main ways to display images:

1. Using AWT (Canvas, Graphics and Image classes)
 2. Using Swing (JLabel with ImageIcon)
2. Using Swing (Recommended)

Swing provides a simpler and more flexible way to display images using ImageIcon and JLabel.

Steps:

1. Create an ImageIcon object from an image file.
2. Add it to a JLabel.

3. Add the JLabel to a JFrame.

Example: Display Image using JLabel

```
import javax.swing.*;
```

```
public class ImageExample {  
    public static void main(String[] args) {  
        // Create a frame  
        JFrame frame = new JFrame("Display Image Example");  
        frame.setSize(500, 500);  
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);  
  
        // Load the image  
        ImageIcon image = new ImageIcon("example.jpg"); // Image file path  
  
        // Add image to JLabel  
        JLabel label = new JLabel(image);  
  
        // Add label to frame  
        frame.add(label);  
  
        frame.setVisible(true);  
    }  
}
```

Explanation:

- `ImageIcon("example.jpg")` loads the image.
- `JLabel label = new JLabel(image)` allows the image to be displayed in the label.
- Adding the label to the frame shows the image in the GUI.

Using AWT and Graphics

AWT allows more control over drawing images using the `Graphics` class.

Steps:

1. Load the image using Toolkit or ImageIO.
2. Override the paint method of a Frame or Canvas.
3. Draw the image using `Graphics.drawImage`.

Example: Display Image using AWT

```
import java.awt.*;  
import java.io.File;  
import java.io.IOException;  
import javax.imageio.ImageIO;
```

```
public class ImageAWTEExample extends Frame {
```

```
    Image img;
```

```
    ImageAWTEExample() {
```

```

setTitle("AWT Image Example");
setSize(500, 500);
setVisible(true);

// Load image
try {
    img = ImageIO.read(new File("example.jpg"));
} catch (IOException e) {
    e.printStackTrace();
}

// Close window
addWindowListener(new java.awt.event.WindowAdapter() {
    public void windowClosing(java.awt.event.WindowEvent e) {
        System.exit(0);
    }
});

public void paint(Graphics g) {
    g.drawImage(img, 50, 50, this); // Draw image at x=50, y=50
}

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

```

Explanation:

- `ImageIO.read()` reads an image file into an `Image` object.
- `paint(Graphics g)` method is overridden to draw the image on the frame.
- `g.drawImage(img, x, y, this)` draws the image at specific coordinates.

Using JPanel with paintComponent (Swing Approach)

If you want to draw the image on a panel instead of a label:

```

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

```

```

public class ImagePanelExample extends JPanel {
    Image img;

    ImagePanelExample() {
        img = new ImageIcon("example.jpg").getImage();
    }
}

```

```

@Override
protected void paintComponent(Graphics g) {
    super.paintComponent(g);
    g.drawImage(img, 0, 0, this); // Draw image starting at (0,0)
}

public static void main(String[] args) {
    JFrame frame = new JFrame("Image on Panel");
    frame.setSize(500, 500);
    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    frame.add(new ImagePanelExample());
    frame.setVisible(true);
}
}

```

Advantages:

- More control over image position, scaling, or custom graphics.
- Can combine text and images in the same panel.

4.Why do we need to create Javabeans? Explain the steps involved in creation of Javabean.

Why do we need JavaBeans?

JavaBeans are reusable software components in Java that can be manipulated visually in a builder tool (like NetBeans, Eclipse GUI builder). They are used to encapsulate data and provide methods to access it.

Reasons for using JavaBeans:

1. Reusability
 - Once created, a JavaBean can be used in multiple programs without rewriting code.
2. Encapsulation
 - JavaBeans follow the principle of encapsulation by keeping properties private and providing public getter and setter methods.
3. Persistence
 - JavaBeans can be saved and restored because they implement Serializable. This allows their state to be preserved.
4. Easy integration with GUI builders
 - They can be dragged and dropped into GUI tools, making development faster.
5. Standard conventions
 - JavaBeans follow a standard naming pattern for methods, which allows automatic introspection by development tools.

Features of a JavaBean

A JavaBean must have:

1. Private properties (fields)
2. private String name;

3. private int age;
4. Public getter and setter methods
5. public String getName() { return name; }
6. public void setName(String name) { this.name = name; }
7. A public no-argument constructor
8. public StudentBean() { }
9. Serializable (optional but recommended)
10. import java.io.Serializable;
11. public class StudentBean implements Serializable { ... }

Steps to Create a JavaBean

Here is a step-by-step procedure:

Step 1: Create a Java class

- The class should be public and follow naming conventions.

import java.io.Serializable;

```
public class StudentBean implements Serializable {
```

// Step 2: Declare private properties

```
private String name;
```

```
private int rollNo;
```

```
private String course;
```

// Step 3: Provide public no-argument constructor

```
public StudentBean() {
```

```
}
```

// Step 4: Provide getter and setter methods

```
public String getName() {
```

```
    return name;
```

```
}
```

```
public void setName(String name) {
```

```
    this.name = name;
```

```
}
```

```
public int getRollNo() {
```

```
    return rollNo;
```

```
}
```

```
public void setRollNo(int rollNo) {
```

```
    this.rollNo = rollNo;
```

```
}
```

```

public String getCourse() {
    return course;
}

public void setCourse(String course) {
    this.course = course;
}
}

```

Step 5: Use the JavaBean

- You can use this bean in other Java programs or GUI builders.

```

public class TestBean {
    public static void main(String[] args) {
        // Creating JavaBean object
        StudentBean student = new StudentBean();

        // Setting values using setter methods
        student.setName("Bibash");
        student.setRollNo(101);
        student.setCourse("BCA");

        // Getting values using getter methods
        System.out.println("Name: " + student.getName());
        System.out.println("Roll No: " + student.getRollNo());
        System.out.println("Course: " + student.getCourse());
    }
}

```

5.What is the use of PreparedStatement? Differentiate between Statement and PreparedStatement.

What is PreparedStatement?

PreparedStatement is an interface in JDBC used to execute precompiled SQL queries.

It is an improved version of Statement, which allows you to:

- Execute parameterized queries (queries with placeholders ?)
- Avoid SQL injection
- Improve performance for repeated queries

Key Features of PreparedStatement:

1. Parameterized Queries:

You can pass values dynamically using ? placeholders.

2. Precompiled SQL:

The SQL query is compiled once and can be executed multiple times with different values.

3. Prevents SQL Injection:

Since the input values are automatically escaped, it is much safer than concatenating strings in Statement.

2. Syntax of PreparedStatement

```
PreparedStatement pst = con.prepareStatement("INSERT INTO students(roll, name, course)  
VALUES(?, ?, ?)");
```

- ? are placeholders for parameters.
- You can set the values using setter methods like setInt, setString, etc.

3. Example: Using PreparedStatement

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

```
public class PreparedStatementExample {  
    public static void main(String[] args) {  
        try {  
            // Load JDBC driver  
            Class.forName("com.mysql.cj.jdbc.Driver");  
  
            // Create connection  
            Connection con = DriverManager.getConnection(  
                "jdbc:mysql://localhost:3306/school", "root", "password");  
  
            // Create PreparedStatement  
            String sql = "INSERT INTO students(roll, name, course) VALUES(?, ?, ?)";  
            PreparedStatement pst = con.prepareStatement(sql);  
  
            // Set values  
            pst.setInt(1, 101);      // first parameter  
            pst.setString(2, "Bibash"); // second parameter  
            pst.setString(3, "BCA");   // third parameter  
  
            // Execute query  
            int rows = pst.executeUpdate();  
            System.out.println(rows + " row(s) inserted.");  
  
            // Close connection  
            pst.close();  
            con.close();  
  
        } catch (Exception e) {  
            e.printStackTrace();  
        }  
    }  
}
```

Advantages:

- Safer and avoids SQL injection
- Faster for repeated queries
- Cleaner and easier to maintain

Difference Between Statement and PreparedStatement

| Feature | Statement | PreparedStatement |
|---------------|---|--|
| Definition | Used to execute static SQL queries. | Used to execute parameterized SQL queries. |
| SQL Injection | Vulnerable to SQL injection. | Safe from SQL injection. |
| Performance | Slower for repeated queries (compiled each time). | Faster for repeated queries (precompiled). |
| Parameters | Cannot use placeholders. | Can use ? placeholders and bind values dynamically. |
| Reusability | Query needs to be rewritten for different values. | Can execute same query multiple times with different values. |
| Use case | Simple queries executed once. | Queries executed multiple times with dynamic values. |

Example Comparison:

Statement (Vulnerable to SQL Injection)

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

```
String name = "Bibash";
```

```
stmt.executeUpdate("INSERT INTO students(name) VALUES('" + name + "')");
```

PreparedStatement (Safe)

```
PreparedStatement pst = con.prepareStatement("INSERT INTO students(name)
```

```
VALUES(?);
```

```
pst.setString(1, "Bibash");
```

```
pst.executeUpdate();
```

6.What are declaration and expression tags in JSP? Explain with example.

JSP Overview

JSP (JavaServer Pages) allows embedding Java code into HTML pages.

In JSP, there are three main scripting elements:

1. Declaration (<%! ... %>)
2. Expression (<%= ... %>)
3. Scriptlet (<% ... %>)

Here, we focus on Declaration and Expression tags.

2. Declaration Tag <%! ... %>

Definition:

- Used to declare variables and methods in JSP.
- Variables/methods declared inside <%! ... %> become instance variables or methods of the generated servlet class.

- They are available throughout the JSP page.

Syntax:

```
<%!
    int count = 0;      // variable declaration
    public int add(int a, int b) { // method declaration
        return a + b;
    }
%>
```

Example: Using Declaration Tag

```
<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>
<html>
<head><title>Declaration Example</title></head>
<body>
<%!
    int counter = 0;
    public int increment() {
        counter++;
        return counter;
    }
%>

<h2>Counter Example</h2>
<%
    int value = increment();
%>
<p>Current Counter Value: <%= value %></p>
</body>
</html>
```

Explanation:

- counter is declared using Declaration tag, so it persists across requests (per servlet instance).
- increment() is a method that can be called anywhere in the JSP.
- Declaration tag cannot be used to directly print values; use Expression tag for that.

Expression Tag <%= ... %>

Definition:

- Used to display the result of Java expression directly in HTML.
- The expression is evaluated and converted to a string automatically.
- No need to use out.println(); JSP does it internally.

Syntax:

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

Example: Using Expression Tag

```

<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>
<html>
<head><title>Expression Example</title></head>
<body>

<%
    int a = 10;
    int b = 20;
%>

<p>Sum of <%= a %> and <%= b %> is <%= (a + b) %></p>

</body>
</html>

```

Explanation:

- <%= a %> prints the value of a directly in HTML.
- <%= (a + b) %> prints the sum of a and b.
- Expression tag evaluates the Java expression and inserts the result into HTML.

Difference Between Declaration and Expression Tags

| Feature | Declaration <%! %> | Expression <%= %> |
|---------------|------------------------------|------------------------------------|
| Purpose | Declare variables or methods | Display value or expression result |
| Direct Output | No (cannot print directly) | Yes (prints result in HTML) |
| Scope | Becomes class-level members | Evaluated at the point of use |
| Example | <%! int x = 10; %> | <%= x %> |

7.What is RMI? Write a RMI client and server program to find the greater number among two numbers.

What is RMI?

RMI (Remote Method Invocation) is a Java API that allows an object running in one Java virtual machine (JVM) to invoke methods on an object running in another JVM.

Key Points:

- Allows distributed computing in Java.
- Uses stub and skeleton objects:
 - Stub: Client-side proxy for the remote object.
 - Skeleton: Server-side entity that receives requests (in Java 8+, skeleton is optional).
- Requires remote interfaces that extend java.rmi.Remote.

Steps to Create RMI Application

1. Define Remote Interface
2. Implement Remote Interface (Server class)
3. Create and Start RMI Registry
4. Bind Server object to RMI registry

5. Create Client to call remote method

RMI Program to Find Greater Number

Step 1: Create Remote Interface

```
import java.rmi.Remote;
import java.rmi.RemoteException;
```

```
public interface Greater extends Remote {
    int findGreater(int a, int b) throws RemoteException;
}
```

Step 2: Implement Server Class

```
import java.rmi.*;
import java.rmi.server.*;
```

```
public class GreaterImpl extends UnicastRemoteObject implements Greater {
```

```
// Constructor
public GreaterImpl() throws RemoteException {
    super();
}
```

```
// Implement remote method
public int findGreater(int a, int b) throws RemoteException {
    return (a > b) ? a : b;
}
```

```
public static void main(String[] args) {
    try {
        // Create server object
        GreaterImpl obj = new GreaterImpl();

        // Bind object in RMI Registry
        Naming.rebind("rmi://localhost:1099/GreaterService", obj);
```

```
        System.out.println("Server is ready...");
    } catch (Exception e) {
        e.printStackTrace();
    }
}
```

Step 3: Create Client Class

```
import java.rmi.*;
```

```
public class GreaterClient {
```

```

public static void main(String[] args) {
    try {
        // Lookup remote object
        Greater obj = (Greater) Naming.lookup("rmi://localhost:1099/GreaterService");

        int a = 15;
        int b = 25;

        // Call remote method
        int greater = obj.findGreater(a, b);
        System.out.println("Greater number between " + a + " and " + b + " is: " + greater);

    } catch (Exception e) {
        e.printStackTrace();
    }
}

```

Steps to Run the RMI Program

1. Compile all classes:

javac *.java

2. Start RMI Registry (from command line in the same directory):

rmiregistry

Note: Default port is 1099. Keep this running.

3. Run Server:

java GreaterImpl

4. Run Client:

java GreaterClient

Expected Output (Client side):

Greater number between 15 and 25 is: 25

8. Write short notes on:

a) Menubar in Java GUI

Definition:

A Menubar is a GUI component in Java that provides a horizontal menu bar at the top of a window. It can contain menus, and each menu can have menu items.

Key Classes:

- MenuBar → Represents the menubar.
- Menu → Represents a menu (e.g., File, Edit).
- MenuItem → Represents individual menu options.
- CheckboxMenuItem → Menu items with checkboxes.

Example:

```

import java.awt.*;
import java.awt.event.*;

```

```

public class MenuExample extends Frame {
    MenuExample() {
        // Create menubar
        MenuBar mb = new MenuBar();

        // Create menu
        Menu file = new Menu("File");

        // Create menu items
        MenuItem newItem = new MenuItem("New");
        MenuItem openItem = new MenuItem("Open");
        MenuItem exitItem = new MenuItem("Exit");

        // Add menu items to menu
        file.add(newItem);
        file.add(openItem);
        file.addSeparator();
        file.add(exitItem);

        // Add menu to menubar
        mb.add(file);

        // Set menubar to frame
        setMenuBar(mb);

        // Exit action
        exitItem.addActionListener(e -> System.exit(0));
    }

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

```

Key Points:

- Menubar provides easy navigation in GUI applications.
- Supports nested menus (submenu).
- Often used in desktop applications.

b) Session Handling in Servlet

Definition:

Session handling is a mechanism to maintain user state across multiple HTTP requests in a web application.

HTTP is stateless, so sessions are used to remember user-specific information (e.g., login, shopping cart).

Techniques for Session Handling in Servlets:

1. Using HttpSession (Preferred)

- Provides a session object to store attributes for a user.
- Example:

```
// Create or get session
```

```
HttpSession session = request.getSession();
```

```
// Set attribute
```

```
session.setAttribute("username", "Bibash");
```

```
// Get attribute
```

```
String user = (String) session.getAttribute("username");
```

2. Using Cookies

- Store small user data in client browser.
- Limited to ~4KB and not secure for sensitive info.

3. URL Rewriting

- Append session ID to the URL.
- Used when cookies are disabled.

Key Methods of HttpSession:

- getSession() → Returns current session or creates a new one
- setAttribute(name, value) → Store data in session
- getAttribute(name) → Retrieve stored data
- invalidate() → Destroy session

Example Use Case:

- User logs in → Session stores username
- On next page → Session retrieves username → Personalized greeting

9.Why do we need to use layout managers in GUI programming? Write a program to create a GUI form that takes data about a customer including cid, name, address, email and age and insert these data into the Customer table of the Shop database.

Why Use Layout Managers in Java GUI?

Definition:

A layout manager is an object that controls the size and position of GUI components (like buttons, labels, text fields) in a container (like JFrame or Panel).

Reasons to Use Layout Managers:

1. Automatic Component Arrangement:

- Avoids manually setting pixel positions with setBounds().
- Components adjust automatically if the window is resized.

2. Platform Independence:
 - Different platforms (Windows, Linux, Mac) have different screen resolutions.
Layout managers ensure components are arranged consistently.
3. Dynamic Resizing:
 - Layout managers automatically adjust components when the **frame size changes**.
4. Cleaner and Maintainable Code:
 - Easier than manually managing coordinates.

Common Layout Managers:

| Layout Manager | Description |
|----------------|---|
| FlowLayout | Components arranged left-to-right in a row. |
| BorderLayout | Divides container into North, South, East, West, Center. |
| GridLayout | Components arranged in equal-sized rows and columns. |
| GridBagLayout | Flexible layout, components can span multiple rows/columns. |

Java GUI Program to Insert Customer Data

This program will:

1. Take customer data from the user.
2. Insert it into the Customer table of the Shop database using JDBC.

Program:

```

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

public class CustomerForm extends JFrame implements ActionListener {
    // Components
    JLabel l1, l2, l3, l4, l5;
    JTextField tf1, tf2, tf3, tf4, tf5;
    JButton submit;

    // Database connection
    Connection con;
    PreparedStatement pst;

    public CustomerForm() {
        // Set Frame properties
        setTitle("Customer Form");
        setSize(400, 350);
        setLayout(new GridLayout(6, 2, 10, 10)); // 6 rows, 2 columns
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

        // Create Labels
        l1 = new JLabel("Customer ID");
        l2 = new JLabel("Name");
        l3 = new JLabel("Address");
        l4 = new JLabel("City");
        l5 = new JLabel("State");
        
```

```

l1 = new JLabel("Customer ID:");
l2 = new JLabel("Name:");
l3 = new JLabel("Address:");
l4 = new JLabel("Email:");
l5 = new JLabel("Age:");

// Create TextFields
tf1 = new JTextField();
tf2 = new JTextField();
tf3 = new JTextField();
tf4 = new JTextField();
tf5 = new JTextField();

// Create Button
submit = new JButton("Insert");
submit.addActionListener(this);

// Add components to Frame
add(l1); add(tf1);
add(l2); add(tf2);
add(l3); add(tf3);
add(l4); add(tf4);
add(l5); add(tf5);
add(new JLabel()); // empty cell
add(submit);

// Make frame visible
setVisible(true);

// Setup DB connection
try {
    Class.forName("com.mysql.cj.jdbc.Driver");
    con = DriverManager.getConnection(
        "jdbc:mysql://localhost:3306/shop", "root", "password");
} catch (Exception e) {
    e.printStackTrace();
}

// Handle button click
public void actionPerformed(ActionEvent ae) {
    try {
        int cid = Integer.parseInt(tf1.getText());

```

```

String name = tf2.getText();
String address = tf3.getText();
String email = tf4.getText();
int age = Integer.parseInt(tf5.getText());

String sql = "INSERT INTO Customer(cid, name, address, email, age) VALUES (?, ?, ?, ?, ?)";
pst = con.prepareStatement(sql);
pst.setInt(1, cid);
pst.setString(2, name);
pst.setString(3, address);
pst.setString(4, email);
pst.setInt(5, age);

int rows = pst.executeUpdate();
if (rows > 0) {
    JOptionPane.showMessageDialog(this, "Customer inserted successfully!");
}

// Clear fields
tf1.setText(""); tf2.setText(""); tf3.setText(""); tf4.setText(""); tf5.setText("");
} catch (Exception e) {
    JOptionPane.showMessageDialog(this, "Error: " + e.getMessage());
}
}

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

```

Explanation of the Program

1. Layout Manager Used:
 - o GridLayout(6, 2) → 6 rows, 2 columns, evenly spaced components.
 - o Makes form neat and resizable.
2. Database Connection:
 - o Connects to Shop database using JDBC.
 - o Uses PreparedStatement for safe insertion.
3. Button Action:
 - o On clicking Insert, values are read from text fields and inserted into the database.
 - o Displays success or error message.

Important Notes

- Ensure MySQL driver is in classpath (mysql-connector-java.jar).

- Database Shop must exist with table Customer:

```
CREATE TABLE Customer (
    cid INT PRIMARY KEY,
    name VARCHAR(50),
    address VARCHAR(100),
    email VARCHAR(50),
    age INT
);
```

10.Explain the lifecycle of Java Servlet. Write a servlet program that takes three numbers from an HTML page and computes the sum of natural numbers up to the greatest number among these numbers.

Lifecycle of a Java Servlet

A Java Servlet is a server-side program that handles HTTP requests and responses.

Its lifecycle is managed by the Servlet Container (like Tomcat).

Servlet Lifecycle Methods

| Phase | Method | Description |
|-------------------------|--|--|
| Loading & Instantiation | Servlet class is loaded by the container and an instance is created. | Done once per servlet. |
| Initialization | init(ServletConfig config) | Called once after instantiation to initialize resources (DB connections, variables). |
| Request Handling | service(HttpServletRequest req, HttpServletResponse res) | Called for every client request. Delegates to doGet() or doPost(). |
| Destruction | destroy() | Called once before servlet is removed. Used to release resources. |

Flow Diagram:

Client Request → Servlet Container → Servlet instance created → init() → service() → doGet()/doPost() → Response → destroy()

HTML Form to Take Three Numbers

```
<!DOCTYPE html>
<html>
<head>
    <title>Number Input</title>
</head>
<body>
    <h2>Enter Three Numbers</h2>
    <form action="SumServlet" method="post">
        Number 1: <input type="number" name="num1" required><br><br>
        Number 2: <input type="number" name="num2" required><br><br>
        Number 3: <input type="number" name="num3" required><br><br>
```

```
<input type="submit" value="Compute Sum">
</form>
</body>
</html>

Servlet Program: Sum of Natural Numbers up to the Greatest Number
import java.io.IOException;
import java.io.PrintWriter;
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

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

    // Lifecycle method: init()
    public void init() throws ServletException {
        System.out.println("Servlet Initialized");
    }

    // Handles POST requests
    protected void doPost(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {

        response.setContentType("text/html");
        PrintWriter out = response.getWriter();

        try {
            // Get numbers from HTML form
            int num1 = Integer.parseInt(request.getParameter("num1"));
            int num2 = Integer.parseInt(request.getParameter("num2"));
            int num3 = Integer.parseInt(request.getParameter("num3"));

            // Find greatest number
            int greatest = Math.max(num1, Math.max(num2, num3));

            // Compute sum of natural numbers up to greatest
            int sum = (greatest * (greatest + 1)) / 2;

            // Display result
            out.println("<html><body>");
        }
    }
}
```

```

        out.println("<h2>Numbers Entered: " + num1 + ", " + num2 + ", " + num3 +
    "</h2>");
        out.println("<h2>Greatest Number: " + greatest + "</h2>");
        out.println("<h2>Sum of natural numbers up to " + greatest + " = " + sum + "</h2>");
        out.println("</body></html>");

    } catch (Exception e) {
        out.println("Error: " + e.getMessage());
    }
}

// Lifecycle method: destroy()
public void destroy() {
    System.out.println("Servlet Destroyed");
}
}

```

Explanation of the Servlet

1. `@WebServlet("/SumServlet")`
 - o Maps the servlet to the URL pattern /SumServlet.
2. `init()` method
 - o Called once when servlet is loaded. Useful for initializing resources.
3. `doPost()` method
 - o Handles the form submission.
 - o Reads three numbers from `HttpServletRequest`.
 - o Finds the greatest number.
 - o Computes sum of natural numbers up to that number using formula:

$$\text{Sum} = \frac{n \cdot (n + 1)}{2}$$

4. `destroy()` method
 - o Called before the servlet is removed.
 - o Use it to release resources.
5. Response Handling
 - o Sends HTML content back to the browser using `PrintWriter`.

5. How to Run

1. Put HTML file in `webapps/YourAppName/` directory.
2. Compile and deploy servlet in Tomcat under `WEB-INF/classes`.
3. Access the HTML page → Submit → Servlet calculates sum → Response displayed.

11.What is directive element in JSP? Explain in detail. Create a JSP page that verifies the username and password entered from an HTML file and if the username and password are correct it displays greetings message. Otherwise redirects to register page.

Directive Element in JSP

Definition:

A directive in JSP is a special instruction to the JSP container that provides global information about the JSP page.

- It does not produce output directly to the client.
- Directives affect the overall structure of the servlet generated from the JSP.

Syntax:

```
<%@ directive attribute="value" %>
```

Types of JSP Directives:

| Directive | Purpose |
|-----------|---|
| page | Defines page-dependent attributes such as language, content type, error page, buffer size, etc. |
| include | Includes a static file (like header/footer) at translation time. |
| taglib | Declares a tag library for custom tags. |

Page Directive Example

```
<%@ page language="java" contentType="text/html; charset=UTF-8" %>
```

```
<%@ page import="java.util.*" %>
```

Attributes of page directive:

| Attribute | Description |
|-------------|---|
| language | Programming language used in JSP (usually Java) |
| contentType | MIME type sent to client |
| import | Import Java packages |
| session | Whether the page participates in session (true/false) |
| errorPage | JSP page to redirect to if exception occurs |
| isErrorPage | Declares a page as error page |

Include Directive Example

```
<%@ include file="header.jsp" %>
```

- Inserts content of header.jsp at translation time.
- Useful for common headers or footers.

Taglib Directive Example

```
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
```

- Declares a tag library with prefix c.
- Useful for using JSTL (JSP Standard Tag Library).

JSP Page to Verify Username and Password

HTML Form

```
<!DOCTYPE html>
<html>
<head>
    <title>Login Page</title>
</head>
<body>
```

```

<h2>Login</h2>
<form action="Login.jsp" method="post">
    Username: <input type="text" name="username" required><br><br>
    Password: <input type="password" name="password" required><br><br>
    <input type="submit" value="Login">
</form>
</body>
</html>

JSP Page: Login.jsp
<%@ page language="java" contentType="text/html; charset=UTF-8" %>
<%@ page import="java.io.*" %>
<%
    // Get username and password from HTML form
    String username = request.getParameter("username");
    String password = request.getParameter("password");

    // Define correct username and password
    String correctUsername = "admin";
    String correctPassword = "12345";

    if(username != null && password != null) {
        if(username.equals(correctUsername) && password.equals(correctPassword)) {
            // Login successful
        } else {
            // Login failed, redirect to register page
            response.sendRedirect("register.html");
        }
    } else {
        // If accessed directly without form
        response.sendRedirect("login.html");
    }
%>

```

Explanation of JSP Code

1. Directive Elements Used:

```
<%@ page language="java" contentType="text/html; charset=UTF-8" %>
<%@ page import="java.io.*" %>
```

- language="java" → Specifies Java as scripting language.
- contentType="text/html" → Sends response as HTML.
- import="java.io.*" → Imports Java classes for use in JSP.

2. Form Data Retrieval:

```
String username = request.getParameter("username");
String password = request.getParameter("password");
    • request.getParameter() fetches input from HTML form.
```

3. Validation Logic:

- Compare user input with predefined values.
- If valid → Show greeting message.
- If invalid → Redirect to register.html using:

```
response.sendRedirect("register.html");
```

Question (2021)

2. Why do we need a top-level container like JFrame to write Java programs with GUI? How can we display two-dimensional objects in java?

Why do we need a top-level container like JFrame in Java GUI programs?

Meaning of Top-Level Container

A top-level container is a GUI component that:

- Represents a window on the screen
- Can exist independently
- Acts as the base container for all other GUI components

Examples of top-level containers:

- JFrame
- JDialog
- JWindow
- Applet (old)

Why JFrame is Needed

1. Provides a Window

- GUI components like JButton, JLabel, JTextField cannot be displayed alone
- They must be added to a container
- JFrame provides the main application window

```
JFrame frame = new JFrame("My Application");
```

2. Holds Other Components

- All GUI components are placed inside the JFrame
- Uses a content pane to manage components

```
frame.add(new JButton("Click Me"));
```

3. Supports Layout Managers

- JFrame manages how components are arranged using layout managers
- Example: FlowLayout, BorderLayout, GridLayout

4. Handles Window Events

- Closing, minimizing, maximizing the window
- Example:

```
frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
```

5. Platform-Independent

- JFrame is part of Swing, which is platform-independent
- Looks consistent across Windows, Linux, macOS

How can we display two-dimensional objects in Java?

Java displays 2D objects using the Graphics and Graphics2D classes from the java.awt package.

Graphics vs Graphics2D

| Graphics | Graphics2D |
|-----------------|--------------------------------------|
| Basic drawing | Advanced 2D drawing |
| Limited control | Supports shapes, transforms, strokes |
| Older API | More powerful and flexible |

Key Classes for 2D Graphics

- Graphics
- Graphics2D
- Color
- BasicStroke
- Shape (Rectangle2D, Ellipse2D, Line2D)

Steps to Display 2D Objects

Step 1: Extend JPanel

```
class MyPanel extends JPanel {
```

Step 2: Override paintComponent()

```
protected void paintComponent(Graphics g) {  
    super.paintComponent(g);  
}
```

Step 3: Cast Graphics to Graphics2D

```
Graphics2D g2 = (Graphics2D) g;
```

Step 4: Draw 2D Objects

Example: Drawing 2D Shapes

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

```
public class TwoDGraphicsExample extends JFrame {
```

```
public TwoDGraphicsExample() {  
    setTitle("2D Graphics Example");  
    setSize(400, 300);
```

```

        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        add(new DrawPanel());
        setVisible(true);
    }

    class DrawPanel extends JPanel {
        protected void paintComponent(Graphics g) {
            super.paintComponent(g);
            Graphics2D g2 = (Graphics2D) g;

            // Draw Line
            g2.drawLine(50, 50, 200, 50);

            // Draw Rectangle
            g2.draw(new Rectangle2D.Double(50, 80, 150, 60));

            // Draw Circle
            g2.draw(new Ellipse2D.Double(50, 160, 100, 100));

            // Fill Rectangle
            g2.setColor(Color.BLUE);
            g2.fillRect(220, 80, 100, 60);
        }
    }

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

```

Common 2D Drawing Methods

| Method | Purpose |
|------------|--------------------|
| drawLine() | Draws a line |
| drawRect() | Draws a rectangle |
| fillRect() | Fills rectangle |
| drawOval() | Draws circle/oval |
| fillOval() | Fills circle |
| setColor() | Sets drawing color |

3.Explain JTextField and JTextArea components of the Java Swing library.

1. JTextField

Definition

JTextField is a Swing component used to input or display a single line of text. It is commonly used for short text inputs like name, username, email, etc.

Package

javax.swing.JTextField

Features of JTextField

- Allows single-line text input
- Editable or non-editable
- Supports text selection and copy/paste
- Can trigger events when Enter key is pressed

Constructors

JTextField()

JTextField(int columns)

JTextField(String text)

JTextField(String text, int columns)

Common Methods

| Method | Description |
|----------------------|--------------------------|
| setText(String) | Sets text in field |
| getText() | Gets entered text |
| setEditable(boolean) | Enables/disables editing |
| setColumns(int) | Sets width |
| addActionListener() | Handles Enter key |

Example of JTextField

```
import javax.swing.*;  
  
public class JTextFieldExample {  
    public static void main(String[] args) {  
        JFrame frame = new JFrame("JTextField Example");  
        JTextField tf = new JTextField(20);  
  
        frame.add(tf);  
        frame.setSize(300, 100);  
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);  
        frame.setVisible(true);  
    }  
}
```

2. JTextArea

Definition

JTextArea is a Swing component used to input or display multiple lines of text. It is ideal for addresses, comments, descriptions, etc.

Package

javax.swing.JTextArea

Features of JTextArea

- Supports multiple lines
- Allows line wrapping and word wrapping
- Can be placed inside a JScrollPane for scrolling
- Editable or read-only

Constructors

JTextArea()

JTextArea(int rows, int columns)

JTextArea(String text)

JTextArea(String text, int rows, int columns)

Common Methods

| Method | Description |
|------------------------|------------------------|
| setText(String) | Sets text |
| getText() | Gets text |
| setLineWrap(true) | Enables line wrapping |
| setWrapStyleWord(true) | Wraps by word |
| setEditable(boolean) | Enable/disable editing |

Example of JTextArea

```
import javax.swing.*;  
  
public class JTextAreaExample {  
    public static void main(String[] args) {  
        JFrame frame = new JFrame("JTextArea Example");  
  
        JTextArea ta = new JTextArea(5, 20);  
        ta.setLineWrap(true);  
        ta.setWrapStyleWord(true);  
  
        JScrollPane scroll = new JScrollPane(ta);  
  
        frame.add(scroll);  
        frame.setSize(300, 200);  
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);  
        frame.setVisible(true);  
    }  
}
```

3. Difference Between JTextField and JTextArea

| Feature | JTextField | JTextArea |
|------------|-----------------------|-------------------|
| Text lines | Single-line | Multi-line |
| Use case | Name, email, username | Address, comments |

| | | |
|----------------|-----------------|------------------------|
| Scroll support | No | Yes (with JScrollPane) |
| Enter key | Triggers action | Moves to next line |
| Line wrap | Not supported | Supported |

4. Real-Life Usage Example

- JTextField → Username, Email, Age
- JTextArea → Address, Feedback, Description

4. How do you execute SQL statements using JDBC? Explain with an example.

How do we execute SQL statements using JDBC?

In JDBC, SQL statements are executed using statement objects created from a Connection object.

The main interfaces used are:

1. Statement
2. PreparedStatement
3. CallableStatement

Steps to Execute SQL Statements Using JDBC

Step 1: Load JDBC Driver

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

Step 2: Establish Connection

```
Connection con = DriverManager.getConnection(
```

```
    "jdbc:mysql://localhost:3306/shop", "root", "password");
```

Step 3: Create Statement Object

| Type | Used For |
|-------------------|-------------------|
| Statement | Simple static SQL |
| PreparedStatement | Parameterized SQL |
| CallableStatement | Stored procedures |

Step 4: Execute SQL Statement

| Method | Purpose |
|-----------------|------------------------|
| executeQuery() | SELECT |
| executeUpdate() | INSERT, UPDATE, DELETE |
| execute() | Any SQL |

Step 5: Process Result

- Use ResultSet for SELECT
- Check affected rows for INSERT/UPDATE/DELETE

Step 6: Close Resources

```
rs.close();
stmt.close();
con.close();
```

Example: Executing SQL Using JDBC

Example 1: INSERT Data using PreparedStatement

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

```
public class InsertExample {  
    public static void main(String[] args) {  
        try {  
            // Step 1: Load driver  
            Class.forName("com.mysql.cj.jdbc.Driver");  
  
            // Step 2: Create connection  
            Connection con = DriverManager.getConnection(  
                "jdbc:mysql://localhost:3306/shop", "root", "password");  
  
            // Step 3: Prepare SQL statement  
            String sql = "INSERT INTO Customer(cid, name, age) VALUES (?, ?, ?);";  
            PreparedStatement pst = con.prepareStatement(sql);  
  
            // Step 4: Set values  
            pst.setInt(1, 101);  
            pst.setString(2, "Bibash");  
            pst.setInt(3, 22);  
  
            // Step 5: Execute SQL  
            int rows = pst.executeUpdate();  
  
            System.out.println(rows + " record inserted");  
  
            // Step 6: Close connection  
            pst.close();  
            con.close();  
  
        } catch (Exception e) {  
            e.printStackTrace();  
        }  
    }  
}
```

Example 2: SELECT Data using Statement

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

```
public class SelectExample {
```

```

public static void main(String[] args) {
    try {
        Connection con = DriverManager.getConnection(
            "jdbc:mysql://localhost:3306/shop", "root", "password");

        Statement stmt = con.createStatement();

        ResultSet rs = stmt.executeQuery("SELECT * FROM Customer");

        while (rs.next()) {
            System.out.println(
                rs.getInt("cid") + " " +
                rs.getString("name") + " " +
                rs.getInt("age")
            );
        }

        rs.close();
        stmt.close();
        con.close();
    } catch (Exception e) {
        e.printStackTrace();
    }
}

```

Methods Used to Execute SQL Statements

1. executeQuery()

- Used for SELECT
- Returns ResultSet

```
ResultSet rs = stmt.executeQuery("SELECT * FROM Customer");
```

2. executeUpdate()

- Used for INSERT, UPDATE, DELETE
- Returns number of affected rows

```
int rows = stmt.executeUpdate("DELETE FROM Customer WHERE cid=101");
```

3. execute()

- Used when SQL type is unknown
- Returns true or false

5.Compare result set with row set. Explain a prepared statement with an example.

Compare ResultSet and RowSet

ResultSet

Definition:

ResultSet is an interface in JDBC that represents a table of data returned by executing a SQL SELECT query.

It is connected to the database.

RowSet

Definition:

RowSet is a JavaBean-style wrapper around ResultSet.

It can work in a connected or disconnected mode and is more flexible.

Differences Between ResultSet and RowSet

| Feature | ResultSet | RowSet |
|----------------|--------------------------|----------------------------------|
| Connection | Connected to database | Can be connected or disconnected |
| Serializable | No | Yes |
| JavaBean | No | Yes |
| Event handling | Not supported | Supports events |
| Usability | Low-level, less flexible | Easier and more flexible |
| Use case | Simple data retrieval | Distributed & disconnected apps |

Types of RowSet

- JdbcRowSet (connected)
- CachedRowSet (disconnected)
- WebRowSet
- FilteredRowSet
- JoinRowSet

Example Difference

```
// ResultSet (connected)
Statement stmt = con.createStatement();
ResultSet rs = stmt.executeQuery("SELECT * FROM Customer");
```

```
// RowSet (disconnected)
```

```
CachedRowSet crs = RowSetProvider.newFactory().createCachedRowSet();
crs.populate(rs);
```

PreparedStatement

What is PreparedStatement?

PreparedStatement is an interface used to execute precompiled, parameterized SQL queries. It improves performance, security, and code readability.

Advantages of PreparedStatement

- Prevents SQL Injection
- Faster execution for repeated queries
- Supports dynamic parameters
- Cleaner and safer code

Syntax

```
PreparedStatement pst = con.prepareStatement(  
    "INSERT INTO Customer(cid, name, age) VALUES (?, ?, ?)");  
Example: PreparedStatement
```

Insert Data using PreparedStatement

```
import java.sql.*;  
  
public class PreparedStatementExample {  
    public static void main(String[] args) {  
        try {  
            // Step 1: Create connection  
            Connection con = DriverManager.getConnection(  
                "jdbc:mysql://localhost:3306/shop", "root", "password");  
  
            // Step 2: Prepare SQL  
            String sql = "INSERT INTO Customer(cid, name, age) VALUES (?, ?, ?)";  
            PreparedStatement pst = con.prepareStatement(sql);  
  
            // Step 3: Set values  
            pst.setInt(1, 201);  
            pst.setString(2, "Bibash");  
            pst.setInt(3, 22);  
  
            // Step 4: Execute query  
            int rows = pst.executeUpdate();  
            System.out.println(rows + " record inserted");  
  
            // Step 5: Close connection  
            pst.close();  
            con.close();  
  
        } catch (Exception e) {  
            e.printStackTrace();  
        }  
    }  
}
```

6.How do you set and get Cookie in Servlet? Explain using a suitable Java program.

Cookies in Servlet

What is a Cookie?

A Cookie is a small piece of data stored on the client's browser by the server.

It is used to maintain state between multiple HTTP requests.

How to Set a Cookie in Servlet

Steps

1. Create a Cookie object
2. Set cookie properties (optional)
3. Add cookie to response

Example: Setting a Cookie

```
import java.io.*;  
import javax.servlet.*;  
import javax.servlet.http.*;  
  
public class SetCookieServlet extends HttpServlet {  
    protected void doGet(HttpServletRequest request, HttpServletResponse response)  
        throws ServletException, IOException {  
  
        response.setContentType("text/html");  
  
        // Create cookie  
        Cookie c = new Cookie("username", "Bibash");  
  
        // Set expiry time (in seconds)  
        c.setMaxAge(60 * 60); // 1 hour  
  
        // Add cookie to response  
        response.addCookie(c);  
  
        PrintWriter out = response.getWriter();  
        out.println("<h3>Cookie has been set</h3>");  
    }  
}
```

How to Get a Cookie in Servlet

Steps

1. Get all cookies from request
2. Loop through cookies
3. Match cookie name and retrieve value

Example: Getting a Cookie

```
import java.io.*;  
import javax.servlet.*;  
import javax.servlet.http.*;  
  
public class GetCookieServlet extends HttpServlet {  
    protected void doGet(HttpServletRequest request, HttpServletResponse response)  
        throws ServletException, IOException {  
    }
```

```

response.setContentType("text/html");
PrintWriter out = response.getWriter();

Cookie[] cookies = request.getCookies();

if (cookies != null) {
    for (Cookie c : cookies) {
        if (c.getName().equals("username")) {
            out.println("<h3>Welcome " + c.getValue() + "</h3>");
        }
    }
} else {
    out.println("<h3>No cookies found</h3>");
}
}
}
}

```

Important Cookie Methods

| Method | Description |
|-----------------------------------|----------------------------|
| Cookie(String name, String value) | Creates cookie |
| setMaxAge(int seconds) | Sets expiry time |
| getName() | Returns cookie name |
| getValue() | Returns cookie value |
| addCookie() | Sends cookie to client |
| getCookies() | Reads cookies from request |

Advantages of Cookies

- Simple to use
- Stored on client side
- Maintains user information (login, preferences)

Limitations of Cookies

- Limited size (~4 KB)
- Less secure
- Client can disable cookies

7.Explain different scripting elements of JSP with examples.

Scripting Elements of JSP

JSP scripting elements are used to embed Java code inside a JSP page.

They allow dynamic content generation and interaction with server-side logic.

There are three types of JSP scripting elements:

1. Scriptlet Tag
2. Expression Tag
3. Declaration Tag

1. Scriptlet Tag

Syntax

```
<% Java code %>
```

Use

- Used to write Java statements
- Code is executed every time the JSP page is requested
- Can include loops, conditions, method calls, etc.

Example

```
<html>
<body>
<%
    int a = 10;
    int b = 20;
    int sum = a + b;
    out.println("Sum = " + sum);
%>
</body>
</html>
```

Output:

Sum = 30

2. Expression Tag

Syntax

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

Use

- Used to display the result of an expression
- Automatically prints output (no need for out.println)
- Does not end with semicolon

Example

```
<html>
<body>
<%
    int x = 5;
    int y = 7;
%>
```

Sum is: <%= x + y %>

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

Output:

Sum is: 12

3. Declaration Tag

Syntax

```
<%! declaration %>
```

Use

- Used to declare variables and methods
- Declared items are available throughout the JSP page
- Executed once, not on every request

Example

```
<html>
<body>
<%!
    int square(int n) {
        return n * n;
    }
%>
```

Square of 6 is: <%= square(6) %>

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

Output:

Square of 6 is: 36

Comparison of JSP Scripting Elements

| Element | Syntax | Purpose |
|-------------|--------|---------------------------|
| Scriptlet | <% %> | Write Java statements |
| Expression | <%= %> | Display output |
| Declaration | <%! %> | Declare variables/methods |

8. Write short notes on:

a) Java Web Frameworks

Definition

Java web frameworks are software libraries that simplify the development of web applications by providing a structured way to build, deploy, and maintain Java-based web apps.

Uses

- Reduces boilerplate code
- Follows MVC architecture
- Improves security and scalability
- Speeds up development

Examples of Java Web Frameworks

- Spring MVC / Spring Boot
- Hibernate
- Struts
- JSF (JavaServer Faces)

Advantages

- Better code organization
- Easy database integration
- Built-in validation and security

b) CORBA (Common Object Request Broker Architecture)

Definition

CORBA is a distributed object-oriented architecture that allows programs written in different programming languages and running on different platforms to communicate with each other.

Key Components

- ORB (Object Request Broker) – handles communication
- IDL (Interface Definition Language) – defines object interfaces
- Stub and Skeleton – enable client-server communication

Uses

- Language-independent communication
- Distributed systems development

Advantages

- Platform independent
- Supports multiple languages (Java, C++, Python)

c) Bean Bound Property

Definition

A bound property in JavaBeans is a property that notifies registered listeners whenever its value changes.

How it Works

- Uses PropertyChangeListener
- Fires PropertyChangeEvent when value changes

Example

```
import java.beans.*;  
  
public class StudentBean {  
    private int age;  
    private PropertyChangeSupport pcs = new PropertyChangeSupport(this);  
  
    public int getAge() {  
        return age;  
    }  
  
    public void setAge(int newAge) {  
        int oldAge = this.age;  
        this.age = newAge;  
        pcs.firePropertyChange("age", oldAge, newAge);  
    }  
}
```

```
    public void addPropertyChangeListener(PropertyChangeListener listener) {  
        pcs.addPropertyChangeListener(listener);  
    }  
}
```

Use

- Used in GUI applications
- Helps in event-driven programming

9. Write a GUI application to find the sum and difference of two integer numbers. Use two integer numbers. Use two text fields for input and a third text field for output. Your program should display the sum if the user presses the mouse and the difference if the user releases the mouse.

Program Requirement Recap

- GUI application
- Two TextFields → input numbers
- Third TextField → output
- Mouse Pressed → display sum
- Mouse Released → display difference

We'll use:

- AWT
- MouseListener

Java GUI Program (AWT + Mouse Events)

```
import java.awt.*;  
import java.awt.event.*;  
  
public class SumDifferenceMouse extends Frame implements MouseListener {  
  
    TextField t1, t2, t3;  
    Label l1, l2, l3;  
  
    public SumDifferenceMouse() {  
        // Labels  
        l1 = new Label("First Number:");  
        l2 = new Label("Second Number:");  
        l3 = new Label("Result:");  
  
        // TextFields  
        t1 = new TextField();  
        t2 = new TextField();  
        t3 = new TextField();  
        t3.setEditable(false);
```

```
// Layout
setLayout(new GridLayout(3, 2));

add(l1); add(t1);
add(l2); add(t2);
add(l3); add(t3);

// Mouse listener
addMouseListener(this);

// Frame settings
setTitle("Sum and Difference using Mouse Events");
setSize(350, 150);
setVisible(true);

// Close window
addWindowListener(new WindowAdapter() {
    public void windowClosing(WindowEvent e) {
        dispose();
    }
});

// Mouse pressed → SUM
public void mousePressed(MouseEvent e) {
    int a = Integer.parseInt(t1.getText());
    int b = Integer.parseInt(t2.getText());
    int sum = a + b;
    t3.setText("Sum = " + sum);
}

// Mouse released → DIFFERENCE
public void mouseReleased(MouseEvent e) {
    int a = Integer.parseInt(t1.getText());
    int b = Integer.parseInt(t2.getText());
    int diff = a - b;
    t3.setText("Difference = " + diff);
}

// Unused methods
public void mouseClicked(MouseEvent e) {}
public void mouseEntered(MouseEvent e) {}
```

```

public void mouseExited(MouseEvent e) {}

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

```

10.What is a bean design pattern? Explain simple, Boolean, and indexed property design patterns with suitable examples in detail.

Bean Design Pattern

What is a Bean Design Pattern?

A Bean design pattern is a set of naming conventions and rules used in JavaBeans to define properties, methods, and events so that tools (IDE, JSP, frameworks) can automatically recognize and use them.

Types of JavaBean Property Design Patterns

JavaBeans properties are classified into:

1. Simple Property
2. Boolean Property
3. Indexed Property

1. Simple Property Design Pattern

Definition

A simple property represents a single value and has:

- One getter
- One setter

Naming Convention

```

public Type getPropertyType();
public void setPropertyType(Type value);

```

Example: Simple Property

```

public class StudentBean {

```

```

    private String name;

```

```

    // Getter

```

```

    public String getName() {
        return name;
    }

```

```

    // Setter

```

```

    public void setName(String name) {
        this.name = name;
    }

```

```
}
```

2. Boolean Property Design Pattern

Definition

A Boolean property represents a true/false value.

Naming Convention

```
public boolean isPropertyName();  
public void setPropertyName(boolean value);
```

Example: Boolean Property

```
public class AccountBean {  
    private boolean active;
```

```
// Getter
```

```
    public boolean isActive() {  
        return active;  
    }
```

```
// Setter
```

```
    public void setActive(boolean active) {  
        this.active = active;  
    }
```

3. Indexed Property Design Pattern

Definition

An indexed property represents a collection of values, usually an array or list.

It has:

- Indexed getter and setter
- Optional array-level getter and setter

Naming Convention

```
public Type getPropertyName(int index);  
public void setPropertyName(int index, Type value);
```

```
public Type[] getPropertyName();  
public void setPropertyName(Type[] values);
```

Example: Indexed Property

```
public class MarksBean {  
    private int[] marks = new int[5];
```

```
// Indexed getter
```

```

public int getMarks(int index) {
    return marks[index];
}

// Indexed setter
public void setMarks(int index, int value) {
    marks[index] = value;
}

// Array getter
public int[] getMarks() {
    return marks;
}

// Array setter
public void setMarks(int[] marks) {
    this.marks = marks;
}
}

```

Why Bean Design Pattern is Important

- Enables automatic introspection
- Used in JSP, JSF, Spring
- Improves reusability and maintainability
- Supports tool-based development

11. Define RMI. What is stub and parameter marshalling? Write a client/server application using RMI to find the product of two numbers.

RMI (Remote Method Invocation)

Definition of RMI

RMI (Remote Method Invocation) is a Java mechanism that allows an object running in one JVM to invoke methods of an object running in another JVM (possibly on a different machine).

What is a Stub?

- A stub is a client-side proxy object
- It represents the remote object
- It forwards method calls from the client to the remote server

Role of Stub

- Receives method call from client
- Performs parameter marshalling
- Sends request to server via RMI runtime

What is Parameter Marshalling?

Definition

Parameter marshalling is the process of converting method parameters and return values into a stream of bytes so they can be transmitted over the network.

In RMI

- Arguments are marshalled by the stub
- Results are unmarshalled by the client
- Java uses serialization for marshalling

RMI Application: Find Product of Two Numbers

Step 1: Remote Interface

```
import java.rmi.*;
```

```
public interface Multiply extends Remote {  
    int product(int a, int b) throws RemoteException;  
}
```

Step 2: Remote Implementation (Server)

```
import java.rmi.*;  
import java.rmi.server.*;
```

```
public class MultiplyImpl extends UnicastRemoteObject implements Multiply {
```

```
    public MultiplyImpl() throws RemoteException {  
        super();  
    }
```

```
    public int product(int a, int b) throws RemoteException {  
        return a * b;  
    }  
}
```

Step 3: RMI Server Program

```
import java.rmi.*;  
import java.rmi.registry.*;
```

```
public class RMIServer {  
    public static void main(String[] args) {  
        try {  
            Multiply obj = new MultiplyImpl();
```

```
            Registry reg = LocateRegistry.createRegistry(1099);  
            reg.rebind("MultiplyService", obj);
```

```
            System.out.println("RMI Server is running...");
```

```

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

Step 4: RMI Client Program
import java.rmi.*;
import java.rmi.registry.*;

public class RMIClient {
    public static void main(String[] args) {
        try {
            Registry reg = LocateRegistry.getRegistry("localhost", 1099);
            Multiply obj = (Multiply) reg.lookup("MultiplyService");

            int result = obj.product(5, 6);
            System.out.println("Product = " + result);

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

```

Advantages of RMI

- Platform independent
- Object-oriented communication
- Built-in security and serialization

Questions (2020)

2. Why do we need top level container like JFrame to write Java programs with GUI?

How can we display two dimensional objects in Java?

Why Do We Need a Top-Level Container Like JFrame in Java GUI?

Definition:

A top-level container in Java Swing is a container that provides a window on the screen where you can add GUI components like buttons, text fields, labels, etc.

Examples of Top-Level Containers:

- JFrame – a window with a title, borders, and buttons like minimize/maximize/close.
- JDialog – a pop-up window for dialogs.
- JApplet – used for applets (older Java).

Reasons for Using a Top-Level Container (JFrame):

1. Provides a Window for Components

- You cannot display buttons, labels, text fields directly on the screen without a container.
 - JFrame serves as the main window where all components are added.
2. Manages Window Properties
 - Title, size, visibility, layout, icon, closing behavior, etc.
 3. Acts as a Root Container
 - All other Swing components like JPanel, JButton, JTextField are added to it.
 4. Supports Event Handling
 - You can attach listeners (like ActionListener, MouseListener) to JFrame or its components.

Example of JFrame Usage:

```
import javax.swing.*;

public class JFrameExample {
    public static void main(String[] args) {
        // Create JFrame
        JFrame frame = new JFrame("My First GUI");
        frame.setSize(400, 300);          // Set window size
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE); // Close operation

        // Add a label
        JLabel label = new JLabel("Hello, Swing!", JLabel.CENTER);
        frame.add(label);              // Add label to JFrame

        // Make the frame visible
        frame.setVisible(true);
    }
}
```

How Can We Display Two-Dimensional Objects in Java?

2D graphics in Java are handled using the java.awt.Graphics and java.awt.Graphics2D classes.

Steps to Display 2D Objects:

1. Extend a Component (like JPanel or Canvas)
2. Override the paintComponent(Graphics g) method
3. Use Graphics methods to draw shapes

Common 2D Drawing Methods:

| Method | Description |
|-------------------------------|---|
| drawLine(x1, y1, x2, y2) | Draw a straight line between two points |
| drawRect(x, y, width, height) | Draw a rectangle outline |
| fillRect(x, y, width, height) | Draw a filled rectangle |
| drawOval(x, y, width, height) | Draw oval outline |
| fillOval(x, y, width, height) | Draw filled oval |

| | |
|--|----------------------------|
| drawString("text", x, y) | Draw text on the panel |
| drawPolygon(int[] x, int[] y, nPoints) | Draw polygon with n points |

Example: Drawing 2D Shapes in Java Swing

```

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

public class Draw2DExample extends JPanel {

    @Override
    protected void paintComponent(Graphics g) {
        super.paintComponent(g);

        // Cast to Graphics2D for more features
        Graphics2D g2 = (Graphics2D) g;

        // Draw a line
        g2.drawLine(50, 50, 200, 50);

        // Draw rectangle
        g2.drawRect(50, 70, 150, 100);

        // Fill rectangle
        g2.setColor(Color.BLUE);
        g2.fillRect(220, 70, 150, 100);

        // Draw oval
        g2.setColor(Color.RED);
        g2.drawOval(50, 200, 150, 100);

        // Draw text
        g2.setColor(Color.BLACK);
        g2.drawString("Hello 2D Graphics!", 50, 350);
    }

    public static void main(String[] args) {
        JFrame frame = new JFrame("2D Drawing Example");
        frame.setSize(500, 500);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.add(new Draw2DExample());
        frame.setVisible(true);
    }
}

```


3.Explain JTextField and JTextArea components of Java swing library.

1. JTextField

Definition:

- JTextField is a single-line text component in Java Swing used to allow users to input text.
- It can also be used to display text.

Key Features:

1. Single-line input only.
2. Can specify columns (width) for display.
3. Supports event handling (ActionListener) to respond to user actions (like pressing Enter).
4. Can set editable or non-editable text.

Constructor Examples:

```
JTextField tf1 = new JTextField();      // empty field  
JTextField tf2 = new JTextField("Hello"); // pre-filled text  
JTextField tf3 = new JTextField(20);     // 20 columns width
```

Common Methods:

| Method | Description |
|-------------------------------------|---------------------------------------|
| getText() | Returns the text entered in the field |
| setText(String s) | Sets text in the field |
| setEditable(boolean b) | Make field editable or read-only |
| addActionListener(ActionListener a) | Respond to Enter key press |

Example: JTextField

```
import javax.swing.*;  
import java.awt.event.*;  
  
public class JTextFieldExample {  
    public static void main(String[] args) {  
        JFrame frame = new JFrame("JTextField Example");  
        JTextField tf = new JTextField(20);  
        JButton btn = new JButton("Show Text");  
  
        btn.addActionListener(e -> {  
            String text = tf.getText();  
            JOptionPane.showMessageDialog(frame, "You entered: " + text);  
        });  
  
        frame.setLayout(new FlowLayout());  
        frame.add(tf);  
        frame.add(btn);  
        frame.setSize(300, 100);  
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);  
    }  
}
```

```

        frame.setVisible(true);
    }
}

```

2. JTextArea

Definition:

- JTextArea is a multi-line text component in Java Swing.
- Used to display or input multiple lines of text, like a text editor.

Key Features:

1. Supports multiple lines of text.
2. Can set rows and columns.
3. Supports line wrapping.
4. Can add scrollbars using JScrollPane.

Constructor Examples:

```

JTextArea ta1 = new JTextArea();           // empty
JTextArea ta2 = new JTextArea(5, 20);       // 5 rows, 20 columns
JTextArea ta3 = new JTextArea("Hello\nWorld"); // pre-filled text

```

Common Methods:

| Method | Description |
|-----------------------------|------------------------------|
| getText() | Returns all text |
| setText(String s) | Sets text |
| append(String s) | Adds text at the end |
| setLineWrap(boolean b) | Enable/disable line wrapping |
| setWrapStyleWord(boolean b) | Wrap at word boundaries |
| | |

Example: JTextArea

```

import javax.swing.*;

public class JTextAreaExample {
    public static void main(String[] args) {
        JFrame frame = new JFrame("JTextArea Example");
        JTextArea ta = new JTextArea(5, 30); // 5 rows, 30 columns
        ta.setLineWrap(true);
        ta.setWrapStyleWord(true);

        JScrollPane sp = new JScrollPane(ta); // add scrollbar

        frame.add(sp);
        frame.setSize(400, 200);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setVisible(true);
    }
}

```

3. Difference Between JTextField and JTextArea

| Feature | JTextField | JTextArea |
|---------------|---|---|
| Lines | Single-line | Multi-line |
| Scrollbars | Not needed | Can use JScrollPane |
| Line Wrapping | Not supported | Supported via setLineWrap(true) |
| Typical Use | Input username, password, single values | Input comments, messages, multi-line text |
| Constructors | JTextField(String s, int columns) | JTextArea(String s, int rows, int cols) |

4. How do you execute SQL statements using JDBC? Explain with example.

JDBC Overview

JDBC (Java Database Connectivity) is an API that allows Java programs to interact with databases.

It provides classes and interfaces to:

- Connect to a database.
- Execute SQL queries and updates.
- Retrieve results.

Steps to Execute SQL Statements Using JDBC

Step 1: Load the JDBC Driver

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

- Loads the MySQL driver class dynamically.
- For newer JDBC versions, this step is optional.

Step 2: Establish a Connection

```
Connection con = DriverManager.getConnection(
```

- ```
"jdbc:mysql://localhost:3306/shop", "root", "password");
```
- Connection object represents a connection to the database.
  - Parameters: database URL, username, password.

##### Step 3: Create a Statement

There are three main types of statements:

1. Statement – For simple SQL queries.
2. PreparedStatement – Precompiled SQL with parameters.
3. CallableStatement – For calling stored procedures.

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

##### Step 4: Execute SQL Statement

Depending on the type of SQL:

1. SELECT Query → executeQuery()

```
ResultSet rs = stmt.executeQuery("SELECT * FROM Customer");
```

- Returns a ResultSet containing query results.

2. INSERT, UPDATE, DELETE → executeUpdate()

```
int rows = stmt.executeUpdate("INSERT INTO Customer VALUES (1, 'John', 'NY',
'john@example.com', 25)");
```

- Returns the number of affected rows.

3. Any SQL → execute()

```
boolean result = stmt.execute("CREATE TABLE Test(id INT)");
```

- Returns true if query produces a ResultSet.

Step 5: Process the Result

```
while(rs.next()) {
```

```
 System.out.println("ID: " + rs.getInt("cid") + ", Name: " + rs.getString("name"));
```

```
}
```

Step 6: Close Resources

```
rs.close();
```

```
stmt.close();
```

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

- Always close ResultSet, Statement, and Connection to free resources.

Complete Example: JDBC Program to Insert and Retrieve Data

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

```
public class JDBCExample {
 public static void main(String[] args) {
 Connection con = null;
 Statement stmt = null;
 ResultSet rs = null;

 try {
 // 1. Load JDBC driver
 Class.forName("com.mysql.cj.jdbc.Driver");

 // 2. Establish connection
 con = DriverManager.getConnection(
 "jdbc:mysql://localhost:3306/shop", "root", "password");

 // 3. Create statement
 stmt = con.createStatement();

 // 4a. Execute INSERT
 int rows = stmt.executeUpdate(
 "INSERT INTO Customer(cid, name, address, email, age) VALUES (1, 'John',
 'NY', 'john@example.com', 25)");
 System.out.println(rows + " row(s) inserted.");
 }
 }
}
```

```

// 4b. Execute SELECT
rs = stmt.executeQuery("SELECT * FROM Customer");

// 5. Process ResultSet
System.out.println("Customer Details:");
while (rs.next()) {
 int cid = rs.getInt("cid");
 String name = rs.getString("name");
 String address = rs.getString("address");
 String email = rs.getString("email");
 int age = rs.getInt("age");

 System.out.println(cid + " | " + name + " | " + address + " | " + email + " | " + age);
}

} catch (Exception e) {
 e.printStackTrace();
} finally {
 try { if (rs != null) rs.close(); } catch (Exception e) {}
 try { if (stmt != null) stmt.close(); } catch (Exception e) {}
 try { if (con != null) con.close(); } catch (Exception e) {}
}
}
}

```

## **5.Compare result set with row set. Explain prepared statement with example.**

Comparison: ResultSet vs RowSet

Both ResultSet and RowSet are used to handle data retrieved from a database, but they have key differences.

| Feature       | ResultSet                                                                              | RowSet                                                             |
|---------------|----------------------------------------------------------------------------------------|--------------------------------------------------------------------|
| Definition    | Represents a table of data returned by executing a SQL query.                          | A wrapper around ResultSet, can be disconnected from the database. |
| Connectivity  | Connected to database; requires live connection.                                       | Can operate disconnected; connects to DB only when needed.         |
| Scrolling     | Scrollable only if explicitly created (TYPE_SCROLL_INSENSITIVE/TYPE_SCROLL_SENSITIVE). | Scrollable by default.                                             |
| Serialization | Not serializable.                                                                      | Serializable; can be sent over network.                            |

|                |                                         |                                                                             |
|----------------|-----------------------------------------|-----------------------------------------------------------------------------|
| Event Handling | Does not support events.                | Supports events (e.g., RowSetListener).                                     |
| Usage          | For simple read-only or update queries. | Useful for disconnected operations, caching, and GUI applications.          |
| Package        | java.sql.ResultSet                      | javax.sql.rowset.RowSet and its implementations (CachedRowSet, JdbcRowSet). |

Example of RowSet (CachedRowSet):

```

import javax.sql.rowset.*;
import java.sql.*;
import com.sun.rowset.CachedRowSetImpl;

public class RowSetExample {
 public static void main(String[] args) throws Exception {
 CachedRowSet crs = new CachedRowSetImpl();
 crs.setUrl("jdbc:mysql://localhost:3306/shop");
 crs.setUsername("root");
 crs.setPassword("password");
 crs.setCommand("SELECT * FROM Customer");
 crs.execute(); // Connects to DB, fetches data

 while(crs.next()) {
 System.out.println(crs.getInt("cid") + " " + crs.getString("name"));
 }
 crs.close();
 }
}

```

## PreparedStatement in JDBC

Definition:

- PreparedStatement is a precompiled SQL statement that allows you to safely pass parameters to SQL queries.
- It is faster for repeated queries and prevents SQL injection.

## Advantages of PreparedStatement

1. Precompiled SQL → Faster execution for repeated queries.

2. Prevents SQL injection → Values are bound safely.
3. Dynamic parameters → Use placeholders (?) for variables.
4. Supports batch updates → Can execute multiple statements efficiently.

Syntax:

```
String sql = "INSERT INTO Customer(cid, name, age) VALUES (?, ?, ?)";
PreparedStatement pst = con.prepareStatement(sql);
pst.setInt(1, 1); // 1st parameter
pst.setString(2, "John"); // 2nd parameter
pst.setInt(3, 25); // 3rd parameter
pst.executeUpdate();
```

Example: Using PreparedStatement

```
import java.sql.*;

public class PreparedStatementExample {
 public static void main(String[] args) {
 String url = "jdbc:mysql://localhost:3306/shop";
 String user = "root";
 String pass = "password";

 try (Connection con = DriverManager.getConnection(url, user, pass)) {

 // 1. Create PreparedStatement
 String sql = "INSERT INTO Customer(cid, name, address, email, age) VALUES (?, ?, ?, ?, ?)";
 PreparedStatement pst = con.prepareStatement(sql);

 // 2. Set parameters
 pst.setInt(1, 1);
 pst.setString(2, "Alice");
 pst.setString(3, "Kathmandu");
 pst.setString(4, "alice@example.com");
 pst.setInt(5, 22);

 // 3. Execute
 int rows = pst.executeUpdate();
 System.out.println(rows + " row(s) inserted successfully.");

 // 4. Use PreparedStatement for SELECT
 pst = con.prepareStatement("SELECT * FROM Customer WHERE age > ?");
 pst.setInt(1, 20);
 ResultSet rs = pst.executeQuery();
 }
 }
}
```

```

 while(rs.next()) {
 System.out.println(rs.getInt("cid") + " " + rs.getString("name") + " " +
rs.getInt("age"));
 }

 } catch (SQLException e) {
 e.printStackTrace();
 }
}
}
}

```

## **6.How do you set and get Cookie in Servlet? Explain using suitable Java program.**

### **What is a Cookie?**

- A Cookie is a small piece of information sent by a server to a client (browser) and stored on the client side.
- Cookies are used to maintain state between HTTP requests because HTTP is stateless.
- Example uses: user login, shopping cart, session tracking.

### How to Set a Cookie in Servlet

- Use the javax.servlet.http.Cookie class.
- Add the cookie to the response using response.addCookie(cookie).

Steps:

1. Create a Cookie object:

```
Cookie cookie = new Cookie("username", "Bibash");
```

2. Set optional properties:

```
cookie.setMaxAge(60*60); // Lifetime in seconds (1 hour)
```

```
cookie.setPath("/"); // Path for which cookie is valid
```

3. Add cookie to response:

```
response.addCookie(cookie);
```

### How to Get a Cookie in Servlet

- Use request.getCookies() to retrieve all cookies sent by the browser.
- Loop through the cookies to find the one you need.

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

```
if(cookies != null){
```

```
 for(Cookie c : cookies){
```

```
 if(c.getName().equals("username")){
```

```
 String value = c.getValue();
```

```
 out.println("Welcome back, " + value);
```

```
 }
```

```
 }
```

```
}
```

Complete Example: Setting and Getting Cookies

HTML Page: login.html

```
<!DOCTYPE html>
<html>
<head>
 <title>Login</title>
</head>
<body>
 <h2>Enter Username</h2>
 <form action="SetCookieServlet" method="post">
 Username: <input type="text" name="username" required>
 <input type="submit" value="Login">
 </form>
</body>
</html>
```

Servlet to Set Cookie: SetCookieServlet.java

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

public class SetCookieServlet extends HttpServlet {
 protected void doPost(HttpServletRequest request, HttpServletResponse response)
 throws ServletException, IOException {

 response.setContentType("text/html");
 PrintWriter out = response.getWriter();

 // Get username from form
 String username = request.getParameter("username");

 // Create a cookie
 Cookie userCookie = new Cookie("username", username);
 userCookie.setMaxAge(60*60); // 1 hour
 userCookie.setPath("/");

 // Add cookie to response
 response.addCookie(userCookie);

 out.println("<html><body>");
 out.println("<h2>Cookie has been set for user: " + username + "</h2>");
 out.println("Go to Welcome Page");
 out.println("</body></html>");
 }
}
```

```

}

Servlet to Get Cookie: GetCookieServlet.java
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class GetCookieServlet extends HttpServlet {
 protected void doGet(HttpServletRequest request, HttpServletResponse response)
 throws ServletException, IOException {
 response.setContentType("text/html");
 PrintWriter out = response.getWriter();

 // Get all cookies
 Cookie[] cookies = request.getCookies();

 out.println("<html><body>");
 if(cookies != null){
 boolean found = false;
 for(Cookie c : cookies){
 if(c.getName().equals("username")){
 out.println("<h2>Welcome back, " + c.getValue() + "</h2>");
 found = true;
 break;
 }
 }
 if(!found){
 out.println("<h2>No username cookie found. Please login first.</h2>");
 }
 } else {
 out.println("<h2>No cookies found. Please login first.</h2>");
 }
 out.println("</body></html>");
 }
}

```

## How it Works

1. User opens login.html and submits username.
2. SetCookieServlet creates a **cookie** with the username and sends it to the browser.
3. Browser stores the cookie for 1 hour.
4. User clicks the link to GetCookieServlet.
5. Servlet reads the cookie using request.getCookies() and displays the username.

## **7.Explain different scripting elements of JSP with example.**

### **1. What are JSP Scripting Elements?**

Definition:

- Scripting elements in JSP are used to embed Java code inside a JSP page.
- They allow dynamic content generation and interaction with Java logic.

There are three main types of JSP scripting elements:

1. Declarations
2. Scriptlets
3. Expressions

### **2. JSP Scripting Elements**

#### **2.1 Declaration**

Syntax:

```
<%! dataType variableName; %>
<%! methodDeclaration() { ... } %>
```

- Used to declare variables and methods that become instance variables/methods of the generated servlet.
- Executed once when servlet is initialized.

Example:

```
<%@ page language="java" contentType="text/html; charset=UTF-8" %>
<%! int counter = 0; %>
<%! int incrementCounter() { return ++counter; } %>
<html>
<body>
<h2>Counter Value: <%= incrementCounter() %></h2>
</body>
</html>
```

#### **2.2 Scriptlet**

Syntax:

```
<% java code here %>

- Embeds Java code inside JSP page.
- Code executes every time the JSP is requested.

```

Example:

```
<%@ page language="java" contentType="text/html; charset=UTF-8" %>
<html>
<body>
<%
 int a = 5;
 int b = 10;
 int sum = a + b;
%>
<h2>Sum of <%=a%> and <%=b%> is: <%=sum%></h2>
</body>
```

```
</html>
```

## 2.3 Expression

Syntax:

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

- Evaluates a Java expression and directly inserts its value into the HTML output.
- Expression must return a value.

Example:

```
<%@ page language="java" contentType="text/html; charset=UTF-8" %>
<html>
<body>
<h2>Current Time: <%= new java.util.Date() %></h2>
</body>
</html>
```

## 3. Summary Table of Scripting Elements

| Element     | Syntax     | Purpose                               | Execution                     |
|-------------|------------|---------------------------------------|-------------------------------|
| Declaration | <%! ... %> | Declare variables/methods for servlet | Once when servlet initializes |
| Scriptlet   | <% ... %>  | Write Java code to perform logic      | Every request                 |
| Expression  | <%= ... %> | Output value directly to client       | Every request                 |

## 4. Complete Example Combining All Scripting Elements

```
<%@ page language="java" contentType="text/html; charset=UTF-8" %>
<%! int counter = 0; %>
<html>
<body>

<%
 counter++; // Scriptlet increments counter
 String message = "Hello JSP User!"; // Scriptlet variable
%>

<h2><%= message %></h2> <!-- Expression prints value -->
<h3>You are visitor number: <%= counter %></h3> <!-- Expression prints counter -->

</body>
</html>
```

## **8. Write short notes on:**

### **a) Java Web Frameworks**

Definition:

- Java web frameworks are predefined structures and libraries that help developers build web applications efficiently in Java.
- They simplify repetitive tasks like handling requests, database interaction, session management, and rendering views.

Types / Examples:

1. Spring MVC – Follows Model-View-Controller pattern, widely used for enterprise applications.
2. Struts – Based on MVC, supports form handling, validation, and centralized configuration.
3. JSF (JavaServer Faces) – Component-based framework for building UI using reusable UI components.
4. Hibernate (ORM framework) – Maps Java objects to database tables, simplifies database operations.

Advantages:

- Reduces boilerplate code.
- Promotes separation of concerns (MVC).
- Supports reusability and modularity.
- Faster development with built-in libraries and tools.

Example Use Case:

- A shopping website where Spring MVC handles HTTP requests, controllers, and JSP views for displaying products.

### **b) CORBA (Common Object Request Broker Architecture)**

Definition:

- CORBA is a standard architecture for distributed object systems.
- It allows objects written in different languages and running on different machines to communicate.

Key Components:

1. ORB (Object Request Broker): Core middleware that handles communication between client and server objects.
2. IDL (Interface Definition Language): Language-independent interface definition for objects.
3. Stubs & Skeletons:
  - Stub – Client-side proxy that represents the remote object.
  - Skeleton – Server-side dispatcher that calls actual object methods.

Advantages:

- Language and platform independent.
- Supports distributed applications.
- Standardized by OMG (Object Management Group).

Example Use Case:

- A banking system where a Java client interacts with a C++ server to perform transactions.

### c) Bean Bound Property

Definition:

- A bound property in JavaBeans is a property that notifies other objects (listeners) whenever its value changes.
- Useful for dynamic updates in GUI or business objects.

Key Points:

1. PropertyChangeListener – Interface that listens for property changes.
2. firePropertyChange() – Method used in JavaBean to notify listeners.
3. Bound properties allow loose coupling between components.

Example:

```
import java.beans.*;

public class PersonBean {
 private String name;
 private PropertyChangeSupport support = new PropertyChangeSupport(this);

 public String getName() { return name; }

 public void setName(String name) {
 String oldName = this.name;
 this.name = name;
 support.firePropertyChange("name", oldName, name); // Notify listeners
 }

 public void addPropertyChangeListener(PropertyChangeListener pcl) {
 support.addPropertyChangeListener(pcl);
 }

 public void removePropertyChangeListener(PropertyChangeListener pcl) {
 support.removePropertyChangeListener(pcl);
 }
}
```

**9. Write a GUI application to find sum and difference of two integer numbers. Use two text fields for input and third text field for output. Your program should display sum if the user presses the mouse and difference if the user releases the mouse.**

1. Steps to Create the Application

1. Use JFrame as the top-level container.
2. Add three JTextFields: two for input, one for output.
3. Use MouseListener to handle mouse events:
  - o mousePressed → calculate sum.
  - o mouseReleased → calculate difference.
4. Display the result in the third JTextField.

2. Complete Java Program

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

public class SumDiffGUI extends JFrame implements MouseListener {

 // Text fields
 JTextField tf1, tf2, tf3;

 public SumDiffGUI() {
 setTitle("Sum and Difference Calculator");
 setSize(400, 200);
 setLayout(new FlowLayout());
 setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

 // Labels
 add(new JLabel("Enter First Number:"));
 tf1 = new JTextField(10);
 add(tf1);

 add(new JLabel("Enter Second Number:"));
 tf2 = new JTextField(10);
 add(tf2);

 add(new JLabel("Result:"));
 tf3 = new JTextField(15);
 tf3.setEditable(false);
 add(tf3);

 // Add MouseListener to output field
 tf3.addMouseListener(this);
 }

 public void mouseClicked(MouseEvent e) {
 int num1 = Integer.parseInt(tf1.getText());
 int num2 = Integer.parseInt(tf2.getText());
 int result = num1 - num2;
 tf3.setText(result + "");
 }

 public void mouseEntered(MouseEvent e) {
 }

 public void mouseExited(MouseEvent e) {
 }

 public void mousePressed(MouseEvent e) {
 int num1 = Integer.parseInt(tf1.getText());
 int num2 = Integer.parseInt(tf2.getText());
 int result = num1 + num2;
 tf3.setText(result + "");
 }

 public void mouseReleased(MouseEvent e) {
 int num1 = Integer.parseInt(tf1.getText());
 int num2 = Integer.parseInt(tf2.getText());
 int result = num1 - num2;
 tf3.setText(result + "");
 }
}
```

```

 setVisible(true);
 }

// MouseListener methods
@Override
public void mousePressed(MouseEvent e) {
 try {
 int num1 = Integer.parseInt(tf1.getText());
 int num2 = Integer.parseInt(tf2.getText());
 int sum = num1 + num2;
 tf3.setText(String.valueOf(sum));
 } catch (NumberFormatException ex) {
 tf3.setText("Invalid Input!");
 }
}

@Override
public void mouseReleased(MouseEvent e) {
 try {
 int num1 = Integer.parseInt(tf1.getText());
 int num2 = Integer.parseInt(tf2.getText());
 int diff = num1 - num2;
 tf3.setText(String.valueOf(diff));
 } catch (NumberFormatException ex) {
 tf3.setText("Invalid Input!");
 }
}

// Unused MouseListener methods
@Override public void mouseClicked(MouseEvent e) {}
@Override public void mouseEntered(MouseEvent e) {}
@Override public void mouseExited(MouseEvent e) {}

// Main method
public static void main(String[] args) {
 new SumDiffGUI();
}
}

```

How it works:

1. User enters two numbers in tf1 and tf2.
2. When user presses mouse on result field, mousePressed() calculates sum.
3. When user releases mouse, mouseReleased() calculates difference.

## **10.What is bean design pattern? Explain simple, Boolean, and indexed property design patterns with suitable examples in detail.**

### **1. What is Bean Design Pattern?**

Definition:

- A Bean Design Pattern defines a conventional way to write Java classes (Beans) so that they can be easily reused, manipulated, and integrated in Java applications, especially GUI or enterprise applications.
- Beans follow naming conventions, use private instance variables, and provide getter/setter methods for accessing properties.

Key Features:

1. Properties are private and accessed via methods.
2. Supports reusability and event handling.
3. Follows standard naming convention for methods:
  - Getter for property prop → getProp()
  - Setter for property prop → setProp(value)

Applications: GUI components, JavaBeans in JSP, enterprise applications.

### **2. Types of Bean Properties**

There are three main types of properties in JavaBeans:

#### **2.1 Simple Property**

- A single-valued property of any type (int, String, etc.).
- Accessed using getter and setter methods.

Naming Convention:

```
private Type propertyName;
public Type getPropertyName() { ... }
public void setPropertyName(Type value) { ... }
```

Example: Simple Property

```
public class PersonBean {
 private String name; // Simple property
```

```
// Getter
public String getName() {
 return name;
}
```

```
// Setter
public void setName(String name) {
 this.name = name;
}
```

```
// Using the Bean
PersonBean person = new PersonBean();
```

```
person.setName("Bibash");
System.out.println(person.getName()); // Output: Bibash
```

## 2.2 Boolean Property

- A property of boolean type.
- Getter uses the prefix is instead of get.

Naming Convention:

```
private boolean propertyName;
public boolean isPropertyName() { ... } // Getter
public void setPropertyName(boolean value) { ... } // Setter
```

Example: Boolean Property

```
public class LightBean {
 private boolean on; // Boolean property
```

```
// Getter
public boolean isOn() {
 return on;
}
```

```
// Setter
public void setOn(boolean on) {
 this.on = on;
}
```

```
// Using the Bean
LightBean light = new LightBean();
light.setOn(true);
System.out.println(light.isOn()); // Output: true
```

## 2.3 Indexed Property

- An array property where individual elements can be accessed using an index.
- Supports get/set element and optionally get/set entire array.

Naming Convention:

```
private Type[] propertyName;
public Type getPropertyName(int index) { ... }
public void setPropertyName(int index, Type value) { ... }
public Type[] getPropertyName() { ... } // optional
public void setPropertyName(Type[] array) { ... } // optional
```

Example: Indexed Property

```
public class ScoresBean {
 private int[] scores = new int[5]; // Indexed property
```

```

// Getter for individual element
public int getScore(int index) {
 return scores[index];
}

// Setter for individual element
public void setScore(int index, int value) {
 scores[index] = value;
}

// Getter for entire array (optional)
public int[] getScores() {
 return scores;
}

// Setter for entire array (optional)
public void setScores(int[] scores) {
 this.scores = scores;
}

// Using the Bean
ScoresBean sb = new ScoresBean();
sb.setScore(0, 95);
sb.setScore(1, 88);
System.out.println(sb.getScore(0)); // Output: 95

```

## **11.Define RMI. What is stub and parameter marshalling? Write a client/server application using RMI to find product of two numbers.**

### **1. What is RMI?**

RMI (Remote Method Invocation) is a Java API that allows an object running in one Java Virtual Machine (JVM) to invoke methods on an object running in another JVM, possibly on a remote machine.

#### **Key Points:**

- RMI allows distributed computing in Java.
- Enables client-server communication using Java objects.
- Works only in Java (unlike CORBA which is language-independent).

### **2. Stub in RMI**

- The stub is a client-side proxy for the remote object.
- Acts as a local representative of the remote object.
- Responsibilities:

1. Marshal parameters – Convert method arguments into a format for transmission.
2. Send request to remote JVM.
3. Unmarshal return values – Convert returned data back to Java objects.

Think of stub as a “middleman” between client and remote server object.

### 3. Parameter Marshalling

- Marshalling is the process of converting parameters into a byte stream so they can be sent over the network.
- Unmarshalling is the reverse – converting byte stream back into Java objects on the server.
- RMI automatically handles marshalling/unmarshalling for serializable objects.

### 4. RMI Client/Server Application: Product of Two Numbers

Step 1: Remote Interface

```
import java.rmi.Remote;
import java.rmi.RemoteException;
```

// Remote interface

```
public interface Product extends Remote {
 int multiply(int a, int b) throws RemoteException;
}
```

Step 2: Server Implementation

```
import java.rmi.server.UnicastRemoteObject;
import java.rmi.RemoteException;
```

// Server class implementing the remote interface

```
public class ProductImpl extends UnicastRemoteObject implements Product {
```

```
protected ProductImpl() throws RemoteException {
 super();
}
```

@Override

```
public int multiply(int a, int b) throws RemoteException {
 return a * b;
}
```

}

Step 3: Server Program

```
import java.rmi.registry.LocateRegistry;
import java.rmi.registry.Registry;
```

```
public class ProductServer {
```

```

public static void main(String[] args) {
 try {
 // Create remote object
 ProductImpl product = new ProductImpl();

 // Start RMI registry on port 1099
 Registry registry = LocateRegistry.createRegistry(1099);

 // Bind remote object in registry
 registry.rebind("ProductService", product);

 System.out.println("Product Server is ready...");
 } catch (Exception e) {
 e.printStackTrace();
 }
}

```

#### Step 4: Client Program

```

import java.rmi.registry.LocateRegistry;
import java.rmi.registry.Registry;
import java.util.Scanner;

public class ProductClient {
 public static void main(String[] args) {
 try {
 // Connect to RMI registry on localhost
 Registry registry = LocateRegistry.getRegistry("localhost", 1099);

 // Lookup remote object
 Product product = (Product) registry.lookup("ProductService");

 Scanner sc = new Scanner(System.in);
 System.out.print("Enter first number: ");
 int a = sc.nextInt();
 System.out.print("Enter second number: ");
 int b = sc.nextInt();

 // Invoke remote method
 int result = product.multiply(a, b);
 System.out.println("Product of " + a + " and " + b + " is: " + result);

 } catch (Exception e) {
 e.printStackTrace();
 }
 }
}

```

```
 }
}
}
```

## 5. How It Works

### 1. Server Side:

- o ProductImpl implements Product interface.
- o Remote object is registered with RMI registry.

### 2. Client Side:

- o Looks up the remote object via registry.
- o Calls multiply(a, b) on the stub.
- o Parameters are marshalled, sent to server, server computes, result is unmarshalled back.

### 3. Stub & Marshalling:

- o Stub acts as a proxy.
- o Converts method call into network call (marshalling).
- o Server executes method and returns result (unmarshalling).

## 6. Output Example

Client Input:

Enter first number: 5

Enter second number: 7

Client Output:

Product of 5 and 7 is: 35

## 2019 Batch

### 2.What is the role of Event Listener in event handling? List different event listener provided by Java.

Role of Event Listener:

- In Java GUI programming (AWT/Swing), events are generated when a user interacts with GUI components, e.g., clicking a button, typing a key, closing a window.
- An Event Listener is an interface that defines methods to handle these events.
- Components register a listener using addXXXListener() method. When an event occurs, the listener's method is called automatically.

Different Event Listeners in Java:

| Listener            | Description                                                     |
|---------------------|-----------------------------------------------------------------|
| ActionListener      | Handles button clicks or menu selections (actionPerformed())    |
| MouseListener       | Handles mouse events: click, press, release, enter, exit        |
| MouseMotionListener | Handles mouse movement: drag and move                           |
| KeyListener         | Handles keyboard events: key pressed, released, typed           |
| WindowListener      | Handles window events: opened, closing, closed, iconified, etc. |

|                   |                                                                              |
|-------------------|------------------------------------------------------------------------------|
| ItemListener      | Handles selection changes in checkbox, radio buttons<br>(itemStateChanged()) |
| FocusListener     | Handles focus gained/lost on components                                      |
| ComponentListener | Handles resizing, moving, hiding, showing of components                      |
| TextListener      | Handles text changes in text components (textValueChanged())                 |

Example:

```
Button b = new Button("Click Me");
b.addActionListener(new ActionListener() {
 public void actionPerformed(ActionEvent e) {
 System.out.println("Button Clicked!");
 }
});
```

### 3.What is keyEvent? Explain with proper example.

Definition:

- A KeyEvent occurs when the user presses, releases, or types a key on the keyboard.
- Handled using KeyListener interface.

Key Methods in KeyListener:

- keyPressed(KeyEvent e) – invoked when a key is pressed.
- keyReleased(KeyEvent e) – invoked when a key is released.
- keyTyped(KeyEvent e) – invoked when a key is typed (character input).

Example:

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

public class KeyEventExample extends Frame implements KeyListener {
 Label label;

 KeyEventExample() {
 setSize(300, 200);
 setLayout(new FlowLayout());
 label = new Label("Press any key");
 add(label);
 addKeyListener(this);
 setVisible(true);
 }

 public void keyPressed(KeyEvent e) {
 label.setText("Key Pressed: " + e.getKeyChar());
 }

 public void keyReleased(KeyEvent e) {
```

```

 label.setText("Key Released: " + e.getKeyChar());
 }

 public void keyTyped(KeyEvent e) {
 label.setText("Key Typed: " + e.getKeyChar());
 }

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

```

#### **4.What is the use of RowSet interface? Explain connected and disconnected rowsets.**

RowSet Interface:

- RowSet is a wrapper over ResultSet that allows easier handling of tabular data.
- Supports scrolling, updating, and event handling.
- Part of javax.sql.rowset package.

Types of RowSet:

1. Connected RowSet
  - Maintains continuous connection to the database.
  - Example: JdbcRowSet.
  - Updates immediately reflect in DB.
2. Disconnected RowSet
  - Can work without being connected to the database.
  - Fetches data once, works locally, then reconnects if needed.
  - Example: CachedRowSet.
  - Useful for GUI applications to reduce DB load.

Example (CachedRowSet):

```

import javax.sql.rowset.*;
import java.sql.*;
import com.sun.rowset.CachedRowSetImpl;

```

```

public class RowSetDemo {
 public static void main(String[] args) throws Exception {
 CachedRowSet crs = new CachedRowSetImpl();
 crs.setUrl("jdbc:mysql://localhost:3306/shop");
 crs.setUsername("root");
 crs.setPassword("password");
 crs.setCommand("SELECT * FROM Customer");
 crs.execute();
 while (crs.next()) {
 System.out.println(crs.getInt("cid") + " " + crs.getString("name"));
 }
 }
}

```

```
}
```

## 5.What are bounded and constrained properties in Javabeans? Explain the advantages of Java beans.

Bound Property:

- Notifies listeners when a property value changes.
- Useful for GUI updates or dependent objects.

Constrained Property:

- Listeners can veto (reject) a property change.
- Allows controlled updates.

Advantages of JavaBeans:

1. Reusable components.
2. Encapsulated properties with getters/setters.
3. Supports event handling.
4. Can be manipulated in IDE visual tools.
5. Can be persisted and transferred between applications.

Example:

```
import java.beans.*;
```

```
public class PersonBean {
 private String name;
 private PropertyChangeSupport support = new PropertyChangeSupport(this);

 public String getName() { return name; }
 public void setName(String name) {
 String old = this.name;
 this.name = name;
 support.firePropertyChange("name", old, name); // Bound property
 }

 public void addPropertyChangeListener(PropertyChangeListener l) {
 support.addPropertyChangeListener(l);
 }
}
```

## 6.What are the key methods provided in HttpSession interface for handling session? Explain.

Definition:

- HttpSession allows session tracking between client and server in web applications.

Key Methods:

| Method                    | Description                  |
|---------------------------|------------------------------|
| getAttribute(String name) | Get object stored in session |

|                                         |                                           |
|-----------------------------------------|-------------------------------------------|
| setAttribute(String name, Object value) | Store object in session                   |
| removeAttribute(String name)            | Remove object from session                |
| invalidate()                            | Destroy the session                       |
| getId()                                 | Returns session ID                        |
| getCreationTime()                       | Returns session creation time             |
| getLastAccessedTime()                   | Returns last time client accessed session |
| isNew()                                 | Checks if session is new                  |

Example:

```
HttpSession session = request.getSession();
session.setAttribute("username", "Bibash");
String user = (String) session.getAttribute("username");
session.invalidate();
```

## 7.Explain RMI architecture in Java in detail.

RMI (Remote Method Invocation) Architecture:

Components:

1. Client – Requests a remote service.
2. Server – Implements remote interface.
3. Remote Interface – Defines methods callable remotely.
4. Stub (Client Proxy) – Represents remote object on client side.
5. Skeleton (Server Proxy) – Receives requests, invokes actual methods (pre-Java 2, now optional).
6. RMI Registry – Directory service for binding remote objects.

Flow:

1. Server creates remote object and binds it to RMI registry.
2. Client looks up object in registry.
3. Client invokes methods on stub.
4. Parameters are marshalled and sent to server.
5. Server executes method, result is unmarshalled back.

Diagram:

Client -> Stub -> RMI Registry -> Skeleton -> Remote Object (Server)

## 8.Write short notes on:

### a) Adapter Classes

- Provide empty implementations of listener interfaces with multiple methods.
- Allows programmer to override only required methods.
- Example: MouseAdapter, KeyAdapter.

```
tf.addMouseListener(new MouseAdapter() {
 public void mouseClicked(MouseEvent e) {
 System.out.println("Clicked!");
 }
});
```

**b) CORBA**

- Common Object Request Broker Architecture.
- Allows distributed objects in different languages to communicate.
- Key components: ORB, IDL, Stub, Skeleton.
- Language and platform independent.

**c) Life Cycle of a Servlet**

Stages:

1. Loading & Instantiation – Servlet class is loaded and instantiated by container.
2. Initialization (init()) – Servlet initialized once.
3. Request Handling (service()) – Handles each client request (doGet, doPost).
4. Destruction (destroy()) – Called once before servlet is removed; used to release resources.

**9.How frame can be created in Java? Explain. Write a program to create a GUI application in Java that identifies the smaller and greater number between two input numbers taken through two text fields and displays the result in a label. If the user presses the mouse it should display the smaller number and if the user releases the mouse it should display the greater number.**

Program:

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

public class CompareNumbersGUI extends JFrame implements MouseListener {
 JTextField tf1, tf2;
 JLabel resultLabel;

 public CompareNumbersGUI() {
 setTitle("Compare Numbers");
 setSize(400,200);
 setLayout(new FlowLayout());
 setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

 add(new JLabel("Enter First Number:"));
 tf1 = new JTextField(10);
 add(tf1);

 add(new JLabel("Enter Second Number:"));
 tf2 = new JTextField(10);
 add(tf2);
 }

 public void mouseClicked(MouseEvent e) {
 if (e.getButton() == MouseEvent.BUTTON1) {
 String s1 = tf1.getText();
 String s2 = tf2.getText();
 int i1 = Integer.parseInt(s1);
 int i2 = Integer.parseInt(s2);
 if (i1 < i2) {
 resultLabel.setText("Smaller Number is " + i1);
 } else {
 resultLabel.setText("Greater Number is " + i2);
 }
 }
 }

 public void mouseEntered(MouseEvent e) {}

 public void mouseExited(MouseEvent e) {}

 public void mousePressed(MouseEvent e) {}

 public void mouseReleased(MouseEvent e) {}
}
```

```

resultLabel = new JLabel("Result: ");
add(resultLabel);

resultLabel.addMouseListener(this);
setVisible(true);
}

public void mousePressed(MouseEvent e) {
try {
 int a = Integer.parseInt(tf1.getText());
 int b = Integer.parseInt(tf2.getText());
 resultLabel.setText("Smaller: " + Math.min(a,b));
} catch(Exception ex) {
 resultLabel.setText("Invalid Input!");
}
}

public void mouseReleased(MouseEvent e) {
try {
 int a = Integer.parseInt(tf1.getText());
 int b = Integer.parseInt(tf2.getText());
 resultLabel.setText("Greater: " + Math.max(a,b));
} catch(Exception ex) {
 resultLabel.setText("Invalid Input!");
}
}

public void mouseClicked(MouseEvent e) {}
public void mouseEntered(MouseEvent e) {}
public void mouseExited(MouseEvent e) {}

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

```

**10.Explain JDBC architecture. Write a program to insert three records into a table Item which is in the database Shop and contains the columns ItemID, Name, UnitPrice, Units and ExpiryDate.**

JDBC Architecture:

1. JDBC API – Interfaces for Java programs.
2. JDBC Driver Manager – Loads drivers, manages connections.
3. JDBC Drivers – Vendor-specific, communicate with DB.

#### 4. Database – Relational database storing data.

Program to Insert Three Records into Item Table:

```
import java.sql.*;

public class InsertItemRecords {
 public static void main(String[] args) {
 String url = "jdbc:mysql://localhost:3306/Shop";
 String user = "root";
 String pass = "password";

 try (Connection con = DriverManager.getConnection(url,user,pass);
 Statement stmt = con.createStatement()) {

 String sql1 = "INSERT INTO Item(ItemID, Name, UnitPrice, Units, ExpiryDate) "+
 "VALUES (1,'Milk',50,20,'2026-01-30')";
 String sql2 = "INSERT INTO Item(ItemID, Name, UnitPrice, Units, ExpiryDate) "+
 "VALUES (2,'Bread',30,15,'2026-01-15')";
 String sql3 = "INSERT INTO Item(ItemID, Name, UnitPrice, Units, ExpiryDate) "+
 "VALUES (3,'Butter',80,10,'2026-02-10')";

 stmt.executeUpdate(sql1);
 stmt.executeUpdate(sql2);
 stmt.executeUpdate(sql3);

 System.out.println("Records inserted successfully!");

 } catch(SQLException e) {
 e.printStackTrace();
 }
 }
}
```

**11. Differentiate between servlet and JSP. Create a servlet that computes and displays factorial of an input number entered from a page when the button from that page is pressed by the user.**

Difference:

| Feature     | Servlet                       | JSP                                          |
|-------------|-------------------------------|----------------------------------------------|
| Execution   | Java code compiled as class   | HTML with embedded Java, compiled to servlet |
| Use         | Business logic, processing    | Presentation, view                           |
| Ease of use | Requires writing HTML in Java | Easier to mix HTML and Java                  |
| Compilation | Precompiled                   | Compiled at first request                    |

Servlet to Compute Factorial:

HTML Input Page: factorial.html

```
<html>
<body>
<h2>Enter a number:</h2>
<form action="FactorialServlet" method="post">
 <input type="number" name="num" required>
 <input type="submit" value="Compute Factorial">
</form>
</body>
</html>
```

Servlet: FactorialServlet.java

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

public class FactorialServlet extends HttpServlet {
 protected void doPost(HttpServletRequest request, HttpServletResponse response)
 throws ServletException, IOException {

 int n = Integer.parseInt(request.getParameter("num"));
 long fact = 1;
 for(int i=1;i<=n;i++){
 fact *= i;
 }

 response.setContentType("text/html");
 PrintWriter out = response.getWriter();
 out.println("<h2>Factorial of " + n + " is " + fact + "</h2>");
 }
}
```

## 2018 Batch

### **2.Why do we use Panels while creating GUI applications in Java? Explain the components of the event handling model in Java.**

Why use Panels:

- A Panel (JPanel or Panel) is a container used to group and organize GUI components.
- Helps to divide complex GUI into smaller, manageable sections.
- Allows applying different layouts to different parts of GUI.
- Supports nested layouts, making GUI design modular.

Example:

```
JFrame frame = new JFrame("Panel Example");
```

```
JPanel panel1 = new JPanel();
```

```
JPanel panel2 = new JPanel();
```

```
panel1.add(new JButton("Button1"));
```

```
panel2.add(new JButton("Button2"));
```

```
frame.add(panel1, BorderLayout.NORTH);
```

```
frame.add(panel2, BorderLayout.SOUTH);
```

```
frame.setSize(300,200);
```

```
frame.setVisible(true);
```

Components of Event Handling Model in Java:

1. Event Source – Component generating event (e.g., Button, TextField).
2. Event Object – Encapsulates information about the event (e.g., ActionEvent, MouseEvent).
3. Event Listener – Interface that defines methods to handle events (e.g., ActionListener).
4. Event Registration – Linking event listener to event source using addXXXListener().

Flow:

Event Source → Event Occurs → Event Object Created → Event Listener Called → Event Handled

### **3.What are adapter classes? What is the benefit of using adapter classes in Java while creating events in GUI programs? Explain with an example**

Definition:

- Adapter classes provide empty implementations of listener interfaces with multiple methods.
- Allows programmers to override only required methods instead of all.

Benefit:

- Reduces boilerplate code.
- Simplifies event handling for interfaces with many methods.

Example: Using MouseAdapter

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

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

```

public class AdapterExample extends Frame {
 Label label;

 AdapterExample() {
 setSize(300,200);
 setLayout(new FlowLayout());
 label = new Label("Click inside the frame");
 add(label);

 addMouseListener(new MouseAdapter() {
 public void mouseClicked(MouseEvent e) {
 label.setText("Mouse Clicked at: " + e.getX() + "," + e.getY());
 }
 });
 }

 setVisible(true);
}

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

```

**4.How the data can be accessed through a resultset object? Explain forward only, scroll-insensitive and scroll-sensitive resultsets.**

Accessing Data Through ResultSet

- ResultSet stores rows retrieved from a database query.
- Methods to access data:
  - next() – Moves cursor forward one row.
  - previous() – Moves cursor backward (scrollable).
  - getInt(), getString() – Get column data.
  - absolute(n) – Moves to nth row.

Types of ResultSet:

| Type               | Features                                                                                    |
|--------------------|---------------------------------------------------------------------------------------------|
| Forward-Only       | Cursor moves only forward. (TYPE_FORWARD_ONLY)                                              |
| Scroll-Insensitive | Can scroll forward/backward, data changes in DB are not reflected (TYPE_SCROLL_INSENSITIVE) |
| Scroll-Sensitive   | Can scroll, reflects changes in DB made by others (TYPE_SCROLL_SENSITIVE)                   |

Example:

```
Statement stmt = con.createStatement	ResultSet.TYPE_SCROLL_INSENSITIVE,
ResultSet.CONCUR_READ_ONLY);
ResultSet rs = stmt.executeQuery("SELECT * FROM Student");
rs.last();
while(rs.previous()) {
 System.out.println(rs.getString("Name"));
}
```

## 5.What is a Javabean? Explain different types of properties in Javabeans.

JavaBean:

- Reusable Java component that follows certain conventions:
  1. Private properties with public getters/setters.
  2. No-argument constructor.
  3. Serializable (implements Serializable).

Types of Properties:

1. Simple Property: Single value (e.g., String, int).
2. private String name;
3. public String getName() { return name; }
4. public void setName(String name) { this.name = name; }
5. Boolean Property: Boolean values, getter uses is prefix.
6. private boolean active;
7. public boolean isActive() { return active; }
8. public void setActive(boolean active) { this.active = active; }
9. Indexed Property: Array-based property, access via index.
10. private int[] scores = new int[5];
11. public int getScore(int index) { return scores[index]; }
12. public void setScore(int index, int value) { scores[index] = value; }

## 6.What are the differences between servlets and Java applications? Explain different methods used for handling cookies in Java.

Differences:

| Feature    | Java Application        | Servlet                                          |
|------------|-------------------------|--------------------------------------------------|
| Execution  | Runs standalone         | Runs inside servlet container                    |
| I/O        | Standard I/O            | HTTP request/response                            |
| Networking | Must implement manually | Provided by container                            |
| Multi-user | Single-user             | Handles multiple clients                         |
| Lifecycle  | Controlled by main()    | Controlled by container (init, service, destroy) |

Methods for Handling Cookies:

```
// Create a cookie
Cookie c = new Cookie("username","Bibash");
response.addCookie(c);
```

```

// Read cookies
Cookie[] cookies = request.getCookies();
for(Cookie ck : cookies) {
 System.out.println(ck.getName() + ":" + ck.getValue());
}

// Delete cookie
c.setMaxAge(0);
response.addCookie(c);

```

## **7.What are marshalling and unmarshalling of arguments in RMI? Differentiate between CORBA and RMI.**

Marshalling:

- Converts method parameters into byte stream for network transfer.

Unmarshalling:

- Converts byte stream back into original objects at the remote JVM.

RMI vs CORBA:

| Feature       | RMI                  | CORBA                               |
|---------------|----------------------|-------------------------------------|
| Language      | Java only            | Language-independent                |
| Object Access | Remote Java objects  | Distributed objects in any language |
| IDL           | Not required         | Required to define interface        |
| Ease of Use   | Easier for Java apps | More complex, enterprise-level      |

## **8.Write short notes on:**

### **a) 2D Shapes in Swing**

- Draw shapes using Graphics and Graphics2D.
- Common methods: drawRect(), drawOval(), drawLine(), fillRect(), fillOval().
- Example:

```

public void paint(Graphics g) {
 g.drawRect(50,50,100,50);
 g.drawOval(50,120,100,50);
}

```

### **b) MouseEvent & MouseListener**

- MouseEvent generated when mouse interacts with component.
- MouseListener interface methods:
  - mouseClicked(), mousePressed(), mouseReleased(), mouseEntered(), mouseExited().
- Adapter classes simplify implementation.

### **c) Session Tracking**

- Mechanism to maintain user state in web apps.

- Techniques: HttpSession, Cookies, URL Rewriting.
- Methods: getSession(), setAttribute(), getAttribute(), invalidate().

**9.Why do we need to use Layout managers while creating GUI application in Java?  
Explain the key interfaces required to develop any database application using Java.  
Also, write a program to display the records from the table Students who have got  
distinction and have a major subject Data science. Assume that the Student table is in  
the database College and contains the columns Rollno, Name, Level, Division and  
Major.**

Why Use Layout Managers:

- Controls position and size of components automatically.
- Avoids manual setBounds() and improves cross-platform look.
- Common Layouts: FlowLayout, BorderLayout, GridLayout, BoxLayout.

JDBC Key Interfaces:

1. DriverManager – Loads driver & establishes connection.
2. Connection – Connects to database.
3. Statement / PreparedStatement – Executes SQL queries.
4. ResultSet – Holds query results.

Program to Display Distinction Students in Data Science:

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

```
public class DistinctionDS {
 public static void main(String[] args) {
 String url = "jdbc:mysql://localhost:3306/College";
 String user = "root";
 String pass = "password";

 try(Connection con = DriverManager.getConnection(url,user,pass);
 Statement stmt = con.createStatement()) {

 String query = "SELECT * FROM Student WHERE Division='Distinction' AND
Major='Data science'";
 ResultSet rs = stmt.executeQuery(query);

 while(rs.next()) {
 System.out.println(rs.getInt("Rollno") + " " +
rs.getString("Name") + " " +
rs.getString("Level") + " " +
rs.getString("Division") + " " +
rs.getString("Major"));
 }
 }
 }
}
```

```

 } catch(SQLException e) {
 e.printStackTrace();
 }
 }
}

```

**10. Differentiate between declaration and expression tags used in JSP. Also create a JSP page that contains three textboxes to input three numbers and a button. It should display the greatest number among three numbers when the button is clicked.**

Differences:

| Tag         | Syntax        | Purpose                                    |
|-------------|---------------|--------------------------------------------|
| Declaration | <%! int x; %> | Declare variables/methods in servlet class |
| Expression  | <%= x %>      | Insert dynamic value into HTML output      |

JSP Page to Find Greatest Number:

```

<%@ page language="java" contentType="text/html; charset=UTF-8" %>
<html>
<body>
<%
int a = 0, b = 0, c = 0;
String result = "";
if(request.getParameter("submit") != null) {
 a = Integer.parseInt(request.getParameter("num1"));
 b = Integer.parseInt(request.getParameter("num2"));
 c = Integer.parseInt(request.getParameter("num3"));
 result = "Greatest Number is: " + Math.max(a, Math.max(b,c));
}
%>
<form method="post">
 Num1: <input type="text" name="num1">

 Num2: <input type="text" name="num2">

 Num3: <input type="text" name="num3">

 <input type="submit" name="submit" value="Find Greatest">
</form>
<h3><%= result %></h3>
</body>
</html>

```

**11. What is RMI? Explain its components. Write a client-server application in RMI to find the selling price of an item with cost price Rs. 5000 after a discount of Rs.50.**

Definition & Components:

- RMI – Remote Method Invocation for calling methods on remote objects.
- Components:

1. Remote Interface – Defines methods.
2. Server – Implements interface & registers with registry.
3. Client – Looks up registry & invokes methods.
4. Stub – Client-side proxy.
5. Skeleton – Server-side dispatcher (optional).

Client-Server Program (Selling Price):

Interface:

```
import java.rmi.Remote;
import java.rmi.RemoteException;
```

```
public interface Price extends Remote {
 double computePrice(double cost, double discount) throws RemoteException;
}
```

Server Implementation:

```
import java.rmi.server.UnicastRemoteObject;
import java.rmi.RemoteException;
```

```
public class PriceImpl extends UnicastRemoteObject implements Price {
 protected PriceImpl() throws RemoteException { super(); }
 public double computePrice(double cost, double discount) throws RemoteException {
 return cost - discount;
 }
}
```

Server Program:

```
import java.rmi.registry.LocateRegistry;
import java.rmi.registry.Registry;

public class PriceServer {
 public static void main(String[] args) {
 try {
 PriceImpl obj = new PriceImpl();
 Registry reg = LocateRegistry.createRegistry(1099);
 reg.rebind("PriceService", obj);
 System.out.println("Price Server Ready...");
 } catch(Exception e) { e.printStackTrace(); }
 }
}
```

Client Program:

```
import java.rmi.registry.LocateRegistry;
import java.rmi.registry.Registry;
```

```

public class PriceClient {
 public static void main(String[] args) {
 try {
 Registry reg = LocateRegistry.getRegistry("localhost", 1099);
 Price p = (Price) reg.lookup("PriceService");
 double sellingPrice = p.computePrice(5000, 50);
 System.out.println("Selling Price: Rs. " + sellingPrice);
 } catch(Exception e) { e.printStackTrace(); }
 }
}

```

Output:

Selling Price: Rs. 4950.0

## 2017 Batch

**2.Why do we need top level container like JFrame to write Java programs with GUI?**

**How can we display two dimensional objects in Java?**

Why we need JFrame:

- A top-level container like JFrame is needed to provide a window where components can be added.
- JFrame handles window decorations, resizing, closing, and painting.
- All Swing GUI components must be added to a container, and JFrame is the most commonly used one.

Displaying 2D Objects:

- Use paint(Graphics g) or paintComponent(Graphics g) in a component.
- Use Graphics methods like:
  - drawRect(x,y,width,height) – Draw rectangle
  - drawOval(x,y,width,height) – Draw oval
  - drawLine(x1,y1,x2,y2) – Draw line
  - fillRect, fillOval for filled shapes

Example:

```

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

```

```

public class DrawShapes extends JFrame {
 DrawShapes() {
 setTitle("2D Shapes Example");
 setSize(400,300);
 setVisible(true);
 setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
 }
}

```

```

 }

public void paint(Graphics g) {
 super.paint(g);
 g.drawRect(50,50,100,50);
 g.drawOval(200,50,100,50);
 g.drawLine(50,150,300,150);
}

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

```

### **3.Explain JTextField and JTextArea components of Java swing library.**

JTextField:

- Single-line text input field.
- Can set columns for width.
- Example methods: getText(), setText().

Example:

```

JTextField tf = new JTextField(20);
tf.setText("Enter Name");
String name = tf.getText();

```

JTextArea:

- Multi-line text area for input/output.
- Supports scrolling with JScrollPane.
- Example methods: append(), setText().

Example:

```

JTextArea ta = new JTextArea(5, 20);
ta.setText("Hello\nWorld");
ta.append("\nNew Line");
JScrollPane sp = new JScrollPane(ta);

```

### **4.How do you execute SQL statements using JDBC? Explain with example.**

Steps to execute SQL:

1. Load JDBC driver (optional in modern Java).
2. Create Connection to database.
3. Create Statement or PreparedStatement.
4. Execute SQL query (executeQuery for SELECT, executeUpdate for INSERT/UPDATE/DELETE).
5. Process ResultSet.
6. Close resources.

Example:

```

import java.sql.*;

public class JdbcExample {
 public static void main(String[] args) {
 try {
 Connection con = DriverManager.getConnection(
 "jdbc:mysql://localhost:3306/Shop","root","password");
 Statement stmt = con.createStatement();
 stmt.executeUpdate("INSERT INTO Item VALUES (1,'Milk',50,20,'2026-01-30')");
 ResultSet rs = stmt.executeQuery("SELECT * FROM Item");
 while(rs.next()) {
 System.out.println(rs.getString("Name") + " " + rs.getInt("UnitPrice"));
 }
 rs.close();
 stmt.close();
 con.close();
 } catch(Exception e) { e.printStackTrace(); }
 }
}

```

## **5.Compare result set with row set. Explain prepared statement with example.**

ResultSet:

- Cursor-based object from executing query.
- Connected to DB; closed when connection closes.
- Requires manual navigation (next(), previous()).

RowSet:

- Wrapper over ResultSet.
- Can be connected or disconnected.
- Supports scrolling, updating, and event handling.

PreparedStatement Example:

```

PreparedStatement ps = con.prepareStatement(
 "INSERT INTO Item(ItemID, Name, UnitPrice) VALUES(?, ?, ?)");
ps.setInt(1, 2);
ps.setString(2, "Bread");
ps.setInt(3, 30);
ps.executeUpdate();

```

Advantages of PreparedStatement:

1. Precompiled – faster execution.
2. Prevents SQL injection.
3. Supports parameters easily.

## **6.How do you set and get Cookie in Servlet? Explain using suitable Java program.**

Set Cookie:

```

Cookie c = new Cookie("username","Bibash");
response.addCookie(c);
Get Cookie:
Cookie[] cookies = request.getCookies();
for(Cookie ck : cookies) {
 System.out.println(ck.getName() + ":" + ck.getValue());
}
Delete Cookie:
c.setMaxAge(0);
response.addCookie(c);

```

## **7.Explain different scripting elements of JSP with example.**

Types of JSP scripting elements:

1. Declarations: <%! int x=0; %> – Declare variables/methods
2. Scriptlets: <% x++; %> – Embed Java code in page
3. Expressions: <%= x %> – Output value to client

Example:

```

<%@ page language="java" contentType="text/html; charset=UTF-8"%>
<%! int x=5; %>
<%
 x++;
%>
Value of x is <%= x %>

```

## **8.Write short notes on:**

### **a) Java Web Frameworks**

- Frameworks to develop web apps: Spring, Struts, JSF.
- Provide MVC architecture, easier database handling, session management.

### **b) CORBA**

- Distributed object architecture.
- Language-independent communication.
- Uses IDL for interface definition.

### **c) Bean Bound Property**

- Notifies listeners when property changes.
- Useful in GUI updates.

## **9.Write a GUI application to find sum and difference of two integer numbers. Use two text fields for input and third text field for output. Your program should display sum if the user presses the mouse and difference if the user releases the mouse.**

Program:

```
import javax.swing.*;
```

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

public class SumDiffGUI extends JFrame implements MouseListener {
 JTextField tf1, tf2, tf3;

 SumDiffGUI() {
 setSize(400,200);
 setLayout(new FlowLayout());
 tf1 = new JTextField(5);
 tf2 = new JTextField(5);
 tf3 = new JTextField(10);
 tf3.setEditable(false);

 add(new JLabel("Num1:")); add(tf1);
 add(new JLabel("Num2:")); add(tf2);
 add(new JLabel("Result:")); add(tf3);

 tf3.addMouseListener(this);
 setVisible(true);
 setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
 }

 public void mousePressed(MouseEvent e) {
 int a = Integer.parseInt(tf1.getText());
 int b = Integer.parseInt(tf2.getText());
 tf3.setText(String.valueOf(a+b));
 }

 public void mouseReleased(MouseEvent e) {
 int a = Integer.parseInt(tf1.getText());
 int b = Integer.parseInt(tf2.getText());
 tf3.setText(String.valueOf(a-b));
 }

 public void mouseClicked(MouseEvent e) {}
 public void mouseEntered(MouseEvent e) {}
 public void mouseExited(MouseEvent e) {}

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

## **10.What is bean design pattern? Explain simple, Boolean, and indexed property design patterns with suitable examples in detail.**

Definition:

- Convention to create reusable Java components with properties and events.

Property Types:

1. Simple Property: Single value

```
private String name;
public String getName(){ return name; }
public void setName(String n){ name=n; }
```

2. Boolean Property: Boolean value

```
private boolean active;
public boolean isActive(){ return active; }
public void setActive(boolean a){ active=a; }
```

3. Indexed Property: Array property

```
private int[] scores = new int[5];
public int getScore(int i){ return scores[i]; }
public void setScore(int i,int v){ scores[i]=v; }
```

## **11.Define RMI. What is stub and parameter marshalling? Write a client/server application using RMI to find product of two numbers.**

RMI Definition:

- Remote Method Invocation – call methods of objects on a different JVM or machine.

Stub:

- Client-side proxy representing remote object. Handles marshalling and communication.

Parameter Marshalling:

- Converts method parameters into byte stream for network transfer.
- Unmarshalling reverses it at server side.

Client/Server Example – Product of Two Numbers

Interface:

```
import java.rmi.Remote;
import java.rmi.RemoteException;

public interface Product extends Remote {
 int multiply(int a,int b) throws RemoteException;
}
```

Server Implementation:

```
import java.rmi.server.UnicastRemoteObject;
import java.rmi.RemoteException;

public class ProductImpl extends UnicastRemoteObject implements Product {
```

```
protected ProductImpl() throws RemoteException { super(); }
public int multiply(int a,int b) { return a*b; }
}
```

Server Program:

```
import java.rmi.registry.LocateRegistry;
import java.rmi.registry.Registry;

public class ProductServer {
 public static void main(String[] args) {
 try {
 ProductImpl obj = new ProductImpl();
 Registry reg = LocateRegistry.createRegistry(1099);
 reg.rebind("ProductService", obj);
 System.out.println("Server Ready... ");
 } catch(Exception e){ e.printStackTrace(); }
 }
}
```

Client Program:

```
import java.rmi.registry.LocateRegistry;
import java.rmi.registry.Registry;

public class ProductClient {
 public static void main(String[] args) {
 try {
 Registry reg = LocateRegistry.getRegistry("localhost",1099);
 Product p = (Product) reg.lookup("ProductService");
 System.out.println("Product of 5 and 7: " + p.multiply(5,7));
 } catch(Exception e){ e.printStackTrace(); }
 }
}
```

Output:

Product of 5 and 7: 35