Advanced Java Programming (4351603) - Winter 2023 Solution

Milav Dabgar

December 8, 2023

Question 1(a) [3 marks]

Draw and explain swing class hierarchy.

Solution

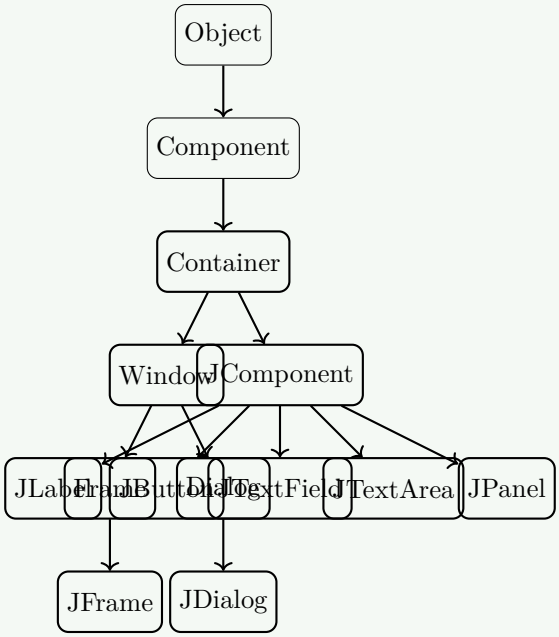

Figure 1. Swing Class Hierarchy

- **Component**: Base class for all GUI components.
- **Container**: Components that can hold other components.
- **JComponent**: Base class for all Swing components.

Mnemonic

"Objects Contain Components Jointly"

Question 1(b) [4 marks]

List out various Layout Managers. Explain Flow Layout manager with Example.

Solution

Table 1. Layout Managers

Layout Manager	Description
FlowLayout	Arranges components left to right
BorderLayout	Five regions: North, South, East, West, Center
GridLayout	Equal-sized rectangular grid
CardLayout	Stack of components
BoxLayout	Single row or column

FlowLayout Example:

```
JFrame frame = new JFrame();
frame.setLayout(new FlowLayout());
frame.add(new JButton("Button1"));
frame.add(new JButton("Button2"));
frame.setSize(300, 100);
frame.setVisible(true);
```

- **Default alignment**: Components flow left to right.
- **Wrapping**: Components wrap to next line when needed.

Mnemonic

"Flow Goes Left Right"

Question 1(c) [7 marks]

Develop a Java Swing program for a counter application having "Increment" and "Decrement" buttons with initial count of 0 displayed in the label. When "Increment" is clicked, the count increases by 1, and when "Decrement" is clicked, the count decreases by 1. A message dialog should be displayed when the counter goes below 0.

Solution

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

public class CounterApp extends JFrame implements ActionListener {
    private int count = 0;
    private JLabel countLabel;
    private JButton incButton, decButton;

    public CounterApp() {
        setTitle("Counter Application");
        setLayout(new FlowLayout());

        countLabel = new JLabel("Count: " + count);
        incButton = new JButton("Increment");
        decButton = new JButton("Decrement");

        incButton.addActionListener(this);
        decButton.addActionListener(this);

        add(countLabel);
```

```
        add(incButton);
        add(decButton);

        setSize(250, 100);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setVisible(true);
    }

    public void actionPerformed(ActionEvent e) {
        if(e.getSource() == incButton) {
            count++;
        } else if(e.getSource() == decButton) {
            count--;
            if(count < 0) {
                JOptionPane.showMessageDialog(this, "Counter below zero!");
            }
        }
        countLabel.setText("Count: " + count);
    }

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

- **Event handling**: ActionListener interface implementation.
- **Dialog display**: JOptionPane for negative counter warning.
- **Label update**: Real-time count display.

Mnemonic

"Increment Decrements Create Dialogs"

Question 1(c OR) [7 marks]

Create a Swing application with a menu bar containing a "File" menu having menu items "New", "Open", and "Exit". When the user clicks "Exit", the application should close. Add keyboard shortcuts for file menu items. Also add a "Help" menu having menu item "About". Display a message box which shows description of this program when 'About' is clicked.

Solution

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

public class MenuApp extends JFrame implements ActionListener {

    public MenuApp() {
        setTitle("Menu Application");

        JMenuBar menuBar = new JMenuBar();

        JMenu fileMenu = new JMenu("File");
        JMenuItem newItem = new JMenuItem("New");
        JMenuItem openItem = new JMenuItem("Open");
        JMenuItem exitItem = new JMenuItem("Exit");

```

```
        newItem.setAccelerator(KeyStroke.getKeyStroke(KeyEvent.VK_N, ActionEvent.CTRL_MASK));
        openItem.setAccelerator(KeyStroke.getKeyStroke(KeyEvent.VK_O, ActionEvent.CTRL_MASK));
        exitItem.setAccelerator(KeyStroke.getKeyStroke(KeyEvent.VK_X, ActionEvent.CTRL_MASK));

        newItem.addActionListener(this);
        openItem.addActionListener(this);
        exitItem.addActionListener(this);

        fileMenu.add(newItem);
        fileMenu.add(openItem);
        fileMenu.addSeparator();
        fileMenu.add(exitItem);

        JMenu helpMenu = new JMenu("Help");
        JMenuItem aboutItem = new JMenuItem("About");
        aboutItem.addActionListener(this);
        helpMenu.add(aboutItem);

        menuBar.add(fileMenu);
        menuBar.add(helpMenu);

        setJMenuBar(menuBar);
        setSize(400, 300);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setVisible(true);
    }

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

        if(command.equals("Exit")) {
            System.exit(0);
        } else if(command.equals("About")) {
            JOptionPane.showMessageDialog(this,
                "Menu Application v1.0\nDemonstrates Swing menus with shortcuts");
        }
    }

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

- **Keyboard shortcuts**: Ctrl+N, Ctrl+O, Ctrl+X accelerators.
- **Menu structure**: File and Help menus with separators.
- **About dialog**: Program description display.

Mnemonic

"Menus Need Shortcuts Always"

Question 2(a) [3 marks]

List out types of JDBC Drivers. Explain Type-4 Driver.

Solution

Table 2. JDBC Drivers

Type	Name	Description
Type-1	JDBC-ODBC Bridge	Uses ODBC driver
Type-2	Native-API Driver	Uses database native libraries
Type-3	Network Protocol Driver	Uses middleware server
Type-4	Thin Driver	Pure Java driver

Type-4 Driver Features:
- **Pure Java**: No native code required.
- **Direct communication**: Connects directly to database.
- **Platform independent**: Works on any OS with JVM.

Mnemonic

"Type Four: Pure Java Door"

Question 2(b) [4 marks]

Explain features of Java Foundation Classes (JFC).

Solution

JFC Components:
- **Swing**: Advanced GUI components.
- **AWT**: Basic GUI toolkit.
- **Accessibility**: Support for disabled users.
- **2D Graphics**: Enhanced drawing capabilities.
- **Drag and Drop**: File transfer support.

Key Features:
- **Pluggable Look and Feel**: Change UI appearance.
- **Lightweight components**: Better performance.
- **MVC architecture**: Separation of concerns.
- **Event handling**: Robust event system.

Mnemonic

"Java Foundation Creates Swing"

Question 2(c) [7 marks]

Draw and explain the architecture of Hibernate.

Solution

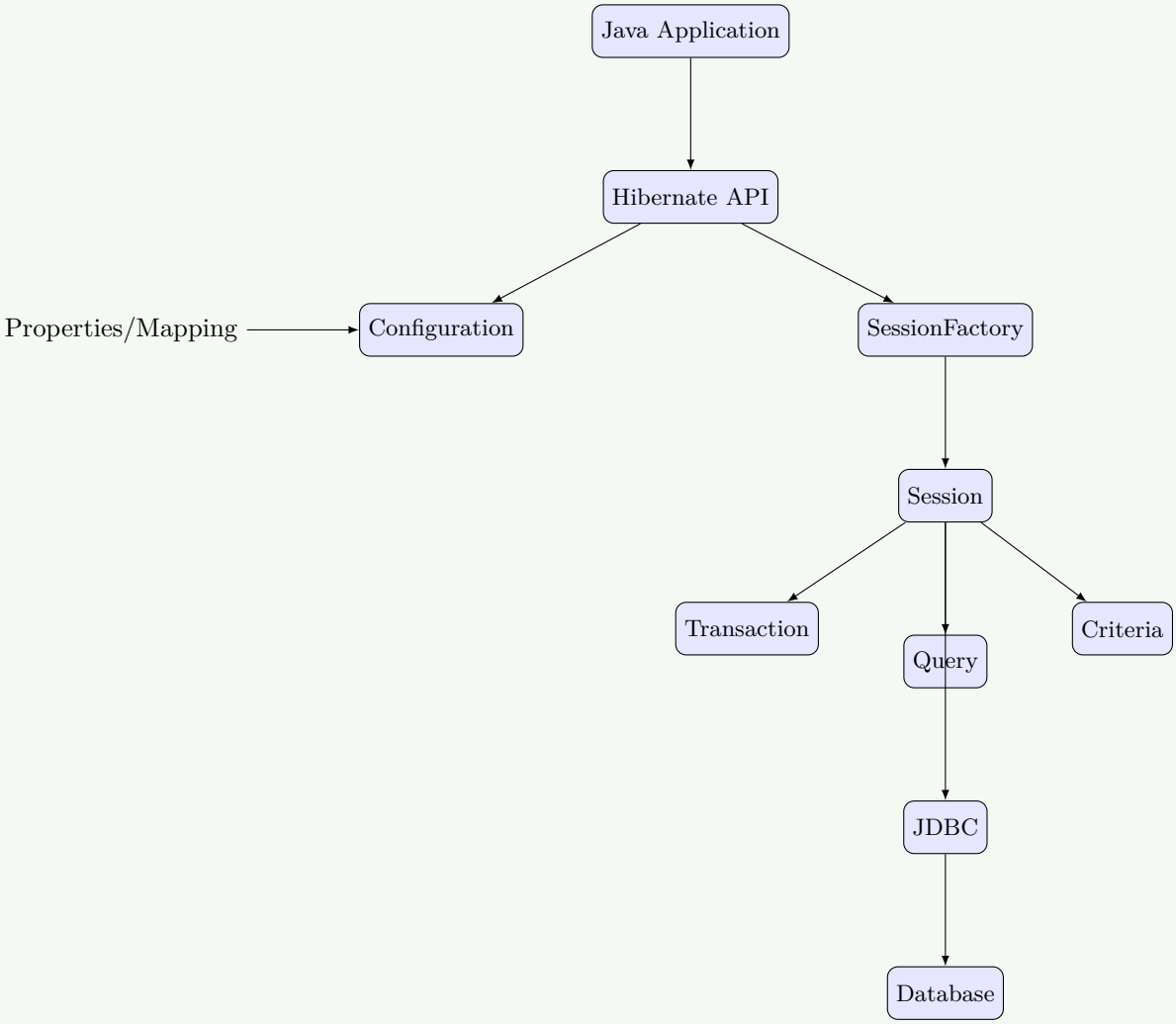

Figure 2. Hibernate Architecture

Architecture Components:

- **Configuration**: Reads mapping files and properties.
- **SessionFactory**: Factory for Session objects.
- **Session**: Interface between application and database.
- **Transaction**: Represents database transaction.
- **Query/Criteria**: For database queries.

Hibernate Benefits:

- **Object-Relational Mapping**: Maps Java objects to database tables.
- **Automatic SQL generation**: No manual SQL writing.
- **Caching**: First-level and second-level caching.
- **Lazy loading**: Load data only when needed.

Mnemonic

"Sessions Configure Factories Automatically"

Question 2(a OR) [3 marks]

Explain the components of the JDBC API.

Solution

JDBC API Components:

- **DriverManager**: Manages database drivers.
- **Connection**: Represents database connection.
- **Statement**: Executes SQL queries.
- **ResultSet**: Holds query results.
- **SQLException**: Handles SQL errors.

Component Functions:

- **Driver registration**: `DriverManager.registerDriver()`
- **Connection establishment**: `DriverManager.getConnection()`
- **Query execution**: `Statement.executeQuery()`

Mnemonic

"Drivers Connect Statements Returning Results"

Question 2(b OR) [4 marks]

Explain any two Swing controls with example.

Solution

JButton Control:

```
JButton button = new JButton("Click Me");
button.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        System.out.println("Button clicked!");
    }
});
```

JTextField Control:

```
JTextField textField = new JTextField(20);
textField.setText("Enter text here");
String text = textField.getText();
```

Features:

- **JButton**: Triggers actions when clicked.
- **JTextField**: Single-line text input field.
- **Event handling**: Both support ActionListener.

Mnemonic

"Buttons Text Fields Handle Events"

Question 2(c OR) [7 marks]

Write a Java program using JDBC to insert enrollment_number, name and age data into the 'student' table of the 'info' database using a prepared statement.

Solution

```
import java.sql.*;

```

```java
public class StudentInsert {
    public static void main(String[] args) {
        String url = "jdbc:mysql://localhost:3306/info";
        String username = "root";
        String password = "password";

        try {
            Class.forName("com.mysql.cj.jdbc.Driver");
            Connection conn = DriverManager.getConnection(url, username, password);

            String sql = "INSERT INTO student (enrollment_number, name, age) VALUES (?, ?, ?)";
            PreparedStatement pstmt = conn.prepareStatement(sql);

            pstmt.setString(1, "21IT001");
            pstmt.setString(2, "John Doe");
            pstmt.setInt(3, 20);

            int rowsAffected = pstmt.executeUpdate();
            System.out.println("Rows inserted: " + rowsAffected);

            pstmt.close();
            conn.close();
        } catch(Exception e) {
            System.out.println("Error: " + e.getMessage());
        }
    }
}
```

Key Components:

- **PreparedStatement**: Prevents SQL injection.
- **Parameter binding**: Using ? placeholders.
- **Connection management**: Proper resource cleanup.
- **Exception handling**: Try-catch for database errors.

Mnemonic

"Prepared Statements Prevent Problems"

Question 3(a) [3 marks]

Explain various features of Servlet.

Solution

Servlet Features:

- **Platform independent**: Runs on any OS with Java.
- **Server-side processing**: Executes on web server.
- **Protocol independent**: Not limited to HTTP.
- **Extensible**: Can be extended for specific needs.
- **Robust**: Built-in exception handling.

Additional Features:

- **Multithreading**: Handles multiple requests simultaneously.
- **Portable**: Write once, run anywhere.
- **Secure**: Java security features.

Mnemonic

"Servlets Process Protocols Portably"

Question 3(b) [4 marks]

Explain Servlet life cycle.

Solution

Table 3. Servlet Life Cycle Stages

Stage	Method	Description
Loading	-	Servlet class loaded by container
Instantiation	-	Servlet object created
Initialization	`init()`	Called once when servlet starts
Service	`service()`	Handles each client request
Destruction	`destroy()`	Called before servlet removal

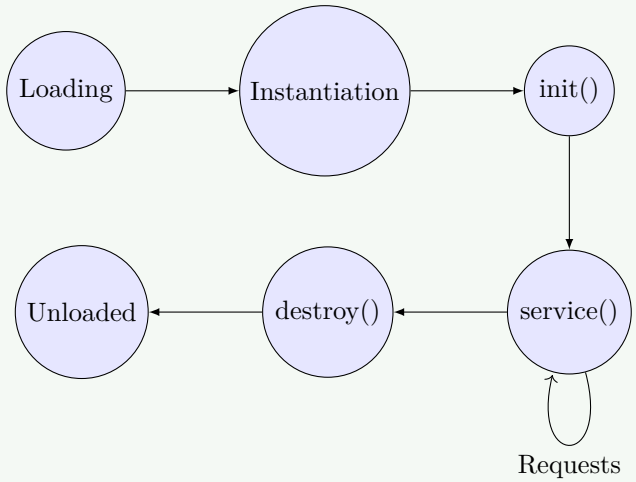

Figure 3. Servlet Life Cycle

Mnemonic

"Load Instance Initialize Service Destroy"

Question 3(c) [7 marks]

What is session? Write a Java servlet program demonstrating how session management can be achieved using HttpSession object, including the necessary HTML files.

Solution

Session Definition: Session is a way to store user-specific data across multiple HTTP requests. It maintains state between client and server.

Servlet Code:

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

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

public class SessionServlet extends HttpServlet {
    protected void doGet(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {

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

        HttpSession session = request.getSession(true);
        String name = request.getParameter("name");

        if(name != null) {
            session.setAttribute("username", name);
        }

        String username = (String)session.getAttribute("username");
        Integer visitCount = (Integer)session.getAttribute("visitCount");

        if(visitCount == null) {
            visitCount = 1;
        } else {
            visitCount++;
        }
        session.setAttribute("visitCount", visitCount);

        out.println("<html><body>");
        out.println("<h2>Session Demo</h2>");
        if(username != null) {
            out.println("<p>Welcome " + username + "!</p>");
        }
        out.println("<p>Visit count: " + visitCount + "</p>");
        out.println("<p>Session ID: " + session.getId() + "</p>");
        out.println("<a href='index.html'>Back to form</a>");
        out.println("</body></html>");
    }
}
```

HTML File (index.html):

```html
<!DOCTYPE html>
<html>
<head>
    <title>Session Demo</title>
</head>
<body>
    <h2>Enter Your Name</h2>
    <form action="SessionServlet" method="get">
        Name: <input type="text" name="name" required>
        <input type="submit" value="Submit">
    </form>
</body>
</html>
```

Session Management Features:

- **getAttribute/setAttribute**: Store and retrieve session data.
- **Session ID**: Unique identifier for each session.
- **Automatic creation**: Session created when needed.

Mnemonic

"Sessions Store State Safely"

Question 3(a OR) [3 marks]

Explain web.xml file in servlet.

Solution

web.xml Purpose: Web.xml is the deployment descriptor file that configures servlet mappings, parameters, and other web application settings.

Key Elements:

- **servlet**: Defines servlet configuration.
- **servlet-mapping**: Maps URL patterns to servlets.
- **init-param**: Servlet initialization parameters.
- **welcome-file-list**: Default pages.

Example Configuration:

```
<servlet>
    <servlet-name>MyServlet</servlet-name>
    <servlet-class>com.example.MyServlet</servlet-class>
</servlet>
<servlet-mapping>
    <servlet-name>MyServlet</servlet-name>
    <url-pattern>/myservlet</url-pattern>
</servlet-mapping>
```

Mnemonic

"Web XML Maps Servlets"

Question 3(b OR) [4 marks]

Explain advantages and disadvantages of servlets.

Solution

Table 4. Advantages vs Disadvantages

Advantages	Disadvantages
Platform independent: Java-based portability	**Java knowledge required**: Need Java skills
Performance: Faster than CGI scripts	**Mixing presentation**: HTML mixed with Java code
Robust: Exception handling	**Debugging complexity**: Server-side challenges
Secure: Java security features	**Limited design separation**: Logic and UI together

Mnemonic

"Performance Portability Presents Problems"

Question 3(c OR) [7 marks]

Write a Java servlet program for deleting a specific entry from the "student" table of 'info' database. The servlet should accept input student ID from an HTML form and delete the corresponding record from the database.

Solution

Servlet Code:

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

public class DeleteStudentServlet extends HttpServlet {
    protected void doPost(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {

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

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

        try {
            Class.forName("com.mysql.cj.jdbc.Driver");
            Connection conn = DriverManager.getConnection(
                "jdbc:mysql://localhost:3306/info", "root", "password");

            String sql = "DELETE FROM student WHERE id = ?";
            PreparedStatement pstmt = conn.prepareStatement(sql);
            pstmt.setString(1, studentId);

            int rowsDeleted = pstmt.executeUpdate();

            out.println("<html><body>");
            out.println("<h2>Delete Student Result</h2>");

            if(rowsDeleted > 0) {
                out.println("<p>Student with ID " + studentId + " deleted successfully!</p>");
            } else {
                out.println("<p>No student found with ID " + studentId + "</p>");
            }

            out.println("<a href='delete.html'>Delete another student</a>");
            out.println("</body></html>");

            pstmt.close();
            conn.close();

        } catch(Exception e) {
            out.println("<p>Error: " + e.getMessage() + "</p>");
        }
    }
}
```

HTML Form (delete.html):

```
<!DOCTYPE html>
<html>
<head>
    <title>Delete Student</title>
</head>
<body>
    <h2>Delete Student Record</h2>
    <form action="DeleteStudentServlet" method="post">
        Student ID: <input type="text" name="studentId" required>
        <input type="submit" value="Delete Student">
    </form>
</body>
```

```
</html>
```

Mnemonic

"Delete Database Data Dynamically"

Question 4(a) [3 marks]

Explain difference between JSP and Servlet.

Solution

Table 5. JSP vs Servlet

Aspect	JSP	Servlet
Code structure	HTML with Java code	Java with HTML output
Development	Easier for web designers	Better for Java developers
Compilation	Auto-compiled to servlet	Manual compilation needed
Maintenance	Easier to maintain	More complex maintenance
Performance	Slower first request	Faster execution

Mnemonic

"JSP Presents, Servlets Serve"

Question 4(b) [4 marks]

Explain life cycle of JSP.

Solution

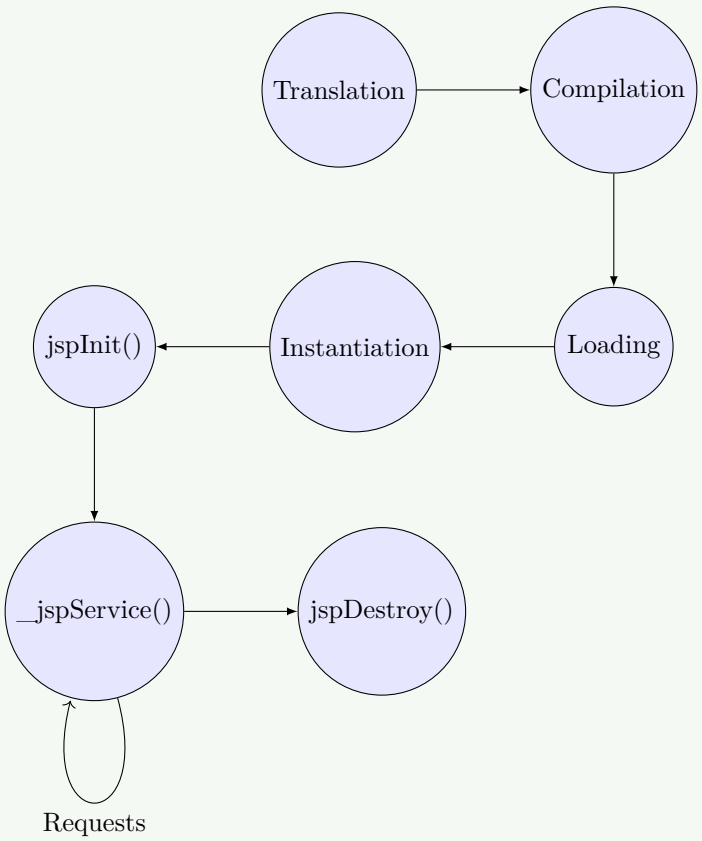

Figure 4. JSP Life Cycle

Phase Descriptions:

- **Translation**: JSP converted to servlet source code.
- **Compilation**: Servlet source compiled to bytecode.
- **Loading**: Servlet class loaded into memory.
- **Instantiation**: Servlet object created.
- **Initialization**: `jspInit()` method called once.
- **Service**: `_jspService()` handles each request.
- **Destruction**: `jspDestroy()` called before removal.

Mnemonic

"Translation Compiles Loading Instances Initialize Service Destroy"

Question 4(c) [7 marks]

Create a JSP program that acts as a simple calculator. The program should have a HTML form with two textboxes where users can input numbers and a dropdown menu to select an operation (addition, subtraction, multiplication, or division). When the user submits the form, the entered numbers and the chosen operation should be sent to the next page. On the next page, the program should calculate the result based on the chosen operation and display it.

Solution

HTML Form (calculator.html):

```html
<!DOCTYPE html>
<html>
<head><title>Simple Calculator</title></head>
<body>
    <h2>Simple Calculator</h2>
    <form action="calculate.jsp" method="post">
        <table>
            <tr><td>First Number:</td><td><input type="number" name="num1" required></td></tr>
            <tr><td>Second Number:</td><td><input type="number" name="num2" required></td></tr>
            <tr><td>Operation:</td>
                <td><select name="operation" required>
                        <option value="add">Addition (+)</option>
                        <option value="subtract">Subtraction (-)</option>
                        <option value="multiply">Multiplication (x)</option>
                        <option value="divide">Division (/)</option>
                    </select>
                </td>
            </tr>
            <tr><td colspan="2"><input type="submit" value="Calculate"></td></tr>
        </table>
    </form>
</body>
</html>
```

JSP Calculator (calculate.jsp):

```jsp
<%@ page language="java" contentType="text/html; charset=UTF-8" %>
<!DOCTYPE html>
<html>
<head><title>Result</title></head>
<body>
    <h2>Calculator Result</h2>
    <%
        String num1Str = request.getParameter("num1");
        String num2Str = request.getParameter("num2");
        String operation = request.getParameter("operation");

        double num1 = Double.parseDouble(num1Str);
        double num2 = Double.parseDouble(num2Str);
        double result = 0;
        String opSym = "";

        switch(operation) {
            case "add": result = num1 + num2; opSym = "+"; break;
            case "subtract": result = num1 - num2; opSym = "-"; break;
            case "multiply": result = num1 * num2; opSym = "*"; break;
            case "divide": result = num1 / num2; opSym = "/"; break;
        }
    %>
    <p>Result: <%= num1 %> <%= opSym %> <%= num2 %> = <%= result %></p>
    <a href="calculator.html">Back</a>
</body>
</html>
```

Mnemonic

"Calculate Add Subtract Multiply Divide"

Question 4(a OR) [3 marks]

Explain page directive in JSP.

Solution

Page Directive Purpose: Page directive provides instructions to JSP container about page configuration and processing.
Syntax: `<%@ page attribute="value" %>`
Common Attributes:

- **language**: Scripting language (default: java).
- **contentType**: MIME type and character encoding.
- **import**: Java packages to import.
- **session**: Enable/disable session.
- **errorPage**: Error handling page URL.

Mnemonic

"Page Directives Direct Processing"

Question 4(b OR) [4 marks]

Explain JSP declaration tag with example.

Solution

JSP Declaration Tag: Declaration tag is used to declare variables, methods, and classes that become part of the servlet class.
Syntax: `<%! declaration code %>`
Example:

```
<%!
    int counter = 0;

    public String getCurrentTime() {
        return new java.util.Date().toString();
    }
%>

<html><body>
    <p>Count: <%= ++counter %></p>
    <p>Time: <%= getCurrentTime() %></p>
</body></html>
```

Key Points:

- **Class-level scope**: Variables are instance variables.
- **Method declaration**: Can declare methods and classes.
- **Thread safety**: Need to handle concurrent access.

Mnemonic

"Declarations Define Class Data"

Question 4(c OR) [7 marks]

What is cookie? Write a JSP program demonstrating how session management can be

achieved using cookies, including the necessary HTML files.

Solution

Cookie Definition: Cookie is a small piece of data stored on the client-side browser to maintain state between HTTP requests.

HTML Form (login.html):

```
<form action="setCookie.jsp" method="post">
    Username: <input type="text" name="username"><br>
    <input type="submit" value="Login">
</form>
```

Set Cookie JSP (setCookie.jsp):

```
<%
    String username = request.getParameter("username");
    if(username != null) {
        Cookie c = new Cookie("user", username);
        c.setMaxAge(60*60*24); // 1 day
        response.addCookie(c);
    }
%>
<a href="welcome.jsp">Go to Welcome</a>
```

Welcome Page JSP (welcome.jsp):

```
<%
    Cookie[] cookies = request.getCookies();
    String user = "Guest";
    if(cookies != null) {
        for(Cookie c : cookies) {
            if("user".equals(c.getName())) user = c.getValue();
        }
    }
%>
<h1>Welcome, <%= user %></h1>
```

Features:

- **Client-side storage**: Data stored in browser.
- **Persistence**: Can survive browser sessions.

Mnemonic

"Cookies Create Client Cache"

Question 5(a) [3 marks]

Compare Spring and Spring Boot.

Solution

Table 6. Spring vs Spring Boot

Feature	Spring Framework	Spring Boot
Configuration	XML/Annotation based	Auto-configuration
Setup time	More time required	Quick setup
Dependency	Manual dependency	Starter dependencies
Server	External server needed	Built-in Tomcat/Jetty

Key Differences:
- **Spring Boot**: Opinionated framework with defaults.
- **Spring Framework**: Flexible but requires more setup.

Mnemonic

"Boot Builds Better Beginnings"

Question 5(b) [4 marks]

List out all Implicit object in JSP and explain any two.

Solution

JSP Implicit Objects List:
- **request**, **response**, **session**, **application**
- **out**, **page**, **pageContext**, **config**, **exception**

Explanation:
- **request**: The `HttpServletRequest` object containing client request data (parameters, headers).
- **session**: The `HttpSession` object for storing user-specific data across requests.

Mnemonic

"Request Response Session Application Out"

Question 5(c) [7 marks]

Explain MVC architecture.

Solution

Figure 5. MVC Architecture

MVC Components:
- **Model**: Data and business logic. Interacts with database.
- **View**: User interface (JSP, HTML). Displays data to user.
- **Controller**: Handles requests, process data with Model, selects View.

Benefits: separation of concerns, maintainability, testability.

Mnemonic

"Models View Controllers Separate"

Question 5(a OR) [3 marks]

Explain Dependency Injection.

Solution

Definition: Design pattern where dependencies are provided to an object rather than created by it.
Types of DI:

- **Constructor Injection**: Passed via constructor.
- **Setter Injection**: Passed via setter methods.
- **Field Injection**: Injected directly into fields.

Example (Constructor DI):

```
public class UserService {
    private UserRepository repo;
    public UserService(UserRepository repo) {
        this.repo = repo;
    }
}
```

Mnemonic

"Dependencies Injected, Not Instantiated"

Question 5(b OR) [4 marks]

List the JSTL core tags and explain any two with example.

Solution

JSTL Core Tags: `c:out`, `c:set`, `c:if`, `c:choose`, `c:forEach`, `c:url`, `c:redirect`, `c:import`.
Examples:

1. **c:out**: Displays value. `<c:out value="${data}" />`
2. **c:if**: Conditional execution.

```
    <c:if test="${age >= 18}">
        <p>Adult</p>
    </c:if>

```

Mnemonic

"Core Tags Control Conditions"

Question 5(c OR) [7 marks]

Explain the architecture of Spring framework.

Solution

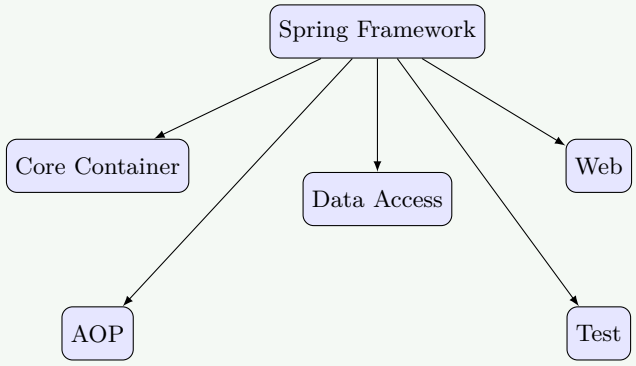

Figure 6. Spring Architecture

Modules:

- **Core Container**: Beans, Core, Context, SpEL. IoC container.
- **Data Access**: JDBC, ORM, OXM, JMS, Transactions.
- **Web**: Web-MVC, WebSocket, Web-Portlet.
- **AOP**: Aspect Oriented Programming support.
- **Test**: Unit and integration testing support.

Mnemonic

"Spring's Architecture Supports Complete Applications"