MIT-ADT UNIVERSITY
PUNE, INDIA
A leap towards World Class Education
(Established by MIT Art, Design and Technology University Act, 2015
(Maharashtra Act No. XXXIX of 2015)

MIT Art Design and Technology University

MIT School of Computing, Pune

Department of Computer Science and Engineering

Lab Manual

Subject - Advanced Java Programming Laboratory

Class - SY. (SEM-II)

Name of the Course Coordinator

Prof. Rahul More

Team Members

1. Prof. Rohidas Sangore
2. Prof. Sneha Deshmukh
3. Prof. Dr. Sumit Hirve
4. Prof. Dr. Anuja Jadhav
5. Prof. Dr. Umang Garg
6. Prof. Dr. Anant Kaulage
7. Prof. Dr. Mayura Shelke
8. Prof. Yuvraj Ganpat Nikam
9. Prof. Swapnil Patil
10. Prof. Deepali Lokare
11. Prof. Kanchan Wankhade
12. Prof. Pratik Kamble
13. Prof. Amol Dande
14. Prof. Komal Munde
15. Prof. Khushal Kunjir
16. Prof. Sneha Singha

Assignment No. 1

Assignment Title: Create a Java program using JDBC (Java Database Connectivity) to connect to a database, retrieve data from a table, and display the results.

Aim: Let's consider a scenario where we have a MySQL database named "example_db" with a table named "employees". The "employees" table has three columns: "id", "name", and "salary". We want to retrieve all records from this table and display them using JDBC in a Java program.

Pre-Requisites:

- Basics of Java programming.
- Understanding of SQL queries (SELECT, CREATE, INSERT).
- Familiarity with database concepts and table structures.

Objective:

1. Establish a connection with a MySQL database.
2. Execute SQL queries to retrieve data from the "employees" table.
3. Display the retrieved data in a readable format in the console.

Outcomes:

1. Understand how to use JDBC to connect to a database.
2. Learn how to execute SQL queries from Java code.
3. Gain experience handling database operations like connection, statement creation, and result processing.
4. Be able to manage resources efficiently using try-catch-finally blocks.
5. Appreciate the integration of Java applications with relational databases like MySQL.

Theory:

What is JDBC?

JDBC (Java Database Connectivity) is an API provided by Java to interact with relational databases. It acts as a bridge between Java applications and databases by providing a standard interface for database operations.

Key Components of JDBC:

1. DriverManager: Manages database drivers and establishes connections to databases.
2. Connection: Represents an active connection to the database.
3. Statement/PreparedStatement: Used to execute SQL queries.
4. ResultSet: Stores the result of a query.
5. SQLException: Handles errors or issues during database operations.

1. DriverManager

- What It Is:
  The DriverManager class is responsible for managing a list of database drivers. It provides methods to register drivers and establish connections to a database.
- Responsibilities:
  - Manages multiple database drivers for various databases (e.g., MySQL, Oracle, PostgreSQL).
  - Matches the requested connection URL with an appropriate driver.
- How It Works:
  - When you call DriverManager.getConnection(), it checks all registered drivers to find one that matches the database URL.
  - Once a suitable driver is found, a connection to the database is established.
- Code Example:

Connection connection
= DriverManager.getConnection("jdbc:mysql://localhost:3306/example_db", "username", "password");

2. Connection

- What It Is:
  The Connection interface represents an active connection between a Java application and the database. It is the backbone of all database operations.
- Responsibilities:
  - Enables communication with the database.
  - Provides methods to create Statement and PreparedStatement objects.
  - Manages transactions (e.g., commit, rollback).
  - Allows for setting connection properties (e.g., auto-commit mode).
- Code Example:

Connection connection =

DriverManager.getConnection("jdbc:mysql://localhost:3306/example_db", "root", "password");

connection.setAutoCommit(false); // Example of managing transactions.

Important Note: Always close the connection after use to release resources:

connection.close();

3. Statement and PreparedStatement

- Statement:
    - What It Is:
      The Statement interface is used to execute static SQL queries against the database.
    - Responsibilities:
        - Sends SQL commands to the database.
        - Retrieves results for queries like SELECT.
    - Limitations:
        - Vulnerable to SQL injection if user inputs are not validated.
    - Code Example:

Statement statement = connection.createStatement();

ResultSet resultSet = statement.executeQuery("SELECT * FROM employees");

PreparedStatement:

- What It Is:
  A subclass of Statement that allows parameterized SQL queries, improving security and performance.
- Advantages:
    - Prevents SQL injection.
    - Optimized by the database (reusable precompiled queries).
- Code Example

PreparedStatement preparedStatement = connection.prepareStatement("SELECT * FROM employees WHERE id = ?");

preparedStatement.setInt(1, 101);

ResultSet resultSet = preparedStatement.executeQuery();

4. ResultSet

- What It Is:
  The ResultSet interface represents the data retrieved from the database. It is essentially a table of rows and columns.
- Responsibilities:
  - Iterates through the results of a query.
  - Provides methods to retrieve data by column index or name.
  - Supports moving the cursor through rows (e.g., next(), previous()).
- Code Example:

```
ResultSet resultSet = statement.executeQuery("SELECT * FROM employees");

while (resultSet.next()) {

   int id = resultSet.getInt("id");

   String name = resultSet.getString("name");

   double salary = resultSet.getDouble("salary");

   System.out.println("ID: " + id + ", Name: " + name + ", Salary: " + salary);

}
```

Important Methods:

- next(): Moves to the next row.
- getString(), getInt(), etc.: Retrieves data from a specific column.
- close(): Frees up resources associated with the ResultSet.

5. SQLException

- What It Is:
  SQLException is an exception class in Java that handles database-related errors during runtime.
- Responsibilities:
  - Indicates issues like connection failures, incorrect SQL syntax, or access violations.
  - Provides detailed information about the error through methods like getMessage(), getErrorCode(), and getSQLState().
- How to Handle: Always use try-catch blocks to handle SQLException gracefully.

```
try {

   Connection connection =
DriverManager.getConnection("jdbc:mysql://localhost:3306/example_db", "root", "password");
```

```
    Statement statement = connection.createStatement();
    ResultSet resultSet = statement.executeQuery("SELECT * FROM employees");
} catch (SQLException e) {
    System.out.println("Error: " + e.getMessage());
    e.printStackTrace();
}
```

Steps to Use JDBC:

1. Load the JDBC driver.
2. Establish a connection to the database.
3. Create a statement object.
4. Execute SQL queries and retrieve results.
5. Process the ResultSet data.
6. Close the connection to release resources.

Conclusion:

This assignment illustrates the practical implementation of Java Database Connectivity (JDBC). By retrieving and displaying data from the "employees" table, you've learned how to interact with databases through Java.

Frequently Asked Questions:

1. What are the main components of a JDBC program, and how do they work together?
2. What is the role of the JDBC URL, and how should it be formatted?
3. What is the role of DriverManager in JDBC, and how does it interact with the database driver? Give syntax.
4. What are some real-world use cases of JDBC in software applications? Explain anyone in detail.
5. What is the difference between executeQuery(), executeUpdate(), and execute() in JDBC? When should each method be used?

Assignment No. 2

Assignment Title: Create a Java GUI application to calculate the simple interest based on user input for principal amount, rate of interest, and time period (JAVA GUI).

Aim: To develop a Java GUI application using Swing to calculate the simple interest based on user input for Principal Amount, Rate of Interest, and Time Period, and to demonstrate the use of Swing components, event handling, and validation in creating interactive and user-friendly desktop applications.

Pre-Requisites:

1. Basic Java Programming
2. Mathematical Concepts
3. Java Swing Basics
4. Event Handling in Java
5. IDE Setup
6. Error Handling

Objective:

1. To design and develop a Java GUI application using the Swing framework.
2. To enable users to input values for Principal Amount, Rate of Interest, and Time Period and compute the Simple Interest using the formula: SI=P×R×T100SI = \frac{P \times R \times T}{100}SI=100P×R×T
3. To demonstrate the practical use of Swing components such as JFrame, JLabel, JTextField, and JButton to build an interactive and user-friendly interface.
4. To implement event handling through the ActionListener interface for button click actions.
5. To showcase the application of error handling in validating user inputs and providing appropriate feedback using JOptionPane.
6. To enhance problem-solving and programming skills by integrating mathematical computations into GUI applications.

Outcomes:

1. Design and develop a GUI application using Java Swing to solve real-world problems, such as calculating financial metrics.
2. Apply Swing components like JFrame, JLabel, JTextField, and JButton effectively in creating interactive user interfaces.

3. Implement event handling using the ActionListener interface to respond to user actions such as button clicks.
4. Validate user inputs to ensure error-free calculations and provide feedback for invalid entries.
5. Perform mathematical computations programmatically, such as calculating simple interest.
6. Enhance debugging and error-handling skills in Java to build robust applications.
7. Gain practical experience in Java GUI programming, improving confidence in creating standalone desktop applications.

Theory:

The Simple Interest Calculator is a graphical user interface (GUI) application developed using Java Swing. It demonstrates how to create an interactive application to perform financial calculations, focusing on both user experience and functionality.

1. Simple Interest Formula

The formula to calculate simple interest is:

SI=P×R×T100SI = \frac{P \times R \times T}{100}SI=100P×R×T

Where:

- PPP: Principal amount (initial sum of money)
- RRR: Annual rate of interest (percentage)
- TTT: Time period in years

2. Java Swing Overview

Swing is a part of Java's Standard Library used to create GUI-based applications. It is part of the Java Foundation Classes (JFC) and provides:

- A rich set of components like JLabel, JButton, JTextField, etc.
- Lightweight components that do not depend on the underlying OS.
- Pluggable look and feel, enabling customization of the UI.

3. Components Used in the Application

1. JFrame: The top-level container to create the main application window.
2. JLabel: Displays text to describe input fields.
3. JTextField: Allows users to input data such as principal, rate, and time.
4. JButton: Triggers specific actions such as calculating or clearing inputs.
5. JOptionPane: Displays dialog boxes for error messages or additional information.

4. Event Handling in Swing

- ActionListener Interface:
  This interface handles user interactions with components like buttons. For example, when the "Calculate" button is clicked, an ActionEvent is triggered, and the listener executes the associated logic.

5. Input Validation

- Validates that the user inputs numeric values for principal, rate, and time.
- Displays an error message for invalid inputs using JOptionPane.

6. Error Handling

- Ensures robust application behavior by catching NumberFormatException for invalid user inputs.
- Prevents crashes and guides the user to enter valid data.

7. Workflow of the Application

1. The user inputs values for the principal amount, rate of interest, and time period in text fields.
2. Upon clicking the "Calculate" button:
   - Inputs are validated.
   - Simple interest is computed using the formula.
   - The result is displayed in a read-only text field.
3. The "Clear" button resets all fields to allow new inputs.

8. Practical Applications

This application showcases the use of Java Swing for financial calculations and can be extended to include:

- Compound interest calculations.
- Exporting results to files.
- Advanced GUI features like graphs and charts.

Algorithm/Steps:

1. Set Up Environment: Install and configure the JDK.
2. Code Implementation:
   - Create a JFrame with GridLayout.
   - Add labels and text fields for user input.
   - Add buttons for calculation and clearing fields.
   - Implement the ActionListener to handle button clicks.
3. Test the Application:

- Run the application.
- Input valid and invalid values to ensure functionality.

Conclusion:

A Java GUI application for calculating simple interest provides an interactive and user-friendly way to compute financial results based on user input.

Frequently Asked Questions:

1. What is Java Swing?
2. What are the key features of Java Swing?
3. How is Swing different from AWT?
4. What are some commonly used Swing components?
5. What is the MVC architecture in Swing?

Assignment No. 3

Assignment Title: Create a Java servlet that takes input from a web form, calculates the area of a circle based on the provided radius, and displays the result on a web page.

Aim: To create a Java servlet that takes user input for the radius of a circle from a web form, calculates the area, and displays the result on a web page.

Pre-Requisites:

1. Basic understanding of Java and servlets.
2. Familiarity with HTML forms and HTTP request/response mechanism.
3. Java Development Kit (JDK) installed.
4. A servlet container like Apache Tomcat.
5. Integrated Development Environment (IDE) like Eclipse.

Objective:

1. To develop a servlet that accepts user input (radius of the circle).
2. To compute the area of the circle using the formula: Area = π * radius^2.
3. To display the result dynamically on a web page.

Outcomes:

1. A functional Java servlet that performs calculations based on user input.
2. Understanding of how Java servlets interact with HTML forms and the web browser.

Theory:

1. **Client-Side (HTML Form):**
   - An HTML form collects the user input. The form contains a text field for the user to enter the radius of the circle and a submit button to send the data to the server.
   - The user's input is sent to the server through an HTTP request, usually a POST request, which passes the input data as parameters to the servlet.
2. **Server-Side (Java Servlet):**
   - The Java servlet receives the input data from the form through the HTTP request. It then reads the radius value, performs the calculation, and generates the appropriate response.
   - The servlet uses Java's Math.PI constant to represent the value of π and computes the area using the formula mentioned above.
   - The servlet then creates an HTTP response, which includes an HTML page

displaying the calculated area to the user.

3. **Servlet Lifecycle:**
    - When the user submits the form, the web server invokes the servlet's doPost method (if using POST) or doGet method (if using GET), depending on the form submission method.
    - Inside the method, the servlet reads the form data, processes it, and sends back an HTML response with the calculated result.
4. **Communication Between Client and Server:**
    - The client (browser) sends an HTTP request to the servlet, and the servlet processes the request and generates an HTTP response. The response typically contains dynamic content, such as the area calculation result.
    - This process allows the user to interact with the application in real-time through a web page.

Key Concepts:

- HTTP Request and Response: Servlets handle HTTP requests (from the client) and send HTTP responses back (to the client). The request contains data sent by the user, and the response contains the result (in this case, the area of the circle).
- Form Handling: Web forms are used to collect input from the user. This input is sent as part of the HTTP request when the form is submitted.
- Servlets and HTTP Methods: Servlets use methods like doGet() and doPost() to handle the incoming HTTP request. doPost() is typically used for handling form submissions as it allows data to be sent in the request body (more secure for sensitive data).
- Dynamic Content Generation: The servlet dynamically generates HTML content based on the calculation results, allowing users to view the updated information on the webpage.

Benefits of Using Servlets:

- Platform Independence: Since servlets are written in Java, they can run on any platform that supports Java.
- Scalability: Servlets can handle multiple requests concurrently and can be easily scaled in a web application environment.
- Extensibility: Servlets can be extended to perform various tasks, making them reusable for other calculations or functionalities beyond this simple example.

Algorithm/Steps:

1. Start

   Initialize the servlet environment and set up necessary configurations such as the servlet

container (e.g., Apache Tomcat)

2. Create an HTML Form

- Design an HTML form that includes:
  - A text input field to enter the radius of the circle.
  - A submit button to send the form data to the servlet.

3. Submit the Form

- When the user enters the radius and clicks the submit button, the form sends an HTTP request (usually a POST request) to the servlet.
- The radius value entered by the user is sent as a form parameter to the servlet.

4. Create a Java Servlet

- Create a servlet class (e.g., CircleAreaServlet) that will handle the POST request from the form.
- The servlet should extend HttpServlet and override the doPost() method (or doGet() if using GET).

5. Retrieve Form Input from the HTTP Request

- Inside the doPost() method of the servlet, retrieve the value of the radius from the form data.
  - Use request.getParameter("radius") to get the input value.

6. Validate Input

- Check if the radius is a valid, positive number.
- If the input is invalid (e.g., a negative or non-numeric value), display an error message to the user.

7. Calculate the Area of the Circle

- If the input is valid, calculate the area of the circle using the formula: Area=π×(radius)2\text{Area} = \pi \times (\text{radius})^2Area=π×(radius)2
- Use Math.PI for the value of π in the formula.

8. Generate the HTML Response

- After performing the calculation, generate an HTML response that includes the calculated area and display it on the webpage.
- The response should also contain information like the radius entered by the user and the result of the calculation.

9. Send Response to the Client

- Send the HTML response back to the client (browser) to display the result.

Conclusion:

This assignment demonstrates the use of Servlet to build a simple web application for performing mathematical calculations.

Frequently Asked Questions

1. What is a Java Servlet?
2. How do servlets handle user input from web forms?
3. What is required to run a servlet?
4. How do I deploy the servlet?
5. How do I handle invalid input, like text instead of a number?

Assignment No. 4

Assignment Title: Java web application using JSP (Java Server Pages).

Aim: Create a Java web application using JSP (Java Server Pages) that takes input from a user, calculates the factorial of the provided number, and displays the result on a web page.

Pre-Requisites:

1. Understanding of core Java concepts like loops, functions, and data types.
2. Knowledge of Java Server Pages (JSP) and Servlets.
3. Familiarity with setting up and deploying a Java web application using an IDE like Eclipse/NetBeans or a server like Apache Tomcat.
4. Basic knowledge of HTML for creating forms.
5. Understanding of deployment descriptors (web.xml).

Objective:

1. To develop a dynamic web application using JSP to accept user input, process it on the server side, and dynamically display the result on a web page.

Outcomes:

1. Ability to create a web application using JSP.
2. Practical understanding of form handling in JSP.
3. Implementing server-side business logic in a web application.
4. Skills in deploying and testing web applications on a local server.

Theory:

Java Server Pages (JSP):

JSP is a server-side technology used to create dynamic web content. It allows developers to embed Java code directly into HTML, enabling the creation of web applications that respond to user input and dynamically generate content. JSP is an integral part of the Java EE platform and is built on top of Servlets, which makes it more convenient for web development. It provides features like tag libraries, expression language, and script lets, which simplify the development process and improve productivity.

In a typical web application, JSP pages act as the view layer in the MVC (Model-View-Controller) architecture. The client sends a request through the browser, the server processes the request, and

JSP renders the response dynamically based on the server-side logic.

Factorial Calculation:

The factorial of a non-negative integer n is the product of all positive integers less than or equal to n. It is denoted as n! For example:

$5! = 5 \times 4 \times 3 \times 2 \times 1 = 120$

Factorial has numerous applications in mathematics, computer science, and statistics, including permutations, combinations, and probability calculations. It is defined recursively, where:

$n! = n \times (n-1)!$ AND $0! = 1$

In this application, we calculate the factorial of a number provided by the user using Java logic embedded in a JSP page.

Form Handling in JSP:

Forms in web development are used to collect input from users. In JSP, we use HTML forms to collect data, and the server processes the input through JSP or Servlets. The data from the form is sent to the server using the HTTP methods:

- **GET Method:** The form data is appended to the URL as a query string. It is visible to the user and has length restrictions.
- **POST Method:** The form data is sent in the request body, making it secure and suitable for larger data.

In this assignment, the input number is collected through a form in index.jsp and sent to result.jsp using the POST method. The request.getParameter("parameterName") method is used to retrieve the submitted value.

Lifecycle of a JSP Page:

The lifecycle of a JSP page ensures it is executed efficiently and consistently. It includes the following steps:

1. **Translation Phase:** The JSP file is converted into a Servlet class.
2. **Compilation Phase:** The Servlet class is compiled into a .class file.
3. **Initialization Phase:** The Servlet is initialized and prepared to handle requests.
4. **Execution Phase:** The JSP page processes user requests and dynamically generates responses.
5. **Destruction Phase:** The Servlet is removed from memory when no longer needed.

This lifecycle ensures that JSP pages are compiled and optimized for performance during execution.

Advantages of JSP Over Static HTML:

- **Dynamic Content Generation:** Unlike static HTML, JSP can create web pages that change based on user input or server-side logic.
- **Integration with Java:** JSP allows seamless integration of Java logic with web design, providing powerful functionality.
- **Ease of Maintenance:** Changes can be made directly to JSP pages without recompiling the entire application.
- **Separation of Concerns:** JSP allows developers to separate business logic from presentation using JavaBeans or Servlets.

Backend Processing in JSP:

JSP allows embedding Java code within special tags:

- **Scriptlets (<% %>):** Contain Java code for processing logic.
- **Declarations (<%! %>):** Define methods and variables.
- **Expressions (<%= %>):** Output the result of an expression directly to the client.

For example, in this project, the factorial of a number is calculated using a for loop in a scriptlet block, and the result is displayed using an expression block.

JSP and Web Servers:

JSP pages run on a web server, such as Apache Tomcat, which interprets the JSP code, executes the embedded Java logic, and serves the generated HTML content to the client's browser. The server acts as a bridge between the client (browser) and the application logic. By understanding these fundamental concepts, you can create a dynamic web application like the factorial calculator with JSP.

Algorithm/Steps:

Setup the Environment:

- Install Apache Tomcat and set up a dynamic web project in an IDE like Eclipse.

Design the Web Form:

- Create an HTML form in index.jsp to accept a number input from the user.

Capture User Input:

- Use the request.getParameter() method in JSP to retrieve the input.

Process the Input:

- Write a JSP scriptlet to calculate the factorial of the given number.

Display the Output:

- Use JSP expressions to display the result dynamically on the web page.

Deploy and Test the Application:

- Deploy the application on Apache Tomcat and test it by providing input through the browser.

Conclusion:

This assignment demonstrates the use of JSP to build a simple web application for performing mathematical calculations. It enhances the understanding of form handling, server-side logic processing, and dynamic content generation.

Frequently Asked Questions:

1. What is JSP used for in a web application?
2. Why is JSP better than static HTML?
3. How does JSP handle user input?

Assignment No. 5

Assignment Title: Implementing a Simple Model-View-Controller (MVC) Architecture for an Online Bookstore

Aim: Implement a simple Model-View-Controller (MVC) architecture for a Java application that simulates a basic online bookstore. The application should allow users to view a list of available books, add books to their cart, and view the contents of their cart.

Pre-Requisites:

1. Familiarity with the MVC architecture.
2. Knowledge of Java Collections Framework (e.g., ArrayList).

Objective:

1. To demonstrate the separation of concerns in software design by implementing the MVC architecture.
2. To create a functional application that allows users to interact with a simulated online bookstore.
3. To apply Java concepts like collections, encapsulation, and event handling effectively.

Outcomes:

1. Successfully implement a Java application with clear separation of the Model, View, and Controller.
2. Gain practical experience in building structured and maintainable code.
3. Develop skills to handle user inputs and manipulate data dynamically.

Theory:

The Model-View-Controller (MVC) architecture is a design pattern used for developing user interfaces. It separates the application into three interconnected components:

1. **Model**: Represents the application's data and business logic. It manages the state of the application and communicates directly with the Controller.
2. **View**: Displays the data (from the Model) and sends user interaction inputs to the Controller.
3. **Controller**: Acts as an intermediary between the Model and the View. It processes user inputs, updates the Model, and refreshes the View.

For this application, the **Model** will store the list of books and the cart, the **View** will handle displaying data and receiving input, and the **Controller** will manage the application's flow.

Algorithm/Steps:

- **Model**:
    - Create classes for Book and Cart.

- Define attributes like title, author, and price for Book.
- Implement methods to add, retrieve, and manipulate books and cart data.

- **View**:
  - Create methods to display the list of books and the cart's contents.
  - Provide methods to capture user input (e.g., book selection or action commands).
- **Controller**:
  - Implement logic to handle user inputs and perform actions like adding a book to the cart or displaying the cart's contents.
  - Ensure proper communication between the Model and View.
- **Main Application**:
  - Initialize the Model, View, and Controller.
  - Start the application loop to interact with the user.

Conclusion:

The application demonstrates the effective use of the MVC architecture in Java. By separating concerns, the code is modular, maintainable, and easy to extend for future functionalities like payment systems or user authentication.

Frequently Asked Questions:

1. What is the purpose of the MVC architecture?
2. Why use an ArrayList for storing books?
3. If this application were to be deployed as a web-based system, explain how the MVC components would change or adapt. What additional technologies or frameworks would be needed?
4. Analyze the advantages and limitations of using the MVC architecture in this project. Under what circumstances might MVC not be the best choice for a small-scale application like this one?

Assignment No. 6

Assignment Title: Write a JavaFX application.

Aim:

A. Write a JavaFX application that displays a "Hello, JavaFX!" message in a window. Your system's Java and JavaFX versions are displayed as well.

B. Write a JavaFX application with multiple components (buttons, labels, text fields) arranged in a VBox or HBox layout.

C. Write a JavaFX application that creates a basic calculator application that can add, subtract, multiply, and divide two numbers entered by the user.

Pre-Requisites:

1. **Basic Knowledge of Java**: Familiarity with Java syntax, OOP concepts (inheritance, polymorphism, etc.), and fundamental programming constructs.
2. **Understanding of GUI Development**: Prior experience with any GUI framework, such as Swing or AWT, is beneficial but not mandatory.
3. **JavaFX Environment Setup**: Ensure JavaFX is installed and configured in your IDE (e.g., IntelliJ IDEA, Eclipse). You will need the JavaFX SDK and its libraries included in your project's build path.

Objective :

1. Develop a **JavaFX application** that organizes and presents structured content in an interactive and user-friendly interface.
2. Enhance understanding of JavaFX components, layout managers, and event-driven programming.
3. Learn how to design a simple application that can serve as a template for more complex projects.

Outcomes:

1. Create basic and interactive JavaFX applications.
2. Use JavaFX components like TabPane, TextArea, VBox, etc., effectively.
3. Understand how to structure and organize an application to make it modular and maintainable.
4. Learn how to handle events in JavaFX and integrate business logic with the UI.
5. Apply the knowledge gained to build more advanced Java desktop applications.

Theory:

Understanding JavaFX

JavaFX is a modern Java library used for developing rich, cross-platform desktop applications. It is designed to replace older GUI frameworks like Swing and Abstract Window Toolkit (AWT). JavaFX provides developers with a wide range of tools and features to create visually appealing, interactive, and dynamic user interfaces.

Key Features of JavaFX:

1. **Rich Set of UI Controls**:
    - JavaFX includes built-in components such as buttons, labels, tables, trees, and more. These controls can be styled and customized easily.
2. **CSS Styling**:
    - JavaFX supports CSS for styling UI components, making it easier to create visually consistent designs.
3. **FXML Support**:
    - FXML is an XML-based markup language used to define the structure of the UI separately from the application logic.
4. **Scene Graph**:
    - The foundation of JavaFX is its **scene graph**, a hierarchical structure of nodes that represent the UI components and their relationships.
5. **Media and Animation**:
    - JavaFX has built-in support for audio, video, and animation, enabling developers to create rich multimedia applications.
6. **Platform Independence**:
    - Applications built with JavaFX run on multiple platforms, including Windows, macOS, and Linux.
7. **Event-Driven Programming**:
    - JavaFX uses an event-handling mechanism that allows developers to define user interactions through event listeners.
8. **Integration with Java**:
    - JavaFX integrates seamlessly with Java code, allowing developers to use existing Java libraries and tools.

Advantages of JavaFX:

- Easy to learn and use for developers familiar with Java.
- Provides a modern toolkit for creating aesthetically pleasing GUIs.
- Cross-platform compatibility ensures consistent behavior on different operating systems.
- Separation of design (FXML) and logic (Java) simplifies application development and

maintenance.

Example Use Cases:

- Dashboard applications for businesses.
- Media players with audio and video support.
- Simulation and visualization tools.
- Educational and training apps.

Algorithm/Steps

Follow these steps to create the JavaFX application:

1. **Set Up the Environment**:
    - Install the JavaFX SDK and configure it in your IDE.
    - Create a new Java project and include JavaFX libraries in the build path.
2. **Define the Application Structure**:
    - Extend the Application class and override the start(Stage primaryStage) method.
    - Set up a TabPane for organizing content into sections (Pre-Requisites, Objective, etc.).
3. **Design the UI**:
    - Use layout managers (VBox, HBox) to arrange components.
    - Add UI controls like Label, TextArea, and Button to display content.
4. **Write the Logic**:
    - Populate each tab with relevant descriptive content.
    - Handle any user interactions (e.g., navigating tabs, closing the app).
5. **Test and Debug**:
    - Run the application to ensure it behaves as expected.
    - Fix any layout or interaction issues.
6. **Package the Application**:
    - Export the project as a JAR file to make it portable.

Conclusion:

The JavaFX application developed as part of this assignment demonstrates how to design and organize a user-friendly desktop application with interactive components. It highlights the core concepts of JavaFX, such as the use of the scene graph, layout managers, and event-driven programming, while also showcasing the flexibility of integrating CSS for styling and FXML for UI structuring.

Frequently Asked Questions

1. How do I set up JavaFX in my IDE?
2. Can I use JavaFX with other programming languages?
3. Is JavaFX better than Swing?
4. What is JavaFX?

Assignment No.7

Assignment Title: Create a Java Enterprise Edition (Java EE) application using Enterprise JavaBeans (EJB).

Aim: Create a Java Enterprise Edition (Java EE) application using Enterprise JavaBeans (EJB) to managing employee information, allowing users to add new employees, view a list of employees, and remove employees.

Pre-Requisites:

1. **Basic Knowledge of Java**: A solid understanding of Java programming concepts, including classes, methods, and OOP principles.
2. **Familiarity with Java EE**: Knowledge of Java EE architecture and components, particularly Enterprise JavaBeans (EJB), servlets, and Java Persistence API (JPA).
3. **Database Knowledge**: Understanding of relational databases (e.g., MySQL, PostgreSQL) and SQL for performing CRUD operations.

Objective

1. Develop a Java EE application using Enterprise JavaBeans (EJB) to manage employee information.
2. Implement functionalities to **add new employees**, **view a list of employees**, and **remove employees**.
3. Demonstrate the use of session beans (stateless or stateful) for business logic and entity beans for persistence.

Outcomes

1. Understand the role of Enterprise JavaBeans (EJB) in building enterprise-level applications.
2. Learn how to use **stateless session beans** for handling business logic.
3. Gain practical experience with **entity beans** and the **Java Persistence API (JPA)** for database operations.
4. Understand how to interact with an application server for deploying and running Java EE applications.
5. Learn how to integrate EJBs with web interfaces or RESTful APIs for user interaction.

Theory

Enterprise JavaBeans (EJB) is a key component of the Java EE platform used to build scalable, robust, and transactional enterprise applications. It simplifies the development of large-scale, distributed applications by managing complexities such as transaction handling, security,

and resource pooling.

Types of EJBs:

1. **Session Beans**:
    - **Stateless Session Beans**: Perform business logic without maintaining any client-specific state.
    - **Stateful Session Beans**: Maintain conversational state with the client across multiple method calls.
2. **Entity Beans** (Deprecated, replaced by JPA):
    - Used for persistent data storage and retrieval, now managed by JPA.

Key Features of EJB:

- **Transaction Management**: Automatically handles transactions using declarative or programmatic approaches.
- **Security**: Built-in mechanisms for role-based access control.
- **Lifecycle Management**: The container manages the creation, pooling, and destruction of EJB instances.
- **Interoperability**: Allows integration with web services, REST APIs, and other Java EE components.

In this assignment, we will use:

- **Stateless Session Beans**: To manage business logic like adding, viewing, and deleting employees.
- **Entity Beans via JPA**: To interact with the database and store employee information.

Algorithm/Steps

Follow these steps to create the Java EE application:

1. **Set Up the Environment**:
    - Install a Java EE-compatible application server (e.g., WildFly, GlassFish).
    - Configure the database (e.g., create a table for employee details in MySQL).
2. **Define the Entity Class**:
    - Create a JPA entity class for the Employee object with fields like id, name, designation, and salary.
3. **Create the Stateless Session Bean**:
    - Write a stateless session bean that contains methods for:
        - Adding an employee.
        - Retrieving a list of all employees.
        - Removing an employee by ID.

4. **Implement the Persistence Logic**:
    - Use JPA to map the Employee entity to the database and perform CRUD operations.
5. **Develop the Web Interface or REST API**:
    - Option 1: Use JSP/Servlets to create a web-based interface for user interaction.
    - Option 2: Expose the EJB methods through a RESTful API for interacting with the application.
6. **Deploy the Application**:
    - Package the application as a WAR/EAR file and deploy it to the application server.
7. **Test the Application**:
    - Add test data by using the addEmployee method.
    - Verify that the list of employees can be retrieved and displayed.
    - Test the delete functionality to ensure employees can be removed.

Conclusion:

This assignment highlights the use of Enterprise JavaBeans (EJB) in building a scalable and modular Java EE application. It demonstrates how to structure an enterprise application by separating business logic (via session beans) from data persistence (via JPA).

Frequently Asked Questions

1. What is the difference between Stateless and Stateful Session Beans?
2. What is JPA, and why is it used?
3. What are the advantages of using EJB?
4. Can I use EJB with other technologies like REST or Web Services?

Assignment No. 8

Assignment Title: Create a simple web application using the Spring Framework that allows users to manage a list of tasks.

Aim: Create a simple web application using the Spring Framework that allows users to manage a list of tasks. The application should provide functionality to view existing tasks, add new tasks, mark tasks as completed, and delete tasks.

Pre-Requisites:

1. **Java Development Kit (JDK)**: Version 11 or higher.
2. **Maven** (Optional if you use an IDE like IntelliJ)
3. **IDE**: Use IntelliJ IDEA, Eclipse, or Visual Studio Code for easier development.
4. **Spring Boot CLI** (Optional): Install it if you want an alternative to Maven for running the application.

Objective:

1. Understand Spring framework
2. Create simple web application using Spring Framework

Outcomes:

1. Students able to understand Spring framework
2. Students able to create simple web application using Spring Framework

Theory:

Spring Framework is a comprehensive and versatile platform for enterprise Java development. It is known for its Inversion of Control (IoC) and Dependency Injection (DI) capabilities that simplify creating modular and testable applications. Key features include Spring MVC for web development, Spring Boot for rapid application setup, and Spring Security for robust authentication and authorization. With a rich ecosystem covering Spring Data for database interactions and Spring Cloud for building microservices, Spring supports scalable and resilient enterprise solutions, making it an essential framework for developers of all experience levels.

What is Spring Framework?

Spring is a lightweight and popular open-source Java-based framework developed by Rod Johnson in 2003. It is used to develop enterprise-level applications. It provides support to many other frameworks such as Hibernate, Tapestry, EJB, JSF, Struts, etc. so it is also called a framework of frameworks. It’s an application framework and IOC (Inversion of Control) container for the Java platform. The spring contains several modules like IOC, AOP, DAO, Context, WEB MVC, etc.

Why to use Spring?

Spring framework is a Java platform that is open source. When it comes to size and transparency, Spring is a featherweight. Spring framework's basic version is about 2MB in size. The Spring Framework's core capabilities can be used to create any Java program, however there are modifications for constructing web applications on top of the Java EE platform. By offering a POJO-based programming model, the Spring framework aims to make J2EE development easier to use and to promote good programming habits.

Algorithm/Steps:

Algorithm for the Task Management Application

1. Initialize the Application:
    - Set up a Spring Boot application.
    - Include dependencies for Spring Web, Spring Data JPA, Thymeleaf, and an embedded H2 database.
2. Define the Task Model:
    - Create a Task class to represent a task entity with fields like id, name, status, and description.
3. Set Up the Database:
    - Use H2 as an in-memory database for storing tasks.
    - Configure the database in application.properties.
4. Create the Repository Layer:
    - Define a TaskRepository interface that extends JpaRepository for basic CRUD operations.
5. Create the Service Layer:
    - Implement a TaskService class to handle business logic, such as retrieving tasks, saving tasks, marking tasks as completed, and deleting tasks.
6. Create the Controller Layer:
    - Define a TaskController to handle HTTP requests for viewing tasks, adding tasks, marking them as completed, and deleting them.
7. Create the Frontend:
    - Use Thymeleaf templates for the user interface to display tasks and handle form submissions.
8. Run and Test:
    - Start the application.
    - Use a browser to access the UI and test all functionalities.

Conclusion :

The web application developed using the Spring Framework provides an efficient and modern solution for task management. It showcases how Spring simplifies web application development by integrating key modules such as Spring MVC for handling HTTP requests, Spring Boot for rapid application setup, and Spring Data JPA for interacting with a database**.**

Frequently Asked Questions

1. What is the Spring Framework?
2. What is Spring Boot, and why is it used?
3. What is Spring MVC?
4. What is the benefit of using RESTful APIs in this application?
5. How can I deploy the Spring application to production?