**Java Swing Assignment**

**1. What is Java Swing?**

Java Swing is a part of Java Foundation Classes (JFC) used for building graphical user interfaces (GUIs) for Java applications. It provides a rich set of components such as buttons, text fields, tables, trees, and more, which can be used to create complex and user-friendly interfaces. Swing is built on top of the Abstract Window Toolkit (AWT) and offers more advanced and flexible components.

**2. Describe the Swing architecture and its components.**

Swing architecture is based on the Model-View-Controller (MVC) design pattern, which separates the data (model), the user interface (view), and the user interaction logic (controller). The main components of Swing include:

- \*\*JComponent\*\*: The base class for all Swing components.

- \*\*JFrame\*\*: The top-level container used to create windows.

- \*\*JPanel\*\*: A generic container for grouping components.

- \*\*JButton\*\*: A button component.

- \*\*JLabel\*\*: A display area for a short text string or an image.

- \*\*JTextField\*\*: A component that allows the editing of a single line of text.

Key differences between Swing and AWT:

- \*\*Lightweight Components\*\*: Swing components are lightweight, meaning they are written entirely in Java and are not tied to platform-specific code.

- \*\*Pluggable Look and Feel\*\*: Swing supports changing the look and feel of applications without changing the code.

- \*\*Rich Set of Components\*\*: Swing offers more advanced components than AWT, such as tables, trees, and text components.

**3. Explain the Model-View-Controller (MVC) architecture in Swing.**

The MVC architecture in Swing separates the concerns of application design into three interconnected components:

- \*\*Model\*\*: Manages the data and business logic of the application. It notifies the view of any data changes.

- \*\*View\*\*: Renders the data from the model to the user and sends user actions to the controller.

- \*\*Controller\*\*: Handles user input, interacts with the model, and updates the view accordingly.

This separation allows for more modular, maintainable, and scalable applications.

**4. Write a simple Java Swing program that displays a JFrame with a title.**

A basic Swing program would create a JFrame and add a JLabel to it, which displays "Hello, World!". This showcases the simplicity of setting up a GUI with Swing components.

**5. Create a Swing application with a JButton that changes the text of a JLabel when clicked.**

This application demonstrates event handling in Swing. It involves a JButton that, when clicked, triggers an event listener to update the text of a JLabel.

**6. Write a program to create a simple Swing form with JTextField and JButton.**

In this program, a JTextField is used for user input, and a JButton, when clicked, displays the entered text in a JLabel. This highlights basic user interaction and event handling in Swing.

**7.** **Explain the different types of layout managers in Swing.**

Swing layout managers are used to arrange components within a container. Key types include:

- BorderLayout: Divides the container into five regions: NORTH, SOUTH, EAST, WEST, and CENTER.

- FlowLayout: Places components in a line, one after another, in the direction specified.

- GridLayout: Arranges components in a grid of cells with equal size.

- BoxLayout: Arranges components either vertically or horizontally.

**8. Write a program that uses BorderLayout to place components in different regions of a JFrame.**

This program would demonstrate how to use the BorderLayout manager to place components in the NORTH, SOUTH, EAST, WEST, and CENTER regions of a JFrame.

**9. Create a Swing application using GridLayout to arrange components in a 3x3 grid.**

In this application, components are arranged in a 3x3 grid using GridLayout, demonstrating equal distribution of space among components.

**10. Design a form using BoxLayout to align components vertically and horizontally.**

BoxLayout is used to align components either vertically or horizontally. This program would show how to create a form where components are neatly aligned in either orientation.

**11. Explain event handling in Swing.**

Event handling in Swing involves managing user interactions with the GUI components. Key concepts include:

- \*\*Event Listeners\*\*: Interfaces that receive and handle events (e.g., ActionListener for button clicks).

- \*\*Event Sources\*\*: Objects that generate events (e.g., JButton).

Listeners are registered with event sources to handle specific events when they occur.

**12. Write a program that changes the background color of a JPanel when different buttons are clicked.**

This program demonstrates how to use buttons to trigger events that change the properties of a JPanel, such as its background color.

**13. Create a simple calculator using Swing components and event handling.**

A basic calculator application would use JTextFields for input, JButtons for operations, and JLabel for displaying results. Event handling is used to perform calculations like addition, subtraction, multiplication, and division.

**14. Write a program to create a Swing application with a JComboBox and a JTextArea.**

This application involves a JComboBox for selecting items and a JTextArea for displaying the selected item. It demonstrates how to use JComboBox and JTextArea together.

**15. Create a Swing application with a JList that displays a list of items.**

A JList is used to display a list of items, and a JLabel shows details of the selected item. This highlights the use of JList for list-based interfaces.

**16. Write a program that uses a JTable to display data in a tabular format.**

A JTable is used to display and manage data in a table format. The application includes functionality to add, update, and delete rows, demonstrating the dynamic nature of JTable.

**17. Explain the different types of dialogs available in Swing.**

Swing provides several types of dialogs using JOptionPane, including:

- \*\*Message Dialog\*\*: Displays a message to the user.

- \*\*Input Dialog\*\*: Prompts the user to input a value.

- \*\*Confirmation Dialog\*\*: Asks the user to confirm an action.

**18. Create a Swing application with a JMenuBar, JMenu, and JMenuItem.**

A JMenuBar at the top of the JFrame contains JMenu, which further contains JMenuItems. These menu items trigger actions like opening a file, saving a file, and exiting the application.

**19. Write a program to create a custom dialog using JDialog.**

A custom dialog created with JDialog includes text fields for user input and buttons to submit or cancel, allowing for custom user interactions.

**20. Create a Swing application with a JTabbedPane to switch between multiple tabs.**

JTabbedPane allows for multiple tabs within a single JFrame, each containing different components. This demonstrates the organization of content in tabbed interfaces.

**21. Write a program to create a Swing application with a JSplitPane.**

JSplitPane divides the frame into two resizable panes, allowing dynamic resizing and effective use of space within the application.

**22. Develop a Swing application that includes a JProgressBar to show progress for a long-running task.**

A JProgressBar is used to visually indicate the progress of a long-running task. The progress bar is updated in a separate thread to keep the UI responsive.

**23. Explain how to change the look and feel of a Swing application.**

Swing supports pluggable look and feel, which allows the appearance of the application to be changed. Look and feel can be set using UIManager to styles like Metal, Nimbus, or Windows.

**24. Write a program that applies a custom look and feel to a Swing application.**

Using UIManager, a custom look and feel can be applied to change the appearance of the entire application, showcasing Swing's flexibility.

**25. Design a Swing-based login form with username and password fields.**

A login form includes JTextField for username, JPasswordField for password, and JButton for submitting. Validation ensures fields are not empty, and appropriate messages are displayed.

**26. Create a Swing application that reads data from a file and displays it in a JTextArea.**

This application uses JFileChooser to open and save files. The contents of the file are read and displayed in a JTextArea, highlighting file handling in Swing.

**27. Develop a simple text editor using Swing components.**

A simple text editor supports basic text editing features like cut, copy, paste, and search. It uses JTextArea for text input and JMenu for editing actions.

**28. Write a Swing application that connects to a database and performs CRUD operations**.

This application connects to a database, displays data in a JTable, and includes buttons to add, update, and delete records, demonstrating database interaction in Swing.