Graphical User Interface (GUI) in Java

3.1 Panels, Scroll Pane, Menu, Scroll Bar

1. Panels

A **Panel** in Java is a container that holds and organizes components within a window. It does not have a title bar, menu bar, or border. It is commonly used inside other containers such as Frames or Applets.

Example of Panel

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

public class PanelExample {
    PanelExample() {
        JFrame frame = new JFrame("Panel Example");
        JPanel panel = new JPanel();
        panel.setBackground(Color.LIGHT_GRAY);

        JButton button1 = new JButton("Button 1");
        JButton button2 = new JButton("Button 2");

        panel.add(button1);
        panel.add(button2);

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

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

---

2. Scroll Pane

A **JScrollPane** is used when the content is too large to fit within the visible area of a component. It provides a scrolling mechanism.

Example of JScrollPane

```
import javax.swing.*;

public class ScrollPaneExample {
    ScrollPaneExample() {
        JFrame frame = new JFrame("Scroll Pane Example");

        JTextArea textArea = new JTextArea(10, 30);
        JScrollPane scrollPane = new JScrollPane(textArea);

        frame.add(scrollPane);
        frame.setSize(400, 300);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setVisible(true);
    }

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

---

3. Menu

A **Menu** in Java is created using `JMenuBar`, `JMenu`, and `JMenuItem`.

Example of Menu in Java

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

public class MenuExample {
    MenuExample() {
        JFrame frame = new JFrame("Menu Example");

        JMenuBar menuBar = new JMenuBar();
        JMenu menu = new JMenu("File");
        JMenuItem open = new JMenuItem("Open");
        JMenuItem save = new JMenuItem("Save");
        JMenuItem exit = new JMenuItem("Exit");

        exit.addActionListener(e -> System.exit(0));

        menu.add(open);
        menu.add(save);
        menu.add(exit);
        menuBar.add(menu);

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

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

---

4. Scroll Bar

A **JScrollBar** is a component that allows scrolling through a range of values.

Example of ScrollBar

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

public class ScrollBarExample {
    ScrollBarExample() {
        JFrame frame = new JFrame("ScrollBar Example");

        JScrollBar scrollBar = new JScrollBar(JScrollBar.VERTICAL, 0, 10, 0,
100);
        JLabel label = new JLabel("Value: 0");

        scrollBar.addAdjustmentListener(e -> label.setText("Value: "
e.getValue()));

        frame.setLayout(new BorderLayout());
        frame.add(scrollBar, BorderLayout.EAST);
        frame.add(label, BorderLayout.CENTER);

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

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

3.2 Working with Frame, Color, Fonts, and Layout Managers

1. Frame

A **JFrame** is a top-level container that represents a window.

Example of JFrame

```
import javax.swing.*;

public class FrameExample {
    FrameExample() {
        JFrame frame = new JFrame("Frame Example");
        frame.setSize(400, 300);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setVisible(true);
    }

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

---

2. Color

The `setBackground(Color c)` method is used to set the background color.

Example

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

```
public class ColorExample {
    ColorExample() {
        JFrame frame = new JFrame("Color Example");
        JPanel panel = new JPanel();
        panel.setBackground(Color.BLUE);
        frame.add(panel);
        frame.setSize(400, 300);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setVisible(true);
    }

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

---

3. Fonts

We use `setFont(new Font("FontName", Font.Style, Size))` to set fonts.

Example

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

public class FontExample {
    FontExample() {
        JFrame frame = new JFrame("Font Example");
        JLabel label = new JLabel("Hello, Java!", JLabel.CENTER);
        label.setFont(new Font("Arial", Font.BOLD, 24));

        frame.add(label);
        frame.setSize(400, 200);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setVisible(true);
    }

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

---

4. Layout Managers

Layout managers help in organizing components in a container.

Types of Layout Managers

1. **FlowLayout** – Arranges components in a row.
2. **BorderLayout** – Divides the container into five regions.
3. **GridLayout** – Arranges components in a grid.
4. **BoxLayout** – Aligns components vertically or horizontally.

Example of BorderLayout

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

public class BorderLayoutExample {
    BorderLayoutExample() {
        JFrame frame = new JFrame("BorderLayout Example");
        frame.setLayout(new BorderLayout());

        frame.add(new JButton("North"), BorderLayout.NORTH);
        frame.add(new JButton("South"), BorderLayout.SOUTH);
        frame.add(new JButton("East"), BorderLayout.EAST);
        frame.add(new JButton("West"), BorderLayout.WEST);
        frame.add(new JButton("Center"), BorderLayout.CENTER);

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

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

3.3 Event Handling

Java uses the **Event Delegation Model (EDM)**, which includes:

- **Event Source** – The component that generates an event (e.g., Button).
- **Event Listener** – A method that listens and responds to events.

Handling Mouse and Keyboard Events

Example of MouseListener

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

public class MouseExample extends JFrame implements MouseListener {
    JLabel label;

    MouseExample() {
        label = new JLabel("Click Anywhere!");
        add(label);

        addMouseListener(this);

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

    public void mouseClicked(MouseEvent e) {
        label.setText("Mouse Clicked at X: "  e.getX()  " Y: "  e.getY());
    }

    public void mouseEntered(MouseEvent e) {}
    public void mouseExited(MouseEvent e) {}
    public void mousePressed(MouseEvent e) {}
    public void mouseReleased(MouseEvent e) {}
    public static void main(String[] args) {
        new MouseExample();
    }
}
```

Summary of GUI and Event Handling in Java

1. Panels, Scroll Pane, Menu, Scroll Bar

- **Panels**: A `JPanel` is a container used to group components inside a window. It is commonly used within other containers like `JFrame`.
- **Scroll Pane**: `JScrollPane` is used to add scroll functionality when content exceeds the visible area. It wraps components like text areas or tables.
- **Menu**: `JMenuBar` holds `JMenu`, which contains `JMenuItem`. It provides a structured way to create menus for user interaction.
- **Scroll Bar**: `JScrollBar` allows users to scroll through a range of values and is commonly used for navigating large content.

2. Working with Frame, Color, Fonts, and Layout Managers

- **Frame (`JFrame`)**: A top-level window that serves as the main container for GUI applications.
- **Color**: Java provides the `Color` class to set background and foreground colors of components.
- **Fonts**: The `Font` class is used to set text styles like bold, italic, and different font families.
- **Layout Managers**: Java provides different layout managers to control component positioning:
    - **FlowLayout**: Arranges components in a row.
    - **BorderLayout**: Divides the container into five regions (North, South, East, West, Center).
    - **GridLayout**: Arranges components in a grid of rows and columns.
    - **BoxLayout**: Aligns components horizontally or vertically.

3. Event Handling

- **Event Delegation Model (EDM)**: Java follows the EDM, where events are generated by components (event sources) and handled by event listeners.
- **Event Sources**: Components like buttons, text fields, and menu items generate events.
- **Event Listeners**: Interfaces that handle events (e.g., `ActionListener`, `MouseListener`, `KeyListener`).
- **Handling Mouse and Keyboard Events**:
    - **Mouse Events**: `MouseListener` captures actions like clicks, movement, and entry/exit of the cursor.
    - **Keyboard Events**: `KeyListener` detects key presses and releases.

4. Adapter Classes and Inner Classes

- **Adapter Classes**: Used to simplify event handling by providing default implementations of listener interfaces.
- **Inner Classes**: Helps in organizing event-handling code by defining event listeners inside another class.

Why Use Swing Over AWT?

Feature	AWT	Swing
Lightweight Components	❌ No	✅ Yes
Advanced Components (`JTable`, `JTree`)	❌ No	✅ Yes
Pluggable Look and Feel	❌ No	✅ Yes
MVC Architecture	❌ No	✅ Yes
Better Event Handling	❌ No	✅ Yes
Double Buffering (Smooth Graphics)	❌ No	✅ Yes
Icons and Tooltips Support	❌ No	✅ Yes
Nested Containers	❌ No	✅ Yes
Undo/Redo Support	❌ No	✅ Yes
Drag and Drop Support	❌ No	✅ Yes
Threading Support (`SwingWorker`)	❌ No	✅ Yes

MODULE 4:

SWINGS

Introduction to Swings

Swing is a part of Java Foundation Classes (JFC) that provides a set of lightweight components for building graphical user interfaces (GUI). It is an extension of the Abstract Window Toolkit (AWT) and provides more powerful and flexible components.

Features of Swing

- **Lightweight:** Components are not dependent on the native operating system.
- **Pluggable Look and Feel:** Swing allows changing the appearance of components.
- **Rich Set of Components:** Includes buttons, tables, trees, etc.
- **MVC Architecture:** Follows Model-View-Controller design pattern.
- **Platform-Independent:** Works on multiple platforms.

Hierarchy of Swing Components

Swing components are part of the `javax.swing` package and are built on top of AWT components.

```
java.lang.Object
   ↳ java.awt.Component
      ↳ java.awt.Container
         ↳ javax.swing.JComponent
```

Swing Component Hierarchy Diagram

```
JComponent
├── JLabel
├── JTextField
├── JButton
├── JCheckBox
├── JRadioButton
├── JList
├── JComboBox
├── JScrollPane
└── JPanel
```

Top-Level Containers

Top-level containers are the base of every Swing application. They are:

1. JFrame

- It is the main window where components like buttons and text fields are added.
- It has a title bar, minimize, maximize, and close buttons.

Syntax

```
import javax.swing.*;
public class MyFrame {
    public static void main(String[] args) {
        JFrame frame = new JFrame("My First Frame");
        frame.setSize(400, 300);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setVisible(true);
    }
}
```

---

2. JWindow

- Similar to `JFrame` but does not have title bars or buttons.
- Used for splash screens.

Example

```
import javax.swing.*;
public class MyWindow {
    public static void main(String[] args) {
        JWindow window = new JWindow();
        window.setSize(300, 200);
        window.setVisible(true);
    }
}
```

---

3. JDialog

- Used to create pop-up dialog boxes.
- Can be modal (blocks other windows) or non-modal.

Example

```
import javax.swing.*;
public class MyDialog {
    public static void main(String[] args) {
        JDialog dialog = new JDialog();
        dialog.setSize(200, 150);
        dialog.setTitle("My Dialog");
        dialog.setVisible(true);
    }
}
```

Swing Components

1. JPanel

- A container to hold multiple components together.
- Used to group components.

Example

```
JPanel panel = new JPanel();
panel.add(new JButton("Click Me"));
```

---

2. JButton

- A push button component used for triggering actions.

Example

```
JButton button = new JButton("Click Here");
```

---

3. JToggleButton

- A button that stays pressed when clicked and toggles on/off.

Example

```
JToggleButton toggleButton = new JToggleButton("ON/OFF");
```

---

4. JCheckBox

- Allows selecting multiple options.

Example

```
JCheckBox checkBox = new JCheckBox("Accept Terms");
```

---

5. JRadioButton

- Allows selecting only one option in a group.

Example

```
JRadioButton radio1 = new JRadioButton("Male");
JRadioButton radio2 = new JRadioButton("Female");
ButtonGroup group = new ButtonGroup();
group.add(radio1);
group.add(radio2);
```

---

6. JLabel

- Displays text or images that are not editable.

Example

```
JLabel label = new JLabel("Welcome to Swing");
```

---

7. JTextField

- Allows users to enter a single-line text input.

Example

```
JTextField textField = new JTextField(20);
```

---

8. JTextArea

- Allows users to enter multi-line text.

Example

```
JTextArea textArea = new JTextArea(5, 20);
```

---

9. JList

- Displays a list of items.

Example

```
String[] items = {"Item 1", "Item 2", "Item 3"};
JList<String> list = new JList<>(items);
```

---

10. JComboBox

- A drop-down list for selecting items.

Example

```
String[] choices = {"Option 1", "Option 2", "Option 3"};
JComboBox<String> comboBox = new JComboBox<>(choices);
```

---

11. JScrollPane

- Adds scrolling ability to other components.

Example

```
JTextArea textArea = new JTextArea(5, 20);
```

```
JScrollPane scrollPane = new JScrollPane(textArea);
```

APPLETS

Life Cycle of an Applet

An applet goes through five stages in its lifecycle:

1. **init() –** Initializes the applet.
2. **start() –** Starts or resumes the applet.
3. **paint(Graphics g) –** Used for drawing on the applet.
4. **stop() –** Stops the applet execution.
5. **destroy() –** Cleans up resources before closing.

Applet Life Cycle Diagram

```
init() → start() → paint() → stop() → destroy()
```

Differences Between Applets and Applications

Feature	Applet	Application
Execution	Runs in a browser	Runs independently
`main()` method	No	Yes
Security	More secure	Less secure
GUI	Uses AWT/Swing	Uses JFrame

Developing Applets

To create an applet:

1. Extend the `Applet` class.
2. Override `init()`, `start()`, and `paint()`.
3. Use `AppletViewer` or a browser to run it.

Simple Applet Program

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

public class MyApplet extends Applet {
    public void paint(Graphics g) {
        g.drawString("Hello, Applet!", 50, 50);
    }
}
```

Running the Applet

Save the file as `MyApplet.java` and compile it using:

```
javac MyApplet.java
```

Run it using:

```
appletviewer MyApplet.html
```

HTML File for Running Applet

```
<html>
    <body>
        <applet code="MyApplet.class" width="300" height="200"></applet>
    </body>
</html>
```

5.1 Event-Driven Programming in Java

Event-Driven Programming in Java:

- In event-driven programming, the flow of the program is controlled by events. These events are typically user actions, such as mouse clicks, key presses, or timer expirations.
- Java provides a robust event handling model through the AWT (Abstract Window Toolkit) and Swing libraries.

Event-Handling Process:

- The basic process of event handling is:
  1. **Event generation**: User interaction (e.g., clicking a button).
  2. **Event notification**: The event is passed to the appropriate listener (an object that is registered to handle events).
  3. **Event handling**: The listener responds by executing a specific method.

Event Handling Mechanism:

- Java provides two main mechanisms for event handling:
  1. **The Listener Model**: Involves implementing specific event listener interfaces and overriding their methods.
  2. **The Delegation Model**: A more flexible and preferred mechanism where event sources (like buttons) delegate events to listener objects.

The Delegation Model of Event Handling:

- In this model, an event source (like a button or checkbox) does not handle the event directly but delegates it to an event listener.
- Event listeners are objects that "listen" for events on event sources.
- Example: When a user clicks a button, the button will delegate that click event to an action listener to handle it.

Event Classes:

- Event classes represent the actual event that occurs, e.g., `ActionEvent`, `MouseEvent`, etc.
- These classes are part of the `java.awt.event` package.
- Event classes contain methods that allow listeners to retrieve information about the event, such as the source of the event or the coordinates of a mouse click.

Event Sources:

- Event sources are the components (like buttons, checkboxes, text fields) that generate events.
- An event source must be linked with an event listener that will handle the event when it occurs.
- Event sources are typically user interface components that the user interacts with.

Event Listeners:

- Event listeners are interfaces in Java that define methods that respond to specific types of events.
- Common listeners include:
  - `ActionListener`: Handles actions like button presses.
  - `MouseListener`: Handles mouse events.
  - `KeyListener`: Handles keyboard events.
- To handle an event, you need to implement the listener interface and override the necessary methods.

Adapter Classes as Helper Classes in Event Handling:

- Java provides adapter classes, which are abstract classes implementing event listener interfaces. These classes provide default (empty) method implementations.
- Adapter classes are helpful when you need to handle only a subset of the methods from an event listener interface, eliminating the need to implement every method.
  - Example: `MouseAdapter`, `KeyAdapter`.

5.2 Database Programming using JDBC

Introduction to JDBC:

- JDBC (Java Database Connectivity) is an API that enables Java applications to interact with databases.
- It provides methods for querying and updating data in a database.
- JDBC supports a wide variety of database operations, including connecting to databases, executing SQL queries, and retrieving results.

JDBC Drivers & Architecture:

- JDBC drivers are Java classes that allow Java programs to connect to a particular database.
- There are four types of JDBC drivers:
  1. **JDBC-ODBC Bridge Driver** (Type 1)
  2. **Native-API Driver** (Type 2)
  3. **Network Protocol Driver** (Type 3)
  4. **Thin Driver** (Type 4)
- **JDBC Architecture**:
  1. **JDBC API**: Java classes and interfaces for connecting to databases.
  2. **JDBC Driver Manager**: Manages a list of database drivers.
  3. **JDBC Driver**: Converts Java calls to database-specific calls.

CRUD Operation Using JDBC:

- CRUD stands for Create, Read, Update, and Delete. These are the basic operations for managing data in a database.
  - **Create**: Insert new records using SQL `INSERT` statements.
  - **Read**: Retrieve data using `SELECT` statements.
  - **Update**: Modify existing records using `UPDATE` statements.

    - **Delete**: Remove records using `DELETE` statements.

Connecting to Non-Conventional Databases:

- JDBC can be used to connect to non-relational (NoSQL) databases by using appropriate JDBC drivers provided by the database vendor.
- The process for connecting to NoSQL databases (e.g., MongoDB, Cassandra) is similar to connecting to relational databases, but with differences in the JDBC driver and SQL syntax.

5.3 Java Server Technologies Servlet

Servlet:

- A Servlet is a Java class used to handle HTTP requests and generate responses. It is used to extend the capabilities of servers, often for web applications.
- Servlets run on a server (like Tomcat, Jetty) and handle client requests (usually from a browser).
- Servlets can interact with databases, process forms, and manage sessions.

5.4 Web Application Basics, Architecture, and Challenges

Web Application Basics:

- A web application typically involves a client (usually a web browser) and a server. The client sends HTTP requests, and the server processes those requests and returns HTTP responses.
- Web applications use front-end technologies like HTML, CSS, and JavaScript and back-end technologies like Java, Python, or PHP.

Architecture and Challenges of Web Applications:

- **Architecture**:
    - Client-side: Involves the user interface (UI) and interacts with the server.
    - Server-side: Manages business logic, database interactions, and processes requests.
    - Database: Stores and retrieves data for the application.
- **Challenges**:
    - Security: Protecting against unauthorized access, data breaches, and attacks like SQL injection and cross-site scripting (XSS).
    - Scalability: Handling increased traffic by scaling up or distributing load.
    - Performance: Optimizing response time and resource consumption.
    - Compatibility: Ensuring the application works across different browsers and devices.

Introduction to Servlet:

- Servlets are server-side Java programs that handle HTTP requests and responses.
- They are part of Java EE (Enterprise Edition) and are commonly used in web applications.
- A servlet can generate dynamic content based on client input or application logic.

Servlet Life Cycle:

1. **Loading and Instantiation**: The web container loads the servlet class when it receives the first request.
2. **Initialization**: The container calls the `init()` method to initialize the servlet.
3. **Request Handling**: For each request, the `service()` method is called, which processes the request and generates the response.
4. **Destruction**: When the servlet is no longer needed, the container calls the `destroy()` method to clean up resources.

Developing and Deploying Servlets:

- To develop a servlet, create a Java class that extends `HttpServlet` and override the `doGet()` or `doPost()` methods for handling GET or POST requests.
- Deploying a servlet involves packaging it in a `.war` file and placing it in the web container's deployment directory.

Exploring Deployment Descriptor (web.xml):

- The `web.xml` file is a configuration file used by the web container to map servlets to specific URLs.
- It contains:
  - Servlet class and its URL pattern.
  - Initialization parameters for the servlet.
  - Welcome file for default application entry.

Handling Request and Response:

- HTTP requests are handled by the `doGet()` and `doPost()` methods in a servlet.
- The `HttpServletRequest` object contains information about the request (e.g., parameters, headers).
- The `HttpServletResponse` object is used to send a response back to the client (e.g., HTML content, redirect).
