REVA UNIVERSITY
Bengaluru, India

M23DE0202 – Object Oriented Programming using JAVA

II Semester MCA Academic Year : 2024 - 2025

School of Computer Science and Applications

Pinaka Pani. R
Assistant Professor

Unit IV

Introduction to GUI and JDBC Programming

- Java Swing is a part of Java Foundation Classes (JFC) and is used for building graphical user interfaces (GUIs) in Java applications.
- It provides a set of "lightweight" (all-Java language) components that, unlike the earlier AWT (Abstract Window Toolkit) components, are platform-independent.

Key Features of Java Swing

- **Platform Independence:**

•Swing components are written entirely in Java and thus run consistently on any platform that supports Java.

- **Rich Set of Components:**

•Swing offers a wide range of components, including buttons, labels, text fields, tables, trees, and more complex components like tabbed panes and scroll panes.

Customization:

•Swing components are highly customizable. You can change the look and feel of the components using the Pluggable Look And Feel (PLAF) architecture.

Event-Driven Programming:

•Swing is based on the Model-View-Controller (MVC) architecture. This means that the data (model), presentation (view), and user input handling (controller) are separated, making it easier to manage and maintain applications.

Lightweight Components:

•Swing components are lightweight in the sense that they are written entirely in Java and do not rely on native code. This makes them more flexible and portable.

Java Foundation Classes (JFC) :

- JFC is short for Java Foundation Classes, which encompass a group of features for building graphical user interfaces (GUIs) and adding rich graphics functionality and interactivity to Java applications.
- JFC was first announced at the 1997 JavaOne developer conference. It is defined as containing the features shown in the table below.

Features of the Java Foundation Classes	
Feature	**Description**
Swing GUI Components	Includes everything from buttons to split panes to tables.
Pluggable Look-and-Feel Support	Gives any program that uses Swing components a choice of look and feel. For example, the same program can use either the Java or the Windows look and feel. Many more look-and-feel packages are available from various sources. As of v1.4.2, the Java platform supports the GTK+ look and feel, which makes hundreds of existing look and feels available to Swing programs.
Accessibility API	Enables assistive technologies, such as screen readers and Braille displays, to get information from the user interface.
Java 2D API	Enables developers to easily incorporate high-quality 2D graphics, text, and images in applications and applets. Java 2D includes extensive APIs for generating and sending high-quality output to printing devices.
Drag-and-Drop Support	Provides the ability to drag and drop between Java applications and native applications.
Internationalization	Allows developers to build applications that can interact with users worldwide in their own languages and cultural conventions. With the input method framework developers can build applications that accept text in languages that use thousands of different characters, such as Japanese, Chinese, or Korean.

- Compiling and Running Swing Programs
- To compile and run a Swing application.

- The compilation instructions work for all Swing programs — applets, as well as applications.

- Create a program that uses Swing components.
- Compile the program.
- Run the program.

Basic Swing Components

JFrame: The main window container in a Swing application.

```
JFrame frame = new JFrame("My Swing Application");
frame.setSize(400,300);
frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
frame.setVisible(true);
```

JPanel: A generic container for grouping other components.

```
JPanel panel = new JPanel();
frame.add(panel);
```

JButton:

A button component that can trigger actions when clicked.

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

JLabel:

A text or image display area.

```
JLabel label = new JLabel("Hello, Swing!");
panel.add(label);
```

JTextField:

A single-line text input field.

```
JTextField textField = new JTextField(20);
panel.add(textField);
```

JTextArea:

A multi-line text input area.

```
JTextArea textArea = new JTextArea(5, 20);
panel.add(new JScrollPane(textArea));
```

```
import javax.swing.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
public class SimpleSwingApp
{
 public static void main(String[] args)
 {
// Create the main frame
 JFrame frame = new JFrame("Simple Swing App");
frame.setSize(400, 300);
 frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
 // Create a panel to hold components
JPanel panel = new JPanel();
frame.add(panel); placeComponents(panel); // Display the frame
frame.setVisible(true);
}
```

```
private static void placeComponents(JPanel panel)
 {
 panel.setLayout(null);
// Create a label
JLabel userLabel = new JLabel("User:");
 userLabel.setBounds(10, 20, 80, 25);
 panel.add(userLabel);
// Create a text field
 JTextField userText = new JTextField(20); userText.setBounds(100, 20,
165, 25); panel.add(userText);
```

```
// Create a button
JButton loginButton = new JButton("Login");
loginButton.setBounds(10, 80, 80, 25); panel.add(loginButton);
// Add action listener to button
loginButton.addActionListener(new ActionListener()
 {
public void actionPerformed(ActionEvent e)
 {
JOptionPane.showMessageDialog(null, "Button Clicked!"); } }); }
}
```

Running the Application

To run this application:

1.Save the code to a file named SimpleSwingApp.java.

2.Compile the file using the command: javac SimpleSwingApp.java.

3.Run the compiled class using: java SimpleSwingApp.

This application creates a simple window with a text field and a button. When the button is clicked, a dialog box is displayed.

JFrame

Introduction to basic Widgets in Java

The JFrame is the main window container for a Swing application.

It provides a window with borders, a title, and buttons to close, minimize, and maximize.

```
import javax.swing.JFrame;

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);
} }
```

Layout managers in Java

- Layout managers in Java Swing are used to arrange components within a container (such as a JFrame or JPanel).
- They control the size and position of the components, making it easier to create complex and responsive user interfaces.

Types of Layout Manager in Java

- Layout managers define how components are arranged within a container, such as a JFrame or JPanel.
- Java provides several layout managers to suit various design needs.

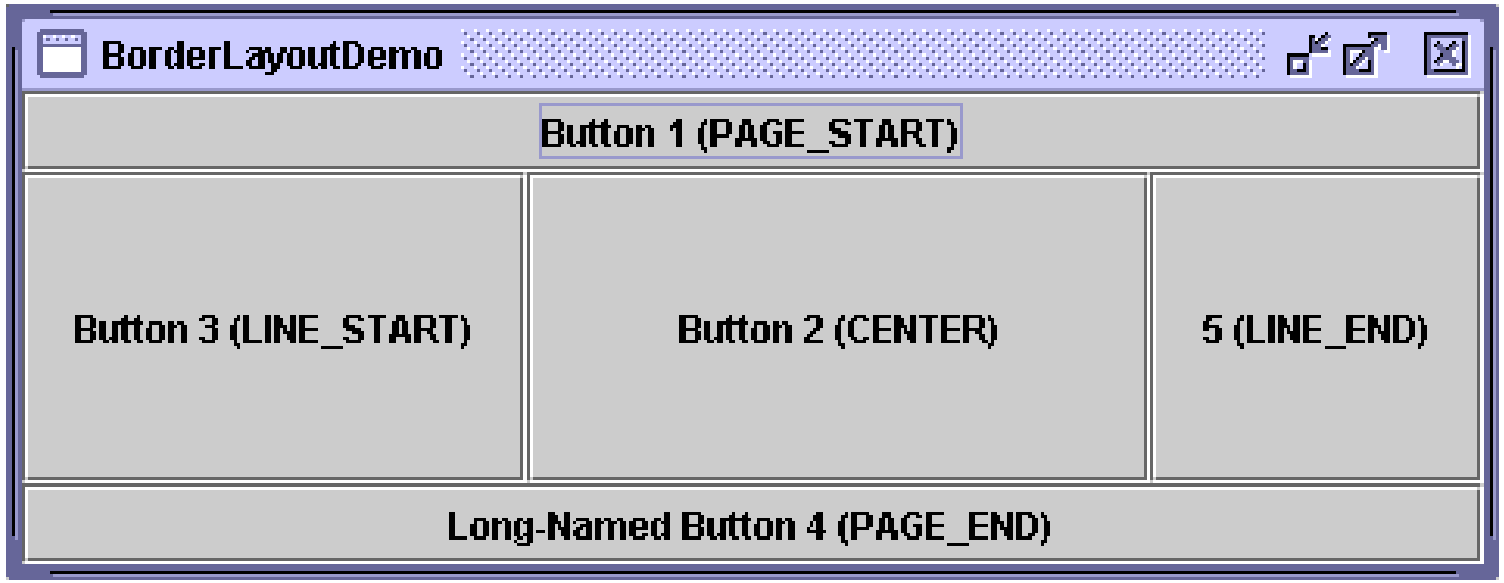
1. **FlowLayout**
2. **BorderLayout**
3. **GridLayout**
4. **CardLayout**
5. **BoxLayout**
6. **GridBagLayout**

FlowLayout

- **FlowLayout** is the default layout manager for every **JPanel**.
- It simply lays out components in a single row, starting a new row if its container isn't sufficiently wide.

BorderLayout

Every content pane is initialized to use a BorderLayout.
(As Using Top-Level Containers (in the Creating a GUI with JFC/Swing trail) explains, the content pane is the main container in all frames, applets, and dialogs.)
A BorderLayout places components in up to five areas: top, bottom, left, right, and center.
All extra space is placed in the center area.

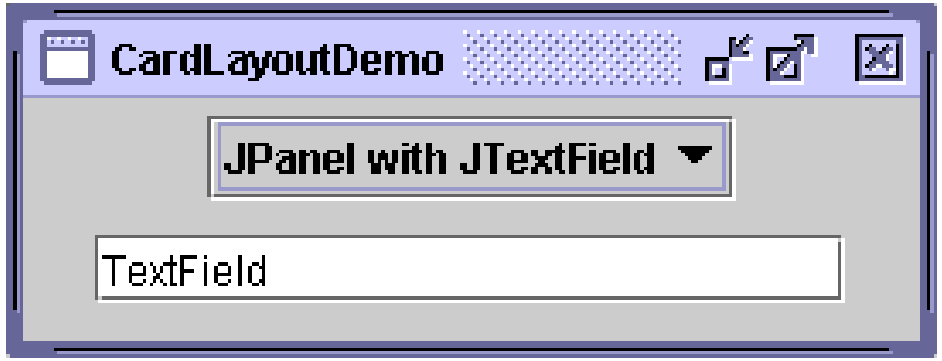


GridLayout

GridLayout simply makes a bunch of components equal in size and displays them in the requested number of rows and columns.

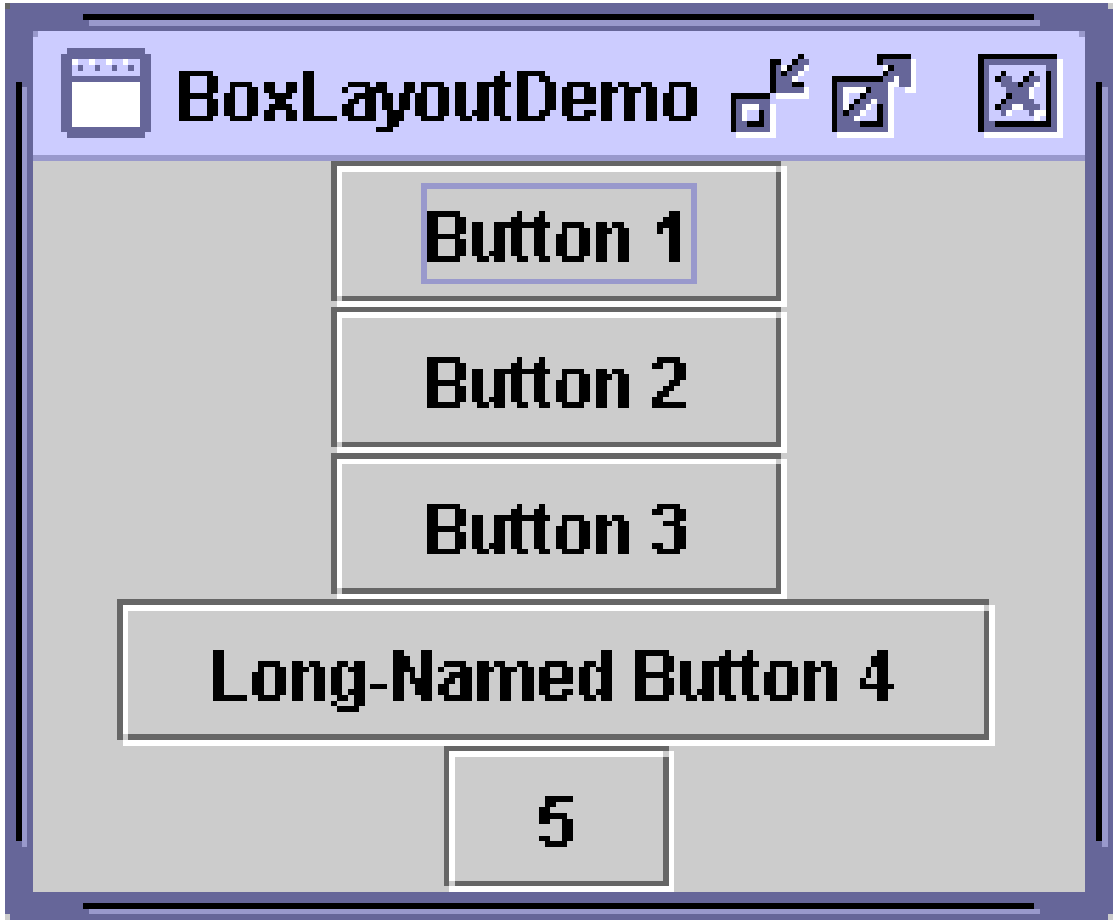
CardLayout

- The CardLayout class lets you implement an area that contains different components at different times.
- A CardLayout is often controlled by a combo box, with the state of the combo box determining which panel (group of components) the CardLayout displays.
- An alternative to using CardLayout is using a tabbed pane (in the Creating a GUI with JFC/Swing trail), which provides similar functionality but with a pre-defined GUI



BoxLayout

- The BoxLayout class puts components in a single row or column. It respects the components' requested maximum sizes and also lets you align components.



GridBagLayout

- GridLayout simply makes a bunch of components equal in size and displays them in the requested number of rows and columns.

Event Handling

- It refers to the process of managing and responding to user interactions or system-generated actions, such as **clicking a button, typing text, moving the mouse, or closing a window.**
- It is a key feature in building interactive GUI applications using Java Swing, AWT, or JavaFX.

- **Event:** An object representing an action or occurrence, such as a button click or a mouse movement.
- This mechanism is central to building interactive applications, especially those with graphical user interfaces (GUIs).

For example:

- ActionEvent: Triggered by actions like clicking a button.
- MouseEvent: Triggered by mouse actions like clicking or moving the mouse.
- KeyEvent: Triggered by keyboard actions like pressing a key.

- **Event Source**: The object that generates an event.
- A JButton can generate an ActionEvent when clicked.
- A JTextField can generate a KeyEvent when a key is pressed.
- **Event Listener**: An interface in Java that listens to events and provides callback methods to respond to them.
- ActionListener: Listens to ActionEvent.
- MouseListener: Listens to MouseEvent.
- KeyListener: Listens to KeyEvent.

- **Event Handler:** A method that implements the logic for responding to the event.
- This method is defined in the listener.

- **Event Delegation Model:** Which separates the event source from the event handling logic.

It works as follows:

- The event source generates the event.
- The listener is registered with the source.
- When the event occurs, the event source notifies the listener.

How Event Handling Works

- **Registering a Listener:**
  An event source (e.g., a button) must register a listener object to listen for events. The listener object implements the listener interface.
- **Generating an Event:**
  When a user interacts with the source (e.g., clicks a button), an event object is created.
- **Notifying the Listener:**
  The event source notifies all registered listeners, and the appropriate method in the listener interface is called.

Listener Interface	Event Type	Methods
ActionListener	Action events	actionPerformed(ActionEvent e)
MouseListener	Mouse events	mouseClicked, mousePressed, etc.
KeyListener	Keyboard events	keyPressed, keyReleased, etc.
ItemListener	Item events (checkboxes, etc.)	itemStateChanged(ItemEvent e)
WindowListener	Window events	windowOpened, windowClosing, etc.

Event Handling Models

- **Delegation Event Model (Preferred)**
  The event handling mechanism in Java uses this model:
  - The event is delegated to a listener object that implements the appropriate listener interface.
  - It improves separation of concerns, as the event source and event handling logic are decoupled.
- **Event Handling in AWT/Swing**
  Java Swing and AWT use the delegation model, where GUI components generate events, and listeners handle them.

Advantages of Event Handling

- **Separation of Logic**: Keeps the user interface code separate from the event handling code.
- **Reusability**: Listener objects can be reused for handling multiple events.
- **Flexibility**: Different listeners can be attached to a single event source, or a single listener can handle multiple events.

JPanel

A JPanel is a generic container that can hold other components.

It helps to organize and group components within a JFrame.

```
import javax.swing.JFrame;
import javax.swing.JPanel;
public class MyPanel
{ public static void main(String[] args)
{ JFrame frame = new JFrame("Panel Example");
JPanel panel = new JPanel(); frame.add(panel);
frame.setSize(400, 300);
frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
frame.setVisible(true); } }
```

JButton

A JButton is a button that can trigger an action when clicked.

JTextField

A JTextField is a single-line text input field

JLabel

A JLabel is a display area for a short text string or an image.

JCheckBox

A JCheckBox is a component that can be either selected or deselected.

JRadioButton

A JRadioButton is a button that can be selected or deselected, and is usually used in a group where only one button can be selected at a time.

JComboBox

- JComboBox is a drop-down list that allows the user to choose one item from a list.
- These basic widgets provide the foundation for creating user interfaces in Java Swing.
- You can combine them, customize their appearance, and add functionality to create rich and interactive applications.

Step-by-Step Example: Simple Swing Application

Java Swing is a powerful toolkit for building graphical user interfaces (GUIs) in Java applications.

Steps for Swing components (a frame, panel, button, text field, label, and some event handling).

1. Setting Up the Main Frame

The JFrame is the main window of the application. We'll set up a basic frame with a title, size, and a close operation.

```
import javax.swing.JFrame;

public class SimpleSwingApp {

public static void main(String[] args) {

JFrame frame = new JFrame("Simple Swing App");

frame.setSize(400, 300);

frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

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

2.Adding a Panel

Next, we add a JPanel to the frame. A panel is a container that can hold other components.

import javax.swing.JFrame; import javax.swing.JPanel; public class SimpleSwingApp { public static void main(String[] args) { JFrame frame = new JFrame("Simple Swing App"); frame.setSize(400, 300); frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE); JPanel panel = new JPanel();

frame.add(panel);

frame.setVisible(true); } }

3.Adding Components to the Panel

We'll add a label, text field, and button to the panel.

```
import javax.swing.*;
public class SimpleSwingApp {
public static void main(String[] args) {
JFrame frame = new JFrame("Simple Swing App");
frame.setSize(400, 300);
frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
JPanel panel = new JPanel();
frame.add(panel);
placeComponents(panel);
frame.setVisible(true);
} private static void placeComponents(JPanel panel)
{
panel.setLayout(null); JLabel userLabel = new JLabel("User:");
```

```
userLabel.setBounds(10, 20, 80, 25);

panel.add(userLabel);

JTextField userText = new JTextField(20);

userText.setBounds(100, 20, 165, 25);

panel.add(userText);

JButton loginButton = new JButton("Login");

loginButton.setBounds(10, 80, 80, 25);

panel.add(loginButton); } }
```

4. Adding Event Handling

Add an action listener to the button to display a message when it is clicked.

```
import javax.swing.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
public class SimpleSwingApp {
public static void main(String[] args)
{
JFrame frame = new JFrame("Simple Swing App");
 frame.setSize(400,300);
frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
JPanel panel = new JPanel();
frame.add(panel); placeComponents(panel);
frame.setVisible(true);
}
```

```
private static void placeComponents(JPanel panel)
{
panel.setLayout(null);
JLabel userLabel = new JLabel("User:");
userLabel.setBounds(10, 20, 80, 25);
panel.add(userLabel);
JTextField userText = new JTextField(20);
userText.setBounds(100, 20, 165, 25);
panel.add(userText); JButton loginButton = new JButton("Login");
loginButton.setBounds(10,80,80,25);
panel.add(loginButton); loginButton.addActionListener(new ActionListener()
{
```

```
public void actionPerformed(ActionEvent e)

{

 String username = userText.getText();

JOptionPane.showMessageDialog(null, "Hello, " + username + "!"); } }); } }
```

```
import javax.swing.*;
 import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
public class SimpleSwingApp
{
 public static void main(String[] args)
{
JFrame frame = new JFrame("Simple Swing App");
frame.setSize(400, 300);
frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
JPanel panel = new JPanel(); frame.add(panel);
placeComponents(panel);
frame.setVisible(true); }
private static void placeComponents(JPanel panel)
{
panel.setLayout(null);
```

```
JLabel userLabel = new JLabel("User:");
userLabel.setBounds(10, 20, 80, 25);
panel.add(userLabel);
JTextField userText = new JTextField(20);
userText.setBounds(100, 20, 165, 25);
panel.add(userText);
JButton loginButton = new JButton("Login");
loginButton.setBounds(10, 80, 80, 25);
panel.add(loginButton);
loginButton.addActionListener(new ActionListener()
{
 public void actionPerformed(ActionEvent e)
{
String username = userText.getText();
JOptionPane.showMessageDialog(null, "Hello, " + username + "!"); } }); } }
```

Running the Application

To run this application:

1.Save the code to a file named SimpleSwingApp.java.

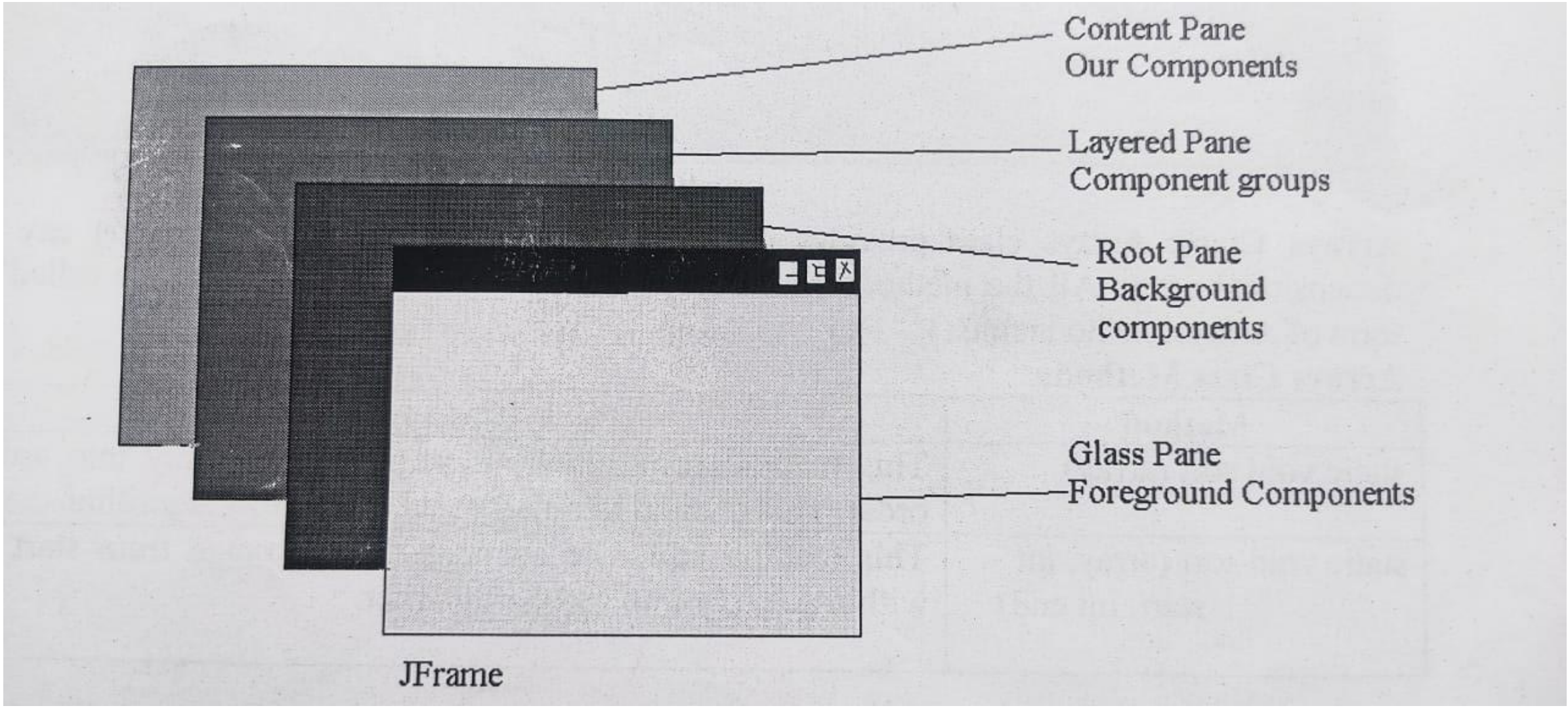
2.Compile the file using the command: javac SimpleSwingApp.java.

3.Run the compiled class using: java SimpleSwingApp.

This application creates a simple window with a text field and a button. When the button is clicked, a dialog box displays a message with the entered username.

Listeners and Listener Methods: Listeners are available for components. A Listener is an interface that listens to an event from a component. Listeners are available in java.awt.event package. The methods in the listener interface are to be implemented, when using that listener.

Component	Listener	Listener methods
Button	ActionListener	public void actionPerformed (ActionEvent e)
Checkbox	ItemListener	public void itemStateChanged (ItemEvent e)
CheckboxGroup	ItemListener	public void itemStateChanged (ItemEvent e)
TextField	ActionListener FocusListener	public void actionPerformed (ActionEvent e) public void focusGained (FocusEvent e) public void focusLost (FocusEvent e)
TextArea	ActionListener FocusListener	public void actionPerformed (ActionEvent e) public void focusGained (FocusEvent e) public void focusLost (FocusEvent e)
Choice	ActionListener ItemListener	public void actionPerformed (ActionEvent e) public void itemStateChanged (ItemEvent e)
List	ActionListener ItemListener	public void actionPerformed (ActionEvent e) public void itemStateChanged (ItemEvent e)
Scrollbar	AdjustmentListener MouseMotionListener	public void adjustmentValueChanged (AdjustmentEvent e) public void mouseDragged (MouseEvent e) public void mouseMoved (MouseEvent e)
Label	No listener is needed	


Content Pane
Our Components
Layered Pane
Component groups
Root Pane
Background
components
Glass Pane
Foreground Components
JFrame

- **Glass Pane:** This is the first pane and is very close to the monitors screen. Any components to be displayed in the foreground are attached to this glass pane. To reach this glass pane, we use getGlassPane () method of JFrame class.
- **Root Pane:** This pane is below the glass pane. Any components to be displayed in the background are displayed in this pane. Root pane and glass pane are used in animations also. For example, suppose we want to display a flying aeroplane in the sky. The aeroplane can be displayed as a .gif or .jpg file in the glass pane where as the blue sky can be displayed in the root pane in the background. To reach this root pane, we use getRootPane () method of JFrame class.
- **Layered Pane:** This pane lies below the root pane. When we want to take several components as a group, we attach them in the layered pane. We can reach this pane by calling getLayeredPane () method of JFrame class.
- **Content Pane:** This is the bottom most pane of all. Individual components are attached to this pane. To reach this pane, we can call getContentPane () method of JFrame class.

Java Database Connectivity (JDBC):

- **Is** a Java-based API that allows Java applications to interact with databases.
- JDBC provides a standard **interface** for connecting to **relational databases**, **executing SQL queries**, and retrieving results.
- **Key Components of JDBC**

1.JDBC Driver

- A JDBC driver is a software component that enables Java applications to interact with a database. Different databases require different JDBC drivers.
- Types of JDBC Drivers:
  - Type 1: JDBC-ODBC Bridge Driver
  - Type 2: Native-API Driver
  - Type 3: Network Protocol Driver
  - Type 4: Thin Driver (Pure Java driver)

2.Connection

- Represents a connection to a specific database. It is used to create statements and manage transactions.
- Example: Connection conn = DriverManager.getConnection(url, user, password);

3.Statement

- Used to execute SQL queries against the database. There are three types:
  - Statement: Used for simple SQL statements without parameters.
  - PreparedStatement: Used for precompiled SQL statements with parameters.
  - CallableStatement: Used to execute stored procedures.

4.ResultSet

- Represents the result set of a query. It is used to iterate over the data returned by the query.
- Example: ResultSet rs = stmt.executeQuery("SELECT * FROM users");

Basic Steps to Use JDBC

1.Load the JDBC Driver

- Ensure that the JDBC driver is available in your classpath. For example, for MySQL, you would need mysql-connector-java.jar.
- Load the driver class (not necessary for newer JDBC versions as the driver auto-registers).

```
try { Class.forName("com.mysql.cj.jdbc.Driver"); } catch (ClassNotFoundException e) {
e.printStackTrace(); }
```

2.Establish a Connection

- Use DriverManager.getConnection() to establish a connection to the database.

```
java
String url = "jdbc:mysql://localhost:3306/mydatabase";
String user = "username";
String password = "password";
Connection conn = null;
try { conn = DriverManager.getConnection(url, user, password); } catch (SQLException e) {
e.printStackTrace(); }
```

3.Create a Statement

- Use the Connection object to create a Statement object.

Statement stmt = null; try { stmt = conn.createStatement(); } catch (SQLException e) { e.printStackTrace(); }

4.Execute a Query

- Execute SQL queries using the Statement object and obtain the result.

ResultSet rs = null; try { rs = stmt.executeQuery("SELECT * FROM users"); } catch (SQLException e) { e.printStackTrace(); }

5.Process the ResultSet

- Iterate over the ResultSet to process the data returned by the query.

try { while (rs.next()) { int id = rs.getInt("id"); String name = rs.getString("name"); System.out.println("ID: " + id + ", Name: " + name); } } catch (SQLException e) { e.printStackTrace(); }

6.Close the Resources

- Close the ResultSet, Statement, and Connection objects to free up resources.

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

Example: JDBC Program to Query a Database

Here's a complete example that demonstrates how to use JDBC to connect to a MySQL database, execute a query, and process the results:

- **Type 1**: JDBC-ODBC Bridge (rarely used, deprecated in Java 8).
- **Type 2**: Native-API Driver (platform-dependent, requires native libraries).
- **Type 3**: Network Protocol Driver (uses middleware server).
- **Type 4**: Thin Driver (pure Java, recommended for most applications).

JDBC (Java Database Connectivity), statements

- Statements are used to execute SQL queries or commands in a database.
- There are three main types of statements:

1. Statement:

To execute simple SQL queries that don’t require any parameters.

Methods:

- executeQuery(String sql) - For SELECT queries.
- executeUpdate(String sql) - For INSERT, UPDATE, DELETE, and DDL commands.
- execute(String sql) - For any SQL command.

Example:

- Statement stmt = connection.createStatement();
- ResultSet rs = stmt.executeQuery("SELECT * FROM employees");

Limitations:

- Not suitable for dynamic or parameterized queries.

- Less secure.

2. **PreparedStatement**

- To execute **precompiled SQL statements** with **dynamic parameters**.

Methods:

- set<Type>(int parameterIndex, <Type> value) - To set parameter values.
- executeQuery() - For SELECT queries.
- executeUpdate() - For INSERT, UPDATE, DELETE, and DDL commands.

Example:

- String query = "SELECT * FROM employees WHERE department = ?";
- PreparedStatement pstmt = connection.prepareStatement(query);
- pstmt.setString(1, "Sales");
- ResultSet rs = pstmt.executeQuery();

Advantages:

- Prevents SQL injection.
- Better performance for repeated execution.
- Easier to work with dynamic queries.

3. CallableStatement

- Purpose: Used to execute stored procedures in the database.

Methods:

- set<Type>(int parameterIndex, <Type> value) - To set input parameter values.
- registerOutParameter(int parameterIndex, int sqlType) - To register output parameters.
- execute() - To execute the stored procedure.

Example:

- CallableStatement cstmt = connection.prepareCall("{call getEmployeeDetails(?)}");
- cstmt.setInt(1, 101);
- ResultSet rs = cstmt.executeQuery();

Advantages:

- Facilitates the use of complex SQL logic encapsulated in stored procedures.
- Reduces network traffic when dealing with complex operations.

- Each type has its unique use cases and should be chosen based on the specific requirements of the application.

Type	Use Case	Security	Performance
Statement	Simple queries without parameters	Vulnerable to SQL Injection	Slower due to query compilation every time
PreparedStatement	Queries with parameters, frequent execution	Prevents SQL Injection	Faster for repeated queries
CallableStatement	Executing stored procedures	Safe for use with stored procedures	Depends on procedure logic

DB Connectivity Steps