REVA UNIVERSITY
Bengaluru, India

M23DE0202 – Object Oriented Programming using JAVA

II Semester MCA - D Academic Year : 2024 - 2025

School of Computer Science and Applications

Prasanna Kumar R B
Assistant Professor

www.reva.edu.in

Unit IV

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. Here's an introduction to Java Swing:

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.

Basic Swing Components

Here are some basic Swing components and their uses:

1.JFrame:

1. 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.

Introduction to basic Widgets in Java

Java Swing provides a wide variety of widgets (components) that can be used to build graphical user interfaces (GUIs).

Here's an introduction to some basic and commonly used Swing widgets:

JFrame

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);
} }
```

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

A 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.

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

Let's dive into creating a simple Java Swing application step-by-step to understand the basics.

We'll build an application with a few common Swing components: a frame, panel, button, text field, label, and some event handling.

Step-by-Step Example: Simple Swing Application

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); } }
```

Adding Event Handling

Finally, we'll 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 + "!"); } }); } }
```

Complete Code

Here is the complete code for the simple Swing application:

```
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.

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.

Here’s an overview of the most commonly used layout managers in Swing:

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. Here's an introduction to JDBC, covering the basics you need to get started.

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:

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
 import java.sql.SQLException;
 import java.sql.Statement;
public class JdbcExample {
public static void main(String[] args) {
// Database URL String url = "jdbc:mysql:
//localhost:3306/mydatabase";
String user = "username"; String password = "password";
 Connection conn = null; Statement stmt = null; ResultSet rs = null;
 try { // Load the JDBC driver
 Class.forName("com.mysql.cj.jdbc.Driver");
// Establish a connection conn = DriverManager.getConnection(url, user, password);
// Create a statement stmt = conn.createStatement();
// Execute a query rs = stmt.executeQuery("SELECT * FROM users");
// Process the result set while (rs.next()) { int id = rs.getInt("id"); String name = rs.getString("name");
System.out.println("ID: " + id + ", Name: " + name); } } catch (ClassNotFoundException e) { e.printStackTrace();
 } catch (SQLException e) { e.printStackTrace();
} finally { // Close the resources try { if (rs != null) rs.close(); if (stmt != null) stmt.close();
if (conn != null) conn.close();
} catch (SQLException e) { e.printStackTrace(); } } } }
```

- **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).

DB Connectivity Steps