Unit 11: Java Applications [8 Hrs.]

1. About AWT & Swing

AWT (Abstract Window Toolkit): AWT is Java's original GUI framework. It provides basic components like buttons, labels, and text fields but relies on the native operating system's GUI components, making it platform-dependent.

Swing: Swing is an extension of AWT and is part of the Java Foundation Classes (JFC). It provides a richer set of components that are entirely written in Java, making them platform-independent. Swing components are lightweight and more flexible than AWT.

---

2. JFrame (Top-Level Window in Swing)

Definition: `JFrame` is a top-level container in Swing used to create windows. It provides a framework for adding other Swing components like buttons, labels, and text fields.

Lab 1: JFrame:

```
import javax.swing.*;

public class JFrameExample {
    public static void main(String[] args) {
        // Create a JFrame
        JFrame frame = new JFrame("My First Swing Application");
        frame.setSize(400, 300); // Set size
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE); // Close operation or
frame.setDefaultCloseOperation(3);
        frame.setVisible(true); // Make the frame visible
    }
}
```

Explanation:

- We create a `JFrame` object and set its title, size, and close operation.
- The `setVisible(true)` method makes the frame visible on the screen.

Default values:

1. **`setSize(width, height)`**
    - Default: **0 × 0** (invisible frame)
2. **`setDefaultCloseOperation()`**
    - Default: **`JFrame.HIDE_ON_CLOSE`** (frame hides but application keeps running)
3. **`setVisible(true)`**
    - Default: **false** (frame is invisible)

Sample Output: A window titled "My First Swing Application" with a size of 400x300 pixels.

---

3. Swing Components

a. JLabel

Definition: `JLabel` is used to display a short string or an image on the screen. It is a **non-editable, static display** — meaning users can't change it directly by clicking/typing. JLabel is used to show fixed text or images inside a Swing window — for titles, messages, form labels, images, and more.

Lab 2: JLabel:

```
import javax.swing.*;

public class JLabelExample {
    public static void main(String[] args) {
        JFrame frame = new JFrame("JLabel Example");
        JLabel label = new JLabel("Hello, Swing!"); // Create a JLabel

        frame.setLayout(new FlowLayout(FlowLayout.LEFT));
        frame.add(label); // Add label to the frame
        frame.setSize(300, 200);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setVisible(true);
    }
}
```

Explanation:

- We create a `JLabel` with the text "Hello, Swing!" and add it to the `JFrame`.
- FlowLayout arranges components left-to-right, and when the space runs out, it wraps them to the next line. The default alignment is center, meaning components will be centered within their allocated space.
- frame.setLayout(new FlowLayout()) tells the JFrame to use FlowLayout as the layout manager.

Sample Output: A window displaying the text "Hello, Swing!".

---

b. JTextField

Definition: `JTextField` is a text component that allows the user to **enter or edit a single line** of text. JLabel = static text (just showing something) JTextField = input box (where user types)

Lab 3: JTextField:

```
import javax.swing.*;

public class JTextFieldExample {
    public static void main(String[] args) {
        JFrame frame = new JFrame("JTextField Example");
        JTextField textField = new JTextField("Enter text here"); // Create a
JTextField

        frame.setLayout(new FlowLayout(FlowLayout.LEFT));
        frame.add(textField); // Add text field to the frame
```

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

Explanation:

- We create a `JTextField` with default text and add it to the `JFrame`.

Sample Output: A window with a text field containing the text "Enter text here".

---

c. JButton

Definition: A `JButton` is a button that can trigger actions when clicked. It's one of the most commonly used components in Java Swing for user interaction.

Lab 4: JButton:

```
import javax.swing.*;

public class JButtonExample {
    public static void main(String[] args) {
        JFrame frame = new JFrame("JButton Example");
        JButton button = new JButton("Click Me"); // Create a JButton

        frame.setLayout(new FlowLayout());
        frame.add(button); // Add button to the frame
        frame.setSize(300, 200);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setVisible(true);
    }
}
```

Explanation:

- We create a `JButton` with the label "Click Me" and add it to the `JFrame`.

Sample Output: A window with a button labeled "Click Me".

---

4. Event Handling in Swing Applications

Definition: Event handling is the mechanism to handle user interactions like button clicks, mouse movements, or key presses.

Lab 5: Event Handling in Swing Applications:

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

public class EventHandlingExample {
    public static void main(String[] args) {
        JFrame frame = new JFrame("Event Handling Example");
        JButton button = new JButton("Click Me");
```

```
        button.addActionListener(new ActionListener() {    //implementing the
ActionListener interface using an anonymous inner class
            @Override
            public void actionPerformed(ActionEvent e) {    //implementation
                JOptionPane.showMessageDialog(frame, "Button Clicked!");
            }
        });

        frame.setLayout(new FlowLayout());
        frame.add(button);
        frame.setSize(300, 200);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setVisible(true);
    }
}
```

Explanation:

- We add an `ActionListener` to the `JButton` to handle button click events by implementing the ActionListener interface using an anonymous inner class.
- When the button is clicked, a dialog box displays the message "Button Clicked!".
- **Anonymous inner classes** are a special type of inner class where we don't provide a name for the class. Instead, we define the class inline where it is needed, typically for short-term use.

`JOptionPane` is a class in Java's Swing library that offers an easy way to display pop-up dialogs for various purposes, such as showing messages, requesting input from users, or seeking confirmation. It belongs to the `javax.swing` package and is widely used for creating message boxes in graphical user interface (GUI) applications.

Common Methods of `JOptionPane`:

- **`showMessageDialog()`**: Displays a message to the user.
- **`showInputDialog()`**: Prompts the user to provide input.
- **`showConfirmDialog()`**: Asks the user a yes/no question and returns their response.

Sample Output: A window with a button. Clicking the button shows a dialog box with the message "Button Clicked!".

---

5. Layout Management

In Java Swing, **layout management** means arranging GUI components (like buttons, labels, text fields) inside a container automatically, without manually setting their position (x, y).

Layout managers help organize components neatly and adjust them properly when the window is resized.

Common Layout Managers:

- **FlowLayout**: Places components left to right in a row.
- **BorderLayout**: Divides the container into five areas — North, South, East, West, Center.
- **GridLayout**: Arranges components in a grid (equal-sized cells).

- **BoxLayout**: Arranges components vertically or horizontally.
- **CardLayout**: Displays one component at a time (like flipping cards).

Advantages:

- Makes GUI flexible and responsive.
- Easier to manage compared to manual positioning.
- Adapts to different screen sizes.

a. FlowLayout

Definition: `FlowLayout` arranges components in a row left-to-right flow, one after the other. If the container is resized and there isn't enough space, the components wrap to the next line.

Lab 6: FlowLayout:

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

public class FlowLayoutExample {
    public static void main(String[] args) {
        JFrame frame = new JFrame("FlowLayout Example");
        frame.setLayout(new FlowLayout()); // Set FlowLayout
        frame.add(new JButton("Button 1"));
        frame.add(new JButton("Button 2"));
        frame.add(new JButton("Button 3"));
        frame.setSize(300, 200);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setVisible(true);
    }
}
```

Explanation:

- We set the layout of the `JFrame` to `FlowLayout` and add three buttons.

Sample Output: A window with three buttons arranged in a row.

---

b. BorderLayout

Definition: `BorderLayout` divides the container into five regions: North, South, East, West, and Center.

Lab 7: BorderLayout:

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

public class BorderLayoutExample {
    public static void main(String[] args) {
        JFrame frame = new JFrame("BorderLayout Example");
        frame.setLayout(new BorderLayout()); // Set 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(300, 200);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setVisible(true);
    }
}
```

Explanation:

- We add buttons to the five regions of the `BorderLayout`.

Sample Output: A window with buttons placed in the North, South, East, West, and Center regions.

---

c. GridLayout

Definition: `GridLayout` arranges components in a grid of rows and columns.

Lab 8: GridLayout:

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

public class GridLayoutExample {
    public static void main(String[] args) {
        JFrame frame = new JFrame("GridLayout Example");
        frame.setLayout(new GridLayout(2, 3)); // Set GridLayout with 2 rows and 3 columns
        frame.add(new JButton("Button 1"));
        frame.add(new JButton("Button 2"));
        frame.add(new JButton("Button 3"));
        frame.add(new JButton("Button 4"));
        frame.add(new JButton("Button 5"));
        frame.add(new JButton("Button 6"));
        frame.setSize(300, 200);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setVisible(true);
    }
}
```

Explanation:

- We set the layout to `GridLayout` with 2 rows and 3 columns and add six buttons.

Sample Output: A window with six buttons arranged in a 2x3 grid.

---

6. Advanced Swing Components

a. JCheckBox

Definition: `JCheckBox` is a component that allows the user to select or deselect an option.

Lab 9: JCheckBox:

```java
import javax.swing.*;

public class JCheckBoxExample {
    public static void main(String[] args) {
        JFrame frame = new JFrame("JCheckBox Example");
        JCheckBox checkBox = new JCheckBox("Enable Feature"); // Create a JCheckBox

        frame.setLayout(new FlowLayout());
        frame.add(checkBox); // Add checkbox to the frame
        frame.setSize(300, 200);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setVisible(true);
    }
}
```

Explanation:

- We create a `JCheckBox` with the label "Enable Feature" and add it to the `JFrame`.

Sample Output: A window with a checkbox labeled "Enable Feature".

---

b. JRadioButton

Definition: `JRadioButton` is used to create a group of mutually exclusive options.

Lab 10: JRadioButton:

```java
import javax.swing.*;

public class JRadioButtonExample {
    public static void main(String[] args) {
        JFrame frame = new JFrame("JRadioButton Example");
        JRadioButton radio1 = new JRadioButton("Option 1");
        JRadioButton radio2 = new JRadioButton("Option 2");
        ButtonGroup group = new ButtonGroup(); // Create a ButtonGroup ensuring that only one radio button in the group can be selected at a time

        frame.setLayout(new FlowLayout());
        group.add(radio1);
        group.add(radio2);
        frame.add(radio1);
        frame.add(radio2);
        frame.setSize(300, 200);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setVisible(true);
    }
}
```

Explanation:

- We create two `JRadioButton` objects and add them to a `ButtonGroup` to ensure mutual exclusivity.

Sample Output: A window with two radio buttons labeled "Option 1" and "Option 2".

---

c. JComboBox

Definition: `JComboBox` is a drop-down list that allows the user to select one option from a list.

Lab 11: JComboBox:

```
import javax.swing.*;

public class JComboBoxExample {
    public static void main(String[] args) {
        JFrame frame = new JFrame("JComboBox Example");
        String[] options = {"Option 1", "Option 2", "Option 3"};
        JComboBox<String> comboBox = new JComboBox<>(options); // Create a JComboBox

        frame.setLayout(new FlowLayout());
        frame.add(comboBox); // Add combo box to the frame
        frame.setSize(300, 200);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setVisible(true);
    }
}
```

Explanation:

- We create a `JComboBox` with three options and add it to the `JFrame`.

Sample Output: A window with a drop-down list containing "Option 1", "Option 2", and "Option 3".

---

d. JList

Definition: `JList` is a component that displays a list of items from which the user can select one or more items.

Lab 12: JList:

```
import javax.swing.*;

public class JListExample {
    public static void main(String[] args) {
        JFrame frame = new JFrame("JList Example");
        String[] items = {"Item 1", "Item 2", "Item 3"};
        JList<String> list = new JList<>(items); // Create a JList

        frame.setLayout(new FlowLayout());
        frame.add(new JScrollPane(list)); // Add JList to a JScrollPane
        frame.setSize(300, 200);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setVisible(true);
    }
}
```

Explanation:

- We create a `JList` with three items and add it to a `JScrollPane` for scrolling.

Sample Output: A window with a list containing "Item 1", "Item 2", and "Item 3".

---

7. Key & Mouse Event Handling

Definition: Key and mouse events are used to handle user interactions like key presses and mouse clicks.

Lab 13: Key & Mouse Event Handling:

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

public class KeyMouseEventExample {
    public static void main(String[] args) {
        JFrame frame = new JFrame("Key & Mouse Event Example");
        JTextField textField = new JTextField();
        textField.addKeyListener(new KeyAdapter() {
            @Override
            public void keyPressed(KeyEvent e) {
                System.out.println("Key Pressed: " + e.getKeyChar());
            }
        });
        frame.addMouseListener(new MouseAdapter() {
            @Override
            public void mouseClicked(MouseEvent e) {
                System.out.println("Mouse Clicked at: " + e.getX() + ", " + e.getY());
            }
        });
        frame.add(textField);
        frame.setSize(300, 200);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setVisible(true);
    }
}
```

Explanation:

- We add a `KeyListener` to handle key presses and a `MouseListener` to handle mouse clicks.

Sample Output:

- When a key is pressed, the key character is printed.
- When the mouse is clicked, the coordinates of the click are printed.

Lab 14: Write a GUI program using components to find factorial and cube of number. Use TextField for giving input and Label for output. The program should display factorial if user press mouse on result button and cube if user release mouse from result button.

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

```java
public class FactorialCubeCalculator {
    public static void main(String[] args) {
        // Create frame
        JFrame frame = new JFrame("Factorial and Cube Finder");

        // Create components
        JTextField inputField = new JTextField(10);
        JLabel resultLabel = new JLabel("Result will appear here");
        JButton resultButton = new JButton("Result");

        // Add mouse listener to the button
        resultButton.addMouseListener(new MouseAdapter() {
            public void mousePressed(MouseEvent e) {
                String text = inputField.getText();
                try {
                    int num = Integer.parseInt(text);
                    long fact = 1;
                    for (int i = 1; i <= num; i++) {
                        fact *= i;
                    }
                    resultLabel.setText("Factorial: " + fact);
                } catch (NumberFormatException ex) {
                    resultLabel.setText("Invalid input!");
                }
            }

            public void mouseReleased(MouseEvent e) {
                String text = inputField.getText();
                try {
                    int num = Integer.parseInt(text);
                    long cube = (long) num * num * num;
                    resultLabel.setText("Cube: " + cube);
                } catch (NumberFormatException ex) {
                    resultLabel.setText("Invalid input!");
                }
            }
        });

        // Layout setting
        frame.setLayout(new FlowLayout());
        frame.add(new JLabel("Enter Number:"));
        frame.add(inputField);
        frame.add(resultButton);
        frame.add(resultLabel);

        // Frame settings
        frame.setSize(300, 150);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setVisible(true);
    }
}
```

Lab 15: Write a GUI program using swing components to calculate sum and difference of two numbers. Use two text fields for input and pre-built dialog box for output. Your program should display sum if Add button and difference if subtract button is clicked.

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

public class SumDifferenceCalculator {
    public static void main(String[] args) {
        // Create frame
        JFrame frame = new JFrame("Sum and Difference Calculator");

        // Create components
        JTextField numField1 = new JTextField(10);
        JTextField numField2 = new JTextField(10);
        JButton addButton = new JButton("Add");
        JButton subtractButton = new JButton("Subtract");

        // Add action listener for Add button
        addButton.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                try {
                    int num1 = Integer.parseInt(numField1.getText());
                    int num2 = Integer.parseInt(numField2.getText());
                    int sum = num1 + num2;
                    JOptionPane.showMessageDialog(frame, "Sum: " + sum);
                } catch (NumberFormatException ex) {
                    JOptionPane.showMessageDialog(frame, "Invalid input! Please enter
numbers.");
                }
            }
        });

        // Add action listener for Subtract button
        subtractButton.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                try {
                    int num1 = Integer.parseInt(numField1.getText());
                    int num2 = Integer.parseInt(numField2.getText());
                    int diff = num1 - num2;
                    JOptionPane.showMessageDialog(frame, "Difference: " + diff);
                } catch (NumberFormatException ex) {
                    JOptionPane.showMessageDialog(frame, "Invalid input! Please enter
numbers.");
                }
            }
        });

        // Layout setting
        frame.setLayout(new FlowLayout());
        frame.add(new JLabel("First Number:"));
```

```
        frame.add(numField1);
        frame.add(new JLabel("Second Number:"));
        frame.add(numField2);
        frame.add(addButton);
        frame.add(subtractButton);

        // Frame settings
        frame.setSize(300, 200);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setVisible(true);
    }
}
```

---

8. Menus in Swing

Definition: Menus are used to create dropdown menus in Swing applications.

Lab 16: Menus in Swing:

```
import javax.swing.*;

public class MenuExample {
    public static void main(String[] args) {
        JFrame frame = new JFrame("Menu Example");
        JMenuBar menuBar = new JMenuBar(); // Create a menu bar
        JMenu fileMenu = new JMenu("File"); // Create a menu
        JMenuItem openItem = new JMenuItem("Open"); // Create a menu item
        fileMenu.add(openItem);
        menuBar.add(fileMenu);
        frame.setJMenuBar(menuBar); // Add menu bar to the frame
        frame.setSize(300, 200);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setVisible(true);
    }
}
```

Explanation:

- We create a `JMenuBar`, add a `JMenu` to it, and add a `JMenuItem` to the menu.

Sample Output: A window with a "File" menu containing an "Open" option.

---

9. Dialog Boxes in Swing

Definition: Dialog boxes are used to display messages or prompt the user for input.

Lab 17: Dialog Boxes in Swing:

```
import javax.swing.*;

public class DialogBoxExample {
    public static void main(String[] args) {
        JFrame frame = new JFrame("Dialog Box Example");
        JOptionPane.showMessageDialog(frame, "This is a message dialog!");
```

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

Explanation:

- We use `JOptionPane.showMessageDialog` to display a message dialog.

Sample Output: A dialog box with the message "This is a message dialog!".

---

10. JTable for Displaying Data in Tabular Form

Definition: `JTable` is used to display data in a tabular format.

Lab 18: JTable for Displaying Data in Tabular Form:

```
import javax.swing.*;

public class JTableExample {
    public static void main(String[] args) {
        JFrame frame = new JFrame("JTable Example");
        String[][] data = {{"1", "Sharat", "Maharjan"}, {"2", "Sujan", "Shrestha"}};
        String[] columns = {"ID", "First Name", "Last Name"};
        JTable table = new JTable(data, columns); // Create a JTable
        frame.add(new JScrollPane(table)); // Add JTable to a JScrollPane
        frame.setSize(300, 200);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setVisible(true);
    }
}
```

Explanation:

- We create a `JTable` with data and column names and add it to a `JScrollPane`.
- Tables (JTable) can have **many rows** or **many columns**.
- If the table becomes **bigger than the frame**, we **won't be able to see** everything.
- JScrollPane solves this by **automatically adding scrollbars (both vertical and horizontal)** so that the user can scroll.

Sample Output: A window with a table displaying the data.

---

11. MDI Using JDesktopPane & JInternalFrame

Definition: MDI (Multiple Document Interface) allows multiple windows to be displayed within a single parent window.

Lab 19: MDI Using JDesktopPane & JInternalFrame:

```
import javax.swing.*;

public class MDIExample {
    public static void main(String[] args) {
        JFrame frame = new JFrame("MDI Example");
```

```
        JDesktopPane desktopPane = new JDesktopPane(); // Create a JDesktopPane
        JInternalFrame internalFrame = new JInternalFrame("Internal Frame", true,
true, true, true);
        internalFrame.setSize(200, 150);
        internalFrame.setVisible(true);
        desktopPane.add(internalFrame);
        frame.add(desktopPane);
        frame.setSize(400, 300);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setVisible(true);
    }
}
```

Explanation:

- We create a `JDesktopPane` and add a `JInternalFrame` to it.
- JDesktopPane is a special container in Swing.
- It is used to hold and manage multiple internal frames (JInternalFrame).
- It looks like a mini desktop inside the application.

Sample Output: A window with an internal frame inside it.

---

12. Using IDE like NetBeans

Definition: IDEs like NetBeans provide drag-and-drop functionality for building Java applications quickly.

Steps:

1. Open NetBeans and create a new Java Application.
2. Use the GUI Builder to drag and drop components like buttons, labels, and text fields.
3. Write event-handling code in the generated methods.

---

13. Adapter Classes

Definition: Adapter classes in Java provide default implementations for event listener interfaces. They are used to simplify event handling by allowing us to override only the methods we need, rather than implementing all methods of an interface.

Common Adapter Classes:

- `MouseAdapter`: Provides default implementations for `MouseListener` methods.
- `KeyAdapter`: Provides default implementations for `KeyListener` methods.
- `WindowAdapter`: Provides default implementations for `WindowListener` methods.

---

15. Summary

In this unit, we explored Java's GUI programming using **AWT** and **Swing**. What we covered:

1. **AWT vs. Swing**:
    - AWT is platform-dependent and uses native components.
    - Swing is platform-independent and provides a richer set of components.
2. **Swing Components**:

- We learned about `JFrame`, `JLabel`, `JTextField`, `JButton`, `JCheckBox`, `JRadioButton`, `JComboBox`, and `JList`.

3. **Event Handling**:
   - We used `ActionListener`, `MouseListener`, and `KeyListener` to handle user interactions.
4. **Layout Management**:
   - We explored `FlowLayout`, `BorderLayout`, and `GridLayout` for organizing components.
5. **Advanced Components**:
   - We worked with `JTable` for displaying tabular data, `JMenu` for creating menus, and `JOptionPane` for dialog boxes.
6. **MDI (Multiple Document Interface)**:
   - We used `JDesktopPane` and `JInternalFrame` to create MDI applications.
7. **Adapter Classes**:
   - We used `MouseAdapter` and `KeyAdapter` to simplify event handling.
8. **Using IDEs**:
   - We discussed how IDEs like NetBeans can speed up GUI development with drag-and-drop tools.

Key Differences Between AWT and Swing

Feature	AWT	Swing
Platform Dependency	Uses native OS components (platform-dependent).	Pure Java implementation (platform-independent).
Component Look	Follows the native OS look and feel.	Customizable look and feel.
Performance	Faster for simple applications.	Slightly slower due to Java rendering.
Flexibility	Limited components and customization.	Rich set of components and customization options.