Experiment No. 8

Implementation of GUI using Swing Components

Instructions:

This manual consists of three parts:
A) Theory and Concepts,
B) Problems for Implementation, and
C) Write-up Questions.

1. Students must understand the **theory and concepts** provided before implementing the problem statement(s) for **Experiment 8**.
2. They should **practice the given code snippets** within the theory section.
3. Later, they need to **implement the problems provided**.
4. **Write-up:** Students are required to **write answers** to the questions on journal pages, **maintain a file**, and get it checked regularly. The file should include index, write-up, and implementation code with results.
5. **Referencing**: Include proper sources or references for the content used.
6. Use of Generative AI: Clearly mention if you have used any AI tools (e.g., ChatGPT, Copilot, Gemini) to generate text, explanations, or code. Cite the AI-generated content appropriately in the write-up.

Part A. Theory and Concepts:

GUI in Java

Graphical User Interface (GUI) applications in Java are developed using **AWT** (Abstract Window Toolkit) and **Swing**. Swing is preferred over AWT as it provides more features, better look-and-feel, and is platform-independent.

1) AWT Components:

AWT (Abstract Window Toolkit) contains numerous classes and methods that allow you to create and manage windows.

- AWT classes are contained in the **java.awt.*** package.

Component	Explanation	Example code
Label.	Used to display static text on a GUI	`Label myLabel = new Label("Hello World");`
Button	Generates an event when clicked.	`Button btn = new Button("Click Me");`
Checkbox	Allows users to select or deselect an option. Generates an ItemEvent.	`Checkbox chk = new Checkbox("Accept Terms");`

Checkbox Group	Creates a group of mutually exclusive checkboxes (radio buttons).	`CheckboxGroup group = new CheckboxGroup();`
		`Checkbox cb1 = new Checkbox("Male", group, true);`
		`Checkbox cb2 = new Checkbox("Female", group, false);`

Choice	Provides a dropdown list for selection.	`Choice choice = new Choice();`
		`choice.add("Option 1");`
		`choice.add("Option 2");`

List	Displays multiple selectable items in a scrolling list.	`List list = new List(3, true);`
		`list.add("Item 1");`
		`list.add("Item 2");`

Scrollbar	Allows users to select continuous values using a slider.	`Scrollbar sb = new Scrollbar(Scrollbar.HORIZONTAL, 50, 10, 0, 100);`

TextArea, TextField	TextArea:A multi-line text input field.
	TextField: A single-line text input field.
		`TextArea ta = new TextArea(5, 20);`
		
		`TextField tf = new TextField(20);`

2) Swing Components:

Swing API is a set of extensible GUI Components to ease the developer's life to create JAVA based Front End/GUI Applications. It is built on top of AWT API and acts as a replacement of AWT API, since it has almost every control corresponding to AWT controls. Swing component follows a Model-View-Controller architecture to fulfil the following criteria.

- A single API is to be sufficient to support multiple look and feel.
- API is to be model driven so that the highest level API is not required to have data.
- API is to use the Java Bean model so that Builder Tools and IDE can provide better services to the developers for use.

Swing Features

- **Lightweight**: Independent of OS API.
- **Rich Components**: Includes trees, sliders, color pickers.
- **Customizable**: Easy visual modifications.
- **Pluggable Look-and-Feel**: Change UI appearance dynamically.

3) MVC Architecture

Swing API architecture follows loosely based MVC architecture in the following manner.

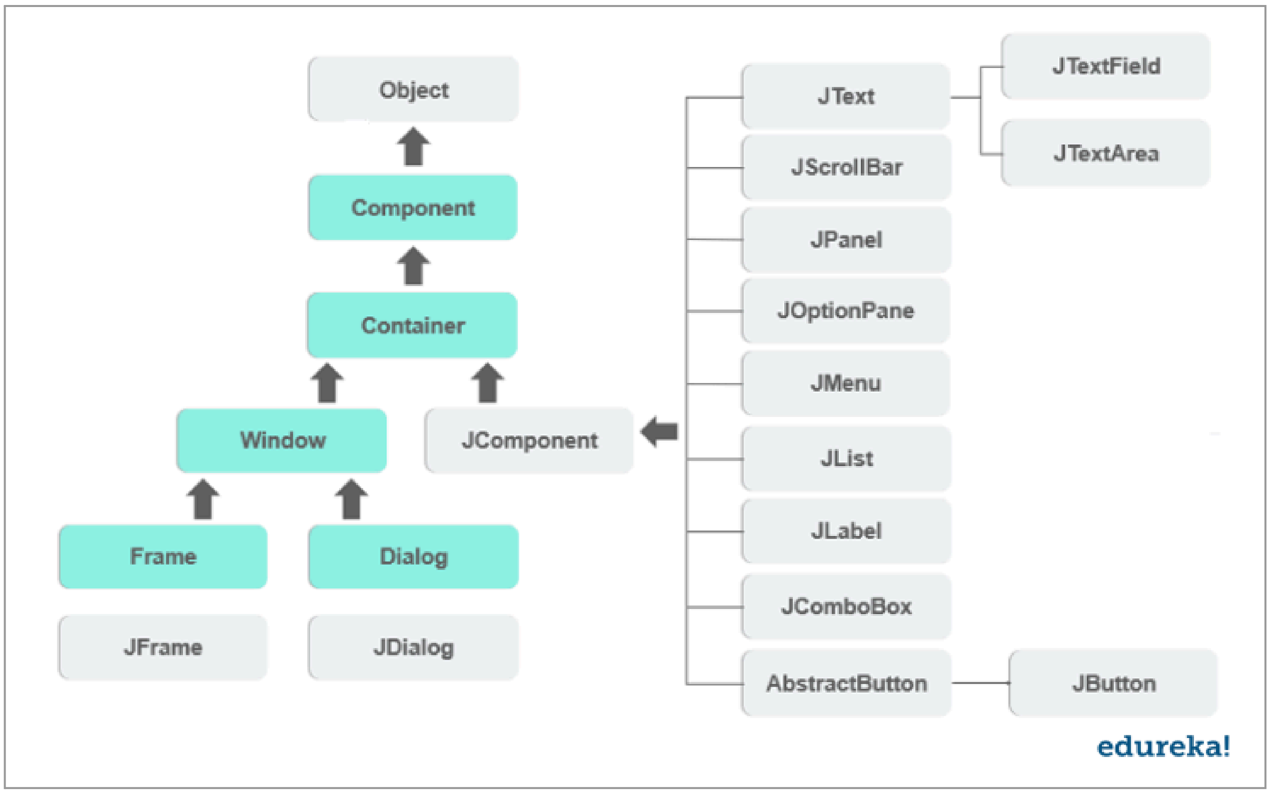
- Model represents the component's data.
- View represents visual representation of the component's data.
- Controller takes the input from the user on the view and reflects the changes in Component's data.
- Swing component has Model as a separate element, while the View and Controller part are clubbed in the User Interface elements. Because of which, Swing has a pluggable look-and-feel architecture.
- Quick Tutorial for understanding MVC architecture: MVC Explained in 4 Minutes

4) Component hierarchy

Every user interface considers the following three main aspects −

- **UI Elements** − These are the core visual elements the user eventually sees and interacts with. GWT provides a huge list of widely used and common elements varying from basic to complex.
- **Layouts** − They define how UI elements should be organized on the screen and provide a final look and feel to the GUI (Graphical User Interface).
- **Behavior** − These are the events which occur when the user interacts with UI elements.

Every SWING controls inherit properties from the following Component class hierarchy.



5) Swing Components in Java:

Quick Read: https://www.naukri.com/code360/library/swing-components-in-java

Example 1: Swing application without using any IDE:

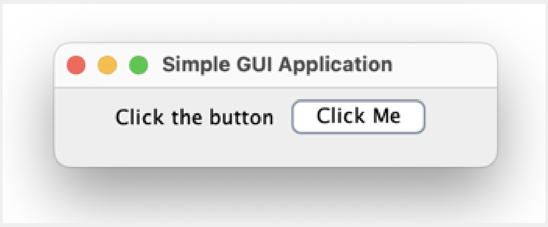
```
import javax.swing.*; // Import Swing components
import java.awt.*; // Import AWT classes for layout
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

public class SimpleGUI {
    public static void main(String[] args) {
        // Create the frame (main window)
        JFrame frame = new JFrame("Simple GUI Application");
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
           // Close the app when window is closed
        frame.setSize(400, 200); // Set frame size
        frame.setLayout(new FlowLayout());
           // Set layout manager to FlowLayout

        // Create a label with initial text
        JLabel label = new JLabel("Click the button");
        frame.add(label); // Add label to frame
        // Create a button with text
        JButton button = new JButton("Click Me");
        frame.add(button); // Add button to frame

        // Add an ActionListener to handle button clicks
        button.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                // Update label text when button is clicked
                label.setText("Button Clicked!");
            }
        });

        // Make the frame visible
        frame.setVisible(true);
    }
}
```

Commands to execute the code

```
javac SimpleGUI.java
java SimpleGUI
```

Output

```
It opens a window with a label saying "Click the button" and a
"Click Me" button.
```



Example 2: Swing application implementation using IDE:

Swing application development can be done using various **IDEs** like **VS Code**, **IntelliJ IDEA**, **Eclipse**, and **NetBeans**. In this case, we will demonstrate the implementation using **IntelliJ IDEA.**

Note:	***IDE*** *stands for* ***Integrated Development Environment.*** *It is a software application that provides comprehensive facilities to computer programmers for software development, including a code editor, compiler or interpreter, debugger, and build tools—all in one place.*

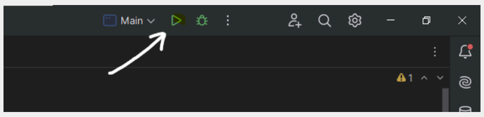
INSTRUCTIONS:

1. **Install IntelliJ IDEA** – Follow the quick video tutorial for installation.
    - https://www.youtube.com/watch?v=oJ9OdOgdTIg

2. **Create a New Project** – Name it as per naming conventions.
3. **Explore the Folder Structure** – You will see a src folder and a Main.java file.
4. **Write Your Code** – Use the existing Main.java file or create a new one.

```
import javax.swing.*;  // Import Swing components
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

public class Main {
    public static void main(String[] args) {
        // Create the main application window
        JFrame frame = new JFrame("IntelliJ Swing Example");
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
              // Exit application on close
        frame.setSize(300, 200); // Set window size

        JPanel panel = new JPanel();
           // Create a panel to hold components
        frame.add(panel); // Add panel to the frame

        // Create and add a label prompting for input
        JLabel label = new JLabel("Enter text:");
        panel.add(label);

        // Create and add a text field with 10 columns
        JTextField textField = new JTextField(10);
        panel.add(textField);

        // Create and add a button labeled "Submit"
        JButton button = new JButton("Submit");
        panel.add(button);

        // Create and add a label to display the result
        JLabel resultLabel = new JLabel("");
        panel.add(resultLabel);

        // Add action listener to the button
        button.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                // Get text from the text field
                String inputText = textField.getText();

                // Display the entered text in result label
                resultLabel.setText("You entered: " + inputText);
            }
        });
        // Make the frame visible
        frame.setVisible(true);
    }
}
```

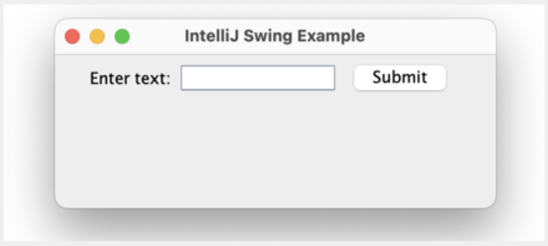
Execution of the code:



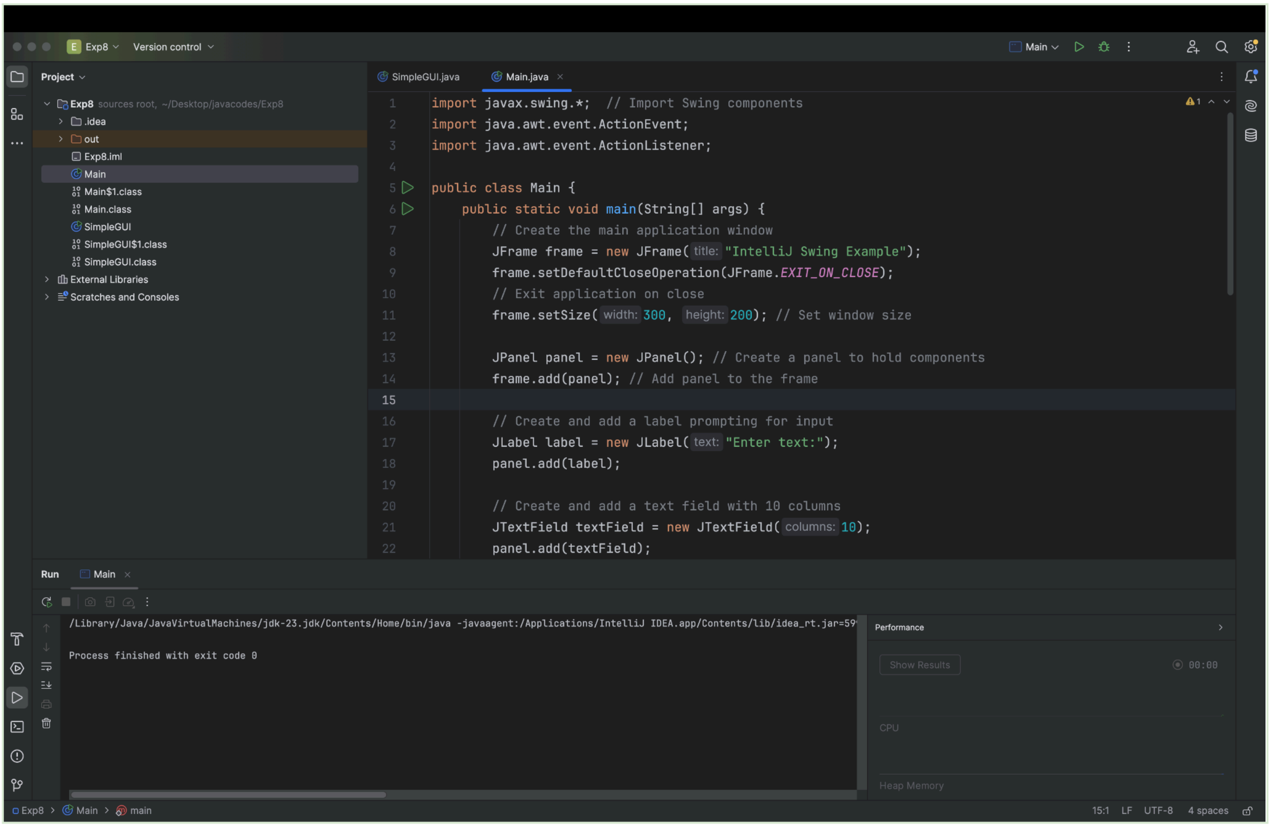
Use the run button to compile and run your GUI application

Output

A window appears with a "Submit" button and a text field.



Below is the full screen shot of **IntelliJ IDEA** while executing the Main.java code.



```
import javax.swing.*;  // Import Swing components
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

public class Main {
    public static void main(String[] args) {
        // Create the main application window
        JFrame frame = new JFrame( title: "IntelliJ Swing Example");
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        // Exit application on close
        frame.setSize( width: 300,  height: 200); // Set window size

        JPanel panel = new JPanel(); // Create a panel to hold components
        frame.add(panel); // Add panel to the frame

        // Create and add a label prompting for input
        JLabel label = new JLabel( text: "Enter text:");
        panel.add(label);

        // Create and add a text field with 10 columns
        JTextField textField = new JTextField( columns: 10);
        panel.add(textField);
```

```
/Library/Java/JavaVirtualMachines/jdk-23.jdk/Contents/Home/bin/java -javaagent:/Applications/IntelliJ IDEA.app/Contents/lib/idea_rt.jar=59

Process finished with exit code 0
```

Example 3: Factorial using Swing

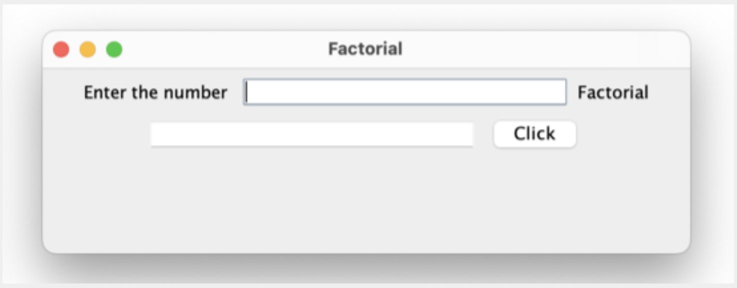
```
import java.awt.*; // Import AWT components
import java.awt.event.*; // Import AWT event classes
import javax.swing.*; // Import Swing components

// Class extending JFrame and implementing ActionListener for
handling button click
class factdemo1 extends JFrame implements ActionListener
{
    JButton b1;        // Button to trigger factorial calculation
    JLabel l1, l2;     // Labels for input and output
    JTextField t1, t2;
      // TextFields for input number and displaying factorial

    public factdemo1()
    {
        setTitle("Factorial"); // Set the window title

        // Initialize GUI components
        l1 = new JLabel("Enter the number ");
        t1 = new JTextField(20);
        l2 = new JLabel("Factorial");
        t2 = new JTextField(20);
        b1 = new JButton("Click");

        setSize(400, 400); // Set window size
        setLayout(new FlowLayout());
           // Use FlowLayout for arranging components

        // Add components to the frame
        add(l1);
        add(t1);
        add(l2);
        add(t2);
        add(b1);

        // Add action listener to the button
        b1.addActionListener(this);

        // Make the frame visible and set default close operation
        setVisible(true);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }

    // Method called when button is clicked
    public void actionPerformed(ActionEvent e)
    {
        int num = Integer.parseInt(t1.getText());
           // Parse input from text field

        if(e.getSource() == b1)
           // Check if the event source is the button
        {
            int fact = 1, i;
            for(i = num; i >= 1; i--)
           // Compute factorial using loop
                fact = fact * i;

            t2.setText(String.valueOf(fact));
                // Display the result in the second text field
        }
    }

    // Main method to launch the application
    public static void main(String args[])
    {
        factdemo1 f = new factdemo1();
    }
}
```

Commands to execute the code

```
javac factdemo1.java
java factdemo1
```

Output

```
Output:
- A window appears with two text fields and a button.
- User enters a number, clicks the button, and the factorial appears
in the second field.
```



Example 4: Checkbox demo

```
import java.awt.*; // AWT classes for GUI components
import java.awt.event.*; // AWT event classes
import javax.swing.*; // Swing classes for modern GUI components
```

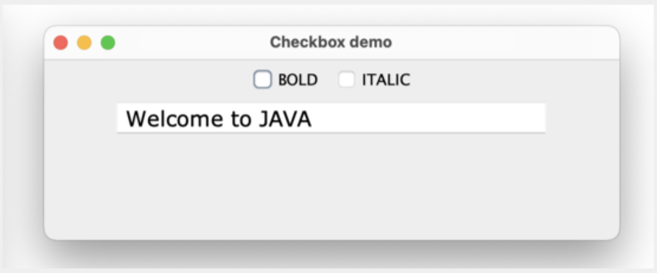
```
// Class extending JFrame and implementing ItemListener to respond
to checkbox events

class cbdemo extends JFrame implements ItemListener
{
    JTextField t1;         // Text field to display styled text
    JCheckBox b1, b2, b3;  // Checkboxes for font styling

    public cbdemo()
    {
        setTitle("Checkbox demo"); // Set window title

        // Create a text field with default message
        t1 = new JTextField(" Welcome to JAVA", 20);
        Font f = new Font("Verdana", Font.PLAIN, 18);
                             // Default font

        t1.setFont(f);

        // Create checkboxes for BOLD and ITALIC
        b1 = new JCheckBox("BOLD");
        b2 = new JCheckBox("ITALIC");

        // Add item listeners to checkboxes
        b1.addItemListener(this);
        b2.addItemListener(this);

        // Set up the frame layout and size
        setSize(500, 500);
        setLayout(new FlowLayout());

        // Add components to the frame
        add(b1);
        add(b2);
        add(t1);

        // Make the frame visible and close properly
        setVisible(true);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }

    // This method is called when a checkbox state changes
    public void itemStateChanged(ItemEvent e)
    {
        int a, b;

        // Check if BOLD is selected; if not, use PLAIN
        a = ((b1.isSelected()) ? Font.BOLD : Font.PLAIN);
        // Check if ITALIC is selected; if not, use PLAIN
        b = ((b2.isSelected()) ? Font.ITALIC : Font.PLAIN);

        // Combine font styles using bitwise OR
        // (or addition as here)
        Font f1 = new Font("Verdana", a + b, 18);

        // Apply the new font to the text field
        t1.setFont(f1);
    }

    // Main method to launch the application
    public static void main(String args[])
    {
        cbdemo r = new cbdemo();
    }
}
```

Commands to execute the code

```
javac cbdemo.java
java cbdemo
```

Output

When the application starts, a window appears with two checkboxes labeled **BOLD** and **ITALIC**, and a text field displaying *“Welcome to JAVA”*.



```
Welcome to JAVA
```

Additional Learning Resources:

1. **Video tutorial:** :https://www.youtube.com/watch?v=5o3fMLPY7qY
   (Note: Try implementing the example discussed in the tutorial)

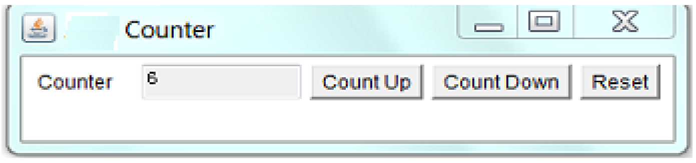
2. **Difference between AWT and Swing:** https://medium.com/@srishaunique/exploring-java-swing-vs-awt-choosing-the-right-gui-toolkit-for-your-java-projects-c51730828780

3. **GUI using Swing:** https://medium.com/@rolandmack63/introduction-to-java-gui-programming-with-swing-c9bedda86ee8

Part B. Problems for Implementation:

Aim: Implement a GUI using Swing Components.

1. Implement the following GUI without any IDE.



2. Write a GUI program to find the reverse of a given number using Swing (with IDE).
3. Write a GUI program to demonstrate the use of radio buttons (e.g., gender selection).

Part C. Write-up Questions:

1. What is Java Swing? List its features.
2. Explain the MVC architecture in Swing.
3. What are the key differences between AWT and Swing?
4. Write a Java program to create a simple form using Swing components.

Conclusion:

After completing this exercise, students should be able to design interactive GUIs for desktop applications using Swing components like buttons, checkboxes, radio buttons, and text fields.