Unit 7: GUI Programming with Swing

Introduction to graphical user interface (GUI) programming in Java

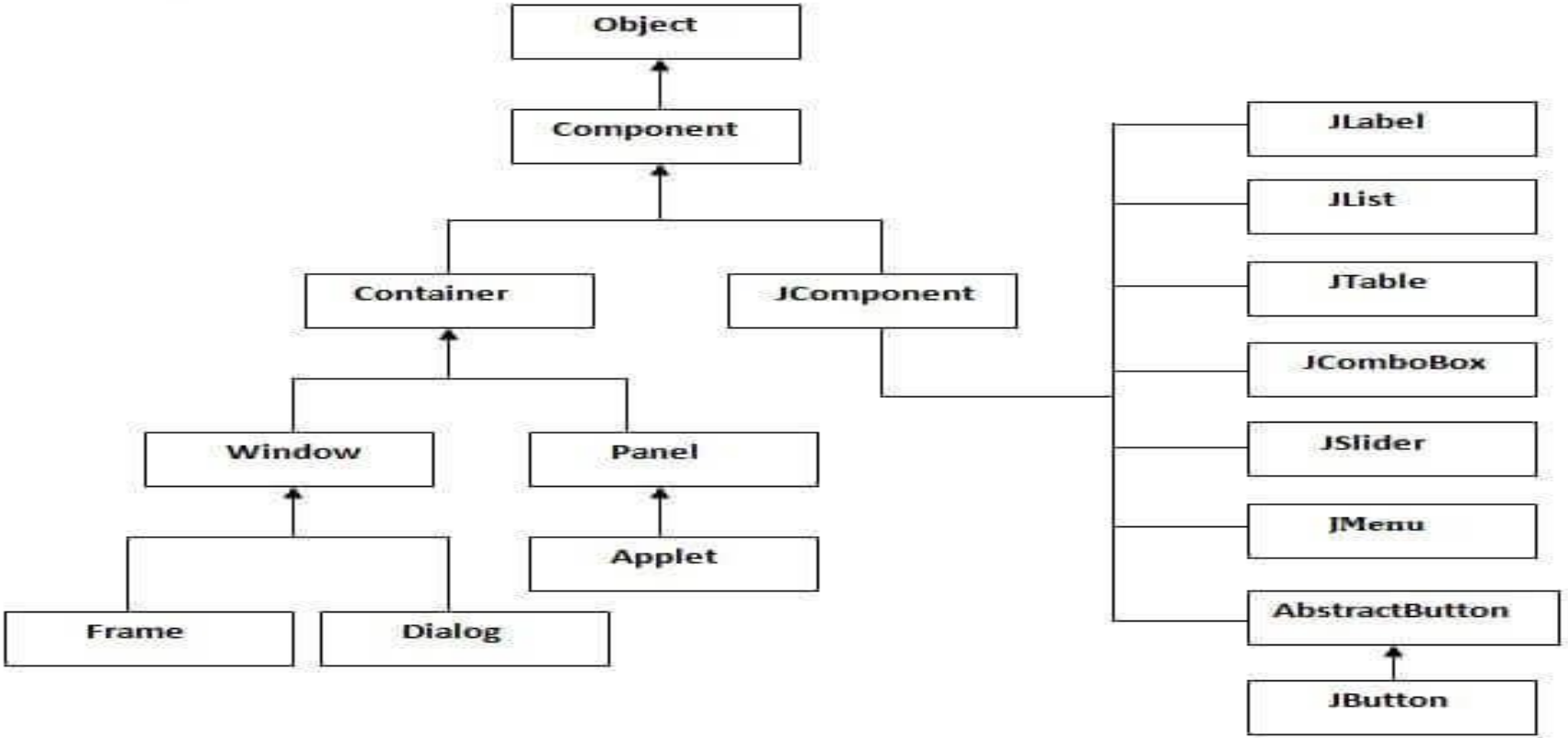
- Graphical User Interface (GUI) programming in Java involves creating interactive applications that allow users to interact with the application through graphical components like buttons, text fields, labels, etc.
- GUI plays an important role to build easy interfaces for Java applications.
- Java provides two main libraries for building GUIs: Abstract Window Toolkit (AWT) and Swing.
- Swing is a Graphical User Interface (GUI) toolkit that includes the GUI components.
- Swing is built on top of AWT and provides more sophisticated and flexible components.
- Swing provides a rich set of widgets and packages to make sophisticated GUI components for Java applications.

Key concepts:

- **Components**: Basic building blocks of a GUI application, like buttons, labels, text fields, etc.
- **Containers**: Special components that can hold other components, such as JFrame, JPanel, etc.
- **Layouts**: Define how components are arranged in a container.
- **Event Handling**: Mechanism to handle user actions like button clicks, mouse movements, etc.

References: https://www.javatpoint.com/java-swing

Hierarchy of Java Swing classes



Difference between Java Swing and Java AWT

Feature	Java Swing	Java AWT
Architecture	Platform-Independent	Platform-Dependent
Look and Feel	Pluggable look and feel	Native look and feel
Components	Richer set of components	Basic set of components
Performance	Slower due to software rendering	Faster due to native OS rendering
Event Model	More flexible and powerful	Simpler and less powerful
Thread Safety	By default, it is not thread-safe	Thread-safe by default
Customization	Highly customizable	Less customizable
Layout Managers	More layout managers are available	Fewer layout managers are available
API	Extensive API with many features	Basic API with fewer features
Graphics Support	It supports more advanced graphics	It only supports basic graphics
File Size	Size is large due to additional APIs	Size is small due to fewer APIs and classes

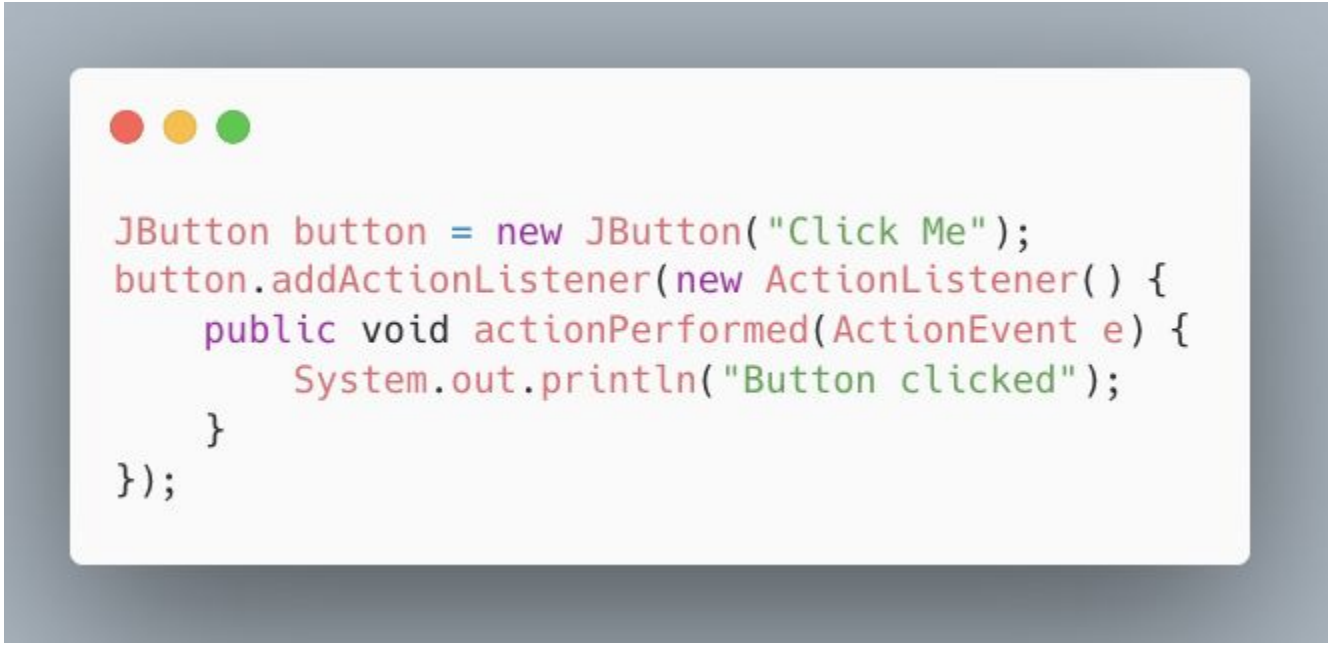
Using Swing components (buttons, labels, text fields, text area, etc.)

Swing components include buttons, tables, text components, and all the rest.

You can use the Java simple GUI programming components like button, textbox, etc., from the library and do not have to create the components from scratch.

The tutorial "Using Swing Components" tells you how to use each of the Swing components and how to use borders and icons.

Buttons (JButton)



```
JButton button = new JButton("Click Me");
button.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        System.out.println("Button clicked");
    }
});
```

Labels (JLabel)

Text Fields (JTextField)

```
JTextField textField = new JTextField(20);   // 20 columns wide
```

Text Areas (JTextArea)

```
JTextArea textArea = new JTextArea(5, 20); // 5 rows and 20 columns
```

Panels (JPanel)

Example: Creating a Simple GUI with Swing

```java
import javax.swing.*;

public class SimpleGUI {
    public static void main(String[] args) {
        JFrame frame = new JFrame("Simple GUI");
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setSize(300, 200);

        JPanel panel = new JPanel();
        JLabel label = new JLabel("Enter your name:");
        JTextField textField = new JTextField(20);
        JButton button = new JButton("Submit");

        button.addActionListener(e -> {
            String name = textField.getText();
            JOptionPane.showMessageDialog(frame, "Hello, " + name);
        });

        panel.add(label);
        panel.add(textField);
        panel.add(button);
        frame.add(panel);

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

Event-driven programming and event handling: Mouse event and key event

Event-driven programming is a paradigm where the flow of the program is determined by events such as user actions (clicks, key presses, etc.).

In Java, this is handled using listeners that respond to specific events.

Mouse Event

To handle mouse events, you can use the MouseListener interface.

```java
import java.awt.event.MouseAdapter;
import java.awt.event.MouseEvent;

import javax.swing.JFrame;

public class MouseEventExample {
    public static void main(String[] args) {
        JFrame frame = new JFrame("Mouse Event Example");
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

        frame.addMouseListener(new MouseAdapter() {
            @Override
            public void mouseClicked(MouseEvent e) {
                System.out.println("Mouse clicked at (" + e.getX() + ", " + e.getY() + ")");
            }

            @Override
            public void mousePressed(MouseEvent e) {
                System.out.println("Mouse pressed at (" + e.getX() + ", " + e.getY() + ")");
            }

            @Override
            public void mouseReleased(MouseEvent e) {
                System.out.println("Mouse released at (" + e.getX() + ", " + e.getY() + ")");
            }
        });

        frame.setSize(400, 300);
        frame.setVisible(true);
    }
}
```

Key Event

To handle key events, you can use the KeyListener interface.

```java
import java.awt.event.KeyAdapter;
import java.awt.event.KeyEvent;
import javax.swing.JFrame;

public class KeyEventExample {
    public static void main(String[] args) {
        JFrame frame = new JFrame("Keyboard Event Example");
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

        frame.addKeyListener(new KeyAdapter() {
            @Override
            public void keyPressed(KeyEvent e) {
                System.out.println("Key pressed: " + e.getKeyChar());
            }

            @Override
            public void keyReleased(KeyEvent e) {
                System.out.println("Key released: " + e.getKeyChar());
            }
        });

        frame.setSize(400, 300);
        frame.setVisible(true);
    }
}
```

Building simple interactive applications

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

public class InteractiveApp {
    public static void main(String[] args) {
        JFrame frame = new JFrame("Interactive App");
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setSize(400, 300);

        JPanel panel = new JPanel();
        panel.setLayout(new GridLayout(3, 2));

        JLabel label = new JLabel("Enter text:");
        JTextField textField = new JTextField(20);
        JTextArea textArea = new JTextArea(5, 20);
        JButton button = new JButton("Show Text");

        button.addActionListener(e -> {
            String text = textField.getText();
            textArea.setText(text);
        });

        textField.addKeyListener(new KeyAdapter() {
            public void keyTyped(KeyEvent e) {
                char keyChar = e.getKeyChar();
                if (!Character.isLetterOrDigit(keyChar)) {
                    e.consume();
                }
            }
        });

        panel.add(label);
        panel.add(textField);
        panel.add(button);
        panel.add(new JScrollPane(textArea));

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

This example demonstrates a simple interactive application where a user can type text into a text field, press a button, and see the text displayed in a text area. The KeyAdapter is used to filter non-alphanumeric characters.

DIY: Layout Managers

https://www.javatpoint.com/java-layout-manager

Practical

1. Create a simple Java Swing application that opens a window with the title "Hello GUI". The window should have a fixed size of 400x300 pixels and close when the user clicks the close button.
2. Create a Java Swing application that contains a JButton labeled "Click Me". When the button is clicked, a message dialog should display "Button Clicked".
3. Write a Java Swing application that contains a JLabel and a JTextField. When the user types in the text field, the label should update to display the text entered.
4. Create a Java Swing application with a JButton. Use a MouseListener to detect when the mouse enters and exits the button, changing its text to "Mouse Entered" and "Mouse Exited" respectively.
5. Create a Java Swing application that contains a JTextField. Use a KeyListener to detect when a key is typed, and display the typed character in a JLabel.
6. Build a simple calculator application using Java Swing that allows the user to add, subtract, multiply, and divide two numbers. Use JTextField for input and JButton for operations.
7. Create a simple Swing application that simulates a login form. The form should contain JTextField for username, JPasswordField for password, and a JButton for submitting. When the button is clicked, check if the username is "admin" and the password is "password". Display a message dialog indicating success or failure.