Week 2: Event Handling & Swing Applets

Introduction to Event Handling in Swing

Event handling is a very important part of making Java programs interactive. It's how your program reacts to what the user does. For example, when someone clicks a button, types in a text field, or moves the mouse, the program can do something in response, like show a message or perform a task.

In **Swing**, the part of Java used to create graphical user interfaces (GUIs), event handling helps your program know when and how to respond to user actions.

The Event Delegation Model (Breaking it Down)

Java uses something called the **event delegation model** to manage these user actions. Let's break down the main parts of this model:

1. **Source**: This is the component in your program that causes or "triggers" an event.
    - **Example**: A button (`JButton`) that someone clicks. The button is the "source" of the event because it's the thing being interacted with.
2. **Event Object**: When something happens (like a button being clicked), an **event object** is created. This object holds information about what just happened.
    - **Example**: If someone clicks a button, an `ActionEvent` object is created. If the user moves the mouse, a `MouseEvent` object is created. These objects tell the program what action occurred.
3. **Event Listener**: This is like a helper that listens for an event to happen. It's a special piece of code that "waits" for the user to do something (like click a button). Once the event happens, the **listener** steps in and tells the program how to respond.
    - **Example**: If you have a button in your program, you can attach an `ActionListener` to it. The `ActionListener` listens for clicks on that button, and when it hears a click, it performs a task (like showing a message).

How Event Handling Works

1. **The Source** (e.g., a button) is clicked.
    - An **Event Object** is created (like an `ActionEvent` for a button click).
2. The **Event Listener** (attached to the button) "hears" the event and performs an action in response (e.g., changes text, shows a message).

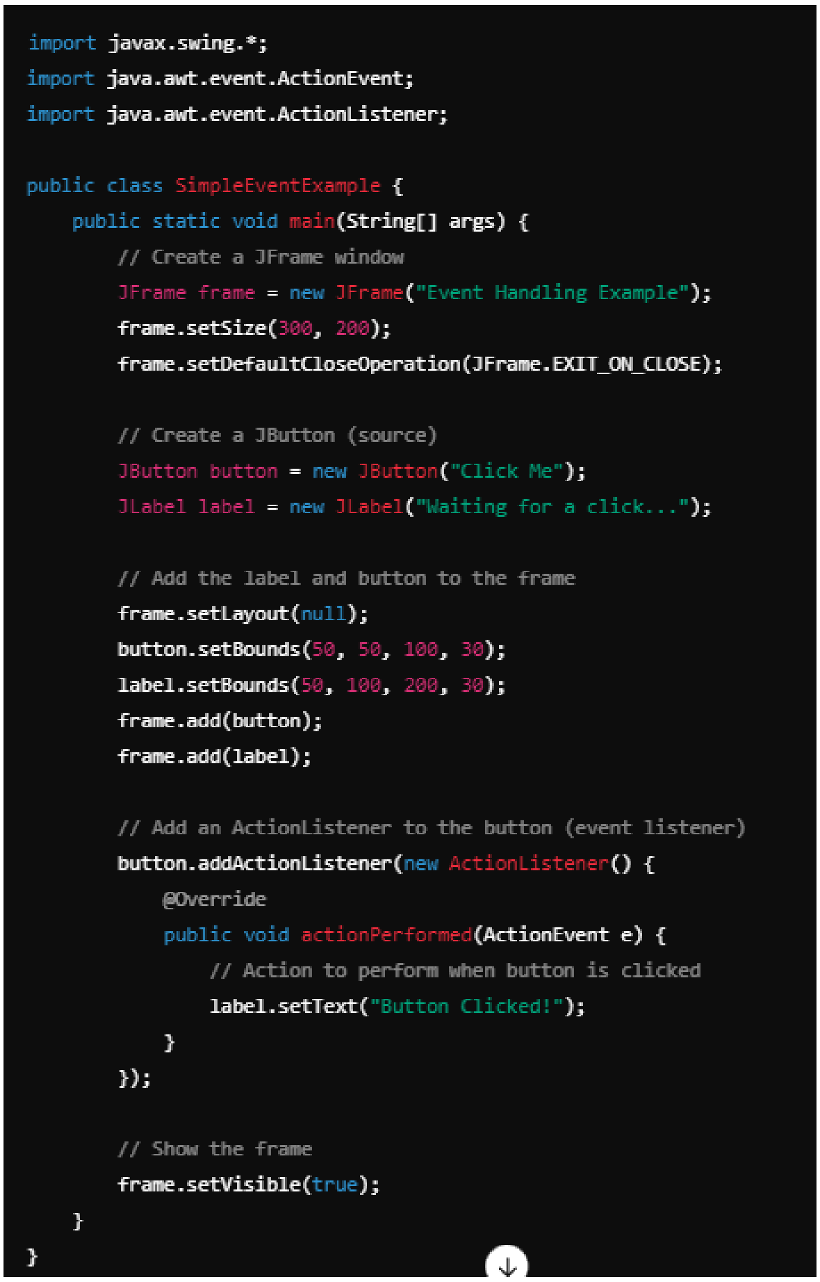
Example of Event Handling in Action

Let's say you have a button in your program, and when someone clicks that button, you want a label to display "Button Clicked!". Here's what happens:

1. **Source**: The button is clicked by the user.
2. **Event Object**: An `ActionEvent` is created because a button was clicked.

3. **Event Listener**: The program has a listener (`ActionListener`) that waits for this click. Once the listener knows the button was clicked, it runs code to update the label text.

Here's a simple Java code example to make this clearer:



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

public class SimpleEventExample {
    public static void main(String[] args) {
        // Create a JFrame window
        JFrame frame = new JFrame("Event Handling Example");
        frame.setSize(300, 200);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

        // Create a JButton (source)
        JButton button = new JButton("Click Me");
        JLabel label = new JLabel("Waiting for a click...");

        // Add the label and button to the frame
        frame.setLayout(null);
        button.setBounds(50, 50, 100, 30);
        label.setBounds(50, 100, 200, 30);
        frame.add(button);
        frame.add(label);

        // Add an ActionListener to the button (event listener)
        button.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                // Action to perform when button is clicked
                label.setText("Button Clicked!");
            }
        });

        // Show the frame
        frame.setVisible(true);
    }
}
```

How the Example Works:

- **Source**: The `button` is the source. When clicked, it triggers an event.
- **Event Object**: When the button is clicked, an `ActionEvent` is created.
- **Event Listener**: The `ActionListener` attached to the button listens for the click. When the button is clicked, it changes the text of the `label` to "Button Clicked!".

The Event Delegation Model

The event delegation model has three key participants:

1. **Event Source**: This is the GUI component that triggers the event (e.g., JButton, JTextField).
2. **Event Object**: Encapsulates information about the event that occurred. Java provides specific classes for different types of events like ActionEvent, KeyEvent, and MouseEvent.
3. **Event Listener**: An interface that processes the event. The listener is linked to a component to "listen" for certain events.

Common event listener interfaces include:

- **ActionListener**: Handles action events like button clicks.
- **MouseListener**: Handles mouse events like clicks and movement.
- **KeyListener**: Handles keyboard input events.

Implementing Event Handling in NetBeans IDE

To handle events in Java, you typically:

1. **Register** the component with an event listener.
2. **Implement** the listener interface by overriding the required methods.
3. **Perform** actions inside the event handler when the event occurs.

Example 1: Handling Button Click Events

Step-by-Step Example in NetBeans:

1. **Create a New Java Application Project**:
    - Open NetBeans and go to File > New Project.
    - Select Java under the Categories section and choose Java Application under Projects.
    - Name the project "EventHandlingExample" and click Finish.
2. **Create a Swing Form**:
    - Right-click on the Source Packages folder in the Projects window and select New > JFrame Form.
    - Name the form EventHandlingForm and click Finish.
3. **Design the GUI**:
    - Use the drag-and-drop functionality in NetBeans to add components from the Palette window. Add:
        - A JLabel (for displaying messages).
        - A JButton (which will trigger an event).

   - You can drag components to the form and resize them as needed.

4. **Add an Event Listener to the Button**:
   - Right-click on the JButton and choose Events > Action > actionPerformed. NetBeans will auto-generate the event handling method for the button click.

5. **Write the Event Handling Code**:
   - Inside the actionPerformed() method, write the code to update the JLabel text when the button is clicked.

Here is the complete code:

```
// Inside EventHandlingForm.java
import javax.swing.*;


public class EventHandlingForm extends javax.swing.JFrame {
   public EventHandlingForm() {
      initComponents();
   }
   @SuppressWarnings("unchecked")
   private void initComponents() {
      // Create components
      jLabel1 = new javax.swing.JLabel();
      jButton1 = new javax.swing.JButton();
      setDefaultCloseOperation(javax.swing.WindowConstants.EXIT_ON_CLOSE);
      jLabel1.setText("Click the Button!");


      jButton1.setText("Click Me");
      jButton1.addActionListener(new java.awt.event.ActionListener() {
         public void actionPerformed(java.awt.event.ActionEvent evt) {
            jButton1ActionPerformed(evt);
         }
      });
      // Layout setup
      javax.swing.GroupLayout layout = new
javax.swing.GroupLayout(getContentPane());
      getContentPane().setLayout(layout);
```

```
        layout.setHorizontalGroup(
          layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
          .addGroup(layout.createSequentialGroup()
            .addGap(100, 100, 100)

    .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
              .addComponent(jButton1)
              .addComponent(jLabel1))
            .addContainerGap(100, Short.MAX_VALUE))
        );
        layout.setVerticalGroup(
          layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
          .addGroup(layout.createSequentialGroup()
            .addGap(50, 50, 50)
            .addComponent(jLabel1)
            .addGap(50, 50, 50)
            .addComponent(jButton1)
            .addContainerGap(50, Short.MAX_VALUE))
        );

        pack();
      }
      private void jButton1ActionPerformed(java.awt.event.ActionEvent evt) {
        // This method is triggered when the button is clicked
        jLabel1.setText("Button Clicked!");
      }
      // Variables declaration - do not modify
      private javax.swing.JButton jButton1;
      private javax.swing.JLabel jLabel1;
      // End of variables declaration
    }
```

Swing Applets

A **Swing Applet** is a Java application that runs in a web browser or an applet viewer. Applets were once widely used for web-based interactive applications, but they are now largely deprecated due to security concerns. However, applets are still useful for learning GUI programming.

Basic Structure of an Applet

- Applets extend javax.swing.JApplet (or java.applet.Applet for AWT).
- They override the init() method to initialize the applet.

Example 2: A Simple Swing Applet

To create an applet, follow these steps in NetBeans:

1. **Create a New Java Class**:
    - Right-click on Source Packages and select New > Java Class.
    - Name the class SimpleApplet and click Finish.
2. **Write the Applet Code**:

```
import javax.swing.JApplet;
import javax.swing.JButton;
import javax.swing.JLabel;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

public class SimpleApplet extends JApplet {
   private JLabel label;
   private JButton button;

   @Override
   public void init() {
      // Initialize the applet with a button and a label
      label = new JLabel("Click the Button");
      button = new JButton("Click Me");

      // Set the layout and add components
      setLayout(null);
```

```
        label.setBounds(50, 50, 150, 20);
        button.setBounds(50, 100, 100, 30);
        add(label);
        add(button);

        // Add event listener to button
        button.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                label.setText("Button Clicked!");
            }
        });
    }
}
```

```java
import javax.swing.JApplet;
import javax.swing.JButton;
import javax.swing.JLabel;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

public class SimpleApplet extends JApplet {
    private JLabel label;
    private JButton button;

    @Override
    public void init() {
        // Initialize the applet with a button and a label
        label = new JLabel("Click the Button");
        button = new JButton("Click Me");

        // Set the layout and add components
        setLayout(null);
        label.setBounds(50, 50, 150, 20);
        button.setBounds(50, 100, 100, 30);
        add(label);
        add(button);

        // Add event listener to button
        button.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                label.setText("Button Clicked!");
            }
        });
    }
}
```

3. **Run the Applet**:
    - To run the applet in NetBeans, you'll need an applet viewer since browsers no longer support applets. Right-click on the class and select Run.

Exercises for Week 2

Exercise 1: Create an Event-Driven Application

- **Objective**: Create a Java Swing application with multiple buttons. Each button should update a different label with a different message.
- **Steps**:
    1. Add three JButton components to your form.
    2. Add three corresponding JLabel components.

3. Write event handling code for each button to update the text of a different label when clicked.

- **Expected Output**: Clicking each button updates its respective label with a different message.

Exercise 2: Develop a Basic Swing Calculator

- **Objective**: Create a simple calculator using Swing with buttons for digits (0-9), addition, subtraction, and equals.
- **Steps**:
  1. Create a form with JButton components for the numbers and operations.
  2. Add a JTextField to display the result.
  3. Implement event handling so that clicking on a number or operator updates the result in the text field.
- **Expected Output**: A basic calculator that performs addition and subtraction operations when buttons are clicked.

Exercise 3: Create a Swing Applet with Event Handling

- **Objective**: Develop a Swing applet that includes two buttons and a label. Each button click should update the label with a different message.
- **Steps**:
  1. Create a JApplet class and override the init() method.
  2. Add two buttons and one label to the applet.
  3. Implement event handling for each button to change the label text.
- **Expected Output**: An applet where clicking either button updates the label with a different message.

Additional Resources

- **NetBeans IDE Official Documentation**: NetBeans Documentation
- **Java Event Handling Documentation**: Java Event Handling (Oracle)
- **Java Applet Tutorial**: Java Applet Basics

Recommended Books

- Holzner Steven (2005), *JAVA 2 Programming Black Book*, DreamTech
- Wigglesworth and Lumby (2002), *JAVA Programming*, NCC