**National University of Computer & Emerging Sciences, Karachi**

**Computer Science Department**

**Fall 2022, Lab Manual - 03**

|  |  |
| --- | --- |
| **Course Code: SL3001** | **Course: Software Development and construction** |
| **Instructor:** | **Miss Nida Munawar** |

**Lab # 03**

**SWING API:**

It contains 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 API :

* Provides a pluggable look and feel
* Offer a wide variety of controls (buttons, text fields, sliders, color pickers, trees, etc)
* Is light weight. It does not use native code resources unlike AWT

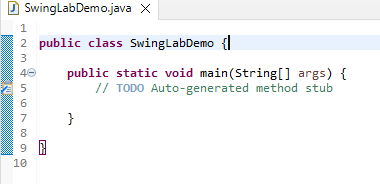
Following is the hierarchy of the classes present in the Java Swing API



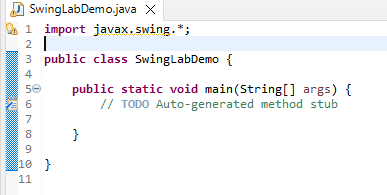
**TUTORIAL**

In this manual we first learn how to create a simple desktop application following a step-by-step approach.

**Step 1: Create a java project called SwingLabDemo (with a main class) in your Eclipse workspace.**



**Step 2: Import the swing library: import javax.swing.\***



**Step 3: Create a Frame by adding these lines of code to your main function:**

JFrame f=**new** JFrame();f.setSize(500,500);f.setVisible(**true**);

**Step 4: Create a JLabel, JTextField and JButton inside the JFrame by adding the following lines of code before the setSize statement:**

JLabel lbl = **new** JLabel("Name: ");

lbl.setBounds(100,100,40, 40);

f.add(lbl);

JTextField txtFld = **new** JTextField();

txtFld.setBounds(150, 100, 200, 30);

f.add(txtFld);

JButton btn = **new** JButton("Click Me!");

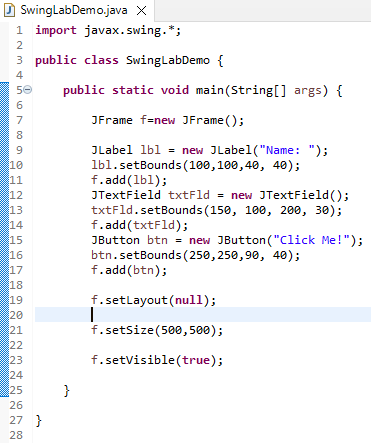
btn.setBounds(250,250,90, 40);

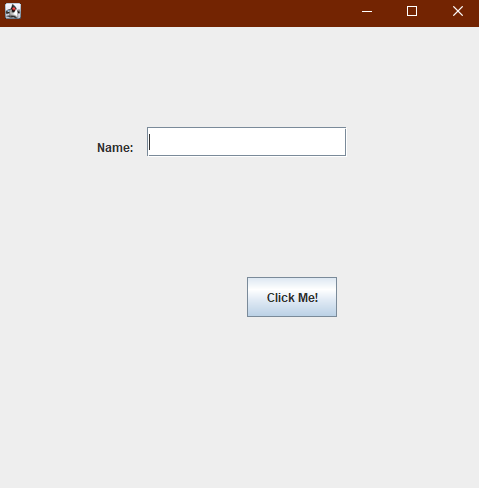
f.add(btn);

The setBounds(int xaxis, int yaxis, int width, int height) is used to set the location, height and width of the components.

**Step 5: Add the following statement after the above code:**

f.setLayout(**null**);

**Step 6: Run your Program. The final program and its output should look like this:**

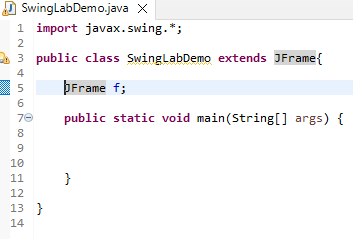


**EXAMPLE OF SWING BY INHERITANCE AND CONSTRUCTORS**

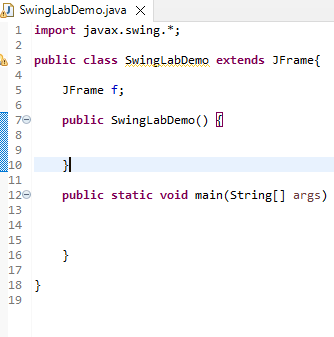
**Step 1: Copy the code from the main function and keep it copied to your clipboard.**

**Step 2: Remove the body of the main function.**

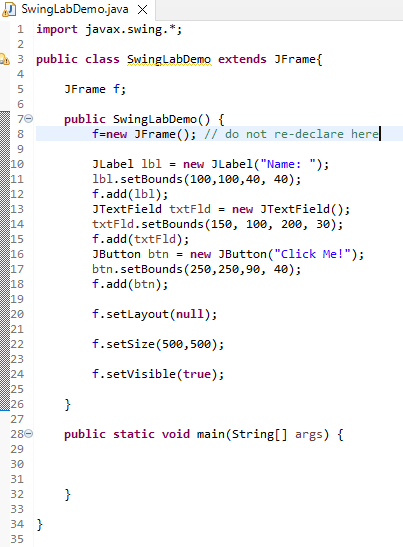
**Step 3: Inherit your SwingLabDemo class from JFrame and Create a JFrame variable within your class.**



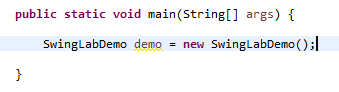
**Step 4: Create a constructor for your SwingLabDemo class.**



**Step 5: Copy the code from your clipboard and paste it within the constructor. However, do not re-declare your JFrame variable.**



**Step 6: Call your constructor from the main function of your class.**



**EVENT HANDLING**

Before we continue with other types of components, let’s first see how we can interact with buttons and text fields with the concept of event handling.

Event handling makes use of 3 major concepts. **Source, Event and Listener.**

A **source** is responsible for generating and **event.**

An **event** notifies the **listener** about any change that has occurred in the **source**.

A **listener** is responsible for accepting the **event** and processing it. Before a **listener** can receive any events, it must first be registered to the source of the event.

**EVENT LISTENERS**

Listener Interfaces provide methods to handle different events. An event listener exists for multiple type of events. For example, for mouse, keyboard, etc.

The mouse listener provides the following methods that you can define when you implement the interface:

* **public** **void** mouseClicked(MouseEvent e);
* **public** **void** mousePressed(MouseEvent e);
* **public** **void** mouseReleased(MouseEvent e);
* **public** **void** mouseEntered(MouseEvent e);
* **public** **void** mouseExited(MouseEvent e);

**Handling Mouse-Based Events Using Listener Interfaces in AWT**

Let’s go through an example of how to handle mouse-based events using listener interfaces:

**public** **class** MouseEventsDemo **extends** Frame **implements** MouseListener, MouseMotionListener {

String msg = "";

**int** mouseX = 0, mouseY = 0;

**public** MouseEventsDemo() {

addMouseListener(**this**);

addMouseMotionListener(**this**);

addWindowListener(**new** MyWindowAdapter());

}

@Override

**public** **void** mouseDragged(MouseEvent e) {

// mouse dragged

mouseX = e.getX();

mouseY = e.getY();

msg = "\*" + "mouse at " + mouseX + ", " + mouseY;

repaint();

}

@Override

**public** **void** mouseMoved(MouseEvent e) {

// mouse moved

msg = "Moving mouse at " + e.getX() + ", " + e.getY();

repaint();

}

@Override

**public** **void** mouseClicked(MouseEvent e) {

// mouse clicked

msg = msg + " -- click received";

repaint();

}

@Override

**public** **void** mousePressed(MouseEvent e) {

// mouse button pressed

mouseX = e.getX();

mouseY = e.getY();

msg = "Button Down";

repaint();

}

@Override

**public** **void** mouseReleased(MouseEvent e) {

// mouse button released

mouseX = e.getX();

mouseY = e.getY();

msg = "Button Released";

repaint();

}

@Override

**public** **void** mouseEntered(MouseEvent e) {

// mouse entered

mouseX = 100;

mouseY = 100;

msg = "Mouse Entered";

repaint();

}

@Override

**public** **void** mouseExited(MouseEvent e) {

// mouse exited

mouseX = 100;

mouseY = 100;

msg = "Mouse Exited";

repaint();

}

**public** **void** paint(Graphics g) {

g.drawString(msg, mouseX, mouseY);

}

**public** **static** **void** main(String args[]) {

MouseEventsDemo appwin = **new** MouseEventsDemo();

appwin.setSize(**new** Dimension(300,300));

appwin.setTitle("Mouse Events Demo");

appwin.setVisible(**true**);

}

}

**class** MyWindowAdapter **extends** WindowAdapter {

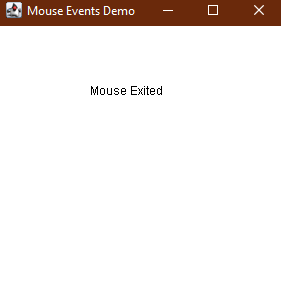
**public** **void** windowClosing (WindowEvent we) {

System.*exit*(0);

}

}

**OUTPUT:**



**ACTION LISTENER**

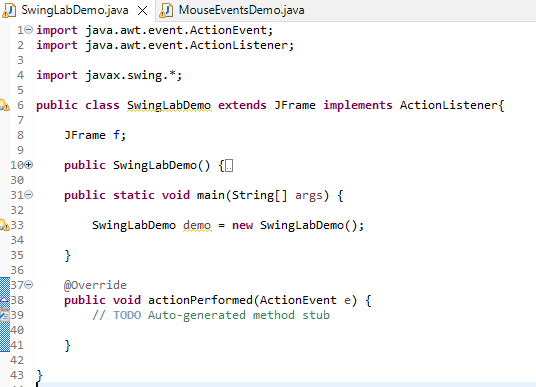
ActionListener in Java is a class that is responsible for handling all action events such as when the user clicks on a component. Mostly, action listeners are used for JButtons. An ActionListener can be used by the implements keyword to the class definition.

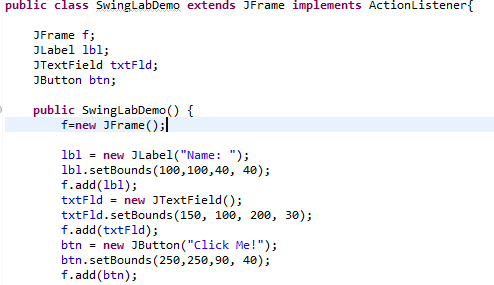
**Handling Button-Based Events Using ActionListener in Swing**

Using the code from the first example, we can continue to see how to handle events.

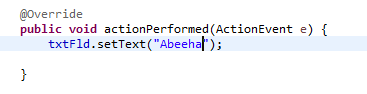
**Step 1: Add the “implements** ActionListener**” statement with the declaration of your class.**

**Step 2: It should display and error, click it and select “add unimplemented methods”. It will add a new method public** **void** actionPerformed(ActionEvent e) **in your code.**



**Step 3: Declare your Text Field, Label and Button variables outside of the constructor so that they can accessed outside of the constructor. Your class will not look like this:** 

**Step 4: Define the actionPerformed method.**



**Step 5: Run your program and see what happens!**