

Course Code: SL3001	Course: Software Development and construction
Instructor(s):	Miss Nida Munawar, Miss Abeeha Sattar

Lab # 03

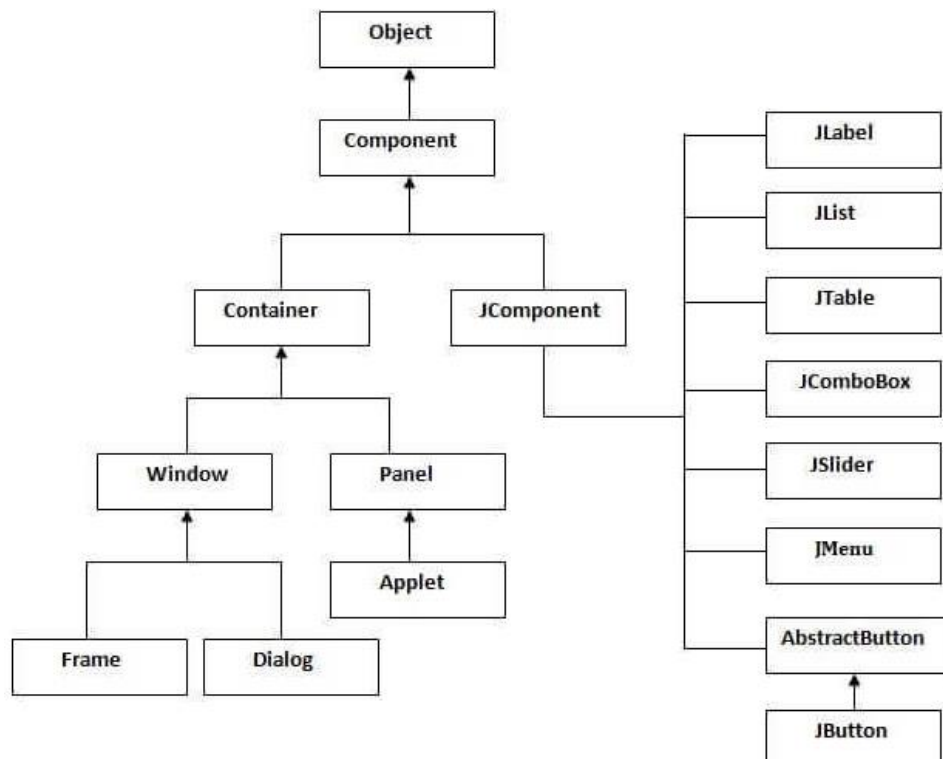
SWING API:

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

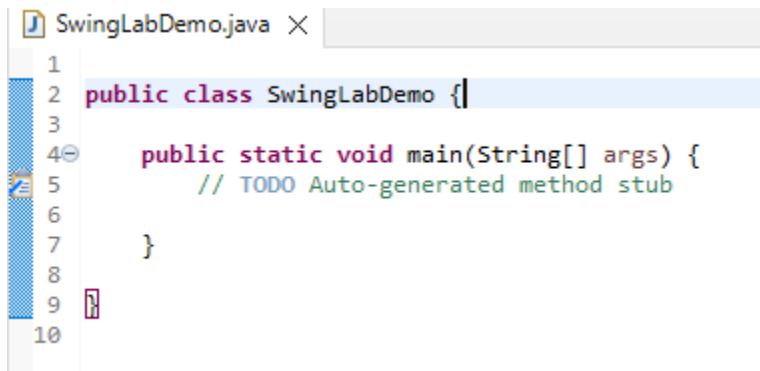
Following is the hierarchy of the classes present in the Java Swing API



TUTORIAL

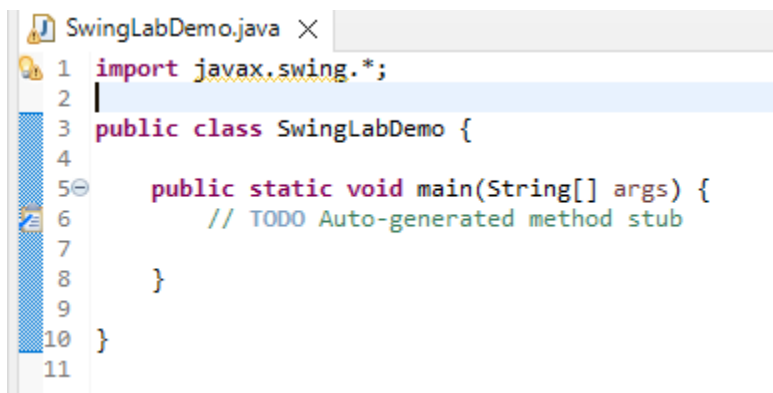
In this manual we first learn how to create a simple desktop application following a step-by-step approach.

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



```
SwingLabDemo.java X
1
2 public class SwingLabDemo {
3
4     public static void main(String[] args) {
5         // TODO Auto-generated method stub
6
7     }
8
9
10
```

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



```
SwingLabDemo.java X
1 import javax.swing.*;
2
3 public class SwingLabDemo {
4
5     public static void main(String[] args) {
6         // TODO Auto-generated method stub
7
8     }
9
10 }
11
```

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:

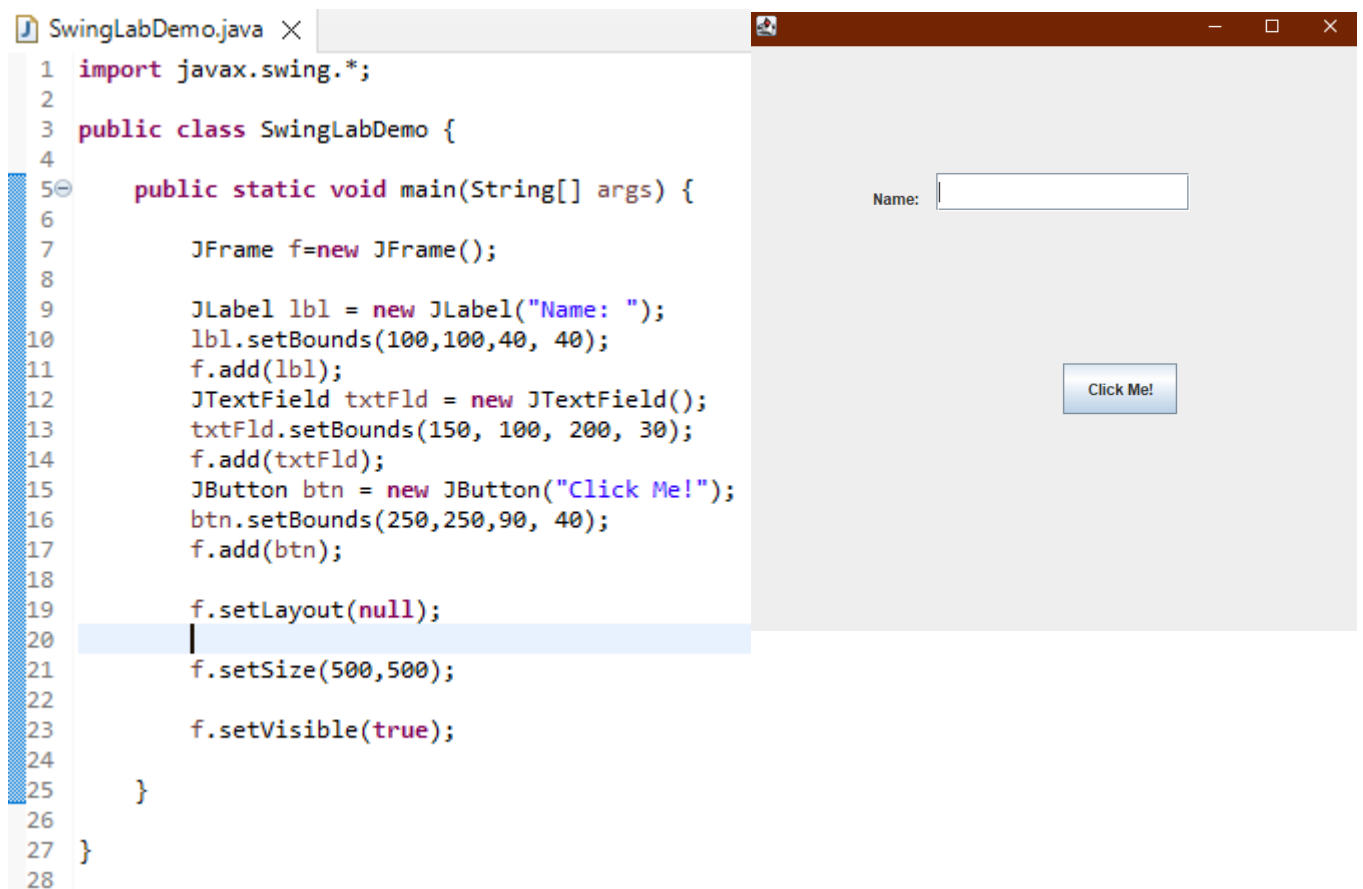
```
JLabel lbl = new JLabel("Name: ");
lbl.setBounds(100,100,40, 40);
f.add(lbl);
JTextField txtFld = new JTextField();
txtFld.setBounds(150, 100, 200, 30);
f.add(txtFld);
JButton btn = new JButton("Click Me!");
btn.setBounds(250,250,90, 40);
f.add(btn);
```

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.

```
SwingLabDemo.java X
1  import javax.swing.*;
2
3  public class SwingLabDemo extends JFrame{
4
5      JFrame f;
6
7      public static void main(String[] args) {
8
9
10
11     }
12
13 }
14
```

Step 4: Create a constructor for your SwingLabDemo class.

```
SwingLabDemo.java X
1  import javax.swing.*;
2
3  public class SwingLabDemo extends JFrame{
4
5      JFrame f;
6
7      public SwingLabDemo() {
8
9
10     }
11
12     public static void main(String[] args)
13
14
15
16     }
17
18 }
19
```

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

```
SwingLabDemo.java X
1 import javax.swing.*;
2
3 public class SwingLabDemo extends JFrame{
4
5     JFrame f;
6
7     public SwingLabDemo() {
8         f=new JFrame(); // do not re-declare here
9
10        JLabel lbl = new JLabel("Name: ");
11        lbl.setBounds(100,100,40, 40);
12        f.add(lbl);
13        JTextField txtFld = new JTextField();
14        txtFld.setBounds(150, 100, 200, 30);
15        f.add(txtFld);
16        JButton btn = new JButton("Click Me!");
17        btn.setBounds(250,250,90, 40);
18        f.add(btn);
19
20        f.setLayout(null);
21
22        f.setSize(500,500);
23
24        f.setVisible(true);
25    }
26 }
```

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

```
public static void main(String[] args) {
    SwingLabDemo demo = new SwingLabDemo();
}
```

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();
}

```

```

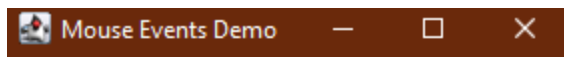
    public void paint(Graphics g) {
        g.drawString(msg, mouseX, mouseY);
    }

    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:



Mouse Exited

ACTION LISTENER

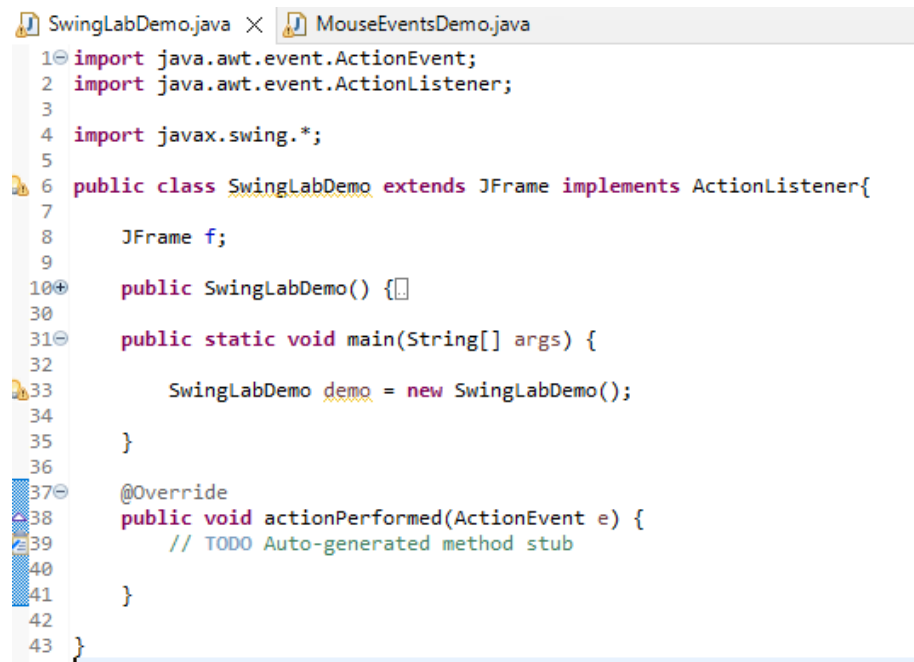
ActionListener in Java is a class that is responsible for handling all action events such as when the user clicks on a component. Mostly, action listeners are used for JButtons. An ActionListener can be used by the implements keyword to the class definition.

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 an error, click it and select “add unimplemented methods”. It will add a new method `public void actionPerformed(ActionEvent e)` in your code.



```
1 import java.awt.event.ActionEvent;
2 import java.awt.event.ActionListener;
3
4 import javax.swing.*;
5
6 public class SwingLabDemo extends JFrame implements ActionListener{
7
8     JFrame f;
9
10    public SwingLabDemo() {}
11
12    public static void main(String[] args) {
13
14        SwingLabDemo demo = new SwingLabDemo();
15
16    }
17
18    @Override
19    public void actionPerformed(ActionEvent e) {
20        // TODO Auto-generated method stub
21
22    }
23 }
```

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

```
public class SwingLabDemo extends JFrame implements ActionListener{

    JFrame f;
    JLabel lbl;
    JTextField txtFld;
    JButton btn;

    public SwingLabDemo() {
        f=new JFrame();

        lbl = new JLabel("Name: ");
        lbl.setBounds(100,100,40, 40);
        f.add(lbl);
        txtFld = new JTextField();
        txtFld.setBounds(150, 100, 200, 30);
        f.add(txtFld);
        btn = new JButton("Click Me!");
        btn.setBounds(250,250,90, 40);
        f.add(btn);
    }
}
```

Step 4: Define the actionPerformed method.

```
@Override
public void actionPerformed(ActionEvent e) {
    txtFld.setText("Abeeha");
}
}
```

Step 5: Run your program and see what happens!