Swing:

* Swing is a collection of classes and interfaces that offer a rich set of visual components.

* Swing is built on the foundation of the AWT.

* Because of the limitations of AWT, swing was included as part of the JFC (Java Foundation Classes).

Advantages of Swings over AWT:

Swing components are lightweight:

* Swings are written entirely in Java and do not map directly to platform-specific peers.

* Lightweight components do not translate into native peers, the look and feel of each component is determined by swing, not by the underlying Operating System.

Swing supports a Pluggable Look and Feel:

* The look and feel of a component is under the control of Swing.

* It is also consistent across all the platforms.

MVC Connection:

* A visual component has three distinct aspects

-- The way the component looks when rendered on the screen (view)
-- The way that the component reacts to the user (Controller)
-- The state information associated with the component (Model)

* The view determines how the component is displayed on the screen.

* The controller determines how the component reacts to the user. The controller reacts by changing the model to reflect the user's choice.

* The model corresponds to the state information associated with the component.

* since the view and controller are separate from the model, the look and feel can be changed without affecting how the component is used within a program.

Components and Containers:

* Component is an independent visual control.

* A container is a special type of component that holds a group of components.

* A container can also hold other containers.

Components:

* Swing components are derived from the Jcomponent class, which inherits the AWT classes Container and Component.

* They are built on and compatible with an AWT component.

Containers:

* Swing defines two types of containers.

* The first are Top-level containers. They inherit the AWT classes Component and Container. They do not inherit JComponent.

  -- JFrame
  -- JApplet
  -- JWindow
  -- JDialog

* The second type of containers supported by swing are lightweight containers. They inherit from JComponent class.

  -- JPanel
  -- JRootPane

Layout Managers:

* The Layout manager controls the position of components within a container. Many are provided by the AWT (java.awt) and swing adds some of its own.

Flow Layout:	A simple layout that positions components from left-to-right, top-to-bottom.
Border Layout:	Positions components within the center or the borders of the container.
GridLayout:	Lays out components within a grid
GridBagLayout:	Lays out different size components within a flexible grid
BoxLayout:	Lays out components vertically or horizontally within a box

Simple Swing program:

```
import javax.swing.*;

class swingdemo
{
        swingdemo()
        {
                Jframe f=new Jframe("Swing Application");
                f.setSize(300,300); // width and height of window
        f.setDefaultCloseOperation(Jframe.EXIT_ON_CLOSE);

        Jlabel lab=new Jlabel("PESIT");

        f.add(lab);
        f.setVisible(true);

        }
}
```

Simple Swing Applet:

* Swing based applets extends JApplet rather than Applet.

* JApplet derived from Applet. Thus swing applets includes all the functionality found in Applet and adds support for swing.

```
import javax.swing.*;  import java.applet.*;
/* <applet code="swingdemo2" width=400 height=100></applet>*/

public class swingdemo2 extends JApplet
{
        public void init()
        {               design();       }
        public void design()
        {
                JLabel l=new JLabel("PESIT");
                add(l);
        }
}
```

Swing Buttons:

* Jbutton, JToggleButton, JCheckBox and JRadioButton.

* All are subclasses of the AbstractButton class, which extends Jcomponent.

* The text associated with a button can be read or written via the following methods:

    -- String getText()
    -- void setText(String str)

* A button generates an ActionEvent when it is pressed.

JButton:

* JButton provides the functionality of a push button.

* Allows an icon, a string or both to be associated with the push button.

JButton(Icon icon)
JButton(String str)
JButton(String str, Icon icon)

* When the button is pressed, an ActionEvent is generated.

* Set the action command by calling setActionCommand() on the button

* Obtain the action command by calling getActionCommand() on the event object.

String getActionCommand()

* The action command identifies the button when two or more buttons are used in the application.

JToggleButton:

* Toggle buttons are objects of the JToggleButton class.

* JToggleButton implements AbstractButton. Also the superclass for two other swing components: JCheckBox and JRadioButton.

    JToggleButton(String str)

* By default, it is in off position.

* JToggleButton generates an ActionEvent each time it is pressed. Also generates an ItemEvent.

* When a JToggleButton is pressed in, it is selected. When it is popped out, it is deselected.

* The toggle button’s state is determined by calling the isSelected() method.

    boolean isSelected()

CheckBoxes:

* JCheckBox class provides the functionality of a check box.

JCheckBox(String str)

* When the user selects or deselects a check box, an ItemEvent is generated.

* The reference to the JCheckBox can be obtained by calling getItem() on the ItemEvent object.

RadioButtons:

* JRadioButton class provides the functionality of a radio button.

JRadioButton(String str)

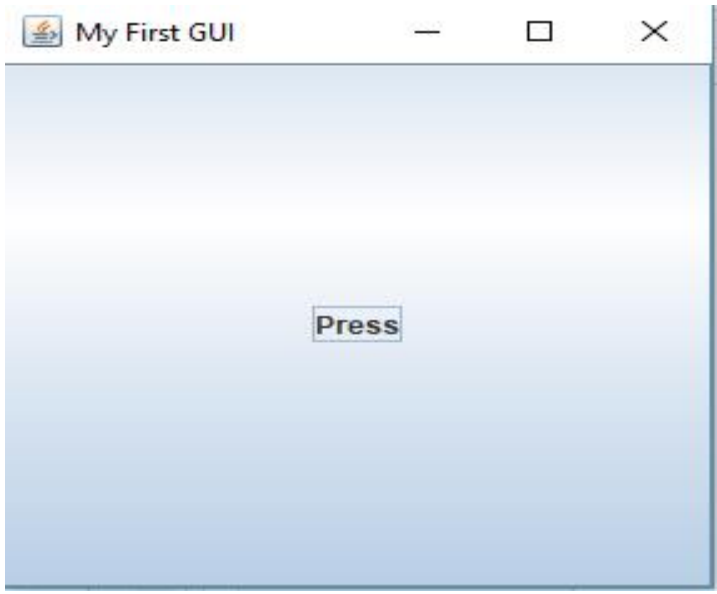
*A button group is created by the ButtonGroup class. Elements are then added to the button group by

void add(AbstractButton ab) // ab is the reference to the button to be
// added

* ActionEvents, ItemEvents and ChangeEvents gets generated.

Example program

```
import javax.swing.*;
  class gui{
     public static void main(String args[]){
      JFrame frame = new JFrame("My First GUI");
      frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
      frame.setSize(300,300);
     JButton button1 = new JButton("Press");
     frame.getContentPane().add(button1);
     frame.setVisible(true);
    }
}
```



Event handling

Event handling in Java is comprised of two key elements:

•**The event source**, which is an object that is created when an event occurs. Java provides several types of these event sources.

•**The event listener**, the object that "listens" for events and processes them when they occur.

There are several types of events and listeners in Java: each type of event is tied to a corresponding listener. Let's consider a common type of event, an *action event* represented by the Java class *ActionEvent*, which is triggered when a user clicks a button or the item of a list.

At the user's action, an *ActionEvent* object corresponding to the relevant action is created. This object contains both the event source information and the specific action taken by the user. This event object is then passed to the corresponding *ActionListener* object's method:

```
void actionPerformed(ActionEvent e)
```

This method is executed and returns the appropriate GUI response, which might be to open or close a dialog, download a file, provide a digital signature, or any other of the myriad actions available to users in an interface.

Types of event

- *ActionEvent*: Represents a graphical element is clicked, such as a button or item in a list. Related listener: *ActionListener.*
- *ContainerEvent*: Represents an event that occurs to the GUI's container itself, for example, if a user adds or removes an object from the interface. Related listener: *ContainerListener.*
- *KeyEvent*: Represents an event in which the user presses, types or releases a key. Related listener: *KeyListener.*
- *WindowEvent: Represents an event relating to a window, for example, when a window is closed, activated or deactivated. Related listener: WindowListener.*
- *MouseEvent: Represents any event related to a mouse, such as when a mouse is clicked or pressed. Related listener: MouseListener.*

```
import java.awt.*;
import java.awt.event.*;
class AEvent extends Frame implements ActionListener{
TextField tf;
AEvent(){
//create components
tf=new TextField();
tf.setBounds(60,50,170,20);
Button b=new Button("click me");
b.setBounds(100,120,80,30);
//register listener
b.addActionListener(this);//passing current instance
//add components and set size, layout and visibility
add(b);add(tf);
setSize(300,300);
setLayout(null);
setVisible(true);  }
public void actionPerformed(ActionEvent e){
tf.setText("Welcome");   }
public static void main(String args[]){
new AEvent();
}  }
```