Chapter-9 Event, Handling Event and Swing

Event

- ❖ An event is an object which specifies the change of state in the source.
- ❖ Event is generated whenever an action takes place like a mouse button is clicked or text is modified.
- ❖ Java's AWT (Abstract Window Toolkit) is responsible for communicating these actions between the program and the user.
- ❖ Java packages such as ***java.util, java.awt, java.awt. event and javax.swing*** support event handling mechanism.
- ❖ The generated event and all the information about it such as time of its occurrence, type of event, etc. are sent to the appropriate event handling code provided within the program. This code determines how the allocated event will be handled so that an appropriate response can be sent to the user.
- ❖ The java.awt.event package provides many event classes and Listener interfaces for event handling.

Types of Events

Here are some of the most common types of events in Java:

- ❖ **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.

Event Handling

❖ Mechanism that controls the event and decides what should happen if an event occurs is called Event Handling.

❖ This mechanism have the code which is known as event handler that is executed when an event occurs.

❖ Java Uses the **Delegation Event Model** to handle the events. Which defines the standard mechanism to generate and handle the events.

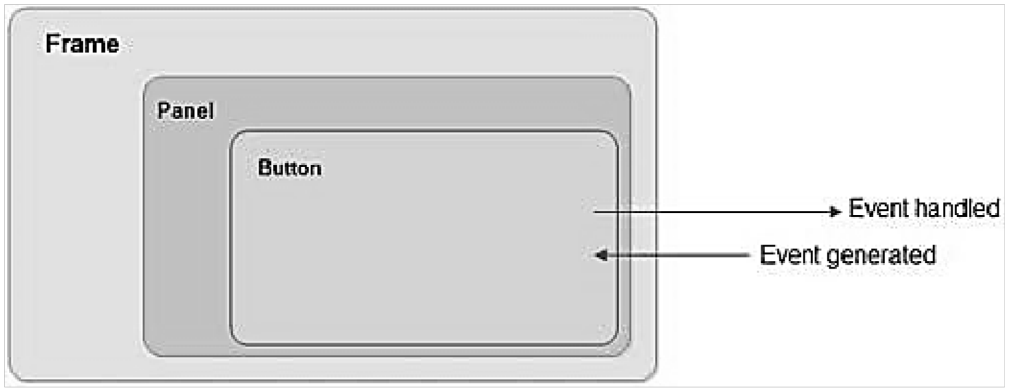
Delegation Event Model

❖ It is java event handling method which defines a logical approach to handle events.
❖ It is based on the concept of source and listener.
❖ A **source** generates an event and sends it to one or more listeners. On receiving the event, **listener** processes the event and returns it.
❖ The notable feature of this model is that the source has a registered list of listeners which will receive the events as they occur.
❖ Only the listeners that have been registered actually receive the notification when a specific event is generated.

For example:

When the mouse is clicked on ***Button***, an event is generated. If the ***Button*** has a registered listener to handle the event, this event is sent to Button, processed and the output is returned to the user. However, if it has no registered listener the event will not be propagated upwards to ***Panel or Frame***.

See following Figure,



The Delegation Event Model has the following key participants

1. **Events**:
   - The event object defines the change in state in the event source class.
   - The event object is used to carry the required information about the state change.
   - However, all events are not cause by user interaction. There are events such as timer event, hardware/software events, and so forth, that do not depend upon user interaction. They occur automatically. We can define the procedure to handle them once they occur.
   - For example, interacting with the graphical interfaces, such as clicking a button or entering text via keyboard in a text box, item selection in a list, all represent some sort of change in the state.
2. **Event source:**
   - An event source is an object that generates a particular kind of event.
   - An event is generated when the internal state of the event source is changed.
   - A source may generate more than one type of event. Every source must register a list of listeners that are interested to receive the notifications regarding the type of event.
   - Event source provides methods to add or remove listeners.

The general form of method to register (add) a listener is:

```
public void addTypeListener(TypeListener eventlistener)
```

Similarly, the general form of method to unregister (remove) a listener is:

```
public void removeTypeListener(TypeListener eventlistener)
```

Where,

Type is the name of the event

eventlistener is a reference to the event listener

3. Event listener:

- ❖ An event listener is an object which receives notification when an event occurs.
- ❖ Only registered listeners can receive notifications from sources about specific types of events. The role of event listener is to receive these notifications and process them.
- ❖ Listener is also known as **event handler.**

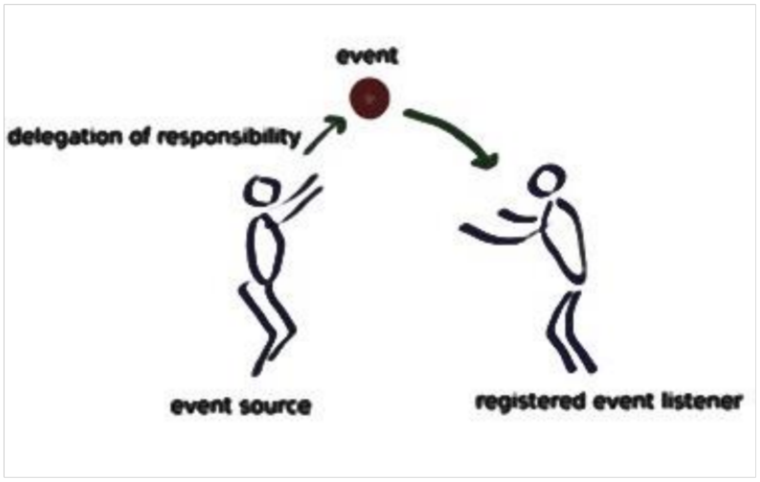


Figure: Events, Source and listeners

<u>The benefit of Event Delegation Method:</u>

- ✓ User interface logic is completely separated from the logic that generates the event.
- ✓ The user interface element is able to delegate the processing of an event to the separate piece of code.
- ✓ In this model, Listener needs to be registered with the source object so that the listener can receive the event notification. This is an efficient way of handling the event because the event notifications are sent only to those listener that want to receive them.

Steps involved in event handling

Following steps are involved.

1. The User clicks the button and the event is generated.
2. Now the object of concerned event class is created automatically and information about the source and the event get populated with in same object.
3. Event object is forwarded to the method of registered listener class.
4. The method is now get executed and returns.

Remember:

- ✓ In order to design a listener class listener interfaces are defined and they forecast some public abstract callback methods which must be implemented by the listener class.
- ✓ If you do not implement the any if the predefined interfaces then your class can not act as a listener class for a source object.

Event Classes and Interface

Event Classes	**Description**	**Listener Interface**
ActionEvent	generated when button is pressed, menu-item is selected, list-item is double clicked	ActionListener
MouseEvent	generated when mouse is dragged, moved,clicked,pressed or released and also when it enters or exit a component	MouseListener
KeyEvent	generated when input is received from keyboard	KeyListener

ItemEvent	generated when check-box or list item is clicked	ItemListener
TextEvent	generated when value of textarea or textfield is changed	TextListener
MouseWheelEvent	generated when mouse wheel is moved	MouseWheelListener
WindowEvent	generated when window is activated, deactivated, deiconified, iconified, opened or closed	WindowListener
ComponentEvent	generated when component is hidden, moved, resized or set visible	ComponentEventListener
ContainerEvent	generated when component is added or removed from container	ContainerListener
AdjustmentEvent	generated when scroll bar is manipulated	AdjustmentListener
FocusEvent	generated when component gains or loses keyboard focus	FocusListener

How Events are handled?

- A source generates an Event and send it to one or more listeners registered with the source.
- Once event is received by the listener, they process the event and then return.
- Events are supported by a number of Java packages, like **java.util**, **java.awt** and **java.awt.event**.

Implementation Steps to handle events

1. Implement the appropriate Listener interface in the class and override its methods
2. Register the component with the listener.

Note: For adding various components public methods are used**,**

<u>Example:</u>

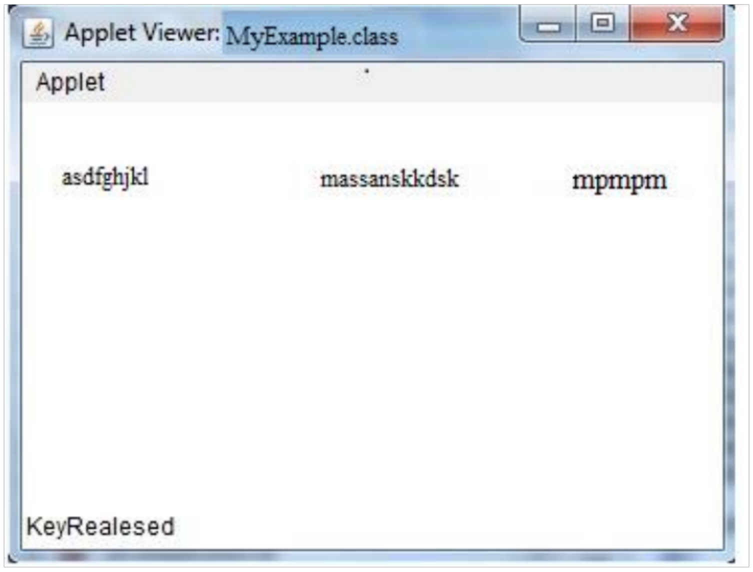
- **Button**
  void addActionListener(ActionListener a)
- **List**
  void addActionListener(ActionListener a)
  void addItemListener(ItemListener a)
- **Choice**
  void addItemListener(ItemListener x)
- **MenuItem**
  void addActionListener(ActionListener x)
- **TextField**
  void addActiontListener(ActionListener x)
  void addTextListener(TextListener x)
- **TextArea**
  void addTextListener(TextListener x)
- **Checkbox**
  void addItemListener(ItemListener x)

Example of Event Handling:

```
import java.awt.*;
import java.awt.event.*;
import java.applet.*;
import java.applet.*;
import java.awt.event.*;

public class EventHandlingEg extends Applet implements KeyListener
{
  String msg="";
  public void init()
  {
    addKeyListener(this);
  }
  public void keyPressed(KeyEvent k)
  {
    showStatus("KeyPressed");
  }
  public void keyReleased(KeyEvent k)
  {
    showStatus("KeyRealesed");
  }
  public void keyTyped(KeyEvent k)
  {
    msg = msg+k.getKeyChar();

      repaint();
  }
  public void paint(Graphics g)
  {
    g.drawString(msg, 30, 50);
  }
}
```

HTML code:

```
<applet code="EventHandlingEg" width=500, height=200>
</applet>
```



```
asdfghjkl          massanskkdsk        mpmpm
```

```
KeyRealesed
```

NOTE: Event Handling provides four types of classes:

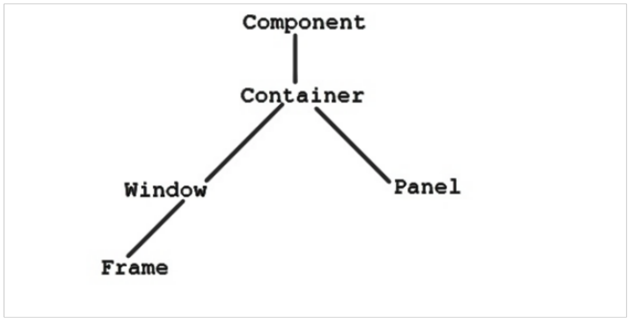
1. Event Adapters
2. Event classes
3. Event Sources
4. Event Listener

Separating GUI and Application Code:

- ❖ The various kinds of AWT/Swing listener classes *do* separate GUI from logic, into altogether separate classes.
- ❖ You do not, however, need to implement listeners as anonymous inner classes, perhaps it would feel better to you to implement them as ordinary, top-level classes instead.
- ❖ Alternatively, you might find that it suits you to model program behaviors via classes implementing the Actioninterface. Then you can assign those behaviors to controls (buttons, menu items) via the controls' setAction() method.
- ❖ however, the code that sets up your GUI has to somehow know about both the GUI components and the logic that needs to be hooked up to them, else there's no way it could do its job

Java Abstract Window Toolkit (AWT)

- ❖ AWT contains large number of classes and methods that allows you to create and manage graphical user interface (GUI) applications, such as windows, buttons, scroll bars, etc.
- ❖ The AWT was designed to provide a common set of tools for GUI design that could work on a variety of platforms.
- ❖ The tools provided by the AWT are implemented using each platform's native GUI toolkit, hence preserving the look and feel of each platform and which is considered as an advantage of using AWT.
- ❖ But the disadvantage of such an approach is that GUI designed on one platform may look different when displayed on another platform.
- ❖ AWT is the foundation upon which Swing is made (i.e. Swing is a set of GUI interfaces that extends the AWT). But now a days AWT is merely used because most GUI Java programs are implemented using Swing because of its rich implementation of GUI controls and light-weighted nature.
- ❖ Java AWT Hierarchy



<u>Component class</u>

- ✓ At the top of AWT hierarchy.
- ✓ An abstract class that encapsulates all the attributes of visual component.
- ✓ Component object is responsible for remembering the current foreground and background colors and the currently selected text font.

<u>Container</u>

- ✓ It is a component in AWT that contains another component like button, text field, tables etc.

- ✓ It is a subclass of component class.
- ✓ Container class keeps track of components that are added to another component.

Panel

- ✓ Panel class is a concrete *subclass* of Container.
- ✓ Panel does not contain title bar, menu bar or border.
- ✓ It is container that is used for holding components.

Window class

- ✓ Window class creates a top level window.
- ✓ Window does not have borders and menu bar.

Frame

- ✓ Frame is a *subclass* of Window and have resizing canvas.
- ✓ It is a container that contain several different components like button, title bar, textfield, label etc.
- ✓ In Java, most of the AWT applications are created using Frame window. Frame class has two different constructors

```
Frame () throws HeadlessException
Frame (String title) throws HeadlessException
```

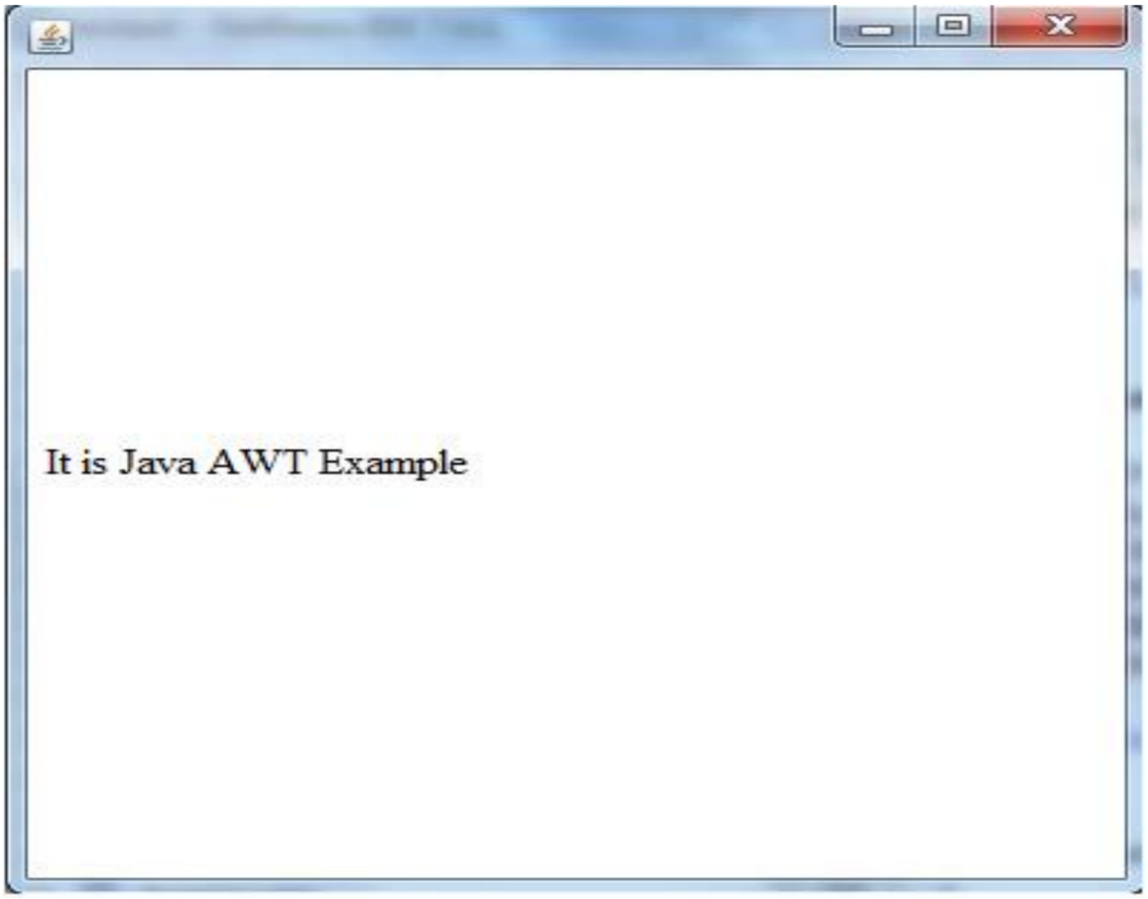
How to Create a Frame?

There are two ways to create a Frame.

1. By Instantiating Frame class
2. By extending Frame class

Creating Frame window by Instantiating Frame Class

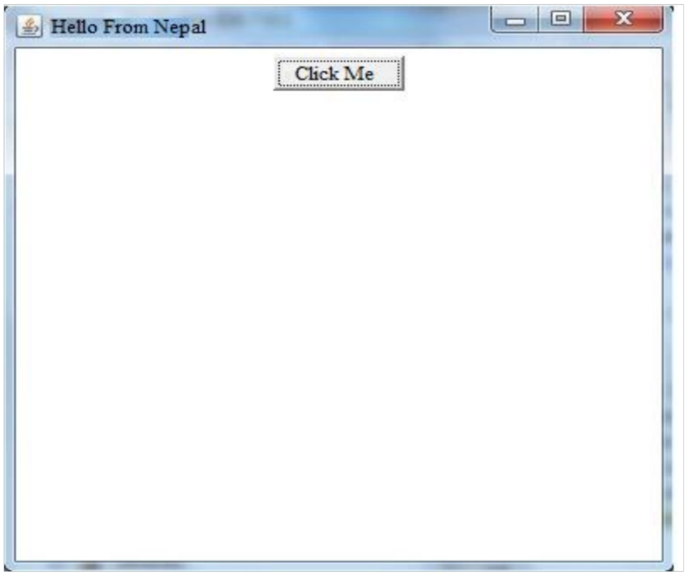
```java
import java.awt.*;
public class ExampleAwt
{
  ExampleAwt()
  {
    Frame frm=new Frame();     //Creating a frame
    Label lbl = new Label("It is java AWT Example");    //Creating a label
    frm.add(lbl);            //adding label to the frame
    frm.setSize(200, 200);   //setting frame size.
    frm.setVisible(true);     //set frame visibilty true
  }
  public static void main(String args[])
  {
    ExampleAwt Egawt = new ExampleAwt();
  }
}
```


It is Java AWT Example

Creating Frame window by extending Frame class

```
package testawt;

import java.awt.*;
import java.awt.event.*;

public class ExampleAwt extends Frame
{
    public ExampleAwt()
    {
        Button btn=new Button("Click Me");
        add(btn);        //adding a new Button.
        setSize(300, 400);         //setting size.
        setTitle("Hello From Nepal");  //setting title.
        setLayout(new FlowLayout());     //set default layout for frame.
        setVisible(true);           //set frame visibilty true.
    }

    public static void main (String[] args)
    {
        ExampleAwt Egawt = new ExampleAwt();    //creating a frame.
    }
}
```

Output:



Code Explanation:

❖ While creating a frame (either by instantiating or extending Frame class), Following two attributes are must for visibility of the frame:

```
setSize(int width, int height);
setVisible(true);
```

❖ When you create other components like Buttons, TextFields, etc. Then you need to add it to the frame by using the method:

```
add(Component's Object);
```

❖ You can add the following method also for resizing the frame

```
setResizable(true);
```

Java Swing

❖ Java Swing is a lightweight Graphical User Interface (GUI) toolkit that includes a rich set of widgets.
❖ Swing library is an official Java GUI tool kit released by Sun Microsystems
❖ It includes package that lets to make GUI components for Java applications, and it is platform independent.
❖ The Swing library is built on top of the Java Abstract Widget Toolkit (AWT)**,** an older, platform dependent GUI toolkit.
❖ Swing provides the Java GUI components like button, textbox, etc. from the library and do not have to create the components from scratch.
❖ Swing Framework contains a set of classes that provides more powerful and flexible GUI components than those of AWT
❖ Swing classes are defined in ***javax.swing*** package and its sub-packages

Main Features of Swing Toolkit

✓ Platform Independent
✓ Customizable
✓ Extensible

- ✓ Configurable
- ✓ Lightweight
- ✓ Rich Controls
- ✓ Pluggable Look and Feel

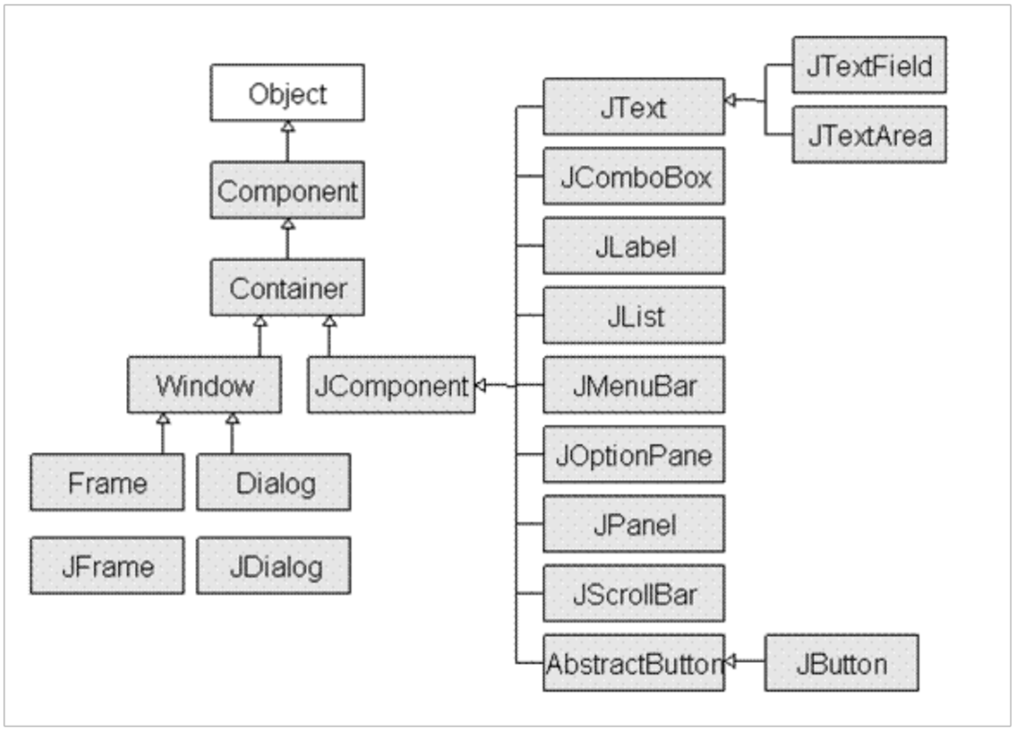
<u>Swing and JFC</u>

- ❖ JFC is an abbreviation for Java Foundation classes, which encompass a group of features for building Graphical User Interfaces (GUI) and adding rich graphical functionalities and interactivity to Java applications.
- ❖ Java Swing is a part of Java Foundation Classes (JFC).

Features of JFC

- Swing GUI components.
- Look and Feel support.
- Java 2D.

<u>AWT and Swing Hierarchy</u>



Note: All components in swing are JComponent which can be added to container classes.

Swing Classes

1. **JPanel:** JPanel is Swing's version of AWT class Panel and uses the same default layout, FlowLayout. JPanel is descended directly from JComponent.
2. **JFrame:** JFrame is Swing's version of Frame and is descended directly from **Frame**class. The component which is added to the **Frame**, is refered as its Content.
3. **JWindow:** This is Swing's version of Window and has descended directly from **Window**class. Like **Window** it uses BorderLayout by default.
4. **JLabel:** JLabel has descended from JComponent, and is used to create text labels.
5. **JButton:** JButton class provides the functioning of push button. JButton allows an icon, string or both associated with a button.
6. **JTextField:** JTextFields allow editing of a single line of text.

Container class

- Container classes are classes that can have other components on it.
- So for creating a GUI, we need at least one container object.
- There are 3 types of containers.

1. **Panel**: It is a pure container and is not a window in itself. The sole purpose of a Panel is to organize the components on to a window.
2. **Frame**: It is a fully functioning window with its title and icons.
3. **Dialog**: It can be thought of like a pop-up window that pops out when a message has to be displayed. It is not a fully functioning window like the Frame.

Creating a JFrame

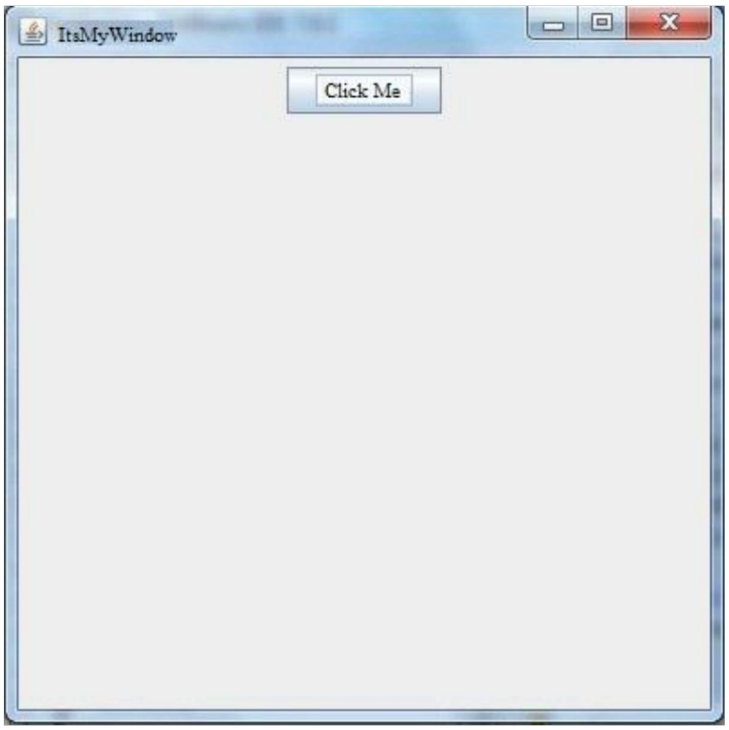
There are two ways to create a JFrame Window.

1. By instantiating JFrame class.
2. By extending JFrame class.

Creating JFrame window by Instantiating JFrame class

```
import javax.swing.*;  //importing swing package
import java.awt.*;     //importing awt package
public class ExampleJFrameSwing
{
    JFrame jf;
    public ExampleJFrameSwing() {
        jf = new JFrame("ItsMyWindow");        //Creating a JFrame with name ItsMyWindow
        JButton btn = new JButton("Click Me");//Creating a Button named Click Me
        jf.add(btn);                       //adding button to frame
        jf.setLayout(new FlowLayout());        //setting layout using FlowLayout object
        jf.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);  //setting close  operation.
        jf.setSize(300, 300);              //setting size
        jf.setVisible(true);               //setting frame visibility
    }
    public static void main(String[] args)
    {
        new ExampleJframeSwing();
    }
}
```

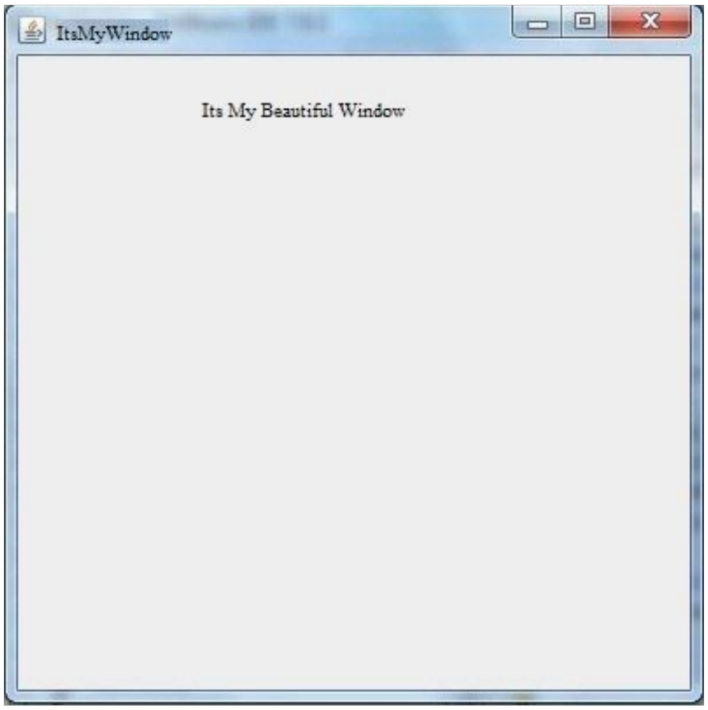
Output:



Creating JFrame window by extending JFrame class

```
import javax.swing.*; //importing swing package
import java.awt.*; //importing awt package
public class ExampleJframeSwing extends JFrame
{
    public ExampleJframeSwing()
    {
        setTitle("ItsMyWindow"); //setting title of frame as  ItsMyWindow
        JLabel lb = new JLabel("Its My Beautiful Window");//Creating a label name
d Its My Beautiful Window
        add(lb);            //adding label to frame.
        setLayout(new FlowLayout());    //setting layout using FlowLayout object.
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE); //setting close operation
.
        setSize(400, 400);      //setting size
        setVisible(true);       //setting frame visibility
    }

    public static void main(String[] args)
    {
        new ExampleJframeSwing();
    }
}
```

Output:



Its My Beautiful Window

Code Explanation

- ❖ Import the javax.swing and java.awt package to use the classes and methods of Swing.
- ❖ While creating a frame (either by instantiating or extending Frame class), following two attributes are must for visibility of the frame:

```
setSize (int width, int height);
setVisible (true);
```

- ❖ When you create objects of other components like Buttons, TextFields, etc. Then you need to add it to the frame by using the method

```
Add (Component's Object);
```

- ❖ You can add the following method also for resizing the frame -

```
setResizable (true);
```

Java Swing Components and Containers

- ❖ A component is an independent visual control.
- ❖ Swing Framework contains a large set of components which provide rich functionalities and allow high level of customization.
- ❖ They all are derived from JComponent class.
- ❖ All these components are lightweight components.
- ❖ This class provides some common functionality like pluggable look and feel, support for accessibility, drag and drop, layout, etc.

A container holds a group of components. It provides a space where a component can be managed and displayed. Containers are of two types:

1. **Top level Containers**
   - ✓ It inherits Component and Container of AWT.
   - ✓ It cannot be contained within other containers.

- ✓ Heavyweight.
- ✓ Example: JFrame, JDialog, JApplet

2. **Lightweight Containers**
   - ✓ It inherits JComponent class.
   - ✓ It is a general purpose container.
   - ✓ It can be used to organize related components together.
   - ✓ Example: JPanel

JButton

- ❖ JButton class provides functionality of a button.
- ❖ JButton class has three constuctors,

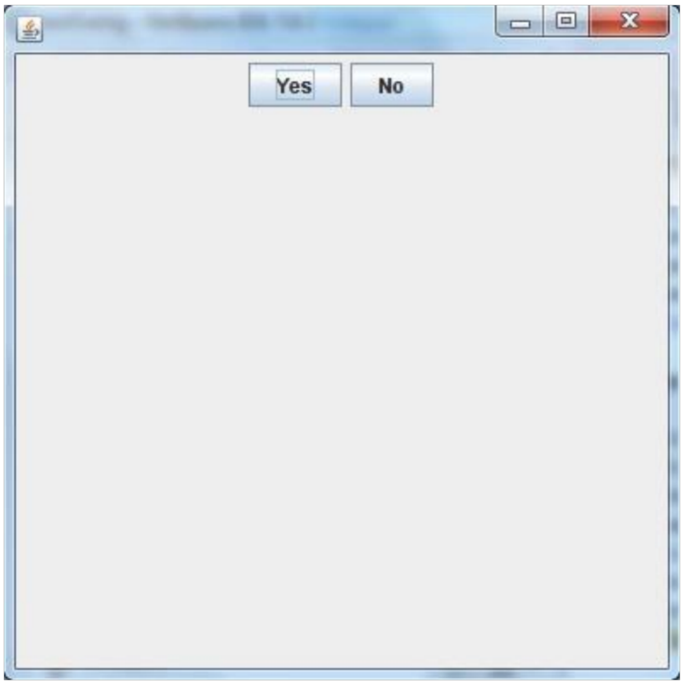
```
JButton(Icon ic)

JButton(String str)

JButton(String str, Icon ic)
```

- ❖ It allows a button to be created using icon, a string or both.
- ❖ JButton supports ActionEvent.
- ❖ When a button is pressed an ActionEvent is generated.

Example using JButton

```
import javax.swing.*;
import java.awt.event.*;
import java.awt.*;
public class MyExample extends JFrame
{

 MyExample()
 {
  JButton btn1 = new JButton("Yes");         //Creating a Yes Button.
  JButton btn2 = new JButton("No");      //Creating a No Button.
  setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE)      //setting close operation.
  setLayout(new FlowLayout());      //setting layout using FlowLayout object
  setSize(300, 300);           //setting size of Jframe
  add(btn1);         //adding Yes button to frame.
  add(btn2);         //adding No button to frame.

  setVisible(true);
 }
 public static void main(String[] args)
 {
  new MyExample();
 }
}
```

Output:



JTextField

- JTextField is used for taking input of single line of text.
- It is most widely used text component.
- It has three constructors

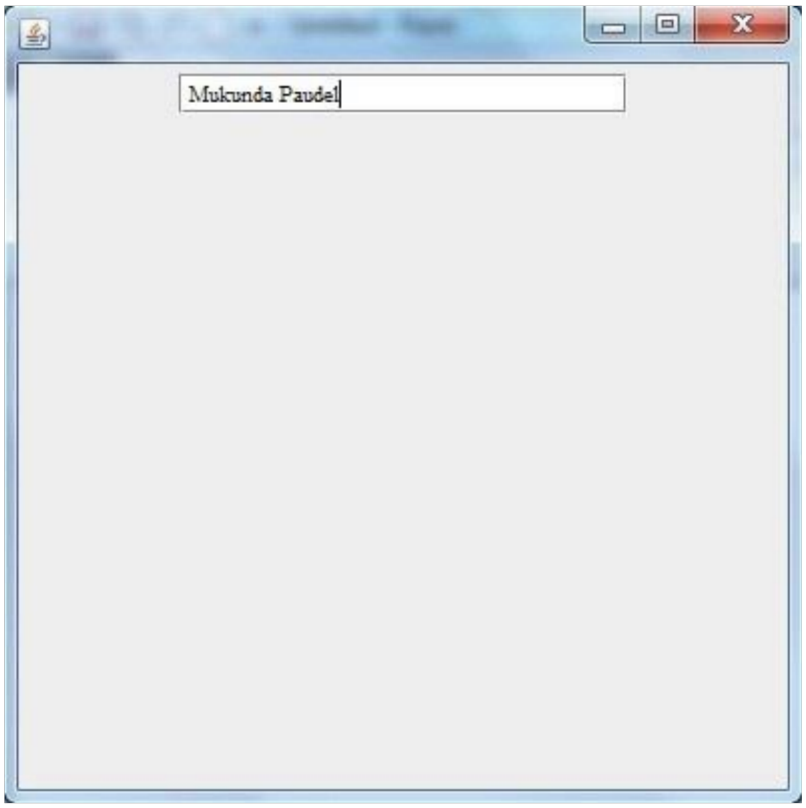
```
JTextField(int cols)
JTextField(String str, int cols)
JTextField(String str)
```

- **cols** represent the number of columns in text field.

Example using JTextField

```
import javax.swing.*;
import java.awt.event.*;
import java.awt.*;
public class MyTextField extends JFrame
{
 public MyTextField()
 {
  JTextField jtf = new JTextField(20);  //creating JTextField.
  add(jtf);              //adding JTextField to frame.
  setLayout(new FlowLayout());
  setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
  setSize(300, 300);
  setVisible(true);
 }
 public static void main(String[] args)
 {
  new MyTextField();
 }
}
```

Output:



```
Mukunda Paudel
```

JCheckBox

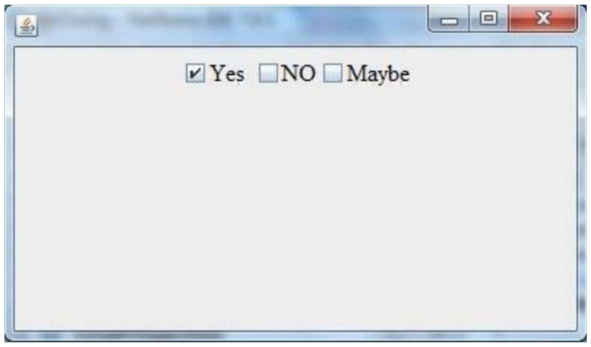
- JCheckBox class is used to create checkboxes in frame.
- Constructor for JCheckBox

```
JCheckBox(String str)
```

Example using JCheckBox

```
import javax.swing.*;
import java.awt.event.*;
import java.awt.*;
public class ExampleJckeckBox extends JFrame
{
 public ExampleJckeckBox()
 {
  JCheckBox jcb = new JCheckBox("Yes");   //creating JCheckBox.
  add(jcb);              //adding JCheckBox to frame.
  jcb = new JCheckBox("No");         //creating JCheckBox.
  add(jcb);              //adding JCheckBox to frame.
  jcb = new JCheckBox("Maybe");       //creating JCheckBox.
  add(jcb);              //adding JCheckBox to frame.
  setLayout(new FlowLayout());
  setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
  setSize(400, 400);
  setVisible(true);
 }
 public static void main(String[] args)
 {
  new ExampleJckeckBox();
 }
}
```

Output:



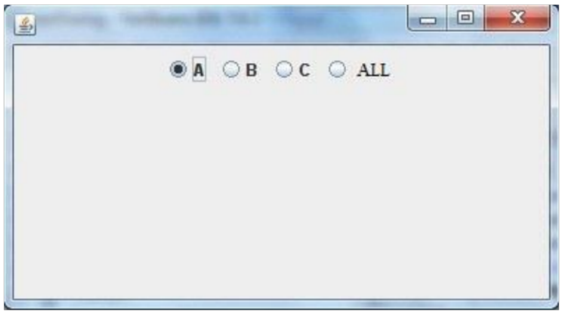
JRadioButton

- Radio button is a group of related button in which only one can be selected.
- JRadioButton class is used to create a radio button in Frames.
- Constructor for JRadioButton

```
JRadioButton(String str)
```

Example using JRadioButton

```
import javax.swing.*;
import java.awt.event.*;
import java.awt.*;
public class ExampleJRadioButton extends JFrame
{
 public ExampleJRadioButton()
 {
  JRadioButton jrb = new JRadioButton("A"); //creating JRadioButton.
  add(jrb);                   //adding JRadioButton to frame.
  jrb = new JRadioButton("B");            //creating JRadioButton.
  add(jrb);                   //adding JRadioButton to frame.
  jrb = new JRadioButton("C");            //creating JRadioButton.
  add(jrb);                   //adding JRadioButton to frame.
  jrb = new JRadioButton("ALL");
  add(jrb);
  setLayout(new FlowLayout());
  setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
  setSize(400, 400);
  setVisible(true);
 }
 public static void main(String[] args)
 {
  new ExampleJRadioButton();
 }
}
```



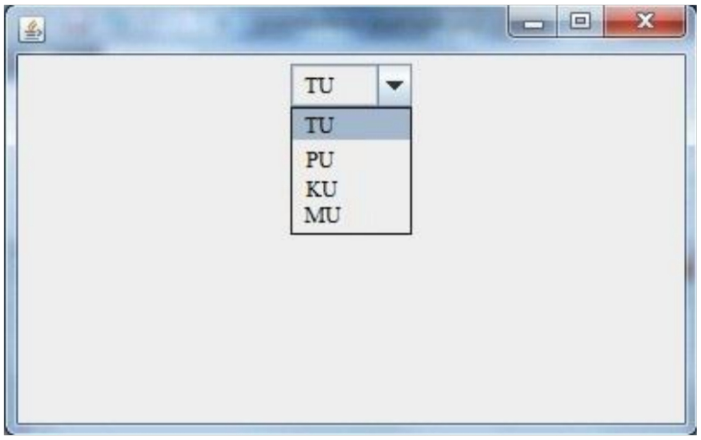
JComboBox

- Combo box is a combination of text fields and drop-down list.
- JComboBox component is used to create a combo box in Swing.
- Constructor for JComboBox,

```
JComboBox(String arr[])
```

Example using JComboBox

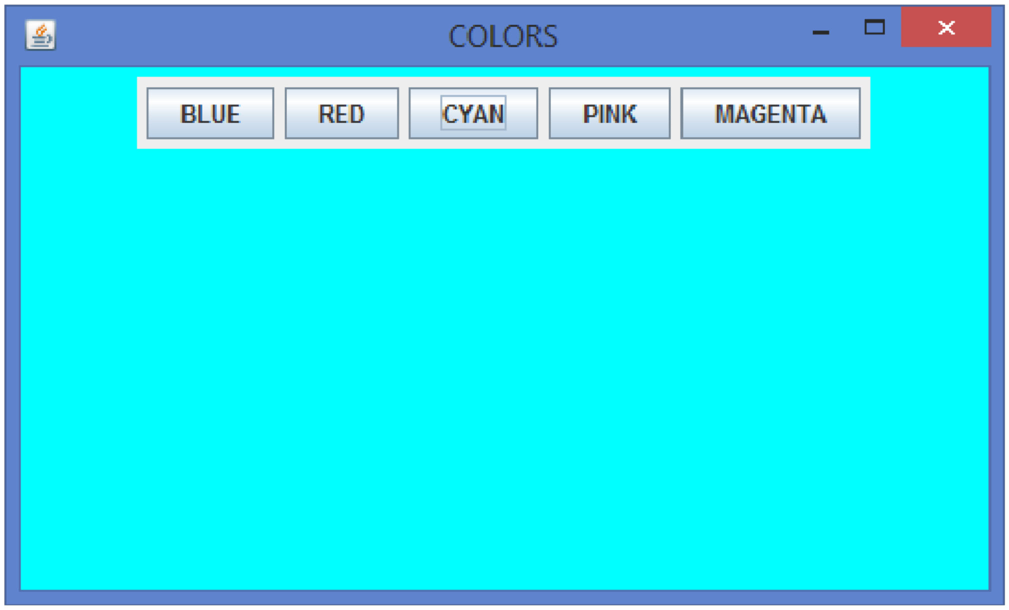
```
import javax.swing.*;
import java.awt.event.*;
import java.awt.*;
public class ExampleJComboBox extends JFrame
{
 String name[] = {"TU","PU","KU","MU"};  //list of name.
 public ExampleJComboBox()
 {
  JComboBox jc = new JComboBox(name);   //initialzing combo box with list of name
.
  add(jc);                //adding JComboBox to frame.
  setLayout(new FlowLayout());
  setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
  setSize(300, 300);
  setVisible(true);
 }
 public static void main(String[] args)
 {
  new ExampleJComboBox();
 }
}
```



Program to change background color of a frame (Using Action Event)

```
import java.awt.*;  //importing awt package
import javax.swing.*;  //importing swing package
import java.awt.event.*;  //importing event package

//For an event to occur upon clicking the button, ActionListener interface should
 be implemented
class BackgroundChangeExample extends JFrame implements ActionListener{

JFrame frame;
JPanel panel;
JButton b1,b2,b3,b4,b5;

BackgroundChangeExample(){

   frame = new JFrame("COLORS");
   frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

   panel = new JPanel();  //Creating a panel which is a container and will hold a
ll the buttons
   panel.setSize(100, 50);

   b1 = new JButton("BLUE");  //Creating a button named BLUE
   b1.addActionListener(this);  //Registering the button with the listener

   b2 = new JButton("RED");  //Creating a button named RED
   b2.addActionListener(this);  //Registering the button with the listener

   b3 = new JButton("CYAN");//Creating a button named CYAN
   b3.addActionListener(this);//Registering the button with the listener

   b4 = new JButton("PINK");  //Creating a button named PINK
   b4.addActionListener(this);  //Registering the button with the listener

   b5 = new JButton("MAGENTA");  //Creating a button named MAGENTA
   b5.addActionListener(this);  //Registering the button with the listener

   //Adding buttons to the Panel
   panel.add(b1);
   panel.add(b2);
   panel.add(b3);
   panel.add(b4);
   panel.add(b5);
   frame.getContentPane().add(panel);  //adding panel to the frame
   frame.setSize(500,300);
   frame.setVisible(true);
   frame.setLayout(new FlowLayout());

}
//The below method is called whenever a button is clicked
    @Override
    public void actionPerformed(ActionEvent e) {

        //This method returns an object of the button on which the Event-
        Pressing of button initially occurred
        Object see = e.getSource();

        if(see ==(b1)){  //Checking if the object returned is of button1
        frame.getContentPane().setBackground(java.awt.Color.blue);  //changing th
e panel color to blue
        }
        if(see == b2){  //Checking if the object returned is of button2
            frame.getContentPane().setBackground(java.awt.Color.red);  //changing
 the panel color to red
        }
        if(see == b3){  //Checking if the object returned is of button3
        frame.getContentPane().setBackground(java.awt.Color.cyan);//changing the
panel color to cyan
        }
        if(see == b4){  //Checking if the object returned is of button4
            frame.getContentPane().setBackground(java.awt.Color.pink);  //changin
g the panel color to pink
        }
        if(see == b5){  //Checking if the object returned is of button5
        frame.getContentPane().setBackground(java.awt.Color.magenta);  //changing
 the panel color to magenta
        }
    }
}

class Check {
    public static void main(String[] args) {
        BackgroundChangeExample chnagebg = new BackgroundChangeExample();
    }
}
```

Ouput:



Difference between AWT and Swing

AWT	SWING
1. AWT stands for Abstract Window Toolkit.	1. Swing is a part of Java Foundation Class (JFC).
2. AWT components are platform-dependent so their look and feel changes according to OS.	2. Java swing components are platform-independent so their looks and feel remains constant.
3. AWT doesn't support pluggable look and feel.	3. Swing supports pluggable look and feel.
4. AWT doesn't follows MVC (Model View Controller) where model represents data, view represents presentation and controller acts as an interface between model and view.	4. Swing follows MVC.

5. AWT components are heavyweight.	5. Swing components are lightweight.
6. AWT provides less components than Swing.	6. Swing provides more powerful components such as tables, lists, scrollpanes, colorchooser, tabbedpane etc.