Unit #2
GUI – Swing & Event Handling

Prof. Ravikumar R Natarajan

Dept. of CE

Marwadi University

Contents

- Swing features
- Swing Containers
- Jframe
- Jpanel
- JWindow
- Swing components : JLabel, ImageIcon, JTextField, JButton, JToggeleButton, JCheckBox, JRadioButton, JTabbedPane, JScrollPane, JList, JComboBox, JTree, JTable
- Layout managers
- Event model in Java
- Event classes
- Event listeners
- Adapter classes

Java Swing

- Java Swing tutorial is a part of Java Foundation Classes (JFC) that is used to create window-based applications. It is built on the top of AWT (Abstract Windowing Toolkit) API and entirely written in java.
- Unlike AWT, Java Swing provides platform-independent and lightweight components.
- The **javax.swing** package provides classes for java swing API such as JButton, JTextField, JTextArea, JRadioButton, JCheckbox, JMenu, JColorChooser etc.

Java Swing

Java AWT	Java Swing
AWT components are **platform-dependent**.	Java swing components are **platform-independent**.
AWT components are **heavyweight**.	Swing components are **lightweight**.
AWT **doesn't support pluggable look and feel**.	Swing **supports pluggable look and feel**.
AWT provides **less components** than Swing.	Swing provides **more powerful components** such as tables, lists, scrollpanes, colorchooser, tabbedpane etc.
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.	Swing **follows MVC**.

Features of Swing Class

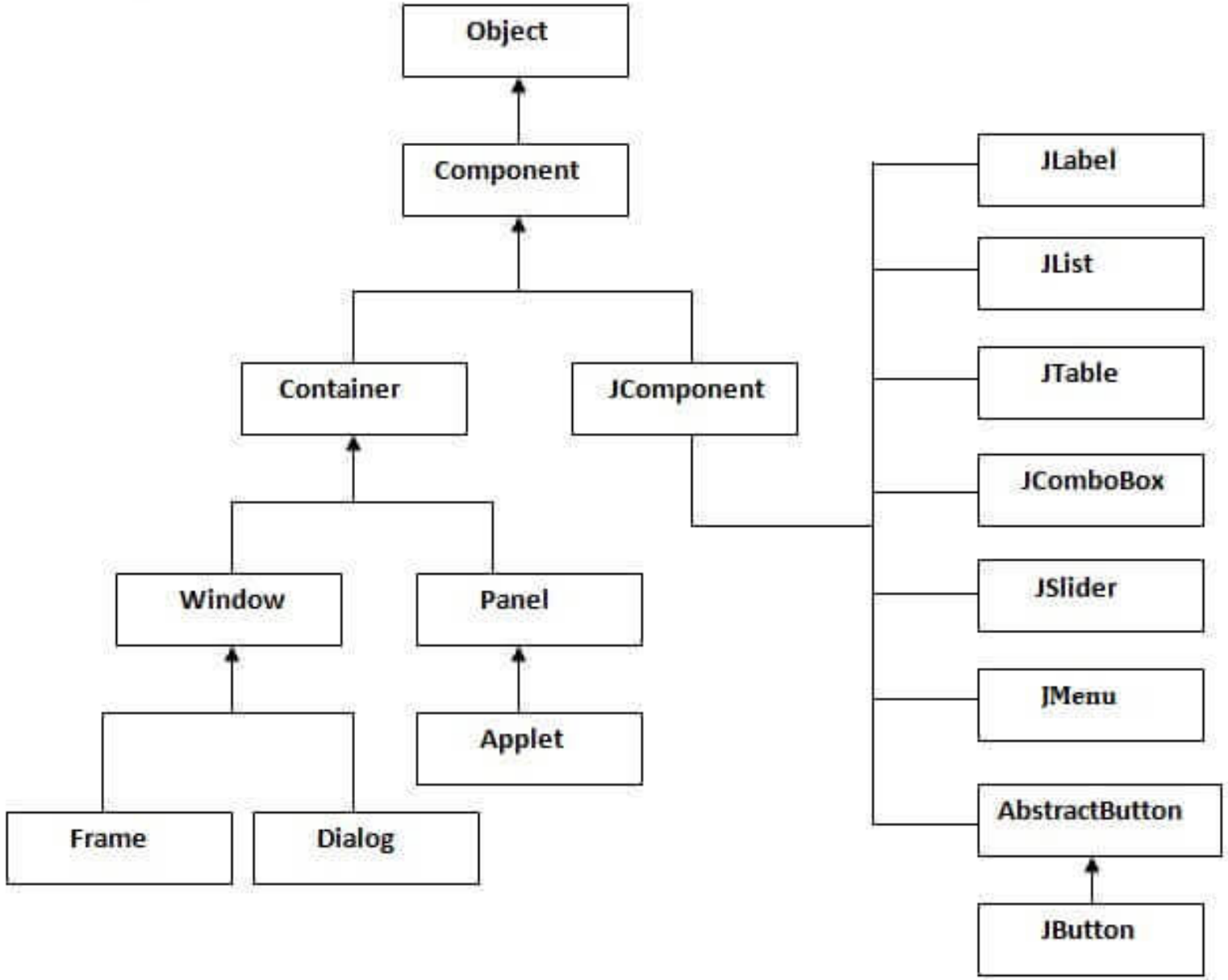
- Pluggable look and feel
- Uses MVC architecture
- Lightweight Components
- Platform Independent
- Advanced features such as JTable, JTabbedPane, JScollPane, etc.
- Lightweight Components
- Pluggable Look and Feel
- Highly customizable
- Rich controls

MVC Architecture

- Swing API architecture follows loosely based MVC architecture in the following manner.
- **Model View Controller**
- Model represents component's data.
- View represents visual representation of the component's data.
- Controller takes the input from the user on the view and reflects the changes in Component's data.

- Swing component has Model as a seperate element, while the View and Controller part are clubbed in the User Interface elements. Because of which, Swing has a pluggable look-and-feel architecture.

Hierarchy of Java Swing Classes



Commonly used Methods of Component Class

The methods of Component class are widely used in java swing that are given below.

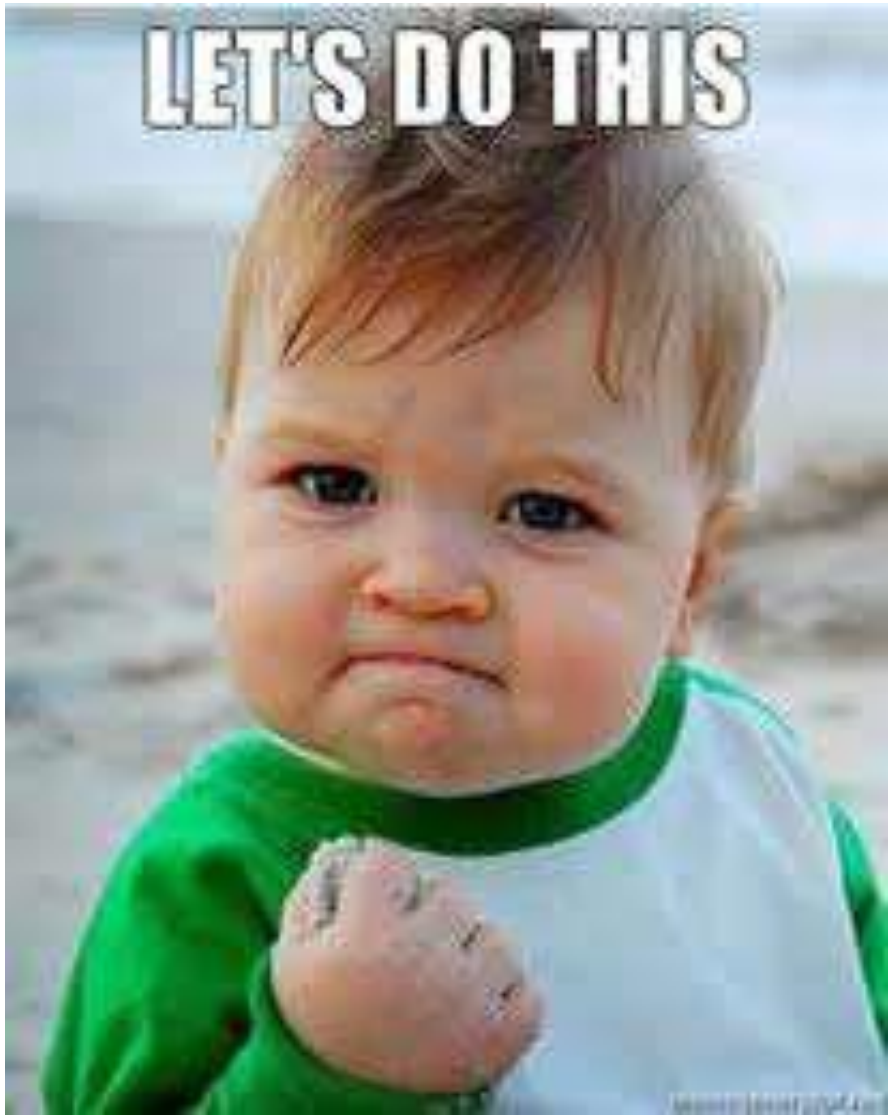
Method	Description
public void add(Component c)	add a component on another component.
public void setSize(int width,int height)	sets size of the component.
public void setLayout(LayoutManager m)	sets the layout manager for the component.
public void setVisible(boolean b)	sets the visibility of the component. It is by default false.

How to Create Java Swing?

- There are two ways to create a frame:

- By creating the object of Frame class (association)
- By extending Frame class (inheritance)

Ready to Create Java Swing Application?



JFrame Object Inside the main() Method

```
import javax.swing.*;
public class FirstSwingExample {
public static void main(String[] args) {
JFrame f=new JFrame(); //creating instance of JFrame
JButton b=new JButton("click"); //creating instance of JButton
b.setBounds(130,100,100, 40); //x axis, y axis, width, height

f.add(b); //adding button in JFrame

f.setSize(400,500); //400 width and 500 height
f.setLayout(null); //using no layout managers
f.setVisible(true); //making the frame visible
} }
```

JFrame Object Inside the Constructor

```
import javax.swing.*;
public class InsideConstructor {
JFrame f;
InsideConstructor(){
f=new JFrame();//creating instance of JFrame
JButton b=new JButton("click");//creating instance of JButton
b.setBounds(130,100,100, 40);
f.add(b);//adding button in JFrame
f.setSize(400,500);//400 width and 500 height
f.setLayout(null);//using no layout managers
f.setVisible(true);//making the frame visible
}
public static void main(String[] args) {
new InsideConstructor();
} }
```

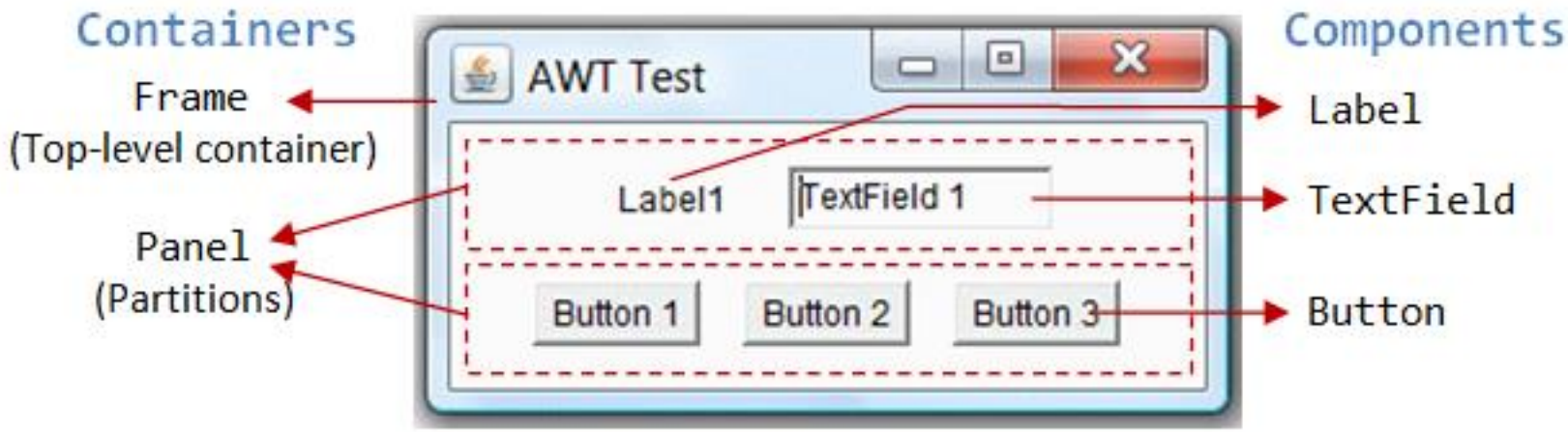
JFrame using Inheritance

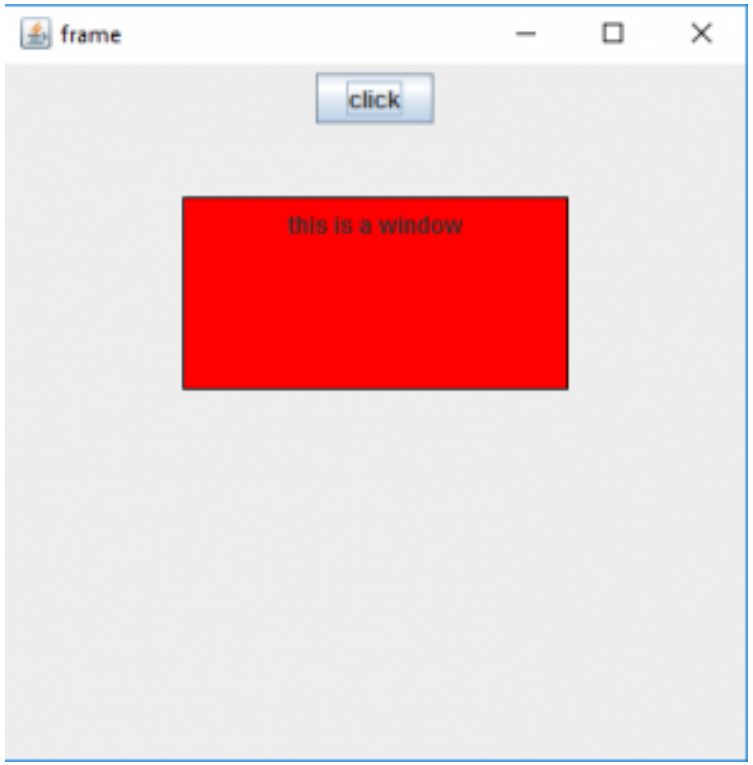
```
import javax.swing.*;
public class UsingInheritance extends JFrame{//inheriting JFrame
//JFrame f;
UsingInheritance(){
JButton b=new JButton("click");//create button
b.setBounds(130,100,100, 40);

add(b);//adding button on frame
setSize(400,500);
setLayout(null);
setVisible(true);
}
public static void main(String[] args) {
new UsingInheritance ();
}}
```

Swing Containers: JFrame, JPanel, JWindow

- JPanel is the simplest container. It provides space in which any other component can be placed, including other panels.
- A JFrame is a top-level window with a title and a border.
- A JWindow object is a top-level window with no borders and no menubar.

Swing Containers: JFrame, JPanel, JWindow

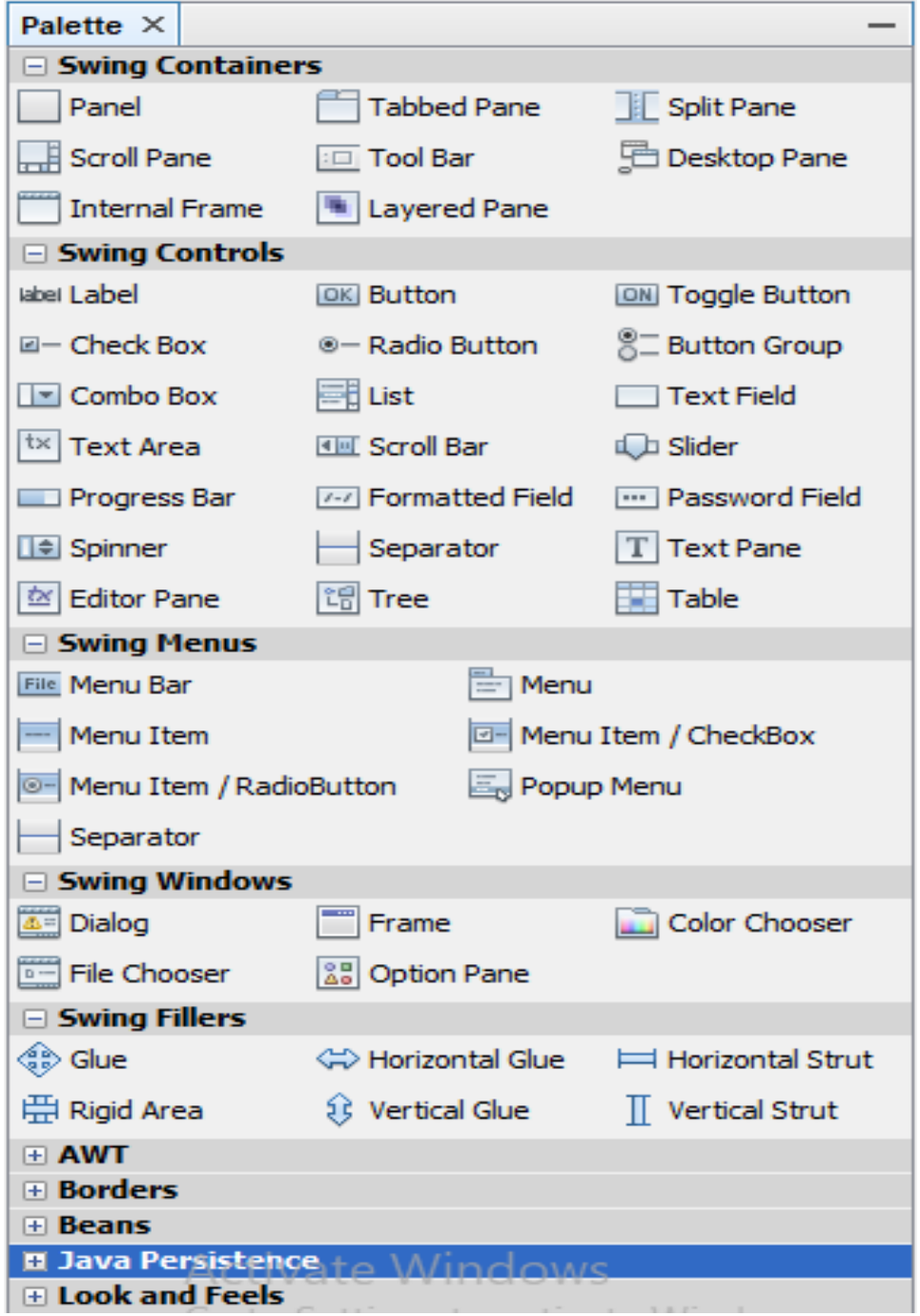




Swing Containers: JFrame, JPanel, JWindow

- DEMO

Swing Components



Swing Components

JLabel	A JLabel is an object component for placing text in a container
JButton	This class creates a labeled button
JColorChooser	A JColorChooser provides a pane of controls designed to allow the user to manipulate and select a color
JCheckBox	A JCheckBox is a graphical(GUI) component that can be in either an on-(true) or off-(false) state
JRadioButton	The JRadioButton class is a graphical(GUI) component that can be in either an on-(true) or off-(false) state. in the group
JList	A JList component represents the user with the scrolling list of text items
JComboBox	A JComboBox component is Presents the User with a show up Menu of choices
JTextField	A JTextField object is a text component that will allow for the editing of a single line of text
JPasswordField	A JPasswordField object it is a text component specialized for password entry

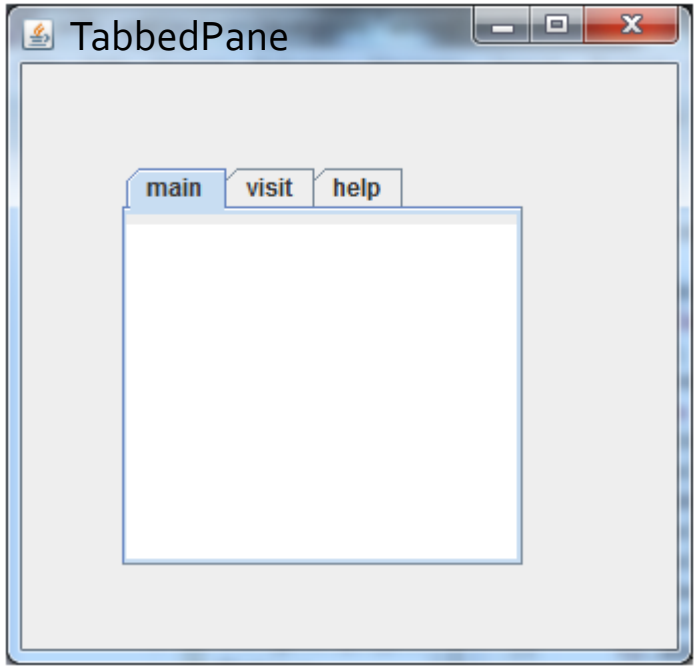
Swing Components

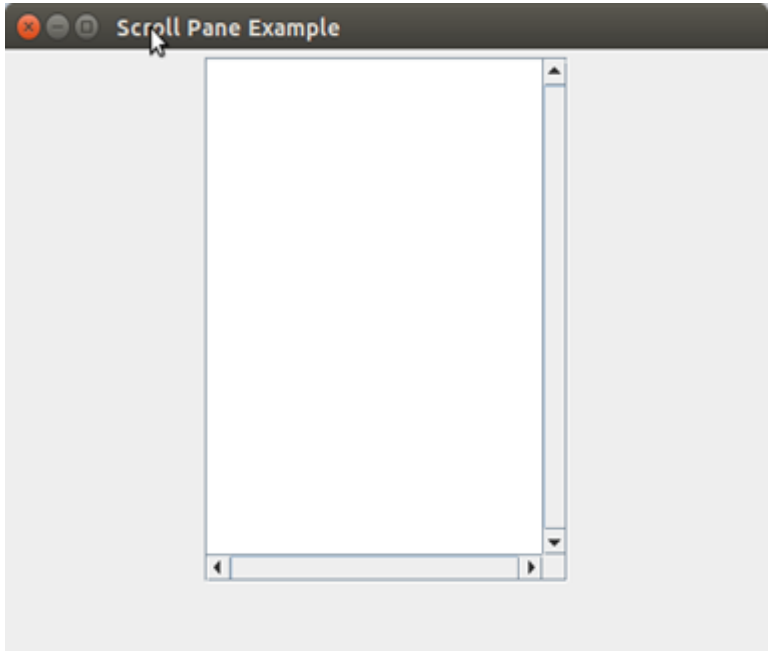
JToggeleButton	A JToggleButton is an extension of AbstractButton and it can be used to represent buttons that can be toggled ON and OFF.
JTabbedPane	The JTabbedPane class is used to switch between a group of components by clicking on a tab with a given title or icon.
JScrollPane	A JscrollPane is used to make scrollable view of a component. When screen size is limited, we use a scroll pane to display a large component or a component whose size can change dynamically.
JTree	The JTree class is used to display the tree structured data or hierarchical data. JTree is a complex component. It has a 'root node' at the top most which is a parent for all nodes in the tree. It inherits JComponent class.
JTable	The JTable class is used to display data in tabular form. It is composed of rows and columns.

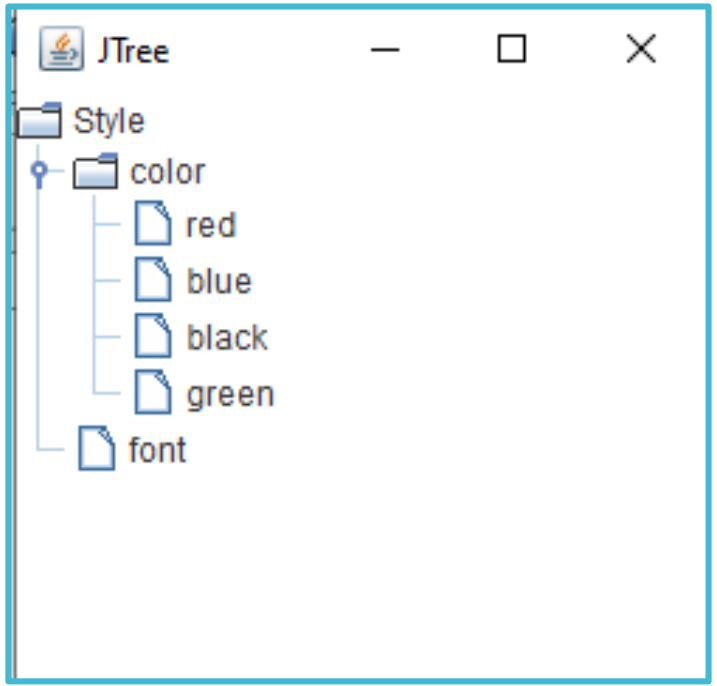
Swing Components

JTextArea	A JTextArea object s a text component that allows for the editing of multiple lines of text
ImageIcon	A ImageIcon control is an implementation of the Icon interface that paints Icons from Images
JScrollbar	A JScrollbar control represents a scroll bar component in order to enable users to Select from range values
JOptionPane	JOptionPane provides set of standard dialog boxes that prompt users for a value or Something
JFileChooser	A JFileChooser it Controls represents a dialog window from which the user can select a file.
JProgressBar	As the task progresses towards completion, the progress bar displays the tasks percentage on its completion
JSlider	A JSlider this class is lets the user graphically(GUI) select by using a value by sliding a knob within a bounded interval.

Swing Components







ImageIcon

```
import javax.swing.*;
public class DispImage{
  DispImage()   {
    JFrame f = new JFrame("Add an image to JFrame");
    ImageIcon icon = new
ImageIcon("C:\\Users\\DELL\\Pictures\\thumb_cries-in-bad-
at-java-72496323.png");
    f.add(new JLabel(icon));
    f.pack();   //f.setSize(400,400);
    f.setVisible(true);
  }
  public static void main(String args[])   {
    new DispImage();    } }
```

Pack() sizes the frame so that all its contents are at or above their preferred sizes. It is a alternate to setSize, setBounds.

Swing Components

- DEMO

Layout Managers

- In Java swing, Layout manager is used to position all its components, with setting properties, such as the size, the shape, and the arrangement. Different layout managers could have varies in different settings on their components.
- The following layout managers are:

1. FlowLayout
2. BorderLayout
3. CardLayout
4. BoxLayout
5. GridLayout
6. GridBagLayout
7. GroupLayout
8. SpringLayout
9. ScrollPaneLayout

Flow Layout

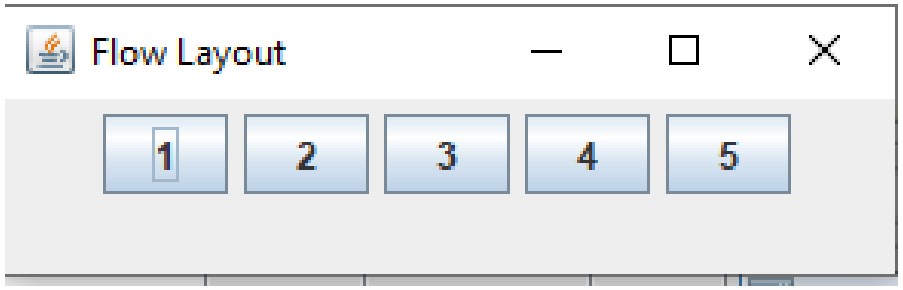
- **The Java FlowLayout class is used to arrange the components in a line, one after another (in a flow). It is the default layout of the applet or panel.**

- **Fields of FlowLayout class:**
- public static final int LEFT
- public static final int RIGHT
- public static final int CENTER
- public static final int LEADING
- public static final int TRAILING

- **Constructors:**
- FlowLayout()
- FlowLayout(int align)
- FlowLayout(int align, int hgap, int vgap)

Flow Layout

```
import java.awt.*;
import javax.swing.*;
public class FlowLayoutExample {
   JFrame f;
   FlowLayoutExample()    {
      f= new JFrame();
      JButton b1 = new JButton("1");       JButton b2 = new JButton("2");
      JButton b3 = new JButton("3");       JButton b4 = new JButton("4");
      JButton b5 = new JButton("5");
      f.add(b1); f.add(b2); f.add(b3); f.add(b4);  f.add(b5);
      f.setLayout(new FlowLayout());
      f.setSize(300, 300);       f.setVisible(true);
   }
   public static void main(String argvs[])
   {       new FlowLayoutExample();     }}
```



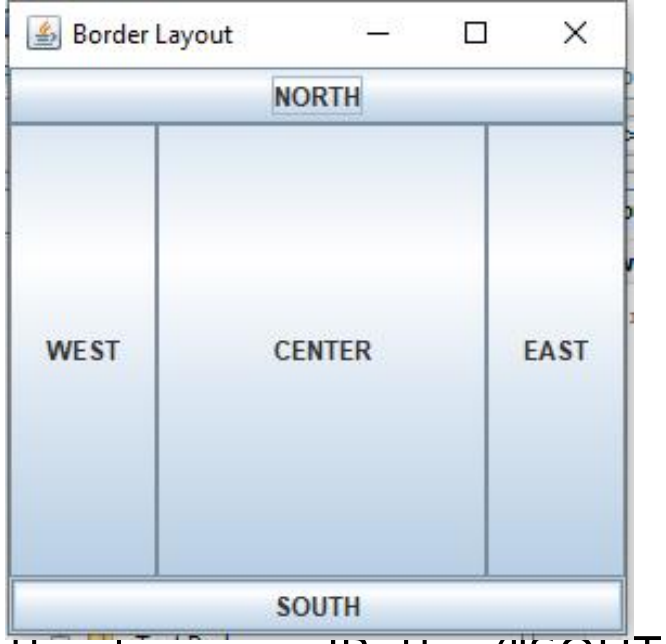
Border Layout

- **The BorderLayout is used to arrange the components in five regions: north, south, east, west, and center. Each region (area) may contain one component only.**

- public static final int NORTH
- public static final int SOUTH
- public static final int EAST
- public static final int WEST
- public static final int CENTER

- **Constructor:**
- BorderLayout()
- BorderLayout(int hgap, int vgap)

Border Layout




```
import java.awt.*;
import javax.swing.*;
public class BorderDemo {
   JFrame f;
   Border()    {
      f = new JFrame();
JButton b1 = new JButton("NORTH");      JButton b2 = new JButton("SOUTH");
JButton b3 = new JButton("EAST");          JButton b4 = new JButton("WEST");
JButton b5 = new JButton("CENTER");
      f.add(b1, BorderLayout.NORTH);      f.add(b2, BorderLayout.SOUTH);
      f.add(b3, BorderLayout.EAST);      f.add(b4, BorderLayout.WEST);
      f.add(b5, BorderLayout.CENTER);
      f.setSize(300, 300);
      f.setVisible(true);    }
   public static void main(String[] args) {
      new BorderDemo();    } }
```

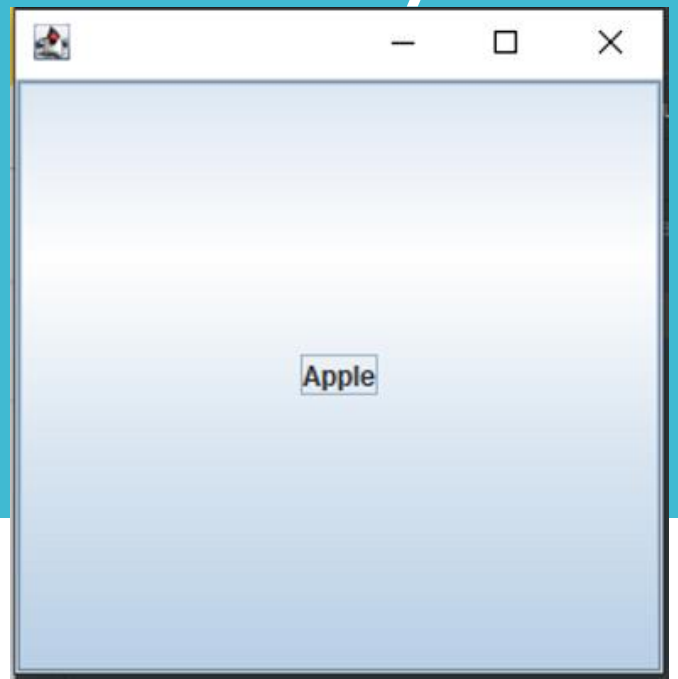
Card Layout

- **The Java CardLayout class manages the components in such a manner that only one component is visible at a time.**
- It treats each component as a card that is why it is known as CardLayout.

- **Methods:**
- next, previous, first, last, show

- **Constructors:**
- CardLayout(),
- CardLayout(int hgap, int vgap)

Card Layout




```
import java.awt.*;
import javax.swing.*;
import java.awt.event.*;
class CardLayoutDemo extends JFrame
implements ActionListener
{
   CardLayout crd;
   JButton btn1, btn2, btn3;
   Container cPane;
   CardLayoutDemo()
   {
      cPane = getContentPane();
      crd = new CardLayout();
      cPane.setLayout(crd);
      btn1 = new JButton("Apple");
      btn2 = new JButton("Boy");
      btn3 = new JButton("Cat");
      btn1.addActionListener(this);
      btn2.addActionListener(this);
      btn3.addActionListener(this);

      cPane.add("a", btn1);
      cPane.add("b", btn2);
      cPane.add("c", btn3);
   }
   public void actionPerformed(ActionEvent e)
   {      crd.next(cPane);    }
   public static void main(String argvs[])
   {CardLayoutDemo crdl = new
CardLayoutDemo();
      crdl.setSize(300, 300);
      crdl.setVisible(true);
}
}
```

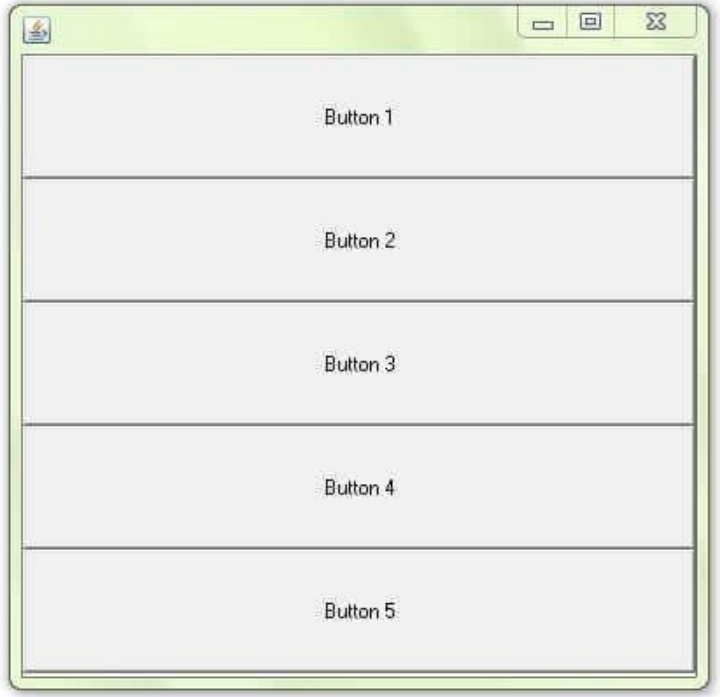
Box Layout

- **The Java BoxLayout class is used to arrange the components either vertically or horizontally.**
- **For this purpose, the BoxLayout class provides four constants. They are as follows:**
- public static final int X_AXIS, Y_AXIS, LINE_AXIS, PAGE_AXIS

- **Constructors:**
- BoxLayout(Container c, int axis)

Box Layout

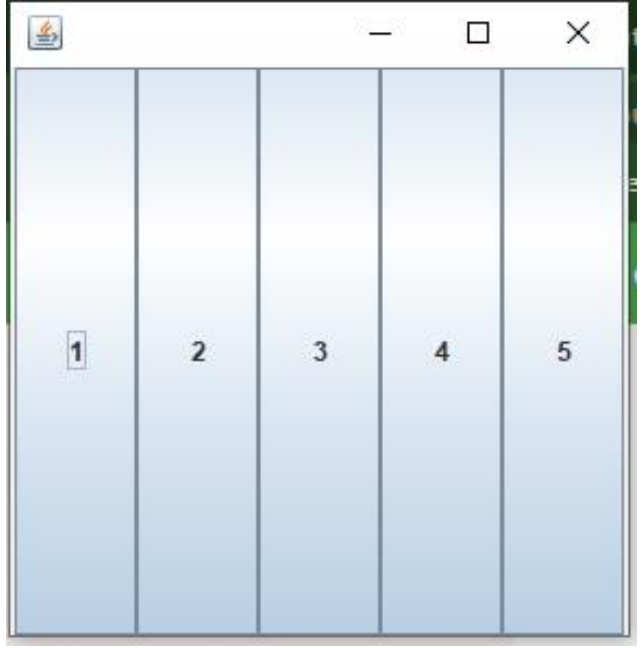
```
import java.awt.*;
import javax.swing.*;
class BoxLayoutDemo extends JFrame {
   JButton buttons[];
   public BoxLayoutDemo () {
      buttons = new JButton [5];
      for (int i = 0;i<5;i++) {
         buttons[i] = new JButton ("Button " + (i + 1));
         add (buttons[i]);       }
      setLayout (new BoxLayout(getContentPane(), BoxLayout.X_AXIS));
      setSize(400,400);
      setVisible(true);    }
   public static void main(String args[]){
      BoxLayoutDemo b=new BoxLayoutDemo();    } }
```



Grid Layout

- **The Java GridLayout class is used to arrange the components in a rectangular grid. One component is displayed in each rectangle.**
- **All components will be in same size.**
- **Constructors of GridLayout class:**
- GridLayout()
- GridLayout(int rows, int columns)
- GridLayout(int rows, int columns, int hgap, int vgap)

Grid Layout

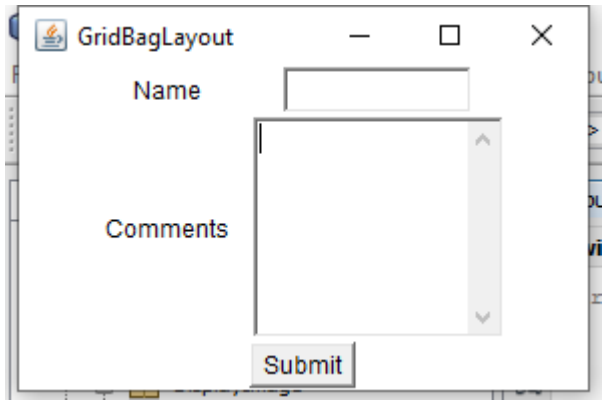



```
import java.awt.*;
import javax.swing.*;
public class GridLayoutDemo {
JFrame f;
GridLayoutDemo()  {
f = new JFrame();
JButton btn1 = new JButton("1");    JButton btn2 = new JButton("2");
JButton btn3 = new JButton("3");    JButton btn4 = new JButton("4");
JButton btn5 = new JButton("5");
f.add(btn1); f.add(btn2); f.add(btn3); f.add(btn4); f.add(btn5);
f.setLayout(new GridLayout());
f.setSize(300, 300);
f.setVisible(true);    }
 public static void main(String argvs[])   {
new GridLayoutDemo();    }    }
```

Grid Bag Layout

- **The Java GridBagLayout class is used to align components vertically, horizontally or along their baseline.**
- **The components may not be of the same size.** Each GridBagLayout object maintains a dynamic, rectangular grid of cells.

- **Constructor:**
- GridBagLayout(): The parameterless constructor is used to create a grid bag layout manager.

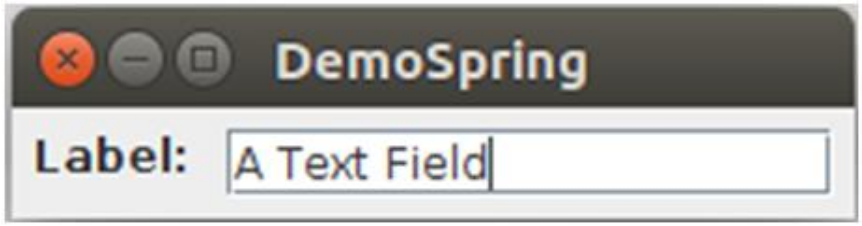


Group Layout

- GroupLayout groups its components and places them in a Container hierarchically. The grouping is done by instances of the Group class.
- Group is an abstract class, and two concrete classes which implement this Group class are SequentialGroup and ParallelGroup.
- *SequentialGroup* positions its child sequentially one after another whereas *ParallelGroup* aligns its child on top of each other.
- The GroupLayout class provides methods such as createParallelGroup() and createSequentialGroup() to create groups.

Spring Layout

- A SpringLayout arranges the children of its associated container according to a set of constraints. Constraints are nothing but horizontal and vertical distance between two-component edges. Every constraint is represented by a SpringLayout.Constraint object.
- Each child of a SpringLayout container, as well as the container itself, has exactly one set of constraints associated with them.
- Each edge position is dependent on the position of the other edge. If a constraint is added to create a new edge, than the previous binding is discarded. SpringLayout doesn't automatically set the location of the components it manages.
- **SpringLayout():** The default constructor of the class is used to instantiate the SpringLayout class.



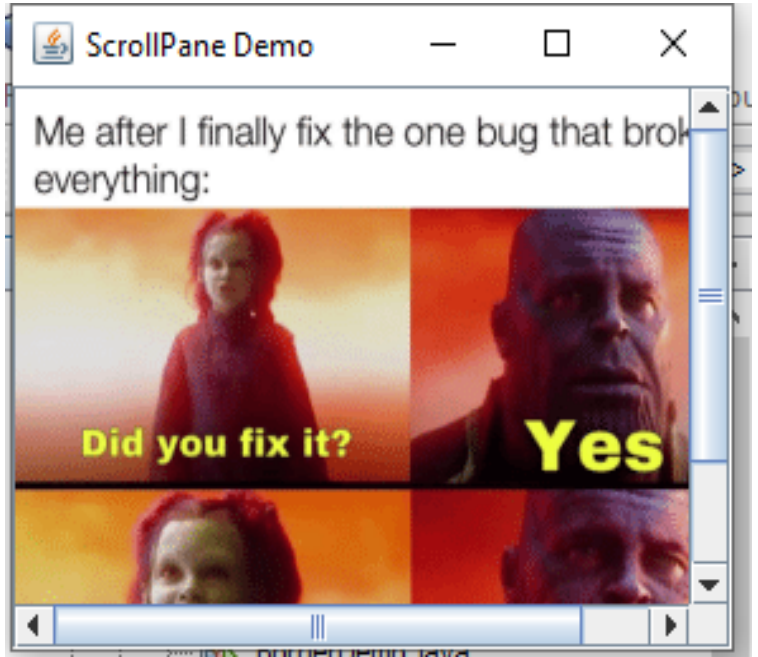
Scroll Pane Layout

- The layout manager is used by JScrollPane.
- JScrollPaneLayout is responsible for nine components:
- a viewport,
- two scrollbars,
- a row header,
- a column header,
- and four "corner" components.

- **Constructor:**
- ScrollPaneLayout(): The parameterless constructor is used to create a new ScrollPanelLayout.

Scroll Pane Layout

```
import javax.swing.*;
public class ScrollPaneLayoutDemo extends JFrame  {
public ScrollPaneLayoutDemo() {
super("ScrollPane Demo");
ImageIcon img = new
ImageIcon("C:\\Users\\DELL\\Desktop\\IEEE_Conference_Paper_Publication.
png");
 JScrollPane png = new JScrollPane(new JLabel(img));
getContentPane().add(png);
setSize(300,250);
setVisible(true);   }
public static void main(String[] args) {
new ScrollPaneLayoutDemo();   }  }
```

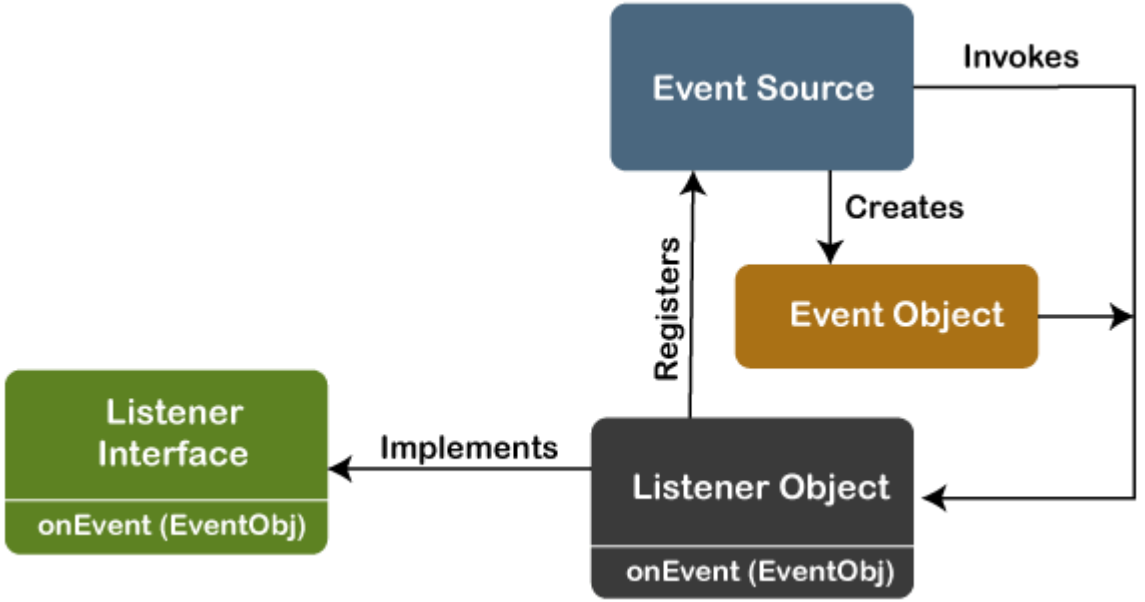


Event Model in Java

- **An event can be defined as changing the state of an object or behavior by performing actions.**
- **Actions can be a button click, cursor movement, keypress through keyboard or page scrolling, etc.**

- **The java.awt.event package can be used to provide various event classes.**
- Two types of events:
- 1. Foreground Events - require user interaction. (GUI) required.
- 2. Background Events – No interaction required. Ex: OS failure

Event Processing in Java



- Java support event processing since Java 1.0. It provides support for AWT (Abstract Window Toolkit), which is an API used to develop the Desktop application.
- Basically, an Event Model is based on the following three components:
- Events
- Events Sources
- Events Listeners

Event Processing in Java

- The Events are the objects that define state change in a source. An event can be generated as a reaction of a user while interacting with GUI elements.

- An event source is an object that causes and generates an event. It generates an event when the internal state of the object is changed. (for a keyboard event listener, the method will be called as **addKeyListener**(), for mouse **addMouseMotionListener**()).
- An event listener is an object that is invoked when an event triggers. The listeners require **two things**; **first**, it must be **registered** with a source; however, it can be registered with several resources to receive notification about the events. **Second**, it must **implement** the **methods** to receive and process the received notifications.

The Delegation Event Model

- The delegation event model, which defines standard and consistent mechanisms to generate and process events. Its concept is quite simple: a source generates an event and sends it to one or more listeners.
- In this scheme, the listener simply waits until it receives an event.
- Once received, the listener processes the event and then returns.
- The advantage of this design is that the application logic that processes events is cleanly separated from the user interface logic that generates those events.

Summary of Event Handling

1.User Interaction with a component is required to generate an event.

2.The object of the respective event class is created automatically after event generation, and it holds all information of the event source.

3.The newly created object is passed to the methods of the registered listener.

4.The method executes and returns the result.

Event Classes in Java

Event Class	Listener Interface	Description
ActionEvent	ActionListener	An event that indicates that a component-defined action occurred like a button click or selecting an item from the menu-item list.
AdjustmentEvent	AdjustmentListener	The adjustment event is emitted by an Adjustable object like Scrollbar.
ComponentEvent	ComponentListener	An event that indicates that a component moved, the size changed or changed its visibility.
ContainerEvent	ContainerListener	When a component is added to a container (or) removed from it, then this event is generated by a container object.
FocusEvent	FocusListener	These are focus-related events, which include focus, focusin, focusout, and blur.
ItemEvent	ItemListener	An event that indicates whether an item was selected or not.
KeyEvent	KeyListener	An event that occurs due to a sequence of keypresses on the keyboard.
MouseEvent	MouseListener & MouseMotionListener	The events that occur due to the user interaction with the mouse (Pointing Device).
MouseWheelEvent	MouseWheelListener	An event that specifies that the mouse wheel was rotated in a component.
TextEvent	TextListener	An event that occurs when an object's text changes.
WindowEvent	WindowListener	An event which indicates whether a window has changed its status or not.

Different Interfaces Consists of Different Methods

Listener Interface	Methods
ActionListener	•actionPerformed()
AdjustmentListener	•adjustmentValueChanged()
ComponentListener	•componentResized() •componentShown() •componentMoved() •componentHidden()
ContainerListener	•componentAdded() •componentRemoved()
FocusListener	•focusGained() •focusLost()
ItemListener	•itemStateChanged()
KeyListener	•keyTyped() •keyPressed() •keyReleased()
MouseListener	•mousePressed() •**mouseClicked()** •mouseEntered() •mouseExited() •mouseReleased()
MouseMotionListener	•mouseMoved() •mouseDragged()
MouseWheelListener	•mouseWheelMoved()
TextListener	•textChanged()
WindowListener	•windowActivated() •windowDeactivated() •windowOpened() •windowClosed() •windowClosing() •windowIconified() •windowDeiconified()

Event Classes in Java

- Demo on ActionListener with JFrame Form Design
- Demo on ActionListener with Code

Adapter Classes

- The Adapter class in Java allows us to edit all of an interface's methods by default; we don't have to modify all of the interface's methods, therefore we may claim it minimizes the coding burden.
- Adapter Class is a simple java class that implements an interface with only an empty implementation.
- The adapter classes are found in java.awt.event, java.awt.dnd and javax.swing.event packages.
- It increases the reusability of the class.

Adapter Classes

```
class WindowEventFrame extends Frame implements
WindowListener {
public void windowClosing(WindowEvent e){
      System.exit(o);
   }
publicvoid windowOpened(WindowEvent e){}
publicvoid windowClosed(WindowEvent e){}
publicvoid windowActivated(WindowEvent e){}
publicvoid windowDeactivated(WindowEvent e){}
publicvoid windowlconified(WindowEvent e){}
publicvoid windowDeiconified(WindowEvent e){}
.....
}
```

Adapter Classes

java.awt.event Adapter classes

Adapter class	Listener interface
WindowAdapter	WindowListener
KeyAdapter	KeyListener
MouseAdapter	MouseListener
MouseMotionAdapter	MouseMotionListener
FocusAdapter	FocusListener
ComponentAdapter	ComponentListener
ContainerAdapter	ContainerListener
HierarchyBoundsAdapter	HierarchyBoundsListener

Adapter Classes

java.awt.dnd Adapter classes

Adapter class	Listener interface
DragSourceAdapter	DragSourceListener
DragTargetAdapter	DragTargetListener

javax.swing.event Adapter classes

Adapter class	Listener interface
MouseInputAdapter	MouseInputListener
InternalFrameAdapter	InternalFrameListener

Adapter Classes

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

public class AdapterClassDemo {
AdapterClassDemo() {
f = new Frame ("Window Adapter");

f.addWindowListener (new WindowAdapter() {
        public void windowClosing (WindowEvent e) {
          f.dispose();
        }
      });
      f.setSize (400, 400);
      f.setLayout (null);
      f.setVisible (true);
   }

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

Adapter Classes

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

public class AdapterClassDemo2 extends MouseAdapter {
   Frame f;
   AdapterClassDemo2() {
      f = new Frame ("Mouse Adapter");
      f.addMouseListener(this);
      f.setSize (300, 300);          f.setLayout (null);    f.setVisible (true);
   }

   public void mouseClicked (MouseEvent e) {
      Graphics g = f.getGraphics();
      g.setColor (Color.BLUE);
      g.fillOval (e.getX(), e.getY(), 30, 30);
  }
public static void main(String[] args) {
   new AdapterClassDemo2();    } }
```

Activity

- https://www.menti.com/al9dwwaath76

Login Form

- Login Form Demo with JDBC Connection

Mini Project

- Add the Login Form, once the login is successful the Registration Form page should be displayed.
- Design Registration Form with Swing Components include
  - Enrollment number,
  - Name,
  - DOB,
  - Department,
  - Semester (ComboBox),
  - Gender (RadioButton),
  - Interested Subjects (Atleast 4 check boxes)
  - Email ID
  - Address (TextArea)
- Add two buttons (Save and Reset). Save button should insert filled form values into the table. Reset should clear all the fields in the form.

Contents

- Swing features
- Swing Containers
- Jframe
- Jpanel
- JWindow
- Swing components : JLabel, ImageIcon, JTextField, JButton, JToggeleButton, JCheckBox, JRadioButton, JTabbedPane, JScrollPane, JList, JComboBox, JTree, JTable
- Layout managers
- Event model in Java
- Event classes
- Event listeners
- Adapter classes

UP Next?

- Servlet Introduction,
- Servlet Life Cycle(SLC),
- Types of Servlet,
- Servlet Configuration with Deployment Descriptor,
- Working with ServletContext and ServletConfig Object,
- Attributes in Servlet,
- Response and Redirection using Request Dispacher and using sendRedirect Method,
- Filter API,
- Manipulating Responses using Filter API,
- Session Tracking: using Cookies,
- HTTPSession,
- Hidden Form Fields and URL Rewriting,
- Types of Servlet Event: ContextLevel and SessionLevel

END OF UNIT - 2