

by

Inna Maximova

Supraja Elecharala

Malav Desai

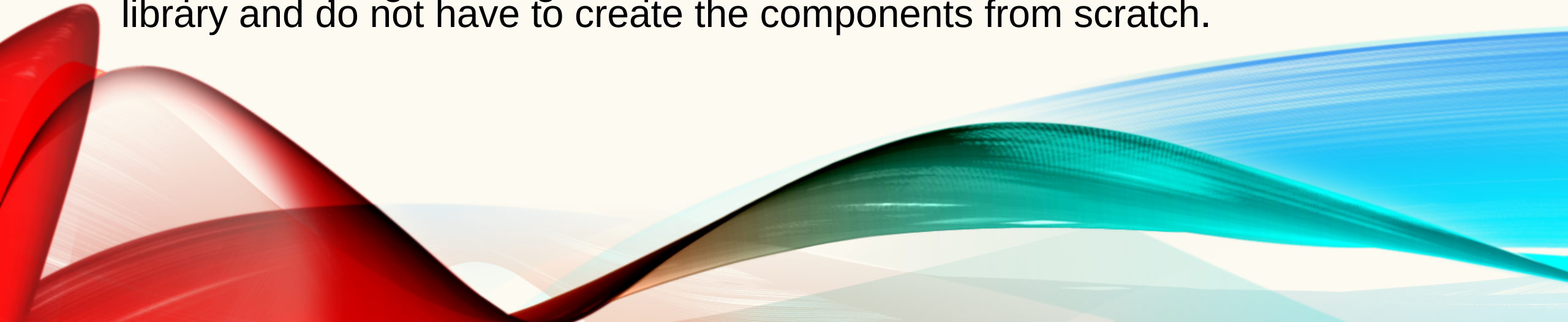
Mark Steven Blue

Oleksandr Shyryayev

# WHAT IS SWING IN JAVA?

**Swing in Java** is a Graphical User Interface (GUI) toolkit that includes the GUI components. Swing provides a rich set of widgets and packages to make sophisticated GUI components for Java applications. Swing is a part of Java Foundation Classes(JFC), which is an API for Java GUI programming that provide GUI.

The Java Swing library is built on top of the Java Abstract Widget Toolkit (**AWT**), an older, platform dependent GUI toolkit. You can use the Java simple GUI programming components like button, textbox, etc., from the library and do not have to create the components from scratch.



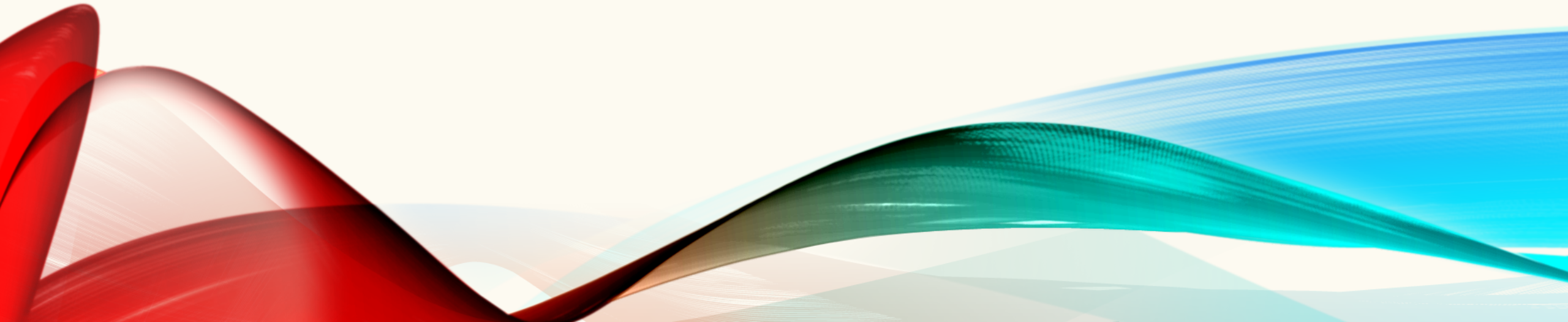
# DIFFERENCE BETWEEN AWT AND SWING

No.	Java AWT	Java Swing
1)	AWT components are <b>platform-dependent</b> .	Java swing components are <b>platform-independent</b> .
2)	AWT components are <b>heavyweight</b> .	Swing components are <b>lightweight</b> .
3)	AWT <b>doesn't support pluggable look and feel</b> .	Swing <b>supports pluggable look and feel</b> .
4)	AWT provides <b>less components</b> than Swing.	Swing provides <b>more powerful components</b> such as tables, lists, scrollpanes, colorchooser, tabbedpane etc.
5)	AWT <b>doesn't follows MVC</b> (Model View Controller) where model represents data, view represents presentation and controller acts as an interface between model and view.	Swing <b>follows MVC</b> .

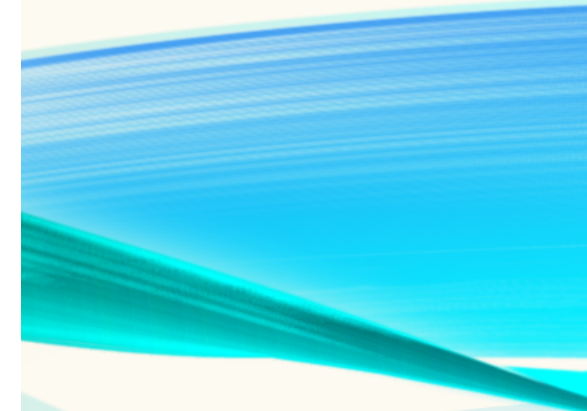


# WHAT IS JFC

The Java Foundation Classes (JFC) are a set of GUI components which simplify the development of desktop applications.

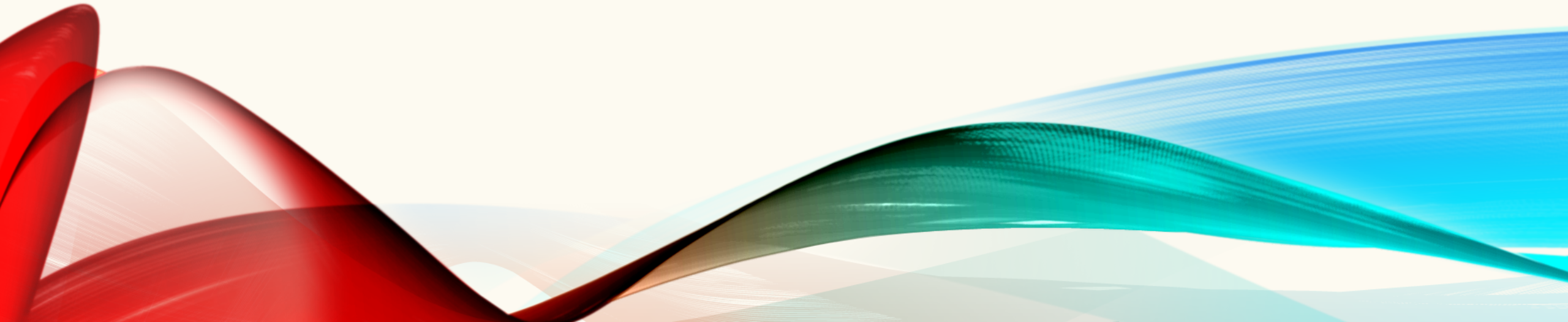


## An abstract graphic design featuring flowing, organic shapes in various shades of red and white. The composition is dynamic, with a large, bright white shape in the center, surrounded by deep red and lighter pinkish-red elements that create a sense of movement and depth. The background is a solid, light cream color.



# WHAT IS GUI IN JAVA?

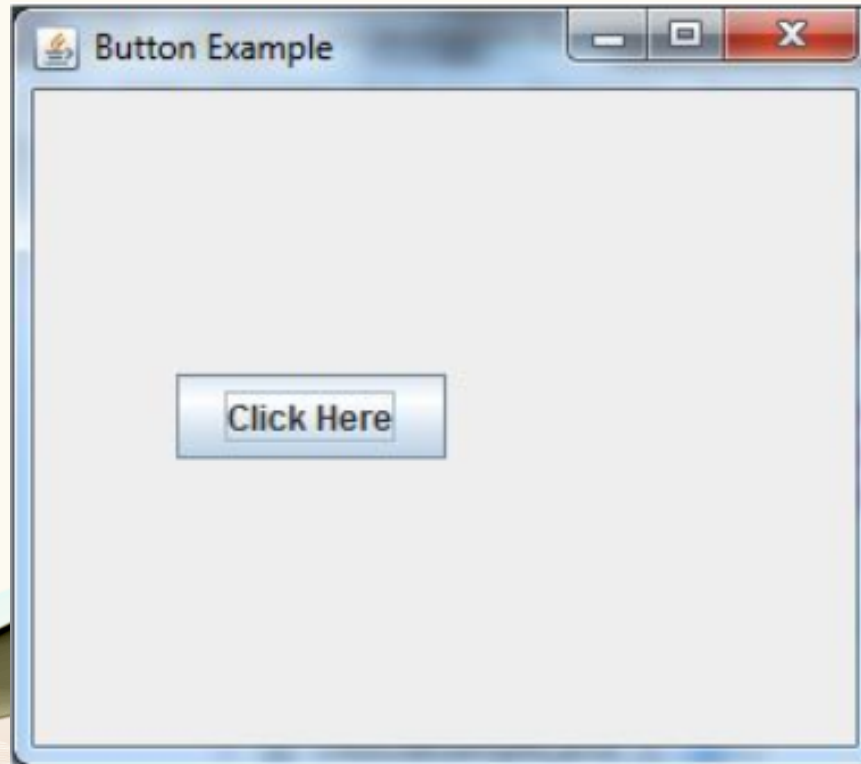
**GUI (Graphical User Interface)** in Java is an easy-to-use visual experience builder for Java applications. It is mainly made of graphical components like buttons, labels, windows, etc. through which the user can interact with an application. GUI plays an important role to build easy interfaces for Java applications.



# JAVA JBUTTON

The JButton class is used to create a labeled button that has platform independent implementation. The application result in some action when the button is pushed. It inherits AbstractButton class.

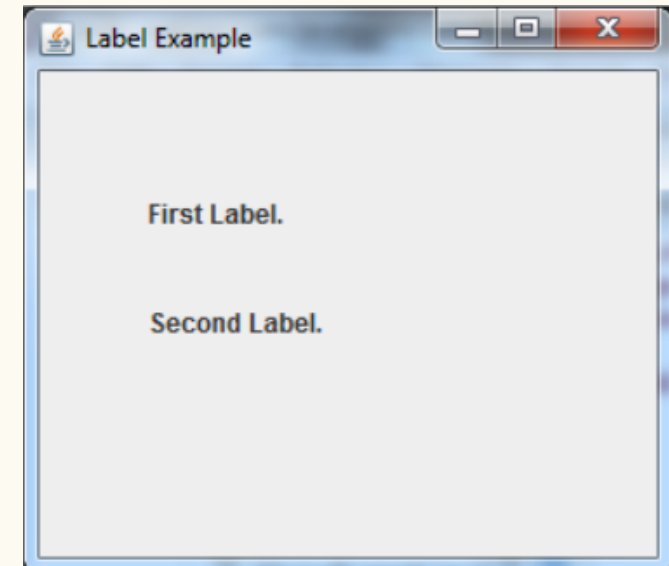
```
import javax.swing.*;  
  
public class ButtonExample {  
    public static void main(String[] args) {  
        JFrame f=new JFrame("Button Example");  
        JButton b=new JButton("Click Here");  
        b.setBounds(50,100,95,30);  
        f.add(b);  
        f.setSize(400,400);  
        f.setLayout(null);  
        f.setVisible(true);  
    }  
}
```



# JAVA JLABEL

The object of JLabel class is a component for placing text in a container. It is used to display a single line of read only text. The text can be changed by an application but a user cannot edit it directly. It inherits JComponent class.

```
import javax.swing.*;
class LabelExample
{
    public static void main(String args[])
    {
        JFrame f= new JFrame("Label Example");
        JLabel l1,l2;
        l1=new JLabel("First Label.");
        l1.setBounds(50,50, 100,30);
        l2=new JLabel("Second Label.");
        l2.setBounds(50,100, 100,30);
        f.add(l1); f.add(l2);
        f.setSize(300,300);
        f.setLayout(null);
        f.setVisible(true);
    }
}
```

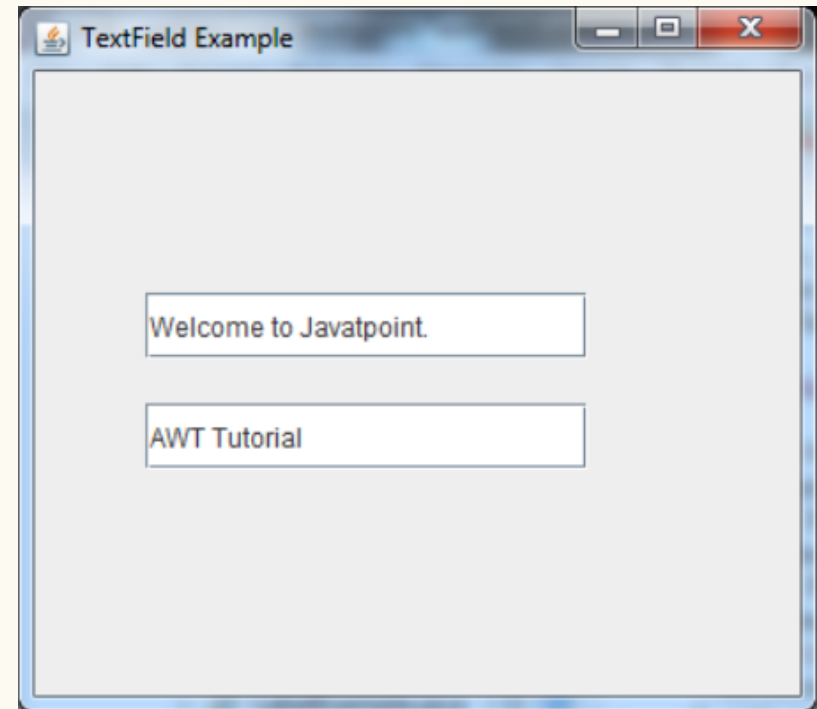




# JAVA JTEXTFIELD

The object of a JTextField class is a text component that allows the editing of a single line text. It inherits JTextComponent class.

```
import javax.swing.*;
class TextFieldExample
{
    public static void main(String args[])
    {
        JFrame f= new JFrame("TextField Example");
        JTextField t1,t2;
        t1=new JTextField("Welcome to Javatpoint.");
        t1.setBounds(50,100, 200,30);
        t2=new JTextField("AWT Tutorial");
        t2.setBounds(50,150, 200,30);
        f.add(t1); f.add(t2);
        f.setSize(400,400);
        f.setLayout(null);
        f.setVisible(true);
    }
}
```



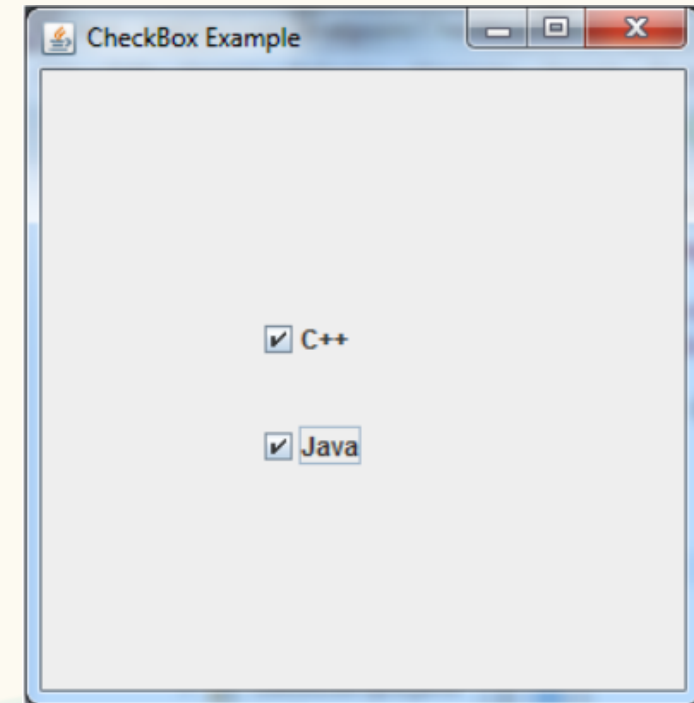
# JAVA JCHECKBOX

The JCheckBox class is used to create a checkbox. It is used to turn an option on (true) or off (false). Clicking on a CheckBox changes its state from "on" to "off" or from "off" to "on". It inherits JToggleButton class.

```
import javax.swing.*;

public class CheckBoxExample
{
    CheckBoxExample(){
        JFrame f= new JFrame("CheckBox Example");
        JCheckBox checkBox1 = new JCheckBox("C++");
        checkBox1.setBounds(100,100, 50,50);
        JCheckBox checkBox2 = new JCheckBox("Java", true);
        checkBox2.setBounds(100,150, 50,50);
        f.add(checkBox1);
        f.add(checkBox2);
        f.setSize(400,400);
        f.setLayout(null);
        f.setVisible(true);
    }

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



# JAVA JRadioButton

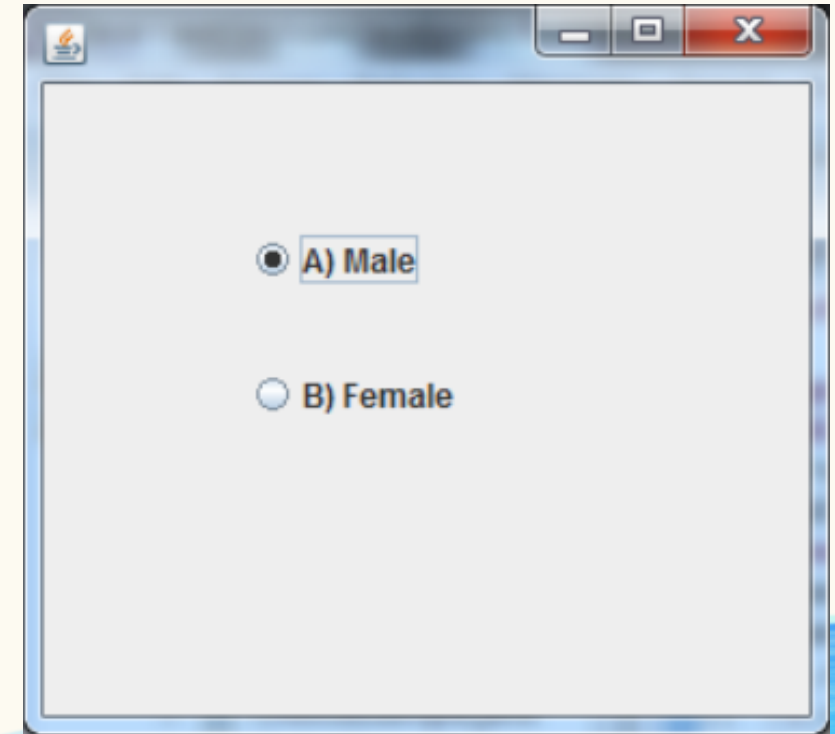
The JRadioButton class is used to create a radio button. It is used to choose one option from multiple options. It is widely used in exam systems or quiz.

It should be added in ButtonGroup to select one radio button only.

```
import javax.swing.*;

public class RadioButtonExample {
    JFrame f;
    RadioButtonExample(){
        f=new JFrame();
        JRadioButton r1=new JRadioButton("A) Male");
        JRadioButton r2=new JRadioButton("B) Female");
        r1.setBounds(75,50,100,30);
        r2.setBounds(75,100,100,30);
        ButtonGroup bg=new ButtonGroup();
        bg.add(r1);bg.add(r2);
        f.add(r1);f.add(r2);
        f.setSize(300,300);
        f.setLayout(null);
        f.setVisible(true);
    }

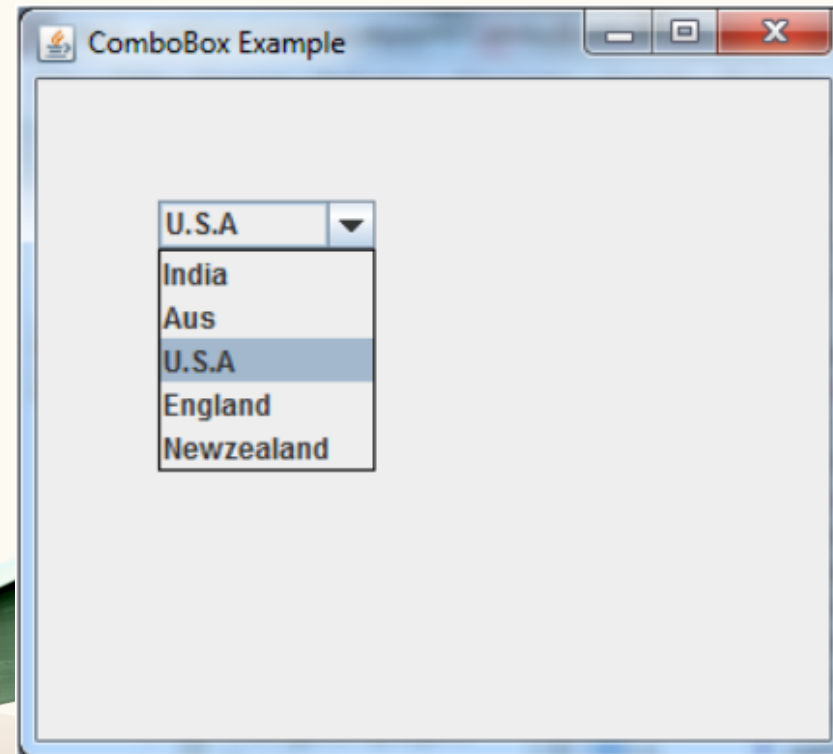
    public static void main(String[] args) {
        new RadioButtonExample();
    }
}
```



# JAVA JCOMBOBOX

The object of Choice class is used to show popup menu of choices. Choice selected by user is shown on the top of a menu. It inherits JComponent class.

```
import javax.swing.*;
public class ComboBoxExample {
    JFrame f;
    ComboBoxExample(){
        f=new JFrame("ComboBox Example");
        String country[]={"India","Aus","U.S.A","England","Newzealand"};
        JComboBox cb=new JComboBox(country);
        cb.setBounds(50, 50,90,20);
        f.add(cb);
        f.setLayout(null);
        f.setSize(400,500);
        f.setVisible(true);
    }
    public static void main(String[] args) {
        new ComboBoxExample();
    }
}
```





# JAVA JTABLE

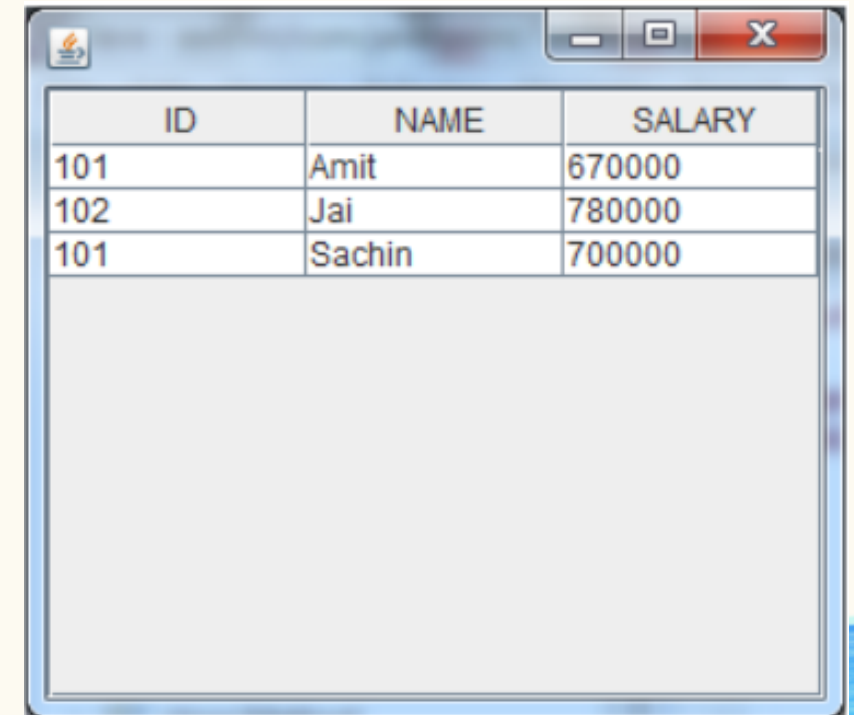
The JTable class is used to display data in tabular form. It is composed of rows and columns.

```
import javax.swing.*;

public class TableExample {
    JFrame f;
    TableExample(){
        f=new JFrame();
        String data[][]={{ "101","Amit","670000"},
                           {"102","Jai","780000"},
                           {"101","Sachin","700000"}};

        String column[]={"ID","NAME","SALARY"};
        JTable jt=new JTable(data,column);
        jt.setBounds(30,40,200,300);
        JScrollPane sp=new JScrollPane(jt);
        f.add(sp);
        f.setSize(300,400);
        f.setVisible(true);
    }

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



ID	NAME	SALARY
101	Amit	670000
102	Jai	780000
101	Sachin	700000

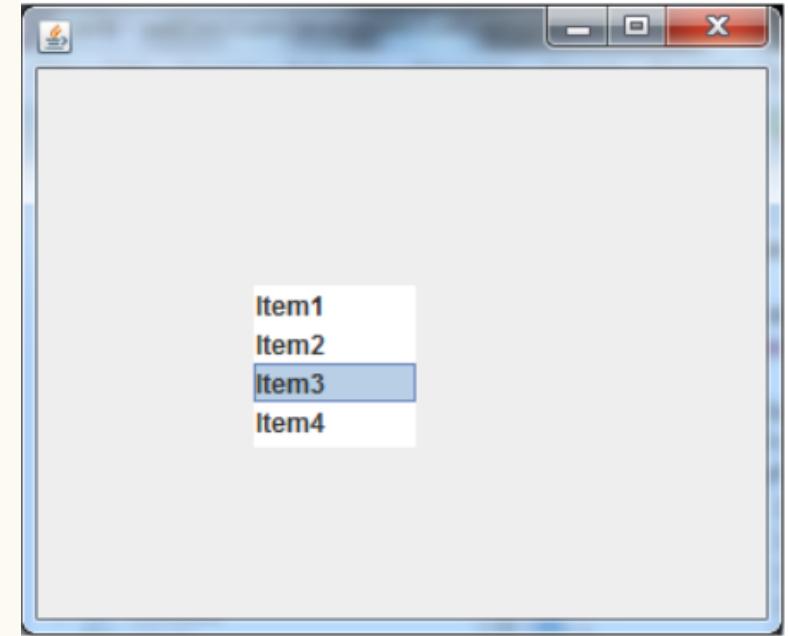
# JAVA JLIST

The object of JList class represents a list of text items. The list of text items can be set up so that the user can choose either one item or multiple items. It inherits JComponent class.

```
import javax.swing.*;

public class ListExample
{
    ListExample(){
        JFrame f= new JFrame();
        DefaultListModel<String> l1 = new DefaultListModel<>();
        l1.addElement("Item1");
        l1.addElement("Item2");
        l1.addElement("Item3");
        l1.addElement("Item4");
        JList<String> list = new JList<>(l1);
        list.setBounds(100,100, 75,75);
        f.add(list);
        f.setSize(400,400);
        f.setLayout(null);
        f.setVisible(true);
    }

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



# JAVA JOPTIONPANE

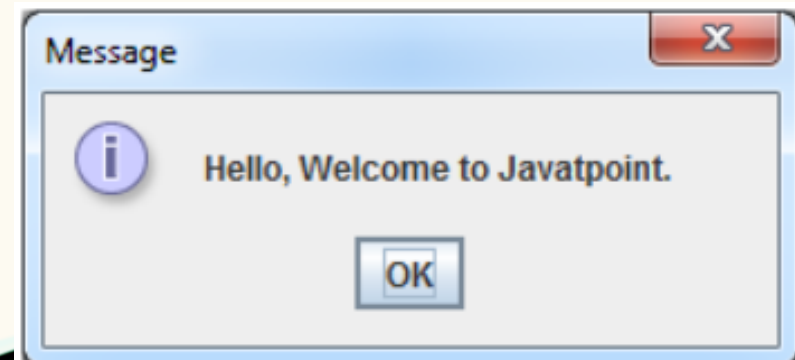
The JOptionPane class is used to provide standard dialog boxes such as message dialog box, confirm dialog box and input dialog box. These dialog boxes are used to display information or get input from the user. The JOptionPane class inherits JComponent class.

```
import javax.swing.*;

public class OptionPaneExample {
    JFrame f;

    OptionPaneExample(){
        f=new JFrame();
        JOptionPane.showMessageDialog(f,"Hello, Welcome to Javatpoint.");
    }

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



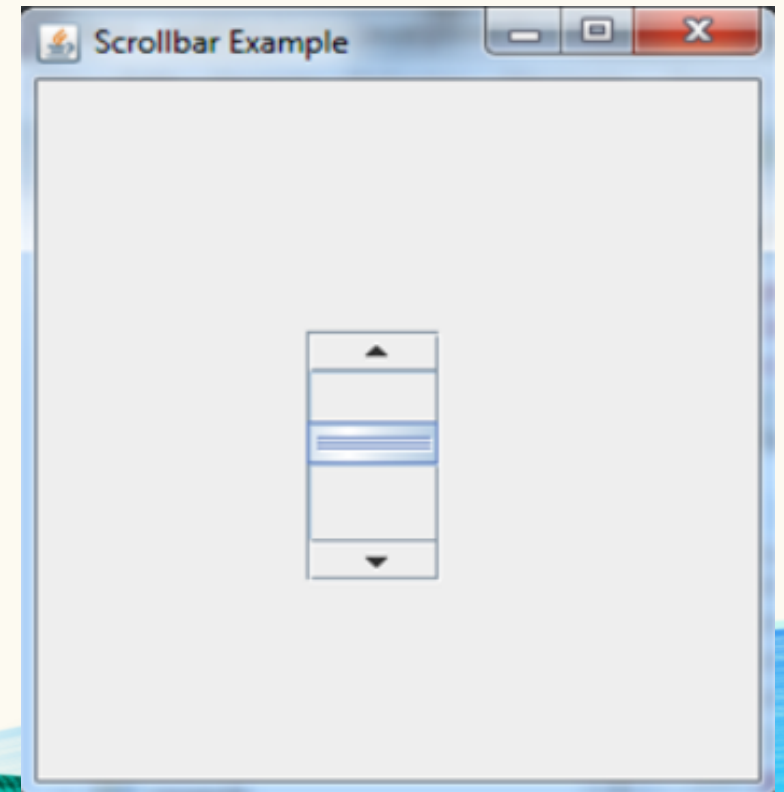
# JAVA JSCROLLBAR

The object of JScrollbar class is used to add horizontal and vertical scrollbar. It is an implementation of a scrollbar. It inherits JComponent class.

```
import javax.swing.*;

class ScrollBarExample
{
    ScrollBarExample(){
        JFrame f= new JFrame("Scrollbar Example");
        JScrollbar s=new JScrollbar();
        s.setBounds(100,100, 50,100);
        f.add(s);
        f.setSize(400,400);
        f.setLayout(null);
        f.setVisible(true);
    }

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





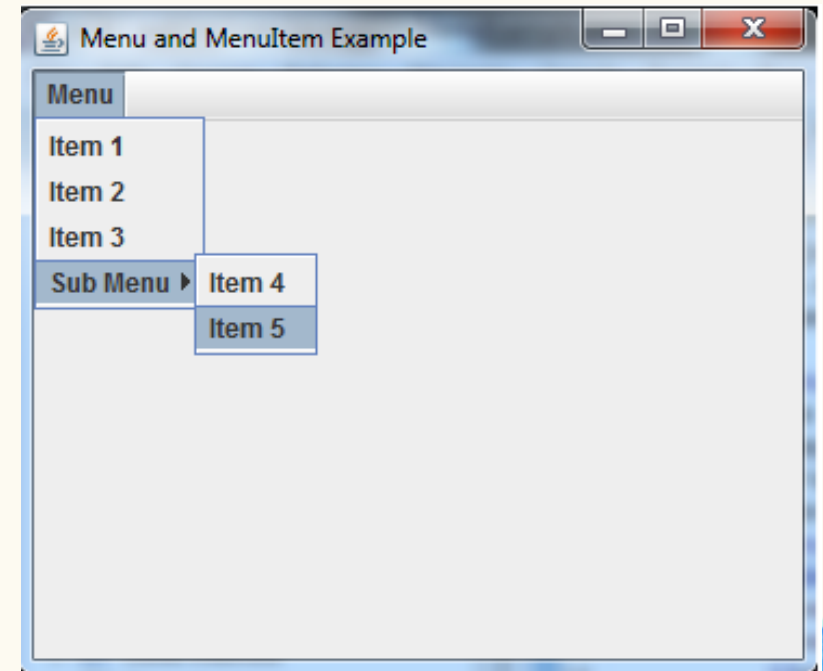
# JMENUBAR, JMENU AND JMENUITEM

The JMenuBar class is used to display menubar on the window or frame. It may have several menus.

The object of JMenu class is a pull down menu component which is displayed from the menu bar. It inherits the JMenuItem class.

The object of JMenuItem class adds a simple labeled menu item. The items used in a menu must belong to the JMenuItem or any of its subclass.

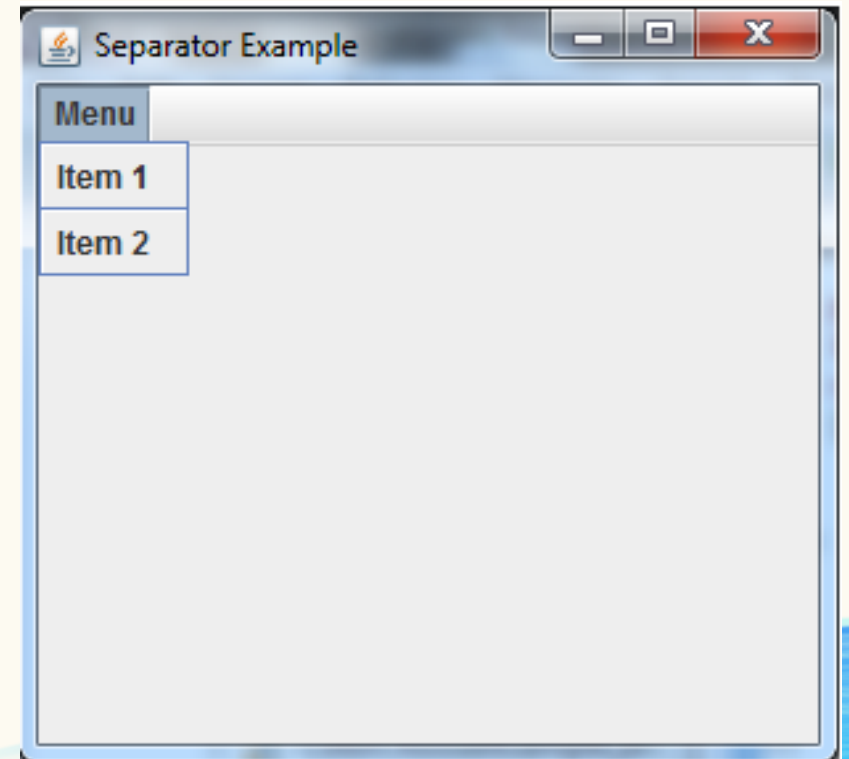
```
import javax.swing.*;
class MenuExample
{
    JMenu menu, submenu;
    JMenuItem i1, i2, i3, i4, i5;
    MenuExample(){
        JFrame f= new JFrame("Menu and MenuItem Example");
        JMenuBar mb=new JMenuBar();
        menu=new JMenu("Menu");
        submenu=new JMenu("Sub Menu");
        i1=new JMenuItem("Item 1");
        i2=new JMenuItem("Item 2");
        i3=new JMenuItem("Item 3");
        i4=new JMenuItem("Item 4");
        i5=new JMenuItem("Item 5");
        menu.add(i1); menu.add(i2); menu.add(i3);
        submenu.add(i4); submenu.add(i5);
        menu.add(submenu);
        mb.add(menu);
        f.setJMenuBar(mb);
        f.setSize(400,400);
        f.setLayout(null);
        f.setVisible(true);
    }
    public static void main(String args[])
    {
        new MenuExample();
    }
}
```



# JAVA JSEPARATOR

The object of JSeparator class is used to provide a general purpose component for implementing divider lines. It is used to draw a line to separate widgets in a Layout. It inherits JComponent class.

```
import javax.swing.*;
class SeparatorExample
{
    JMenu menu, submenu;
    JMenuItem i1, i2, i3, i4, i5;
    SeparatorExample() {
        JFrame f= new JFrame("Separator Example");
        JMenuBar mb=new JMenuBar();
        menu=new JMenu("Menu");
        i1=new JMenuItem("Item 1");
        i2=new JMenuItem("Item 2");
        menu.add(i1);
        menu.addSeparator();
        menu.add(i2);
        mb.add(menu);
        f.setJMenuBar(mb);
        f.setSize(400,400);
        f.setLayout(null);
        f.setVisible(true);
    }
    public static void main(String args[])
    {
        new SeparatorExample();
    }
}
```

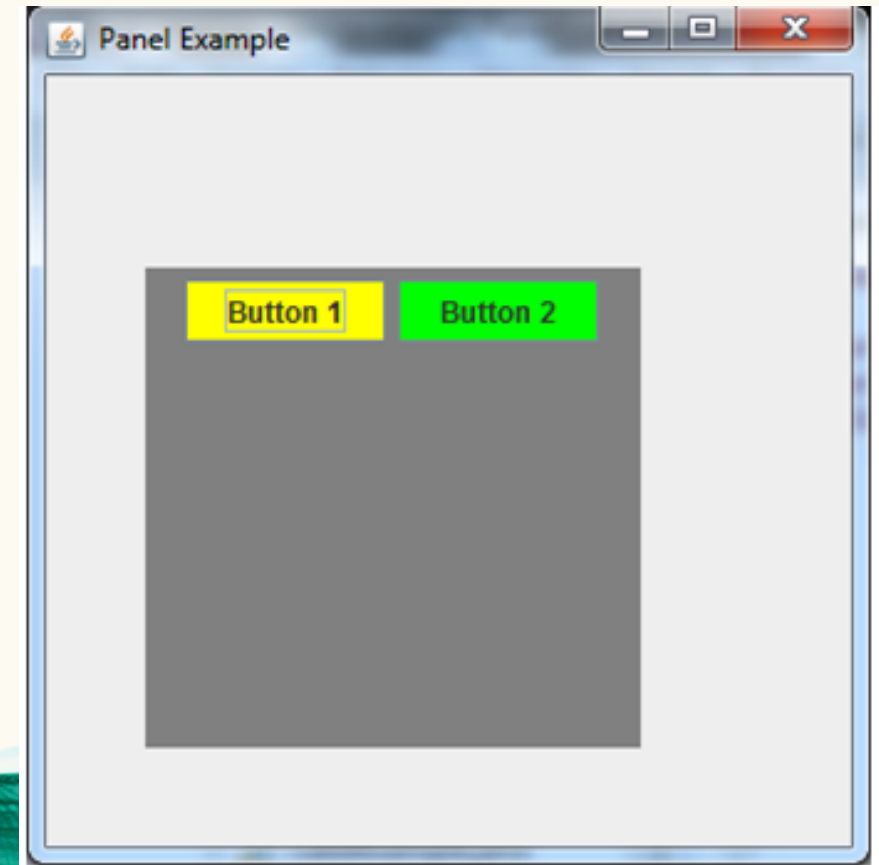


# JAVA JPANEL

The JPanel is a simplest container class. It provides space in which an application can attach any other component. It inherits the JComponents class.

It doesn't have title bar.

```
import java.awt.*;
import javax.swing.*;
public class PanelExample {
    PanelExample()
    {
        JFrame f= new JFrame("Panel Example");
        JPanel panel=new JPanel();
        panel.setBounds(40,80,200,200);
        panel.setBackground(Color.gray);
        JButton b1=new JButton("Button 1");
        b1.setBounds(50,100,80,30);
        b1.setBackground(Color.yellow);
        JButton b2=new JButton("Button 2");
        b2.setBounds(100,100,80,30);
        b2.setBackground(Color.green);
        panel.add(b1); panel.add(b2);
        f.add(panel);
        f.setSize(400,400);
        f.setLayout(null);
        f.setVisible(true);
    }
    public static void main(String args[])
    {
        new PanelExample();
    }
}
```

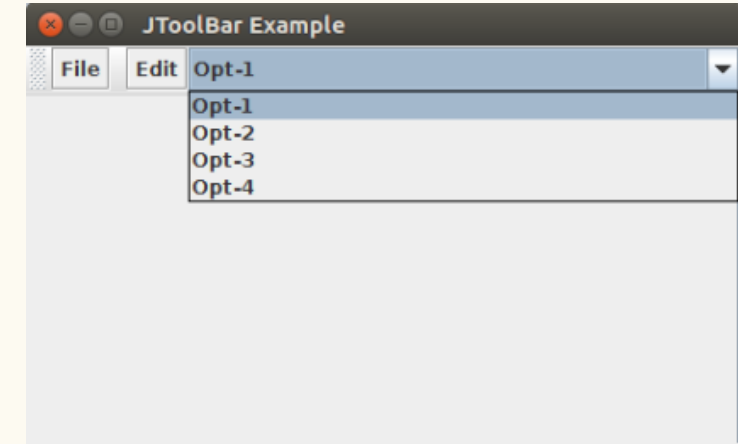


# JAVA JTOOLBAR

JToolBar container allows us to group other components, usually buttons with icons in a row or column. JToolBar provides a component which is useful for displaying commonly used actions or controls.

```
import java.awt.BorderLayout;
import java.awt.Container;
import javax.swing.JButton;
import javax.swing.JComboBox;
import javax.swing.JFrame;
import javax.swing.JScrollPane;
import javax.swing.JTextArea;
import javax.swing.JToolBar;

public class JToolBarExample {
    public static void main(final String args[]) {
        JFrame myframe = new JFrame("JToolBar Example");
        myframe.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        JToolBar toolbar = new JToolBar();
        toolbar.setRollover(true);
        JButton button = new JButton("File");
        toolbar.add(button);
        toolbar.addSeparator();
        toolbar.add(new JButton("Edit"));
        toolbar.add(new JComboBox(new String[] { "Opt-1", "Opt-2", "Opt-3", "Opt-4" }));
        Container contentPane = myframe.getContentPane();
        contentPane.add(toolbar, BorderLayout.NORTH);
        JTextArea textArea = new JTextArea();
        JScrollPane mypane = new JScrollPane(textArea);
        contentPane.add(mypane, BorderLayout.EAST);
        myframe.setSize(450, 250);
        myframe.setVisible(true);
    }
}
```



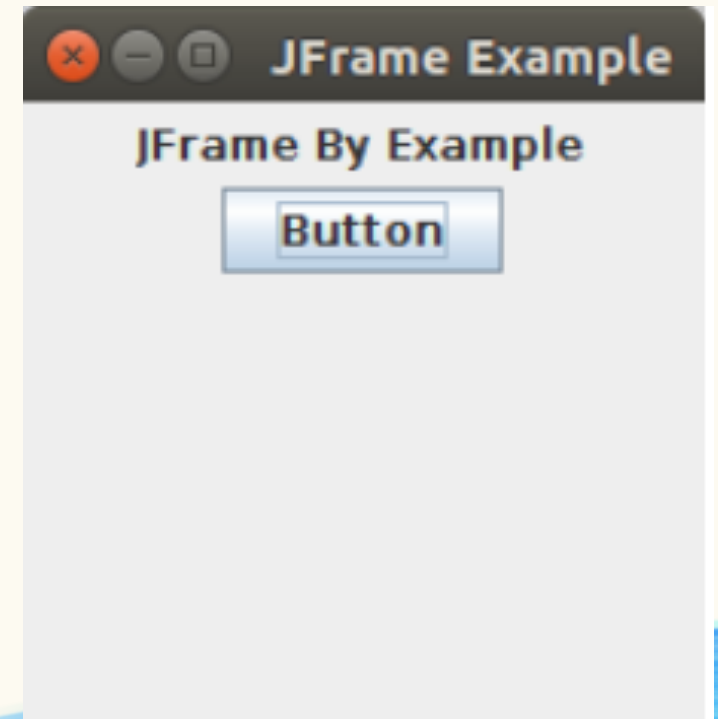


# JAVA JFRAME

The `javax.swing.JFrame` class is a type of container which inherits the `java.awt.Frame` class. `JFrame` works like the main window where components like labels, buttons, textfields are added to create a GUI.

Unlike `Frame`, `JFrame` has the option to hide or close the window with the help of `setDefaultCloseOperation(int)` method.

```
import java.awt.FlowLayout;
import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JPanel;
public class JFrameExample {
    public static void main(String s[]) {
        JFrame frame = new JFrame("JFrame Example");
        JPanel panel = new JPanel();
        panel.setLayout(new FlowLayout());
        JLabel label = new JLabel("JFrame By Example");
        JButton button = new JButton();
        button.setText("Button");
        panel.add(label);
        panel.add(button);
        frame.add(panel);
        frame.setSize(200, 300);
        frame.setLocationRelativeTo(null);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setVisible(true);
    }
}
```



# JAVA 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.

```
import java.awt.FlowLayout;
import javax.swing.JFrame;
import javax.swing.JScrollPane;
import javax.swing.JTextArea;

public class JScrollPaneExample {
    private static final long serialVersionUID = 1L;

    private static void createAndShowGUI() {

        // Create and set up the window.
        final JFrame frame = new JFrame("Scroll Pane Example");

        // Display the window.
        frame.setSize(500, 500);
        frame.setVisible(true);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

        // set flow layout for the frame
        frame.getContentPane().setLayout(new FlowLayout());

        JTextArea textArea = new JTextArea(20, 20);
        JScrollPane scrollableTextArea = new JScrollPane(textArea);

        scrollableTextArea.setHorizontalScrollBarPolicy(JScrollPane.HORIZONTAL_SCROLLBAR_ALWAYS);
        scrollableTextArea.setVerticalScrollBarPolicy(JScrollPane.VERTICAL_SCROLLBAR_ALWAYS);

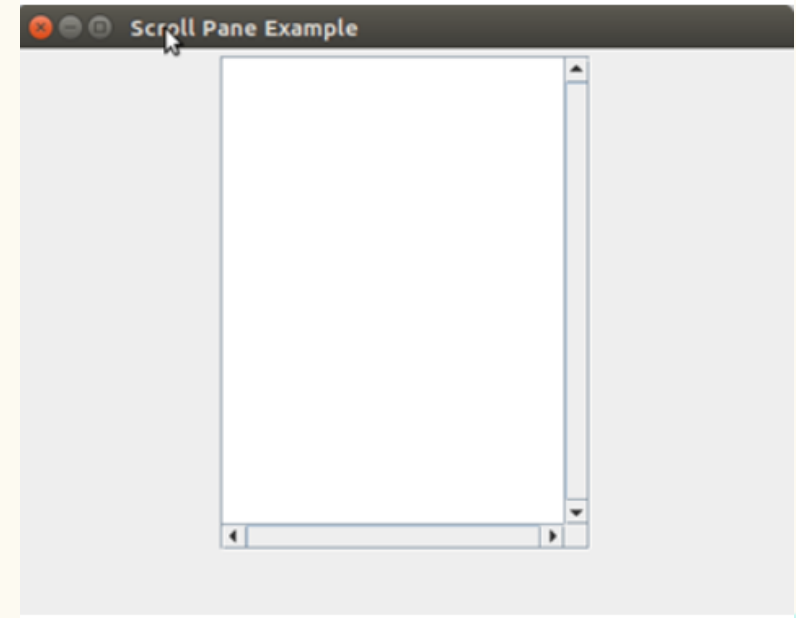
        frame.getContentPane().add(scrollableTextArea);
    }

    public static void main(String[] args) {

        javax.swing.SwingUtilities.invokeLater(new Runnable() {

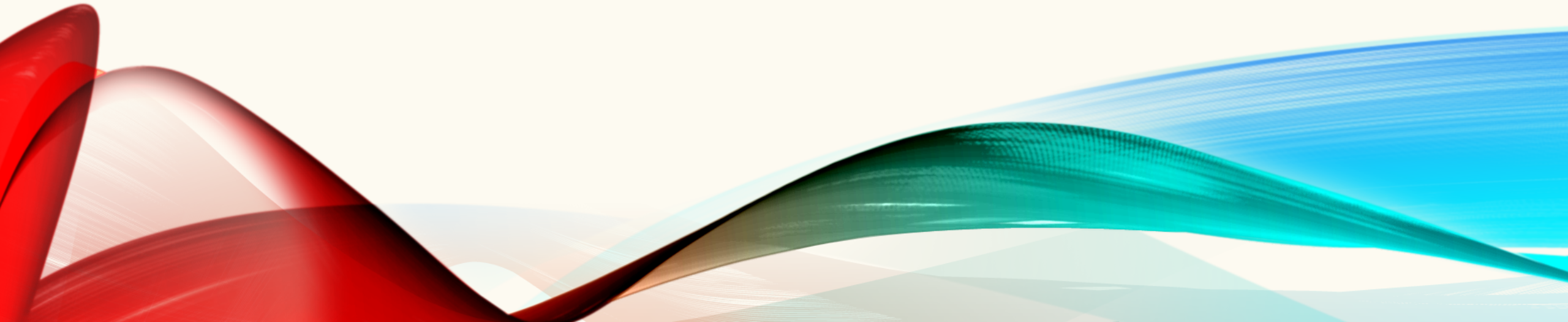
            public void run() {
                createAndShowGUI();
            }

        });
    }
}
```



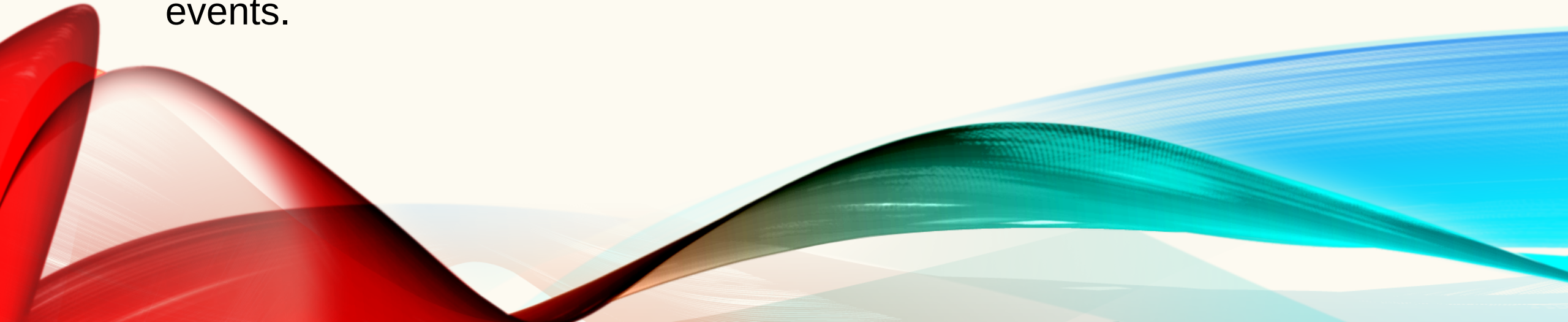
# WHAT IS AN EVENT?

Change in the state of an object is known as Event, i.e., event describes the change in the state of the source. Events are generated as a result of user interaction with the graphical user interface components.



# TYPES OF EVENT

- **Foreground Events** – These events require direct interaction of the user. They are generated as consequences of a person interacting with the graphical components in the Graphical User Interface. For example, clicking on a button, moving the mouse, entering a character through keyboard, selecting an item from list, scrolling the page, etc.
- **Background Events** – These events require the interaction of the end user. Operating system interrupts, hardware or software failure, timer expiration, and operation completion are some examples of background events.





# WHAT IS AN EVENT HANDLING?

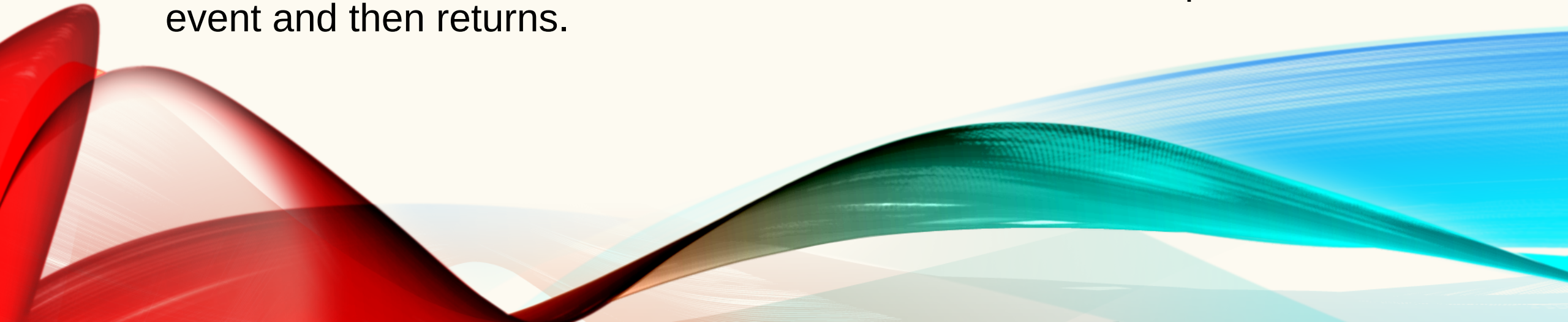
Event Handling is the mechanism that controls the event and decides what should happen if an event occurs. This mechanism has a code which is known as an event handler, that is executed when an event occurs.

Java uses the Delegation Event Model to handle the events. This model defines the standard mechanism to generate and handle the events.



# THE DELEGATION EVENT MODEL HAS THE FOLLOWING KEY PARTICIPANTS.

- **Source** – The source is an object on which the event occurs. Source is responsible for providing information of the occurred event to its handler. Java provides us with classes for the source object.
- **Listener** – It is also known as event handler. The listener is responsible for generating a response to an event. From the point of view of Java implementation, the listener is also an object. The listener waits till it receives an event. Once the event is received, the listener processes the event and then returns.



# EVENT CLASSES

## 1. [AWTEvent](#)

It is the root event class for all SWING events. This class and its subclasses supercede the original java.awt.Event class.

## 2. [ActionEvent](#)

The ActionEvent is generated when the button is clicked or the item of a list is double-clicked.

## 3. [InputEvent](#)

The InputEvent class is the root event class for all component-level input events.

## 4. [KeyEvent](#)

On entering the character the Key event is generated.

## 5. [MouseEvent](#)

This event indicates a mouse action occurred in a component.

## 6. [WindowEvent](#)

The object of this class represents the change in the state of a window.

## 7. [AdjustmentEvent](#)

The object of this class represents the adjustment event emitted by Adjustable objects.

## 8. [ComponentEvent](#)

The object of this class represents the change in the state of a window.

## 9. [ContainerEvent](#)

The object of this class represents the change in the state of a window.

## 10. [MouseMotionEvent](#)

The object of this class represents the change in the state of a window.

## 11. [PaintEvent](#)

The object of this class represents the change in the state of a window.

# EVENT LISTENER INTERFACES

## 1. ActionListener

This interface is used for receiving the action events.

## 2. ComponentListener

This interface is used for receiving the component events.

## 3. ItemListener

This interface is used for receiving the item events

## 4. KeyListener

This interface is used for receiving the key events.

## 5. MouseListener

This interface is used for receiving the mouse events.

## 6. WindowListener

This interface is used for receiving the window events.

## 7. AdjustmentListener

This interface is used for receiving the adjustment events.

## 8. ContainerListener

This interface is used for receiving the container events.

## 9. MouseMotionListener

This interface is used for receiving the mouse motion events.

## 10. FocusListener

This interface is used for receiving the focus events.



# EVENT ADAPTERS

Adapters are abstract classes for receiving various events. The methods in these classes are empty. These classes exist as convenience for creating listener objects.

1. [FocusAdapter](#)

An abstract adapter class for receiving focus events.

2. [KeyAdapter](#)

An abstract adapter class for receiving key events.

3. [MouseAdapter](#)

An abstract adapter class for receiving mouse events.

4. [MouseMotionAdapter](#)

An abstract adapter class for receiving mouse motion events.

5. [WindowAdapter](#)

An abstract adapter class for receiving window events.

# JAVA LAYOUTMANAGERS

The LayoutManagers are used to arrange components in a particular manner.

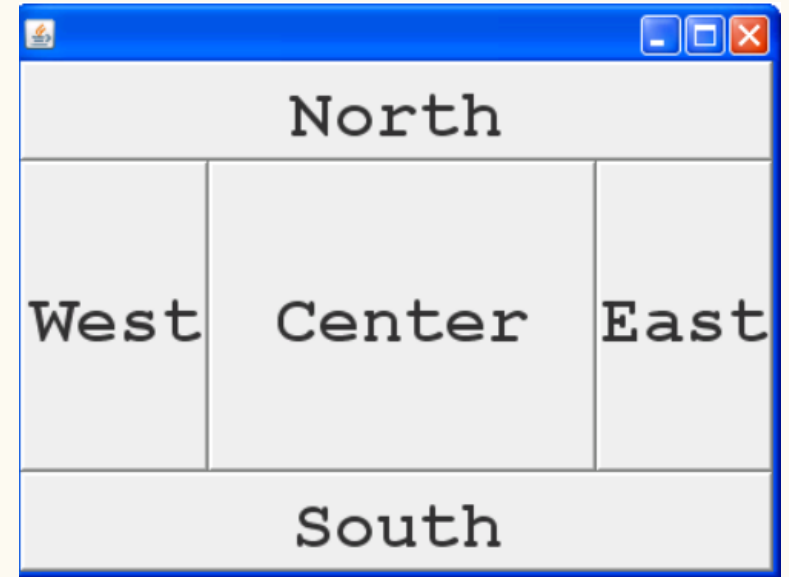
The Java

LayoutManagers facilitates us to control the positioning and size of the components in GUI forms. LayoutManager is an interface that is implemented by all the classes of layout managers. There are the following classes that represent the layout managers:

1. java.awt.BorderLayout
2. java.awt.FlowLayout
3. java.awt.GridLayout
4. java.awt.CardLayout
5. java.awt.GridBagLayout
6. javax.swing.BoxLayout
7. javax.swing.GroupLayout
8. javax.swing.ScrollPaneLayout
9. javax.swing.SpringLayout etc.

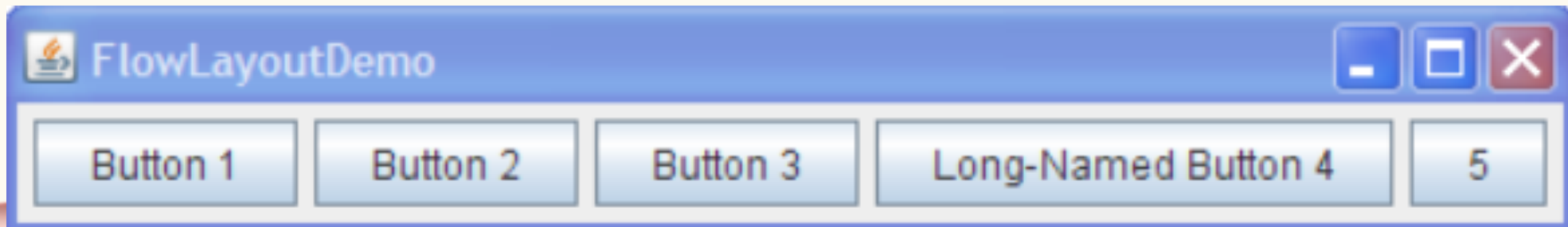
# BORDERLAYOUT

- Used to arrange components around a center panel.
- Sizes components to use all available space for their region.
  - North / South components will be as tall as their preferred height and as wide as the width of the component they're in.
  - East / West components will be as wide as their preferred width and as tall as the height of the component they're in minus the preferred heights of the North and South components.
  - Center will use all remaining space in the component after the North / South / East / West components have been sized.



# FLOWLAYOUT

- Used to arrange components in a straight horizontal line.
- If there is not enough space for all the components to fit, they'll be moved to the next line.
- Components are set at their preferred sizes.





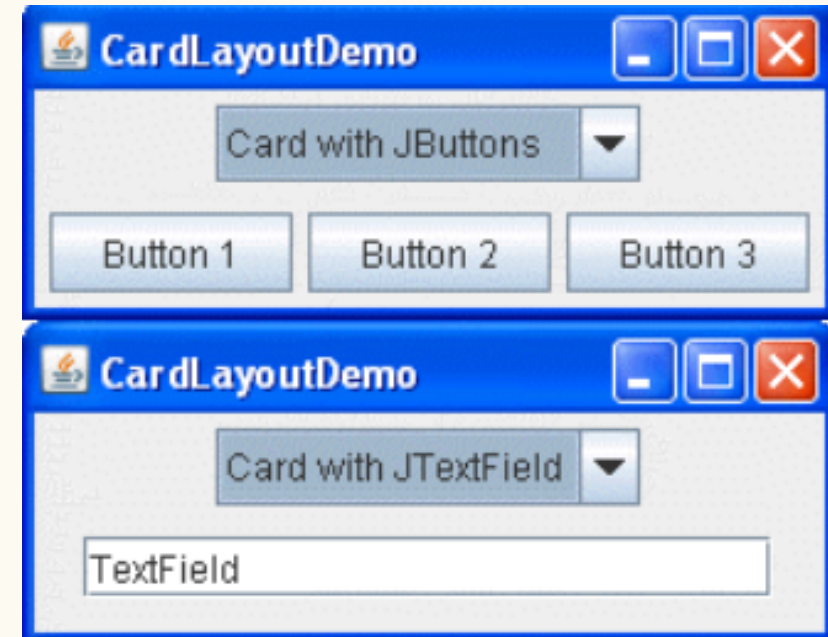
# GRIDLAYOUT

- Used to arrange components in an evenly spaced grid.



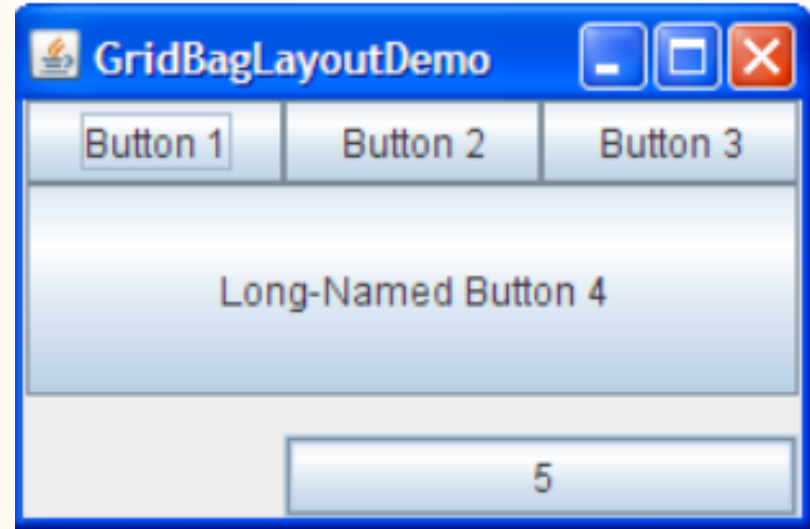
# CARDLAYOUT

- Used to show one component at a time from a set of components. It is like a deck of flash cards, where you show one of the cards at a time, hence the name. You can quickly change which component is shown at any time, e.g., when the user clicks a button.



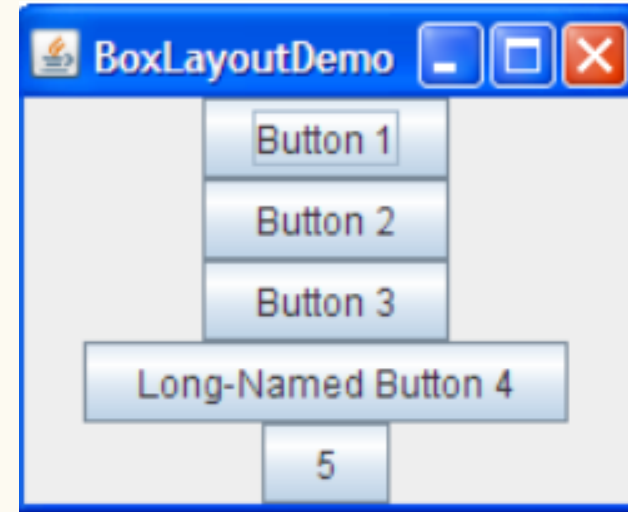
# GRIDBAGLAYOUT

- Aligns components by placing them within a grid of cells, allowing components to span more than one cell. The rows in the grid can have different heights, and grid columns can have different widths.



# BOXLAYOUT

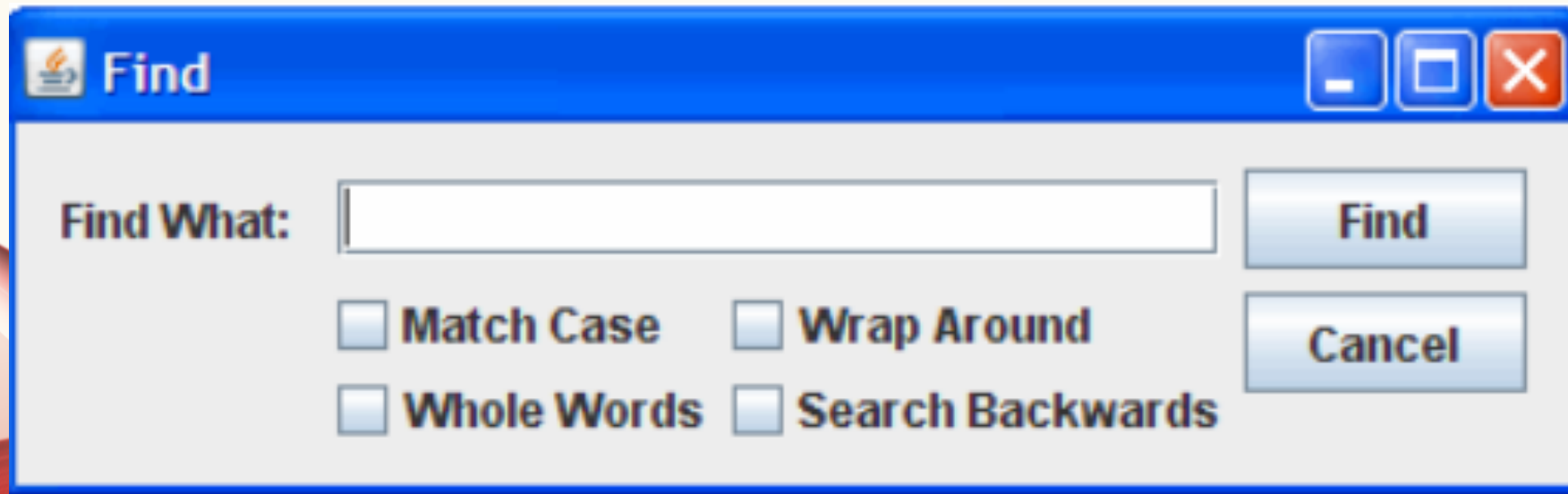
- Used to arrange components vertically or horizontally. BoxLayout is essentially a replacement for FlowLayout that is more powerful.





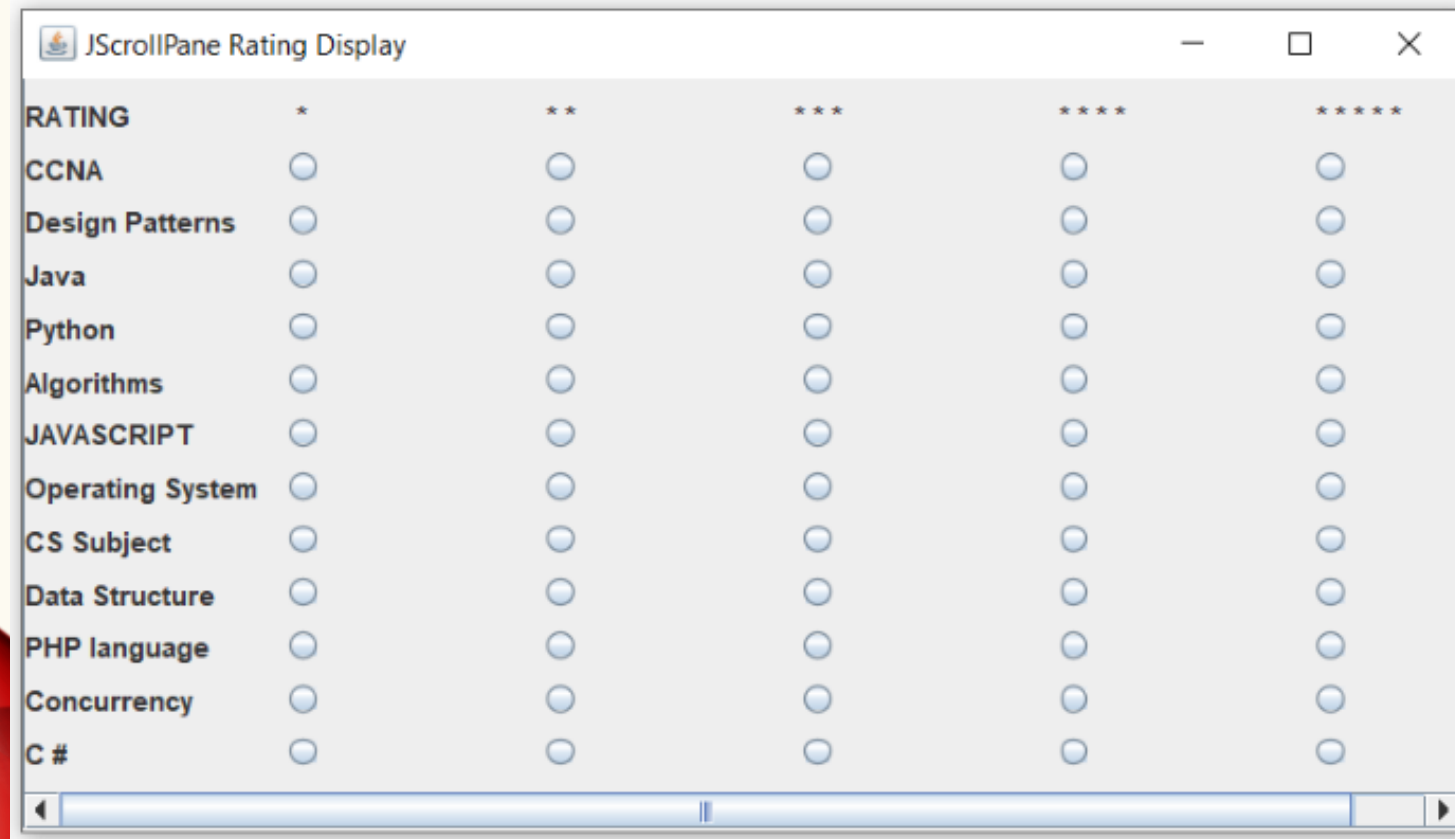
# GROUPLAYOUT

- GroupLayout is a layout manager that was developed for use by GUI builder tools, but it can also be used manually. GroupLayout works with the horizontal and vertical layouts separately. The layout is defined for each dimension independently. Consequently, however, each component needs to be defined twice in the layout.



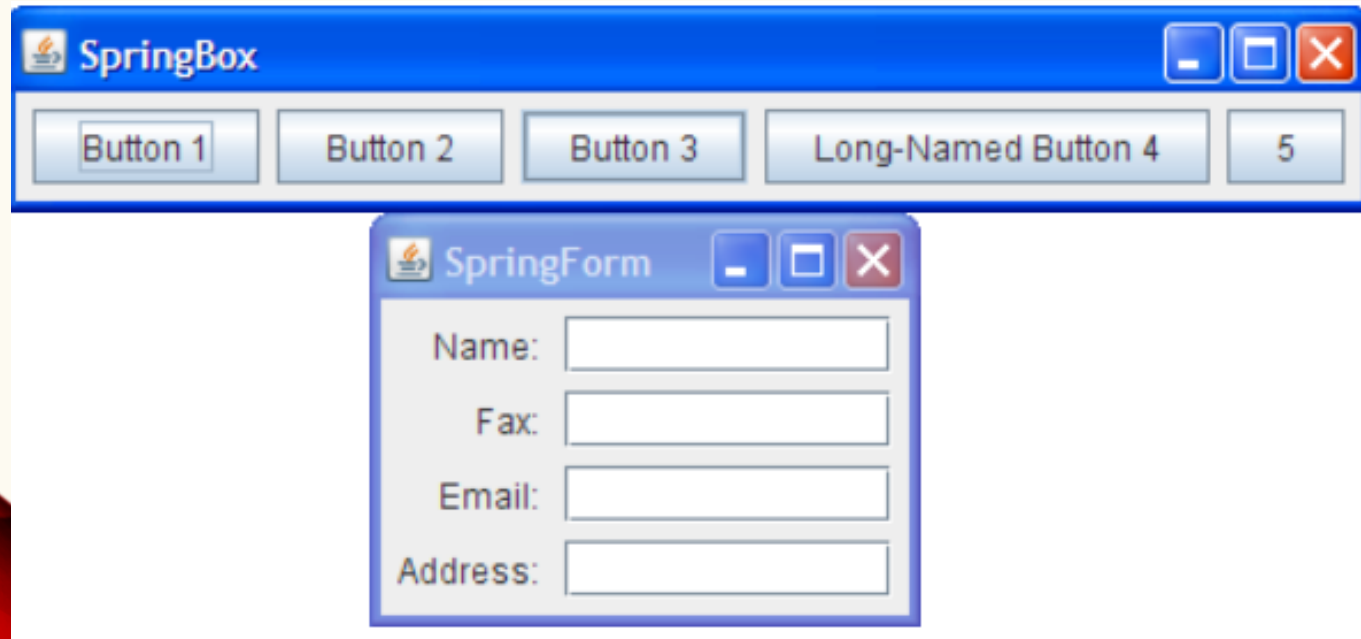
# SCROLLPANELAYOUT

- Responsible for nine components: a viewport, two scrollbars, a row header, a column header, and four "corner" components.



# SPRINGLAYOUT

- It lets you specify precise relationships between the edges of components under its control. For example, you might define that the left edge of one component is a certain distance (which can be dynamically calculated) from the right edge of a second component. SpringLayout lays out the children of its associated container according to a set of constraints.



# REFERENCES:

- [Tutorials List – Javatpoint](#)
- [The Java™ Tutorials \(oracle.com\)](#)
- [Java Swings - Wikibooks, open books for an open world](#)
- [Java Swing Layouts Example - Examples Java Code Geeks – 2023](#)
- [What is Swing? | Key Concept | Features | Example & Advantage of Swing \(educba.com\)](#)
- [Java Tutorial \(java2s.com\)](#)
- [Java Swing Tutorial: How to Create a GUI Application in Java \(guru99.com\)](#)
- [Introduction to Java Swing - GeeksforGeeks](#)