Java SWING Tutorial: Container, Components and Event Handling

This comprehensive Java Swing video tutorial explains various components of the GUI Swing Framework and related concepts like JPanel, JFrame, JButton, etc:

We use graphical user interfaces (commonly called GUI) to build applications that have a visual interface by making it easy for the user to use the application.

Having a visual interface for an application makes the application easy to navigate, use controls more efficiently, and also it is visually appealing to the user.

Swing is mainly used for creating the GUI for applications

What You Will Learn:

Video Tutorial On Java Swing

What Is Java Swing

Java provides many GUI frameworks that help us in developing a variety of GUI applications. We have seen one in our previous tutorial i.e. Abstract Window Toolkit or AWT. AWT is one of the oldest GUI frameworks in Java and is also platform dependent. Another disadvantage of AWT is its heavyweight components.

In this tutorial, we will discuss yet another GUI framework in Java i.e. “SWING”. The Swing framework in Java is a part of Java Foundation Classes or commonly called JFCs. JFC is an API that is similar to MFCs (Microsoft Foundation Classes) in C++. JFC contains Swing, AWT, and Java2D.

The Swing framework in Java is built on top of the AWT framework and can be used to create GUI applications just like AWT. But unlike AWT, the Swing components are light-weight and are platform-independent.

The Swing framework is written entirely in Java. The Swing framework in Java is provided through the ‘javax.swing’ package. The classes in the javax.swing package begins with the letter ‘J’. So in a javax.swing package, we will have classes like JButton, JFrame, JTextField, JTextArea, etc.

In general, the Swing API has every control defined in javax.swing package that is present in AWT. So swing in a way acts as a replacement of AWT. Also, Swing has various advanced component tabbed panes. Swing API in Java adapts MVC (Model View Controller) Architecture.

- The controller component of the MVC architecture reads input from the user on the view and then these changes are passed to the component data.
- In each Swing component, the view and controller are clubbed together while the model is a separate one. This gives swing a pluggable look and feel feature.

The features of the swing API are summarized below.

1. Swing components are platform-independent.
2. The API is extensible.
3. Swing components are light-weight. The swing components are written in pure Java and also components are rendered using Java code instead of underlying system calls.
4. Swing API provides a set of advanced controls like TabbedPane, Tree, Colorpicker, table controls, etc. which are rich in functionality.
5. The swing controls are highly customizable. This is because the appearance or look-and-feel of the component is independent of internal representation and hence we can customize it in the way we desire.
6. We can simply change the values and thus alter the look-and-feel at runtime.

Java Swing Components

applications.

So what is a component?

A component can be defined as a control that can be represented visually and is usually independent. It has got a specific functionality and is represented as an individual class in Swing API.

For example, class JButton in swing API is a button component and provides the functionality of a button.

One or more components form a group and this group can be placed in a “Container”. A container provides a space in which we can display components and also manage their spacing, layout, etc.

In Java, Containers are divided into two types as shown below:

As seen from the above hierarchy we have Container classes – frame, dialog, Panel, Applet, etc. There are also Component classes derived from the JComponent class of Swing API. Some of the classes that inherit from JComponent are JLabel, JList, JTextBox, etc.

Some of the important classes of Swing API are as follows:

- **JWindow:** The JWindow class of Swing inherits the Window class directly. The JWindow class uses 'BorderLayout' as the default layout.
- **JPanel:** JPanel is a descendent of JComponent class and is on similar lines to AWT class Panel and has 'FlowLayout' as the default layout.
- **JFrame:** JFrame descends from the Frame class. The components added to the Frame are called contents of the Frame.
- **JLabel:** JLabel class is a subclass of the JComponent. It is used to create text labels in the application.
- **JButton:** The push-button functionality in Swing is provided by JButton. We can associate a string, an icon, or both with the JButton object.
- **JTextField:** JTextField class provides a text field in which we can edit a single line of text.

JFrame In Java

A Frame, in general, is a container that can contain other components such as buttons, labels, text fields, etc. A Frame window can contain a title, a border, and also menus, text fields, buttons, and other components. An application should contain a frame so that we can add components inside it.

The Frame in Java Swing is defined in class javax.swing.JFrame. JFrame class inherits

#1) By Extending The JFrame Class

The first approach is creating a new class to construct a Frame. This class inherits from the JFrame class of the javax.swing package.

The following program implements this approach.

```
import javax.swing.*;
class FrameInherited extends JFrame{ //inherit from JFrame class
    JFrame f;
    FrameInherited(){
        JButton b=new JButton("JFrame_Button");//create button
object
        b.setBounds(100,50,150, 40);

        add(b);//add button on frame
        setSize(300,200);
        setLayout(null);
        setVisible(true);
    }
}
public class Main {
    public static void main(String[] args) {
      new FrameInherited(); //create an object of FrameInherited
class
    }
}
```

Output:

#2) By Instantiating The JFrame Class

```
import javax.swing.*;
public class Main {

    public static void main(String[] args) {
        JFrame f=new JFrame("JFrameInstanceExample");//create a
JFrame object

        JButton b=new JButton("JFrameButton");//create instance of
JButton
        b.setBounds(100,50,150, 40);//dimensions of JButton object

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

        f.setSize(300,200);//set frame width = 300 and height = 200
        f.setLayout(null);//no layout manager specified
        f.setVisible(true);//make the frame visible
    }
}
```

Output:

In the above program, we have created a frame from the JFrame class by creating an instance of the JFrame class.

JPanel In Java

A panel is a component that is contained inside a frame window. A frame can have more than one-panel components inside it with each panel component having several other components.

In easier terms, we can use panels to partition the frame. Each panel groups several other components inside it. In other words, we use panels to organize components inside the frame.

The swing API class that implements the panel component is JPanel. JPanel class inherits from JComponent and has FlowLayout as its default layout.

The following program demonstrates the creation of a panel container in a frame using javax.swing package classes.

```
import javax.swing.*;
class JPanelExample {
    JPanelExample(){
        JFrame frame = new JFrame("Panel Example"); //create a f
```

```
object
        b.setBounds(60,50,80,40); //set dimensions for button
        panel.add(b); //add button to the panel
        frame.add(panel); //add panel to frame
        frame.setSize(400,400);
        frame.setLayout(null);
        frame.setVisible(true);
    }

}
public class Main {
    public static void main(String[] args) {
      new JPanelExample(); //create an object of FrameInherited
class
    }
}
```

Output:

Here we have a Frame. Inside the frame, we create a panel. Then inside the panel, we create a button. This way we can use a panel to hold the other components.

JTextArea In Java

TextArea defines an editable text field. It can have multiple lines. The swing class that defines the text area is JTextArea and it inherits the JTextComponent class.

JTextArea class contains 4 constructors that allow us to create a text area with various options.

- **JTextArea ():** Default constructor. Create an empty text area.
- **JTextArea (String s):** Creates a textarea with s as the default value.
- **JTextArea (int row, int column):** Creates a text area with a specified row x column.
- **JTextArea (String s, int row, int column):** Creates a text are2a with specified row x column and default value s.

The following Java program shows an example of the JTextArea component in the swing.

```
import javax.swing.*;
class JTextAreaExample {
    JTextAreaExample(){
        JFrame frame= new JFrame();
        JTextArea t_area=new JTextArea("JTextArea example");
//create object of JTextArea
        t_area.setBounds(10,30, 150,100);  //set its dimensions
        frame.add(t_area);      //add it to the frame
        frame.setSize(200,200);
        frame.setLayout(null);
```

```
}
public class Main {
    public static void main(String[] args) {
      new JTextAreaExample(); //create an object of TextAreaExample
class
    }
}
```

Output:

JButton In Java

A button is a component that is used to create a push button with a name or label on it. In swing, the class that creates a labeled button is JButton. JButton inherits the AbstractButton class. We can associate the ActionListener event to the button to make it take some action when it is pushed.

Let's implement an example program for JButton in Java swings.

```
import javax.swing.*;

public class Main {
    public static void main(String[] args) {

        JFrame frame=new JFrame("JButton Example"); //create JFrame
object
        JButton button=new JButton("Button");        //Create a
JButton object
```

```
        frame.setSize(250,200);
        frame.setLayout(null);
        frame.setVisible(true);
    }
}
```

Output:

JList In Java

A list consists of multiple text items. Users can either select a single item or multiple items at a time. The class that implements the list in swing API is JList. JList is a descendent of the JComponent class.

Given below are the constructors of the JList class.

- **JList ():** Default constructor that creates an empty, read-only list.
- **JList (array[] listItem):** Create a JList which initially contains elements of array listItem.
- **JList (ListModel<array> dataModel):** Creates a list with elements from the specified model dataModel.

A simple demonstration of the JList component is given below.

```
import javax.swing.*;

public class Main {
```

```
DefaultListModel&lt;&gt;();
        colors.addElement("Red");
        colors.addElement("Green");
        colors.addElement("Blue");
        //create JList object and add listModel to it
        JList&lt;String&gt; colorsList = new JList&lt;&gt;(colors);
        colorsList.setBounds(100,100, 75,50);
        frame.add(colorsList);              //add list to the frame
        frame.setSize(400,400);
        frame.setLayout(null);
        frame.setVisible(true);
    }
}
```

Output:

In the above program, we first define a listModel with color entries in it. Then we create a JList object and add the listModel to it. Next, the JList object is added to the frame object which is then displayed.

JComboBox in Java

The JCombobox class shows a list of choices from which a user can select an option. The selected choice is at the top. JComboBox derives from the JComponent class.

- **JComboBox (Object[] items):** This constructor creates a ComboBox having items as elements of the given array items.
- **JComboBox (Vector<?> items):** This constructor reads the elements of the given vector and constructs a ComboBox with these elements as its items.

JComboBox class also provides methods to add/remove items, add ActionListener, ItemListener, etc.

The following example demonstrates the JComboBox implementation in Java.

```
import javax.swing.*;
class ComboBoxExample {
    JFrame frame;
    ComboBoxExample(){
        frame=new JFrame("ComboBox Example");
        //create a string array
        String country[]=
{"India","SriLanka","Singapore","Maldives","SeyChelles"};
        //create a combobox object with given string array
        JComboBox countries=new JComboBox(country);
        countries.setBounds(50, 50,90,20);
        frame.add(countries);       //add it to the frame
        frame.setLayout(null);
        frame.setSize(200,300);
        frame.setVisible(true);
    }
}
public class Main {
    public static void main(String arg[]) {
       new ComboBoxExample();
    }
}
```

JSlider In Java

A slider allows us to select a specific range of values. In Java Swing API, JSlider is the class that is used to implement the slider.

The following are the constructors provided by JSlider class.

- JSlider (): A default constructor that creates a slider having 50 as the initial value and range 0 -100.
- JSlider (int orientation): This constructor creates a slider just like above but with a specified orientation. Orientation value can be either JSlider.HORIZONTAL or JSlider.VERTICAL.
- JSlider (int min, int max): This constructor is used to create a horizontal slider using the given min and max.
- JSlider (int min, int max, int value): This constructor creates a slider that is horizontal with the specified value of min, max, and value.
- JSlider (int orientation, int min, int max, int value): This constructor constructs a slider with specified orientation, min, max, and value.

The following program demonstrates the JSlider in Java with ticks. This program also demonstrates the usage of the methods supported by the JSlider class.

```
import javax.swing.*;
class SliderExample extends JFrame {
    public SliderExample() {
```

```
        slider.setMinorTickSpacing(2);
        slider.setMajorTickSpacing(10);
        slider.setPaintTicks(true);
        slider.setPaintLabels(true);

        JPanel panel = new JPanel();
        panel.add(slider);  //add slider to the panel
        add(panel);
    }

}
public  class Main{
    public static void main(String s[]) {
        SliderExample frame=new SliderExample();
        frame.pack();
        frame.setVisible(true);
    }
}
```

Output:

Event Handling In Java

An event can be defined as a change of state of an object. From the GUI point of view, an event occurs when the end-user interacts with the GUI components. The events that get triggered in the GUI can be the click of a button, scrolling, selecting list items, changing text, etc.

expiration, etc.

Event handling is a mechanism through which an action is taken when an event occurs. For this, we define a method which is also called an event handler that is called when an event occurs. Java uses a standard mechanism called the “Delegation event model” to generate as well as handle events.

The Delegation event model consists of:

#1) Source: The Source of the event is the object. The object on which an event occurs is the source and the source is responsible for sending information about the event to the event handler.

#2) Listener: The listener is nothing but the event handler responsible for taking an action when an event occurs. In Java, a listener is an object that waits on an event. Once the event occurs, the listener processes the event.

The requirement is to register the listener with the object so that when an event occurs, the listener can process it.

For example, for a button click event, we can have the following sequence of steps.

1. The user clicks the button that generates a Click event.
2. The appropriate event class object is created and the source and event data are passed to this object.
3. This event object is then passed to the listener class registered with the object.
4. The listener executes and returns.

Now let’s discuss some of the listeners provided by Java.

ActionListener In Java

notified in the ActionEvent.

The java.awt.an event package defines the ActionListener interface. This interface has only one method actionPerformed ().

public abstract void actionPerformed (ActionEvent e);

When a registered component like a Button is clicked, then the actionPerformed () method is automatically invoked.

The most common approach to include ActionListener in the program is to implement the ActionListener interface and then implement the actionPerformed () method.

The steps to implement ActionListener class are as follows:

#1) Implement the interface ActionListerner.

public class ActionListenerImpl Implements ActionListener

#2) Register the component with this listener. If the button is a component that we want to register with the listener then we will register it as follows:

button.addActionListener (instanceOfListenerclass);

#3) Implement/override the actionPerformed () method.

```
public void actionPerformed (ActionEvent e){
//code to perform action
}
```

So using the above steps, we can associate any event with the GUI component.

```
public class Main {
    public static void main(String[] args) {
        JFrame frame=new JFrame("Button Click Example");
        final JTextField text_field=new JTextField();
//JTextField object
        text_field.setBounds(50,100, 150,20);
        JButton click_button=new JButton("Click Me!!!");
//JButton object
        click_button.setBounds(20,50,75,30);
        click_button.addActionListener(new ActionListener(){
//add an event and take action
            public void actionPerformed(ActionEvent e){
                text_field.setText("You Clicked the button");
            }
        });

//add button and textfield to the frame
        frame.add(click_button);frame.add(text_field);
        frame.setSize(400,400);
        frame.setLayout(null);
        frame.setVisible(true);
    }
}
```

Output:

text field on clicking the button.

KeyListener In Java

Whenever there is a change in the state of the key, a KeyListener is notified. Just like ActionListener, the KeyListener is also found in the java.awt.event package.

KeyListener interface provides the following methods:

public abstract void keyPressed (KeyEvent e);

public abstract void keyReleased(KeyEvent e);

public abstract void keyTyped(KeyEvent e);

We need to implement the above methods to associate the key events with the component. We leave it to the user to implement a KeyListener example using swings in Java.

Swing Layouts In Java

When we arrange various components in a container, then we say we are laying out those components. So a layout can be defined as the positioning of components in a container.

As long as there are fewer components, they can be placed by drag-drop manually. But it becomes difficult to arrange the components large in numbers. At this juncture, the Layout Manager of Java comes to our aid.

LayoutManager is responsible for the components' layout in GUI applications. LayoutManager is an interface and it is implemented by all the layout manager classes. Java provides the following LayoutManager classes.

java.awt.BorderLayout	Components are laid out to fit in five directions namely center, east, west, south, north.
java.awt.FlowLayout	This is the default layout. It lays the components in the directional flow.
java.awt.GridLayout	Arranges the components in a rectangular grid.
javax.swing.BoxLayout	Components are arranged in a box.
java.awt.CardLayout	Each component is viewed as a card in a deck and at a time only one component is visible.
java.awt.GridBagLayout	Arranges components vertically, horizontally, or even along their baselines. Components need not be of the same size.
javax.swing.GroupLayout	Groups the components and then positions them in the container.
javax.swing.ScrollPaneLayout	Used by JScrollPane class and is responsible for arranging components in scrollable containers.
javax.swing.SpringLayout etc.	A set of constraints like the horizontal and vertical distance between components etc. is provided and the components are arranged according to these set of constraints.

In this tutorial, we will only discuss FlowLayout and GridLayout.

FlowLayout In Java

The FlowLayout arranges the components in a flow direction, one after another. This is the default layout for the containers like Panel and Applet.

The FlowLayout class in Java that represents the FlowLayout manager contains the following Fields and constructors.

Fields Of FlowLayout Class

- public static final int RIGHT
- public static final int CENTER

The above fields define the positions at which the components will be placed or aligned.

Constructors Of FlowLayout Class

- **FlowLayout ():** This is a default constructor. This constructor creates a flow layout having centrally aligned components with a default gap of 5 units in the horizontal and vertical direction.
- **FlowLayout (int align):** This constructor creates a flow layout with the specified alignment value and with a horizontal and vertical gap of 5 units.
- **FlowLayout (int align, int hgap, int vgap):** Creates a flow layout with specified alignment value and horizontal and vertical gap.

Given below is an example of FlowLayout in Java.

```
import javax.swing.*;
import java.awt.*;

class FlowLayoutClass {
    JFrame frame;
    FlowLayoutClass() {
        frame = new JFrame("FlowLayout Example");
        //create button components
        JButton b1 = new JButton("A");
        JButton b2 = new JButton("B");
        JButton b3 = new JButton("C");
        JButton b4 = new JButton("D");
        JButton b5 = new JButton("E");
        //add components to the frame
        frame.add(b1);
```

```
        frame.add(b5);
        //set layout as 'FlowLayout.CENTER'
        frame.setLayout(new FlowLayout(FlowLayout.CENTER));
        //setting flow layout of right alignment

        frame.setSize(300, 300);
        frame.setVisible(true);
    }

}
public class Main{
    public static void main(String[] args) {
        new FlowLayoutClass();
    }
}
```

Output:

GridLayout In Java

Using GridLayout we can layout the components in a rectangular grid fashion i.e. each component is arranged in each rectangle.

Constructors Of GridLayout Class

1. **GridLayout ():** default constructor that generates a grid layout having one column per one component in a row.
2. **GridLayout (int rows, int columns): This constructor generates a grid layout** with specified rows and columns. There is no gap between the

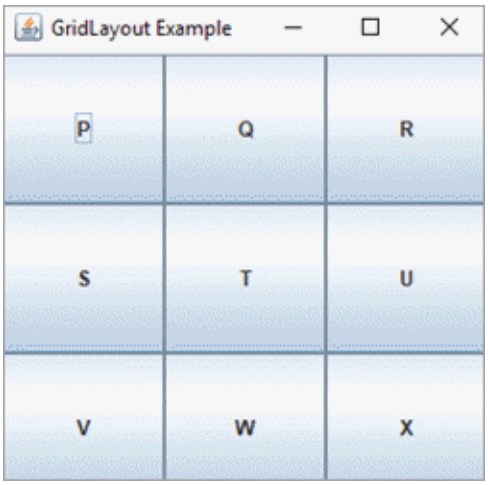
horizontal and vertical gaps.

The following example implements the GridLayout in Java.

```
import javax.swing.*;
import java.awt.*;

class GridLayoutClass {
    JFrame frame;

    GridLayoutClass() {
        frame=new JFrame("GridLayout Example");
        //create components to be laid out as per GridLayout
        JButton b1=new JButton("P");
        JButton b2=new JButton("Q");
        JButton b3=new JButton("R");
        JButton b4=new JButton("S");
        JButton b5=new JButton("T");
        JButton b6=new JButton("U");
        JButton b7=new JButton("V");
        JButton b8=new JButton("W");
        JButton b9=new JButton("X");
        //add components to the frame

frame.add(b1);frame.add(b2);frame.add(b3);frame.add(b4);frame.add(b5);

        frame.add(b6);frame.add(b7);frame.add(b8);frame.add(b9);
        //set frame layout to GridLayout of 3 rows and 3 columns
        frame.setLayout(new GridLayout(3,3));

        frame.setSize(300,300);
        frame.setVisible(true);
```

```
public class Main{
    public static void main(String[] args) {
        new GridLayoutClass();
    }
}
```

Output:



Setbounds In Java

If we check the programming examples in this tutorial before the layout topic, we can see that we have set the layout as null in these examples (setLayout(null)). We have seen that when we use layout managers in our program, they automatically position the components.

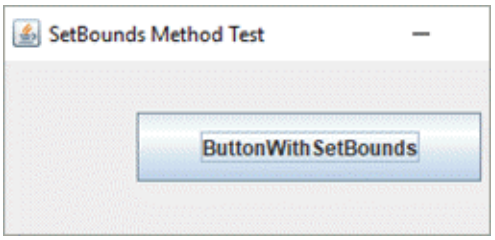
When layout managers are not used, we can use the setBounds method to the size and position of the component. So the method setBounds is used to manually position the component and also set the size.

The general syntax of the setBounds method is as follows:

setBounds (int x-coordinate, int y – coordinate, int width, int height)

```
    public static void main(String arg[]) {
        JFrame frame = new JFrame("SetBounds Method Test");
        frame.setSize(375, 250);
        // Set layout as null
        frame.setLayout(null);
        // Create a Button
        JButton button = new JButton("ButtonWithSetBounds");
        // Set position and size of a button using setBounds
        button.setBounds(80,30,200,40);
        frame.add(button);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setLocationRelativeTo(null);
        frame.setVisible(true);
    }
}
```

Output:



In the above program, we have a Button component. We have not set any layout but we have used the setBounds method to set its position and dimensions.

Swing Vs JavaFX

Swing provides an API to create GUI components.	JavaFX provides scripts and fast UI development associated with screen builder.
There is going to be no new functionality added to Swing in future versions.	JavaFX provides rich functionality and has the potential for more features in future versions.
We can create all standard components using Swing API.	JavaFX allows us to create rich GUI components using advanced look and feel.
A large number of components are present in Swing.	JavaFX has a comparatively lesser number of components.
Swing is a fully-features UI library.	JavaFX is a new and upcoming API with rich UI components.
Swing has loose MVC support.	JavaFX supports MVC pattern consistently.

Frequently Asked Questions

Q #1) Is Swing still used in Java?

Answer: Yes, Swing is still being used in Java and that too heavily. Sometimes it is used as a complete replacement for AWT. Sometimes it is also used along with some of the AWT components. It is even used with the latest JavaFX. So Swing is still used and will be used for a long time to come.

Q #2) How does Java Swing work?

Answer: Swing in Java is written on top of the AWT framework. So the event handling of AWT is inherited by swing completely. Swing also provides a large number of components that we can use to develop efficient GUI applications.

Q #3) Does Swing follow MVC?

Controller and View are clubbed together in UI elements. This clubbing allows the swing to have a pluggable look and feel.

Q #4) Is JavaFX better than Swing?

Answer: Swing has been around for a long time and has more mature IDE support. It also had a very big library of components. JavaFX is comparatively newer and has a small library of components but with more consistent updates and consistent MVC support. Thus it depends on how JavaFX develops further and provides more features.

Q #5) Which is better AWT or Swing?

Answer: Swing is built on top of AWT and provides a rich and large set of UI components when compared to AWT. Swing components also can have their look and feel as against AWT components that take a look and feel of the Operating system.

Swing components are faster than AWT. All these factors make the swing better than AWT.

More About Java Swing

When you create an application, initially you should have a base container and you have to add the required components like buttons and text fields in the container.

And when you click or perform any operation on any field, the event will occur and your code should listen to the events and also handle the event.

Swing Container

A container is a root element for an Application. All the other components are added to that root and it forms a hierarchy.

- JApplet

Container Demo using JFrame:

```
 import java.awt.Color;

import javax.swing.JFrame;
import javax.swing.JPanel;

public class ContainerDemo {

public static void main(String[] args) {

JFrame baseFrame =new JFrame();
baseFrame.setTitle(&amp;quot;Base Container&amp;quot;);
JPanel contentPane=new JPanel();

contentPane.setBackground(Color.pink);
baseFrame.setSize(400, 400);

baseFrame.add(contentPane);

baseFrame.setDefaultCloseOperation(baseFrame.EXIT_ON_CL
OSE);

baseFrame.setVisible(true);
}

}
```

```
import javax.swing.JFrame;
import javax.swing.JPanel;

public class ContainerDemo {

      public static void main(String[] args) {

      JFrame baseFrame =new JFrame();
      baseFrame.setTitle("Base Container");
     JPanel contentPane=new JPanel();
      contentPane.setBackground(Color.pink);
         baseFrame.setSize(400, 400);

            baseFrame.add(contentPane);

      baseFrame.setDefaultCloseOperation(baseFrame.EXIT_O
N_CLOSE);
            baseFrame.setVisible(true);
      }


}
```

Creating JFrame object,and setting tittle

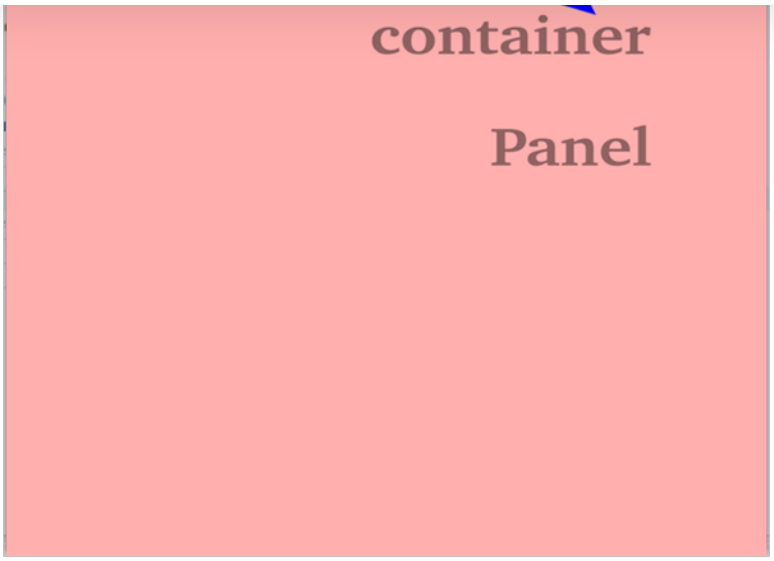
Creating JPanel object And setting pink background color

Setting Frame size

Adding content pane to frame

You need to set visible true for frame to be appeared on the screen

When you run the above program, you will get the below output.



Components

JComponent class is a base class for all the components in a swing.

The frequently used components include,

- JButton
- JTextField
- JTextArea
- JRadioButton
- JComboBox etc.

All these components should be added to the container if not, it will not appear on the application.

Example:

To create the button instance,

```
JButton clickButton=new JButton();
```

To add the button to the container

Event Handling

All the Applications are driven by events like button clicks, mouse clicks, user text input etc. When the event occurs, you have to add a listener and must pass the source event object.

With an inner class, you can handle the event with your logic as shown below.

```
 public class ContainerDemo {

public void createApp() {
JFrame baseFrame =new JFrame();
JPanel contentPane=new JPanel();
baseFrame.setTitle("Base Container");
baseFrame.setSize(400, 400);
baseFrame.add(contentPane);
JButton demoButton =new JButton("click");
demoButton.setBounds(100,95,95,30);
JTextArea result =new JTextArea();
result.setBounds(130,140,95,30);
contentPane.add(demoButton);
contentPane.add(result);

baseFrame.setDefaultCloseOperation(baseFrame.EXIT_ON_CL
OSE);

baseFrame.setVisible(true);
demoButton.addMouseListener(new
MouseAdapter() {
@Override
public void mouseClicked(MouseEvent e) {
result.setText("button clicked");
```

```
public static void main(String[] args) {

ContainerDemo c =new ContainerDemo();
c.createApp();
}
}
```

```
public class ContainerDemo {
public void createApp() {
JFrame baseFrame =new JFrame();
JPanel contentPane=new JPanel();
baseFrame.setTitle("Base Container");
baseFrame.setSize(400, 400);
baseFrame.add(contentPane);
JButton demoButton =new JButton("click");
demoButton.setBounds(100,95,95,30);
JTextArea result =new JTextArea();
result.setBounds(130,140,95,30);
contentPane.add(demoButton);
contentPane.add(result);
        baseFrame.setDefaultCloseOperation(baseFrame.EXIT_ON_CLOSE);
                baseFrame.setVisible(true);
                demoButton.addMouseListener(new MouseAdapter() {
                        @Override
                        public void mouseClicked(MouseEvent e) {

                                result.setText("button clicked");
        }

                });

        }

public static void main(String[] args) {

        ContainerDemo c =new ContainerDemo();
        c.createApp();

}

}
```

Creating JFrame JPanel objects, setting size and tittle.

Creating button named "click" and setting boundaries.

Creating Text Area , setting boundaries and the button and text area in content pane

Handling the listener and setting the text area to "button clicked "string when the button click is happened.

Here, we are creating the instance of the class to the method.

Conclusion

components and their implementation.

We have also discussed event handling in Swing. Although the event handling mechanism is of AWT, swing implements the events in an efficient manner. Then we discussed the various layout managers provided by Swing API that allow us to layout or arrange various components in the Swing GUI applications.

=> **Check Out The Perfect Java Training Guide Here.**

Recommended Reading

- Java Reflection Tutorial With Examples
- JAVA Tutorial For Beginners: 100+ Hands-on Java Video Tutorials
- Java Collections Framework (JCF) Tutorial
- Access Modifiers In Java - Tutorial With Examples
- Java String Data Type With String Buffer And String Builder
- Introduction To Java Programming Language - Video Tutorial