UNIT 3: Java Swing

Create a Swing Applet

The second type of program that commonly uses Swing is the applet. Swing-based applets are similar to AWT-based applets, but with an important difference: A Swing applet extends **JApplet** rather than **Applet**. **JApplet** is derived from **Applet**. Thus, **JApplet** includes all of the functionality found in **Applet** and adds support for Swing. **JApplet** is a top-level Swing container, which means that it is *not* derived from **JComponent**. Because **JApplet** is a top-level container, it includes the various panes described earlier. This means that all components are added to **JApplet**'s content pane in the same way that components are added to **JFrame**'s content pane.

Swing applets use the same four life-cycle methods as described in Chapter 23: **init( )**, **start( )**, **stop( )**, and **destroy( )**. Of course, you need override only those methods that are needed by your applet. Painting is accomplished differently in Swing than it is in the AWT, and a Swing applet will not normally override the **paint( )** method. (Painting in Swing is described later in this chapter.)

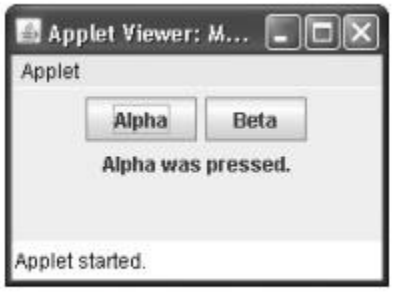


Figure 31-3 Output from the example Swing applet

One other point: All interaction with components in a Swing applet must take place on the event dispatching thread, as described in the previous section. This threading issue applies to all Swing programs.

Here is an example of a Swing applet. It provides the same functionality as the previous application, but does so in applet form. Figure 31-3 shows the program when executed by **appletviewer**.

```
// A simple Swing-based applet

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

/*

This HTML can be used to launch the applet:

<applet code="MySwingApplet" width=220 height=90> </applet>

*/

public class MySwingApplet extends JApplet { JButton jbtnAlpha;

JButton jbtnBeta;

JLabel jlab;

// Initialize the applet.
public void init() {
```

```
try {

SwingUtilities.invokeAndWait(new Runnable () { public void run() {

makeGUI(); // initialize the GUI

}

});

} catch(Exception exc) {

System.out.println("Can’t create because of "+ exc);

}

}
  //This applet does not need to override start(), stop(),

  //or destroy().

  //Set up and initialize the GUI.

private void makeGUI() {

  //Set the applet to use flow layout.
```

```
  setLayout(new FlowLayout());

  //Make two buttons.

jbtnAlpha = new JButton("Alpha"); jbtnBeta = new JButton("Beta");


  Add action listener for Alpha. jbtnAlpha.addActionListener(new ActionListener() {

public void actionPerformed(ActionEvent le) { jlab.setText("Alpha was pressed.");


}


});

  //Add action listener for Beta.
  jbtnBeta.addActionListener(new ActionListener() {

public void actionPerformed(ActionEvent le) { jlab.setText("Beta was pressed.");


}


});


  //Add the buttons to the content pane.
  add(jbtnAlpha);

add(jbtnBeta);

  //Create a text-based label.
```

```
jlab = new JLabel("Press a button.");


// Add the label to the content pane.
add(jlab);


}


}
```

There are two important things to notice about this applet. First, **MySwingApplet** extends **JApplet.** As explained, all Swing-based applets extend **JApplet** rather than **Applet**. Second, the **init( )** method initializes the Swing components on the event dispatching thread by setting up a call to **makeGUI( )**. Notice that this is accomplished through the use of **invokeAndWait( )** rather than **invokeLater( )**. Applets must use **invokeAndWait( )** because the **init( )** method must not return until the entire initialization process has been completed. In essence, the **start( )** method cannot be called until after initialization, which means that the GUI must be fully constructed.

Inside **makeGUI( )**, the two buttons and label are created, and the action listeners are added to the buttons. Finally, the components are added to the content pane. Although this example is quite simple, this same general approach must be used when building any Swing GUI that will be used by an applet.

Java JLayeredPane

The JLayeredPane class is used to add depth to swing container. It is used to provide a third dimension for positioning component and divide the depth-range into several different layers.

JLayeredPane class declaration

1. **public class** JLayeredPane **extends** JComponent **implements** Accessible

Commonly used Constructors:

Constructor	Description
JLayeredPane	It is used to create a new JLayeredPane

Commonly used Methods:

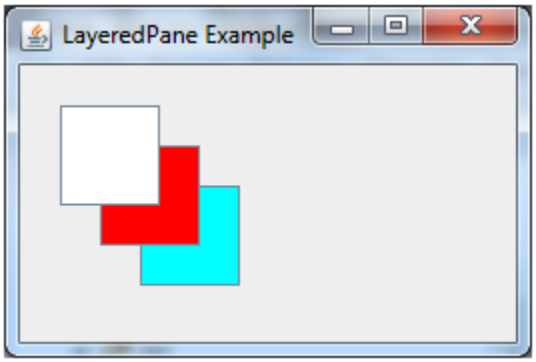
Method	Description
int getIndexOf(Component c)	It is used to return the index of the specified Component.
int getLayer(Component c)	It is used to return the layer attribute for the specified Component.
int getPosition(Component c)	It is used to return the relative position of the component within its laye

Java JLayeredPane Example

```
1. import javax.swing.*;
2. import java.awt.*;
3. public class LayeredPaneExample extends JFrame {
4.   public LayeredPaneExample() {
5.     super("LayeredPane Example");
6.     setSize(200, 200);
7.     JLayeredPane pane = getLayeredPane();
8.     //creating buttons
9.     JButton top = new JButton();
10.    top.setBackground(Color.white);
11.    top.setBounds(20, 20, 50, 50);
12.    JButton middle = new JButton();
13.    middle.setBackground(Color.red);
14.    middle.setBounds(40, 40, 50, 50);
15.    JButton bottom = new JButton();
```

```
16.     bottom.setBackground(Color.cyan);
17.     bottom.setBounds(60, 60, 50, 50);
18.     //adding buttons on pane
19.     pane.add(bottom, new Integer(1));
20.     pane.add(middle, new Integer(2));
21.     pane.add(top, new Integer(3));
22.   }
23.   public static void main(String[] args) {
24.       LayeredPaneExample panel = new  LayeredPaneExample();
25.       panel.setVisible(true);
26.   }
27. }
```

Output:



Swing's pluggable look and feel

The idea of a "pluggable look and feel," or PLaF for short, plays a prominent role in the Swing set of widgets (user-interface components), in JavaSoft's Java Foundation Classes (JFC) release in JDK 1.2. A "look and feel" (LaF) refer to components' appearance (grey vs. white, flat vs. three-dimensional, etc.) and their interactions with users (how they react to keypresses, mouseclicks and the features they offer, such as floating toolbars). A PLaF means that you can write your program using a particular look-and-feel, for example the Windows 95 LaF, and then at runtime you can switch to another LaF, such as Motif, or the new "Organic" LaF (developed specifically for use in Java), or even a pen-based LaF for palm computers. This saves developers the trouble of changing user-interface code.

However, this advantage has some ramifications that you need to be aware of. In particular, you can't always assume that a particular LaF offers the features that you want. An example (prompted by a question in the JavaGUI section of developer.com's Discussions area) is found in the dockable toolbar and, in particular, the ability to programatically control whether the toolbar is docked or floating. As of Swing 0.7.1, the JToolBar component (the Swing toolbar) does not offer methods to support the control of a dockable toolbar, even though the Windows LaF and the Java LaF both support dockable toolbars (alas, the dockable feature is somewhat buggy in this release of Swing).

So what to do? The answer is to use the code included with this article, which shows how to access methods in the underlying user interface (i.e., the specific LaF being used). This demo application creates a window with three buttons (ok, four, but the first one is just for show): float, unfloat and exit. When you click on the Float button, the toolbar floats in its own top-level window that you can drag around the screen. When you click on the Unfloat button, the toolbar returns to its home frame. I'll leave the effect of the Exit button to your imagination.

These tasks are accomplished with the

```
actionPerformed
```

method (this method gets called when you click on the button):
First, we get a reference to the underlying user-interface object, in this case a

```
ToolBarUI
```

object:

```
ToolBarUI tbUI = m_tb.getUI(); // m_tb is a reference to our JToolBar instance
```

Then we see if this particular UI object is a

`BasicToolBarUI`

, because we know that

`BasicToolBarUI`

s support floating toolbars:

```
if ( tbUI instanceof BasicToolBarUI ) {
```

Then, if it *is* a

`BasicToolBarUI`

, we can access the

`BasicToolBarUI`

‘s methods for controlling the floating aspect of the toolbar. First, we cast the tbUI reference to

`BasicToolBarUI`

so we can invoke the

`setFloatingLocation()`

method (we’ll float the toolbar at coordinates 50,50 on the screen):

```
( ( BasicToolBarUI ) tbUI).setFloatingLocation( 50, 50 );
```

Then we’ll actually force the toolbar to float by invoking

`setFloating()`

, again casting to a

`BasicToolBarUI`

:

```
( ( BasicToolBarUI ) tbUI).setFloating( true, null );
```

And, since there are some painting bugs in Swing, we force some repaints:

```
m_tb.repaint();

repaint();
```

To Unfloat, we simply use the same method to invoke

`setFloating()`

with a false parameter to stop floating.
And that’s it! Keep in mind that this fix is specific to

`BasicToolBarUI`

types. We could check for other types, but any other LaF implementation would be specific to that LaF.
There are other

`BasicToolBarUI`

-specific methods that we can invoke in the same manner. Take a look at the Swing documentation in the com.java.swing.basic package for more.

Note that JavaSoft may change the JToolBar implementation and “pull” the floating functionality from the underlying UI, thereby forcing all LaFs to support floating toolbars, which would make this example moot. However, there will always be times when you need to access specific features of the underlying LaF user-interface implementation.

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.

JLabel class declaration

Let's see the declaration for javax.swing.JLabel class.

1. **public class** JLabel **extends** JComponent **implements** SwingConstants, Accessible

Commonly used Constructors:

Constructor	Description
JLabel()	Creates a JLabel instance with no image and with an empty str
JLabel(String s)	Creates a JLabel instance with the specified text.
JLabel(Icon i)	Creates a JLabel instance with the specified image.
JLabel(String s, Icon i, int horizontalAlignment)	Creates a JLabel instance with the specified text, image, alignment.

Commonly used Methods:

Methods	Description
String getText()	t returns the text string that a label displays.

void setText(String text)	It defines the single line of text this component will displa
void setHorizontalAlignment(int alignment)	It sets the alignment of the label's contents along the X a
Icon getIcon()	It returns the graphic image that the label displays.
int getHorizontalAlignment()	It returns the alignment of the label's contents along the

Java JLabel Example

```
1. import javax.swing.*;
2. class LabelExample
3. {
4. public static void main(String args[])
5.     {
6.     JFrame f= new JFrame("Label Example");
7.     JLabel l1,l2;
8.     l1=new JLabel("First Label.");
9.     l1.setBounds(50,50, 100,30);
10.    l2=new JLabel("Second Label.");
11.    l2.setBounds(50,100, 100,30);
12.    f.add(l1); f.add(l2);
13.    f.setSize(300,300);
14.    f.setLayout(null);
15.    f.setVisible(true);
16.    }
17.    }
```

Output:

Java JLabel Example with ActionListener

```
1. import javax.swing.*;
2. import java.awt.*;
3. import java.awt.event.*;
4. public class LabelExample extends Frame implements ActionListener{
5.     JTextField tf; JLabel l; JButton b;
6.     LabelExample(){
7.         tf=new JTextField();
8.         tf.setBounds(50,50, 150,20);
9.         l=new JLabel();
10.        l.setBounds(50,100, 250,20);
11.        b=new JButton("Find IP");
12.        b.setBounds(50,150,95,30);
13.        b.addActionListener(this);
14.        add(b);add(tf);add(l);
15.        setSize(400,400);
16.        setLayout(null);
17.        setVisible(true);
18.    }
19.    public void actionPerformed(ActionEvent e) {
20.        try{
21.        String host=tf.getText();
```

```
22.         String ip=java.net.InetAddress.getByName(host).getHostAddress();
23.         l.setText("IP of "+host+" is: "+ip);
24.         }catch(Exception ex){System.out.println(ex);}
25.     }
26.     public static void main(String[] args) {
27.         new LabelExample();
28.     } }
```

Output:

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.

JTextField class declaration

Let's see the declaration for javax.swing.JTextField class.

```
1. public class JTextField extends JTextComponent implements SwingConstants
```

Commonly used Constructors:

Constructor	Description

JTextField()	Creates a new TextField
JTextField(String text)	Creates a new TextField initialized with the specified text.
JTextField(String text, int columns)	Creates a new TextField initialized with the specified text and colum
JTextField(int columns)	Creates a new empty TextField with the specified number of colum

Commonly used Methods:

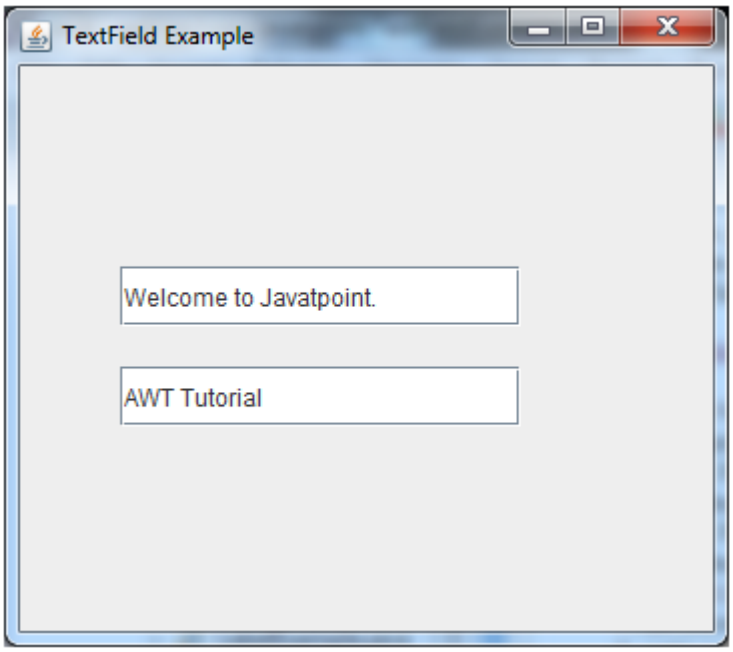
Methods	Description
void addActionListener(ActionListener l)	It is used to add the specified action listener to receive action textfield.
Action getAction()	It returns the currently set Action for this ActionEvent source, or n set.
void setFont(Font f)	It is used to set the current font.
void removeActionListener(ActionListener l)	It is used to remove the specified action listener so that it no longe events from this textfield.

Java JTextField Example

```
1. import javax.swing.*;
2. class TextFieldExample
3. {
4. public static void main(String args[])
5.     {
6.     JFrame f= new JFrame("TextField Example");
7.     JTextField t1,t2;
8.     t1=new JTextField("Welcome to Javatpoint.");
9.     t1.setBounds(50,100, 200,30);
```

```
10.     t2=new JTextField("AWT Tutorial");
11.     t2.setBounds(50,150, 200,30);
12.     f.add(t1); f.add(t2);
13.     f.setSize(400,400);
14.     f.setLayout(null);
15.     f.setVisible(true);
16.     }
17.     }
```

Output:

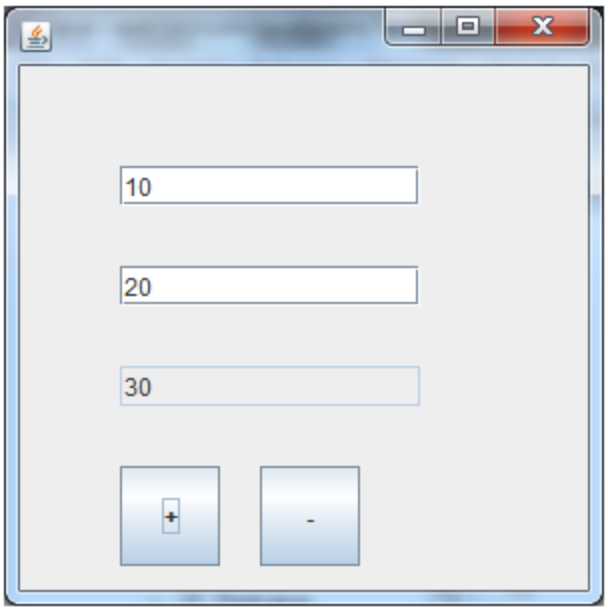


Java JTextField Example with ActionListener

```
1.  import javax.swing.*;
2.  import java.awt.event.*;
3.  public class TextFieldExample implements ActionListener{
4.      JTextField tf1,tf2,tf3;
5.      JButton b1,b2;
6.      TextFieldExample(){
7.          JFrame f= new JFrame();
8.          tf1=new JTextField();
```

```
9.          tf1.setBounds(50,50,150,20);
10.         tf2=new JTextField();
11.         tf2.setBounds(50,100,150,20);
12.         tf3=new JTextField();
13.         tf3.setBounds(50,150,150,20);
14.         tf3.setEditable(false);
15.         b1=new JButton("+");
16.         b1.setBounds(50,200,50,50);
17.         b2=new JButton("-");
18.         b2.setBounds(120,200,50,50);
19.         b1.addActionListener(this);
20.         b2.addActionListener(this);
21.         f.add(tf1);f.add(tf2);f.add(tf3);f.add(b1);f.add(b2);
22.         f.setSize(300,300);
23.         f.setLayout(null);
24.         f.setVisible(true);
25.     }
26.     public void actionPerformed(ActionEvent e) {
27.         String s1=tf1.getText();
28.         String s2=tf2.getText();
29.         int a=Integer.parseInt(s1);
30.         int b=Integer.parseInt(s2);
31.         int c=0;
32.         if(e.getSource()==b1){
33.             c=a+b;
34.         }else if(e.getSource()==b2){
35.             c=a-b;
36.         }
37.         String result=String.valueOf(c);
38.         tf3.setText(result);
39.     }
40. public static void main(String[] args) {
41.     new TextFieldExample();
42. } }
```

Output:



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.

JButton class declaration

Let's see the declaration for javax.swing.JButton class.

1. **public class** JButton **extends** AbstractButton **implements** Accessible

Commonly used Constructors:

Constructor	Description
JButton()	It creates a button with no text and icon.
JButton(String s)	It creates a button with the specified text.
JButton(Icon i)	It creates a button with the specified icon object.

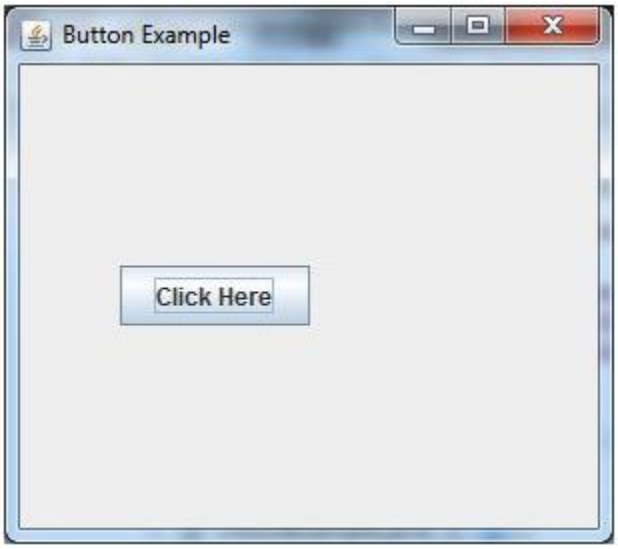
Commonly used Methods of AbstractButton class:

Methods	Description
void setText(String s)	It is used to set specified text on button
String getText()	It is used to return the text of the button.
void setEnabled(boolean b)	It is used to enable or disable the button.
void setIcon(Icon b)	It is used to set the specified Icon on the button.
Icon getIcon()	It is used to get the Icon of the button.
void setMnemonic(int a)	It is used to set the mnemonic on the button.
void addActionListener(ActionListener a)	It is used to add the action listener to this object.

Java JButton Example

```
1. import javax.swing.*;
2. public class ButtonExample {
3. public static void main(String[] args) {
4.     JFrame f=new JFrame("Button Example");
5.     JButton b=new JButton("Click Here");
6.     b.setBounds(50,100,95,30);
7.     f.add(b);
8.     f.setSize(400,400);
9.     f.setLayout(null);
10.    f.setVisible(true);
11. }
12. }
```

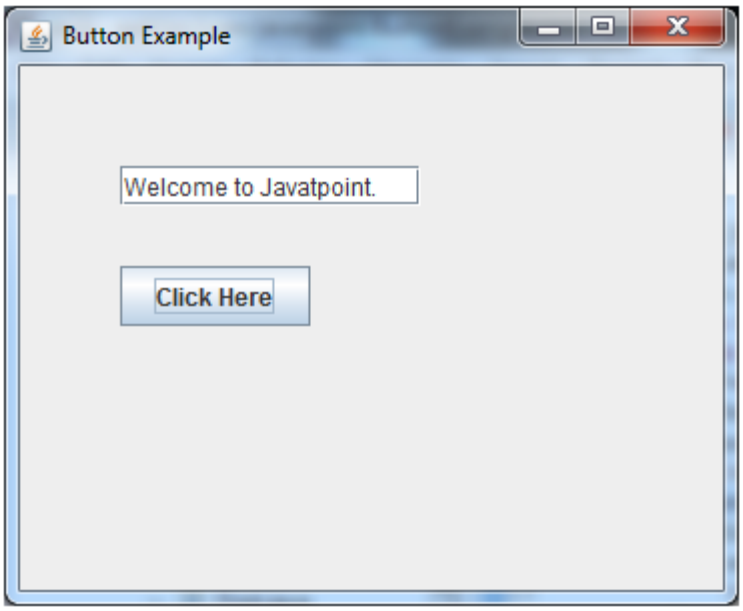
Output:



Java JButton Example with ActionListener

```
1. import java.awt.event.*;
2. import javax.swing.*;
3. public class ButtonExample {
4. public static void main(String[] args) {
5.     JFrame f=new JFrame("Button Example");
6.     final JTextField tf=new JTextField();
7.     tf.setBounds(50,50, 150,20);
8.     JButton b=new JButton("Click Here");
9.     b.setBounds(50,100,95,30);
10.    b.addActionListener(new ActionListener(){
11. public void actionPerformed(ActionEvent e){
12.             tf.setText("Welcome to Javatpoint.");
13.         }
14.    });
15.    f.add(b);f.add(tf);
16.    f.setSize(400,400);
17.    f.setLayout(null);
18.    f.setVisible(true);
19. }
20. }
```

Output:



Example of displaying image on the button:

```
1. import javax.swing.*;
2. public class ButtonExample{
3. ButtonExample(){
4. JFrame f=new JFrame("Button Example");
5. JButton b=new JButton(new ImageIcon("D:\\icon.png"));
6. b.setBounds(100,100,100, 40);
7. f.add(b);
8. f.setSize(300,400);
9. f.setLayout(null);
10. f.setVisible(true);
11. f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
12.     }
13. public static void main(String[] args) {
14.     new ButtonExample();
15. }
16. }
```

Output:

Java JToggleButton

JToggleButton is used to create toggle button, it is two-states button to switch on or off.

Nested Classes

Modifier and Type	Class	Description
protected class	JToggleButton.AccessibleJToggleButton	This class implements accessibility su JToggleButton class.
static class	JToggleButton.ToggleButtonModel	The ToggleButton model

Constructors

Constructor	Description
JToggleButton()	It creates an initially unselected toggle button without se image.
JToggleButton(Action a)	It creates a toggle button where properties are taken from the

JToggleButton(Icon icon)	It creates an initially unselected toggle button with the specif text.
JToggleButton(Icon icon, boolean selected)	It creates a toggle button with the specified image and selec text.
JToggleButton(String text)	It creates an unselected toggle button with the specified text
JToggleButton(String text, boolean selected)	It creates a toggle button with the specified text and selectio
JToggleButton(String text, Icon icon)	It creates a toggle button that has the specified text and in initially unselected.
JToggleButton(String text, Icon icon, boolean selected)	It creates a toggle button with the specified text, image, and

Methods

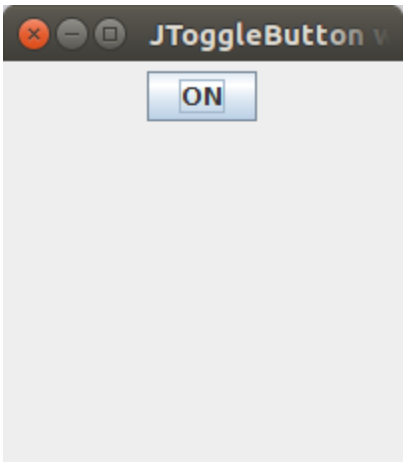
Modifier and Type	Method	Description
AccessibleContext	getAccessibleContext()	It gets the AccessibleContext associated with this JToggleButt
String	getUIClassID()	It returns a string that specifies the name of the l&f class component.
protected String	paramString()	It returns a string representation of this JToggleButton.
void	updateUI()	It resets the UI property to a value from the current look and

JToggleButton Example

```
1. import java.awt.FlowLayout;
2. import java.awt.event.ItemEvent;
3. import java.awt.event.ItemListener;
4. import javax.swing.JFrame;
```

```
5.  import javax.swing.JToggleButton;
6.
7.  public class JToggleButtonExample extends JFrame implements ItemListener {
8.      public static void main(String[] args) {
9.          new JToggleButtonExample();
10.     }
11.     private JToggleButton button;
12.     JToggleButtonExample() {
13.         setTitle("JToggleButton with ItemListener Example");
14.         setLayout(new FlowLayout());
15.         setJToggleButton();
16.         setAction();
17.         setSize(200, 200);
18.         setVisible(true);
19.         setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
20.     }
21.     private void setJToggleButton() {
22.         button = new JToggleButton("ON");
23.         add(button);
24.     }
25.     private void setAction() {
26.         button.addItemListener(this);
27.     }
28.     public void itemStateChanged(ItemEvent eve) {
29.         if (button.isSelected())
30.             button.setText("OFF");
31.         else
32.             button.setText("ON");
33.     }
34. }
```

Output



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.

JCheckBox class declaration

Let's see the declaration for javax.swing.JCheckBox class.

1. **public class** JCheckBox **extends** JToggleButton **implements** Accessible

Commonly used Constructors:

Constructor	Description
JJCheckBox()	Creates an initially unselected check box button with no text, no ic
JChechBox(String s)	Creates an initially unselected check box with text.
JCheckBox(String text, boolean selected)	Creates a check box with text and specifies whether or not it is init
JCheckBox(Action a)	Creates a check box where properties are taken from the Action su

Commonly used Methods:

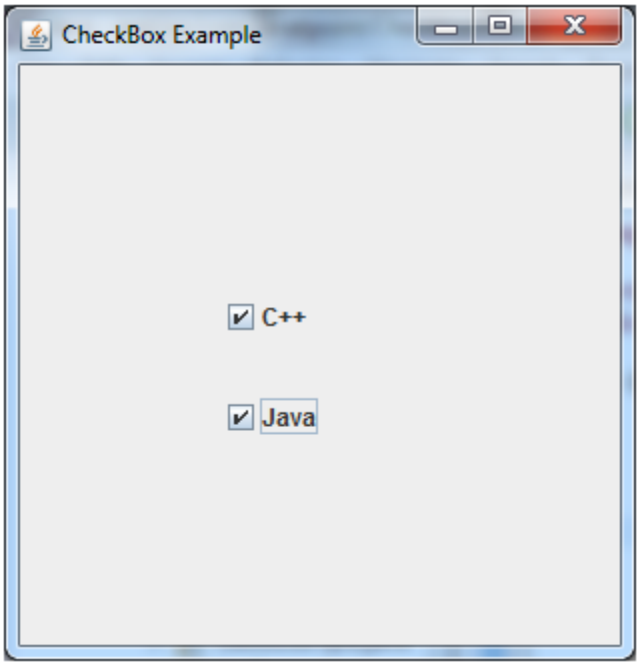
Methods	Description

AccessibleContext getAccessibleContext()	It is used to get the AccessibleContext associated with this JC
protected String paramString()	It returns a string representation of this JCheckBox.

Java JCheckBox Example

```
1. import javax.swing.*;
2. public class CheckBoxExample
3. {
4.     CheckBoxExample(){
5.       JFrame f= new JFrame("CheckBox Example");
6.       JCheckBox checkBox1 = new JCheckBox("C++");
7.       checkBox1.setBounds(100,100, 50,50);
8.       JCheckBox checkBox2 = new JCheckBox("Java", true);
9.       checkBox2.setBounds(100,150, 50,50);
10.      f.add(checkBox1);
11.      f.add(checkBox2);
12.      f.setSize(400,400);
13.      f.setLayout(null);
14.      f.setVisible(true);
15.   }
16. public static void main(String args[])
17.   {
18.   new CheckBoxExample();
19.   }}
```

Output:

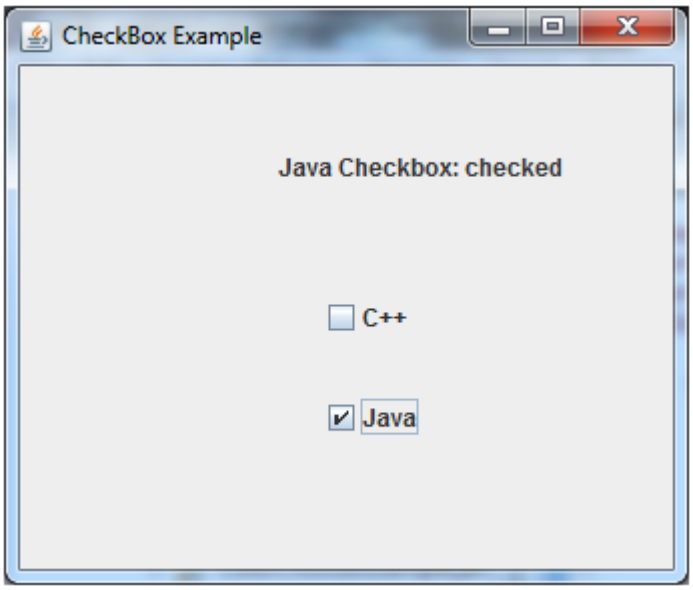


Java JCheckBox Example with ItemListener

```
1. import javax.swing.*;
2. import java.awt.event.*;
3. public class CheckBoxExample
4. {
5.     CheckBoxExample(){
6.       JFrame f= new JFrame("CheckBox Example");
7.       final JLabel label = new JLabel();
8.       label.setHorizontalAlignment(JLabel.CENTER);
9.       label.setSize(400,100);
10.      JCheckBox checkbox1 = new JCheckBox("C++");
11.      checkbox1.setBounds(150,100, 50,50);
12.      JCheckBox checkbox2 = new JCheckBox("Java");
13.      checkbox2.setBounds(150,150, 50,50);
14.      f.add(checkbox1); f.add(checkbox2); f.add(label);
15.      checkbox1.addItemListener(new ItemListener() {
16.          public void itemStateChanged(ItemEvent e) {
17.            label.setText("C++ Checkbox: "
18.            + (e.getStateChange()==1?"checked":"unchecked"));
```

```
19.             }
20.           });
21.         checkbox2.addItemListener(new ItemListener() {
22.             public void itemStateChanged(ItemEvent e) {
23.                 label.setText("Java Checkbox: "
24.                 + (e.getStateChange()==1?"checked":"unchecked"));
25.             }
26.           });
27.         f.setSize(400,400);
28.         f.setLayout(null);
29.         f.setVisible(true);
30.     }
31. public static void main(String args[])
32. {
33.     new CheckBoxExample();
34. }
35. }
```

Output:



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.

JRadioButton class declaration

Let's see the declaration for javax.swing.JRadioButton class.

1. **public class** JRadioButton **extends** JToggleButton **implements** Accessible

Commonly used Constructors:

Constructor	Description
JRadioButton()	Creates an unselected radio button with no text.
JRadioButton(String s)	Creates an unselected radio button with specified text.
JRadioButton(String s, boolean selected)	Creates a radio button with the specified text and selected s

Commonly used Methods:

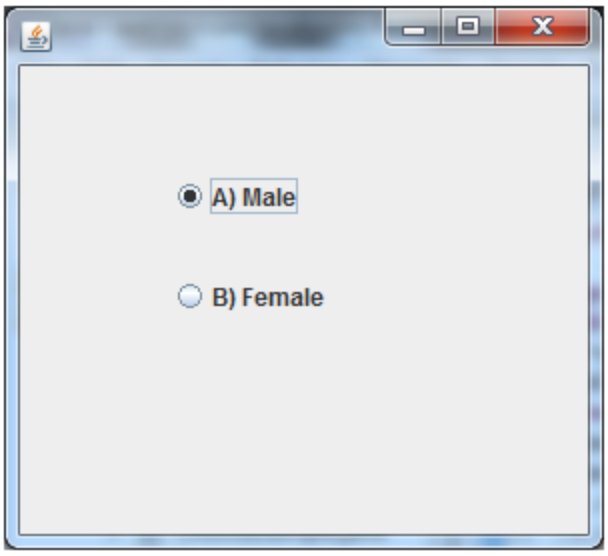
Methods	Description
void setText(String s)	It is used to set specified text on button.
String getText()	It is used to return the text of the button.
void setEnabled(boolean b)	It is used to enable or disable the button.
void setIcon(Icon b)	It is used to set the specified Icon on the button.
Icon getIcon()	It is used to get the Icon of the button.
void setMnemonic(int a)	It is used to set the mnemonic on the button.

void addActionListener(ActionListener a)	It is used to add the action listener to this object.

Java JRadioButton Example

```
1. import javax.swing.*;
2. public class RadioButtonExample {
3. JFrame f;
4. RadioButtonExample(){
5. f=new JFrame();
6. JRadioButton r1=new JRadioButton("A) Male");
7. JRadioButton r2=new JRadioButton("B) Female");
8. r1.setBounds(75,50,100,30);
9. r2.setBounds(75,100,100,30);
10. ButtonGroup bg=new ButtonGroup();
11. bg.add(r1);bg.add(r2);
12. f.add(r1);f.add(r2);
13. f.setSize(300,300);
14. f.setLayout(null);
15. f.setVisible(true);
16. }
17. public static void main(String[] args) {
18.     new RadioButtonExample();
19. }
20. }
```

Output:



Java JRadioButton Example with ActionListener

```
1. import javax.swing.*;
2. import java.awt.event.*;
3. class RadioButtonExample extends JFrame implements ActionListener{
4. JRadioButton rb1,rb2;
5. JButton b;
6. RadioButtonExample(){
7. rb1=new JRadioButton("Male");
8. rb1.setBounds(100,50,100,30);
9. rb2=new JRadioButton("Female");
10. rb2.setBounds(100,100,100,30);
11. ButtonGroup bg=new ButtonGroup();
12. bg.add(rb1);bg.add(rb2);
13. b=new JButton("click");
14. b.setBounds(100,150,80,30);
15. b.addActionListener(this);
16. add(rb1);add(rb2);add(b);
17. setSize(300,300);
18. setLayout(null);
19. setVisible(true);
20. }
```

```
21. public void actionPerformed(ActionEvent e){
22. if(rb1.isSelected()){
23. JOptionPane.showMessageDialog(this,"You are Male.");
24. }
25. if(rb2.isSelected()){
26. JOptionPane.showMessageDialog(this,"You are Female.");
27. }
28. }
29. public static void main(String args[]){
30. new RadioButtonExample();
31. }}
```

Output:

Java JViewport

The JViewport class is used to implement scrolling. JViewport is designed to support both logical scrolling and pixel-based scrolling. The viewport's child, called the view, is scrolled by calling the JViewport.setViewPosition() method.

Nested Classes

Modifier and Type	Class	Description
protected class	JViewport.AccessibleJViewport	This class implements accessibility support for the J
protected class	JViewport.ViewListener	A listener for the view.

Fields

Modifier and Type	Field	Description
static int	BACKINGSTORE_SCROLL_MODE	It draws viewport contents into an offscreen image.
protected Image	backingStoreImage	The view image used for a backing store.
static int	BLIT_SCROLL_MODE	It uses graphics.copyArea to implement scrolling.
protected boolean	isViewSizeSet	True when the viewport dimensions have been determin
protected Point	lastPaintPosition	The last viewPosition that we've painted, so we know h backing store image is valid.
protected boolean	scrollUnderway	The scrollUnderway flag is used for components like JLis
static int	SIMPLE_SCROLL_MODE	This mode uses the very simple method of redrawing th of the scrollpane each time it is scrolled.

Constructor

Constructor	Description
JViewport()	Creates a JViewport.

Methods

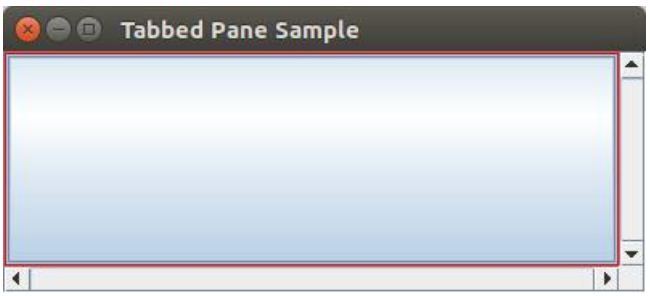
Modifier and Type	Method	Description
void	addChangeListener(ChangeListener l)	It adds a ChangeListener to the list that is n the view's size, position, or the viewport' changed.
protected LayoutManager	createLayoutManager()	Subclassers can override this to install a manager (or null) in the constructor.
protected Jviewport.ViewListener	createViewListener()	It creates a listener for the view.
int	getScrollMode()	It returns the current scrolling mode.
Component	getView()	It returns the JViewport's one child or null.
Point	getViewPosition()	It returns the view coordinates that appear hand corner of the viewport, or 0,0 if there
Dimension	getViewSize()	If the view's size hasn't been explicitly preferred size, otherwise return the view's
void	setExtentSize(Dimension newExtent)	It sets the size of the visible part of the coordinates.
void	setScrollMode(int mode)	It used to control the method of scrolli contents.
void	setViewSize(Dimension newSize)	It sets the size of the view.

JViewport Example

```
1. import java.awt.BorderLayout;
2. import java.awt.Color;
```

```
3.  import java.awt.Dimension;
4.  import javax.swing.JButton;
5.  import javax.swing.JFrame;
6.  import javax.swing.JLabel;
7.  import javax.swing.JScrollPane;
8.  import javax.swing.border.LineBorder;
9.  public class ViewPortClass2 {
10.     public static void main(String[] args) {
11.         JFrame frame = new JFrame("Tabbed Pane Sample");
12.         frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
13.
14.         JLabel label = new JLabel("Label");
15.         label.setPreferredSize(new Dimension(1000, 1000));
16.         JScrollPane jScrollPane = new JScrollPane(label);
17.
18.         JButton jButton1 = new JButton();
19.         jScrollPane.setHorizontalScrollBarPolicy(JScrollPane.HORIZONTAL_SCROLLBAR_ALWAYS);
20.         jScrollPane.setVerticalScrollBarPolicy(JScrollPane.VERTICAL_SCROLLBAR_ALWAYS);
21.         jScrollPane.setViewportBorder(new LineBorder(Color.RED));
22.         jScrollPane.getViewport().add(jButton1, null);
23.
24.         frame.add(jScrollPane, BorderLayout.CENTER);
25.         frame.setSize(400, 150);
26.         frame.setVisible(true);
27.     }
28. }
```

Output:



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.

Constructors

Constructor	Purpose
JScrollPane()	It creates a scroll pane. The Component parameter, when present, sets the scroll two int parameters, when present, set the vertical and horizontal scroll bar polici
JScrollPane(Component)	
JScrollPane(int, int)	
JScrollPane(Component, int, int)	

Useful Methods

Modifier	Method	Description
void	setColumnHeaderView(Component)	It sets the column header for the scroll pane.
void	setRowHeaderView(Component)	It sets the row header for the scroll pane.
void	setCorner(String, Component)	It sets or gets the specified corner. The int parameter speci and must be one of the following constants defined in Scro UPPER_LEFT_CORNER, UPPER_RIGHT_CORNER, LOWE LOWER_RIGHT_CORNER, LOWER_LE LOWER_TRAILING_CORNER, UPPER_LE UPPER_TRAILING_CORNER.
Component	getCorner(String)	
void	setViewportView(Component)	Set the scroll pane's client.

JScrollPane Example

```
1.  import java.awt.FlowLayout;
2.  import javax.swing.JFrame;
3.  import javax.swing.JScrollPane;
4.  import javax.swing.JtextArea;
5.
6.  public class JScrollPaneExample {
7.      private static final long serialVersionUID = 1L;
8.
9.      private static void createAndShowGUI() {
10.
11.         // Create and set up the window.
12.         final JFrame frame = new JFrame("Scroll Pane Example");
13.
14.         // Display the window.
15.         frame.setSize(500, 500);
16.         frame.setVisible(true);
17.         frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
18.
19.         // set flow layout for the frame
20.         frame.getContentPane().setLayout(new FlowLayout());
21.
22.         JTextArea textArea = new JTextArea(20, 20);
23.         JScrollPane scrollableTextArea = new JScrollPane(textArea);
24.
25.         scrollableTextArea.setHorizontalScrollBarPolicy(JScrollPane.HORIZONTAL_SCROLLBAR_ALW
    AYS);
26.         scrollableTextArea.setVerticalScrollBarPolicy(JScrollPane.VERTICAL_SCROLLBAR_ALWAYS);
27.
28.         frame.getContentPane().add(scrollableTextArea);
29.     }
30.     public static void main(String[] args) {
31.
32.
```

```
33.         javax.swing.SwingUtilities.invokeLater(new Runnable() {
34.
35.             public void run() {
36.                 createAndShowGUI();
37.             }
38.         });
39.     }
40. }
```

Output: