The Swing Buttons

Swing defines four types of buttons: JButton, JToggleButton, JCheckBox, and JRadioButton. All are subclasses of the AbstractButton class, which extends JComponent. Thus, all buttons share a set of common traits.

AbstractButton contains many methods that allow you to control the behavior of buttons. For example, you can define different icons that are displayed for the button when it is disabled, pressed, or selected.

Another icon can be used as ***a rollover icon***, which is displayed when the mouse is positioned over a button. The following methods set these icons:

- void setDisabledIcon(Icon di)
- void setPressedIcon(Icon pi)
- void setSelectedIcon(Icon si)
- void setRolloverIcon(Icon ri)

Here, di, pi, si, and ri are the icons to be used for the indicated purpose.

The text associated with a button can be read and written via the following methods:

- String getText( )
- void setText(String str)

Here, str is the text to be associated with the button.

The model used by all buttons is defined by the ButtonModel interface. A button generates an action event when it is pressed.

- ## 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.

public class JButton extends AbstractButton implements Accessible

Commonly used Constructors:

JButton allows an icon, a string, or both to be associated with the push button.

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.
JButton(String str, Icon icon)	It creates a button with the specified text and 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.

When the button is pressed, an **ActionEvent** is generated. Using the **ActionEvent** object passed to the **actionPerformed( )** method of the registered **ActionListener**, you can obtain the *action command* string associated with the button. By default, this is the string displayed inside the button. However, you can set the action command by calling **setActionCommand( )** on the button. You can obtain the action command by calling **getActionCommand( )** on the event object. It is declared like this:

- String getActionCommand( )

The action command identifies the button. Thus, when using two or more buttons within the same application, the action command gives you an easy way to determine which button was pressed.

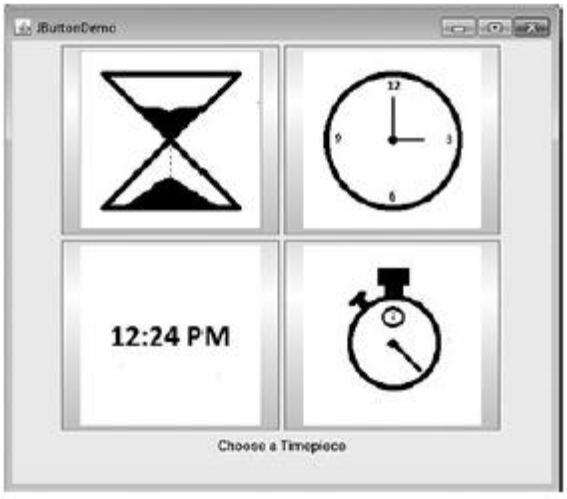
Example

Following program demonstrates an icon-based button. It displays four push buttons and a label. Each button displays an icon that represents a timepiece. When a button is pressed, the name of that timepiece is displayed in the label.

```
// Demonstrate an icon-based JButton.
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

public class JButtonDemo implements ActionListener {
  JLabel jlab;

  public JButtonDemo() {

    // Set up the JFrame.
    JFrame jfrm = new JFrame("JButtonDemo");
    jfrm.setLayout(new FlowLayout());
    jfrm.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    jfrm.setSize(500, 450);

    // Add buttons to content pane.
    ImageIcon hourglass = new ImageIcon("hourglass.png");
    JButton jb = new JButton(hourglass);
    jb.setActionCommand("Hourglass");
    jb.addActionListener(this);
    jfrm.add(jb);

    ImageIcon analog = new ImageIcon("analog.png");
    jb = new JButton(analog);
    jb.setActionCommand("Analog Clock");
    jb.addActionListener(this);
    jfrm.add(jb);

    ImageIcon digital = new ImageIcon("digital.png");
    jb = new JButton(digital);
    jb.setActionCommand("Digital Clock");
    jb.addActionListener(this);
    jfrm.add(jb);

    ImageIcon stopwatch = new ImageIcon("stopwatch.png");
    jb = new JButton(stopwatch);
    jb.setActionCommand("Stopwatch");
    jb.addActionListener(this);
    jfrm.add(jb);
```

```
    // Create and add the label to content pane.
    jlab = new JLabel("Choose a Timepiece");
    jfrm.add(jlab);

    // Display the frame.
    jfrm.setVisible(true);
  }

  // Handle button events.
  public void actionPerformed(ActionEvent ae) {
    jlab.setText("You selected " + ae.getActionCommand());
  }

  public static void main(String[] args) {
    // Create the frame on the event dispatching thread.

    SwingUtilities.invokeLater(
      new Runnable() {
        public void run() {
          new JButtonDemo();
        }
      }
    );

  }
}
```

Output:-



- # Java JToggleButton

JToggleButton is used to create toggle button, it is two-states button to switch pushed and released. That is, when you press a toggle button, it stays pressed rather than popping back up as a regular push button does. When you press the toggle button a second time, it releases (pops up).

Toggle buttons are objects of the **JToggleButton** class. **JToggleButton** implements **AbstractButton**. **JToggleButton** is a superclass for two other Swing components.These are **JCheckBox** and **JRadioButton**,

Nested Classes

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

Constructors

By default, the button is in the off position.

Constructor	Description
JToggleButton()	It creates an initially unselected toggle button without setting the text or image.
JToggleButton(Action a)	It creates a toggle button where properties are taken from the Action supplied.
JToggleButton(Icon icon)	It creates an initially unselected toggle button with the specified image but no text.
JToggleButton(Icon icon, boolean selected)	It creates a toggle button with the specified image and selection state, but no 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 selection state.
JToggleButton(String text, Icon icon)	It creates a toggle button that has the specified text and image, and that is initially unselected.

JToggleButton(String text, Icon icon, boolean selected)	It creates a toggle button with the specified text, image, and selection state.

Methods

Modifier and Type	Method	Description
AccessibleContext	getAccessibleContext()	It gets the AccessibleContext associated with this JToggleButton.
String	getUIClassID()	It returns a string that specifies the name of the l&f class that renders this 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 feel.

JToggleButton uses a model defined by a nested class called **JToggleButton.Toggle-ButtonModel**. Normally, you won't need to interact directly with the model to use a standard toggle button.

Like **JButton**, **JToggleButton** generates an action event each time it is pressed. **JToggleButton** also generates an item event. *This event is used by those components that support the concept of selection*. When a **JToggleButton** is pressed in, it is selected. When it is popped out, it is deselected.

To handle item events, you must implement the **ItemListener** interface. Each time an item event is generated, it is passed to the **itemStateChanged( )** method defined by **ItemListener**. Inside **itemStateChanged( )**, the **getItem( )** method can be called on the **ItemEvent** object to obtain a reference to the **JToggleButton** instance that generated the event. It is shown here:

- Object getItem( )

The easiest way to determine a toggle button's state is by calling the **isSelected( )** method (inherited from **AbstractButton**) on the button that generated the event. It is shown here:

- boolean isSelected( )

It returns **true** if the button is selected and **false** otherwise.

Example:-

```
// Demonstrate JToggleButton.
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

public class JToggleButtonDemo {

  public JToggleButtonDemo() {

    // Set up the JFrame.
    JFrame jfrm = new JFrame("JToggleButtonDemo");
    jfrm.setLayout(new FlowLayout());
    jfrm.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    jfrm.setSize(200, 100);

    // Create a label.
    JLabel jlab = new JLabel("Button is off.");

    // Make a toggle button.
    JToggleButton jtbn =  new JToggleButton("On/Off");

    // Add an item listener for the toggle button.
    jtbn.addItemListener(new ItemListener() {
      public void itemStateChanged(ItemEvent ie) {
        if(jtbn.isSelected())
          jlab.setText("Button is on.");
        else
          jlab.setText("Button is off.");
      }
    });

    // Add the toggle button and label to the content pane.
    jfrm.add(jtbn);
    jfrm.add(jlab);

    // Display the frame.
    jfrm.setVisible(true);
  }

  public static void main(String[] args) {
    // Create the frame on the event dispatching thread.

    SwingUtilities.invokeLater(
      new Runnable() {
        public void run() {
          new JToggleButtonDemo();
        }
      }
    );

  }
}
```

Output:-