INTRODUCTION TO SWING

Difference between AWT and Swing

There are many differences between java awt and swing that are given below.

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

Limitations of AWT, MVC Architecture, Components & Containers

Limitations of AWT:

- The AWT defines a basic set of controls, windows, and dialog boxes that support a usable, but limited graphical interface.
- One reason for the limited nature of the AWT is that it translates its various visual components into their corresponding, platform-specific equivalents or peers.
- This means that the look and feel of a component is defined by the platform, not by java.
- Because the AWT components use native code resources, they are referred to as heavy weight.
- The use of native peers led to several problems.
    - First, because of variations between operating systems, a component might look, or even act, differently on different platforms.
    - Second, the look and feel of each component was fixed and could not be changed.
    - Third, the use of heavyweight components caused some frustrating restrictions.
- Due to these limitations Swing came and was integrated to java.
- Swing is built on the AWT.
- Two key Swing features are:
    - Swing components are light weight,
    - Swing supports a pluggable look and feel.

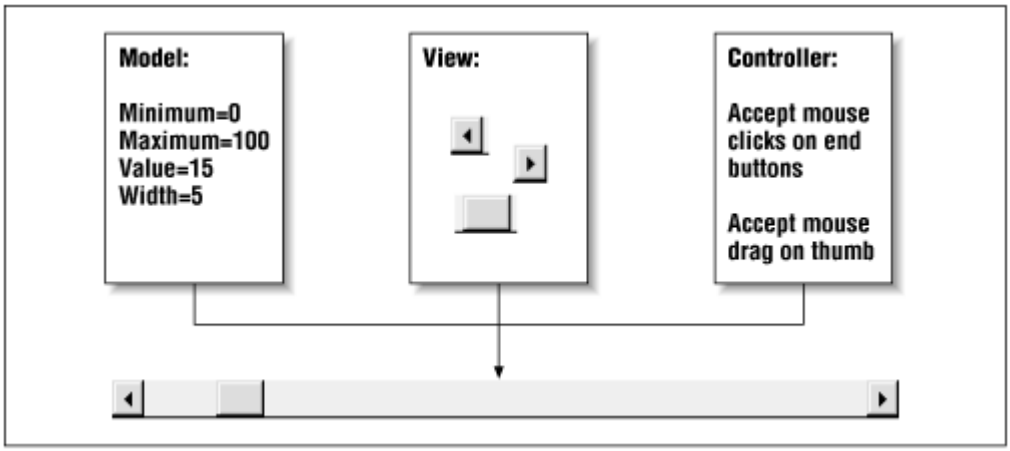
The MVC Connection:

- In general, a visual component is a composite of three distinct aspects: • The way that the component looks when rendered on the screen(view)

  • The way that the component reacts to the user (controller)

  • The state information associated with the component.(model)

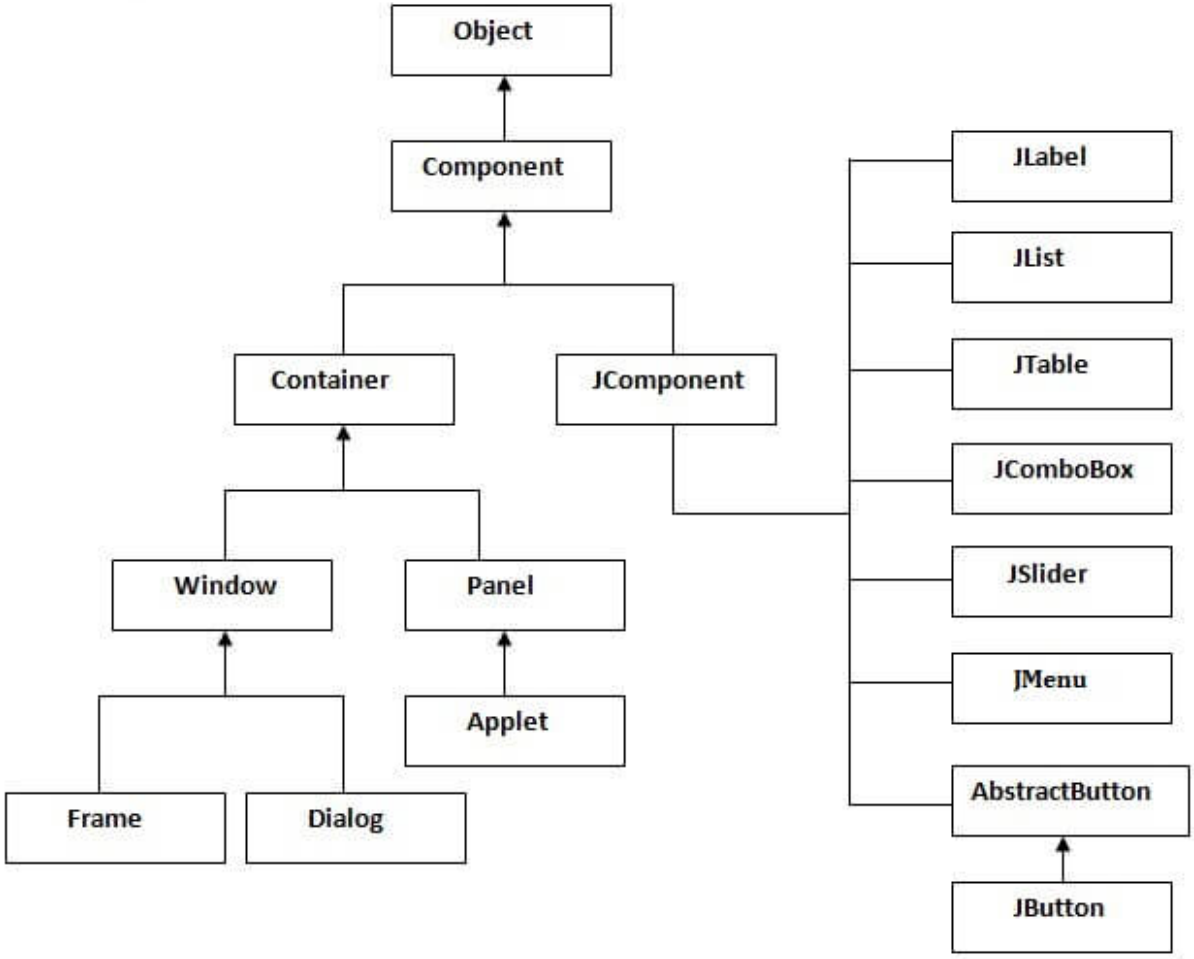
  The Model-View-Controller architecture is successful for all these.

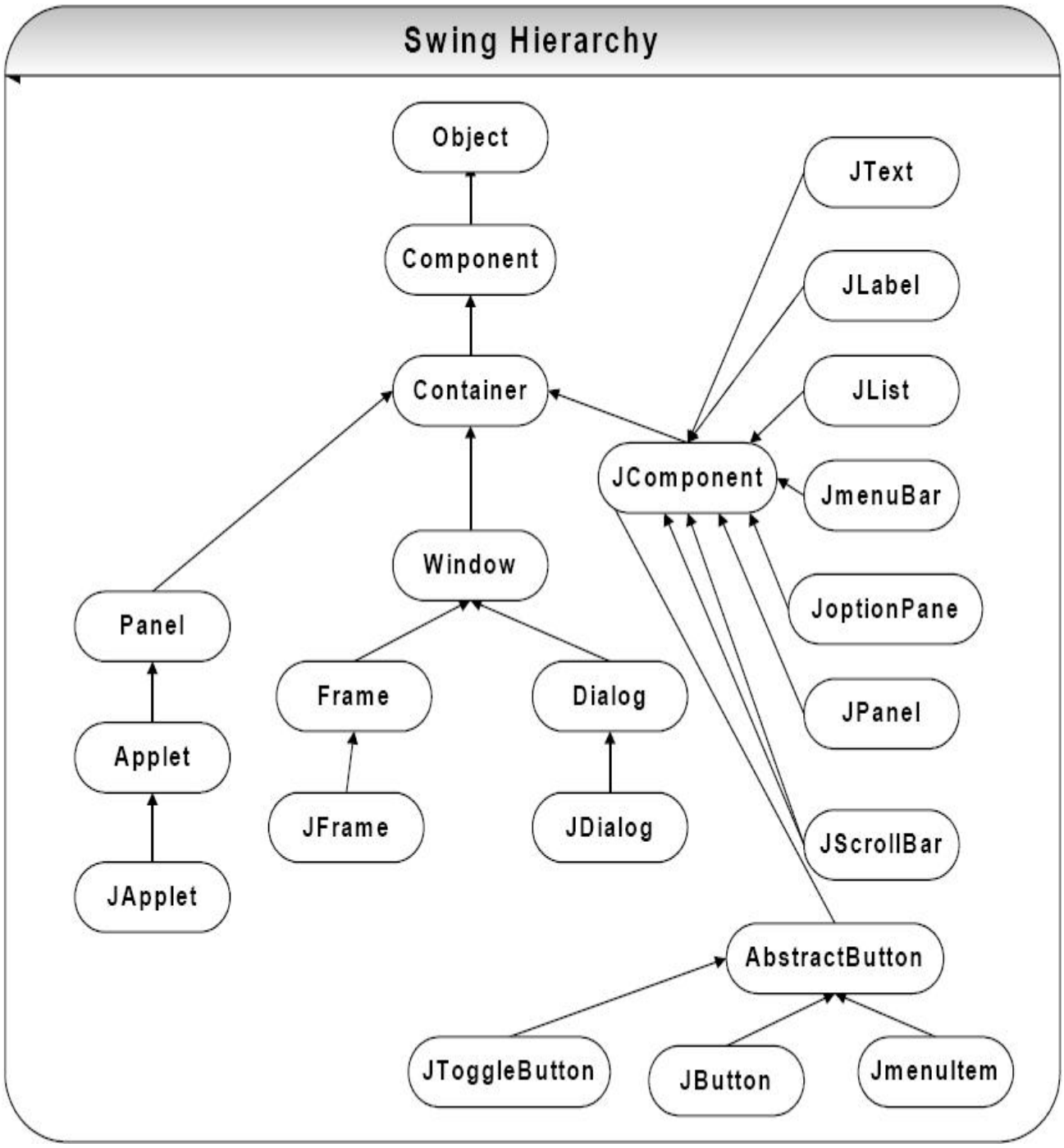


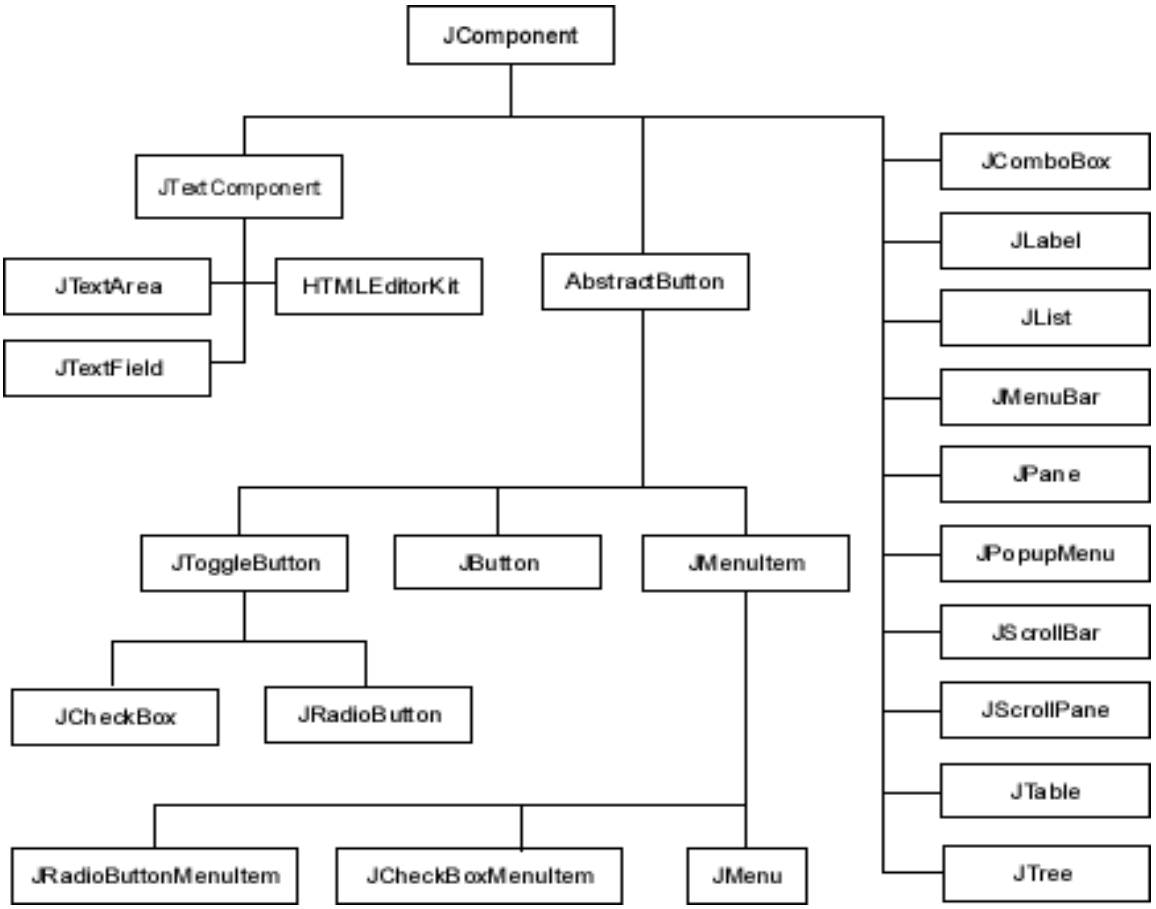
Above figure shows how the model, view, and controller work together to create a scrollbar component. **The scrollbar uses the information in the model to determine how far into the scrollbar to render the thumb and how wide the thumb should be.** Note that the model specifies this information relative to the minimum and the maximum. It does not give the position or width of the thumb in screen pixels—the view calculates that. **The view determines exactly where and how to draw the scrollbar, given the proportions offered by the model.** The view knows whether it is a horizontal or vertical scrollbar, and it knows exactly how to shadow the end buttons and the thumb. Finally, the controller is responsible for handling mouse events on the component. The controller knows,

for example, that dragging the thumb is a legitimate action for a scroll bar, and pushing on the end buttons is acceptable as well. The result is a fully functional MVC scrollbar.

SWINGHIERARCHY




Swing Hierarchy
Object
Component
Container
Panel
Applet
JApplet
Window
Frame
JFrame
Dialog
JDialog
JComponent
JText
JLabel
JList
JmenuBar
JoptionPane
JPanel
JScrollBar
AbstractButton
JToggleButton
JButton
JmenuItem



Components

- In general, Swing components are derived from the **JComponent** class.
- All of Swing's components are represented by classes defined within the package **javax.swing**.
- The following table shows the class names for Swing components (including those used as containers).
  - ✓ JApplet
  - ✓ JButton

- ✓ JCheckBox
- ✓ JDialog
- ✓ JFileChooser
- ✓ JFrame
- ✓ JLabel
- ✓ JList
- ✓ JMenu
- ✓ JMenuBar
- ✓ JMenuItem
- ✓ JPanel
- ✓ JPasswordField
- ✓ JProgressBar
- ✓ JRadioButton
- ✓ JScrollBar
- ✓ JScrollPane
- ✓ JTable
- ✓ JTextArea
- ✓ JTextField
- ✓ JToolBar
- ✓ JToolTip
- ✓ JTree
- ✓ JWindow

Containers

- Swing defines two types of containers. The first are
    - top-level containers: **JFrame**, **JApplet**, **JWindow**, and **JDialog**.
- inherit the AWT classes **Component** and **Container**.

- the top-level containers are heavyweight.
- The second type of containers supported by Swing are lightweight containers.
- Lightweight containers inherit **JComponent**.
- An example of a lightweight container is **JPanel**,

A Simple Swing Application

- There are two types of Java programs in which Swing is typically used.

1. The first is a desktop application.
2. The second is the applet.

//JFRAME

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

}

Frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

- After this call executes, closing the window causes the entire application to terminate. (closes all frames i.e frames within frames also.)
- There are several other options in addition to **JFrame.EXIT_ON_CLOSE**. They are shown here:

JFrame.DISPOSE_ON_CLOSE(closes only that frame)

JFrame.HIDE_ON_CLOSE

JFrame.DO_NOTHING_ON_CLOSE

- Their names reflect their actions.
- By default, a **JFrame** is invisible, so **setVisible(true)** must be called to show it.
- jfrm.setVisible(true);

//CloseOpDemo.java in Notepad

Create a Swing Applet

- A Swing applet extends **JApplet** rather than **Applet**.
- **JApplet** is derived from **Applet**.
- **JApplet** is a top-level Swing container, which means that it is *not* derived from **JComponent**.
- Swing applets use the same four lifecycle methods **init( )**, **start( )**, **stop( )**, and **destroy( )**.
- an example of a Swing applet.

```
// JAPPLET
import java.applet.*;
import javax.swing.*;
import java.awt.event.*;
/*<applet code="EventJApplet.class" width="300" height="300">
</applet>  */
 public class EventJApplet extends JApplet implements ActionListener{
JButton b;
JTextField tf;
public void init(){
 tf=new JTextField();
tf.setBounds(30,40,150,20);
 b=new JButton("Click");
b.setBounds(80,150,70,40);
 add(b);add(tf);
b.addActionListener(this);
 setLayout(null);
}
 public void actionPerformed(ActionEvent e){
tf.setText("Welcome");
}
}
```

PAINTING IN SWING: A PAINT EXAMPLE

Displaying graphics in swing:

java.awt.Graphics class provides many methods for graphics programming.

Commonly used methods of Graphics class:

1. **public abstract void drawString(String str, int x, int y):** is used to draw the specified string.
2. **public void drawRect(int x, int y, int width, int height):** draws a rectangle with the specified width and height.
3. **public abstract void fillRect(int x, int y, int width, int height):** is used to fill rectangle with the default color and specified width and height.
4. **public abstract void drawOval(int x, int y, int width, int height):** is used to draw oval with the specified width and height.
5. **public abstract void fillOval(int x, int y, int width, int height):** is used to fill oval with the default color and specified width and height.
6. **public abstract void drawLine(int x1, int y1, int x2, int y2):** is used to draw line between the points(x1, y1) and (x2, y2).
7. **public abstract boolean drawImage(Image img, int x, int y, ImageObserver observer):** is used draw the specified image.
8. **public abstract void drawArc(int x, int y, int width, int height, int startAngle, int arcAngle):** is used draw a circular or elliptical arc.
9. **public abstract void fillArc(int x, int y, int width, int height, int startAngle, int arcAngle):** is used to fill a circular or elliptical arc.

10. **public abstract void setColor(Color c):** is used to set the graphics current color to the specified color.
11. **public abstract void setFont(Font font):** is used to set the graphics current font to the specified font.

```
import java.awt.*;
import javax.swing.JFrame;
 public class DisplayGraphics extends Canvas{
    public void paint(Graphics g) {
      g.drawString("Hello",40,40);
      setBackground(Color.WHITE);
      g.fillRect(130, 30,100, 80);
      g.drawOval(30,130,50, 60);
      setForeground(Color.RED);
      g.fillOval(130,130,50, 60);
      g.drawArc(30, 200, 40,50,90,60);
      g.fillArc(30, 130, 40,50,180,40);
    }
      public static void main(String[] args) {
      DisplayGraphics m=new DisplayGraphics();
      JFrame f=new JFrame();
      f.add(m);
      f.setSize(400,400);
      f.setVisible(true);
  }
 }
```

Exploring Swing

- The Swing component classes described in this chapter are shown here:
    - JLabel and ImageIcon
    - JTextField
    - Swing Buttons:
        - JButton
        - JToggleButton
    - JCheckBox
    - JRadioButton
    - JTabbedPane
    - JScrollPane
    - JList
    - JComboBox
    - Swing Menus
    - Dialogs

These components are all lightweight, which means that they are all derived from **JComponent**.

JLabel and ImageIcon

Commonly used Constructors:

Constructor	Description
JLabel()	Creates a JLabel instance with no image and with an empty string for the title.
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, and horizontal alignment.

Commonly used Methods:

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

```
// Demonstrate JLabel and ImageIcon.
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.ImageIcon;
import java.awt.FlowLayout;
class Frame extends JFrame{
  Frame()    {
      setTitle("ImageIcon");
      setLayout(new FlowLayout());
        ImageIcon icon = new ImageIcon("three.jpg");
      JLabel jl = new JLabel("LOVE",icon,JLabel.CENTER);
      add(jl);
      setSize(700, 200);
      setVisible(true);
      setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);   }}
public class Javaapp {
   public static void main(String[] args) {
      Frame frame = new Frame();    }}
```

JTextField

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 columns.

JTextField(int columns)	Creates a new empty TextField with the specified number of columns.

```
        // Demonstrate JTextField.
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 Java.");
    t1.setBounds(50,100, 200,30);
    t2=new JTextField("SWINGS");
    t2.setBounds(50,150, 200,30);
    f.add(t1); f.add(t2);
    f.setSize(400,400);
    f.setLayout(null);
    f.setVisible(true);
    }
    }
```

The Swing Buttons

- Swing defines four types of buttons: **JButton**, **JToggleButton**, **JCheckBox**, and **JRadioButton**.

All are subclasses of the **AbstractButton** class, which extends **JComponent**.

JButton

Three of its constructors are shown here:

JButton(Icon *icon*)

JButton(String *str*)

JButton(String *str*, Icon *icon*)

```
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.setVisible(true);
}
}
```

JToggleButton

- A useful variation on the push button is called a *toggle button*.
- A toggle button looks just like a push button, but it acts differently because it has two states: 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).
- Therefore, each time a toggle button is pushed, it toggles between its two states.

JToggleButton(String *str*)

This creates a toggle button that contains the text passed in *str*. By default, the button is in the off position.

```
// Demonstrate JToggleButton.
import java.awt.FlowLayout;
import java.awt.event.ItemEvent;
import java.awt.event.ItemListener;
import javax.swing.JFrame;
import javax.swing.JToggleButton;

public class JToggleButtonExample extends JFrame implements ItemListener {
   public static void main(String[] args) {
      new JToggleButtonExample();
   }
   private JToggleButton button;
   JToggleButtonExample() {
      setTitle("JToggleButton with ItemListener Example");
      setLayout(new FlowLayout());
      setJToggleButton();
      setAction();
      setSize(200, 200);
```

```
        setVisible(true);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }
    private void setJToggleButton() {
        button = new JToggleButton("ON");
        add(button);
    }
    private void setAction() {
        button.addItemListener(this);
    }
    public void itemStateChanged(ItemEvent eve) {
        if (button.isSelected())
            button.setText("OFF");
        else
            button.setText("ON");
    }
}
```

Check Boxes

- The **JCheckBox** class provides the functionality of a check box.
- Its immediate superclass is **JToggleButton**, which provides support for two-state buttons, as just described.
- **JCheckBox** defines several constructors. The one used here is

JCheckBox(String *str*)

```
// Demonstrate JCheckbox.
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, 100,100);
      f.add(checkBox1);
      f.add(checkBox2);
      f.setSize(400,400);
      f.setLayout(null);
      f.setVisible(true);
    }
public static void main(String args[])
   {
   new CheckBoxExample();
   }}
```

Radio Buttons

- Radio buttons are a group of mutually exclusive buttons, in which only one button can be selected at any one time.
- They are supported by the **JRadioButton** class, which extends **JToggleButton**.

JRadioButton provides several constructors. The one used in the example is shown here:

JRadioButton(String *str*)

```
// Demonstrate JRadioButton
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();
}
}
```

JTabbedPane

- **JTabbedPane** encapsulates a *tabbed pane*.
- It manages a set of components by linking them with tabs.
- Selecting a tab causes the component associated with that tab to come to the forefront.

```
import javax.swing.*;
public class TabbedPaneExample {
JFrame f;
TabbedPaneExample(){
    f=new JFrame();
    JTextArea ta=new JTextArea(200,200);
    JPanel p1=new JPanel();
    p1.add(ta);
    JPanel p2=new JPanel();
    JButton b=new JButton("HELLO");
    p2.add(b);
    JPanel p3=new JPanel();
    JLabel l=new JLabel("HI");
    p3.add(l);
    JTabbedPane tp=new JTabbedPane();
    tp.setBounds(50,50,200,200);
    tp.add("main",p1);
```

```
    tp.add("visit",p2);
    tp.add("help",p3);
    f.add(tp);
    f.setSize(400,400);
    f.setLayout(null);
    f.setVisible(true);
}
public static void main(String[] args) {
    new TabbedPaneExample();
}}
```

JScrollPane

- **JScrollPane** is a lightweight container that automatically handles the scrolling of another component.

JScrollPane defines several constructors.

JScrollPane(Component *comp*)

Here are the steps to follow to use a scroll pane:

1. Create the component to be scrolled.
2. Create an instance of **JScrollPane**, passing to it the object to scroll.
3. Add the scroll pane to the content pane.

```
// Demonstrate JScrollPane.
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_SC
ROLLBAR_ALWAYS);
scrollableTextArea.setVerticalScrollBarPolicy(JScrollPane.VERTICAL_SCROLL
BAR_ALWAYS);

        frame.getContentPane().add(scrollableTextArea);
    }
    public static void main(String[] args) {
        javax.swing.SwingUtilities.invokeLater(new Runnable() {
            public void run() {
```

```
            createAndShowGUI();
          }
       });
   }
}
```

JList

- In Swing, the basic list class is called **JList**.
- It supports the selection of one or more items from a list.

```
// Demonstrate JList.
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();
   }}
```

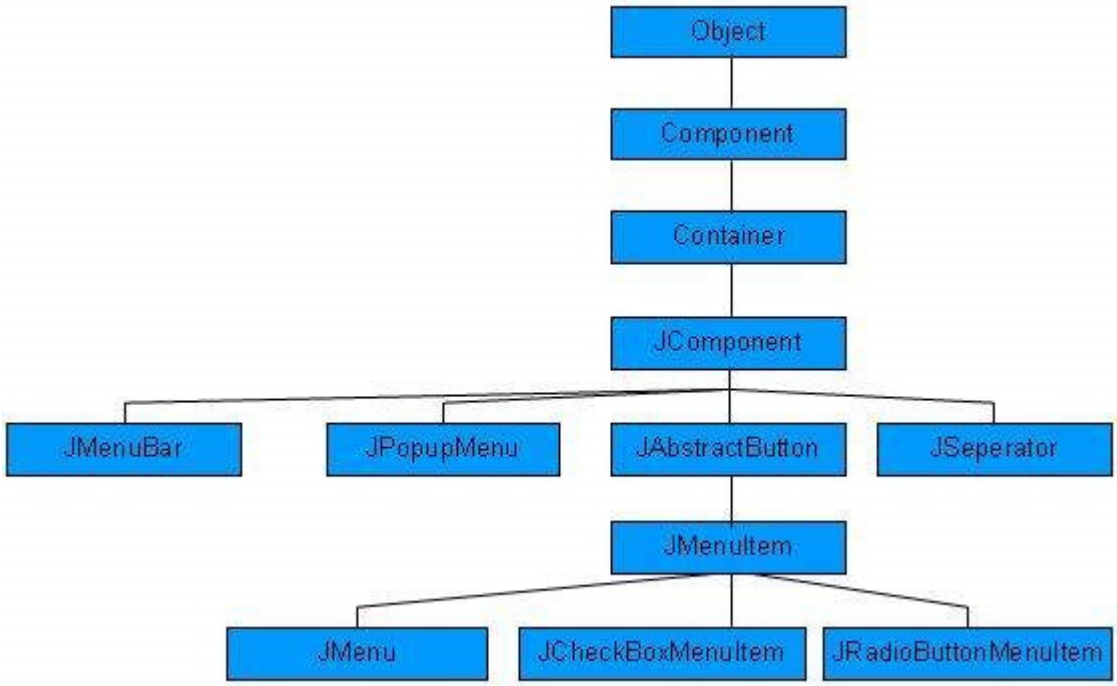
JComboBox

- Swing provides a *combo box* (a combination of a text field and a drop-down list) through the **JComboBox** class.
- A combo box normally displays one entry, but it will also display a drop-down list that allows a user to select a different entry.

```
// Demonstrate JComboBox.
import javax.swing.*;
public class ComboBoxExample {
JFrame f;
ComboBoxExample(){
   f=new JFrame("ComboBox Example");
   String country[]={"India","Aus","U.S.A","England","Newzealand"};
   JComboBox<String> 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();
}
}
```

SWING MENUS

Menu Hierarchy



Menu Controls

Sr.No.	Class & Description
1	**JMenuBar** The JMenuBar object is associated with the top-level window.
2	**JMenuItem** The items in the menu must belong to the JMenuItem or any of its subclass.
3	**JMenu** The JMenu object is a pull-down menu component which is

	displayed from the menu bar.
4	**JCheckboxMenuItem** JCheckboxMenuItem is the subclass of JMenuItem.
5	**JRadioButtonMenuItem** JRadioButtonMenuItem is the subclass of JMenuItem.
6	**JPopupMenu** JPopupMenu can be dynamically popped up at a specified position within a component.

```
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.setVisible(true);
}
public static void main(String args[])
{
new MenuExample();
}}
```

Java JDialog

The JDialog control represents a top level window with a border and a title used to take some form of input from the user. It inherits the Dialog class.

Unlike JFrame, it doesn't have maximize and minimize buttons.

Commonly used Constructors:

Constructor	Description
JDialog()	It is used to create a modeless dialog without a title and without a specified Fram
JDialog(Frame owner)	It is used to create a modeless dialog with specified Frame as its owner and an empty title.

JDialog(Frame owner, String title, boolean modal)	It is used to create a dialog with the specified title, owner Frame and modality.

Java JDialog Example

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
public class DialogExample {
    private static JDialog d;
    DialogExample() {
        JFrame f= new JFrame();
        d = new JDialog(f , "Dialog Example", true);
        d.setLayout( new FlowLayout() );
        JButton b = new JButton ("OK");
        b.addActionListener ( new ActionListener()
        {
            public void actionPerformed( ActionEvent e )
            {
                DialogExample.d.setVisible(false);
            }
        });
```

```
        d.add( new JLabel ("Click button to continue."));
        d.add(b);
        d.setSize(300,300);
        d.setVisible(true);
    }
    public static void main(String args[])
    {
        new DialogExample();
    }
}
```