# CS205 Object Oriented Programming in Java

## Module 5 - **Graphical User Interface and Database support of Java**

## (Part 5)

Prepared by

**Renetha J.B.**

AP

Dept.of CSE,

Lourdes Matha College of Science and Technology

## Topics

- ☑ **Swings**
  - ☑ Exploring Swings
    - ☑JFrame
    - ☑Jlabel
    - ☑The Swing Buttons
    - ☑JTextField

- Some of the swing components are:
  - JButton
  - JCheckBox
  - JComboBox
  - JLabel
  - JList
  - JRadioButton
  - JScrollPane
  - JTabbedPane
  - JTable
  - JTextField
  - JToggleButton
  - JTree

These components are all lightweight. They are all derived from **JComponent.**

## JFrame

- Every containment hierarchy must begin with a top-level container.
- **JFrame** is a top level container that is commonly used for Swing applications.
- JFrame  do not inherit **JComponent.**
- **JFrame** inherit the AWT classes **Component and Container.**
- The top-level containers are heavyweight.

# **JFrame**(contd.)

**JFrame jfrm** = new **JFrame**(“Swing Example);

- This creates a container called **jfrm** that
  - defines a rectangular window complete with a title bar; close, minimize, maximize, and restore buttons; and a system menu.
  - Thus, it creates a standard, top-level window.
  - The title of the window is passed to the constructor.
    - Here it is ***Swing Example***

## **JFrame**(contd.)

- The **setSize( )** method (which is inherited by JFrame from the AWT class Component) sets the dimensions of the window, which are specified in pixels.
- Its general form is :

void setSize(int *width, int height)*

*E.g.* **jfrm**.**setSize**(275, 100);

## **JFrame**(contd.)

- If we want the entire application to terminate when its top-level window is closed the easiest way is to call **setDefaultCloseOperation( )**

E.g.

**jfrm**.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

- After this call executes, closing the window causes the entire application to terminate.
- The general form of **setDefaultCloseOperation( ) is :**

  void **setDefaultCloseOperation**(int *what)*

  – *what* can be
    - JFrame.EXIT_ON_CLOSE
    - JFrame.DISPOSE_ON_CLOSE
    - JFrame.HIDE_ON_CLOSE
    - JFrame.DO_NOTHING_ON_CLOSE

## **JFrame**(contd.)

- The content pane can be obtained by calling **getContentPane( )** on a JFrame instance.
- The **getContentPane( )** method is :

Container **getContentPane**( )

- The setVisible( ) method is inherited from the AWT Component class.
  - If its argument is true, the window will be displayed. Otherwise, it will be hidden.
    - By default, a JFrame is invisible, so setVisible(true) must be called to show it.
  - E.g.

```
jfrm.setVisible(true);
```

# JLabel

- JLabel is Swing's easiest-to-use component.
- It creates a label.
- **JLabel** can be used **to display text and/or an icon**.
- It is a **passive component** because it does not respond to user input.
- JLabel defines several constructors:

  JLabel(Icon *icon)*

  JLabel(String *str)*

  JLabel(String *str, Icon icon, int align)*

  – The *align argument specifies the* horizontal alignment of the text and/or icon within the dimensions of the label.
    - It must be one of the following values: **LEFT, RIGHT, CENTER, LEADING, or TRAILING**

## **JLabel**(contd.)

- The easiest way to obtain an icon is to use the ImageIcon class.
- ImageIcon implements Icon and encapsulates an image.
- The following **ImageIcon** constructor obtains the image in the file named *filename*. the Icon parameter of JLabel's constructor

  **ImageIcon**(String *filename)*
- The icon and text associated with the label can be obtained by the following methods:

  **Icon** getIcon( )

  String getText( )
- The icon and text associated with a label can be set by these methods:

  void setIcon(Icon *icon)*

  void setText(String *str)*

## The Swing Buttons

- Swing defines four types of buttons:

  **JButton**

  **JToggleButton**

  **JCheckBox**

  **JRadioButton**

- All are subclasses of the **AbstractButton class** (which extends JComponent)

## The Swing Buttons(contd.)

- **AbstractButton** contains many methods that allow you to control the behavior of buttons.
- E.g. We 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.

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* for specific purpose

## The Swing Buttons (contd.)

- We can get the text associated with a button using:

  String **getText**( )

- We can modify the text associated with a button using:

  void **setText**(String *str)*

- The model used by all buttons is defined by the **ButtonModel interface.**

## JButton

- The **JButton** class provides the functionality of a **push button.**
- **JButton** allows an **icon, a string, or both** to be associated with the push button.
- Three of its constructors are shown here:

  JButton(Icon *icon)*

  JButton(String *str)*

  JButton(String *str, Icon icon)*
- When the button is pressed, an **ActionEvent** is generated**.**

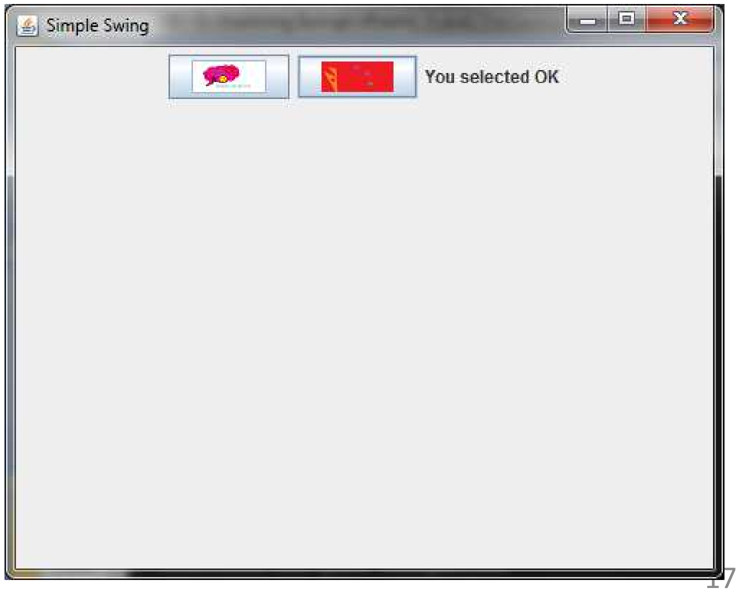
## JButton(contd.)

- Using the **ActionEvent** object passed to the **actionPerformed( )** method of the registered **ActionListener,** we can obtain the *action command string associated with the button.*
- We can set the action command by calling **setActionCommand( )** on the button.
- We can obtain the action command by calling **getActionCommand( )**

String **getActionCommand**( )

- The action command helps to identify 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.

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
class SwingButton extends
    JFrame implements
    ActionListener
{   JFrame jfrm;
    JButton jbok , jbcancel;
    JLabel jlab;
    SwingButton()
    {
jfrm = new JFrame("Simple
    Swing ");
jfrm.setSize(500, 400);
jfrm.setLayout(new
    FlowLayout());
jfrm.setDefaultCloseOperation(J
    Frame.EXIT_ON_CLOSE);
ImageIcon imgok= new
    ImageIcon("C:\\RJB\\image1.jpg");
    jbok = new JButton(imgok);
    jbok.setActionCommand("OK");
    jfrm.add(jbok);
ImageIcon imgcancel= new
    ImageIcon("C:\\RJB\\image2.jpg");
    jbcancel = new JButton(imgcancel);
jbcancel.setActionCommand("Cancel");
jbok.addActionListener(this);
jbcancel.addActionListener(this);
    jfrm.add(jbcancel);
jlab = new JLabel("Waiting button press ");
    jfrm.add(jlab);
    jfrm.setVisible(true);
    }
public void actionPerformed(ActionEvent ae)
    {
    jlab.setText("You selected " + ae.getActionCommand());
    }
    public static void main(String args[])
    {       SwingUtilities.invokeLater(new Runnable()
                                   {
                                            public void run()
                                            {
                                            new SwingButton ();
                                            }
                                   }
                                   );
    }
}
```



You selected OK

*Prepared by Renetha J.B.*

## JToggleButton

- A toggle button looks just like a push button,
- It acts differently from push button because it has two states:
  - **Pushed**
  - **Released**
- When we press a toggle button, it **stays pressed.**
  - It does not pop back up as a regular push button.
- When we press the toggle button a second time, it releases (pops up).
- Each time a toggle button is pushed, it toggles between its two states
- Toggle buttons are objects of the **JToggleButton** class.

## **JToggleButton**(contd.)

- JToggleButton implements **AbstractButton**.
- JToggleButton is a superclass for **JCheckBox** and **JRadioButton**
- **JToggleButton** defines several constructors.

  JToggleButton(String *str)*

This creates a toggle button that contains the text passed in *str*.

- By **default**, the button is in the **off** position.
- **JToggleButton** uses a model defined by a nested class called JToggleButton .ToggleButtonModel.
- **JToggleButton** generates an action event each time it is pressed.
- When a **JToggleButton is pressed** in, it is **selected**.
- When it is popped out, it is **deselected**.

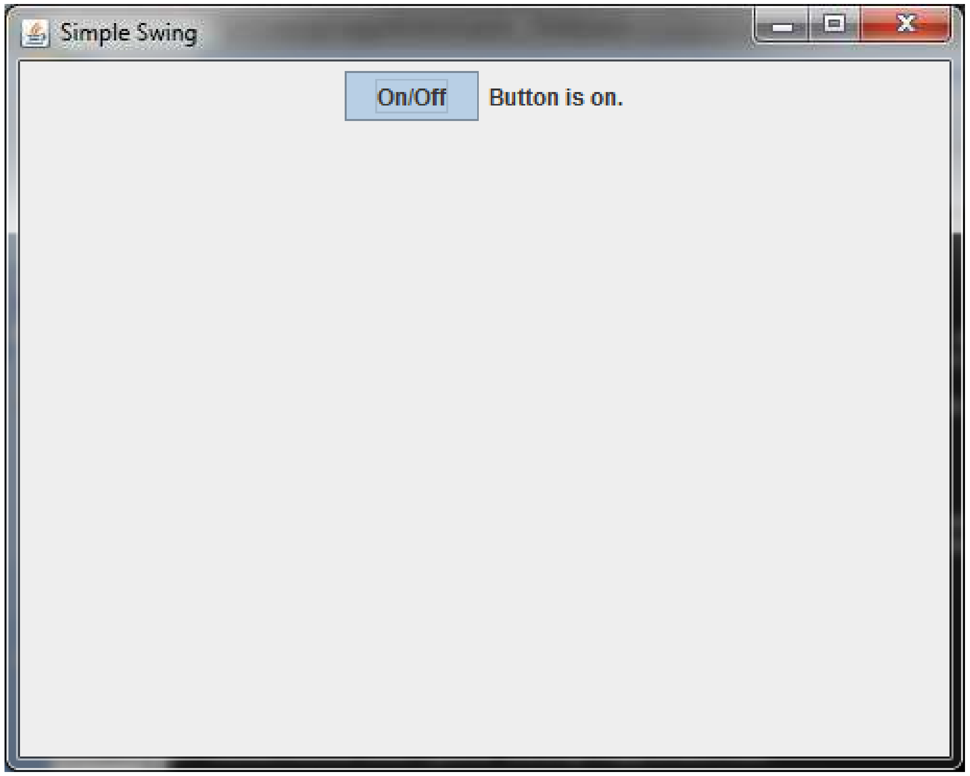
## **JToggleButton**(contd.)

- To handle item events, we 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.

  Object **getItem( )**
- The easiest way to determine a toggle button's state is by calling the **isSelected( )** method.

  boolean **isSelected( )**
  – It returns **true if the button is selected** and false otherwise.

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
class SwingToggleButton extends
    JFrame implements ItemListener
{   JFrame jfrm;
    JToggleButton jtbn;
    JLabel jlab;
SwingToggleButton()
    {
jfrm = new JFrame("Simple Swing ");
jfrm.setSize(500, 400);
jfrm.setLayout(new FlowLayout());
jfrm.setDefaultCloseOperation(JFrame
    .EXIT_ON_CLOSE);
    jtbn=new JToggleButton("On/Off");
    jtbn.addItemListener(this);
    jfrm.add(jtbn);
    jlab = new JLabel("Button is OFF");
    jfrm.add(jlab);
    jfrm.setVisible(true);
    }
public void itemStateChanged(ItemEvent ie)
    {
        if(jtbn.isSelected())
            jlab.setText("Button is on.");
        else
             jlab.setText("Button is off.");
    }
    public static void main(String args[])
    {       SwingUtilities.invokeLater(new Runnable()
                                      {
                                              public void run()
                                              {
                                              new SwingToggleButton ();
                                              }
                                      }
                                      );
    }
}
```



Button is on.

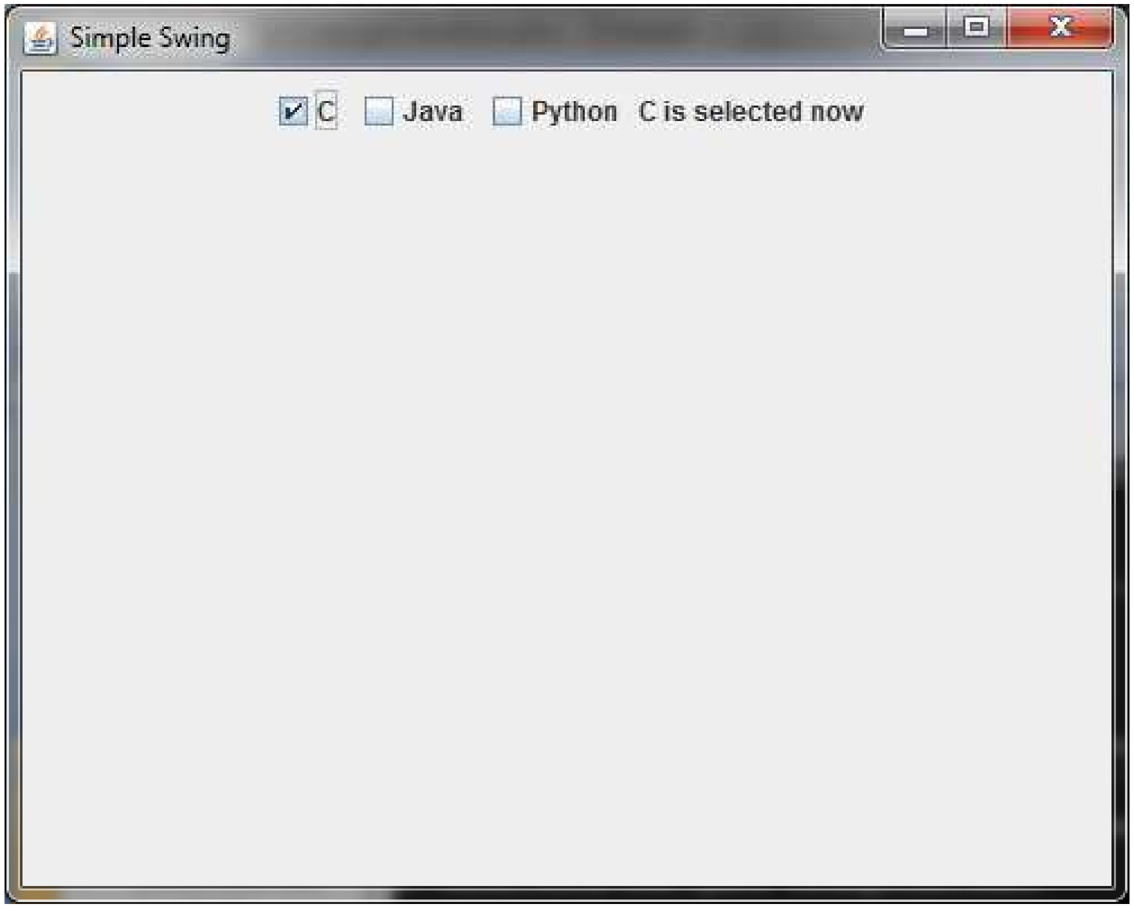
# JCheckBox

- **JCheckBox** class provides the functionality of a check box.
- Its immediate **superclass** is **JToggleButton**,
- **JCheckBox** defines several constructors.

  JCheckBox(String *str)*
- When the user selects or deselects a check box, an ItemEvent is generated.
- Inside the **itemStateChanged(** ) method, **getItem(** ) is called on ItemEvent object to obtain a reference to the JCheckBox object that generated the event
- To determine the selected state of a check box is to call **isSelected(** ) on the **JCheckBox** instance.

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
class SwingJCheckBox extends JFrame
    implements ItemListener
{   JFrame jfrm;
     JCheckBox cb;
    JLabel jlab;
SwingJCheckBox()
    {
jfrm = new JFrame("Simple Swing ");
jfrm.setSize(500, 400);
jfrm.setLayout(new FlowLayout());
jfrm.setDefaultCloseOperation(JFrame
    .EXIT_ON_CLOSE);
 JCheckBox cb = new JCheckBox("C");
cb.addItemListener(this);
jfrm.add(cb);
cb = new JCheckBox("Java");
cb.addItemListener(this);
jfrm.add(cb);
cb = new JCheckBox("Python");
cb.addItemListener(this);
jfrm.add(cb);

jlab = new JLabel("Select language");
jfrm.setVisible(true);
}
public void itemStateChanged(ItemEvent ie)
    {       JCheckBox cb = (JCheckBox)ie.getItem();
            if(cb.isSelected())
                    jlab.setText(cb.getText() + " is selected now");
            else
                    jlab.setText(cb.getText() + " is cleared now");
    }
public static void main(String args[])
    {               SwingUtilities.invokeLater(new Runnable()
                    {
                            public void run()
                            {
                                    new SwingJCheckbox();
                            }
                    });
    } }
```



```
C is selected now
```

## JRadioButton

- 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

JRadioButton(String *str)*

- A button group is created by the **ButtonGroup** class.
- Its default constructor is invoked for this purpose.
- Elements are then added to the button group via the following method:

void **add**(AbstractButton *ab)*

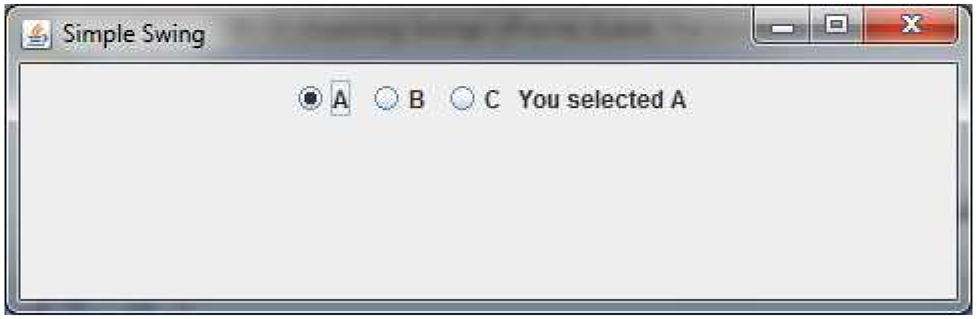
  – Here, *ab is a reference to the button to be added to the group.*

- A **JRadioButton** generates **action events, item events,** and **change**

## **JRadioButton**(contd.)

- We will normally implement the **ActionListener interface** with method **actionPerformed( ).**
  - Inside this method we can check its action command by calling **getActionCommand( ).**
    - By **default**, the action command is the same as the **button label**, but we can set the action command to something else by calling **setActionCommand( ) on the radio button.**
- We can call **getSource( ) on the ActionEvent object and check that reference against the buttons.**
- We can simply check each radio button to find out which one is currently selected by calling **isSelected( )** on each button.

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
class SwingJRadioButton extends
    JFrame implements ActionListener
{   JFrame jfrm;
    JLabel jlab;
SwingJRadioButton()
    {
jfrm = new JFrame("Simple Swing ");
jfrm.setSize(220, 100);
jfrm.setLayout(new FlowLayout());
jfrm.setDefaultCloseOperation(JFrame
    .EXIT_ON_CLOSE);
JRadioButton b1 = new JRadioButton("A");
        b1.addActionListener(this);
        jfrm.add(b1);
JRadioButton b2 = new JRadioButton("B");
        b2.addActionListener(this);
        jfrm.add(b2);
JRadioButton b3 = new JRadioButton("C");
        b3.addActionListener(this);
        jfrm.add(b3);
    ButtonGroup bg = new ButtonGroup();
        bg.add(b1);
        bg.add(b2);
        bg.add(b3);
  jlab = new JLabel("Select language");
  jfrm.setVisible(true);
  }
public void actionPerformed(ActionEvent ae)
    {
            jlab.setText("You selected " + ae.getActionCommand());
    }
    public static void main(String args[])
    {                       SwingUtilities.invokeLater(new Runnable()
                            {
                                    public void run()
                                    {
                                            new SwingJRadioButton ();
                                    }
                            });
    } }
```



Simple Swing

● A ○ B ○ C You selected A

## JTextField.

- **JTextField** is the simplest Swing text component.
- JTextField allows you to **edit <u>one line</u> of text.**
  - It is derived from **JTextComponent,** which provides the basic functionality common to Swing text components.
- Three of **JTextField's constructors** are **:**

  JTextField(int *cols)*

  JTextField(String *str, int cols)*

  JTextField(String *str)*

  - Here, *str* is the string to be initially presented, and *cols* is the number of columns in the text field. If no string is specified, the text field is initially empty.
  - If the number of columns is not specified, the text field is sized to fit the specified string

## **JTextField**(contd.)

- **JTextField** generates events in response to user interaction.
  - For example, an **ActionEvent** is fired when the user presses ENTER.
  - A **CaretEvent** is fired each time the caret (i.e., the cursor) changes position.
    - **CaretEvent** is packaged in **javax.swing.event**
- To obtain the text currently in the text field, call **getText( )**

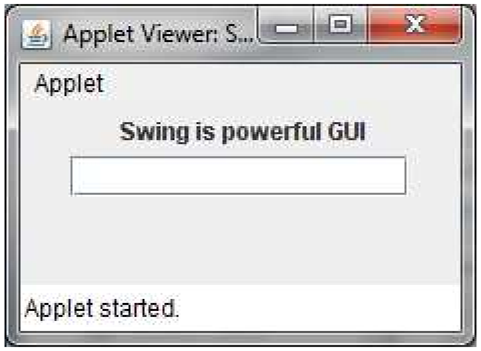
## **JTextField**(contd.)

```
// A simple Swing application.
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
/* <object code="SwingText" width=220 height=90>
</object>
*/
public class SwingText extends JApplet implements ActionListener
{    JLabel jlab;
     JTextField jtf;
```

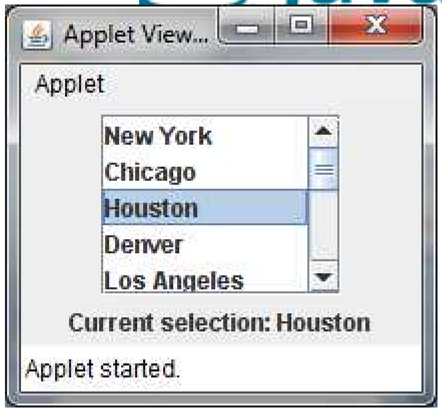
```
private void makeGUI()
    {     setLayout(new FlowLayout());
          jlab = new JLabel(" Swing is powerful GUI");
          add(jlab);
          jtf = new JTextField(15);
          jtf.addActionListener(this);
          add(jtf);
    }
public void actionPerformed(ActionEvent ae)
    {
          showStatus(jtf.getText());
    }
```

```
public void init()
    {     try {    SwingUtilities.invokeAndWait(new Runnable ()
                                          {
                                          public void run()
                                          {
                                                  makeGUI();          }
                                          });
                    } catch(Exception exc)
          { System.out.println("Can't create because of "+ exc); }
    }
}
```

COMPILE USING
javac SwingText.java
RUN
appletviewer SwingText.java


Applet Viewer: S...
Applet
Swing is powerful GUI
Applet started.

# JList



Current selection: Houston

Applet started.

- In Swing, the basic list class is called **JList.**
- **JList** provides several constructors

JList(Object[ ] *items)*

- A JList generates a **ListSelectionEvent** when the user makes or changes a selection or deselects an item. It is handled by implementing **ListSelectionListener**
- **ListSelectionListener** interface specifies only one method, called **valueChanged(),**

void **valueChanged**(ListSelectionEvent *le)*

## JList(contd.)

- We can change this behavior by calling **setSelectionMode( )**,

  void **setSelectionMode**(int *mode)*

  Here *mode can be*

  SINGLE_SELECTION

  SINGLE_INTERVAL_SELECTION

  MULTIPLE_INTERVAL_SELECTION
- We can obtain the index of the item selected from list by calling **getSelectedIndex( ) :**

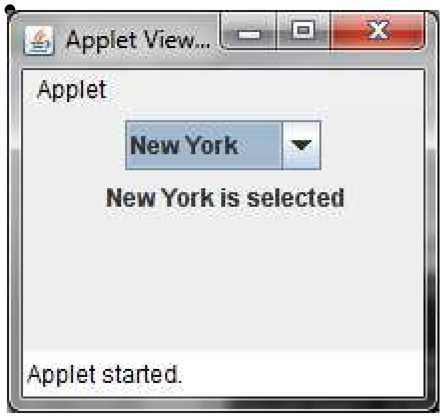
  int **getSelectedIndex**( )

  – Indexing begins at zero. So, if the first item is selected, this method will return 0. If no item is selected, –1 is returned.
- We can obtain the value associated with
- the selection by calling **getSelectedValue( ):**

  Object **getSelectedValue**( )

## JComboBox

- Swing provides a combo box (a combination of a text field and a drop-down list) through the **JComboBox class.**



New York is selected

Applet started.

- **JComboBox constructor** is:

  JComboBox(Object[ ] *items)*

- Items can also be dynamically added to the list

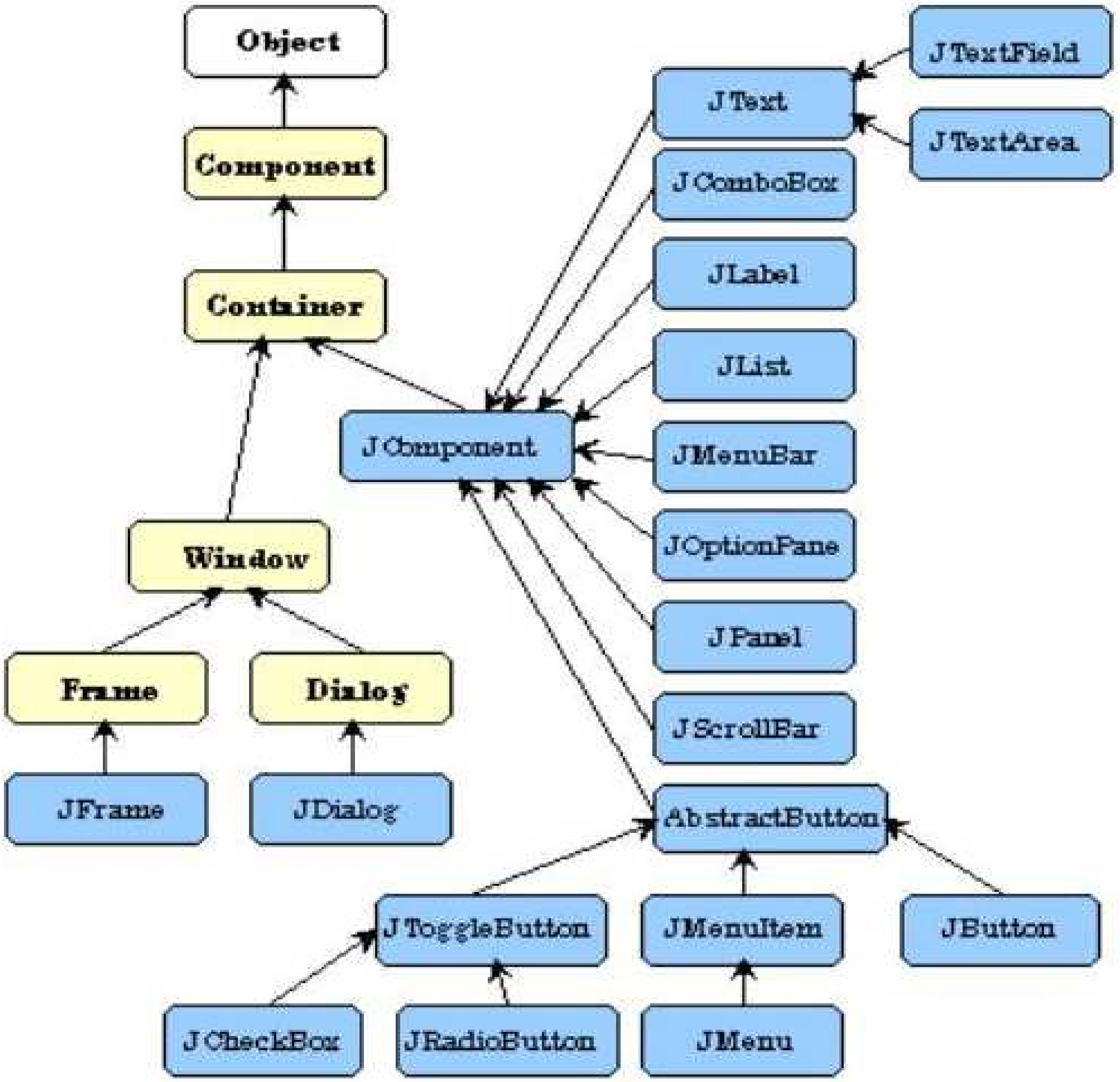
 of choices via the **addItem( ) method:**

  void **addItem**(Object *obj)*

- *To* obtain the item selected in the list is to call **getSelectedItem( ) on the combo** box.

Object **getSelectedItem**( )

# Class hierarchy of swing components



## Reference

- Herbert Schildt, Java: **The Complete Reference, 8/e, Tata McGraw Hill, 2011**.