Unit 2

AWT Controls,Layout Manager,Menu and Menubar

AWT Control Fundamentals

- The AWT supports the following types of controls:
  - Labels
  - Push buttons
  - Check boxes
  - Choice lists
  - Lists
  - Scroll bars
  - Text Editing
- These controls are subclasses of **Component**.
- Although this is not a particularly rich set of controls, it is sufficient for simple applications.
- (Note that both Swing and JavaFX provide a substantially larger, more sophisticated set of controls.)

Adding and Removing Controls

- To include a control in a window, you must add it to the window.
- To do this, you must first create an instance of the desired control and then add it to a window by calling **add( ),** which is defined by **Container**.
- The **add( )** method has several forms. The following form is the one that is used for the first part of this chapter:
  - Component add(Component compRef)
- Here, compRef is a reference to an instance of the control that you want to add. A reference to the object is returned. Once a control has been added, it will automatically be visible whenever its parent window is displayed.

- Sometimes you will want to remove a control from a window when the control is no longer needed. To do this, call **remove( )**. This method is also defined by Container. Here is one of its forms:
  - void remove(Component compRef)
- Here, compRef is a reference to the control you want to remove. You can remove all controls by calling removeAll( ).

Responding to Controls

- Except for labels, which are passive, all other controls generate events when they are accessed by the user.
- For example, when the user clicks on a push button, an event is sent that identifies the push button.
- In general, your program simply implements the appropriate interface and then registers an event listener for each control that you need to monitor.
- Once a listener has been installed, events are automatically sent to it.

Labels

- The easiest control to use is a label.
- A label is an object of type **Label**, and it contains a string, which it displays.
- Labels are passive controls that do not support any interaction with the user.
- **Label** defines the following constructors:
  - Label( ) throws HeadlessException
  - Label(String str) throws HeadlessException
  - Label(String str, int how) throws HeadlessException
- The first version creates a blank label.
- The second version creates a label that contains the string specified by str. This string is left-justified.
- The third version creates a label that contains the string specified by str using the alignment specified by how. The value of how must be one of these three constants: Label.LEFT, Label.RIGHT, or Label.CENTER.

```
import java.awt.*;
import java.awt.event.*;

public class LabelDemo {
    public LabelDemo(){
    Frame f=new Frame("Frame");
    Label firstLabel=new Label("this is the first Label",
Label.RIGHT);
    f.add(firstLabel);
    f.addWindowListener(new WindowAdapter(){
            public void windowClosing(WindowEvent we)
        {
                System.exit(0);
             }
        });
    f.setSize(300,300);
    f.setVisible(true);
```

Contd…

```
}
    public static void main(String[] args) {
        new LabelDemo();
    }
}
```

- You can set or change the text in a label by using the **setText( )** method. You can obtain the current label by calling **getText( )**. These methods are shown here:
  - void setText(String str)
  - String getText( )
- For setText( ), str specifies the new label.
- For getText( ), the current label is returned.

- You can set the alignment of the string within the label by calling **setAlignment( )**. To obtain the current alignment, call **getAlignment( ).** The methods are as follows:
  - void setAlignment(int how)
  - int getAlignment( )

Buttons

- Perhaps the most widely used control is the push button.
- A push button is a component that contains a label and that generates an event when it is pressed.
- Push buttons are objects of type **Button**. **Button** defines these two constructors:
  - Button( ) throws HeadlessException
  - Button(String str) throws HeadlessException
- The first version creates an empty button.
- The second creates a button that contains str as a label.
- After a button has been created, you can set its label by calling **setLabel( )**. You can retrieve its label by calling **getLabel( )**. These methods are as follows:
  - void setLabel(String str)
  - String getLabel( )
- Here, str becomes the new label for the button.

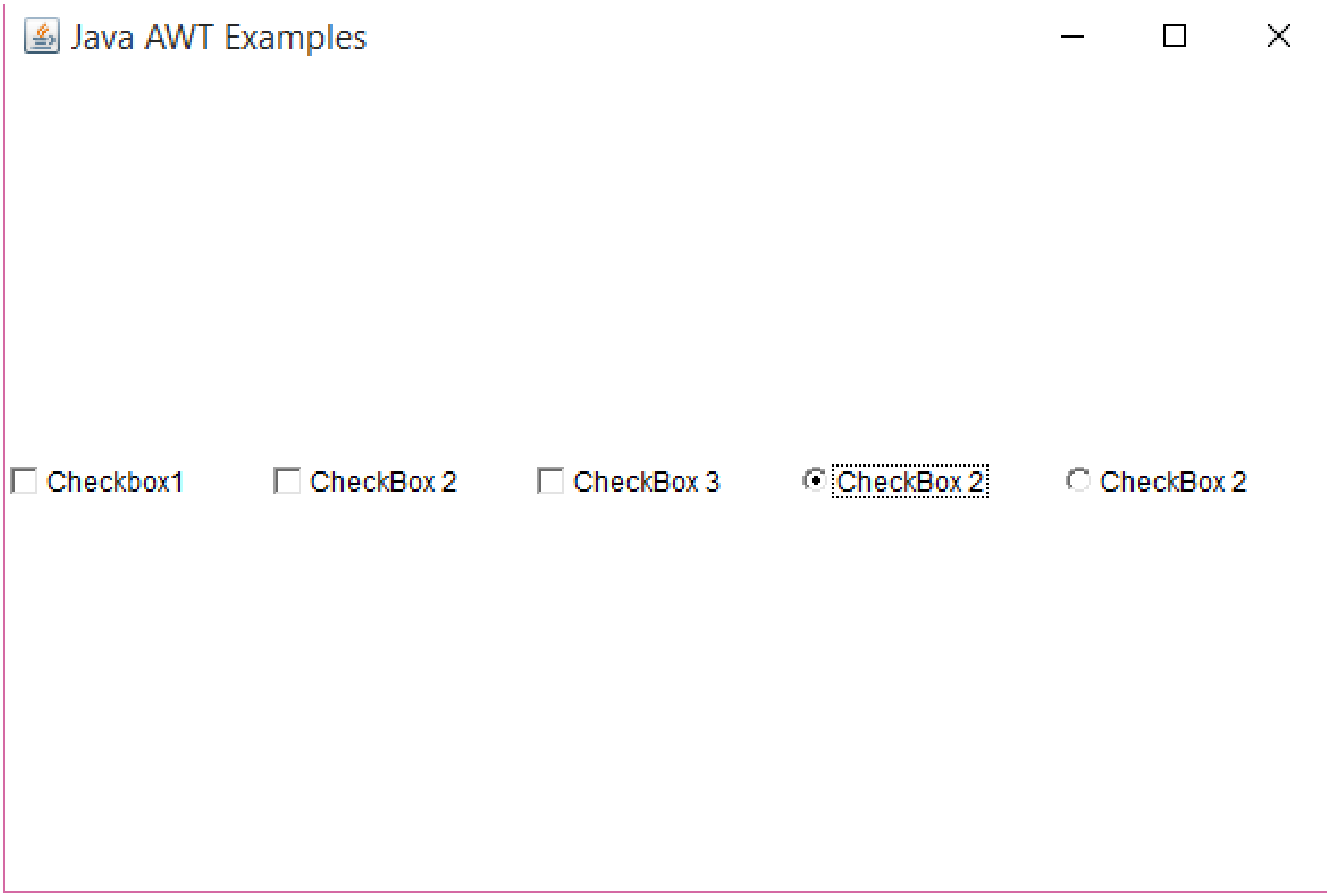
Check Boxes

- A check box is a control that is used to turn an option on or off.
- It consists of a small box that can either contain a check mark or not.
- There is a label associated with each check box that describes what option the box represents.
- You change the state of a check box by clicking on it.
- Check boxes can be used individually or as part of a group.
- Check boxes are objects of the Checkbox class.
  - Checkbox( ) throws HeadlessException
  - Checkbox(String str) throws HeadlessException
  - Checkbox(String str, boolean on) throws HeadlessException
  - Checkbox(String str, boolean on, CheckboxGroup

- The first form creates a check box whose label is initially blank. The state of the check box is unchecked.
- The second form creates a check box whose label is specified by str. The state of the check box is unchecked.
- The third form allows you to set the initial state of the check box. If on is true, the check box is initially checked; otherwise, it is cleared.
- The fourth and fifth forms create a check box whose label is specified by str and whose group is specified by cbGroup.
- If this check box is not part of a group, then cbGroup must be null.

Checkbox Example

```
import java.awt.*;
import java.awt.event.*;
public class CheckBoxDemo {
        public CheckBoxDemo() {
            Frame mf = new Frame("Java AWT Examples");
              mf.setSize(600, 400);
              mf.setLayout(new GridLayout(1,5));
              mf.addWindowListener(new WindowAdapter() {
                          public void windowClosing(WindowEvent
       windowEvent) {
                                  System.exit(0);
                                  }});
        Checkbox cb1 = new Checkbox();
        Checkbox cb2 = new Checkbox("CheckBox 2");
        Checkbox cb3 = new Checkbox("CheckBox 3", true);
        CheckboxGroup  cg=new CheckboxGroup();
        Checkbox cb4 = new Checkbox("CheckBox 2",true,cg);
        Checkbox cb5 = new Checkbox("CheckBox 2",cg,false);
        cb1.setLabel("Checkbox1");
        cb3.setState(false);
        mf.add(cb1);
        mf.add(cb2);
        mf.add(cb3);
        mf.add(cb4);
        mf.add(cb5);
        mf.setVisible(true);
    }
    public static void main(String[] args) {
        new CheckBoxDemo();
    }
}
```


Java AWT Examples
Checkbox1
CheckBox 2
CheckBox 3
CheckBox 2
CheckBox 2

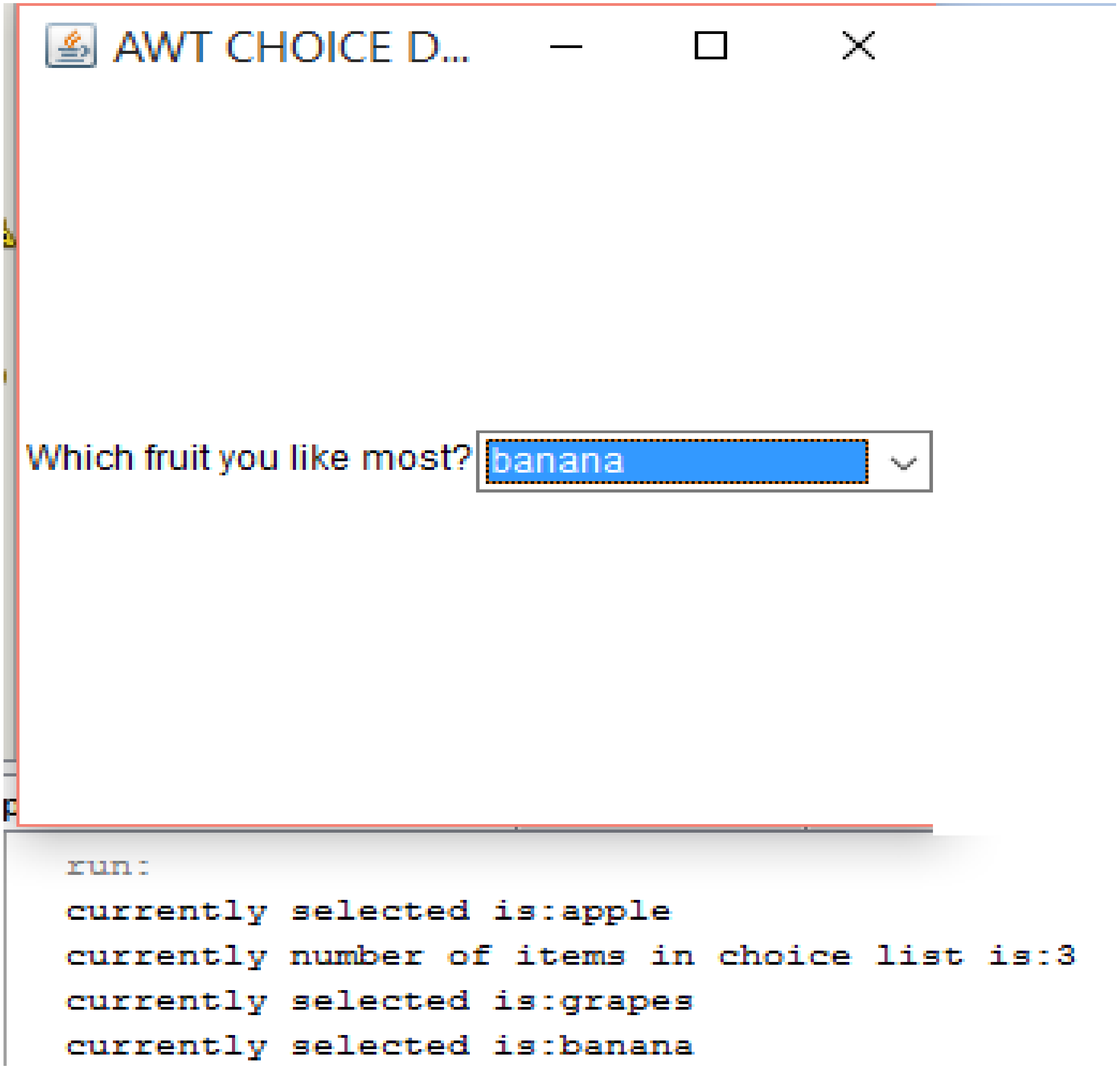
Choice Controls

- The **Choice** class is used to create a pop-up list of items from which the user may choose.
- Thus, a **Choice** control is a form of menu. When inactive, a **Choice** component takes up only enough space to show the currently selected item.
- When the user clicks on it, the whole list of choices pops up, and a new selection can be made.
- Each item in the list is a string that appears as a left-justified label in the order it is added to the Choice object.
- Choice defines only the default constructor, which creates an empty list.
- To add a selection to the list, call **add( )**. It has this general form:
  - void add(String name)
- Here, name is the name of the item being added. Items

- To determine which item is currently selected, you may call either **getSelectedItem( )** or **getSelectedIndex( )**. These methods are shown here:
  - String getSelectedItem( )
  - int getSelectedIndex( )
- The getSelectedItem( ) method returns a string containing the name of the item. getSelectedIndex( ) returns the index of the item. The first item is at index 0. By default, the first item added to the list is selected.
- To obtain the number of items in the list, call getItemCount( ).
- You can set the currently selected item using the select( ) method with either a zero-based integer index or a string that will match a name in the list. These methods are shown here:
  - int getItemCount( )
  - void select(int index)
  - void select(String name)
- Given an index, you can obtain the name associated with the item at that index by calling getItem( ), which has this general form:
  - String getItem(int index)
- Here, index specifies the index of the desired item.

Choice Example

```
import java.awt.*;
import java.awt.event.*;

public class ChoiceDemo {
    public ChoiceDemo(){
        Frame f=new Frame("AWT CHOICE DEMO");
        f.setLayout(new GridLayout(1,2));
        Label l1=new Label("Which fruit you like most?");
        Choice c1=new Choice();
        c1.add("apple");
        c1.add("banana");
        c1.add("grapes");
        f.add(l1);
        f.add(c1);

        System.out.println("currently selected
is:"+c1.getSelectedItem());
        System.out.println("currently number of items in
choice list is:"+c1.getItemCount());
        f.addWindowListener(new WindowAdapter() {
            @Override
            public void windowClosing(WindowEvent
windowEvent) {
                System.exit(0);
            }
        });
        c1.select(2);
         System.out.println("currently selected is:"+c1.getSelectedItem());

         c1.select("banana");
         System.out.println("currently selected is:"+c1.getSelectedItem());

         f.setSize(300,300);
         f.setVisible(true);
    }
    public static void main(String[] args) {
         new ChoiceDemo();
    }
}
```



```
run:
currently selected is:apple
currently number of items in choice list is:3
currently selected is:grapes
currently selected is:banana
```

Using Lists

- The List class provides a compact, multiple-choice, scrolling selection list.
- Unlike the Choice object, which shows only the single selected item in the menu, a List object can be constructed to show any number of choices in the visible window.
- It can also be created to allow multiple selections. List provides these constructors:
  - List( ) throws HeadlessException
  - List(int numRows) throws HeadlessException
  - List(int numRows, boolean multipleSelect) throws HeadlessException
- The first version creates a List control that allows only one item to be selected at any one time.

- In the second form, the value of numRows specifies the number of entries in the list that will always be visible (others can be scrolled into view as needed).
- In the third form, if multipleSelect is true, then the user may select two or more items at a time. If it is false, then only one item may be selected.
- To add a selection to the list, call add( ). It has the following two forms:
  - void add(String name)
  - void add(String name, int index)
- Here, name is the name of the item added to the list.
- The first form adds items to the end of the list. The second form adds the item at the index specified by index.
- Indexing begins at zero. You can specify –1 to add the item to the end of the list.

- For lists that allow only single selection, you can determine which item is currently selected by calling either getSelectedItem( ) or getSelectedIndex( ). These methods are shown here:
  - String getSelectedItem( )
  - int getSelectedIndex( )
- The getSelectedItem( ) method returns a string containing the name of the item. If more than one item is selected, or if no selection has yet been made, null is returned.
- getSelectedIndex( ) returns the index of the item. The first item is at index 0. If more than one item is selected, or if no selection has yet been made

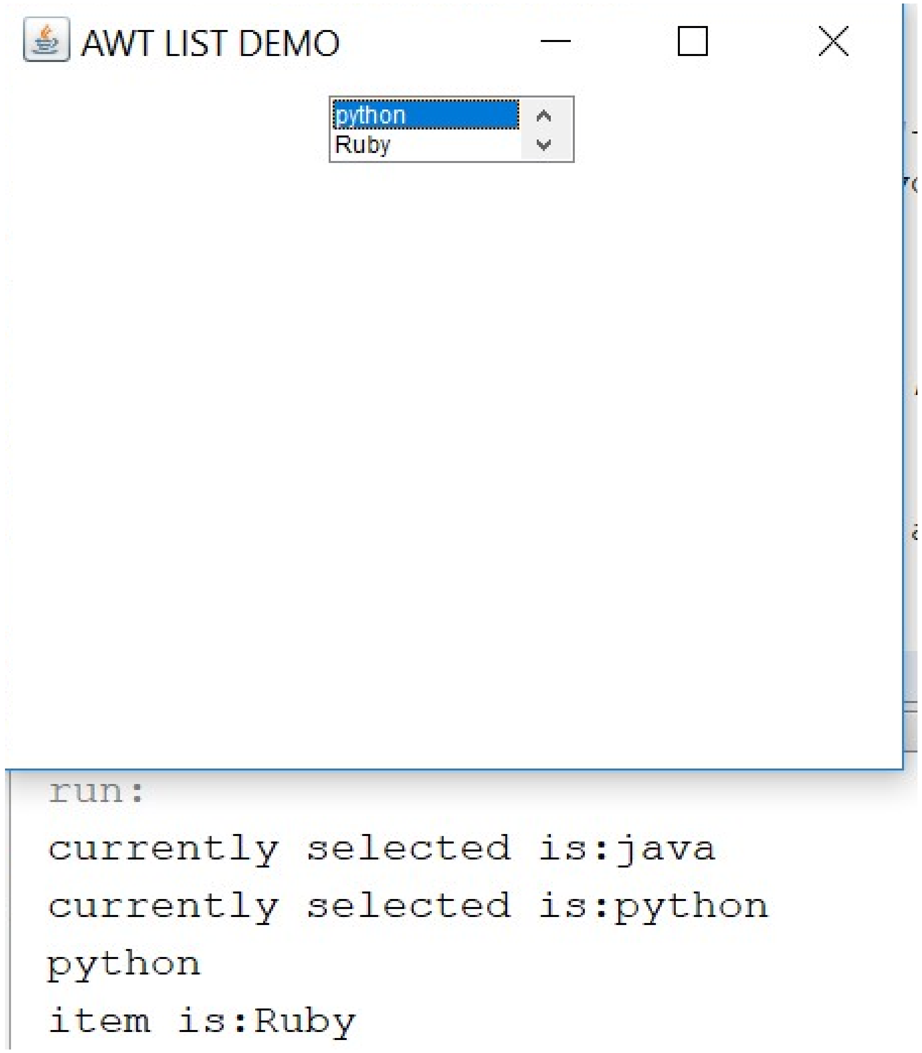
- For lists that allow multiple selection, you must use either getSelectedItems( ) or getSelectedIndexes( ), shown here, to determine the current selections:
  - String[ ] getSelectedItems( )
  - int[ ] getSelectedIndexes( )
- getSelectedItems( ) returns an array containing the names of the currently selected items. getSelectedIndexes( ) returns an array containing the indexes of the currently selected items.
- To obtain the number of items in the list, call getItemCount( ).
- You can set the currently selected item by using the select( ) method with a zero-based integer index. These methods are shown here:
  - int getItemCount( )
  - void select(int index)
- Given an index, you can obtain the name associated with the item at that index by calling getItem( ), which has this general form:
  - String getItem(int index)
- Here, index specifies the index of the desired item.

```
import java.awt.*;
import java.awt.event.*;

public class ListDemo {
    public ListDemo(){
        Frame f=new Frame("AWT LIST DEMO");
        f.addWindowListener(new WindowAdapter() {
            @Override
            public void windowClosing(WindowEvent windowEvent)
{
                System.exit(0);
            }
        });
```

```
       List lst=new List(2,false);
        lst.add("java");
        lst.add("C programming");
        lst.add("web Technology");
        lst.add("python");
        lst.add("Ruby");
        lst.add("JavaScript");
        lst.add("Object Oriented");
        lst.add("Dot Net");
        lst.add("Networking",2);
        lst.add("AJAX",7);

        lst.select(0);
        System.out.println("currently selected
is:"+lst.getSelectedItem());
```

```
        lst.select(4);
        System.out.println("currently selected 
is:"+lst.getSelectedItem());
                /* String []s=lst.getSelectedItems();
        for(String z:s){
            System.out.println(z);
        }*/
        System.out.println("item is:"+lst.getItem(5));
        Panel p=new Panel(new FlowLayout());
        p.add(lst);
        f.add(p);
        f.setSize(200,200);
        f.setLocationRelativeTo(null);
        f.setVisible(true);
    }
    public static void main(String[] args) {
        new ListDemo();
    }
```

Output



```
run:
currently selected is:java
currently selected is:python
python
item is:Ruby
```

Managing Scroll Bars

- Scroll bars are used to select continuous values between a specified minimum and maximum.
- Scroll bars may be oriented horizontally or vertically.
- A scroll bar is actually a composite of several individual parts. Each end has an arrow that you can click to move the current value of the scroll bar one unit in the direction of the arrow.
- The current value of the scroll bar relative to its minimum and maximum values is indicated by the slider box (or thumb) for the scroll bar.
- The slider box can be dragged by the user to a new position. The scroll bar will then reflect this value

- In the background space on either side of the thumb, the user can click to cause the thumb to jump in that direction by some increment larger than 1.
- Typically, this action translates into some form of page up and page down. Scroll bars are encapsulated by the Scrollbar class.
- Scrollbar defines the following constructors:
  - Scrollbar() throws Headless Exception
  - Scrollbar(int style) throws Headless Exception
  - Scrollbar(int style, int initialValue, int thumbSize,int min,int max) throws Headless Exception

- The first form creates a vertical scroll bar.
- The second and third forms allow you to specify the orientation of the scroll bar.
- If style is Scrollbar.VERTICAL, a vertical scroll bar is created.
- If style is Scrollbar.HORIZONTAL, the scroll bar is horizontal.
- In the third form of the constructor, the initial value of the scroll bar is passed in initialValue.
- The number of units represented by the height of the thumb is passed in thumbSize.
- The minimum and maximum values for the scroll bar are specified by min and max.

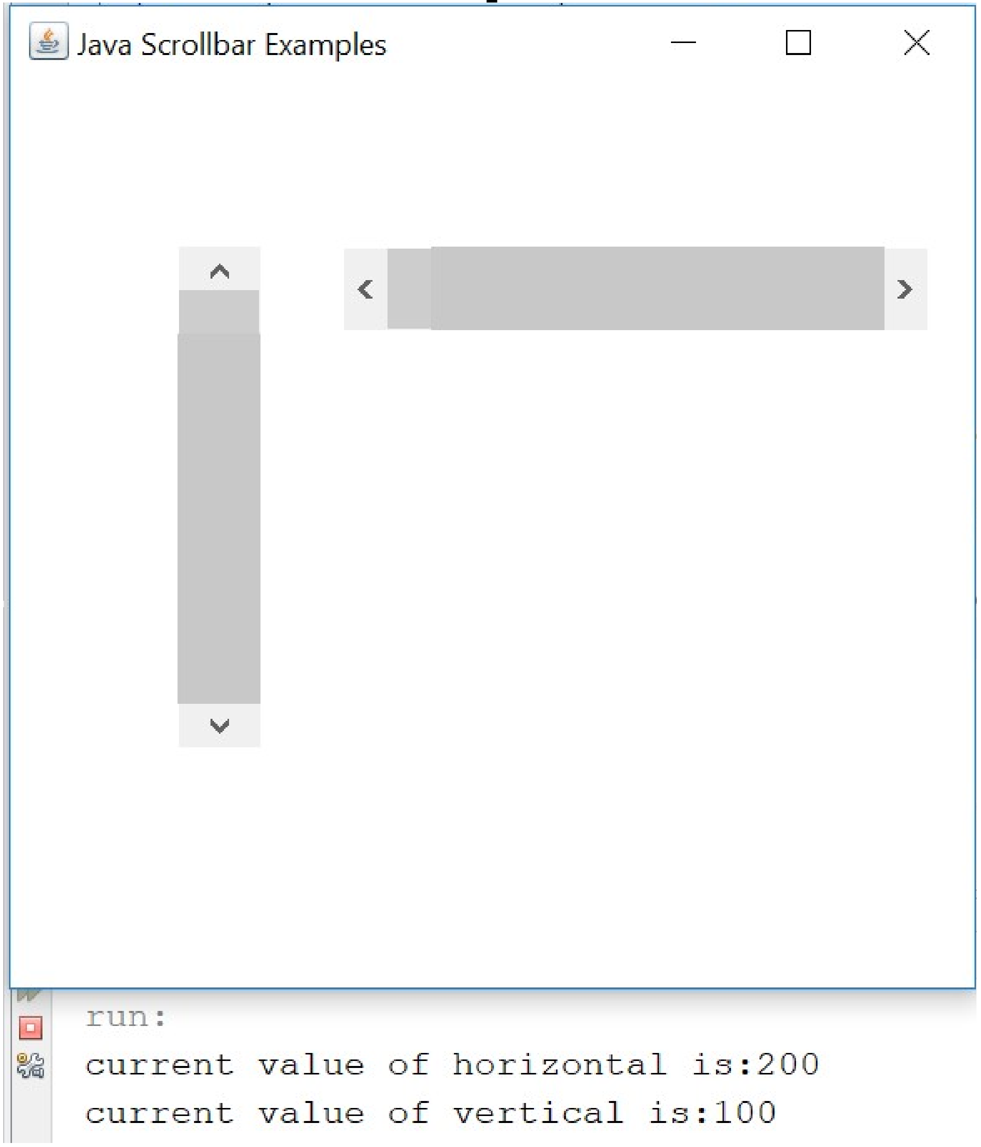
- If you construct a scroll bar by using one of the first two constructors, then you need to set its parameters by using setValues( ), shown here, before it can be used:
  - void setValues(int initialValue, int thumbSize, int min, int max)
- The parameters have the same meaning as they have in the third constructor just described.
- To obtain the current value of the scroll bar, call getValue( ). It returns the current setting.
- To set the current value, call setValue( ). These methods are as follows:
  - int getValue( )
  - void setValue(int newValue)
- Here, newValue specifies the new value for the scroll bar.
- When you set a value, the slider box inside the scroll bar will be positioned to reflect the new value.

- You can also retrieve the minimum and maximum values via getMinimum( ) and getMaximum( ), shown here:
  - int getMinimum( )
  - int getMaximum( )
- They return the requested quantity.
- By default, 1 is the increment added to or subtracted from the scroll bar each time it is scrolled up or down one line. You can change this increment by calling setUnitIncrement( ).
- By default, page-up and page-down increments are 10. You can change this value by calling setBlockIncrement( ). These methods are shown here:
  - void setUnitIncrement(int newIncr)
  - void setBlockIncrement(int newIncr)

Example 1

```
import java.awt.*;
import java.awt.event.*;

public class ScrollBarDemo {

    public ScrollBarDemo() {
        Frame f = new Frame("Java Scrollbar Examples");
        f.setSize(600, 400);
         Scrollbar vsb = new Scrollbar(Scrollbar.VERTICAL,1,5,100,200);
        Scrollbar hsb = new Scrollbar(Scrollbar.HORIZONTAL);
        hsb.setValues(1, 5, 200, 200);
        System.out.println("current value of horizontal
is:"+hsb.getValue());
        System.out.println("current value of vertical is:"+vsb.getValue());
        Panel p=new Panel();
        p.setLayout(null);
        vsb.setBounds(100, 100, 50, 300);
        hsb.setBounds(200, 100, 350, 50);
        p.add(hsb);
        p.add(vsb);

        f.add(p);
        f.setSize(400, 400);
        f.setLocationRelativeTo(null);
        f.setVisible(true);
    }
    public static void main(String[] args) {
        new ScrollBarDemo();
    }
}
```

Output



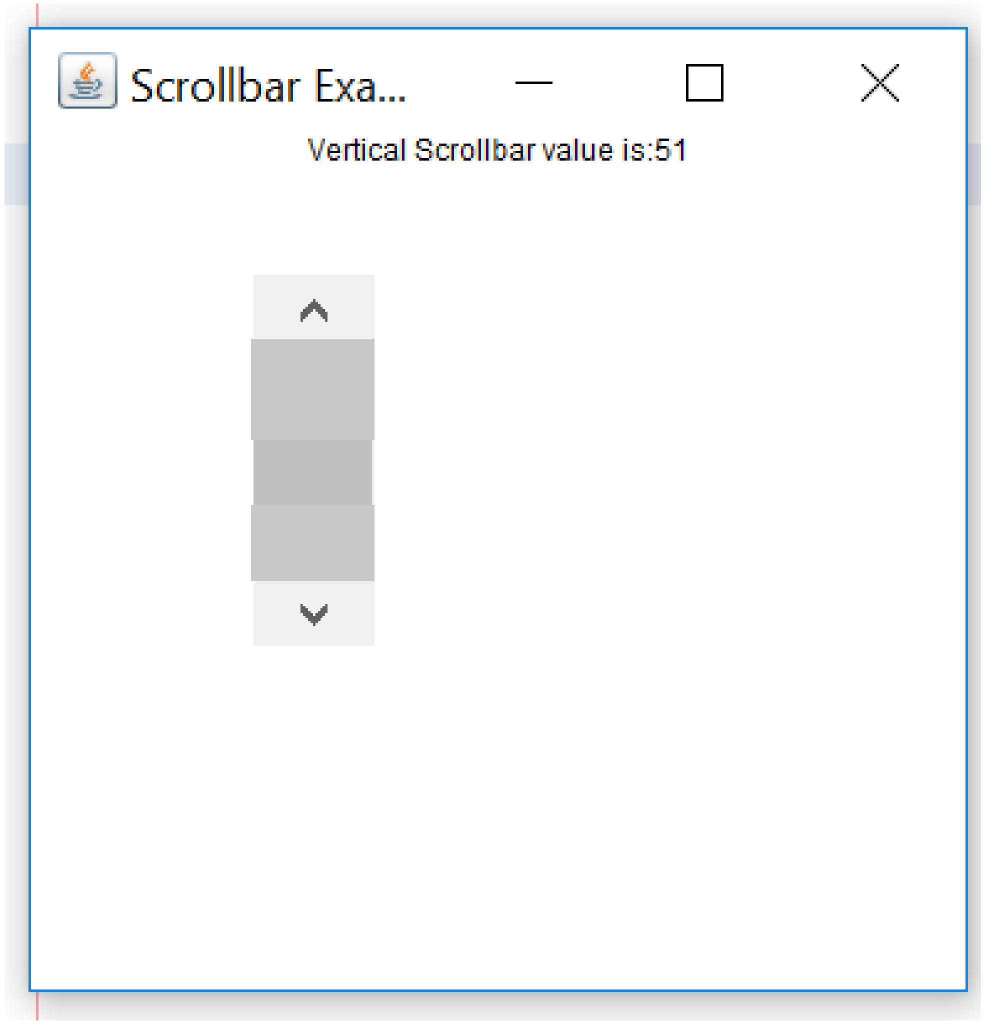
```
run:
current value of horizontal is:200
current value of vertical is:100
```

Example 2

```
import java.awt.*;
import java.awt.event.*;

public class ScrollBarExample {

    ScrollBarExample() {
        Frame f = new Frame("Scrollbar Example");
        final Label l = new Label();
        l.setAlignment(Label.CENTER);
        l.setSize(400, 100);
        final Scrollbar s = new Scrollbar();
        s.setBounds(100, 100, 50, 150);
        f.add(s);
        f.add(l);
        f.setSize(400, 400);
        f.setLayout(null);
        f.setVisible(true);
```

```
 s.addAdjustmentListener(new AdjustmentListener() {
           public void adjustmentValueChanged(AdjustmentEvent
e) {
               l.setText("Vertical Scrollbar value is:" + s.getValue());
           }
       });
   }

   public static void main(String args[]) {
       new ScrollBarExample();
   }

}
```

Output



```
Vertical Scrollbar value is:51
```

Using a TextField

- The TextField class implements a single-line text-entry area, usually called an edit control.
- Text fields allow the user to enter strings and to edit the text using the arrow keys, cut and paste keys, and mouse selections.
- TextField is a subclass of TextComponent. TextField defines the following constructors:
  - TextField( ) throws HeadlessException
  - TextField(int numChars) throws HeadlessException
  - TextField(String str) throws HeadlessException
  - TextField(String str, int numChars) throws HeadlessException
- The first version creates a default text field. The second form creates a text field that is numChars characters wide. The third form initializes the text field with the string contained in str. The fourth form initializes a text field and sets its width.

- TextField (and its superclass TextComponent) provides several methods that allow you to utilize a text field. To obtain the string currently contained in the text field, call getText( ). To set the text, call setText( ). These methods are as follows:
  - String getText( )
  - void setText(String str)
- Here, str is the new string.
- The user can select a portion of the text in a text field. Also, you can select a portion of text under program control by using select( ). Your program can obtain the currently selected text by calling getSelectedText( ). These methods are shown here:
  - String getSelectedText( )
  - void select(int startIndex, int endIndex)
- getSelectedText( ) returns the selected text. The select( ) method selects the characters beginning at startIndex and ending at endIndex –1.

- You can control whether the contents of a text field may be modified by the user by calling setEditable( ). You can determine editability by calling isEditable( ). These methods are shown here:
  - boolean isEditable( )
  - void setEditable(boolean canEdit)
- isEditable( ) returns true if the text may be changed and false if not. In setEditable( ), if canEdit is true, the text may be changed. If it is false, the text cannot be altered.
- There may be times when you will want the user to enter text that is not displayed, such as a password.
- You can disable the echoing of the characters as they are typed by calling setEchoChar( ).
- This method specifies a single character that the TextField will display when characters are entered (thus, the actual characters typed will not be shown).

- You can check a text field to see if it is in this mode with the echoCharIsSet( ) method. You can retrieve the echo character by calling the getEchoChar( ) method. These methods are as follows:
  - void setEchoChar(char ch)
  - boolean echoCharIsSet( )
  - char getEchoChar( )
- Here, ch specifies the character to be echoed. If ch is zero, then normal echoing is restored.

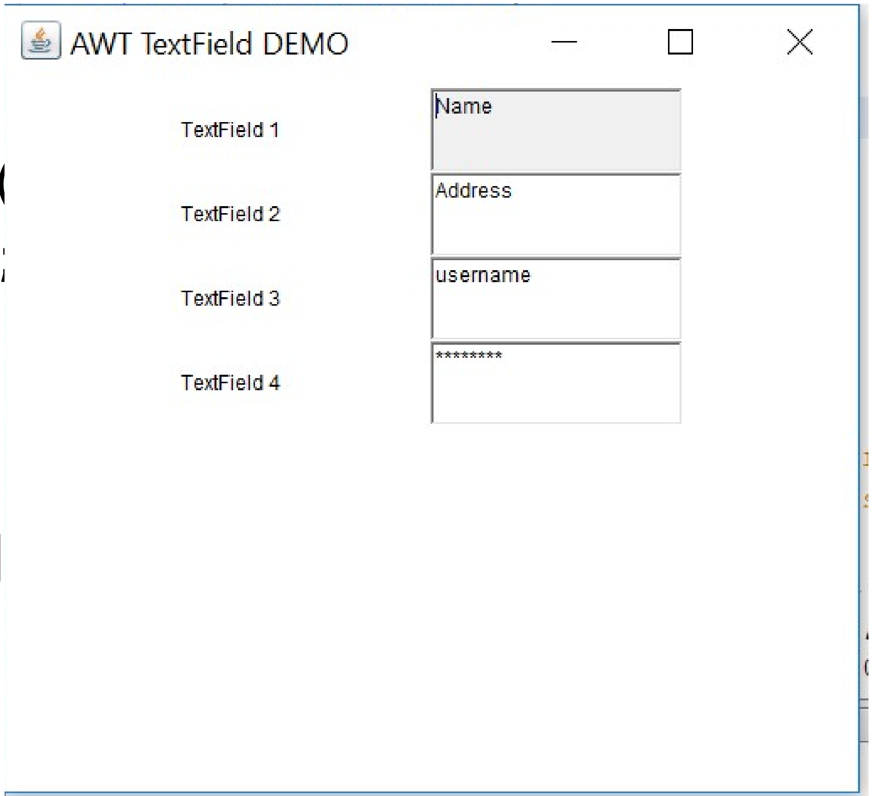
Example

```
import java.awt.*;
import java.awt.event.*;

public class TextFieldDemo {

    public TextFieldDemo() {
        Label l1=new Label("TextField 1");
        Label l2=new Label("TextField 2");
        Label l3=new Label("TextField 3");
        Label l4=new Label("TextField 4");
        TextField tx1 = new TextField();
        TextField tx2 = new TextField(10);
        TextField tx3 = new TextField("username", 20);
        TextField tx4 = new TextField("password", 20);
        Frame f = new Frame();
```

```
Panel p=new Panel(new GridLayout(4,2));
p.setPreferredSize(new Dimension(300,200));

p.add(l1);
p.add(tx1);
p.add(l2);
p.add(tx2);
p.add(l3);
p.add(tx3);
p.add(l4);
p.add(tx4);

f.add(p);
tx1.setText("Name");
tx2.setText("Address");
```

```
System.out.println("first textfield is:" + tx1.getText());
        System.out.println("first textfield is:" + tx2.getText());

        tx3.select(2, 5);
        System.out.println("selected text of textfield 3 is:" +
tx3.getSelectedText());

        tx1.setEditable(false);
        System.out.println(tx1.isEditable() ? "editable" : "not
editable");

        tx4.setEchoChar('*');
        System.out.println("echo text is:" + tx4.getEchoChar());

        System.out.println(tx4.echoCharIsSet()? "echo set" : "echo
not set");
```

```
f.setSize(400, 400);
        f.setLayout(new FlowLayout(
        f.setLocationRelativeTo(null);
        f.setVisible(true);
    }

    public static void main(String[]
        new TextFieldDemo();
    }

}
```



```
first textfield is:Name
first textfield is:Address
selected text of textfield 3 is:ern
not editable
echo text is:*
echo set
```

Using a TextArea

- Sometimes a single line of text input is not enough for a given task. To handle these situations, the AWT includes a simple multiline editor called TextArea.
- Following are the constructors for TextArea:
  - TextArea( ) throws HeadlessException
  - TextArea(int numLines, int numChars) throws HeadlessException
  - TextArea(String str) throws HeadlessException
  - TextArea(String str, int numLines, int numChars) throws HeadlessException
  - TextArea(String str, int numLines, int numChars, int sBars) throws HeadlessException

- Here, numLines specifies the height, in lines, of the text area, and numChars specifies its width, in characters.
- Initial text can be specified by str.
- In the fifth form, you can specify the scroll bars that you want the control to have. sBars must be one of these values:

SCROLLBARS_BOTH	**SCROLLBARS_NONE**
SCROLLBARS_HORIZONTAL_ONLY	**SCROLLBARS_VERTICAL_ONLY**

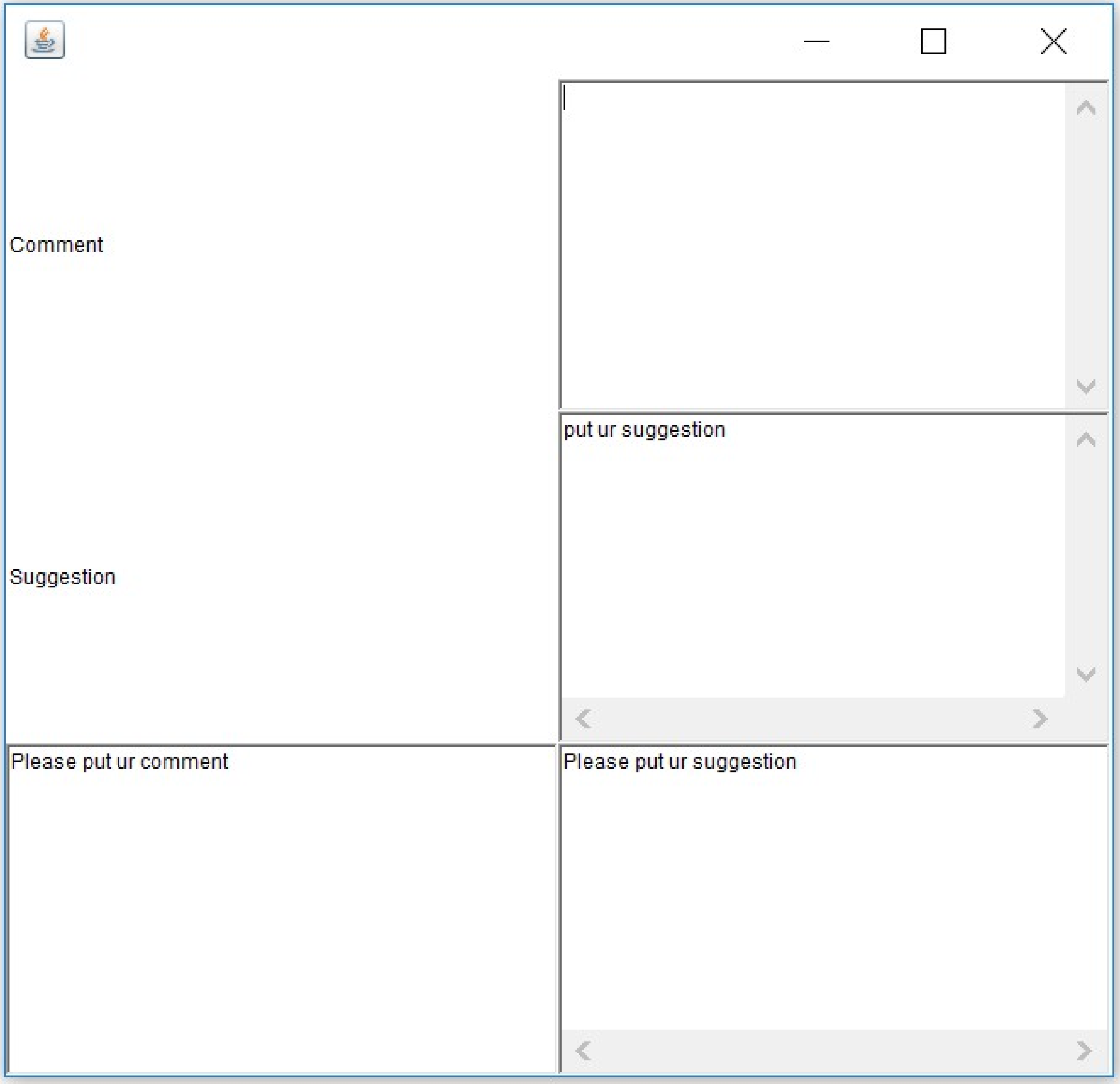
- TextArea is a subclass of TextComponent. Therefore, it supports the getText( ), setText( ), getSelectedText( ), select( ), isEditable( ), and setEditable( ) methods described in the preceding section.

- TextArea adds the following editing methods:
  - void append(String str)
  - void insert(String str, int index)
  - void replaceRange(String str, int startIndex, int endIndex)
- The append( ) method appends the string specified by str to the end of the current text.
- insert( ) inserts the string passed in str at the specified index.
- To replace text, call replaceRange( ). It replaces the characters from startIndex to endIndex–1, with the replacement text passed in str.

```
import java.awt.*;
import java.awt.event.*;
public class TextAreaDemo extends Frame {
    public TextAreaDemo() {
        setSize(400, 400);
        setLayout(new GridLayout(3, 2));
        TextArea ta1 = new TextArea();
        TextArea ta2 = new TextArea("put ur suggestion");
        TextArea ta3 = new TextArea("Please put ur
comment", 4, 4, TextArea.SCROLLBARS_NONE);

        TextArea ta4 = new TextArea("Please put ur
suggestion", 6, 9,
TextArea.SCROLLBARS_HORIZONTAL_ONLY);
```

```
        Label comment = new Label("Comment");
        Label suggestion = new Label("Suggestion");

        add(comment);
        add(ta1);
        add(suggestion);
        add(ta2);
        add(ta3);
        add(ta4);
        setVisible(true);
    }
    public static void main(String[] args) {
        new TextAreaDemo();
    }
}
```

Output



Comment

Suggestion

put ur suggestion

Please put ur comment

Please put ur suggestion

Understanding Layout Managers

- All of the components that we have shown so far have been positioned by the default layout manager.
- As we mentioned, a layout manager automatically arranges your controls within a window by using some type of algorithm.
- The layout manager is set by the setLayout( ) method. If no call to setLayout( ) is made, then the default layout manager is used.
- Whenever a container is resized (or sized for the first time), the layout manager is used to position each of the components within it.
- The setLayout( ) method has the following general form:
  - void setLayout(LayoutManager layoutObj)

- If you wish to disable the layout manager and position components manually, pass null for layoutObj.
- If you do this, you will need to determine the shape and position of each component manually, using the setBounds( ) method defined by Component.
- Each layout manager keeps track of a list of components that are stored by their names.
- The layout manager is notified each time you add a component to a container.

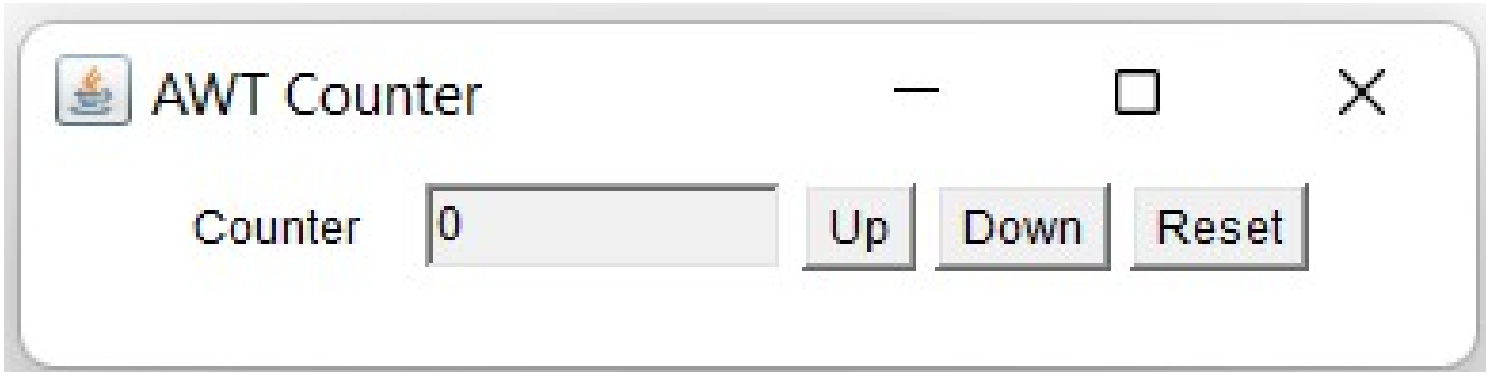
FlowLayout

- FlowLayout is the default layout manager.
- FlowLayout implements a simple layout style, which is similar to how words flow in a text editor.
- The direction of the layout is governed by the container’s component orientation property, which, by default, is left to right, top to bottom.
- Therefore, by default, components are laid out line-by-line beginning at the upper-left corner. In all cases, when a line is filled, layout advances to the next line.
- A small space is left between each component, above and below, as well as left and right. Here are the constructors for FlowLayout:
  - FlowLayout( )
  - FlowLayout(int how)
  - FlowLayout(int how, int horz, int vert)

- The first form creates the default layout, which centers components and leaves five pixels of space between each component.
- The second form lets you specify how each line is aligned. Valid values for how are as follows:
  - FlowLayout.LEFT
  - FlowLayout.CENTER
  - FlowLayout.RIGHT
  - FlowLayout.LEADING
  - FlowLayout.TRAILING
- These values specify left, center, right, leading edge, and trailing edge alignment, respectively.
- The third constructor allows you to specify the horizontal and vertical space left between components in horz and vert, respectively.

Example

```
import java.awt.*;

public class FlowLayoutExample extends Frame {
    Label l1;
    TextField t1;
    Button b1,b2,b3;
    int count=0;
    public FlowLayoutExample(){
      setLayout(new FlowLayout());
      l1 = new Label("Counter");
      add(l1);
      t1 = new TextField(count + "", 10);
      t1.setEditable(false);
      add(t1);
      b1 = new Button("Up");
      b2 = new Button("Down");
      b3 = new Button("Reset");
      add(b1);
      add(b2);
      add(b3);

      setSize(400, 100);
      setTitle("AWT Counter");
      setVisible(true);
    }
    public static void main(String[] args) {
        new FlowLayoutExample();
    }
}
```


AWT Counter
Counter
0
Up
Down
Reset

BorderLayout

- The BorderLayout class implements a common layout style for top-level windows.
- It has four narrow, fixed-width components at the edges and one large area in the center.
- The four sides are referred to as north, south, east, and west. The middle area is called the center.
- Here are the constructors defined by BorderLayout:
  - BorderLayout( )
  - BorderLayout(int horz, int vert)
- The first form creates a default border layout.
- The second allows you to specify the horizontal and vertical space left between components in horz and vert, respectively.

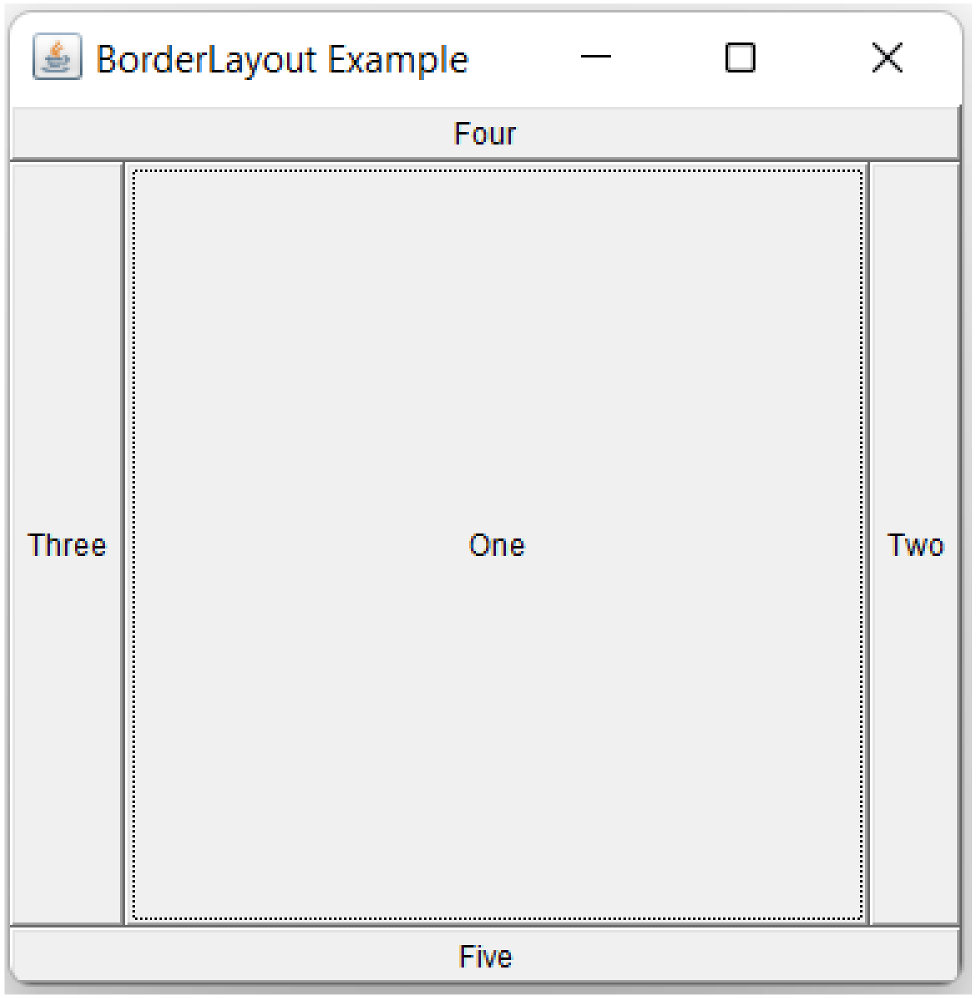
- BorderLayout defines the following constants that specify the regions:

BorderLayout.CENTER	BorderLayout.SOUTH	BorderLayout.EAST
BorderLayout.WEST	BorderLayout.NORTH	

- When adding components, you will use these constants with the following form of add( ), which is defined by Container:
- void add(Component compRef, Object region)
- Here, compRef is a reference to the component to be added, and region specifies where the component will be added.

Example

```
import java.awt.*;

public class BorderLayoutExample {

    public BorderLayoutExample() {
        Frame f = new Frame("BorderLayout Example");
        f.setLayout(new BorderLayout());

        Button b1 = new Button("One");
        Button b2 = new Button("Two");
        Button b3 = new Button("Three");
        Button b4 = new Button("Four");
        Button b5 = new Button("Five");
        f.add(b1, BorderLayout.CENTER);
         f.add(b2, BorderLayout.EAST);
         f.add(b3, BorderLayout.WEST);
         f.add(b4, BorderLayout.NORTH);
         f.add(b5, BorderLayout.SOUTH);
         f.setSize(400, 400);
         f.setVisible(true);

    }
    public static void main(String[] args) {
        new BorderLayoutExample();
    }
}
```

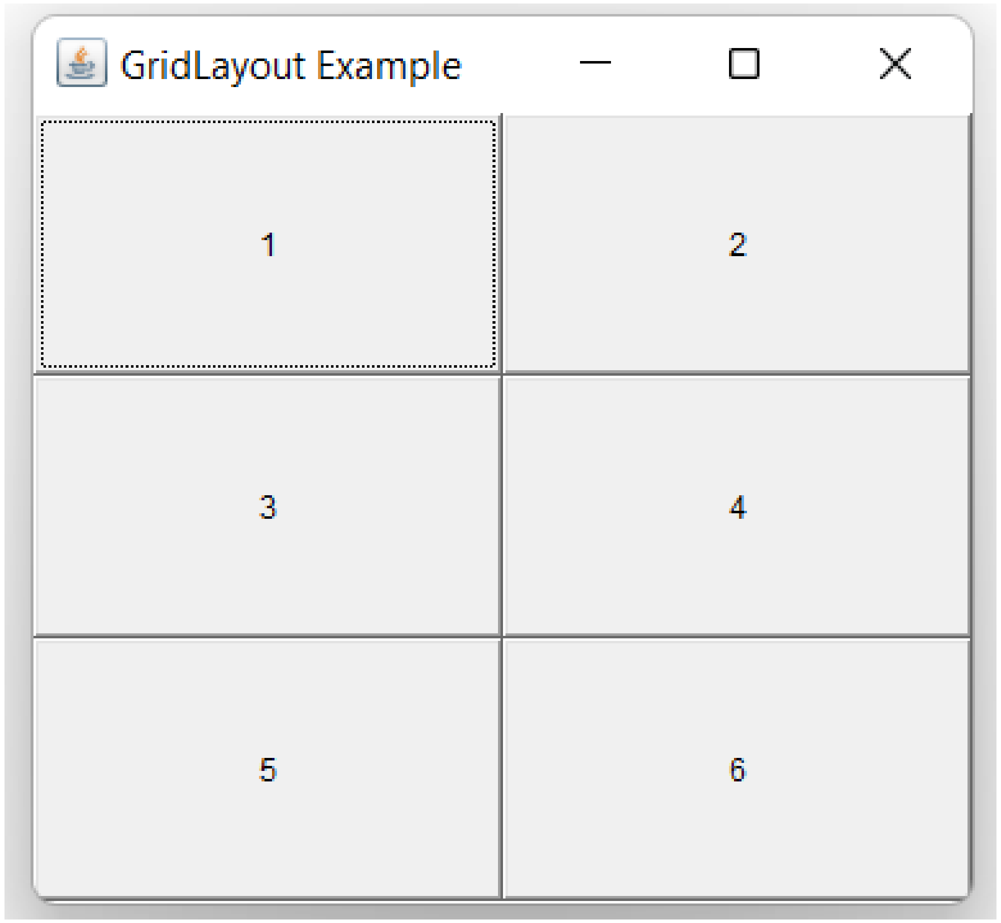

BorderLayout Example
Four
Three
One
Two
Five

GridLayout

- GridLayout lays out components in a two-dimensional grid.
- When you instantiate a GridLayout, you define the number of rows and columns.
- The constructors supported by GridLayout are shown here:
  - GridLayout( )
  - GridLayout(int numRows, int numColumns)
  - GridLayout(int numRows, int numColumns, int horz, int vert)
- The first form creates a single-column grid layout.
- The second form creates a grid layout with the specified number of rows and columns.
- The third form allows you to specify the horizontal and vertical space left between components in horz and vert, respectively.
- Either numRows or numColumns can be zero. Specifying numRows as zero allows for unlimited-length columns. Specifying numColumns as zero allows for unlimited-length rows.

Example

```
import java.awt.*;
public class GridLayoutExample {

    public GridLayoutExample() {
        Frame f = new Frame("GridLayout Example");
        f.setLayout(new GridLayout(3, 2));
        Button b1 = new Button("1");
        Button b2 = new Button("2");
        Button b3 = new Button("3");
        Button b4 = new Button("4");
        Button b5 = new Button("5");
        Button b6 = new Button("6");
        f.add(b1);
        f.add(b2);
        f.add(b3);
        f.add(b4);
        f.add(b5);
        f.add(b6);

        f.setSize(300, 300);
        f.setLocationRelativeTo(null);
        f.setVisible(true);
    }
    public static void main(String[] args) {
        new GridLayoutExample();
    }
}
```


GridLayout Example
1
2
3
4
5
6

CardLayout

- The CardLayout class is unique among the other layout managers in that it stores several different layouts.
- Each layout can be thought of as being on a separate index card in a deck that can be shuffled so that any card is on top at a given time.
- This can be useful for user interfaces with optional components that can be dynamically enabled and disabled upon user input.
- You can prepare the other layouts and have them hidden, ready to be activated when needed.
- CardLayout provides these two constructors:
  - CardLayout( )
  - CardLayout(int horz, int vert)
- The first form creates a default card layout.
- The second form allows you to specify the horizontal and vertical space left between components in horz and vert, respectively.

- When card panels are added to a panel, they are usually given a name.
- Thus, most of the time, you will use this form of add( ) when adding cards to a panel:
  - void add(Component panelRef, Object name)
- Here, name is a string that specifies the name of the card whoseAfter you have created a deck, your program activates a card by calling one of the following methods defined by CardLayout:
  - void first(Container deck)
  - void last(Container deck)
  - void next(Container deck)
  - void previous(Container deck)
  - void show(Container deck, String cardName)
- panel is specified by panelRef.

```
import java.awt.*;
import java.awt.event.*;
public class CardLayoutDemo extends Frame implements
ActionListener{
CardLayout card;
Button b1,b2,b3;
    CardLayoutDemo(){
        card=new CardLayout(40,30);
//create CardLayout object with 40 hor space and 30 ver space

        setLayout(card);
        b1=new Button("Apple");
        b2=new Button("Boy");
        b3=new Button("Cat");
        b1.addActionListener(this);
        b2.addActionListener(this);
        b3.addActionListener(this);
Panel p=new Panel();
Panel p1=new Panel();
Panel p2=new Panel();

p.add(b1);
p1.add(b2);
p2.add(b3);

add(p);
add(p1);
add(p2);
setSize(400, 400);
setVisible(true);
     }
```

```
 public void actionPerformed(ActionEvent e) {
    card.next(this);
    }

    public static void main(String[] args) {
        CardLayoutDemo cl=new CardLayoutDemo();

    }
}
```

GridBagLayout

- The Java GridBagLayout class is used to align components vertically, horizontally or along their baseline.
- The components may not be of the same size.
- Each GridBagLayout object maintains a dynamic, rectangular grid of cells.
- Each component occupies one or more cells known as its display area.
- Each component associates an instance of GridBagConstraints.
- With the help of the constraints object, we arrange the component's display area on the grid.

- Constructor
- **GridBagLayout():** The parameterless constructor is used to create a grid bag layout manager.

Example

```
import java.awt.*;

public class GridBagDemo {
    public GridBagDemo(){
        Frame f=new Frame("GridBagDemo");
        Panel p=new Panel();
        p.setLayout(new GridBagLayout());

        Button b1=new Button("Hello");
        Button b2=new Button("Click");
        Button b3=new Button("Nepal");
        Button b4=new Button("Java");
```

```
      GridBagConstraints gbc=new
GridBagConstraints();
        gbc.gridx=0;//column 0
        gbc.gridy=0;//row 0
        gbc.ipadx = 15;
        gbc.ipady = 50;
        p.add(b1,gbc);

        gbc.gridx = 1;//column 1
        gbc.gridy=0;//row 0
        gbc.ipadx = 60;
        gbc.ipady = 40;
        p.add(b2,gbc);
```

```
gbc.gridx=0;//column 0
gbc.gridy=1;//row 1
gbc.ipadx = 60;
gbc.ipady = 40;
p.add(b3,gbc);

gbc.gridx=1;//column 0
gbc.gridy=1;//row 1
gbc.ipadx = 80;
gbc.ipady = 60;
p.add(b4,gbc);
```

```
        f.add(p);
        f.setSize(400,400);
        f.setLocationRelativeTo(null);
        f.setVisible(true);
    }
    public static void main(String[] args) {
        new GridBagDemo();
    }
}
```

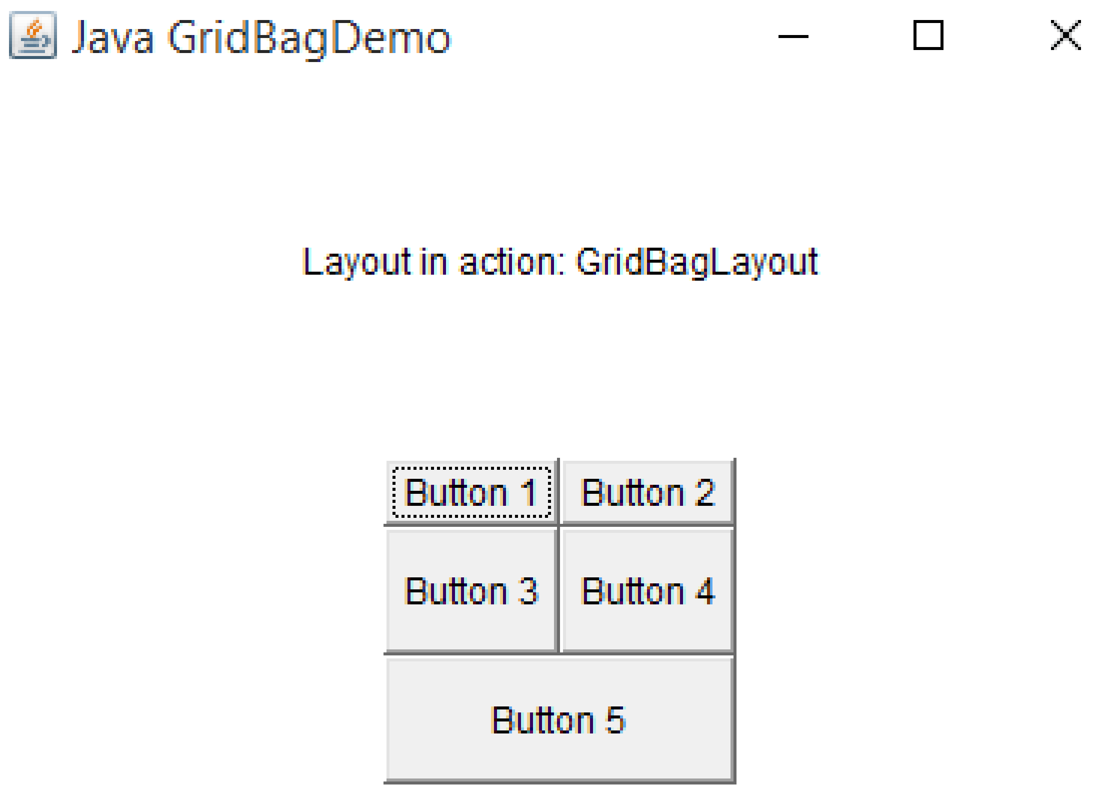
Example 2

```
import java.awt.*;

public class GridBagExample {

    Frame f;
    Panel p;
    Button b1,b2,b3,b4,b5;

    public GridBagExample() {
        f = new Frame("Java GridBagDemo");
        f.setSize(400, 400);
        p = new Panel();
```

```
b1=new Button("Button 1");
b2=new Button("Button 2");
b3=new Button("Button 3");
b4=new Button("Button 4");
b5=new Button("Button 5");
p.setLayout(new GridBagLayout());
GridBagConstraints gbc = new GridBagConstraints();
gbc.fill = GridBagConstraints.HORIZONTAL;
gbc.gridx = 0;
gbc.gridy = 0;
p.add(b1, gbc);
```

```
gbc.gridx = 1;
gbc.gridy = 0;
p.add(b2, gbc);

gbc.fill = GridBagConstraints.HORIZONTAL;
gbc.ipady = 20;
gbc.gridx = 0;
gbc.gridy = 1;
p.add(b3, gbc);

gbc.gridx = 1;
gbc.gridy = 1;
p.add(b4, gbc);
```

```
        gbc.gridx = 0;
        gbc.gridy = 2;
        gbc.fill = GridBagConstraints.HORIZONTAL;
        gbc.gridwidth = 2;
        p.add(b5, gbc);

        f.add(p);
        f.add(p);
        f.setVisible(true);
    }
    public static void main(String[] args) {
        new GridBagExample();
    }
}
```


Java GridBagDemo
Layout in action: GridBagLayout
Button 1
Button 2
Button 3
Button 4
Button 5

Menus and Menu Bars

- A top-level window can have a menu bar associated with it.
- A menu bar displays a list of top-level menu choices. Each choice is associated with a drop-down menu.
- This concept is implemented in the AWT by the following classes: MenuBar, Menu, and MenuItem.
- In general, a menu bar contains one or more Menu objects. Each Menu object contains a list of MenuItem objects. Each MenuItem object represents something that can be selected by the user.
- Since Menu is a subclass of MenuItem, a hierarchy of nested submenus can be created. It is also possible to include checkable menu items. These are menu options of type CheckboxMenuItem and will have a check mark next to them when they are selected.
- To create a menu bar, first create an instance of

- Following are the constructors for Menu:
- Menu( ) throws HeadlessException
- Menu(String optionName) throws HeadlessException
- Menu(String optionName, boolean removable) throws HeadlessException
- Here, optionName specifies the name of the menu selection. If removable is true, the menu can be removed and allowed to float free. Otherwise, it will remain attached to the menu bar. (Removable menus are implementation-dependent.)
- The first form creates an empty menu.
- Individual menu items are of type MenuItem. It defines these constructors:

- MenuItem( ) throws HeadlessException
- MenuItem(String itemName) throws HeadlessException
- MenuItem(String itemName, MenuShortcut keyAccel) throws HeadlessException
- Here, itemName is the name shown in the menu, and keyAccel is the menu shortcut for this item.
- You can disable or enable a menu item by using the setEnabled( ) method. Its form is shown here:
  - void setEnabled(boolean enabledFlag)
- If the argument enabledFlag is true, the menu item is enabled. If false, the menu item is disabled.
- You can determine an item’s status by calling isEnabled( ). This method is shown here:
- boolean isEnabled( )
- isEnabled( ) returns true if the menu item on which it is called is enabled. Otherwise, it returns false.

- You can change the name of a menu item by calling setLabel( ). You can retrieve the current name by using getLabel( ). These methods are as follows:
  - void setLabel(String newName)
  - String getLabel( )
- Here, newName becomes the new name of the invoking menu item. getLabel( ) returns the current name.
- You can create a checkable menu item by using a subclass of MenuItem called CheckboxMenuItem. It has these constructors:
  - CheckboxMenuItem( ) throws HeadlessException
  - CheckboxMenuItem(String itemName) throws HeadlessException
  - CheckboxMenuItem(String itemName, boolean on) throws HeadlessException
- Here, itemName is the name shown in the menu. Checkable items operate as toggles. Each time one is selected, its state changes. In the first two forms, the

- You can obtain the status of a checkable item by calling getState( ). You can set it to a known state by using setState( ). These methods are shown here:
  - boolean getState( )
  - void setState(boolean checked)
- If the item is checked, getState( ) returns true. Otherwise, it returns false. To check an item, pass true to setState( ). To clear an item, pass false.
- Once you have created a menu item, you must add the item to a Menu object by using add( ), which has the following general form:
  - MenuItem add(MenuItem item)
- Here, item is the item being added. Items are added to a menu in the order in which the calls to add( ) take place. The item is returned.
- Once you have added all items to a Menu object, you can add that object to the menu bar by using this version of add( ) defined by MenuBar:

- Menu add(Menu menu)
- Here, menu is the menu being added. The menu is returned.
- Menus generate events only when an item of type MenuItem or CheckboxMenuItem is selected.
- They do not generate events when a menu bar is accessed to display a drop-down menu, for example. Each time a menu item is selected, an ActionEvent object is generated.
- By default, the action command string is the name of the menu item. However, you can specify a different action command string by calling setActionCommand( ) on the menu item.
- Each time a check box menu item is checked or unchecked, an ItemEvent object is generated. Thus, you must implement the ActionListener and/or ItemListener interfaces in order to handle these menu events.
- The getItem( ) method of ItemEvent returns a reference to the item that generated this event. The general form

```
import java.awt.*;
import java.awt.event.*;

public class MenuDemo {
    public MenuDemo() {
        Frame f = new Frame("DashBoard");
        f.setSize(400, 400);
        MenuBar menubar = new MenuBar();
        Menu fileMenu = new Menu("File",true);
        Menu editMenu = new Menu("Edit");
        Menu helpMenu = new Menu("Help");
        menubar.add(fileMenu);
        menubar.add(editMenu);
        menubar.add(helpMenu);
        f.setMenuBar(menubar);
        MenuItem newfile = new MenuItem("New");
        MenuItem openfile = new MenuItem("Open");
        MenuItem newsave = new MenuItem("Save");
        MenuItem newsaveas = new MenuItem("SaveAs");
        MenuItem newexit = new MenuItem("Exit");
        CheckboxMenuItem c1 = new CheckboxMenuItem("Check 1",
true);
        CheckboxMenuItem c2 = new CheckboxMenuItem("Check 2");
         fileMenu.add(newfile);
         fileMenu.add(openfile);
         fileMenu.add(newsave);
         fileMenu.add(newsaveas);
         fileMenu.add(newexit);
         fileMenu.add(c1);
         fileMenu.add(c2);
        newsaveas.setEnabled(false);

        f.setLocationRelativeTo(null);
        f.setVisible(true);

    }

    public static void main(String[] args) {
        new MenuDemo();
    }
}
```

Dialog Boxes

- Often, you will want to use a dialog box to hold a set of related controls.
- Dialog boxes are primarily used to obtain user input and are often child windows of a top-level window.
- Dialog boxes don’t have menu bars, but in other respects, they function like frame windows. (You can add controls to them, for example, in the same way that you add controls to a frame window.)
- Dialog boxes may be modal or modeless. When a modal dialog box is active, all input is directed to it until it is closed. This means that you cannot access other parts of your program until you have closed the dialog box.
- When a modeless dialog box is active, input focus can be directed to another window in your program. Thus, other parts of your program remain active and accessible.
- In the AWT, dialog boxes are of type Dialog. Two commonly used constructors are shown here:
- Dialog(Frame parentWindow, boolean mode)
- Dialog(Frame parentWindow, String title, boolean mode)

- Here, parentWindow is the owner of the dialog box. If mode is true, the dialog box is modal. Otherwise, it is modeless. The title of the dialog box can be passed in title. Generally, you will subclass Dialog, adding the functionality required by your application.

Dialog Example

```
import java.awt.*;
import java.awt.event.*;

public class DialogDemo {
    public DialogDemo() {
        Frame f = new Frame();
        Dialog d = new Dialog(f, "Dialog Demo", true);
        Panel p = new Panel(new GridLayout(3, 2));
        Panel p1 = new Panel(new FlowLayout());
        Label l1 = new Label("Name");
        Label l2 = new Label("Address");
        TextField t1 = new TextField(10);
        TextField t2 = new TextField(10);
```

```
Button b1=new Button("Submit");
Button b2=new Button("Cancel");
Dialog d1 = new Dialog(f, "new Dialog");
Panel p3 = new Panel(new FlowLayout());
Label l3=new Label("Number");
TextField t3 = new TextField(10);
Button b3=new Button("Up");
Button b4=new Button("Down");
p.add(l1);
p.add(t1);
p.add(l2);
p.add(t2);
```

```
p.add(b1);
p.add(b2);
p3.add(l3);
p3.add(t3);
p3.add(b3);
p3.add(b4);
Button click = new Button("Click here to see result");
Button next = new Button("Next Window");
p1.add(click);
p1.add(next);
f.add(p1);
```

```
 click.addActionListener(new ActionListener() {
public void actionPerformed(ActionEvent e) {
                d.add(p);
                d.setSize(400, 300);
                d.setLocationRelativeTo(null);
                d.setVisible(true);
            }
        });
        next.addActionListener(new ActionListener() {
public void actionPerformed(ActionEvent e) {
                d1.add(p3);
                d1.setSize(400, 400);
                d1.setLocationRelativeTo(null);
                d1.setVisible(true);
            }
        });
 d.addWindowListener(new WindowAdapter() {
            public void windowClosing(WindowEvent
windowEvent) {
              d.dispose();
            }
        });
        d1.addWindowListener(new WindowAdapter() {
            public void windowClosing(WindowEvent
windowEvent) {
             d1.dispose();
            }
        });
        f.setSize(400, 400);
        f.setLocationRelativeTo(null);
        f.setVisible(true);
     }
     public static void main(String[] args) {
        new DialogDemo();
```

```
import java.awt.*;
import java.awt.event.*;

public class DialogBoxDemo{
    private Frame mainFrame;
   private Label headerLabel;
   private Label statusLabel;
   private Panel controlPanel;

   public DialogBoxDemo(){
      prepareGUI();
   }
}
public static void main(String[] args){
      DialogBoxDemo  dialogControlDemo = new
DialogBoxDemo();
      dialogControlDemo.showDialogDemo();
   }
```

```java
private void prepareGUI(){
   mainFrame = new Frame("Java AWT Examples");
   mainFrame.setSize(400,400);
   mainFrame.setLayout(new GridLayout(3, 1));

   headerLabel = new Label();
   headerLabel.setAlignment(Label.CENTER);
   statusLabel = new Label();
   statusLabel.setAlignment(Label.CENTER);
   statusLabel.setSize(350,100);

   controlPanel = new Panel();
   controlPanel.setLayout(new FlowLayout());

   mainFrame.add(headerLabel);
   mainFrame.add(controlPanel);
   mainFrame.add(statusLabel);
   mainFrame.setVisible(true);
}
```

```
 private void showDialogDemo(){
      headerLabel.setText("Control in action: Dialog");
      Button showAboutDialogButton = new Button("Show
About Dialog");
      showAboutDialogButton.addActionListener(new
ActionListener() {
            @Override
         public void actionPerformed(ActionEvent e) {
            AboutDialog aboutDialog = new
AboutDialog(mainFrame);
            aboutDialog.setVisible(true);
         }
      });

      controlPanel.add(showAboutDialogButton);
      mainFrame.setVisible(true);
   }
```

```
class AboutDialog extends Dialog {
      public AboutDialog(Frame parent){
         super(parent, true);
         setBackground(Color.gray);
         setLayout(new BorderLayout());
         Panel panel = new Panel();
         panel.add(new Button("Close"));
         add("South", panel);
         setSize(200,200);

         addWindowListener(new WindowAdapter() {
            public void windowClosing(WindowEvent
windowEvent){
               dispose();
            }
         });
      }
```

FileDialog

- Java provides a built-in dialog box that lets the user specify a file.
- To create a file dialog box, instantiate an object of type FileDialog. This causes a file dialog box to be displayed.
- Usually, this is the standard file dialog box provided by the operating system. Here are three FileDialog constructors:
- FileDialog(Frame parent)
- FileDialog(Frame parent, String boxName)
- FileDialog(Frame parent, String boxName, int how)
- Here, parent is the owner of the dialog box.
- The boxName parameter specifies the name displayed in the box’s title bar. If boxName is omitted, the title of the dialog box is empty.
- If how is FileDialog.LOAD, then the box is selecting a

- FileDialog provides methods that allow you to determine the name of the file and its path as selected by the user. Here are two examples:
- String getDirectory( )
- String getFile( )
- These methods return the directory and the filename, respectively.

```
import java.awt.FileDialog;
import java.awt.Frame;
import java.awt.event.WindowAdapter;
import java.awt.event.WindowEvent;
public class FileDialogDemo {

    public FileDialogDemo() {
        Frame f = new Frame("File Dialog Demo");
        f.setVisible(true);
        f.setSize(100, 100);
```

```
 f.addWindowListener(new WindowAdapter() {
            @Override
            public void windowClosing(WindowEvent 
windowEvent) {
                System.exit(0);
            }
        });
FileDialog fd = new FileDialog(f, "File Dialog");
        fd.setVisible(true);


    }


    public static void main(String[] args) {
        new FileDialogDemo();
    }
}
```