Richard G Baldwin (512) 223-4758, baldwin@austin.cc.tx.us,
http://www2.austin.cc.tx.us/baldwin/

Event Handling in JDK 1.1, Scrollbar (Adjustment) Events

Java Programming, Lecture Notes # 90, Revised 12/18/98.

Preface

Students in Prof. Baldwin's **Intermediate Java Programming** classes at ACC are responsible for knowing and understanding all of the material in this lesson.

Introduction

JDK 1.1 was formally released on February 18, 1997. This lesson was originally written on February 21, 1997 using the software in JDK 1.1. It was upgraded to JDK 1.2 on 12/18/98.

Possible Bugs in Scrollbar Component

The version of this lesson that was published in February of 1997 listed a number of serious bugs in the JDK 1.1 Scrollbar component. I have now recompiled and retested the program described herein using JDK 1.1.3. Most of the bugs appear to have been corrected. However, a couple of bugs still exist that are described in the program.

The program contains workaround code for one of the bugs identified by the program (incorrect negative values). Note that this bug still exists in JDK 1.1.6 but has been fixed in the first release of JDK 1.2.

The other bug (excessively large positive values) was simply ignored, and can be seen if the program is compiled using JDK 1.1.3 under Windows 95. This bug still exists in JDK 1.1.6 but has been fixed in the first release of JDK 1.2.

Overview

Event handling on a **Scrollbar** object differs from the event handling in previous lessons in several respects. For example, lets contrast the **Scrollbar** with the **Mouse**.

To create a listener object for the mouse, you either implement the **MouseListener** interface, or extend the **MouseAdapter** class.

However, to create a listener object for a **Scrollbar**, you do not implement a *ScrollbarListener* interface or extend a *ScrollbarAdapter* class, because they do not exist.

Rather, you implement an **AdjustmentListener** interface. The method declared in the **AdjustmentListener** interface receives an **AdjustmentEvent** object as a parameter. This is one of the *semantic* events (as opposed to *low-level* events) similar to **ActionEvent**, **ItemEvent**, and **TextEvent.**

Note that there is no *AdjustmentAdapter* class. It isn't needed because the **AdjustmentListener** interface declares only one method:

```
public abstract void adjustmentValueChanged(AdjustmentEvent e)
```

The **adjustmentValueChanged()** method is invoked when the value of the adjustable object (the Scrollbar object in this case) has changed.

Recall also that there are five different types of mouse events declared in the **MouseListener** interface:

- mouseClicked(),
- mouseEntered(),
- mouseExited(),
- mousePressed(), and
- mouseReleased().

Although we haven't discussed them yet, there are two additional types of mouse events which are declared in the **MouseMotionListener** interface:

- mouseDragged()
- mouseMoved()

Each of these different types of mouse events is represented by a method declaration in one or the other of the two interfaces defined for creating listener classes for mouse activity. You *override the method(s)* for those types of events which interest you.

There are five different types of adjustment events which correspond to the five ways of moving the bubble in a **Scrollbar** object. However, as mentioned above, there is only one method declared in the **AdjustmentListener** interface. It is named **adjustmentValueChanged()**. It receives an object of type **AdjustmentEvent** as a parameter when invoked.

The type of event is encoded into the object that is passed in as a parameter along with other information such as the **value** of the **Scrollbar** and the **name** of the **Scrollbar** object. The **value** of the **Scrollbar** is determined by the position of the bubble on the **Scrollbar**.

The **AdjustmentEvent** class defines several methods that can be used to extract information from the object in case you need access to that information. This is discussed in more detail later.

The types of adjustment events (corresponding to physical methods of moving the bubble on the **Scrollbar**) are defined as static variables in the AdjustmentEvent class. (This is similar to the approach used in the earlier event model in JDK 1.0.2.) The five different types of adjustment events and their relationship to moving the bubble are:

UNIT_INCREMENT - click button on one end of the Scrollbar

UNIT_DECREMENT - click button on the other end of the Scrollbar

BLOCK_INCREMENT - click in space between bubble and one button

BLOCK_DECREMENT - click in space between bubble and other button

TRACK - drag the bubble

As indicated, the two UNIT types are generated by clicking on the buttons at either end of the Scrollbar.

The two BLOCK types are generated by clicking interior to the Scrollbar on either side of the bubble.

The TRACK event is generated by dragging the bubble.

Methods are available for setting various parameters of the **Scrollbar** including the range, the width of the bubble (alternately referred to in the documentation as *page size* or *visible*), the size of the unit and block increments or decrements, etc.

One might surmise that this rather general purpose adjustment approach was defined to support a family of components that operate on an adjustment basis. However, a quick search of the index in the JDK 1.1.3 documentation package did not identify any other components that use the adjustment interface. Perhaps other adjustable objects are planned for future versions of the AWT.

The Sample Program

This section presents a discussion of the program followed by the program listing.

Discussion

This program places a **Scrollbar** object and a **TextField** object in a **Frame**. Whenever the *bubble* in the **Scrollbar** is moved using any of the five available physical methods for moving the bubble, the **value** of the **Scrollbar** (which normally should represent the position of the center of the *bubble*) is extracted from the event object and displayed in the TextField object. (Certain arithmetic corrections are required to cause the value to represent the center of the bubble because the actual value returned represents the left-hand edge of the bubble for a horizontal **Scrollbar**.)

Also, whenever the bubble is moved, several other pieces of information are extracted from the object and displayed on the screen. This includes the identification of the adjustable object and the type of adjustable event.

Note also that although five different initialization parameters are included as parameters to the **Scrollbar** constructor, other important initialization parameters such as **BlockIncrement** and **UnitIncrement** are not included in the parameter list. They must be set following instantiation using methods such as **setBlockIncrement()**.

Experimentation indicates that the default value for **BlockIncrement** is ten units and the default value for **UnitIncrement** is one unit.

Reiterating what was mentioned earlier, the Scrollbar listener object is not added as a *ScrollbarListener* but rather is defined by implementing the **AdjustmentListener** interface and then added as an **AdjustmentListener**.

The **AdjustmentListener** interface declares only one method that may be overridden: **adjustmentValueChanged()**.

This method receives an object of type **AdjustmentEvent** which provides four methods that can be used to obtain information about the event.

- The **getAdjustable()** method of the **AdjustmentEvent** class returns information containing the identification of the object that generated the event.
- The **getAdjustmentType()** method returns information containing the identification of the type of event (click the buttons on the Scrollbar, drag the bubble, etc.).
- The **paramString()** method contains a variety of information which overlaps the two above methods to some extent.
- The **getValue()** method returns the value produced by the adjustment.

Three of these four methods are used in the following program. The **paramString()** method is not used.

There are many important comments in the following program which add to this discussion, particularly with respect to bugs in the **Scrollbar** component, and arithmetic adjustments required to compensate for the width of the bubble.

Interesting Code Fragments

I begin with a fragment showing the controlling class that contains the requiste **main()** method. The **main()** method instantiates an object of type **GUI** where all of the work is done.

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

class Event18{
  public static void main(String[] args){
    GUI gui = new GUI();
  }//end main
}//end class Event18
```

The next fragment shows the beginning of the **GUI** class along with some instance variables used later in the program.

```
class GUI{
  Scrollbar myScrollbar;
  TextField displayWindow;
  int bubbleWidth;//needs to be accessible by event handler
```

The next fragment begins the constructor for the **GUI** class. The constructor begins by instantiating a **Frame** object to serve as a top-level window for the application.

This is followed by instantiation of a horizontal Scrollbar object with range from 0 to 100, an initial position at 50, a bubble width (page size) of 20, a unitIncrement of 2, and a blockIncrement of 15.

The Scrollbar component does not center the bubble on the value. Rather, the left edge of the bubble is lined up with the value. The result is that values in the upper end of the range cannot be reached unless appropriate adjustments are made using half the bubble width. This distributes the unreachable values at each end of the Scrollbar. You can then set the min and max values to extend beyond the desired values by one-half the bubble width.

```
  GUI(){
    Frame myFrame = new Frame("Copyright 1997, R.G.Baldwin");

    bubbleWidth = 20;
    int initialPosition = 50 - bubbleWidth/2;
    int min = 0 - bubbleWidth/2;
    int max = 100 + bubbleWidth/2;
    myScrollbar = new Scrollbar(Scrollbar.HORIZONTAL,
                    initialPosition, bubbleWidth, min, max);
```

The next fragment sets the unit increment to 2 and the block increment to 15. The unit increment is the amount of change resulting from clicking a button on the end of the Scrollbar. The block increment is the amount of change resulting from clicking between the bubble and the button on the end.

```
    myScrollbar.setBlockIncrement(15);
    myScrollbar.setUnitIncrement(2);
```

Code of the type shown in the next fragment should be very familiar to you by now. It really isn't very interesting at this point, and is included in this section solely for completeness.

This fragment creates the display window using a **TextField** object. It also adds the two components to the **Frame**, sets the size of the **Frame**, and makes the whole thing visible.

The fragment also includes typical code to instantiate a listener object for the **Scrollbar** and register it to receive notification of adjustment events.

This is followed by typical code to register a window listener to terminate the program when the user clicks the close box on the **Frame**.

That ends the constructor and also ends the definition of the **GUI** class.

```
    displayWindow = new TextField("Initial Text");
    displayWindow.setEditable(false);//make non-editable

    //add components to the GUI
    myFrame.add("South", myScrollbar);
    myFrame.add("North", displayWindow);
    myFrame.setSize(300,75);
    myFrame.setVisible(true);

    MyScrollbarListener myScrollbarListener =
      new MyScrollbarListener(this);
    //Note that the Scrollbar listener is not added as a
    // ScrollbarListener but rather is added as an
    // AdjustmentListener.
    myScrollbar.addAdjustmentListener(myScrollbarListener);

    //Close Frame to terminate.
    myFrame.addWindowListener(new MyWindowListener());
  }//end constructor
}//end class GUI
```

The next fragment begins to get interesting again. It shows the beginning of the **AdjustmentListener** class from which a listener object is instantiated and registered on the **Scrollbar** to listen for adjustment events. This fragment shows the constructor for the class that saves a reference to the **Scrollbar**.

```
class MyScrollbarListener implements AdjustmentListener{
```

```
  GUI thisObject; //save ref to GUI object here

  MyScrollbarListener(GUI objectIn){//constructor
    thisObject = objectIn;
  }//end constructor
```

The following fragment begins the code that serves as the heart of the program. This is the **adjustmentValueChanged()** method, which is the only method declared by the **AdjustmentListener** interface. This method receives a parameter which is an object of type **AdjustmentEvent**.

This fragment uses the incoming object to obtain and display the incoming object, the source of the event, and the type of the event.

```
  public void adjustmentValueChanged(AdjustmentEvent e){
    int value;
    //Display the entire AdjustmentEvent object
    System.out.println(e);
    System.out.println("Adjustable = " + e.getAdjustable());
    System.out.println("AdjustmentType = "
                                    + e.getAdjustmentType());
```

The following output was produced by the above fragment when the button on the right end of the scrollbar was clicked (line breaks were manually inserted by the author).

```
java.awt.event.AdjustmentEvent[ADJUSTMENT_VALUE_CHANGED,
     adjType=UNIT_INCREMENT,value=42] on scrollbar0
Adjustable = java.awt.Scrollbar[scrollbar0,4,56,292x15,
     val=42,vis=20,min=-10,max=110,horz]
AdjustmentType = 1
```

The next fragment uses the incoming object to get the *value* of the scrollbar and display it in the **TextField** object. The value should represent the position of the bubble (it really represents the position of the left edge of the bubble, not the center).

This fragment contains code to serve as a workaround for a bug in JDK 1.1.6 where small negative values are reported as very large positive values when the Scrollbar is adjusted by dragging the bubble. Note that this bug still exists in JDK 1.1.6 but has been fixed in JDK 1.2.

The fragment also contains code to adjust the reported value to the center of the bubble instead of the left edge of the bubble by increasing the value by one-half the width of the bubble. The adjusted value is displayed in the **TextField** object.

It also contains code to resolve another bug which causes the bubble to jump back to its previous value when you attempt to drag it in JDK 1.1.6. This also has been fixed in JDK 1.2.

```
    value = e.getValue();//get and save the value
```

```
    //The following code is here to work around a
    // bug in the Scrollbar object for JDK 1.1.6
    if(value > 65000)
      value = value - 65536;

    //Adjust value to center of the bubble
    thisObject.displayWindow.setText("Value = "
                     + (value + thisObject.bubbleWidth/2));

    //The following is required to make the bubble stay put
    // in JDK 1.1.6 (not required for JDK 1.2)
    thisObject.myScrollbar.setValue(value);
  }//end adjustmentValueChanged()

}//end class MyScrollbarListener
```

That concludes the interesting code fragments for this program. A complete listing is provided in the next section.

Program Listing

A listing of the program with additional comments follows:

```
/*File Event18.java Copyright 1997, R.G.Baldwin
Revised 12/18/97
This program was designed to be compiled and executed under
JDK 1.1.3 or later version.

In an earlier version of this program compiled under
JDK 1.1, in February 1997, I reported numerous bugs in the
JDK Scrollbar handling.  I have now recompiled the program
and can report that most of the bugs seem to have been fixed,
although a couple of bugs still seem to exist.

One remaining bug is that if you adjust the scrollbar by
dragging the bubble into an area which would report negative
values, large positive values are reported instead of small
negative values (ie: something like 65536 instead of -1).
This does not happen if the scrollbar is adjusted by
clicking on the end buttons or in the area between the
bubble and the buttons.  A numeric workaround was
implemented in this program to work around that bug. Note
that the bug still exists in JDK 1.1.6 but was fixed in the
first release of JDK 1.2.

Another possible bug is that the scrollbar seems to be able
to report a value that is one unitIncrement too large for
the specified maximum value and bubble width.  Note
that the bug still exists in JDK 1.1.6 but was fixed in the
first release of JDK 1.2.

This program places a Scrollbar object and a TextField
object in a Frame.  Whenever the bubble in the Scrollbar is
```

```
moved using any of the five available methods for moving
the bubble the value of the Scrollbar (which should
represent the position of the bubble) is displayed in the
TextField object.

Also, whenever the bubble is moved, several other pieces of
information are displayed on the screen which identify
various parameters of the adjustment.

Closing the frame terminates the program.
*/
//=========================================================
import java.awt.*;
import java.awt.event.*;

class Event18{
  public static void main(String[] args){
    GUI gui = new GUI();
  }//end main
}//end class Event18
//=========================================================
class GUI{
  Scrollbar myScrollbar;
  TextField displayWindow;
  int bubbleWidth;//needs to be accessible by event handler

  GUI(){
    Frame myFrame = new Frame("Copyright 1997, R.G.Baldwin");

    //Instantiate a horizontal Scrollbar object with range
    // from 0 to 100, initial position at 50, bubble width
    // (page size) of 20, unitIncrement of 2 and a
    // blockIncrement of 15.

    // The Scrollbar component does not center the bubble on
    // the value.  Rather, the left edge of the bubble is
    // lined up with the value.  The result is that values
    // in the upper end of the range cannot be reached
    // unless appropriate adjustments are made using half
    // the bubble width.  This distributes the unreachable
    // values at each end of the Scrollbar. You can then
    // set the min and max values to extend beyond the
    // desired values by one-half the bubble width.
    bubbleWidth = 20;
    int initialPosition = 50 - bubbleWidth/2;
    int min = 0 - bubbleWidth/2;
    int max = 100 + bubbleWidth/2;
    myScrollbar = new Scrollbar(Scrollbar.HORIZONTAL,
                    initialPosition, bubbleWidth, min, max);

    //It would have been nice to have included unitIncrement
    // and blockIncrement as parameters to the constructor
    // along with the other constructor parameteters. See
    // next two statements which set the unitIncrement and
    // the blockIncrement.  The unitIncrement is the amount
    // of change resulting from clicking a button on the
```

```
    // end of the Scrollbar.  The blockIncrement is the
    // amount of change resulting from clicking between the
    // bubble and the button on the end.
    myScrollbar.setBlockIncrement(15);
    myScrollbar.setUnitIncrement(2);

    displayWindow = new TextField("Initial Text");
    displayWindow.setEditable(false);//make non-editable

    //add components to the GUI
    myFrame.add("South", myScrollbar);
    myFrame.add("North", displayWindow);
    myFrame.setSize(300,75);
    myFrame.setVisible(true);

    //Instantiate a listener object for the Scrollbar and
    // register it to receive notification of adjustment
    // events.
    MyScrollbarListener myScrollbarListener =
      new MyScrollbarListener(this);
    //Note that the Scrollbar listener is not added as a
    // ScrollbarListener but rather is added as an
    // AdjustmentListener.
    myScrollbar.addAdjustmentListener(myScrollbarListener);

    //Close Frame to terminate.
    myFrame.addWindowListener(new MyWindowListener());
  }//end constructor
}//end class GUI
//=========================================================

//Note that unlike some other components which have their
// own listener interface, this class does not implement
// ScrollbarListener because there is no such interface.
// Rather, the AdjustmentListener interface is used as a
// listener interface for Scrollbars.
class MyScrollbarListener implements AdjustmentListener{
  GUI thisObject; //save ref to GUI object here

  MyScrollbarListener(GUI objectIn){//constructor
    thisObject = objectIn;
  }//end constructor

  public void adjustmentValueChanged(AdjustmentEvent e){
    int value;
    //Display the entire AdjustmentEvent object
    System.out.println(e);
    System.out.println("Adjustable = " + e.getAdjustable());
    System.out.println("AdjustmentType = "
                                    + e.getAdjustmentType());

    //Display the value of the Scrollbar object in the
    // TextField.  The value should represent the position
    // of the bubble.
    value = e.getValue();//get and save the value
```

```
    //The following code is here to work around a possible
    // bug in the Scrollbar object where small negative
    // values are reported as very large positive values
    // when the Scrollbar is adjusted by sliding the bubble.
    // Note that this bug still exists in JDK 1.1.6 but has
    // been fixed in JDK 1.2.
    System.out.println("value = " + value);
    if(value > 65000) value = value - 65536;
    System.out.println("value = " + value);
    thisObject.displayWindow.setText("Value = "
                     + (value + thisObject.bubbleWidth/2));

    //The following seems like a kludge but is required to
    // make the scrollbar bubble stay put. Otherwise, it
    // jumps back to the previous value when you try to
    // move it.
    //Note that this statement is required for JDK 1.1.6
    // but is not required for the first release of JDK 1.2
    thisObject.myScrollbar.setValue(value);
  }//end adjustmentValueChanged()

}//end class MyScrollbarListener
//=========================================================
//Listener to terminate the program when the Frame is
// closed.
class MyWindowListener extends WindowAdapter{
  public void windowClosing(WindowEvent e){
    System.exit(0);
  }//end windowClosing()
}//end class MyWindowListener
//=========================================================
```

.

Review

Q - All of the **Scrollbar** bugs were eliminated in JDK 1.1.3: True or False? If false, explain why.

A - False. JDK 1.1.3 appears to still have at least two Scrollbar bugs. One bug causes the *track* event (which involves dragging the bubble in the Scrollbar object) to return large positive values such as 65535 when it should be returning small negative values such as -1. The other apparent bug allows the different versions of the event to return a value larger than would be calculated using the minimum and maximum values along with the width of the bubble.

Q - The **Scrollbar** bugs listed above cause the Scrollbar component to be completely unusable: True or False? If false, explain why.

A - False. While these two bugs represent a significant inconvenience, it is probably possible to work around both of them by making numeric corrections.

Q - To create a listener object for a **Scrollbar**, you must implement the *ScrollbarListener* interface or extend a *ScrollbarAdapter* class: True or False? If false, explain why.

A - False. To create a listener object for a **Scrollbar**, you do not implement a *ScrollbarListener* interface or extend a *ScrollbarAdapter* class, because they do not exist. Instead, you implement an **AdjustmentListener** interface.

Q - To create a listener object for a **Scrollbar**, you can either implement the *AdjustmentListener* interface or extend the *AdjustmentAdapter* class: True or False? If False, explain why.

A - False. There is no *AdjustmentAdapter* class. It isn't needed because the **AdjustmentListener** interface declares only one method whose signature follows:

```
public abstract void adjustmentValueChanged(AdjustmentEvent e)
```

Q - The **adjustmentValueChanged()** method is invoked when the value of the the **Scrollbar** object is changed: True or False? If false, explain why.

A - True.

Q - There are five different types of adjustment events which correspond to the different methods of moving the bubble in a **Scrollbar** object: True or False. If false, explain why.

A - True.

Q - As with most of the other classes in the AWT, the *AdjustmentListener* interface declares a separate method for each type of adjustment event. Each of these methods must be implemented in any class that implements the *AdjustmentListener* interface: True or False? If false, explain why.

A - False. Only one method is declared in the *AdjustmentListener* interface. The name of the method is **adjustmentValueChanged()**.

Q - The **adjustmentValueChanged()** method receives an object of type **AdjustmentEvent** as a parameter, and the type of event is encoded into the object: True or False? If false, explain why.

A - True.

Q - In addition to the type of the event, additional information is also encoded into the object that is passed to the **adjustmentValueChanged()** method. Give examples of two other kinds of information encoded into the object..

A - Information such as the **value** of the **Scrollbar** and the **name** of the **Scrollbar** component is also encoded into the object.

Q - The types of adjustment events are defined as static variables in the AdjustmentEvent class. What are the names of these variables, and how to they correspond to user actions on the **Scrollbar** object?

A - The names of the **static** variables and their relationship to the **Scrollbar** object are:

- UNIT_INCREMENT - click button on one end of the Scrollbar
- UNIT_DECREMENT - click button on the other end of the Scrollbar
- BLOCK_INCREMENT - click in space between bubble and one button
- BLOCK_DECREMENT - click in space between bubble and other button
- TRACK - drag the bubble

Q - The width of the bubble is also referred to by at least two other terms in the **Scrollbar** documentation. What are they?

A - The other terms used to describe the bubble width are *page size* and *visible*. These terms seem to be related to the use of a **Scrollbar** on the side or bottom of a window of text where the size of the bubble is indicative of the amount of text that is visible in the window in relation to the amount of text in the entire document..

Q - In JDK 1.1.3, the **value** of the **Scrollbar** object is represented by the center of the bubble: True or False? If false, explain why.

A - False. The **value** of the **Scrollbar** object is represented by an edge of the bubble. For example, the **value** is represented by the position of the left edge of the bubble for a horizontal **Scrollbar**. If you want the position of the center of the bubble to represent a *value*, you must perform a numeric correction (involving half the width of the bubble) on the **value** of the **Scrollbar** actually extracted from the event object.

Q - All necessary initialization parameters for a **Scrollbar** object are provided as parameters to one of the **Scrollbar** constructors: True or False: If false, explain why.

A - False. Although five different initialization parameters are included as parameters to the **Scrollbar** constructor, other important initialization parameters such as **BlockIncrement** and **UnitIncrement** are not included in the parameter list. They must be set following instantiation using methods such as **setBlockIncrement()**.

Q - The **AdjustmentEvent** class provides four methods that can be used to obtain information about the event. What are they and what kinds of information do they return?

A - The four methods and the information that they return are listed below:

- The **getAdjustable()** method returns the identification of the object that generated the event.
- The **getAdjustmentType()** method returns the identification of the type of event.
- The **paramString()** method returns a variety of information which overlaps the two above methods to some extent.
- The **getValue()** method returns the **value** produced by the adjustment.

Q - Write an application that meets the specifications given in the comments in the following program.

A - See the following program.

```
/*File SampProg125.java  from lesson 90
Copyright 1997, R. G. Baldwin
Without viewing the following solution, write a Java
application that meets the specifications given below.
```

```
To accommodate this program, you will need to be using
JDK 1.1.3 or a later version.

Write an application that places a vertical Scrollbar and a
Label on a Frame.  The Label is used to display the
position of the center of the bubble in the Scrollbar.

The Scrollbar values should range from approximately -50
to +50 (exact end points are difficult to achieve).

The width of the bubble should be 10 units.

When you click on the buttons at the end of the Scrollbar,
the bubble should move by 3 units.

When you click in the area between the bubble and the
buttons, the bubble should move by 6 units.

When the program starts, the bubble should be centered
and the Label should indicate a bubble position of 0.

Closing the frame terminates the program.

End of specifications.
*/
//=========================================================
import java.awt.*;
import java.awt.event.*;

class SampProg125{
  public static void main(String[] args){
    GUI gui = new GUI();
  }//end main
}//end class SampProg125
//=========================================================
class GUI{
  Scrollbar myScrollbar;
  Label displayWindow;
  int bubbleWidth;//needs to be accessible by event handler

  GUI(){
    Frame myFrame = new Frame("Copyright 1997, R.G.Baldwin");

    bubbleWidth = 10;
    int initialPosition = 0 - bubbleWidth/2;
    int min = -50 - bubbleWidth/2;
    int max = 50 + bubbleWidth/2;
    myScrollbar = new Scrollbar(Scrollbar.VERTICAL,
                    initialPosition, bubbleWidth, min, max);

    myScrollbar.setBlockIncrement(6);
    myScrollbar.setUnitIncrement(3);

    displayWindow = new Label("value = 0");

    //add components to the GUI
```

```
    myFrame.add("East", myScrollbar);
    myFrame.add("North", displayWindow);
    myFrame.setSize(300,300);
    myFrame.setVisible(true);

    //Instantiate a listener object for the Scrollbar and
    // register it to receive notification of adjustment
    // events.
    MyScrollbarListener myScrollbarListener =
                            new MyScrollbarListener(this);
    //Note that the Scrollbar listener is not added as a
    // ScrollbarListener but rather is added as an
    // AdjustmentListener.
    myScrollbar.addAdjustmentListener(myScrollbarListener);

    //Close Frame to terminate.
    myFrame.addWindowListener(new MyWindowListener());
  }//end constructor
}//end class GUI
//===================================================================

//Note that unlike some other components which have their
// own listener interface, this class does not implement
// ScrollbarListener because there is no such interface.
// Rather, the AdjustmentListener interface is used as a
// listener interface for Scrollbars.
class MyScrollbarListener implements AdjustmentListener{
  GUI thisObject; //save ref to GUI object here

  MyScrollbarListener(GUI objectIn){//constructor
    thisObject = objectIn;
  }//end constructor

  public void adjustmentValueChanged(AdjustmentEvent e){
    int value;

    //Display the value of the Scrollbar object in the
    // Label.  The value should represent the position
    // of the bubble.
    value = e.getValue();//get and save the value

    //The following code is here to work around a possible
    // bug in the Scrollbar object where small negative
    // values are reported as very large positive values
    // when the Scrollbar is adjusted by sliding the bubble.
    if(value > 65000) value = value - 65536;
    thisObject.displayWindow.setText("Value = "
                      + (value + thisObject.bubbleWidth/2));

    //The following seems like a kludge but is required to
    // make the scrollbar bubble stay put. Otherwise, it
    // jumps back to the previous value when you try to
    // move it.
    thisObject.myScrollbar.setValue(value);
  }//end adjustmentValueChanged()
```

```
}//end class MyScrollbarListener
//===========================================================
//Listener to terminate the program when the Frame is
// closed.
class MyWindowListener extends WindowAdapter{
  public void windowClosing(WindowEvent e){
    System.exit(0);
  }//end windowClosing()
}//end class MyWindowListener
```

-end-