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

Event Handling in JDK 1.1, Sharing a Listener Object Among, Visual Components

Java Programming, Lecture Notes # 82, Revised 02/14/98.

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

Previous lessons discussed the sharing of a single event Source object among two or more Listener objects for *low-level* events.

This lesson discusses the sharing of a single Listener object among two or more visual components for *low-level events* generated by any of the visual components.

The event-handling method of the listener object

- responds to each event,
- determines which visual component generated the event, and
- takes the appropriate action.

The primary issue in this type of operation boils down to determining which component generated the event.

This is accomplished by assigning a unique name to each visual component when it is instantiated and using that name to distinguish among the visual components when an event occurs.

This approach is completely straightforward and intuitive. An improved approach to accomplishing the same objective is illustrated in some sample programs near the end of the lesson. While possibly less intuitive, the improved approach is more object-oriented and probably easier to maintain.

Sample Program

This program illustrates the sharing of a single listener object among two different visual components of the same type.

The program detects mouse events occurring on either of two different Frame **objects**. It distinguishes between the two objects on the basis of the component name, and displays the coordinates of a mouse click on whichever object experienced the mouse click.

This program was tested using JDK 1.1.3 under Win95.

Note that when the program starts, the two visual components overlay one another. It is necessary to physically move one of them in order to access the other.

Interesting Code Fragments

This program is really quite simple. The only new code involves cracking the event object to obtain the name of the visual component that generated the **mousePressed()** event.

The **main()** method instantiates an object of type GUI (named **gui)** which serves the purpose of providing the visual interface.

The **Frame** class is *extended* into a new class named **MyFrame** to make it possible to override the **paint()** method of the class. This is necessary to display the coordinates of mouse clicks on the interior of the frame using the graphics method **drawString()**.

The Event Source Objects

The constructor of the **GUI** class instantiates two objects of type **MyFrame** and makes them visible. This is accomplished using code such as the following:

```
    MyFrame myFrame1 = new MyFrame("Frame1");
    myFrame1.setVisible(true);
```

The reference variables for the two objects are named **myFrame1** and **myFrame2**.

When they are instantiated, unique names (**Frame1** and **Frame2)** are given to the objects using the following code in the constructor for the extended **Frame** objects (the **MyFrame** class *extends* the **Frame** class in order to override its **paint()** method).

```
  MyFrame(String name){//constructor
    setTitle("Copyright 1997, R.G.Baldwin");
    setSize(300,200);
    //name used to distinguish between the two objects
    setName(name);
  }//end constructor
```

The unique names are used later by the Listener object to determine which object generated a mouse event.

The constructor in the GUI class also uses the following code to instantiate a single Listener object which will process low-level mouse events on either of the two visual objects.

```
MouseProc mouseProcCmd = new MouseProc(myFrame1,myFrame2);
myFrame1.addMouseListener(mouseProcCmd);
myFrame2.addMouseListener(mouseProcCmd);
```

To review the syntax, the first statement simply instantiates the new Listener object named **mouseProcCmd** passing references to the two visual components as parameters.

The next two statements add that Listener object (*register* the Listener object) to a list of Listener objects which are automatically notified whenever mouse events occur on the visual objects referred to as **myFrame1**, and **myFrame2**, respectively.

You will recall that once the programmer causes the name of a Listener object to be added to the list, no further programming effort is required to cause the notification to occur.

The notifications are carried out by invoking specific overridden instance methods of the Listener object upon the occurrence of a specific types of mouse events.

The declarations for all of the methods which match up with all of the possible mouse event types are defined by the **MouseListener** interface which matches the **MouseEvent** class.

The class from which the **Listener** object is instantiated must define, either directly or indirectly, all the methods declared in the **MouseListener** interface.

In addition to the *registration* of the **MouseListener** object to receive mouse events, the program also instantiates and *registers* a Listener object which monitors for Window events and terminates the program whenever the user closes either of the two visual objects. In this case, the code in the Listener object makes no attempt to distinguish between the two visual objects. The instantiation and registration code is shown below.

```
    WProc1 winProcCmd1 = new WProc1();
    myFrame1.addWindowListener(winProcCmd1);
    myFrame2.addWindowListener(winProcCmd1);
```

The MouseListener Object

Most of the programming complexity is tied up in the **mouseListener** object, and it isn't very complicated.

The central issue for the code in the **Listener** object is how to determine which one of several visual components generated an event.

This particular **Listener** object only responds to **mousePressed** events, but the following information applies to all of the different types of mouse events, and probably to most or all of the low-level events as well.

The **MouseProc** (listener) class in this program

- extends the **MouseAdapter** class and
- overrides the **mousePressed()** method that is declared in the **MouseListener** interface.

You will recall that the **MouseAdapter** class overrides all of the methods of the **MouseListener** interface with empty methods, thus freeing our code from the requirement to override all of those methods.

When the **mousePressed()** method is invoked, an object of type **MouseEvent** (known locally as **e)** is passed in as a parameter.

The following statement was used in the **MouseListener** object to determine if the name of the object that generated the event is **Frame1**.

```
if( e.getComponent().getName().compareTo("Frame1") == 0 ){
```

If it is determined that the name of the component that generated the event is **Frame1**, code is executed to display the coordinates of the mouse pointer on the visual object named **Frame1**.

Otherwise, an **else** clause is used to display the coordinate information on the visual object named **Frame2**. (No provisions were made for the event to have been generated by any visual objects other than these two.)

The code to display the coordinate information is essentially the same as was discussed in a similar program in an earlier lesson, and so it won't be discussed again here.

Now let's take a look at the details of cracking the MouseEvent object to obtain the name of the visual component that generated the event.

The **getComponent()** method is a method of the **java.awt.event.ComponentEvent** class which, according to the JDK 1.1 documentation, *"Returns the component where this event originated."* It is returned as an object of type **Component** which is acted upon by the **getName()** method.

The **getName()** method was added to the **java.awt.Component** class by JDK 1.1. This method *"Gets the name of the component"* and returns it as a **String** object. The **String** object is acted upon by the **compareTo()** method.

The **compareTo()** method is a standard method of the **String** class, carried forward from JDK 1.0.2, that can be used to compare two **String** objects. It is used to determine if the name of the component matches the String "*Frame1"*.

Note that it is also possible to to perform tests directly on the **MouseEvent** object to match it to a component name. The procedure for doing this will be included in a subsequent lesson.

By the way, in case you haven't noticed, the **java.awt.event** package is different from the **java.awt** package. The **java.awt.event** package was added in JDK 1.1. It can be very confusing if you drop into the **java.awt** package of the documentation when you really need to be in **java.awt.event**.

The WindowListener Object

This program also contains a **WindowListener** object which terminates the program whenever the user closes either of the **Frame** objects. Except for the fact that this object is registered to receive **Window** events from either of the two **Frame** objects, it is no different from similar **Listener** used in an earlier example program, and therefore, won't be discussed further here.

Note that an improved version of this program is presented at the end of this lesson. The improved version does not require the establishment of **Source** object names, and does not require the passing of references to the constructor for the **WindowListener** in order to distinguish between between the two sources. You should become familiar with the methodology used in the improved version as well as the methodology used in the following version.

Program Listing

```
/*File Event10.java Copyright 1997, R.G.Baldwin
This program is designed to be compiled and run under
JDK 1.1

The program illustrates the sharing of a single listener
object among two different visual components of the same
type.

The program detects mouse events occurring on either of two
different Frame objects.  It distinguishes between the two
objects on the basis of the component name, and displays
the coordinates of a mouse click on whichever object
experienced the mouse click.

This program was tested using JDK 1.1.3 under Win95.

Note that when the program starts, the two visual
components overlay one another.  It is necessary to move
```

```
one of them in order to access the other.
**********************************************************/

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

public class Event10 {
  public static void main(String[] args){
    GUI gui = new GUI();//instantiate a GUI
  }//end main
}//end class Event10
//=======================================================//

//Subclass Frame in order to override the paint method.
class MyFrame extends Frame{
  int xCoor;
  int yCoor;

  MyFrame(String name){//constructor
    setTitle("Copyright 1997, R.G.Baldwin");
    setSize(300,200);
    //Name used to distinguish between the two objects
    setName(name);
  }//end constructor

  public void paint(Graphics g){
    //display coordinate information on the visual object
    g.drawString("" + xCoor + ", " + yCoor, xCoor, yCoor);
  }//end paint()
}//end class MyFrame
//=======================================================//

//The following class is used to instantiate a graphical
// user interface object.
class GUI {
  public GUI(){//constructor
    //Create two visual objects of type MyFrame and make
    // them visible. Name them Frame1 and Frame2.
    MyFrame myFrame1 = new MyFrame("Frame1");
    myFrame1.setVisible(true);

    MyFrame myFrame2 = new MyFrame("Frame2");
    myFrame2.setVisible(true);

    //Instantiate and register Listener object which will
    // terminate the program when the user closes either
    // window.
    WProc1 winProcCmd1 = new WProc1();
    myFrame1.addWindowListener(winProcCmd1);
    myFrame2.addWindowListener(winProcCmd1);

    //Instantiate and register Listener object which will
    // process mouse events on either MyFrame object.
    MouseProc mouseProcCmd =
                          new MouseProc(myFrame1,myFrame2);
    myFrame1.addMouseListener(mouseProcCmd);
```

```
    myFrame2.addMouseListener(mouseProcCmd);
  }//end constructor
}//end class GUI definition
//=======================================================//

//This listener class monitors for mouse presses and
// displays the coordinates of the mouse pointer when the
// mouse is pressed.  The listener object distinguishes
// between two different visual objects on the basis of
// their component names and displays the coordinate
// information on the visual object which generated the
// mouse event.

class MouseProc extends MouseAdapter{
  //save references to the objects here
  MyFrame refToFrame1,refToFrame2;

  MouseProc(MyFrame inFrame1,MyFrame inFrame2){//construct
    refToFrame1 = inFrame1;//save references to the frames
    refToFrame2 = inFrame2;
  }//end constructor

  //Override the mousePressed() method to respond whenever
  // the mouse is pressed on one of the frame objects.
  // Distinguish between the two frame objects using the
  // component name and display the coordinates of the
  // mouse on the correct object.
  public void mousePressed(MouseEvent e){
    if( e.getComponent().getName().compareTo("Frame1")
                                                  == 0 ){
      //Get X and Y coordinates of mouse pointer
      // and store in the Frame object
      refToFrame1.xCoor = e.getX();
      refToFrame1.yCoor = e.getY();
      //display coordinate information
      refToFrame1.repaint();
    }else{
      //Get X and Y coordinates of mouse pointer
      //and store in the Frame object
      refToFrame2.xCoor = e.getX();
      refToFrame2.yCoor = e.getY();
      //display coordinate information
      refToFrame2.repaint();
    }//end if-else
  }//end mousePressed()
}//end class MouseProc
//=======================================================//

//The following listener is used to terminate the program
// when the user closes either frame object. Note that
// class extends the adapter class
class WProc1 extends WindowAdapter{
  public void windowClosing(WindowEvent e){
    System.exit(0);
  }//end windowClosing()
}//end class WProc1
```

```
//=========================================================//
```

.

Another Sample Program

A single Listener object can be registered to process events of a given class generated by two or more different visual objects.

Unique names can be assigned to the objects when they are instantiated.

When an event occurs, the code in the Listener object can obtain the name of the visual object that generated the event and use it to distinguish among the objects.

The name of the object which generated the event can be determined using the following statement where

- **e** is the local name of the object passed into the low-level event-handling method, and
- **"Frame"** is a string being tested against the component name of a visual object that may have generated the event.

```
if( e.getComponent().getName().compareTo("Frame") == 0 ){}
```

Although the previous program used two visual objects of the same type, there is no reason that the visual objects have to be of the same type, as long as all of the visual objects which share the **Listener** object are capable of generating events of the event class for which the Listener is designed.

This is illustrated in the following program which modifies the previous program to use

- a visual Frame object and
- a visual Window object,

instead of two Frame objects.

The ability to display the coordinates of mouse clicks was also removed for simplification of the program. The operation of the program is discussed in the comments.

Note that an improved version of this program is presented at the end of this lesson. The improved version does not require the establishment of **Source** object names, and does not require the passing of references to the constructor for the Window Listener in order to distinguish between between the two sources. You should become familiar with the methodology used in the improved version as well as the methodology used in the following version.

```
/*File Event11.java Copyright 1997, R.G.Baldwin
```

```
This program is designed to be compiled and run under
JDK 1.1

The program illustrates the sharing of a single listener
object between two different visual components of
different types.

The program detects mouse events occurring on either a
visual Frame object or on a visual Window object.  It
distinguishes between the two objects on the basis of the
component name, and displays a message indicating which
object generated the event.

Clicking inside the Frame object but outside the Window
object produces the following message:

Got mousePressed event from Frame object

Clicking inside the Window object produces the following
message:

Got mousePressed event from Window object

Closing the Frame object produces the following message
and terminates the program:

Got windowClosing event from Frame object

These results were produced using JDK 1.1.3, running under
Windows 95.
**********************************************************/

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

public class Event11 {
  public static void main(String[] args){
    GUI gui = new GUI();//instantiate a GUI
  }//end main
}//end class Event11
//=========================================================//

//The following class is used to instantiate a graphical
// user interface object.
class GUI {
  public GUI(){//constructor
    //Create a visual Frame object and name it Frame
    Frame myFrame = new Frame();
    myFrame.setSize(200,300);
    myFrame.setTitle("Copyright 1997, R.G.Baldwin");
    myFrame.setName("Frame");
    myFrame.setVisible(true);

    //Create a visual Window object inside the Frame
    // object and name it Window
    Window myWindow = new Window(myFrame);
```

```
    myWindow.setSize(100,100);
    myWindow.setName("Window");
    myWindow.setVisible(true);

    //Instantiate and register a Listener object which
    // will process mouse events on either the Frame
    // object or the Window object.
    MouseProc mouseProcCmd = new MouseProc();
    myFrame.addMouseListener(mouseProcCmd);
    myWindow.addMouseListener(mouseProcCmd);

    //Instantiate and register a Listener object which
    // will display a message and terminate the program
    // when the user closes the Frame object
    WProc1 winProcCmd1 = new WProc1();
    myFrame.addWindowListener(winProcCmd1);
  }//end constructor
}//end class GUI definition
//=========================================================//

//This listener class monitors for mouse presses and
// displays a message when a mousEpressed() event occurs on
// either the Frame object or the Window object.  The
// message identifies which visual object generated
// the event. The listener object distinguishes between
// the two visual objects on the basis of their component
// names.

class MouseProc extends MouseAdapter{

  //Override the mousePressed() method to respond whenever
  // the mouse is pressed on one of the visual objects.
  public void mousePressed(MouseEvent e){
    if( e.getComponent().getName().compareTo("Frame")
                                                    == 0 ){
      System.out.println(
              "Got mousePressed event from Frame object");
    }//end if
    if( e.getComponent().getName().compareTo("Window")
                                                    == 0 ){
      System.out.println(
             "Got mousePressed event from Window object");
    }//end if
  }//end mousePressed()
}//end class MouseProc
//=========================================================//

//The following listener is used to display a message and
// terminate the program when the user closes the Frame
// object.
class WProc1 extends WindowAdapter{
  public void windowClosing(WindowEvent e){
    System.out.println(
             "Got windowClosing event from Frame object");
    System.exit(0);
  }//end windowClosing()
```

```
}//end class WProc1
//=========================================================//
```

.

Review

Q - Write a Java application that displays two Frame objects on the screen. Each object has a width of 300 pixels and a height of 200 pixels.

One object is located in the upper left-hand corner of the screen. The top left-hand corner of the other object barely touches the bottom right-hand corner of the first object.

Make your name and the name of each object appear in the banner at the top of each object. Do not assign names to the objects.

Cause the two objects to share a single Listener object to respond to mouse events.

Whenever the mouse is pressed internal to either object, the coordinates of the mouse pointer are displayed near the pointer on that object with the horizontal coordinate being displayed first followed by the vertical coordinate. The two coordinate values are separated by a comma and a space.

Also cause the two objects to share a single Listener object that will terminate the program whenever the user clicks the "close" button on either object.

Make certain that your application terminates and returns control to the operating system when the user clicks on the "close" button in the upper right-hand corner of the object.

A - See program below.

```
/*File SampProg120.java from lesson 82
Copyright 1997, R.G.Baldwin
*/

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

public class SampProg120 {
  public static void main(String[] args){
    GUI gui = new GUI();
  }//end main
}//end class SampProg120
//---------------------------------------------------------

//Subclass Frame in order to override the paint method.
class MyFrame extends Frame{
```

```
  int xCoor;
  int yCoor;

  MyFrame(){//constructor
    setTitle("Baldwin " + this.getName());
    setSize(300,200);
  }//end constructor

  public void paint(Graphics g){//override paint() method
    //display coordinate information on the object
    g.drawString("" + xCoor + ", " + yCoor, xCoor, yCoor);
  }//end paint()
}//end class MyFrame
//-----------------------------------------------------------

class GUI {
  public GUI(){//constructor
    //Create two visual objects of type MyFrame, specify
    // their locations, and make them visible.
    MyFrame myFrame1 = new MyFrame();
    myFrame1.setLocation(new Point(0,0));
    myFrame1.setVisible(true);

    MyFrame myFrame2 = new MyFrame();
    myFrame2.setLocation(new Point(300,200));
    myFrame2.setVisible(true);

    //Instantiate and register Listener object which will
    // terminate the program when the user closes either
    // window.
    WProc1 winProcCmd1 = new WProc1();
    myFrame1.addWindowListener(winProcCmd1);
    myFrame2.addWindowListener(winProcCmd1);

    //Instantiate and register Listener object which will
    // process mouse events on either MyFrame object.
    MouseProc mouseProcCmd =
      new MouseProc(myFrame1,myFrame2);
    myFrame1.addMouseListener(mouseProcCmd);
    myFrame2.addMouseListener(mouseProcCmd);
  }//end constructor
}//end class GUI definition
//-----------------------------------------------------------

//This listener class monitors for mouse presses and
// displays the coordinates of the mouse pointer when the
// mouse is pressed.  The listener object distinguishes
// between two different visual objects on the basis of
// their component names and displays the coordinate
// information on the visual object which generated the
// mouse event.

class MouseProc extends MouseAdapter{
  //save references to the objects here
  MyFrame refToFrame1,refToFrame2;
```

```
  MouseProc(MyFrame inFrame1,MyFrame inFrame2){//constructor
    refToFrame1 = inFrame1;//save references to the frames
    refToFrame2 = inFrame2;
  }//end constructor

  //Override the mousePressed() method to respond whenever
  // the mouse is pressed on one of the frame objects.
  // Distinguish between the two frame objects using the
  // component name and display the coordinates of the
  // mouse on the correct object.
  public void mousePressed(MouseEvent e){
     if( e.getComponent().getName().
          compareTo(refToFrame1.getName()) == 0)
    {
      refToFrame1.xCoor = e.getX();
      refToFrame1.yCoor = e.getY();
      //display coordinates on Frame1
      refToFrame1.repaint();
    }else{
      refToFrame2.xCoor = e.getX();
      refToFrame2.yCoor = e.getY();
      //display coordinates on Frame2
      refToFrame2.repaint();
    }//end if-else
  }//end mousePressed()
}//end class MouseProc
//-----------------------------------------------------------

//The following listener is used to terminate the program
// when the user closes either frame object.
class WProc1 extends WindowAdapter{
  public void windowClosing(WindowEvent e){
    System.exit(0);
  }//end windowClosing()
}//end class WProc1
```

Q - Write a Java application that displays two Frame objects on the screen. Place a red Panel object in the upper-left quadrant of the second Frame object.

Each Frame object has a width of 300 pixels and a height of 200 pixels.

One Frame object is located in the upper left-hand corner of the screen. The top left-hand corner of the second Frame object barely touches the bottom right-hand corner of the first Frame object.

Make your name and the name of each Frame object appear in the banner at the top of each Frame object.

Cause the first Frame object and the Panel object to share a single Listener object to respond to mouse events. Do not allow the second Frame object (which contains the Panel object) to share the Listener object for mouse events.

Whenever the mouse is pressed internal to the first Frame object, or on the red portion of the Panel object, the coordinates of the mouse pointer are displayed near the pointer on that object with the horizontal coordinate being displayed first followed by the vertical coordinate. The two coordinate values are separated by a comma and a space.

Whenever the mouse is pressed internal to the second Frame object, but not on the red Panel object, coordinate values are not displayed.

Also cause the two Frame objects to share a single Listener object that will terminate the program whenever the user clicks the "close" button on either Frame object.

Make certain that your application terminates and returns control to the operating system when the user clicks on the "close" button in the upper right-hand corner of either Frame object.

A - See program below.

```
/*File SampProg121.java from lesson 82
Copyright 1997, R.G.Baldwin
*/

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

public class SampProg121 {
  public static void main(String[] args){
    GUI gui = new GUI();
  }//end main
}//end class SampProg121
//---------------------------------------------------------

//Subclass Frame in order to override the paint method.
class MyFrame extends Frame{
  int xCoor;
  int yCoor;

  MyFrame(){//constructor
    setTitle("Baldwin " + this.getName());
    setSize(300,200);
  }//end constructor

  public void paint(Graphics g){//override paint() method
    //display coordinate information on the object
    g.drawString("" + xCoor + ", " + yCoor, xCoor, yCoor);
  }//end paint()
}//end class MyFrame
//---------------------------------------------------------

//Subclass Panel in order to override the paint method.
class MyPanel extends Panel{
  int xCoor;
  int yCoor;
```

```
  MyPanel(){//constructor
    setBounds(new Rectangle(0,0,150,100));
    setBackground(Color.red);
  }//end constructor

  public void paint(Graphics g){//override paint() method
    //display coordinate information on the object
    g.drawString("" + xCoor + ", " + yCoor, xCoor, yCoor);
  }//end paint()
}//end class MyFrame
//-----------------------------------------------------------

class GUI {
  public GUI(){//constructor
    //Instantiate an object of type MyPanel
    MyPanel myPanel = new MyPanel();

    //Instantiate two objects of type MyFrame, specify
    // their locations, and make them visible.  Place the
    // MyPanel object in the second MyFrame object.
    MyFrame myFrame1 = new MyFrame();
    myFrame1.setLocation(new Point(0,0));
    myFrame1.setVisible(true);

    MyFrame myFrame2 = new MyFrame();
    myFrame2.setLayout(null);
    myFrame2.setLocation(new Point(300,200));
    myFrame2.add(myPanel);
    myFrame2.setVisible(true);

    //Instantiate and register a Listener object which will
    // terminate the program when the user closes either
    // window.
    WProc1 winProcCmd1 = new WProc1();
    myFrame1.addWindowListener(winProcCmd1);
    myFrame2.addWindowListener(winProcCmd1);

    //Instantiate and register a Listener object which will
    // process mouse events on either the MyFrame object
    // or the myPanel object.
    MouseProc mouseProcCmd =
      new MouseProc(myFrame1,myPanel);
    myFrame1.addMouseListener(mouseProcCmd);
    myPanel.addMouseListener(mouseProcCmd);

  }//end constructor
}//end class GUI definition
//-----------------------------------------------------------

//This listener class monitors for mouse presses and
// displays the coordinates of the mouse pointer when the
// mouse is pressed.  The listener object distinguishes
// between two different visual objects on the basis of
// their component names and displays the coordinate
// information on the visual object which generated the
```

```
// mouse event.

class MouseProc extends MouseAdapter{
  //save references to the objects here
  MyFrame refToFrame1;
  MyPanel refToPanel;

  MouseProc(MyFrame inFrame1,MyPanel inPanel){//constructor
    refToFrame1 = inFrame1;//save references to the frames
    refToPanel = inPanel;
  }//end constructor

  //Override the mousePressed() method to respond whenever
  // the mouse is pressed on one of the frame objects.
  // Distinguish between the two frame objects using the
  // component name and display the coordinates of the
  // mouse on the correct object.
  public void mousePressed(MouseEvent e){
     if( e.getComponent().getName().
          compareTo(refToFrame1.getName()) == 0)
    { //display coordinates on the Frame object
      refToFrame1.xCoor = e.getX();
      refToFrame1.yCoor = e.getY();
      refToFrame1.repaint();
    }else{//display coordinates on the Panel object
      refToPanel.xCoor = e.getX();
      refToPanel.yCoor = e.getY();
      refToPanel.repaint();
    }//end if-else
  }//end mousePressed()
}//end class MouseProc
//-----------------------------------------------------------

//The following listener is used to terminate the program
// when the user closes either frame object.
class WProc1 extends WindowAdapter{
  public void windowClosing(WindowEvent e){
    System.exit(0);
  }//end windowClosing()
}//end class WProc1
```

Q - Rewrite the program named Event10 and eliminate the requirement to pass parameters to the constructors for the event Listener objects.

A - See solution below.

```
/*File Event10A.java Copyright 1997, R.G.Baldwin
Rewrite the program named Event10 and eliminate the
requirement to pass parameters to the constructors for the
event Listener objects.

This program is designed to be compiled and run under
```

```
JDK 1.1

The program illustrates the sharing of a single listener
object among two different visual components of the same
type.

The program detects mouse events occurring on either of two
different Frame objects.  It distinguishes between the two
objects and displays the coordinates of a mouse click on
whichever object experienced the mouse click.

These results were produced using JDK 1.1.3 running under
Windows 95.

Note that when the program starts, the two visual
components overlay one another.  It is necessary to move
one of them in order to access the other.
*/
//=======================================================//

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

public class Event10A {
  public static void main(String[] args){
    //instantiate a Graphical User Interface object
    GUI gui = new GUI();
  }//end main
}//end class Event10A
//=======================================================//

//Subclass Frame in order to override the paint method.
class MyFrame extends Frame{
  int xCoor;
  int yCoor;

  MyFrame(){//constructor
    setTitle("Copyright 1997, R.G.Baldwin");
    setSize(300,200);
  }//end constructor

  public void paint(Graphics g){
    //display coordinate information on the visual object
    g.drawString("" + xCoor + ", " + yCoor, xCoor, yCoor);
  }//end paint()
}//end class MyFrame
//=======================================================//

//The following class is used to instantiate a graphical
// user interface object.
class GUI {
  public GUI(){//constructor
    //Create two visual objects of type MyFrame and make
    // them visible.
    MyFrame myFrame1 = new MyFrame();
    myFrame1.setVisible(true);
```

```
    MyFrame myFrame2 = new MyFrame();
    myFrame2.setVisible(true);

    //Instantiate and register Listener object which will
    // terminate the program when the user closes either
    // window.
    WProc1 winProcCmd1 = new WProc1();
    myFrame1.addWindowListener(winProcCmd1);
    myFrame2.addWindowListener(winProcCmd1);

    //Instantiate and register Listener object which will
    // process mouse events on either MyFrame object.
    MouseProc mouseProcCmd = new MouseProc();
    myFrame1.addMouseListener(mouseProcCmd);
    myFrame2.addMouseListener(mouseProcCmd);
  }//end constructor
}//end class GUI definition
//=======================================================//

//This listener class monitors for mouse presses and
// displays the coordinates of the mouse pointer when the
// mouse is pressed.  The listener object distinguishes
// between two different visual objects and displays the
// coordinate information on the visual object which
// generated the mouse event.

class MouseProc extends MouseAdapter{
  //Override the mousePressed() method to respond whenever
  // the mouse is pressed on one of the frame objects.
  public void mousePressed(MouseEvent e){
    //Get X and Y coordinates of mouse pointer and store in
    // the Frame object.  Distinguish between the two
    // components on the basis of the source of the event.
    // Note that the following two formulations for X and Y
    // can be used to produce the same results in this
    // situation.
    ((MyFrame)e.getComponent()).xCoor = e.getX();
    ((MyFrame)e.getSource()).yCoor = e.getY();

    //display coordinate information
    e.getComponent().repaint();
  }//end mousePressed()
}//end class MouseProc
//=======================================================//

//The following listener is used to terminate the program
// when the user closes either frame object.
class WProc1 extends WindowAdapter{
  public void windowClosing(WindowEvent e){
    System.exit(0);
  }//end windowClosing()
}//end class WProc1
//=======================================================//
```

Q - Rewrite the program named Event11 and eliminate the requirement to pass parameters to the constructor for the listener objects.

A - See solution below.

```
/*File Event11A.java Copyright 1997, R.G.Baldwin
Rewrite the program named Event11 and eliminate the
requirement to pass parameters to the constructor for the
listener objects.

This program is designed to be compiled and run under
JDK 1.1

The program illustrates the sharing of a single listener
object between two different visual components of different
types.

The program detects mouse events occurring on either a
visual Frame object or on a visual Window object.  It
distinguishes between the two objects and displays a
message indicating which object generated the event.

Clicking inside the Frame object but outside the Window
object produces the following message:

Got mousePressed event from Frame object

Clicking inside the Window object produces the following
message:

Got mousePressed event from Window object

Closing the Frame object produces the following message and
terminates the program:

Got windowClosing event from Frame object

These results were produced using JDK 1.1.3 running under
Windows 95.
*/
//=======================================================//

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

public class Event11A {
  public static void main(String[] args){
    //instantiate a Graphical User Interface object
    GUI gui = new GUI();
  }//end main
}//end class Event11A
//=======================================================//

//The following class is used to instantiate a graphical
```

```
// user interface object.
class GUI {
  public GUI(){//constructor
    //Create a visual Frame object
    Frame myFrame = new Frame();
    myFrame.setSize(200,300);
    myFrame.setTitle("Copyright 1997, R.G.Baldwin");
    myFrame.setVisible(true);

    //Create a visual Window object inside the Frame object
    Window myWindow = new Window(myFrame);
    myWindow.setSize(100,100);
    myWindow.setVisible(true);

    //Instantiate and register a Listener object which will
    // process mouse events on either the Frame object or
    // the Window object.
    MouseProc mouseProcCmd = new MouseProc();
    myFrame.addMouseListener(mouseProcCmd);
    myWindow.addMouseListener(mouseProcCmd);

    //Instantiate and register a Listener object which will
    // display a message and terminate the program when the
    // user closes the Frame object
    WProc1 winProcCmd1 = new WProc1();
    myFrame.addWindowListener(winProcCmd1);
  }//end constructor
}//end class GUI definition
//=======================================================//

//This listener class monitors for mouse presses and
// displays a message when a mousePressed() event occurs on
// either the Frame object or the Window object.  The
// message identifies which visual object generated the
// event. The listener object distinguishes between the two
// visual objects.

class MouseProc extends MouseAdapter{

  //Override the mousePressed() method to respond whenever
  // the mouse is pressed on one of the visual objects.
  public void mousePressed(MouseEvent e){
    System.out.print("Got mousePressed event from ");
    if(e.getSource().toString().indexOf("Frame") >= 0)
      System.out.println("Frame object");
    else
      System.out.println("Window object");
  }//end mousePressed()
}//end class MouseProc
//=======================================================//

//The following listener is used to display a message and
// terminate the program when the user closes the Frame
// object.
class WProc1 extends WindowAdapter{
  public void windowClosing(WindowEvent e){
```

```
    System.out.println(
               "Got windowClosing event from Frame object");
    System.exit(0);
  }//end windowClosing()
}//end class WProc1
//=========================================================//
```

-end-