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

Event Handling in JDK 1.1, Using Abbreviated Inner Classes

Java Programming, Lecture Notes # 94, Revised 02/21/98.

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 27, 1997 using the software and documentation in JDK 1.1.

One of the new features in JDK 1.1 in the capability to create *inner classes* and also to use a somewhat cryptic abbreviated syntax for the *definition of anonymous classes* and the *instantiation of anonymous objects* from those classes.

The instantiation of **Listener** objects is a strong candidate for the use of anonymous classes and objects as provided by the *inner class* capability.

An entire series of lessons will be dedicated to a discussion of inner classes. The purpose of this lesson is simply to introduce you to the concept so that you will know what you are seeing whenever you encounter code using the abbreviated *inner class* syntax.

Two sample programs will be presented and discussed. The first will be implemented using Inner Classes, but will not use the anonymous class, anonymous object syntax.

Then the same program will be presented after having been modified to use the anonymous class, anonymous object syntax.

This will make it possible for you to compare the two and hopefully gain a better understanding the abbreviated syntax version.

The topic of *inner classes* is a broad topic. The discussion of the technical aspects of inner classes will be very brief. Also there are many aspects of inner classes that won't be discussed even briefly.

You will probably need to refer to either the JDK 1.1 documentation package, or the lessons dedicated to inner classes to fully understand the material presented in this lesson.

According to the JDK 1.1 Documentation:

- "Inner classes allow classes to be defined in any scope. In previous releases, Java supported only top-level classes, which must be members of packages. In this release, the programmer can now define inner classes as members of other classes, locally within a block of statements, or (anonymously) within an expression."

First Sample Program

As mentioned earlier, in order to avoid some of the cryptic aspects of anonymous classes/objects, the first sample program makes use of inner classes in a relatively straightforward manner. Classes are defined inside of classes and those class definitions are used to instantiate the necessary objects.

This differs significantly from previous lessons on the Delegation Event Model where the classes needed to instantiate **Listener** objects were top-level classes (direct subclasses of the class **object** or subclasses of **adapter** classes such as **MouseAdapter**).

In this program, the classes needed to instantiate **Listener** objects, as well as some others are nested within other classes.

Discussion of First Sample Program

The inner class capability does not exist in JDK 1.0. Therefore, this program is designed to be compiled and run under JDK 1.1

The program combines the use of inner classes with event handling under the JDK 1.1 Delegation Event Handling model.

The controlling class is named Event23. An inner class named **GUI** is defined inside the class named **Event23** and an object of that class is instantiated inside the **main()** method.

In addition, two "data processing" methods named **sing()** and **whistle()** are defined inside the controlling class. They are defined in such a way as to be separated from the event handling activity except that they are invoked by the event handler methods.

Three additional inner classes are defined inside the inner class named **GUI**. They are named **SingActionListener**, **WhistleActionListener**, and **Terminator**. All three of these are **Listener** classes. The first two implement the **ActionListener** interface while the third extends the **WindowListener** adapter.

The first two override the **actionPerformed()** method of the **ActionListener** interface while the third overrides the **windowClosing()** method of the **WindowListener** interface.

The class named **GUI** extends **Frame**. Two **Button** objects labeled *Sing* and *Whistle* are instantiated in the **GUI** constructor. They are referenced by variables named **singButton** and **whistleButton** respectively. Both buttons are added to the **Frame** object.

Although this sample program does not use anonymous classes, it does use anonymous objects. Anonymous **Listener** objects of the classes **SingActionListener** and **WhistleActionListener** are instantiated and registered for event handling on the two **Button** objects having similar names.

Another anonymous **Listener** object of the class **Terminator** is registered for event handling on the **Frame** object of class **GUI**.

When the program starts, a **Frame** object containing the two buttons appears on the screen. When the user clicks on the button labeled *Sing*, the event handler registered to listen for **Action** events on that button invokes the method named **sing()** which in turn displays the message

"I am singing, Tra la la"

When the user clicks on the button labeled *Whistle*, the event handler registered to listen for **Action** events on that button invokes the method named **whistle()** which in turn displays the message

"I am whistling, Tweet, Tweet, Tweet"

When the user clicks the "close" box on the **Frame**, the event handler registered to listen for **windowClosing()** events on the **Frame** terminates the program.

These results were produced using JDK 1.1 running under Windows 95. */

Program Listing of First Sample Program

A listing of the program with additional comments follows:

```
/*File Event23.java Copyright 1997, R.G.Baldwin
Revised 9/17/97
```

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

The program illustrates the use of Inner Classes with event
handling under the JDK 1.1 Delegation Event Handling model.

The controlling class is named Event23.  An Inner Class
named GUI is defined inside the class named Event23 and an
object of that class is instantiated inside the main()
method.

In addition, two "data processing" methods named sing() and
whistle() are defined inside the controlling class.  They
are defined in such a way as to be separated from the event
handling activity except that they are invoked by the event
handler methods.

Three more Inner Classes are defined inside the Inner Class
named GUI. They are named SingActionListener,
WhistleActionListener, and Terminator.  All three of these
are Listener classes.  The first two implement the
ActionListener interface while the third extends the
WindowListener adapter.

The first two override the actionPerformed() method of the
ActionListener interface while the third overrides the
windowClosing() method of the WindowListener interface.

The class named GUI extends Frame.  Two Button objects
labeled Sing and Whistle are instantiated in the GUI
constructor. They are referenced by variables named
singButton and whistleButton respectively.  Both buttons
are added to the Frame object.

Anonymous Listener objects of the classes
SingActionListener and WhistleActionListener are
instantiated and registered for event handling on the two
Button objects having similar names.

An anonymous Listener object of the class Terminator is
registered for event handling on the Frame object of class
GUI.

When the program starts, a Frame object containing the two
buttons appears on the screen.  When the user clicks on the
button labeled Sing, the event handler registered to
listen for Action events on that button invokes the method
named sing() causing the message

"I am singing, Tra la la"

to be displayed on the screen.

When the user clicks on the button labeled Whistle, the
event handler registered to listen for Action events on
that button invokes the method named whistle() causing the
```

```
message

"I am whistling, Tweet, Tweet, Tweet"

to be displayed on the screen.

When the user clicks the "close" box on the Frame, the
event handler registered to listen for windowClosing()
events on the Frame terminates the program.

This version uses long form notation as opposed to the use
of anonymous inner classes.  See the program named Event22
for a version that uses the more cryptic abbreviated
notation attributable to the use of anonymous inner
classes.

These results were produced using JDK 1.1 running under
Windows 95.
*/
//===========================================================

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

public class Event23 {
  //The following two methods are invoked directly from
  // code in methods defined in classes which are inner-
  // classes of this class.
  void sing() {System.out.println(
                               "I am singing, Tra la la");}
  void whistle() {System.out.println(
                     "I am whistling, Tweet Tweet Tweet");}

  //---------------------------------------------------------
  static public void main(String[] args){
    Event23 app = new Event23();
    //Note the syntax in the following statement where the
    // new operator is joined to the reference to the
    // object of a class in which the GUI inner class is
    // defined in order to instantiate an object of the
    // GUI class.
    GUI gui = app.new GUI();
  }//end main()

  //---------------------------------------------------------
  //Note that the following GUI class is defined inside
  // the Event23 class and thus is an inner-class of the
  // Event23 class.
  class GUI extends Frame{

    //-------------------------------------------------------
    //Note that the SingActionListener class is defined
    // inside the GUI class which is defined inside the
    // Event23 class.
    class SingActionListener implements ActionListener{
      //Implement the actionPerformed method which is
      // declared in the ActionListener interface.
      public void actionPerformed(ActionEvent e){
        //Note that because this class is defined inside
        // the GUI class which is defined inside the
        // Event23 class, this method has direct
        // access to the members of the Event23 class.
        // Therefore, this method can directly access the
        // method named sing() without having to access
        // it via an object of type Event23.
        sing();
      }//end actionPerformed()
    }//end SingActionListener class defined inside GUI

    //-------------------------------------------------------
    //Note that the WhistleActionListener class is defined
    // inside the GUI class which is defined inside the
    // Event23 class.
    class WhistleActionListener implements ActionListener{
      public void actionPerformed(ActionEvent e){
        //As mentioned earlier, this inner-class of the
        // has direct access to the members of the
        // Event23 class without the requirement to
        // access via an object of type Event23.
        whistle();
      }//end actionPerformed()
    }//end WhistleActionListener class defined inside GUI

    //-------------------------------------------------------
    //Note that the Terminator class is defined inside the
    // GUI class which is defined inside the Event23 class.
    class Terminator extends WindowAdapter{
      public void windowClosing(WindowEvent e){
        System.exit(0);
      }//end windowClosing()
    }//end class Terminator defined inside GUI class

    //-------------------------------------------------------
    public GUI(){//constructor for GUI class
      setTitle("Copyright 1997, R.G.Baldwin");

      Button singButton;
      add(singButton = new Button("Sing"),"North");
      singButton.addActionListener(
                                new SingActionListener() );

      Button whistleButton;
      add(whistleButton = new Button("Whistle"),"South");
      whistleButton.addActionListener(
                             new WhistleActionListener() );

      //Register a Listener object for event handling on
      // the Frame object of class GUI.
      this.addWindowListener(new Terminator());

      //Set frame size and make visible
      setSize(300,100);
```

```
      setVisible(true);
    }//end GUI constructor
    //-------------------------------------------------------
  }//end class GUI which is defined inside Event23 class
  //---------------------------------------------------------
}//end class Event23
//===========================================================
```

Second Sample Program

The second sample program is similar to the first, except that it makes use of anonymous classes for instantiation of the **Listener** objects.

Discussion of Second Sample Program

In order to make small adapter classes (such as **Listener** classes) as concise as possible, Java 1.1 allows an abbreviated notation for local objects.

A single expression combines

- the definition of an anonymous class with
- the allocation of the instance.

Here is a code fragment from the second sample program which illustrates this concept.

```
singButton.addActionListener(
      new ActionListener(){
        public void actionPerformed(ActionEvent e){
          sing();//call the sing() method
        }//end actionPerformed()
      }//end ActionListener
    );//end addActionListener()
```

In this case, the code defines a new anonymous class that automatically implements the **ActionListener** interface (without use of the keyword **implements**) and automatically instantiates an anonymous object of that new class.

In this case, the word *anonymous* indicates that neither the class nor the object are named.

In previous lessons we have instantiated anonymous objects by using the **new** operator and making a call to the constructor for the object as part of a larger overall expression. However, we have not done that using classes that had no name.

Another interesting aspect of this syntax is the following code fragment which looks like a constructor call in a typical object instantiation.

```
new ActionListener(){...
```

However, **ActionListener** is not a class, it is an interface and since an interface cannot contain any fully defined methods, it doesn't seem that it could have a constructor (but it works anyway).

Pay particular attention to the positions of the open and close parentheses which define the argument list for the **addActionListener()** method.

The definition of the anonymous class as well as the instantiation of the anonymous object are both included in the argument list of **addActionListener()**.

The addActionListener**()** method is used to register this anonymous object to listen for **actionPerformed()** events on the **Button** object named **singButton**. Whenever an **Action** event occurs on that specific button, the overridden **actionPerformed()** method invokes the "data processing" method named **sing()**. Because this is an inner-class of the outer-class in which the method **sing()** is defined, code in the inner-class has direct access to the method named **sing()** without the requirement to instantiate an object of the outer-class.

When using the abbreviated syntax, a **new** operator expression can end with a class body. According to the JDK 1.1 documentation:

- "The effect of this is to take the class (or interface) named after the **new** token, and subclass it (or implement it) with the given body. The resulting anonymous inner-class has the same meaning as if the programmer had defined it locally, with a name, in the current block of statements."

It is important to note that an anonymous class can have initializers but it cannot have a constructor. The argument list of the associated **new** expression is implicitly passed to a constructor of the superclass.

For the second sample program which follows, the controlling class is named **Event22**. As before, an inner class named **GUI** is defined inside the class named **Event22** and an object of that class is instantiated inside the **main()** method.

Also as before, two "data processing" methods named **sing()** and **whistle()** are defined inside the controlling class. They are defined in such a way as to be separated from the event handling activity except that they are invoked by the event handler methods.

However, unlike the first sample program, abbreviated notation is used to anonymously define the classes for and instantiate three anonymous listener objects inside the inner class named **GUI**. The first two implement the **ActionListener** interface while the third extends the **WindowListener** adapter class.

As before, the first two override the **actionPerformed()** method of the **ActionListener** interface while the third overrides the **windowClosing()** method of the **WindowListener** interface.

The class named **GUI** extends **Frame**. Two **Button** objects labeled *Sing* and *Whistle* are instantiated in the **GUI** constructor. They are referenced by variables named **singButton** and **whistleButton** respectively. Both buttons are added to the **Frame** object.

The first two anonymous listener objects mentioned above are registered to listen for **actionPerformed()** events on the two buttons. The third anonymous listener object is registered to listen for **windowClosing()** events on the **Frame** object.

When the program starts, a **Frame** object containing the two buttons appears on the screen. When the user clicks on the button labeled *Sing*, the event handler registered to listen for **Action** events on that button invokes the method named **sing()** which displays the message:

"I am singing, Tra la la"

When the user clicks on the button labeled *Whistle*, the event handler registered to listen for **Action** events on that button invokes the method named **whistle()** which displays the message:

"I am whistling, Tweet, Tweet, Tweet"

When the user clicks the "close" box on the **Frame**, the event handler registered to listen for **windowClosing()** events on the Frame terminates the program.

These results were produced using JDK 1.1 running under Windows 95.

Program Listing of Second Sample Program

A program listing with additional comments follows:

```
/*File Event22.java Copyright 1997, R.G.Baldwin
Revised 9/17/97
This program is designed to be compiled and run under
JDK 1.1

The program illustrates the use of Inner Classes with event
handling under the JDK 1.1 Delegation Event Handling model.

The use of anonymous Inner Classes is also illustrated.
See the program named Event23 for an example of Inner
Classes that does not use anonymous Inner Classes.

The controlling class is named Event22.  An Inner Class
named GUI is defined inside the class named Event22 and an
object of that class is instantiated inside the main()
method.

In addition, two "data processing" methods named sing() and
whistle() are defined inside the controlling class named
Event22.  They are defined in such a way as to be separated
from the event handling activity except that they are
invoked by the event handler methods.
```

```
Abbreviated notation is used to anonymously define the
classes for and instantiate three anonymous listener
objects inside the Inner Class named GUI. The first two
implement the ActionListener interface while the third
extends the WindowListener adapter.

The first two implement the actionPerformed() method of the
ActionListener interface while the third overrides the
windowClosing() method of the WindowListener interface.

The class named GUI extends Frame.  Two Button objects
labeled Sing and Whistle are instantiated in the GUI
constructor. They are referenced by variables named
singButton and whistleButton respectively.  Both buttons
are added to the Frame object.

The first two anonymous listener objects mentioned above
are registered to listen for actionPerformed() events on
the two buttons respectively.  The third anonymous listener
object is registered to listen for windowClosing() events
on the Frame object.

When the program starts, a Frame object containing the two
buttons appears on the screen.  When the user clicks on the
button labeled Sing, the event handler registered to listen
for Action events on that button invokes the method named
sing() causing the message

"I am singing, Tra la la"

to be displayed on the screen.

When the user clicks one the button labeled Whistle, the
event handler registered to listen for Action events on
that button invokes the method named whistle() causing the
message

"I am whistling, Tweet, Tweet, Tweet"

to be displayed on the screen.

When the user clicks the "close" box on the Frame, the
event handler registered to listen for windowClosing()
events on the Frame terminates the program.

This version of the program uses abbreviated notation to
define anonymous classes and instantiate anonymous listener
objects.  The abbreviated notation is fairly cryptic.  See
the program named Event23 for a version that does the same
thing without using the abbreviated notation.

These results were produced using JDK 1.1 running under
Windows 95.
*/
//===========================================================
import java.awt.*;
import java.awt.event.*;

public class Event22 {
  void sing() {System.out.println(
                               "I am singing, Tra la la");}
  void whistle() {System.out.println(
                     "I am whistling, Tweet Tweet Tweet");}

  //---------------------------------------------------------
  static public void main(String[] args){
    Event22 app = new Event22();
    //Note the following syntax which instantiates an
    // object of the class GUI which is an inner-class of
    // the class Event22.  The new operator is invoked on
    // the object named app which is of class Event22.
    GUI gui = app.new GUI();
  }//end main()

  //---------------------------------------------------------
  //Note that the GUI class is defined inside the Event22
  // class and thus becomes an inner-class of Event22.
  class GUI extends Frame{

    public GUI(){//constructor for GUI inner-class
      this.setTitle("Copyright 1997, R.G.Baldwin");
      Button singButton;
      this.add(singButton = new Button("Sing"),"North");
      Button whistleButton;
      this.add(whistleButton = new Button("Whistle"),
                                                  "South");

      //-----------------------------------------------------
      //The code which follows instantiates three
      // anonymous objects of types ActionListener and
      // WindowAdapter, and registers them for handling
      // events on the two corresponding Button objects and

      // the Frame object.  This code uses the abbreviated
      // syntax which defines the listener classes
      // anonymously (the listener classes do not have
      // class names and the objects instantiated from
      // those classes do not have names).

      //Begin statement -----------------------------------
      singButton.addActionListener(
        //The following object is passed as a parameter
        // to the addActionListener() method.
        new //instantiate anonymous object of the class
          ActionListener(){//anonymous class definition
            //Implement the actionPerformed() method
            // which is declared in the ActionListener
            // interface.
            public void actionPerformed(ActionEvent e){
              //The methods in this inner-class have direct
              // access to the members of the enclosing
              // outer-class named Event22.  Thus, the
              // direct invocation of the sing() method
              // is possible without the requirement to
              // instantiate an object of type Event22.
              sing();//call the sing() method
            }//end actionPerformed()
          }//end ActionListener class definition
        );//end addActionListener() statement
      //End statement ------------------------------------

      //Begin statement ----------------------------------
      whistleButton.addActionListener(
      //See above discussion for explanation of this code
          new ActionListener(){//anonymous class definition
            public void actionPerformed(ActionEvent e){
              //See note above regarding the method sing()
              whistle();//call the whistle() method
            }//end actionPerformed()
          }//end ActionListener
        );//end addActionListener()
      //End statement ------------------------------------

      //Begin statement ----------------------------------
      this.addWindowListener(
      //See above discussion for explanation of this code
           new WindowAdapter(){//anonymous class definition
             public void windowClosing(WindowEvent e){
               System.exit(0);//terminate the program
             }//end windowClosing()
           }//end WindowAdapter
         );//end addWindowListener
      //End statement ------------------------------------

      //----------------------------------------------------
      //Set frame size and make it visible.
      this.setSize(300,100);
      this.setVisible(true);
    }//end GUI constructor
  }//end class GUI
}//end class Event22
//=========================================================
```

Review

Q - Without viewing the solution given below, write a Java application that meets the following specifications:

```
The program must be written using the following skeleton
and inserting any additional code that may be necessary.


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

public class SampProg127 {
//All new code must be inserted after this comment

//Insert the necessary additional code here

//All new code must be inserted before this comment
}//end class SampProg127
```

When the program starts, a Frame object containing a button labeled Button and a Label object containing the string Initial Text in Label appears on the screen. Your name must appear in the title at the top of the frame. When the user clicks the button, the text in the label changes to Ouch.

When the user clicks the "close" box on the Frame, the program terminates and control is properly returned to the operating system.

A - See solution below:

```
/*File SampProg127.java Copyright 1997, R.G.Baldwin
Revised 9/17/97

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

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

public class SampProg127 {

  //--------------------------------------------------------
  static public void main(String[] args){
    SampProg127 app = new SampProg127();
    //Note the syntax in the following statement where the
    // new operator is joined to the reference to the
    // object of a class in which the GUI inner class is
    // defined in order to instantiate an object of the
    // GUI class.
    GUI gui = app.new GUI();
  }//end main()

  //--------------------------------------------------------
  //Note that the following GUI class is defined inside
  // the SampProg127 class and thus is an inner-class of
  // the SampProg127 class.
  class GUI extends Frame{
    //The object referenced by the following reference
    // variable is accessed directly by code in an
    // inner-class of the GUI class.
    Label myLabel;

    //-----------------------------------------------------
    public GUI(){//constructor for GUI class
      setTitle("Copyright 1997, R.G.Baldwin");

      Button myButton;
      this.add(myButton = new Button("Button"),"North");
      myButton.addActionListener(
                                new ButtonActionListener() );

      this.add(myLabel = new Label(
                           "Initial Text in Label"),"South");

      //Register a Listener object for event handling on
      // the Frame object of class GUI.
      this.addWindowListener(new Terminator());

      //Set frame size and make visible
      setSize(300,100);
      setVisible(true);
    }//end GUI constructor

    //-----------------------------------------------------
    //Note that the ButtonActionListener class is defined
    // inside the GUI class which is defined inside the
    // SampProg127 class.
    class ButtonActionListener implements ActionListener{
      //Implement the actionPerformed method which is
      // declared in the ActionListener interface.
      public void actionPerformed(ActionEvent e){
        //Note that because this class is defined inside
        // the GUI class which is defined inside the
        // SampProg127 class, this method has direct
        // access to the members of both the SampProg127
        // class and the GUI class.  Therefore, this method
        // can directly access the reference variable named
        // myLabel without having to access it via an
        // object of type GUI.
        myLabel.setText("Ouch");

      }//end actionPerformed()
    }//end ButtonActionListener class defined inside GUI

    //-----------------------------------------------------
    //Note that the Terminator class is defined inside the
    // GUI class which is defined inside the SampProg127
    // class.
    class Terminator extends WindowAdapter{
      public void windowClosing(WindowEvent e){
        System.exit(0);
      }//end windowClosing()
    }//end class Terminator defined inside GUI class
    //-----------------------------------------------------
  }//end class GUI defined inside SampProg127 class
  //-----------------------------------------------------------
}//end class SampProg127
//===========================================================
```

.

Q - Without viewing the solution given below, write a Java application that meets the following specifications:

```
The program must be written using the following skeleton
and inserting any additional code that may be necessary.


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

public class SampProg128 extends Frame{
//All new code must be inserted after this comment

//Insert the necessary additional code here.

//Do not define any named classes other than the class
// named SampProg128

//All new code must be inserted before this comment
}//end class SampProg128


When the program starts, a Frame object containing a button
labeled Button and a label containing the string
Initial Text in Label appears on the screen.  Your name
must appear in the title at the top of the frame. When the
user clicks the button, the text in the label changes to
Ouch.

When the user clicks the "close" box on the Frame, the
program terminates and control is properly returned to the
operating system.
```

A - See the solution below:

```
/*File SampProg128.java from lesson 94
Revised 02/21/98 to correct an earlier problem which
caused the program to violate the specifications.

Copyright 1997, R.G.Baldwin
This program is designed to be compiled and run under
JDK 1.1

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

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

public class SampProg128 extends Frame {
  Label myLabel;
  //----------------------------------------------------//
  static public void main(String[] args){
    SampProg128 app = new SampProg128();
  }//end main()
  //----------------------------------------------------//

  public SampProg128(){//constructor
    this.setTitle("Copyright 1997, R.G.Baldwin");
    Button myButton;
    this.add(myButton = new Button("Button"),"North");
    this.add(myLabel = new Label(
                       "Initial Text in Label"),"South");
    //--------------------------------------------------//

    //The code which follows instantiates two
    // anonymous objects of types ActionListener and
    // WindowAdapter, and registers them for handling
    // events on the corresponding Button object and
    // the Frame object.  This code uses the abbreviated
    // syntax which defines the listener classes
    // anonymously (the listener classes do not have
    // class names and the objects instantiated from
    // those classes do not have names).

    //Begin statement ------------------------------------
    myButton.addActionListener(
      //The following object is passed as a parameter
      // to the addActionListener() method.
      new //instantiate anonymous object of the class
        ActionListener(){//anonymous class definition
          //Implement the actionPerformed() method
          // which is declared in the ActionListener
          // interface.
          public void actionPerformed(ActionEvent e){
            //The methods in this inner-class have direct
            // access to the members of the enclosing
            // outer-classes named SampProg128.
            // Thus, direct access to the reference
            // variable named myLabel
            // is possible
            myLabel.setText("Ouch");
          }//end actionPerformed()
        }//end ActionListener class definition
      );//end addActionListener() statement
    //End statement --------------------------------------

    //Begin statement ------------------------------------
    this.addWindowListener(
    //See above discussion for explanation of this code
         new WindowAdapter(){//anonymous class definition
```

```
           public void windowClosing(WindowEvent e){
             System.exit(0);//terminate the program
           }//end windowClosing()
         }//end WindowAdapter
       );//end addWindowListener
    //End statement -------------------------------------

    //---------------------------------------------------
    //Set frame size and make it visible.
    this.setSize(300,100);
    this.setVisible(true);
  }//end SampProg128 constructor
}//end class SampProg128
//=========================================================
```

-end-