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

Event Handling in JDK 1.1, Program-Generated Events and the AWTEventMulticaster Class

Java Programming, Lecture Notes # 96, Revised 10/19/98.

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.

JDK 1.1 was formally released on February 18, 1997. This lesson was originally written on March 23, 1997 using the software and documentation in the JDK 1.1 download package.

Introduction

In previous lessons on event handling under JDK 1.1, you have learned how to use the *Source/Listener* concept embodied in the *Delegation Event Model* to handle different types of events generated by visual components on the Graphical User Interface.

In this lesson, we take the subject of event handling one step further by teaching you how to create and dispatch events *under program control* which produce the same response as if the events were caused by actions of the user on visual components. In other words, you will learn how to create and dispatch events under program control which *simulate the behavior of a user*. For example, you might dispatch an event of a given type if a particular character were read from a file.

Although you may not immediately see the need to create and dispatch events under program control, an understanding of this material is critical to an understanding of **Lightweight Components** that will be presented in a subsequent lesson. Therefore, it is very important that you gain a solid understanding of the concept of program-generated events under JDK 1.1 before embarking on your studies of **Lightweight Components**.

In addition, an understanding of this material will give you a much better understanding of what you are really doing when you implement the *Delegation Event Model* using visual components from the **AWT**.

Main Theme

The main theme of this lesson is the use of the **AWTEventMulticaster** class of the *Delegation Event Model.* This class is used to maintain a list of **Listener** objects which are registered to be notified whenever an event of a particular type occurs on a particular object and to cause those **Listener** objects to be notified. In previous lessons, the use of the **AWTEventMulticaster** class has occurred "under the covers." In this lesson, we will provide some exposure as to how that class does what it does.

As is frequently the case, we will use sample programs to illustrate the concepts. In this lesson, we will take it in two steps.

- First we will take a look at a bare-bones sample program which illustrates the essential ingredients of program-generated events.
- Then we will take a look at a more substantive program which illustrates the **AWTEventMulticaster** class of the *Delegation Event Model*.

Essential Ingredients of Program-Generated Events

In order to use program-generated events with the techniques described in this lesson, you will need to define a class that is capable of generating the type of events of interest. In this lesson we concentrate on **Action** events. However, there is no reason that you could not use similar techniques to generate low-level events such as mouse events and key events.

The class must be a subclass of the **Component** class and must include the following three members (as a minimum):

- *An instance variable* that is a reference to a list of registered **Listener** objects. In this program, the **Listener** objects are of type **ActionListener**. The instance variable is of type **ActionListener** and could contain a reference to a single object of that type or a reference to a list of objects of that type.
- *A method* for creating the list mentioned above. In the bare-bones program the method is named **makeActionListenerList()** just to illustrate that the name is not technically important. However, for consistency with documentation on the *Delegation Event Model* (and other good reasons as well), it should be named **addActionListener()**. The list

should be constructed within the body of the method using a call to the **AWTEventMulticaster.add()** method which returns a reference to the list. We will discuss the reasons for doing it this way later.
- *A method* which invokes the appropriate action method of the **Listener** class on the list of registered **Listener** objects. In the sample programs in this lesson, the **Listener** objects are of type **ActionListener** so the appropriate action method is **actionPerformed()**. For other types of **Listener** objects, the appropriate action method would be a different method. In these programs, the method that invokes the action method of the **Listener** object is called **generateActionEvent()**. In the bare-bones program, there is only one **Listener** object in the list. In the second program, there are multiple **Listener** objects in the list.

Bare-Bones Sample Program

This program is designed to be compiled and run under JDK 1.1 This is a bare-bones program that illustrates Program Generated Events under JDK 1.1.

A non-visual object class (named **NonVisual**) is defined. Objects of this class are capable of generating **Action** events.

A single object of the **NonVisual** class is instantiated. A single **ActionListener** class is also defined. An object of this class is instantiated and registered to listen for **Action** events on the **NonVisual** object.

Then the **generateActionEvent()** method is invoked on the **NonVisual** object causing an **Action** event to be trapped and processed by the **ActionListener** object. The processing is simply to display information identifying the **NonVisual** object that generated the event.

The output from the program is:

```
Copyright 1997, R.G.Baldwin
actionPerformed() method invoked on NonVisualObject
```

The program was tested using JDK 1.1 running under Win95.

Interesting Code Fragments from Bare-Bones Sample Program

The first interesting code fragment is the code in the constructor that

- instantiates an object of the new **NonVisual** type,
- invokes the **makeActionListenerList()** method on the **NonVisual** object to register a **Listener** object on the **NonVisual** object, and
- invokes the **generateActionEvent()** method on the **NonVisual** object to cause an **Action** event to be generated by the **NonVisual** object.

```
    NonVisual nonVisualObject = new NonVisual("NonVisualObject");
    nonVisualObject.makeActionListenerList(new AnActionListenerClass());
    nonVisualObject.generateActionEvent();
```

The next interesting code fragment is the declaration of the instance variable in the **NonVisual** class definition that will refer to a list of registered **Listener** objects. As mentioned earlier, this instance variable could just as easily refer to a single object of the **ActionListener** type as to refer to a list of such objects. In fact, in this bare-bones program it does refer to a single instance of an **ActionListener** object as we will discuss in more detail later.

```
 ActionListener actionListener;//list of registered action listener objects
```

The next interesting code fragment is the statement that is used to construct the list of registered **Listener** objects by adding a new object to the list.

The first time this statement is executed in a program, it simply returns a reference to the object being added to the list.

If it is executed a second time, it returns a reference to a list of objects that is being maintained separately (this is explained more fully later).

In the bare-bones program, only one **Listener** object is added to the list. If we were to examine the contents of the reference returned by the **AWTEventMulticaster.add()** method, we would find that it is simply a reference to the object. In a later program, we will see how it refers to a list when more than one object is added to the list.

```
    actionListener = AWTEventMulticaster.add(actionListener, listener);
```

The final interesting code fragment in this bare-bones program is the statement that invokes the **actionPerformed()** method of the **ActionListener** object, or more properly invokes that method on all the objects in the list of registered **ActionListener** objects.

Fortunately, all we have to do is to invoke the method once on the instance variable that refers to the list and the system takes care of invoking the method on all the objects in the list. This is the **central feature** of the **AWTEventMulticaster** class.

As you can see, this code instantiates an object of type **ActionEvent** and passes that object as a parameter to the **actionPerformed()** method thus satisfying the signature requirements of the **actionPerformed()** method.

```
    actionListener.actionPerformed(
        new ActionEvent(this, ActionEvent.ACTION_PERFORMED, ID));
```

The parameters for the **ActionEvent** constructor are shown below.

```
public ActionEvent(Object source, int id, String command)

Constructs an ActionEvent object with the specified source object.

Parameters:
source - the object where the event originated
id - the type of event
command - the command string for this action event
```

So there you have the essential ingredients of program-generated events reduced to code fragments. A complete listing of the program is contained in the next section.

Program Listing for Bare-Bones Sample Program

This section contains a complete listing of the bare-bones program. Refer to previous sections for an operational description of the program.

```
/*File Event25.java Copyright 1997, R.G.Baldwin
This program is designed to be compiled and run under JDK 1.1

This is a bare-bones program that illustrates Program Generated
Events under JDK 1.1.

The output from the program is:

Copyright 1997, R.G.Baldwin
actionPerformed() method invoked on NonVisualObject

The program was tested using JDK 1.1 running under Win95.
*/
//=========================================================================
import java.awt.*;
import java.awt.event.*;
//=========================================================================

public class Event25 {
  public static void main(String[] args){
    new Event25();//instantiate an object of this type
  }//end main
//---------------------------------------------------------------------------
  public Event25(){//constructor
    System.out.println("Copyright 1997, R.G.Baldwin");
    NonVisual nonVisualObject = new NonVisual("NonVisualObject");
    nonVisualObject.makeActionListenerList(new AnActionListenerClass());
    nonVisualObject.generateActionEvent();
  }//end constructor
}//end class Event25
//=========================================================================

//Class to respond to action events
class AnActionListenerClass implements ActionListener{
  public void actionPerformed(ActionEvent e){
    System.out.println(
      "actionPerformed() method invoked on " + e.getActionCommand());
  }//end actionPerformed
}//end class AnActionListenerClass
//=========================================================================

//Class to create object capable of generating Action events.
//--------------------------------------------------------------------------
class NonVisual extends Component {
  String ID; // The ID of the object
  ActionListener actionListener;//list of registered action listener objects
  //------------------------------------------------------------------------
  public NonVisual(String ID) {//Constructs a NonVisual object
      this.ID = ID;
  }//end constructor
  //------------------------------------------------------------------------
  public void makeActionListenerList(ActionListener listener) {
    actionListener = AWTEventMulticaster.add(actionListener, listener);
  }//end makeActionListenerList()
  //------------------------------------------------------------------------
  public void generateActionEvent() {
    actionListener.actionPerformed(
        new ActionEvent(this, ActionEvent.ACTION_PERFORMED, ID));
  }//end generateActionEvent
}//end class NonVisual
```

A More Substantive Sample Program

The next program is more substantial and is designed to illustrate the ability of the **AWTEventMulticaster** class to dispatch events to more than one **Listener** object in a list of registered **Listener** objects.

This program is designed to be compiled and run under JDK 1.1 and illustrates Program Generated Events under JDK 1.1.

A non-visual object class (named **NonVisual**) is defined. Objects of this class are capable of generating **Action** events. The mechanism for causing an object of this type to generate an **Action** event is to invoke the **generateActionEvent()** method on an object of the class.

Two objects of the **NonVisual** class are instantiated. Also, two different **ActionListener** classes are defined.

An **ActionListener** object of one of these classes is instantiated and registered to listen for **Action** events on both of the objects of the **NonVisual** class.

In addition, an object of the other **ActionListener** class is instantiated and registered to listen for **Action** events on only one of the NonVisual objects.

Thus, one **NonVisual** object is registered with only one **ActionListener** object. The other **NonVisual** object is registered with two different **ActionListener** objects.

The registration of two different **ActionListener** objects on a single **NonVisual** object requires the **AWTEventMulticaster** class to dispatch **Action** events to two different **Listener** objects and illustrates the central feature of the **AWTEventMulticaster** class.

After the **NonVisual** objects are instantiated and the **ActionListener** objects are registered on the **NonVisual** objects, the **generateActionEvent()** method is invoked on each of the **NonVisual** objects. This causes **Action** events to be generated and to be trapped and processed by the respective **ActionListener** objects.

Numerous statements are included to explain what is happening along the way. All of the output from the program goes to the standard output device.

The output from the program for one particular run was as shown below, Some hard return characters were manually inserted to force the text to fit on the printed page.

```
Copyright 1997, R.G.Baldwin
Instantiate two NonVisual objects with the ability to generate
     Action events.
Name of first NonVisual object: NonVisualObjA
Name of second NonVisual object: NonVisualObjB

Register ActionListener objects on the NonVisual objects

addActionListener() method invoked
NonVisualObj A:   Listener to add is: FirstActionListener@1cc728
Invoke AWTEventMulticaster.add() to get reference to ActionListener
NonVisualObj A:   Ref to ActionListener is: FirstActionListener@1cc728


addActionListener() method invoked
NonVisualObj B:   Listener to add is: FirstActionListener@1cc761
Invoke AWTEventMulticaster.add() to get reference to ActionListener
NonVisualObj B:   Ref to ActionListener is: FirstActionListener@1cc761


addActionListener() method invoked
NonVisualObj B:   Listener to add is: SecondActionListener@1cc783
Invoke AWTEventMulticaster.add() to get reference to ActionListener
NonVisualObj B:
         Ref to ActionListener is: java.awt.AWTEventMulticaster@1cc799
```

```
Invoke generateActionEvent() method on the object named NonVisualObjA
which has only one registered ActionListener object.
In generateActionEvent() method, dispatching ACTION_PERFORMED event to
FirstActionListener@1cc728 for NonVisualObj A
In actionPerformed() method of FirstActionListener object
actionPerformed() method invoked on NonVisualObj A

Invoke generateActionEvent() method on the object named NonVisualObjB
which has two registered ActionListener objects.
In generateActionEvent() method, dispatching ACTION_PERFORMED event to
java.awt.AWTEventMulticaster@1cc799 for NonVisualObj B
In actionPerformed() method of FirstActionListener object
actionPerformed() method invoked on NonVisualObj B
In actionPerformed() method of SecondActionListener object
actionPerformed() method invoked on NonVisualObj B
```

The program was tested using JDK 1.1 running under Win95.

Interesting Code Fragments

Much of the code in this program replicates code that you have seen in previous programs. In addition, much of the code consists of **System.out.println()** statements that are there to explain what is happening when you view the output. For the most part, we will skip code that you have seen before and concentrate only on the code that is germane to the objective of this program -- *illustration of program-generated events with particular emphasis on the **AWTEventMulticaster** class*.

The first interesting code fragment instantiates two objects of the **NonVisual** class with labels and assigns names to the objects. When you assign names in this manner, you should make certain that you assign a *unique* name to each object.

```
    NonVisual aNonVisual = new NonVisual("NonVisualObj A");
    aNonVisual.setName("NonVisualObjA");

    NonVisual bNonVisual = new NonVisual("NonVisualObj B");
    bNonVisual.setName("NonVisualObjB");
```

The next interesting code fragment registers **ActionListener** objects on the two **NonVisual** objects as described earlier. Note that unlike the bare-bones program, this program uses the conventional name of **addActionListener()** for the method that generates the list of registered **ActionListener** objects.

```
    aNonVisual.addActionListener(new FirstActionListener());
    bNonVisual.addActionListener(new FirstActionListener());
    bNonVisual.addActionListener(new SecondActionListener());
```

The next interesting code fragment causes each of the **NonVisual** objects to generate an **Action** event. Note that some uninteresting code was deleted between these two statements shown below.

```
    aNonVisual.generateActionEvent();
    ...

    bNonVisual.generateActionEvent();
```

This is followed in the program by two relatively-standard **ActionListener** class definitions which won't be repeated here.

The next interesting code fragment is the beginning of the **NonVisual** class that *extends* the **Component** class and the declaration of two instance variables of the class. (Note that some comments were removed from the code fragment.)

```
class NonVisual extends Component {
  ...
  String ID; // The ID of the object
  ActionListener actionListener;
```

The first instance variable in the above code fragment is a reference to the identification of the object which is passed in as a parameter when the object is instantiated. In our program, this is a **String** object provided as a parameter to the constructor when the **NonVisual** object is instantiated.

The constructor consists of a single assignment statement which assigns its incoming parameter to this instance variable, so we won't show it here.

The next instance variable with the identifier of **actionListener** is vital to this program.

Once **ActionListener** objects are registered on this **NonVisual** object, this instance variable will either

- contain a reference to a single **ActionListener** object (if there is only one **ActionListener** object), or
- contain a reference to an object of type **AWTEventMulticaster** (if there is more than one **ActionListener** object).

In the second case, when we later invoke the **actionPerformed()** method on the instance variable named **actionListener**, we will in fact be invoking that method on an object of type **AWTEventMulticaster** which will, in turn, invoke the method on all of the objects of type **ActionListener** contained in the list of **ActionListener** objects registered on the **NonVisual** object. As mentioned earlier, this is the central feature of the **AWTEventMulticaster** class.

The next interesting code fragment is the code which constructs the list of objects registered as **Listener** objects on a specific **NonVisual** object.

New objects are added to the list by calling the *static* **add()** method of Class **java.awt.AWTEventMulticaster** and passing to it the instance variable which references the list along with the new **Listener** object to be added to the list.

When the first object is added to the list, a reference to the **Listener** object itself is returned. Hence, in the case of a list containing a single **Listener** object, the reference to the list is simply a reference to the **Listener** object.

When additional **Listener** objects are added to the list, a reference to an object of type **java.awt.AWTEventMulticaster** is returned by the **add()** method of class **AWTEventMulticaster**. According to the JDK 1.1 documentation on class **AWTEventMulticaster**,

```
"This class will manage the structure of a
chain of event listeners and dispatch events
to those listeners."
```

When the **actionPerformed()** method is later invoked on the reference to the list, the method will be invoked either on a single object of type **ActionListener** or will be invoked on an object of type **AWTEventMulticaster**. In the second case, the **actionPerformed()** method of class **AWTEventMulticaster** assumes responsibility for invoking the **actionPerformed()** method on all the objects in the list.

With that long introduction, the interesting code fragment is not too complex:

```
actionListener = AWTEventMulticaster.add(actionListener, listener);
```

Of particular interest to us is the *output* produced by the program when two different **ActionListener** objects are registered on a single **NonVisual** object. In this case, pay particular attention to the identification (highlighted in boldface) of the **Listener** object passed in as a parameter to the **addActionListener()** method and the identification of the object ultimately referenced by the instance variable which references the list.

When the first **Listener** object was added to the list, the reference was set to the object itself. When the second Listener object was added to the list, the reference was set to **java.awt.AWTEventMulticaster@1cc799.**

A portion of the output from the program illustrating this behavior is shown below (with some hard returns manually inserted).

```
addActionListener() method invoked
NonVisualObj B:   Listener to add is: FirstActionListener@1cc761
Invoke AWTEventMulticaster.add() to get reference to ActionListener
NonVisualObj B:   Ref to ActionListener is: FirstActionListener@1cc761


addActionListener() method invoked
NonVisualObj B:   Listener to add is: SecondActionListener@1cc783
Invoke AWTEventMulticaster.add() to get reference to ActionListener
NonVisualObj B:
         Ref to ActionListener is: java.awt.AWTEventMulticaster@1cc799
```

The next interesting code fragment occurs in a method named **generateActionEvent()**. The purpose of this method is to instantiate an **ActionEvent** object and to invoke the **actionPerformed()** method on the reference to the list of **ActionListener** objects, passing the **ActionEvent** object as a parameter.

```
actionListener.actionPerformed(new ActionEvent(
      this, ActionEvent.ACTION_PERFORMED, ID));
```

As mentioned earlier, a lot of output statements were included in this program so that the output would contain an explanation of what is happening as the program executes. This section has highlighted only the code necessary to implement program-generated events with multicasting. A complete listing of the program along with the output produced by the program is contained in the next section.

Program Listing

This section contains a complete listing of the program with additional comments. The comments include the output produced by the program. See the previous sections for an operational description of the program.

```
/*File Event24.java Copyright 1997, R.G.Baldwin
This program is designed to be compiled and run under JDK 1.1

This program illustrates Program Generated Events under JDK 1.1.

The output from the program for one particular run was:

Copyright 1997, R.G.Baldwin
Instantiate two NonVisual objects with the ability to generate
     Action events.
Name of first NonVisual object: NonVisualObjA
Name of second NonVisual object: NonVisualObjB

Register ActionListener objects on the NonVisual objects

addActionListener() method invoked
```

```
NonVisualObj A:   Listener to add is: FirstActionListener@1cc728
Invoke AWTEventMulticaster.add() to get reference to ActionListener
NonVisualObj A:   Ref to ActionListener is: FirstActionListener@1cc728


addActionListener() method invoked
NonVisualObj B:   Listener to add is: FirstActionListener@1cc761
Invoke AWTEventMulticaster.add() to get reference to ActionListener
NonVisualObj B:   Ref to ActionListener is: FirstActionListener@1cc761


addActionListener() method invoked
NonVisualObj B:   Listener to add is: SecondActionListener@1cc783
Invoke AWTEventMulticaster.add() to get reference to ActionListener
NonVisualObj B:
     Ref to ActionListener is: java.awt.AWTEventMulticaster@1cc799

Invoke generateActionEvent() method on the object named NonVisualObjA
which has only one registered ActionListener object.
In generateActionEvent() method, dispatching ACTION_PERFORMED event to
FirstActionListener@1cc728 for NonVisualObj A
In actionPerformed() method of FirstActionListener object
actionPerformed() method invoked on NonVisualObj A

Invoke generateActionEvent() method on the object named NonVisualObjB
which has two registered ActionListener objects.
In generateActionEvent() method, dispatching ACTION_PERFORMED event to
java.awt.AWTEventMulticaster@1cc799 for NonVisualObj B
In actionPerformed() method of FirstActionListener object
actionPerformed() method invoked on NonVisualObj B
In actionPerformed() method of SecondActionListener object
actionPerformed() method invoked on NonVisualObj B


The program was tested using JDK 1.1 running under Win95.
*/
//=========================================================================
import java.awt.*;
import java.awt.event.*;
//=========================================================================

public class Event24 {
  public static void main(String[] args){
    new Event24();//instantiate an object of this type
  }//end main
//---------------------------------------------------------------------------
  public Event24(){//constructor
    System.out.println("Copyright 1997, R.G.Baldwin");
    System.out.println("Instantiate two NonVisual objects with the "
      + "ability to generate Action events.");
    NonVisual aNonVisual = new NonVisual("NonVisualObj A");
    aNonVisual.setName("NonVisualObjA");
    NonVisual bNonVisual = new NonVisual("NonVisualObj B");
    bNonVisual.setName("NonVisualObjB");

    System.out.println("Name of first NonVisual object: " +
         aNonVisual.getName());
    System.out.println("Name of second NonVisual object: " +
         bNonVisual.getName());

    //Register cross-linked ActionListener
    // objects on the NonVisual objects.
    // One NonVisual object is registered on a
    // single ActionListener object.
    // The other NonVisual object is registered on
    // two different ActionListener
    // objects, one of which is of the same class
    // as the object registered on
    // the first NonVisual object.
    System.out.println(
      " ActionListener objects on the NonVisual objects");
    aNonVisual.addActionListener(new FirstActionListener());
    bNonVisual.addActionListener(new FirstActionListener());
    bNonVisual.addActionListener(new SecondActionListener());

    //Now cause each of the NonVisual objects to generate an Action event.
    System.out.println(
      "Invoke generateActionEvent() method on the object named "
      + aNonVisual.getName());
    System.out.println(
      "which has only one registered ActionListener object.");
    aNonVisual.generateActionEvent();//do it
    System.out.println();//blank line
    System.out.println(
      "Invoke generateActionEvent() method on the object named "
      + bNonVisual.getName());
    System.out.println(
      "which has two registered ActionListener objects.");
    bNonVisual.generateActionEvent();//do it

  }//end constructor
}//end class Event24
//===========================================================================
//The following two classes are standard ActionListener classes. Objects
// of these classes simply trap Action events and display some information
// about them.

//First class to respond to action events
class FirstActionListener implements ActionListener{
  public void actionPerformed(ActionEvent e){
    System.out.println(
      "In actionPerformed() method of FirstActionListener object");
    System.out.println(
      "actionPerformed() method invoked on " + e.getActionCommand());
  }//end actionPerformed
}//end class FirstActionListener
//===========================================================================
//Second class to respond to action events
class SecondActionListener implements ActionListener{
  public void actionPerformed(ActionEvent e){
    System.out.println(
      "In actionPerformed() method of SecondActionListener object");
    System.out.println(
      "actionPerformed() method invoked on " + e.getActionCommand());
  }//end actionPerformed
}//end class FirstActionListener


//////////////////////////////////////////////////////////////////////////
//  The following class produces a NonVisual object capable of generating
//  Action events.
//=======================================================================
class NonVisual extends Component {
  //The state of a NonVisual object at any time is defined by the following
  // instance variables.
  String ID; // The ID of the object
  ActionListener actionListener;//Refers to a list of ActionListener
                                // objects to be notified when an Action
                                // event occurs.  (See addActionListener
                                // description below).

  //---------------------------------------------------------------------
  public NonVisual(String ID) {//Constructs a NonVisual object
      this.ID = ID;
  }//end constructor

  //---------------------------------------------------------------------
  //The behavior of a NonVisual object is defined by the following
  // instance methods.
  //---------------------------------------------------------------------

  /*
  The following method adds ActionListener objects passed in as parameters
   to the list of ActionListener objects designated to be notified of
   action events from a NonVisual object.

  Notification takes place in a different method by invoking the
   actionPerformed() method of each of the ActionListener objects on
   the list.

  New objects are added to the list by calling the static add() method
   of Class java.awt.AWTEventMulticaster and passing to it the instance
   variable which references the list along with the new listener object
   to be added.

  For the first listener object added to the list, a reference to the
   Listener object itself is returned.  Hence, in that case the reference
   to the list is simply a reference to the Listener object.

  When additional listener objects are added to the list, a reference
   to an object of type java.awt.AWTEventMulticaster is returned by the
   add() method of Class AWTEventMulticaster.

  According to the JDK 1.1 documentation on Class AWTEventMulticaster,
   "This class will manage the structure of a chain of event
   listeners and dispatch events to those listeners."

  When the actionPerformed() method is later invoked on the reference to
```

```
   the list, either the actionPerformed() method is invoked on a single
   object, or the AWTEventMulticaster object assumes responsibility for
   invoking the actionPerformed() method on all of the Listener objects
   that it is maintaining in its list of Listener objects.
  */
  public void addActionListener(ActionListener listener) {
    System.out.println();//blank line
    System.out.println("addActionListener() method invoked");
    System.out.println(ID + ":   Listener to add is: " + listener);
    System.out.println("Invoke AWTEventMulticaster.add() to get "
      + "reference to ActionListener");
    actionListener = AWTEventMulticaster.add(actionListener, listener);
    System.out.println(ID + ":   Ref to ActionListener is: " +
      actionListener);
    System.out.println();//blank line
  }//end addActionListener()
  //-----------------------------------------------------------------------
  //The  purpose of this method is to invoke the actionPerformed() method
  // on all the Listener objects that are contained in a list of Listener
  // objects that are registered to listen for Action events being
  // generated by this NonVisual object.  This is accomplished by invoking
  // the actionPerformed() method on the reference to the list.  When this
  // is done, an ActionEvent object is instantiated and passed
  //as a parameter.
  public void generateActionEvent() {
    if(actionListener != null) {
      //confirm that an ActionListener is registered
      System.out.println("In generateActionEvent() method, dispatching "
        + "ACTION_PERFORMED event to ");
      System.out.println(actionListener + " for " + ID);
      actionListener.actionPerformed(new ActionEvent(
        this, ActionEvent.ACTION_PERFORMED, ID));
    }//end if on actionListener
  }//end paint
}//end class NonVisual
//=============================================================
```

Review

Q - Without viewing the solution that follows, write a Java application that meets the specifications provided in the comments to the following application.

A - See solution below:

```
/*File SampProg129.java Copyright 1997, R.G.Baldwin
This program illustrates Program Generated Events
under JDK 1.1.

A non-visual object class (named NonVisual) is defined.
Objects of this class are capable of generating Action
events.
```

```
One object of the NonVisual class is instantiated.

One ActionListener class is defined.  An object of this
class is instantiated and registered to listen for
Action events on the NonVisual object mentioned above.

The user is prompted to enter some characters at
the keyboard.  Whenever the user enters the character 'x'
the generateActionEvent() method is invoked on the
NonVisual object causing an Action event to occur and to
be trapped and processed by the ActionListener object.
This causes the following message to be displayed on the
screen:

actionPerformed() method invoked on NonVisualObject

The program was tested using JDK 1.1.3 running under Win95.
*/
//===========================================================
import java.awt.*;
import java.awt.event.*;
import java.io.*;
//===========================================================

public class SampProg129 {
  NonVisual nonVisualObject; //reference to a NonVisual obj
  //---------------------------------------------------------

  public static void main(String[] args){
    //instantiate an object of this type
    SampProg129 thisObj = new SampProg129();
    int data = 0;

    System.out.println("Enter some characters");
    System.out.println(
               "Enter an 'x' to generate an Action Event");
    System.out.println("Enter Ctrl-z to terminate");

    try{
      while((data = System.in.read()) != -1){
        if( (char)data == 'x')
          //When the user enters an 'x', cause an Action
          // Event to be generated on the NonVisual object
          // referred to by the instance variable of this
          // class named nonVisualObject.
          thisObj.nonVisualObject.generateActionEvent();
      }//end while loop
    }catch(IOException e){}

  }//end main
//-----------------------------------------------------------
  public SampProg129(){//constructor
    System.out.println("Copyright 1997, R.G.Baldwin");
    nonVisualObject = new NonVisual("NonVisualObject");
    nonVisualObject.makeActionListenerList(
                              new AnActionListenerClass());
  }//end constructor
}//end class SampProg129
//===========================================================

//Class to respond to action events
class AnActionListenerClass implements ActionListener{
  public void actionPerformed(ActionEvent e){
    System.out.println(
      "actionPerformed() method invoked on " +
                                     e.getActionCommand());
  }//end actionPerformed
}//end class AnActionListenerClass
//===========================================================

//Class to create object capable of generating
// Action events.
//----------------------------------------------------------
class NonVisual extends Component {
  String ID; // The ID of the object
  //list of registered action listener objects
  ActionListener actionListener;
  //---------------------------------------------------------

  public NonVisual(String ID) {//Construct a NonVisual obj
      this.ID = ID;
  }//end constructor
  //---------------------------------------------------------

  //Method to construct a list of registered listeners
  public void makeActionListenerList(
                                 ActionListener listener) {
    actionListener =
         AWTEventMulticaster.add(actionListener, listener);
  }//end makeActionListenerList()
  //---------------------------------------------------------

  //Method to dispatch an action event to the registered
  // listeners.
  public void generateActionEvent() {
    actionListener.actionPerformed(
        new ActionEvent(
                  this, ActionEvent.ACTION_PERFORMED, ID));
  }//end generateActionEvent
}//end class NonVisual
//===========================================================
```

-end-