GUI EVENTS AND LISTENERS

The structure of containers and components sets up the physical appearance of a GUI, but it doesn't say anything about how the GUI **behaves**. That is, what can the user do to the GUI and how will it respond? GUIs are largely event-driven; that is, the program waits for events that are generated by the user's actions (or by some other cause). When an event occurs, the program responds by executing an event-handling method. In order to program the behavior of a GUI, you have to write event-handling methods to respond to the events that you are interested in.

The most common technique for handling events in Java is to use event listeners. A listener is an object that includes one or more event-handling methods. When an event is detected by another object, such as a button or menu, the listener object is notified and it responds by running the appropriate event-handling method. An event is detected or generated by an object. Another object, the listener, has the responsibility of responding to the event. The event itself is actually represented by a third object, which carries information about the type of event, when it occurred, and so on. This division of responsibilities makes it easier to organize large programs.

As an example, consider the OK button in the sample program. When the user clicks the button, an event is generated. This event is represented by an object belonging to the class *ActionEvent*. The event that is generated is associated with the button; we say that the button is the source of the event. The listener object in this case is an object belonging to the class *ButtonHandler*, which is defined as a nested class inside*HelloWorldGUI2*:

```
private static class ButtonHandler implements ActionListener {
   public void actionPerformed(ActionEvent e) {
      System.exit(0);
```

```
    }
}
```

This class implements the *ActionListener* interface -- a requirement for listener objects that handle events from buttons. (Interfaces were introduced in Subsection 5.7.1.) The event-handling method is named `actionPerformed`, as specified by the *ActionListener* interface. This method contains the code that is executed when the user clicks the button; in this case, the code is a call to `System.exit()`, which will terminate the program.

There is one more ingredient that is necessary to get the event from the button to the listener object: The listener object must register itself with the button as an event listener. This is done with the statement:

```
okButton.addActionListener(listener);
```

This statement tells `okButton` that when the user clicks the button, the ActionEvent that is generated should be sent to `listener`. Without this statement, the button has no way of knowing that some other object would like to listen for events from the button.

This example shows a general technique for programming the behavior of a GUI: Write classes that include event-handling methods. Create objects that belong to these classes and register them as listeners with the objects that will actually detect or generate the events. When an event occurs, the listener is notified, and the code that you wrote in one of its event-handling methods is executed. At first, this might seem like a very roundabout and complicated way to get things done, but as you gain experience with it, you will find that it is very flexible and that it goes together very well with object oriented programming.

Source : http://math.hws.edu/javanotes/c6/s1.html