GUI Event Driven Programming

When programming for a Graphics User Interface (GUI), the designer must take into account the wide variety of possible interactions with the user. In old style programming, the user is limited to providing a single stream of character input from a keyboard and is limited to viewing a single stream of character output on the terminal. The graphical interface allows a user many more possible actions. For example, the user might press a graphic button, or type characters into a text field, or move a scrollbar. The GUI program must respond to all of these action events in a timely fashion.

The best way to handle all the possible user interactions is to use interrupts. In this way the CPU does not waist time waiting for user action events to occur; it simply responds to the events and resumes normal processing. Ordinarily, programming languages do not give direct access to interrupts. The Java API allows the programmer to make classes of objects, called **listeners**, that provide indirect access to interrupts caused by the GUI. The Java API has interfaces that listener classes must implement. The methods in the interfaces get called when the specified events occur.

As an example of listeners handling events, let us look at a Java application that makes a closeable window on the screen.

■
Dr. Louis G. Johnson
School of Electrical and Computer Engineering
Oklahoma State University

```
import.java.event.*;
import.javax.swing.*;

class CloseableFrame extends JFrame {
   public CloseableFrame() {
      setTitle("My Closeable Frame");
      setSize(300, 200);
      //cause window events to be sent to window listener object
      addWindowListener(new MyWindowListener());
   }
}

class MyWindowListener implements WindowListener {
   //Do nothing methods required by interface
   public void windowActivated(WindowEvent e) {}
   public void windowDeactivated(WindowEvent e) {}
   public void windowIconified(WindowEvent e) {}
   public void windowDeiconified(WindowEvent e) {}
   public void windowOpened(WindowEvent e) {}
   public void windowClosed(WindowEvent e) {}
   //override windowClosing method to exit program
   public void windowClosing(WindowEvent e) {
```


■
Dr. Louis G. Johnson
School of Electrical and Computer Engineering
Oklahoma State University

```
        System.exit(0);     //normal exit
    }
}

class Main {
    public static void main(String[] args) {
        CloseableFrame f = new CloseableFrame();
        f.show();   //makes the frame visible
    }
}
```

1. Objects in the **CloseableFrame** class cause a window to appear on the users screen. Only an application (not applet) can do this and an application can make as many windows as desired by making multiple **CloseableFrame** objects.
2. A **MyWindowListener** object is registered to listen to the window with the **addWindowListener** method. When an event happens to the window, the Java runtime environment automatically calls the appropriate method in the **WindowListener** interface.
3. The **WindowListener** interface is implemented by objects in the **MyWindowListener** interface so that the objects can respond to events in the window in which they are registered. There are seven methods

■
Dr. Louis G. Johnson
School of Electrical and Computer Engineering
Oklahoma State University

required by the **WindowListener** interface (see the Java API Documentation for a detailed description of when each method is called), of which we are only interested in the **windowClosing** method. Empty methods which do nothing are provided for the other 6 methods.

Most other event interfaces are not as complicated as the WindowListener interface. The Mimic class, below is a simple example of event driven programming that takes user input from a **TextField** and echoes it into a **Label**.

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

public class Mimic extends JFrame {
   MimicGUI gui = new MimicGUI();

   public Mimic() {
      setTitle("Mimic");
      setSize(250, 100);
      //fast way to setup closeable window listener object
      addWindowListener(new WindowAdapter() {
```


■
Dr. Louis G. Johnson
School of Electrical and Computer Engineering
Oklahoma State University

```
            public void windowClosing(WindowEvent e) {
                System.exit(0);
            }
        } );
        getContentPane().add(gui);
        setVisible(true);
    }

    public static void main(String[] args) {
        Mimic mimic = new Mimic();
    }
}

class MimicGUI extends JPanel {
    private JLabel label = new JLabel(“Echo appears here”);
    private JTextField quote = new JTextField(20);
    private MimicListener listener = new MimicListener(this);

    public MimicGUI() {
        //add quote and label to window
        add(quote);
        add(label);
```


■
Dr. Louis G. Johnson
School of Electrical and Computer Engineering
Oklahoma State University

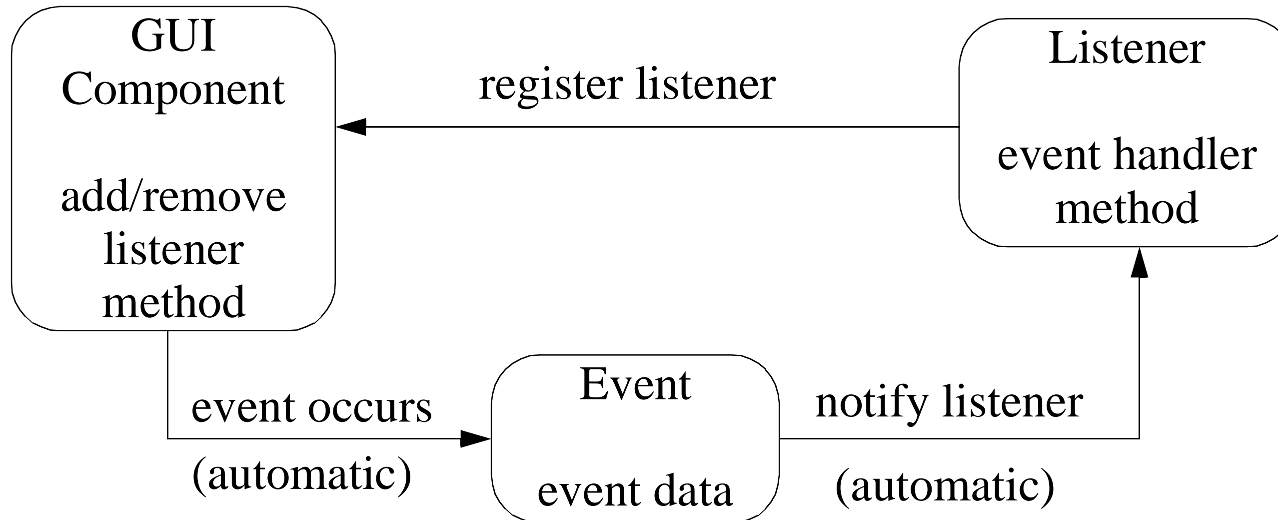
```
        //register listener with quote object
        quote.addActionListener(listener);
    }

    public void updateLabel() {
        label.setText(quote.getText());
    }
}

class MimicListener implements ActionListener {
    private MimicGUI gui;

    public MimicListener(MimicGUI guiref) {
        gui = guiref;
    }

    //method required by action listener interface
    public void actionPerformed(ActionEvent e) {
        gui.updateLabel();
    }
}
```


Dr. Louis G. Johnson
School of Electrical and Computer Engineering
Oklahoma State University

1. The listener is registered with the **JTextField quote** in **MimicGUI()** by using the **quote.addActionListener(listener)** method. When the listener is registered, it is added to an internal list which keeps track of which listeners to notify when events occur.
2. When a text field event occurs (user types into text field and hits enter), the **listener.actionPerformed(ActionEvent event)** method gets called as defined in class **MimicListener**. Note that data about the event are in the event object passed as a parameter of the method.
3. The code in the listener method handles the event by calling the **gui.updateLabel()** method (which echoes the text field contents into the label displayed by the application window).

■
Dr. Louis G. Johnson
School of Electrical and Computer Engineering
Oklahoma State University

In general, the objects required to deal with events are:



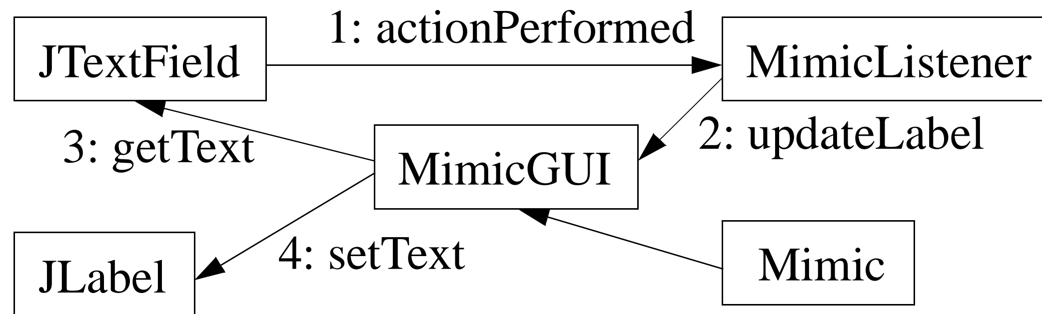
The GUI components and the Events are objects from classes defined in the Java API. The Listeners are objects from user defined classes that implement the appropriate listener interfaces defined in the Java API. It is not necessary for the user to create Event objects; these are created automatically (by the mouse/keyboard interrupt service routines) and then passed as arguments to the listener's event handling method.

The above pattern is repeated for all GUI components that can be used for user input. The GUI component classes are listed in the Java API documentation for the javax.swing package. The component class names

Dr. Louis G. Johnson
School of Electrical and Computer Engineering
Oklahoma State University

all start with J and end with names that are similar to GUI components defined in the old java.awt package. The new swing components are superior to the old components. Refer to the documentation for javax.swing.JComponent for a discussion of the differences.

Mimic Example. Now that we have the basic idea of event programming, let us go back and examine the Mimic program in more detail. Before writing code, it often helps to make diagrams which show the user defined classes and the methods through which they interact.



Since the JTextField and JLabel classes are from the Java API, the only code we need to write is for the other three classes. Arrow 1 in the figure corresponds to the event occurrence diagram we drew earlier. Note that the ActionEvent class is not in this diagram since the user does not create objects from this class; the system does so automatically.

Dr. Louis G. Johnson
School of Electrical and Computer Engineering
Oklahoma State University

The Mimic class describes the primary object which creates the application window. Just as in the earlier example of graphics animation, the drawing surface of the window is described by another class, MimicGUI, which will describe all of the GUI components used to make the window display. Putting the description of the GUI components in a separate class is not necessary, but it may be a good idea since the physical organization of the GUI display can be quite complicated. This separates the description of the physical organization of the display from other code.

Another thing that we can do is to take advantage of the greater capability of the swing components. The **ActionEvent** for the **JTextField** class only occurs when the user finishes typing and presses enter. This means that the label does not change as the user types; it only changes when the user finishes. The new **JTextField** class also has a **DocumentEvent** which is triggered immediately upon any change to the text. We can cause the label to change continuously with the text entry by using this event instead. Here are the changes necessary to use the **DocumentEvent** instead of the **ActionEvent**.

1. Register the listener in MimicGUI() as

    quote.getDocument().addDocumentListener(listener);

2. The MimicListener class must implement the new interface

■
Dr. Louis G. Johnson
School of Electrical and Computer Engineering
Oklahoma State University

```
class MimicListener implements DocumentListener {
```

3. The actionPerformed method in MimicListener must be replaced by three methods required by the DocumentListener interface.

```
    public void insertUpdate(DocumentEvent e) {
        gui.updateLabel();
    }

    public void removeUpdate(DocumentEvent e) {
        gui.updateLabel();
    }

    public void changedUpdate(DocumentEvent e) {
    }
```

The first two methods are called when characters are added or deleted. The third method is called when the text format is changed which cannot occur for objects in the JTextField class which is why a null method was provided.

■
Dr. Louis G. Johnson
School of Electrical and Computer Engineering
Oklahoma State University