

Events

Our GUI interfaces look nice but as yet do not do anything. It is time to add some functionality to our GUI's. When we click on a component such as a button, we expect some sort of action to take place. To accomplish this we need to perform several steps. Lets look at the steps involved in getting a button, when clicked, to show a `showMessageDialog()` message.

If you click a button in a program and an interesting thing happens. The button sends out a signal which in Java we call an event. There are many different kinds of events. Think of all the different ways in which a user interacts with a computer.

Clicking a button, rolling the mouse over an icon, typing a key.

Each one of these actions and many other like them cause a signal to be sent out that states that that particular events has occurred.

Button Events

In the case of a button, the most common event associated with it is an `ActionEvent` caused by the user clicking on it. When the user clicks on the button an object of type `ActionEvent` is created. We refer to the buttons as the sources of the event and the event as an object of type `ActionEvent`.

All of the buttons that we created in past lesson were actually busy broadcasting objects of type `ActionEvent` every time we clicked them. The trouble was, no one was listening.



ActionListener Class

In order to hear and respond to these event objects there needs to be a special class written. Generally speaking, any class whose purpose is to listen for event objects is called an event listener.

In the case of a class that will respond to our action event the signature of the class will need to look like this.

<classname> *implements ActionListener*

For the sake of example lets make the class name of this ActionListener class ***MyButtonListener***

MyButtonListener implements ActionListener

ActionListener Interface

The words ***implements ActionListener*** mean that this class is going to respond to an ActionEvent.

It also means that this class **MUST** contain the following method.

```
public void actionPerformed(ActionEvent e)
{
    // this code dictates the actions taken when the button is pressed.
    //Assuming you wanted a message to pop up when the button was
    //pressed you might include code such as,
        System.out.println("Hello World");
}
```

If you try and write a class that ***implements ActionListener*** that does include the actionPerformed(ActionEvent e) method you will invoke the wrath of the compiler.

Argument to Parameter

Remember the argument `e` of type `ActionEvent` that was passed when the button was clicked. Looking at the parameter of the `actionPerformed(ActionEvent e)` method we now see where this argument ends up.

Connecting the Event Source to the Event Handler

We now have our event listener class that implements ActionListener and thereby contains the required actionPerformed() method.

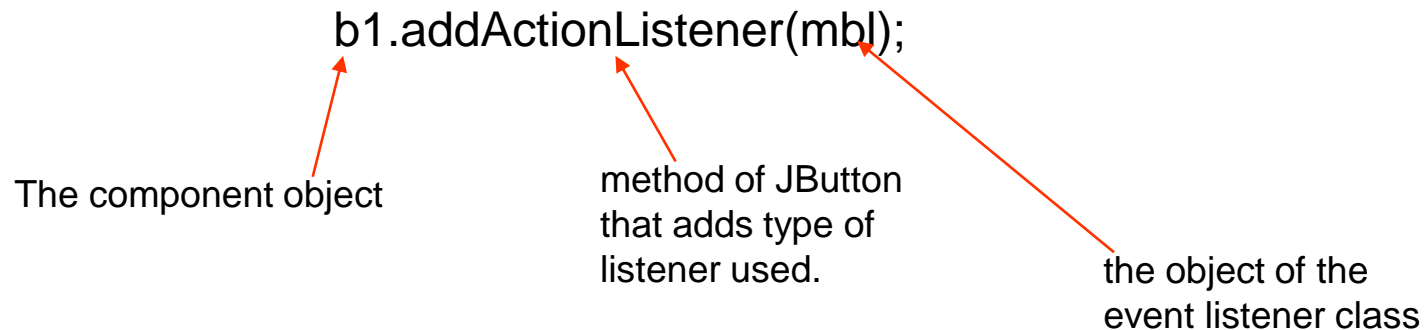
We have our event source, in this case a button which, when pressed, generates an ActionEvent. Lets assume our button has been initialized as follows:

```
JButton b1 = new JButton("Click Me");
```

What we need now is to connect them together. This requires 2 lines of code. First we instantiate an object of type MyButtonListener

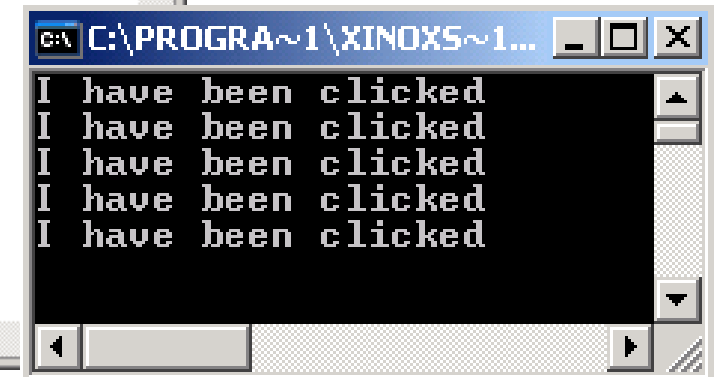
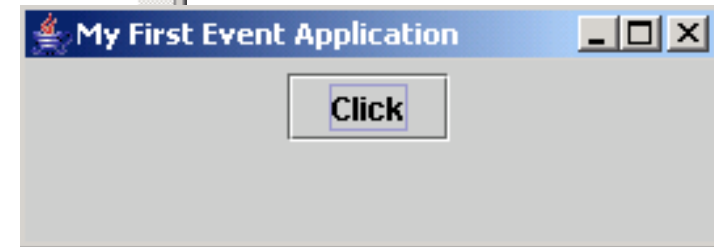
```
MyButtonListen mbl = new MyButtonListen();
```

Now the actual connection takes place using a method of the JButton class called addActionListener() as follows:



The Code

```
EventDemo.java
1 import java.awt.*;
2 import java.awt.event.*;
3 import javax.swing.*;
4
5 public class EventDemo
6 {
7     public static void main(String args[])
8     {
9         JFrame jf = new JFrame("My First Event Application");
10        Container c = jf.getContentPane();
11
12        FlowLayout f = new FlowLayout();
13        c.setLayout(f);
14
15        JButton b1 = new JButton("Click");
16        c.add(b1);
17
18        MyButtonListener mbl = new MyButtonListener();
19        b1.addActionListener(mbl);
20
21        jf.pack();
22        jf.setVisible(true);
23    }
24 }
25
26 class MyButtonListener implements ActionListener
27 {
28     public void actionPerformed(ActionEvent e)
29     {
30         System.out.println("I have been clicked");
31     }
32 }
```



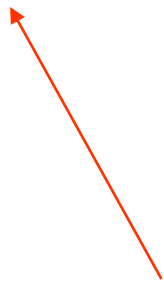
The Steps: Summarized

- Write an event class that implements ActionListener and contains the method actionPerformed()
- Within the actionPerformed method write the code that dictates what actions will take place.
- Instantiate your button
- Instantiate an object of your ActionListener class.
- using dot notation connect your button to the object reference of your ActionListener class.

Steps 1 and 2. Memorize

Write an event class that implements ActionListener and contains the method actionPerformed()

```
class MyButtonListener implements ActionListener
{
    public void actionPerformed(ActionEvent e)
    {
        System.out.println("I have been clicked");
    }
}
```



Within the actionPerformed method write the code that dictates what actions will take place.

Remaining Steps

Instantiate button and add it to container

```
JButton b1 = new JButton("Click Me");  
c.add(b1);
```

```
MyButtonListener mbl = new MyButtonListener();
```

```
b1.addActionListener(mbl);
```

add listener object to button



Instantiate object of type listener
that you created.

