Handling Mouse Events

This handout is designed to give you a quick introduction to handling events generated by mouse actions. For a more detailed explanation, we encourage you to look at the tutorial "How to Write a Mouse Listener" provided by Oracle at
`https://docs.oracle.com/javase/tutorial/uiswing/events/mouselistener.html`.

For more information on any of the classes mentioned here, please look at the official Java class documentation at
`https://docs.oracle.com/javase/8/docs/api/index.html?overview-summary.html`.

Handling mouse events in Java is a bit different from handling events generated by GUI components. In some ways it is easier – you don't have to "create" or "initialize" a mouse. It is already there. On the other hand, there are more things you can do with a mouse. Here are a few: press the mouse button, release it, move it into a window or other component, exit the window or component, click the mouse button. All of these are reflected in the `MouseListener` interface.[1]

The `MouseListener` interface is defined as follows:

```
interface MouseListener {
   void mouseClicked(MouseEvent e)
   void mouseEntered(MouseEvent e)
   void mousePressed(MouseEvent e)
   void mouseReleased(MouseEvent e)
}
```

You can see the complete specification of each of these methods in the JavaDoc implementation at
`https://docs.oracle.com/javase/8/docs/api/index.html?overview-summary.html`.

Another set of mouse actions involve moving the mouse. These are specified by a different interface, `MouseMotionListener`, which looks like:

```
interface MouseMotionListener {
   void mouseDragged(MouseEvent e)
   void mouseMoved(MouseEvent e)
}
```

Move later. If you need an interface that includes all six method, recent versions of Java provide a new interface, `MouseInputListener` that includes all six methods.

You will notice that each of these six methods takes an object of class `MouseEvent` as a parameter.[2] The only methods we will need from this class are `getX()` and `getY()`, which return the x and y-coordinates of where the mouse is on the window when the action took place (e.g., where the mouse was pressed or moved to or ...).[3]

1 Defining a listener for mouse events

In order for your program to react to mouse events, we need to designate an object to "listen" for those events and then to write code to specify what action your program should take when the event occurs.

[1]To be complete we should note that MouseListener is in library `java.awt.event`, so to access it, you will need to import `java.awt.event.MouseListener` or `java.awt.event.*`.

[2]Unfortunately the standard Java libraries include both a class `MouseEvent` and an interface `MouseEvent`. You will *always* want the class, which is from `java.awt.event`.

[3]For those of you who took CS51 at Pomona, the objectdraw library used similarly named methods, but they took `Location` parameters rather than events, so that you didn't have to do the extra work of pulling out the x and y-coordinates in your code.

There are many ways of specifying these listeners. We begin by describing one of the most common, creating an inner class.

For an inner class to be a mouse listener, it must implement whichever listener interface has the events you wish to respond to. For example if you want this object to respond to mouse presses and releases, it should implement `MouseListener`. Of course if it implements `MouseListener`, it must implement *all* of the methods of that interface, even if you aren't interested in them. In Figure 1 where we illustrate a simple program to print the location of the mouse each time the mouse is pressed, released, or clicked.

There are a couple of things to note about this program. In each of the relevant methods we use the methods `getX()` and `getY()` on the event parameter to get the location of the mouse when the event occurred. Where does the event parameter come from? It is automatically generated by the operating system when the user makes a mouse action.

Here is the sequence of what happens:

1. The user makes a mouse action.
2. The system responds by creating a mouse event that includes all relevant information, including the x and y coordinates of the mouse when the action occurred.
3. The system places the mouse event on the "event queue", which is a list of events handled by the operating system. (We'll talk about it in more detail later.)
4. When the event has moved to the front of the event queue, the "event thread" will remove it from the queue and pass it along to any mouse listeners that are associated with the component the user interacted with.
5. Finally, the listener will execute the appropriate method with that mouse event as a parameter.

Notice that you need never directly call a mouse method. The system will take care of that for you.

Be aware that Java considers a click to be a mouse press followed by a mouse release without any mouse movement in between. It does not care how long the mouse if pressed. Notice that this also means that a mouse click generates a mouse press event, followed by a mouse release event, followed by a mouse click event. Thus all three methods will be triggered by a click.

Now the only thing we have to take care of is to tell the system which object should be notified when a mouse action happens. That is, we must register an object as the listener.

In the sample program, this happens in the constructor for the `Demo` class, which is a specialized `JFrame`. The command `addMouseListener(new MouseLocator());` creates a new instance of inner class `MouseLocator` and then tells the system that it is the one to be informed if a mouse event occurs.

We don't actually have to implement this as part of the constructor as we can send the message to the component from outside. For example we could do this in the `main` method by writing

```
d.addMouseListener(new MouseLocator());
```

there.

By the way, we've been focusing our attention on mouse actions handled by interface `MouseListener`, but if we can also handle mouse motion actions (`move` or `drag`) if we create another listener class that implements `MouseMotionListener` and attach it using `addMouseMotionListener`.

2 Advanced topics

In this section we will describe a number of variations that you can use in your programs when you are responding to mouse actions. It is fine to skip this and just use the material in the above section.

```java
import java.awt.event.MouseEvent;
import java.awt.event.MouseListener;
import javax.swing.JFrame;

public class Demo extends JFrame {

   public Demo() {
      super("listener demo");
      addMouseListener(new MouseLocator());
   }
   private class MouseLocator implements MouseListener {
      @Override
      public void mousePressed(MouseEvent e) {
         System.out.println("Mouse was pressed at (" + e.getX() + ","
               + e.getY() + ").");
      }

      @Override
      public void mouseClicked(MouseEvent e) {
         System.out.println("Mouse was clicked at (" + e.getX() + ","
               + e.getY() + ").");
      }

      @Override
      public void mouseReleased(MouseEvent e) {
         System.out.println("Mouse was released at (" + e.getX() + ","
               + e.getY() + ").");
      }

      @Override
      public void mouseEntered(MouseEvent e) {
      }

      @Override
      public void mouseExited(MouseEvent e) {
      }
   }

   public static void main(String[] args) {
      Demo d = new Demo();
      d.setSize(200,200);
      d.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
      d.setVisible(true);
   }
}
```

Figure 1: Handling mouse events

2.1 Extending rather than implementing

As noted earlier, when implementing the interface MouseListener, you must include methods for all those mentioned in the interface. I.e., even if you only want a mousePressed method, you have to include the other 3 methods. One way around this is to have your listener extend MouseAdapter rather than implementing MouseListener. The class MouseAdapter includes default implementations of all of the methods in MouseListener. These default implementation do nothing. Thus if you only want to use mousePressed, then override the implementation in MouseAdapter and just ignore the other methods. Your listener will inherit the default implementations of the other methods (which do nothing, as desired). MouseMotionAdapter does the same thing for MouseMotionListener.

One word of warning! You can only do this if your class does not already extend another class, as Java only allows a class to extend one other class. For example, if you have a class that extends JFrame, then it cannot also extend MouseAdapter.

2.2 Combining MouseListener and MouseMotionListener

Some applications use methods from both MouseListener and MouseMotionListener. If you would like the same object to serve as a listener for all of these methods you can either have it implement interface MouseInputListener or extend class MouseInputAdapter.

2.3 Who can listen?

The example show here uses an inner class as a listener for mouse actions. However, we can use nearly anything as a listener. For example, class Demo2Self, which extended JFrame could listen for mouse actions on itself. To do this, declare that the class implements MouseListener, add itself as the listener, and then include the mouse methods directly in the class. The code for this is in Figure 2 .

```
import java.awt.event.MouseEvent;
import java.awt.event.MouseListener;

import javax.swing.JFrame;

public class Demo2Self extends JFrame implements MouseListener {
   public Demo2Self() {
      super("listener demo2self");
      addMouseListener(this);
   }

   @Override
   public void mousePressed(MouseEvent e) {
      System.out.println("Mouse was pressed at (" + e.getX() + "," + e.getY()
            + ").");
   }

   @Override
   public void mouseClicked(MouseEvent e) {
      System.out.println("Mouse was clicked at (" + e.getX() + "," + e.getY()
            + ").");
   }

   @Override
   public void mouseReleased(MouseEvent e) {
      System.out.println("Mouse was released at (" + e.getX() + ","
            + e.getY() + ").");
   }

   @Override
   public void mouseEntered(MouseEvent e) {
   }

   @Override
   public void mouseExited(MouseEvent e) {
   }

   public static void main(String[] args) {
      Demo d = new Demo();
      d.setSize(200, 200);
      d.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
      d.setVisible(true);
   }

}
```

Figure 2: Demo listening to itself