

Mouse and Keyboard Listeners

The good news is that mouse and keyboard input is handled in basically the same way as other listeners. We select the component that we want to handle a listener and implement the mouse or keyboard interfaces. When a mouse or keyboard event occurs, the appropriate method is invoked in the interface.

Mouse Listeners

To catch mouse events we import **java.awt.event.MouseListener** and the class we want to handle events should implement the **MouseListener** interface. This interface requires that we define the following methods:

- mouseClicked
- mouseEntered
- mousePressed
- mouseReleased
- mouseExited

Each method takes a `MouseEvent` parameter which gives us the X and Y coordinates of the mouse.

The component to handle the mouse events calls the `addMouseListener(component)` method.

Here is an example that outputs the X and Y coordinates when the mouse is clicked:

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

...
public class GUI_Deme extends JFrame implements MouseListener
{
    ...
    In constructor:
        this.addMouseListener(this);

    Event handlers:

        @Override
        public void mouseClicked(MouseEvent e) {
            System.out.println("X = " + e.getX() + " Y = " +
                               e.getY());
        }

        @Override
```

```

    public void mousePressed(MouseEvent e) {
    }

    @Override
    public void mouseReleased(MouseEvent e) {
        System.out.println("Mouse released");
    }

    @Override
    public void mouseEntered(MouseEvent e) {
    }

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

```

Upon clicking the mouse you should see the coordinates as long as some other component isn't handling the mouse click (e.g. a button). Note that the XY coordinates are relative to the upper left corner of the component. You can see this more clearly if you add a mouse listener to a JPanel or some subcomponent inside a GUI window.

Class exercise: Make a "paint" program that draws a red circle at the location of the mouse click. Circles drawn from past clicks should remain on the screen.

The MouseListener interface is geared around mouse clicks. If you want to capture mouse motion then there is a separate interface, the **MouseMotionListener**. This is also found in java.awt.event. The MouseMotionListener requires that we implement these methods:

- mouseMoved
- mouseDragged

As you can infer, the methods are invoked when the mouse is either moved or dragged. Both take a MouseEvent as an input parameter.

The following outputs the mouse coordinates as the mouse is moved:

```

import java.awt.event.MouseMotionListener;

...
public class GUI_Deme extends JFrame implements MouseMotionListener
{
    ...
    In constructor:
        this.addMouseMotionListener(this);

    Event handlers:

```

```

        @Override
        public void mouseDragged(MouseEvent e) {
        }

        @Override
        public void mouseMoved(MouseEvent e) {
            System.out.println(e.getX() + " " + e.getY());
        }
    }
}

```

Class Exercise: Modify the “paint” program from the previous exercise so a red circle is drawn as the mouse moves if the mouse button is not clicked, but a green circle is drawn if the mouse button is held down (this is considered a drag event). Do the same thing but just toggle colors if the mouse button is clicked.

Keyboard Listeners

It is a similar drill for keyboard listeners. We just have to learn what interface to implement and what methods need to be defined. There are two kinds of events: typing a character and pressing/releasing a key on the keyboard.

Typing a character is a key-typed event while pressing or releasing a key is a key-pressed or key-released event.

To respond to a keyboard event, the component must have focus.

Here is an example that prints out keys pressed as long as the JFrame window has focus:

```

import java.awt.event.KeyEvent;

import java.awt.event.KeyListener;

...

public class GUI_Deme extends JFrame implements KeyListener
{
    ...
    In constructor:
        this.addKeyListener(this);
        ...
        setVisible(true);
        toFront();           // Set focus to the frame
        requestFocus();
}

```

Event handlers:

```
@Override
public void keyTyped(KeyEvent e) {
    System.out.println("Key Typed");
    displayInfo(e);
}

@Override
public void keyPressed(KeyEvent e) {
    System.out.println("Key Pressed");
    displayInfo(e);
}

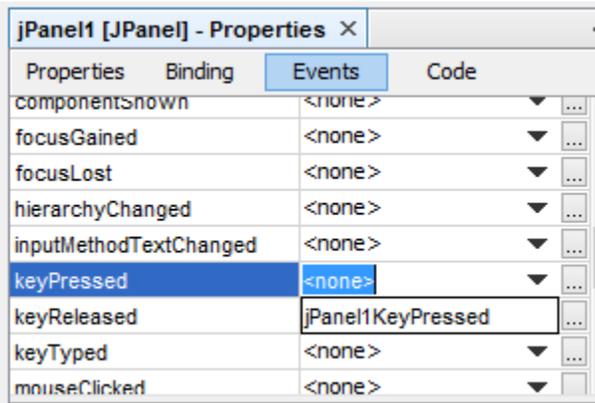
@Override
public void keyReleased(KeyEvent e) {
    System.out.println("Key Released");
    displayInfo(e);
}

private void displayInfo(KeyEvent e)
{
    int id = e.getID();
    if (id == KeyEvent.KEY_TYPED)
    {
        System.out.println(e.getKeyChar());
    }
    else
    {
        int keyCode = e.getKeyCode();
        System.out.println("Code = " + keyCode +
            " Text:" + KeyEvent.getKeyText(keyCode));
    }
    if (e.isActionKey())
        System.out.println("Action key pressed!");
        // Arrows, FN keys
}
}
```

Class exercise: Move the red circle around using the WASD keys or arrow keys.

Using an IDE

If you are using an IDE then adding these events and listeners is pretty much done for you by the GUI. Select the component of interest and then under events select what kind of event you want to handle and the IDE will create the method for you and you need to fill in the code.



Properties	Binding	Events	Code
componentsown		<none>	...
focusGained		<none>	...
focusLost		<none>	...
hierarchyChanged		<none>	...
inputMethodTextChanged		<none>	...
keyPressed		<none>	...
keyReleased		jPanel1KeyPressed	...
keyTyped		<none>	...
mouseClicked		<none>	...

```
private void jPanel1KeyPressed(java.awt.event.KeyEvent evt) {  
    // TODO add your handling code here:  
}
```