Singer, Shah, Kulczak

Event Listeners

In a GUI the program is driven by the user interactions. So, any action, like moving the mouse or pressing a key, performed by the user is registered as an event in the program. Event Listeners are used to translate user interaction into actual code. To listen for a particular event, the programmer must know the event source. A common way of implementing an event listener is by declaring an inner class that implements a particular event listener. This allows the event listener to use any Final variables within the surrounding blocks. This also makes the project less cluttered, as the inner classes are located only where they are needed.

JButtons are some of the most common users of event listeners. To do so, the programmer must declare that an instance of JButton will use a particular listener interface. It does so by calling the addActionListener(ActionListener e) method. This tells the program to use the ActionListener method, actionPerformed, to respond to button events like button click.

Example:

```
public static void main(String[] args){
. . .
        JButton button = new JButton();
        final JLabel label = new JLabel("This can be used by the inner class");

        class Test implements ActionListener{
        …
                public void actionPerformed(ActionEvent e){
                …
                }
        }
        ActionListener listener = new Test();
        Button.addActionListener(listener);
}
```

If the programmer only wants to update an action at a certain time interval, he could use the Timer class. For example, *Timer t = new Timer(interval, eventListener);*

A class that responds to mouse events implements the MouseListener interface. The MouseListener interface has five methods: mousePressed, mouseReleased, mouseClicked, mouseEntered, mouseExited.

Example:

```
Class MousePressListener implements MouseListener{
        Public void mousePressed(MouseEvent e){
                Int x = event.getX();
                Int y = event.getY();
                Component.moveTo(x,y);
        }
}
```