Keyboard Input

Public Interface KeyListener

This interface is used for Keyboard input

Documentation:

http://docs.oracle.com/javase/7/docs/api/java/awt/event/KeyListener.html

http://docs.oracle.com/javase/7/docs/api/java/awt/event/KeyEvent.html

KeyListener Methods:

keyPressed(KeyEvent e)
Invoked when a key has been pressed.

keyReleased(KeyEvent e)
Invoked when a key has been released.

keyTyped(KeyEvent e)
Invoked when a key has been typed.

Public Class KeyEvent

KeyListener Methods:

getKeyChar()
Returns the character associated with the key in this event. (if the 'a' key is pressed, the code for the 'a' character is returned)

getKeyCode()
Returns the event's keycode. (uses virtual key codes to tell what key on the keyboard was pressed, like the arrow keys)

Below is an example of an application using a KeyListener to take input from the user:

In this example, the overall goal is to have a row of several squares with labels, 1 through 8. When the user presses one of the keys from '1' to '8', the corresponding square will be highlighted. In particular, when the user presses a button, an event occurs. Then, there will be an action corresponding to that event. That action will be to change instance variables which indicate where to highlight.

First, let's take a look at our instance variables (along with the beginning of the file):

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

class StudentWorkSolution extends JComponent {

	public static int WINDOW_WIDTH = 500;
	public static int WINDOW_HEIGHT = 200;

	// Instance variables - colors we'll use.
	private Color outlineColor;
	private Color backgroundColor;
	private Color highlightColor;

	// Where we are highlighting and the # of squares.
	private int numSquares;
	private int sqSize;
	private int xCoordinate = -50;
	private int yCoordinate = -50;
```

The outline color is the original color of the squares, the background color is the color of the background and the highlight color is the color that we'll draw a square when the corresponding button on the keyboard is pressed. The variable numSquares will store the number of squares in a row and sqSize will store the length in pixels of a single square.

Here are the initialize method and draw methods. The draw method gets called over and over again in the main graphics loop:

```
	public void initialize() { }

	public void draw(Graphics g) {
		g.setColor(this.backgroundColor);
		g.fillRect(0, 0, WINDOW_WIDTH, WINDOW_HEIGHT);
		g.setColor(this.outlineColor);

		for (int i = 0; i < numSquares; i++) {

			g.drawRect(i*sqSize + sqSize, sqSize, sqSize, sqSize);

			g.drawString("" + (i+1), i*sqSize + sqSize + sqSize/2,
sqSize + sqSize/2);
		}

		g.setColor(this.highlightColor);
```

```
            g.drawRect(xCoordinate, yCoordinate, sqSize, sqSize);
      }
```

First, we just set up the background, the change the color. After that, we want to loop through and draw each square. We use the loop index to tell us where to draw the square and what number (drawString) to draw in it. Then we draw the highlighted square. If nothing is highlighted, nothing gets drawn because our default values for xCoordinate and yCoordinate are off the screen.

Here is the rest of the StudentWorkSolution class:

```
      public int squareCnt() {
            return numSquares;
      }

      public void highlightSquare(int idx) {
            if (idx >= 1 && idx <= numSquares) {
                  xCoordinate = idx*sqSize;
                  yCoordinate = sqSize;
            }
            else {
                  xCoordinate = -50;
                  yCoordinate = -50;
            }
            this.repaint();
      }

      public StudentWorkSolution(int numSq, int size) {
            this.initialize();
            outlineColor = Color.black;
            backgroundColor = Color.white;
            highlightColor = Color.red;
            numSquares = numSq;
            sqSize = size;
      }

      public void paintComponent(Graphics g) {
            this.draw(g);
      }
}
```

The highlight square method takes in the number of which square is being highlighted. If it's in range, then we update the xCoordinate and yCoordinate accordingly. If not, we set both to -50. Then we repaint the screen.

The rest of our code is a class that has our main method and our KeyboardListener. The object for our class needs both a JFrame and a StudentWorkSolution object, which is managing all of the different components on the screen. Ultimately, we create both objects and then add the StudentWorkSolution object to the JFrame we created:

```
public class keyListenerPractice {

	static JFrame frame;
	static StudentWorkSolution student;

	public static void main(String[] args) {

		frame = new JFrame("Press a digit 1 to 8!");
		frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
		frame.setSize(student.WINDOW_WIDTH, student.WINDOW_HEIGHT);

		student = new StudentWorkSolution(8, 50);
		student.addKeyListener(new KeyboardListener());
		student.setFocusable(true);
		frame.add(student);

		frame.setResizable(false);
		frame.setVisible(true);
	}

	static class KeyboardListener implements KeyListener {

		public void keyTyped(KeyEvent e) { }

		// If a key is pressed this code runs.
		public void keyPressed(KeyEvent e) {

			// We have hard-coded where to highlight when 1 is pressed.
			if (e.getKeyChar() >= '1' && e.getKeyChar() <=
(char)('0'+student.squareCnt()))
				student.highlightSquare(e.getKeyChar() - '0');
		}

		public void keyReleased(KeyEvent e) {
			student.highlightSquare(0);
		}
	}
}
```

In the KeyboardListener class, we just have to define the keyPressed method that takes in a KeyEvent. For a KeyEvent, we can access the key that was pressed. Based on that key, we can take some action. In this case, we check to see if that key is in range. If it is, we highlight the appropriate square by calling that method on the student object. When the key is released, we highlight square 0, which essentially says to highlight nothing.

Here is another example but this time utilizing the getKeyCode() method to take the arrow keys as an input:

```
// This class holds the main method
public class Swingbabyswing
{
    // Main Method
    public static void main(String[] args)
    {
        // Make new stuff to use
        // TextArea
        JTextArea textme = new JTextArea();

        // Add the KeyListener, set TextArea un-editable
        textme.addKeyListener(new ActOnKey(textme));
        textme.setEditable(false);

        // Make a scrolling pane for the text area
        JScrollPane scrolly = new JScrollPane(textme);

        // Make the window to put it all in
        JFrame myFrame = new JFrame("Swing it!");

        // Set up the window
        // Make it close when exit button is clicked
        // Set size and layout
        // Add in the components
        // Make this window visible
        myFrame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        myFrame.setSize(500, 500);
        myFrame.setLayout(new BorderLayout());
        myFrame.add(scrolly, BorderLayout.CENTER);
        myFrame.setVisible(true);
    }

}

// Class used to make custom KeyListener
class ActOnKey implements KeyListener
{
    JTextArea T;

    public ActOnKey(JTextArea T)
    {
        this.T = T;
    }

    public void keyPressed(KeyEvent e)
    {
        char c = e.getKeyChar();
        int k = e.getKeyCode();

        // KeyCharacter events
```

```
        if (c == 'w' || c == 'W')
            T.append("UP\n");

        else if (c == 'a' || c == 'A')
            T.append("LEFT\n");

        else if (c == 'd' || c == 'D')
            T.append("RIGHT\n");

        else if (c == 's' || c == 'S')
            T.append("DOWN\n");

        else if (c == ' ')
            T.append("JUMP\n");

        // KeyCode events
        if (k == KeyEvent.VK_DOWN)
            T.append("DOWN (Key)\n");

        if (k == KeyEvent.VK_UP)
            T.append("UP (Key)\n");

        if (k == KeyEvent.VK_LEFT)
            T.append("LEFT (Key)\n");

        if (k == KeyEvent.VK_RIGHT)
            T.append("RIGHT (Key)\n");
    }

    public void keyReleased(KeyEvent e)
    {

    }

    public void keyTyped(KeyEvent e)
    {

    }
}
```

In the keyPressed method we can see how to handle some other key code events, such as the arrow keys. Namely, from a key we can get an integer that is the keyCode and then compare this integer to constants in the KeyEvent class. The arrow key constants are VK_DOWN, VK_UP, VK_LEFT and VK_RIGHT.