Lab 15

Objectives:

Learn about listeners, events, and simple animation for interactive graphical user interfaces.

Files:

http://www.csc.villanova.edu/~map/1051/Chap04/SmilingFace.java
http://www.csc.villanova.edu/~map/1051/Chap04/SmilingFacePanel.java

http://www.csc.villanova.edu/~map/1051/Chap04/PushCounter.java
http://www.csc.villanova.edu/~map/1051/Chap04/PushCounterPanel.java

http://www.csc.villanova.edu/~map/1051/Chap09/Rebound.java
http://www.csc.villanova.edu/~map/1051/Chap09/ReboundPanel.java
http://www.csc.villanova.edu/~map/1051/Chap09/happyFace.gif

A. Adding a button to SmilingFacePanel.java:

Before you begin, download and compile SmilingFace.java and SmilingFacePanel.java; run SmilingFace.java and observe the panel it creates.

1. Import the class JButton from the package javax.swing, i.e., add the import statement:

   ```java
   import javax.swing.JButton;
   ```

2. Add an instance variable for the button – let’s call it `clicker`

   ```java
   private JButton clicker;
   ```

3. In the SmilingFacePanel constructor, instantiate a JButton object and assign it to `clicker`; then add the button to the panel.

   ```java
   clicker = new JButton("click here");
   add (clicker);
   ```

`clicker` represents a button that may be associated with an action. For the moment it does nothing when clicked, but compile your program and run it anyway. You should see the same picture, but with a button on it. When you click on the button, nothing happens. Next, we will fix this - make something happen...
B. Experiment with PushCounterPanel:

Download and compile PushCounter.java and PushCounterPanel.java; run PushCounter and observe the behavior of the button. The PushCounterPanel.java uses a JButton object. Just like our program, above, it is instantiated and added to the panel in the constructor.

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

public class PushCounterPanel extends JPanel {
    private int count;
    private JButton push;
    private JLabel label;

    public PushCounterPanel () {
        count = 0;
        push = new JButton("Push Me!");
        push.addActionListener (new ButtonListener());
        label = new JLabel("Pushes: " + count);
        add (push);
        add (label);
        setPreferredSize (new Dimension(300, 40));
        setBackground (Color.cyan);
    }

    class ButtonListener implements ActionListener {
        public void actionPerformed (ActionEvent event) {
            count++;
            label.setText("Pushes: " + count);
        }
    }
}
```
**Associating an action with a button**

The action(s) associated with clicking on a button are handled through something called an `ActionListener`. Observe that the `PushCounterPanel` class contains, inside it, the definition of another class: the `ButtonListener` class (the class that handles the event that a button gets pushed in the `PushCounterPanel` class). Because it is defined inside the `PushCounterPanel.java` class, `ButtonListener` class is called an **inner class**.

The `ButtonListener` class only has one method, `actionPerformed()` that specifies what has to happen (in this case a counter is incremented and the text in a label is replaced with the new value of the counter).

**Before proceeding, answer/do the following:**

1. What is the name of the variable representing the button in `PushCounterPanel` class? __________

2. What is the name of the variable representing the button in the `SmilingFacePanel` class (as modified in part A above)? __________

3. Circle the line of code that associates the action with the button in `PushCounterPanel` class. Copy it out here:

   __________________________________________

4. Modify `actionPerformed()` (method in the inner class) so that when the button is clicked, the background color changes to a different color.

**C. Implement an action for the button in SmilingFacePanel**

1. Import the package `java.awt.event.*`

2. We will create a listener for `clicker` named `ClickListener` as an inner class in `SmilingFacePanel`. The action will be to cause the background to change to **black** when clicker is pressed. It should not do anything else. We will do this as follows:

   - Add instance variables for red, green, and blue to be used in the background color. These are integers:

     ```java
     private int red, green, blue;
     ```

   - Add the following code to your `SmilingFacePanel` class.
private class ClickerListener implements ActionListener
{
    public void actionPerformed (ActionEvent event)
    {
        red = 0; green = 0; blue = 0;
        setBackground (new Color(red, green, blue));
    }
}

The clicker button should be controlled by the ClickerListener – we specify this in the constructor, as follows:

clicker.addActionListener(new ClickerListener());

3. Now we will add another button that, rather than causing the background to turn black with a single click, it will increase the amount of red in the background, gradually.

a. Follow the steps you used to implement the clicker button to add another JButton called redClicker with the text "More red".
b. In the constructor, assign the values: red=0; green=0; blue=0;
c. Still in the constructor, create a new color for the background, using red, green, blue:

        setBackground (new Color(red, green, blue));

        // this is the color black when red=green=blue=0

d. Add a listener for redClicker named RedListener (copy/paste the code for ClickerListener and adapt it). In the actionPerformed() method of this listener:

        red +=20;
        setBackground (new Color(red, green, blue));

What happens when you keep clicking the “More red” button repeatedly?

( Fix the actionPerformed() method to prevent this from happening.)

4. Once you get the redClicker to work, repeat the above steps to create buttons to increment the amount of blue and green.

D. Add a button to another GUI

Add a button to one of your earlier programs. For example you can add a button to HappinessPanel from lab 9 that causes a die to roll. The actionPerformed method of the listener should cause the corresponding action and then repaint the panel, eg:

die2.roll();
E. Make the color change (or the die to roll) by itself!

Using the Timer class, you can cause the color change in the background of SmilingFacePanel to happen automatically, at specified intervals. Follow these steps:

1. Import the class Timer from the package javax.swing, i.e., add the import statement:

   ```java
   import javax.swing.Timer;
   ```

2. Add an instance variable for the Timer – let’s call it ticker

   ```java
   private Timer ticker;
   ```

3. Add a constant for the delay:

   ```java
   private final int DELAY = 100; // (milliseconds)
   ```

4. In the SmilingFacePanel constructor, instantiate the Timer object and start it up:

   ```java
   ticker = new Timer(DELAY, new TickerListener());
   ticker.start();
   ```

5. Implement TickerListener as an inner class. For starters, have it function exactly like RedListener. Since the delay is set to 100 milliseconds, it will be as if you click the button 10 times every second. Observe it change color! Experiment with different functionality. For example, you might make the image move, by changing BASEX or BASEY in TickerListener. (you will first need to change BASEX to a regular variable by removing the “final” from its declaration). See ReboundPanel.java for a more interesting example of movement.

F. Add a button to make it stop

Add another JButton object with label “stop” to make the ticker (the timer) stop. The Listener for this button should simply invoke the stop() method for the Timer object, i.e.,

```java
ticker.stop();
```