Lab 6

In this lab, you will learn some of the tools used to build Graphical User Interfaces (GUIs) in Java.

Graphical User Interfaces (GUIs)

Begin by creating a new project and adding the file Lab6.java containing the following:

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

public class Lab6 {
    public Lab6() {
        JFrame frame = new JFrame("Lab 6");
        JButton button = new JButton("the button");
        frame.add(button);
        frame.pack();
        frame.setVisible(true);
    }

    public static void main(String[] args) {
        new Lab6();
    }
}
```

This is the general outline for most GUI code. You will create a window (in our case a JFrame) and then put different components in it. In this context, a component is an object that either takes input from the user (like the JButton in the example above) or displays something to them (we'll see an example of this later). At the bottom of the constructor are a couple of calls that make the window ready to display. The pack method arranges all the components and figures out how big the window needs to be in order to fit them. Then setVisible makes the window actually display. (You can also call setVisible(false) to make a window disappear without ending the program, though we don’t do that in this lab.)

Compile and run this program. You should get a little window with a button that doesn’t do anything. (You will get a couple of warnings as well, but ignore those.) Not too exciting. Exit the program by hitting the red button above the bottom panel of your Eclipse window. (Closing the window may not exit the program.)

To make the button do something, we need to install a listener. Listeners are objects that respond to actions such as a button being pressed. They run when the action occurs rather than being called by another part of the program. Using listeners is often called event-driven programming because actions are also called events.

To define our first listener, we implement the ActionListener interface, which contains the single method actionPerformed. Add the following to your code inside the Lab6 class (but outside its methods):

```java
private class QuitListener implements ActionListener {
    public void actionPerformed(ActionEvent e) {
        System.exit(0);
    }
}
```

By putting this class inside the Lab6 class, you are making it an inner class. Inner classes cannot be used except from within objects of the class containing them, but have the advantage of being allowed to access the attributes of that class (since they are part of it). When the actionPerformed method of a QuitListener
object is called, it causes the program to exit. Note the argument to `System.exit();` 0 means a “normal exit” and non-zero would indicate that the program exited due to an error.

Once you have the listener, associate it with the button using the following: (in the `Lab6` constructor)

```java
button.addActionListener(new QuitListener());
```

This creates a `QuitListener` object to receive events from the button. Also change the button’s label (the argument to its constructor) to “Quit”. Now when you run the program, the button actually does something!

How do we get multiple buttons in our window? Use cut and paste to create a second button object (with a different label) and add it to the frame. When you compile and run this code, the results are disappointing; the new button covers the old one. That’s because we didn’t tell Java how to arrange the buttons. We need to specify a layout manager, an object that arranges different components in the frame. Do so by adding

```java
frame.setLayout(new FlowLayout());
```

right after creating the frame (near the method top). Now the buttons should appear side by side.

Add two more buttons and see how they appear. Then resize the window and see what happens.

Next, change the layout manager to `GridLayout` using `new GridLayout(2,2)`. See how the buttons appear and how resizing the window works now. Also, try a 3 by 2 grid (still with only 4 buttons).

The last layout manager we will try is `BorderLayout`. To use this, set the layout manager to a `BorderLayout` object. Then, when you add a button to the frame, include `BorderLayout.NORTH`, `BorderLayout.SOUTH`, `BorderLayout.EAST`, `BorderLayout.WEST`, or `BorderLayout.CENTER` as a second argument to `add`. Assign each button to a different region and see what happens. Be sure to try resizing the window.

There are other layout managers, but more complicated layouts are typically constructed by nesting simpler ones. Use a `JPanel` object to create a nested layout as follows:

```java
JPanel panel = new JPanel(new FlowLayout());
panel.add(new JButton("button 5"));
panel.add(new JButton("button 6"));
frame.add(panel, BorderLayout.SOUTH);
```

Note that the `JPanel` itself contains buttons, which are organized by the layout manager established when the panel is constructed, but also serves as a component in the frame. Put this code into the constructor and see how it appears. Try different variations to see how they are drawn.

Our last lesson is how to use the `JLabel` component for output. Create a `JLabel` by passing a String to its constructor. Add it to either `panel` or `frame`. When you run the program, the `JLabel` appears as text. Unlike a button, however, this text cannot be pushed; it is just for output. To make the label more useful, make it an attribute instead of a local variable. Then create a listener whose `actionPerformed` method calls the label’s `setText` method. (Recall that inner classes can touch object attributes). The `setText` method takes a string argument, which becomes the label’s new text. Add this listener to a button, run the program, and see what happens when the button is pressed.

For the rest of the lab period, try to create the GUI for a calculator, with a `JLabel` at the top and a grid of numbers and operators below. For this you will need to use multiple layout managers nested within each other. Before writing code, draw a picture of the GUI and figure out how to specify the desired layout.

If you have more time, see if you can make your calculator actually work. A good first step is to have it display an entered number. What attributes and listeners will you need? I suggest you use the same type of listener for multiple buttons. For example, all the number keys can use listeners from the same class since they do nearly the same thing; just have a distinct object (with distinct attribute values) for each button.