CS 2300 Lab Week 11

Basic MVC GUI – Part 3

Control Panel and Programmer Panel

The goal of this project is to set up a set of basic classes implementing a minimal MVC (Model View Controller) GUI (Graphical User Interface). You can subsequently use it as a template for full featured GUI projects.

1. Continue on with the project from the last lab. This should closely parallel the classroom development.

2. Create a new class called `ControlPanel`. This should extend the class `JPanel`. Put a field in it:

```
protected final Model model;
```

Note that this is `protected`, not `private` as usual.

2.1 Write (or, automatically generate) a constructor for this class. In addition to the usual, add a line so that it looks like this:

```
public ControlPanel(Model model) {
    this.model = model;
    setBorder(BorderFactory.createEtchedBorder());
}
```

This class will be used as a parent class for several "mini panel" classes, defined next. You might get a warning about an "overridable method call in c-tor". We can ignore it.

2.2. The purpose of the `ControlPanel` class is to:

2.2.1. give each mini panel a consistent look (with an etched edge border), and

2.2.2. define the `model` field in one place (in the parent class) rather than four. Because that the `model` is `protected` (not `private`), six **child classes** will be able use the `model`.

3. Create a new class called `ProgrammerPanel`, and make it extend the class `ControlPanel`, like this:

```
public final class ProgrammerPanel extends ControlPanel {}.
```

This will cause an error balloon to pop up. Click the yellow balloon and allow NetBeans to fix the error by creating a c-tor for the class. Next do the **exact same thing** to create five more classes. Give them the names `SizeXPanel`, `SizeYPanel`, `ColorPanel`, `SolidPanel`, and `ViewPanel`. (Each of these should extend `ControlPanel` and have the c-tor). Each week we will add code to make these panels form the controls for the application.

4. Go to the `Controller` class and upgrade the c-tor with several statements, so that it looks like this:

```
    public Controller(Model model) {
        this.model = model;
        setBorder(BorderFactory.createLineBorder(Color.GREEN));
        setLayout(new GridLayout(6, 1));
        add(new ProgrammerPanel(model));
        add(new SizeXPanel(model));
        add(new SizeYPanel(model));
        add(new ColorPanel(model));
        add(new SolidPanel(model));
        add(new ViewPanel(model));
    }
```

This is a legal program, so run it and see how it looks. It should look almost the same as before. However, note that there are now six little boxes down the left edge. Note the borders of these six boxes have an "etched look". That is because of the type of border we used in the parent class, `ControlPanel`.

We will put some stuff in these boxes to make the application look more like a GUI.

In the c-tor above, we are creating instances of the six new classes right inside the add statement, exactly where they are needed. Instances like this (that do not have a name) are called **anonymous instances**.

5. For the first control panel we will just put a title and the name of the programmer (That is YOU!). Go to the `ProgrammerPanel` class. Add some code so that it looks exactly like the following, EXCEPT put in your name, NOT Ralph Kelsey:

```
public final class ProgrammerPanel extends ControlPanel {
        JLabel jlProgrammer = new JLabel("Programmer:   ");
        JLabel jlYourName = new JLabel("Ralph Kelsey");


    public ProgrammerPanel(Model model) {
        super(model);
        add(jlProgrammer);
        add(jlYourName);
    }
}
```

Here we used the Swing component called `JLabel`. A `JLabel` is just a panel with some text (or pictures, whatever) on it. We are instantiating two labels (`jlProgrammer` and `jlYourName`) and `add`ing them to the `ProgrammerPanel` we are constructing. Notice we ALWAYS use a standard convention for naming the elements comprising our GUI. All `JLabel`s start with the letters `jl` and then the name of what they are. If you do this 100% of the time, it is a lot easier to keep track of what components are.

Run the program. Now you should see the Control part of the GUI (left hand side) as a stack of six boxes. You see the etchedBorder we put in the superclass (`ProgrammerPanel`), so all the child classes get it automatically. This is how **inheritance** pays off big! We only added some "stuff" to the top box, the "Programmer Panel". Note how the `GridLayout` manager figures out how much width it needs for that panel, and makes all the other ones the same size.

6. Show the lab instructor that your application shows your name as the programmer.