CS 2300 Week 10 Project: Basic MVC GUI – Part 2

Continue on with the project from the last lab. This should closely parallel the classroom development. This week we will put three basic areas onto the frame; the title, controller, and view panels.

1. Go to the `Model` class code. Find the spot right after the c-tor but before the main method. Left click on the code at this spot, so that the next thing added will be positioned there. Click and use alt+ins to automatically generate four **getters**, one for each of the four fields. For example, one should look like this:

```
public Controller getController() {
    return controller;
}
```

Do not create **setters** (NetBeans will not allow you to anyway). These values should never be reset. Furthermore, the compiler will NOT allow you to change these fields, because they have been declared `final`.

2. Now the title, view, and controller panels can be added to the frame. Go to the `FrameMVC` class, and add the following, right before the `setVisible` command:

```
setLayout(new BorderLayout());
add(model.getTitle(),BorderLayout.NORTH);
add(model.getController(),BorderLayout.WEST);
add(model.getView(), BorderLayout.CENTER);
```

Here is what is going on. This code is in the c-tor for the `FrameMVC` class, so the `setLayout` statement defines how we will paste the three panels onto the frame.

A statement of form `add(X,Y);` will paste a panel named `X` into region `Y`, which is one of the five regions (N, S, W, E, C) defined by the border layout manager. To paste the title panel into the top position, we need to replace `X` with the address of the title panel. We can get that by asking the model to send us the address, which is what `model.getTitle()` does. We replace `Y` with the official name of the top region, which is `BorderLayout.NORTH`. The statement `BorderLayout.NORTH` does all of these things. The other two panels are placed similarly.

Now run the program. You cannot see any change because the three panels do not have anything on them to see, so it is just three blank panels on a frame; nothing to see. Be sure to close the running GUI so that you do not wind up with a bunch running at once.

In order to see where the three panels are on the frame, we can add borders to the panels. Go to the `Controller` class and add the following line in the c-tor, right after the part about the model:

```
setBorder(BorderFactory.createLineBorder(Color.GREEN, 2));
```

When you fix the imports, make sure you select `java.awt.color`, because there are some other color classes that you don't want.

Do the same thing in the `View` class, but use BLUE rather than GREEN.

Next go to the `Title` class. Have NetBeans create a c-tor. Add the `setBorder` code, but use RED rather than GREEN. Run the program.

Run the program. Now you can see where the controller and view live on the frame. The controller and Title are very thin, because we have not put anything in them yet! Thus most of the frame goes to the View by default because it is located in the Center region.

Try to experiment with changing the number in the borders from 2 to 1, 0, 3, 4, 5, etc., to see how it looks. Probably 2 is a good value to leave it at.

3. Now we are ready to paint something on the view.

3.1. Go to the `Title` class, and add the following lines to the c-tor:

```
public Title() {

    setBorder(BorderFactory.createLineBorder(Color.RED, 2));

    Dimension dimension = new Dimension(144,144);

    setPreferredSize(dimension);

}
```

3.2. Fix import as needed.

3.3. The lines `Dimension dimension = new Dimension(144,144)` and `setPreferredSize(dimension)` are usually the best way to try make a panel be some given size. Be aware that setting sizes in a GUI is very tricky, because the user can change the size of things, and so on. Here the number 144 is just a guess to make the title panel look OK. Even though we requested a 144 pixel by 144 pixel square for the preferred size, the North region of the Border Layout stretches all the way across the frame, which should be 1000 pixels long. This illustrates the fact that the layout managers balance a lot of requests and do what they think is best for setting up where stuff goes on the frame.

3.4. Now add the lines

```
    repaint();              // keep in last line of c-tor
```

at the end of the constructor. The `repaint` method lives up near the top of the GUI class hierarchy, in a class called `Component`. This tells the JVM (Java Virtual Machine) to repaint the area of the screen controlled by the class where the `repaint` statement is. The `repaint` method is looking for a method named `paintComponent` to see what to do. This method is called by the JVM whenever the screen needs to be repainted. If we do not add such a method, a "do nothing" default method will be used.

3.5. Next we will add a `repaint` method to write out a title for us.. Add the following code to the class, right AFTER the c-tor, but before the final `}` at the end of the class:

```
protected void paintComponent(Graphics g) {

    super.paintComponents(g);

    int halfWidthGuess = 180;                                  // guess
```

```
        int xCenter = getWidth() / 2;
        int startTitleX = xCenter - halfWidthGuess;
        double magicFactor = 0.58;                              // guess
        int startTitleY = (int) (magicFactor * getHeight());

        Color titleFontColor = new Color(15,15,255);
        g.setColor(titleFontColor);
        Font font = new Font("Arial", Font.BOLD, 30);
        g.setFont(font);
        g.drawString("Model - View - Controller",
                startTitleX, startTitleY);
    }
```

NetBeans will offer to add the `@Override` annotation, because our code **overrides** the "do nothing" method with the same name somewhere higher up in the inheritance hierarchy.

3.6. Here is how graphics works. The JVM actually calls your `paintComponent` method to paint whatever component it is putting on the monitor. The JVM sends along an object `g` of class `Graphics`. You can use `g` to call any of the many built in drawing functions of the `Graphics` class, and thus draw stuff on the panel. The first line of code:

```
super.paintComponents(g);
```

sends the graphics object `g` to the `paintComponents` method of `JPanel` (the parent class), in case it wants to initialize anything first. The word `super` means the parent class of the class where the code is.

The next five lines are some calculations to try to get the title to be put in a good location. This a very ad hoc (because I am too lazy to figure out how to do it right now). By adjusting the 180 and 0.58 you can move the title around. Try some changes to see what happens. Also change the **font size** 30 to see what happens. Try making it real big. Probably a good idea to restore the original values afterwards.

The next line defines a color; a slightly light blue. The following line sets this color to be used until further notice. If you want to, you can reset the color before each pixel is drawn!

The next line defines a font, and the one after that sets this font to be used by the `drawString` method until further notice.

Finally, the last line uses the `drawString` method to draw the title (`Model - View - Controller`) at the designated starting place.

3.7. Run the program and see how it goes. You should have a nifty title. Try putzing around with the parameters that define the title to see what happens.

4. Now we will also do something in the view panel. Go to the tab with the `View` class and do the following.

4.1. Add `repaint();` to the end of the constructor.

4.2. Add the following method right after the c-tor.

```
@Override
protected void paintComponent(Graphics g) {
    super.paintComponents(g);                // call the super class c-tor
    // center of view panel, in pixels:
    int xCenter = getWidth() / 2;            // calc center of panel
    int yCenter = getHeight() / 2;
    // add your painting operations here
}
```

Run the program. Nothing much should change, but at least you know you did not do anything to crash the program. This is the "Add one little thing. Test before you add the next thing". If anything goes wrong, you want to deal with it ASAP.

5. Add some temporary drawing stuff to see how it works. Type `g.` and check out the pulldown menu of all the stuff you can do. Try out a couple of things, such as `g.drawLine(10, 20, 30, 40);`, to see what happens. Change the numbers and try again. Maybe try some of the other things, if you can figure out what parameters they need. Then erase them.

6. Go to the `model` class. Add the following fields and default values, right after the other four fields:

```
private int xSize = 66;
private int ySize = 44;
private Color color = Color.BLUE;
private boolean solid = true;
```

Notice that these fields are **not** `final`.

6. Click on the model class after all the existing methods, but right before the `main` method. Hit Alt + Ins and let NetBeans create getters and setters for these four new fields.

7. Next return to the `View` class and add the following code to the `paintComponent` method (after the existing code):

```
    int xStart = xCenter - model.getxSize();
    int yStart = yCenter - model.getxSize();
    int width = 2 * model.getxSize();
    int height = 2 * model.getySize();
    g.setColor(model.getColor());                // changes drawing color
```

```
if (model.isSolid()) {
    g.fillOval(xStart, yStart, width, height);
} else {
    g.drawOval(xStart, yStart, width, height);
}
```

Note how the decision to draw a solid circle or an outline circle is made by asking the model. The `isSolid` method returns `true` or `false`, and the `if-else` statement takes care of it! This illustrates how the `model` object holds all the information about how to paint the view.

8. Try this out. Experiment around in the `Model` class by changing the default values to see what happens when you run the program. For example, you could try adding this code (after the place where the color is set).

```
private Color newColor = new Color(255, 0, 255);            // temp
g.setColor(newColor);                                       // temp
```

This should create a magenta color. Try it and see. Try out some other number combinations. What happens if you choose an out of range number? (the valid range is [0:255].) (The worst that can happen is you crash the program!) After you experiment with some colors, remove these temporary lines.

What to turn in. Show the lab instructor that you have the application window up and running. There should be a blue ellipse in the view panel.