CS 2300 Lab 09 MVC GUI – Part 2 Week 09

Title Panel and View Panel

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

Reading suggestions. Several Java topics will be introduced this week:

JPanel + JLabel classes in the Swing GUI package.

Layout. Layout managers: Flow LO, Border LO, Grid LO.

Color class. RGB system. Foreground Color, Background Color.

Font classes. Font Name, Font Size.

Graphics class. Repaint method.

Pixels: getWidth, getHeight methods

2. Next we place three panels on the frame: the title, controller, and view panels, so that the title, view, and controller panels can be added to the frame.

In the c-tor for the `Frame` class:

Locate the spot right after `System.out.println("Frame c-tor")` but before the line `setDefaultCloseOperation`:

Add the following

```
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 `Frame` 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 other two panels are placed similarly.

3. Now run the program. Now you can see the three panels because we have set different colored borders on them.

The Controller and Title panels 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.

View Panel

4. Now we will make something happen in the view panel. Go to the the `View` class and do the following.

4.1. Add

```
repaint();
```

to the end of the constructor. This statement calls a method of the `JPanel` superclass to cause the **view** to be redrawn. It is at the end of the c-tor to paint the view the first time. We will also call `repaint` after any change is made to the model, so that we can see the result of the changes.

4.2. **KEY STEP!** So far, the `repaint` step uses a default "do nothing" paint job. We will fix that by writing our own `paintComponent` method. This is how we define what happens when the view gets repainted.

Add the following code right after the c-tor for the `View` class:

```
@Override

protected void paintComponent(Graphics g) {

    super.paintComponents(g);           // call the super class c-tor

                                        // painting operations go here

}
```

Fix imports and 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.

4.3. Here is how **graphics** works. The JVM (Java Virtual Machine) actually calls your `paintComponent` method to paint whatever component it is painting 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. In other words, before any paint commands we add, any parent classes paint operations are done, and maybe others if there is as "grandparent class", and so on.

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);
```

Run the program to see what happens. See the diagonal line up in the left corner of the View panel?

Change the numbers and try again. Maybe try some of the other things, if you can figure out what parameters they need.

6. In the paintComponent method, erase the g.drawLine statement (that was just to see how it works). After the `super.paintComponents(g)` statement, put in the following:

```
  // center of view panel, in pixels:

int xCenter = getWidth() / 2;

int yCenter = getHeight() / 2;


  // upper left corner of object we will draw:

int xStart = xCenter - model.getxSize();

int yStart = yCenter - model.getySize();


  // width of object we will draw:

int width = 2 * model.getxSize();

int height = 2 * model.getySize();


  // update drawing color

g.setColor(model.getColor());


  // clear out the old picture

g.clearRect(0,0,getWidth(),getHeight());
```

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

Run the program to try this out. Now we finally have something to look at!

7. Discussion:

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.

Because we are drawing stuff on an object of type `View`, which is a subclass of `JPanel`, we have to calculate where we want to draw things. The `getWidth` and `getHeight` methods get the size of the `JPanel` we are drawing in, measured in pixels.

7.1. Experiment around in the `Model` class by changing the default values to see what happens when you run the program. For example, you could change

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

one at a time to

```
    private int xSize =444;
    private int ySize = 111;
    private Color color = new Color(255, 11, 200);
    private boolean solid = false;
```

These four variables are an example of what are called **state variables** because the define the “state” of the system we are working with. This c-tor for the `Color` class takes three integer arguments, each in the range

[0, 255], for the proportion of RED, GREEN, and BLUE to use to define a color. Try to guess what the combination (255, 11, 200) will give you.

Soon we will add controls to the project so that we can change `xSize`, `ySize`, `color`, and `solid` while the program is running, and see what happens immediately.

7.2. A good place to experiment with features of the `Graphics` class is right in the `paintComponent` method of the `View` class.

For example, you could try adding this code (after the place where the color is set).

```
Color newColor = new Color(255, 0, 255);                    // temp

g.setColor(newColor);                                       // temp
```

Unless you want to save `newColor` for future reuse, you can combine these above two lines as:

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

In this “shortcut” way of doing it, we are **not** saving a reference to the new color, because we do not need it for anything. Instead we are just creating the new `Color` object and sending it off to where it is needed. This only works because memory is **garbage collected** in Java. That means the memory holding this color will eventually be reclaimed by the system, without any code from the programmer. (Very different – and much easier than in C++).

The combination above should create a magenta (Red + Blue) 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.

8. Put a nice looking title in the Title panel.

8.1. Go to the `Title` class, and add the following fields at the top of the class (above the c-tor)

```
    // user defined parameters. Adjust as needed or desired

private final int fontSize = 36;

private final String fontName = "Arial";

private final String titleString = "Model - View - Controller";

private final Color titleColor = new Color(15, 15, 255);

private final Font titleFont = new Font(fontName, Font.BOLD, fontSize);
private final JLabel jlTitle = new JLabel(titleString);
```

We use six fields (`fontSize`, `fontName`, `titleString`, `titleColor`, `titleFont`, `jlTitle`). The first five define the look of the title. The last field is a `JLabel`, which allows any text or graphics to be placed on our GUI.

The objects of type `JLabel` and `Font` will be used to place the title at the top of the app. Here we follow the plan: Do as much as possible at the top of a class, before the c-tor. This helps keep the c-tor from being cluttered.

8.2. Next add the following to the c-tor:

```
jlTitle.setFont(titleFont);

jlTitle.setForeground(titleColor);

add(jlTitle);
```

Fix imports as needed.

The statements inside the c-tor are **method calls**. Method calls are **executable statements**, so they must be inside a method, such as the c-tor. After setting the border style, the next two statements adjust the font and color of the text on the `JLabel`. The last line **adds** the `JLabel` to the Title Panel. The default layout manager (flow layout) will place the label in the middle of the title panel.

8.3. 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.

What to turn in.

1. Open file explorer

2. Find your project (likely C:\Users\<user>\IdeaProjects\MVC22F)

3. Right click on your project folder (this should contain src\mvc22f with all your .java files)

4. Select Send to > Compressed (zipped) folder

5. Submit MVC22F.zip to Blackboard