CS 2300 Lab Week 9

Basic MVC GUI – Part 1

Start Project. Model Class. JPanels and JFrames

The goal is to build a simple application with a simple GUI (Graphical User Interface). The project is organized according to the MVC (Model View Controller) paradigm. While this project is as simple as possible so that it can be completed in less than half a semester, much more useful applications can be created similarly. Because a lot of code is required for a GUI, we will let NetBeans provide as much as possible. This project will be continued for several weeks.

0. **Preview**. There will be five classes that are "top level" in the sense that all other classes are "contained in" one of the top level classes:

0.1. Model. This class holds all the "state data" that defines how the view is drawn.

0.2. View. This class and its subclasses are responsible for drawing the view.

0.3. Controller. This class and its many subclasses present various controls that allow the data in the model to be changed.

0.4. Title. This is a very simple class that simply draws a large title across the top of the application.

0.5. MvcFrame. This class is like a canvas that lives on a computer monitor, and has controls and a view of the application painted on it.

Develop the GUI: Step by Step.

In the early pasts of this project, detailed instructions for each step will be provided.

1. Fire up NetBeans and create a new project called `MVC17F`.

2. Create a package in this project named `mvc17F`. There will be many classes in this project. Put them all in this package.

2.1. On the toolbar at the top of NetBeans, click on "Run". On the resulting pulldown menu, click on "Select main project". Pick `MVC17F`.

3. Create a class called `Model` (this and all other classes go inside the package). Leave it empty for the moment. Eventually this will be the central class for the entire project. We will be adding one thing at a time to this model.

4. Next we will make some classes that can show up on the monitor! This is easy to do with **inheritance**. The basic rectangle that can be placed on the monitor is called a `JPanel`. By making some class **extend** `JPanel`, the child class knows how to present itself on the monitor.

5. Create a final class called `Title`. NetBeans will write it like this:

```
public class Title{}
```

Change it to this:

```
public final class Title extends JPanel{}
```

You should get some error messages. You can cure them by hitting `alt+shift+I` to fix the imports. We will put some more stuff in this class later.

6. Create a class called `Controller`. NetBeans will create it like this:

```
public class Controller {}
```

Change it to this:

```
public final class Controller extends JPanel{}
```

The word `final` is to avoid an obscure problem. Whenever you get an error message about "Constructor Calls Overridable Method", this will fix it.

This class needs to communicate with the model. This is accomplished by having a **reference to the Model class**. You can think of this as being similar to having a phone number for the Model class, so you can call and ask for stuff whenever you need to. To make this happen, add one field (at the top of the class):

```
private final Model model;
```

The keyword `final` means that once the value is set, it can never be changed. Because there is no reason why this should ever be changed, the `final` allows the compiler to help us out by guarding for an accidental change to the variable. It is a good program practice to always make variables `final` unless there is a reason not to.

You will get an error message about no default c-tor. We will fix that next. Recall that hitting `alt+ins` brings up a context sensitive menu for creating simple methods for us. Do it like this: click `alt+ins` to bring up the menu, and select constructor (also called a c-tor). Check the bottom box on the left, and the box for model on the right. The reason there is a box on the left, is to choose options for the c-tor for the **superclass**, which is `JPanel`. When you have a class that extends another class, the first thing a c-tor does is automatically call the c-tor of the parent class. This goes all the way up to the class `Object`, which is at the top of the class hierarchy.

You might still be getting some “red spots” indicating something not compiling. On the second row toolbar click on the “hammer and broom” icon. This will “clean and build” the project. You need to do this whenever the compiler gets confused about a change you have made.

Add this line at the end of the c-tor:

```
System.out.println("Controller c-tor");
```

This will print a line to the output area of NetBeans, and is just to help us understand what is going on.

7. Create a class called `View`, and do the exact same stuff as you did for the `Controller` class.

8. A **Frame** is a class that can be drawn on the monitor, and contain an entire project. Create a class called `MvcFrame`, like this:

```
public final class MvcFrame extends JFrame
```

Add one field:

```
private final Model model;
```

Add a c-tor (check the top box on the left) and check the box to initialize `model`. When we have to click a box on the left side, the simplest one is usually the best bet. At the end of the c-tor, put in

```
System.out.println("MvcFrame c-tor");
```

9. The `Model` class is center of the entire project. It holds the **state** (all the information) of the application. It is also the communication system that allows the other parts of the project to communicate. Return to the `Model` class and add the following lines:

```
private final Title title = new Title();

private final Controller controller;

private final View view;

private final MvcFrame mvcFrame;
```

The field `title` is **instantiated** outside of a c-tor because it can be. The other fields need special treatment in a c-tor. This is causing some red spots for compilation issues.

8. We fix the error by adding a c-tor. Let NetBeans generate a c-tor, but do not check any of the boxes (or, uncheck them if they are already checked). The other fields `controller`, `view`, and `mvcFrame`, are **instantiated** inside the c-tor with the following code:

```
controller = new Controller(this);

view = new View(this);

mvcFrame = new MvcFrame(this);

System.out.println("Model c-tor");
```

It is important that the line instantiating the `MvcFrame` object come after lines instantiating the `Controller` and `View` objects.

Three of these classes need to be able to communicate with the model. The **argument** sent to the constructors -- `this` -- is like sending the model's phone number to the other classes, so that they can phone home to get information when needed. The `Title` class does not need to communicate with the model.

9. We will also need a `main` method somewhere to launch the entire project. A good place for this is right in the `model` class.

Go to the bottom of the class (right before the last closing brace: `}`), and enter `psvm+tab` to generate a `main` method. Put the line

```
System.out.println("MVC GUI Application");
```

inside the `main` method.

Click the green triangle in the second tool bar to run the project. A pop up menu will ask you to select the main class. The only class listed is `Model` (because it is the only one with a `main` method), so select it. Now click the green triangle again to run the program. It should write

```
MVC GUI Application
```

on the console. This is just to help us understand what is going on:

The first thing to happen is the `main` method of `Model`, which writes `MVC GUI Application`, and that is the end of this preliminary version of the project. Note that we are **checking every step as we go!!!**.

10. New we will add some code to launch a very simple (so far) GUI. Write the following code exactly as it is given:

```
SwingUtilities.invokeLater(() -> {
    Model model = new Model();
});
```

This is the new concise way (using **lambda**s, added in Java 8) of the "old style" version:

```
SwingUtilities.invokeLater(new Runnable() {
    @Override
    public void run(){
        Model model = new Model();
    }
});
```

which is slightly easier to understand. Feel free to consider this pure magic that nicely launches the GUI framework as a **separate thread**. Most GUI programs are multi-threaded to improve performance.

If you happen to be interested in how this all works, consider Googling on `Runnable` and `SwingUtilities`. Roughly speaking, here is what is happening. We are creating an **anonymous object** of the **`Runnable` interface**. An anonymous object means we do not give the object a name or a type, but just construct it where it is needed, specifying whatever code is needed. This maneuver is done all the time in GUI programming.

The **`Runnable` interface** requires that the object be provided with a method called **`run`**. In this case the **`run`** method only contains one line: `Model model = new Model()`. What the `SwingUtilities.invokeLater` does is launch a new thread that that will "run the GUI". Congratulations! You are now writing a **multithreaded** program! You can now click shift+F6 to run the program. It does not do anything, so it will be hard to know it is running! Next we will make it do something we can see.

11. Now click the green triangle to run the program again. You should see this:

```
MVC GUI Application

Controller c-tor

View c-tor

MvcFrame c-tor

Model c-tor
```

on the console. Here is what is happening:

(1) The program starts in the `main` method of `Model`, which starts by writing `MVC GUI Application`.

(2) Next the `SwingUtilities...` line instantiates a new object of type `Model`.

(3) In the `Model` c-tor, an object of type `Controller` is instantiated. So, next the `Controller` c-tor method gets called and writes out: `Controller c-tor` among other things.

(4) Same deal for the `View` and `MvcFrame` objects, each of which writes out something.

(5) Finally, we get to the end of the `Model` c-tor, and write out `Model c-tor`.

All of this output is to help develop an understanding of what is going on. (Remember: everything is easy once you figure it out).

12. Now we will add some stuff to the `MvcFrame` class so a GUI is produced. In the c-tor, `MvcFrame` after the `this.model = model;` line already there, add the following lines:

```
setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
```

```
setSize(1000,700);

setTitle("MVC 17F");

setVisible(true);
```

Try to guess what each of these lines does. The names are pretty good, so you should be able to figure out the basic idea!

13. Run the program (click the green triangle). This time a GUI window should appear. Pretty cool. You cannot click on the green triangle twice (because it is "grayed out"). But you can click on the `Model` class and hit Shift + F6 and launch additional copies of the GUI. That is because each one is launched in a separate thread, so you can have as many as you want!. Close the windows by clicking the `X` in the upper right corner. Make sure you close ALL the instances of the program that are running. Each one looks like a little coffee cup down in the task bar.

14. Do a little experimenting with the last three lines you added. For example, eliminate one and see what happens when you run it. Or, use a different size or title. This is how you figure out how stuff works.

What to turn in. Show the lab instructor that you have the empty GUI window up and running.