

# CPSC 2151

## Lab 11

Due: Friday November 30<sup>th</sup> at 11:59 pm

In this lab you will be working with the Model View Controller design pattern with a Graphical User Interface created using Java Swing. The View (GUI) is provided for you, and you have already completed the Model. For this lab you will just need to complete the controller.

### Instructions

You will need to create a new project with a package called cpssc2150.mortgages. Add your Mortgages.java and Customer.java classes to that package. You will then need 3 more classes and one interface:

#### MortgageApp.java

MortgageApp will be a very simple class. It will contain our main function that will be the entry point of the program. All it will do is declare an instance of IMortgageView and MortgageController, and connect them. After that the system will wait for an event from the IMortgageView object. The code should look like:

```
package cpssc2150.mortgages;

public class MortgageApp {
    public static void main(String [] args)
    {
        IMortgageView view = new MortgageView();
        MortgageController controller = new MortgageController(view);
        view.setController(controller);
    }
}
```

This type of set up is common when using MVC architectural pattern in Java. We just need the entry point to set up our Controller and View.

Set up your configuration so this is the entry point for your program.

#### IMortgageView.java and MortgageView.java

This interface and implementation will serve as the view layer of our system. They provide several methods to get input from the user and display output to the screen. The view layer does not provide any input validation, that will need to be handled by the Controller.

Inspect the interface to make sure you understand what methods are available to you. Read through the code for the view as an example of building an interface in Java Swing. You should not need to make any changes to these files.

#### MortgageController.java

The MortgageController class serves as the Controller layer in our MVC architecture. It controls the flow of control in our program. This class is able to use both the view and the model layers to

accomplish this task. This class will check to make sure the data that the user provides (through the View layer) meets the preconditions that exist in our model layer. If the input is not valid, the controller will (through the view) alert the user to the error and wait for them to correct their input and resubmit. If there are multiple errors the controller only needs to report one to the user. The MortgageController class will also use the model layer (Mortgage and Customer class) to actually apply for the mortgage. After a mortgage has been applied for, the Controller will (through the View layer) say whether or not the loan was approved. If the loan was approved, then the rate and monthly payments will be displayed as well. If the loan was denied then 0 will be displayed for the rate and monthly payment.

The MortgageController class will have one private field, an IMortgageView Object, which is set by an argument passed into the constructor. The Mortgage Controller class will have one public method, called submitAppication() which is called after the user hits the submit button. The view layer is already observing the submit button, and will call submitApplication when the user clicks the submit button.

All of the logic concerning validating the Models preconditions, and the order of events, is handled by the Controller layer. The logic concerning the Mortgage application or the Customer itself are handled by the Model Layer.

Your starter code for the Controller layer is as follows:

```
package cpsc2150.mortgages;

public class MortgageController {
    private IMortgageView view;

    public MortgageController(IMortgageView v){
        view = v;
    }

    public void submitApplication()
    {
        //Your code here
    }
}
```

### TIPS and additional Requirements

- You do not need to provide a makefile for this assignment, nor do you need to test your code on unix (GUIs and Unix don't play well)
- You will zip up your project directory and submit that on handin
- You do not need to provide any automated testing, although you should test your program.
- You do not need to provide any contracts for MortgageView, MortgageController, or MortgageApp. Mortgage and Customer should have contracts from a previous assignment.
- You do need to comment your code
- You need to follow the Model View Controller architectural pattern for this assignment.
  - o All interaction with the user goes through the View class. View does not have access to the Model

- Controller handles input validation, and the order of events of the program. The controller has access to the view and the model, so it is the go between for those two layers
- The model handles our entity objects and the “lower level” logic. It does not have any access to the other two layers.
- Follow our best practices: No magic Numbers, information hiding, separation of concerns, etc.
- The Controller will be very different from lab 10, as the flow of control is very different now that we are using a GUI and are reacting to button clicks.
- Make sure to understand what all the methods in our View layer do. They will be very helpful.
- Depending on the resolution of your screen, it may be difficult to read the text on the GUI. Java Swing is not great with font sizes.

### **Groups**

You may, but are not required to, work with a partner on this lab. Your partner must be in the same lab section as you, not just the same lecture section. If you work with a partner, only one person should submit the assignment. You should put the names of both partners in a comment at the top of the file in order for both partners to get credit. This assignment may take more than just the lab time. Make sure you are able to meet outside of class to work on the assignment before you decide to work with someone else. Remember to actively collaborate and communicate with your partner. Trying to just divide up the work evenly will be problematic.

### **Before Submitting**

You do not need to test your code on Unix or provide a makefile. You will zip up your project directory and submit that.

### **Submitting your file**

You will submit your file using handin in the lab section you are enrolled in. If you are unfamiliar with handin, more information is available at <https://handin.cs.clemson.edu/help/students/>