W. P. CAREY SCHOOL of BUSINESS
ARIZONA STATE UNIVERSITY

CIS 494 – Business Systems Development with Java – Spring 2015

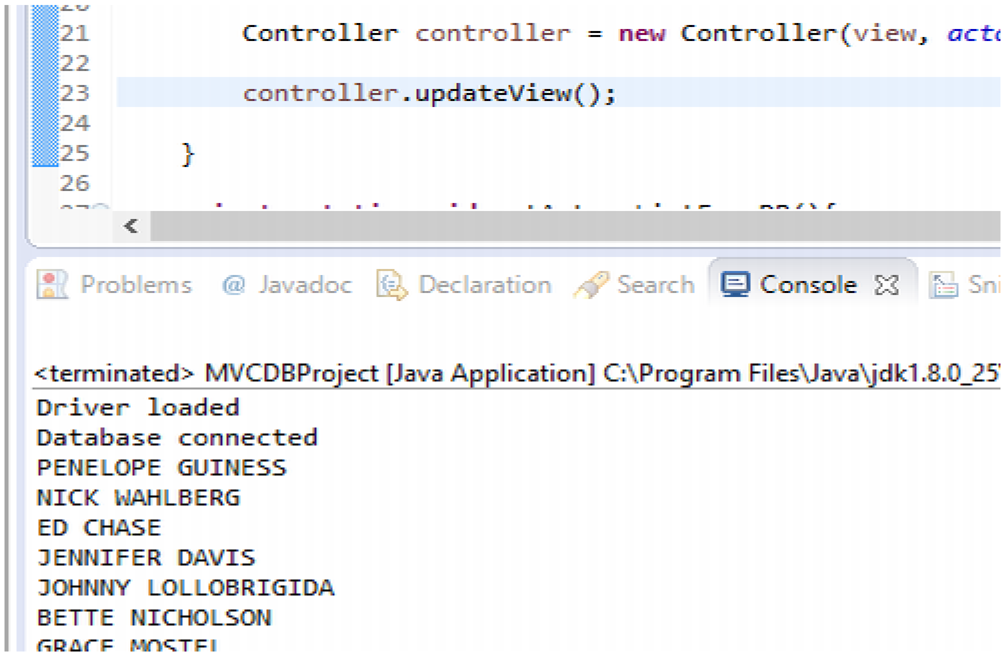
In-class Exercise 10: MVC Design Pattern

Due on Blackboard: Tomorrow, Thursday, April 9 by 11:59 PM

In this exercise, create a basic command-line application that implements a Model-View-Controller architecture. You will query the MySQL database to retrieve a result set. Your SQL query will retrieve a list of all the Actors in the database. You will take the resultset and populate a “model” with it. The model is merely a class with an ArrayList of Actors.

There will be a dedicated class, called the actorView, which will be able to display the model. For the purpose of brevity and simplicity, we will only use the console to display the names using println statements. However, in a larger project, this view could be a JavaFX stage.

A Controller class will have instances of both the view and the model. The controller will be able to direct the view to “update” and show the latest model. In a larger application, the controller could also respond to events in the view, i.e. a user updates the name of an existing actor in a Window environment or adds a new Actor. In that case, the controller would update the model as well.



```
        Controller controller = new Controller(view, act

        controller.updateView();

    }
```

```
<terminated> MVCDBProject [Java Application] C:\Program Files\Java\jdk1.8.0_25
Driver loaded
Database connected
PENELOPE GUINESS
NICK WAHLBERG
ED CHASE
JENNIFER DAVIS
JOHNNY LOLLOBRIGIDA
BETTE NICHOLSON
GRACE MOSTEL
```

1. Create a new Java Project and make sure you add the MySQL driver JAR file as an external JAR file.

2. Create your outermost class with the main method in it – call it ***MVCDBProject***. Leave this class empty for now. We will create other classes and come back and fill in details here once those classes are ready.

3. Create a new ***Actor*** class with attributes for firstName and lastName. Add getters, setters, and a constructor that accepts firstName and lastName.

4. Create a new ***ActorList*** class.

    4.1 It should have it in an ArrayList of actors as an instance variable. Use the Actor class you just created as part of the data type. Remember that ArrayLists have slightly different syntax than arrays.

    4.2 Create a constructor for ActorList. Instantiate your ArrayList, actors, as part of the constructor.

    4.3 Create getters and setters for your ArrayList, actors.

5. Create an ***ActorView*** class.

    5.1 ActorView should have only one method in it, printActors. It should accept an ActorList as as a parameter. Name this parameter actorList.

    This method should loop through all the actors in actorList and print out their first name and last name. You may want to use a for-each loop.

6. Create a ***Controller*** class.

The controller class is the class, which will be in control. It will be able to ask both the "Model" (i.e. the ActorList) and the View (i.e. ActorView) to update themselves. If/When additional functionality is implemented, the Controller should be able to respond to changes in either the model or the view and ask the other to update.

> 6. 1 Create instance variables for ActorView and ActorList and call them view and model respectively.
>
> 6.2 Create a constructor that accepts references for ActorView and ActorList and assigns the parameters to the instance variables.
>
> 6.3. Create a method updateView(). This method should call the printActors method of the view.

7. Come back to your original class, ***MVCDBProject***.

> 7.1 Declare a static variable actorList, which is of type ActorList.
>
> 7.2 Write a method, getActorListFromDB().
>
> Within this method, copy the code you wrote for Inclass9 to access the database and run a query. Your query today should get a list the first and last names of all actors. ***Within your while loop*** which traverses the resultSet,
>
> > Create a new instance of Actor and provide the constructor with the first name and last name that you get from the resultSet.
> >
> > Add the new instance of Actor to actorList. Use the add method! You can either create a temporary reference or you can create an anonymous object and pass it directly to the add method – either way works!
>
> 7.3 In your main method,
>
> - Use your static actorList variable and instantiate it.
> - Call getActorsFromDB, which should populate the actorList.
> - Declare and instantiate a new ActorView variable. Call it view.
> - Declare and instantiate a new Controller variable. Call it controller. Pass the constructor your two object references: view and actorList.
> - Ask your controller object to update the view!
>
> *With this technique, developers need not worry about how the actorList is going to be displayed. It will automatically be taken care of by the "View." You could even change the View and no other program logic would need to be significantly altered.*

Discussion

Even with this extensive list of classes, this MVC model is pretty basic. It is only hooked to a command line interface and cannot respond to user events. A more extensive MVC model would implement a GUI class as a view and then a Controller for that view. It could then again choose a push vs pull model. Functionality could be built in to the view to throw events (e.g. for creating a new student) to which the Controller would respond. The Controller an also force updates onto the model/view or initiate actions.

Option Enhancement

Because we are working in the console environment, the view cannot independently raise events, like edit, or button clicks for adding new actors, etc. However, do the following to implement a a slightly different model for adding new users.

<u>View class</u>

Within the view, create an AddActor() method returns a variable of type Actor. (*We would ideally want to pass a reference to an Actor as an argument but Java dislikes nulls and explicitly prohibits uninstantiated variables being passed as arguments).*

Within this method, ask the user for a first name and last name. Create a new Actor with the specified names and return the reference to the instance.

<u>Controller Class</u>

Within your Controller class, create a method called AddActor(). This method should call the AddActor() method and then take the reference to a new Actor returned by it and store it in to the actorList model. If you want to take this action to its logical consequence, this method should then create an INSERT statement and insert the new Actor into the database. *There would ideally be an updateModel method for this*. Then this method should finally update the view again, so that a final current list of actors is shown to the user. When the user is shown the list, it will be an accurate list that is reflective of the actual model – it's not just un-updated data from the GUI/View but rather an actual update reflecting the current model.