CST8288 OOP with Design Patterns

Assignment 1

Purpose

In this assignment we are going to have a better lookat a simple design with GUI which has many components.

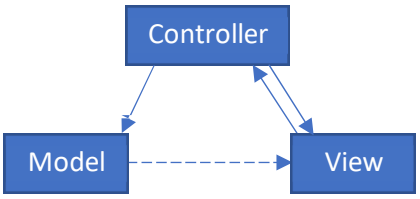
Refactoring

Using some DPs (Design Pattern), we will attempt to refactor the code based on functionality. DPs allow the code to be more organized which in turn will make it easier to upgrade and maintain applications. Sometimes using DPs will require more coding and will make the code more complicated. However, the benefits of decoupling (Layering) and code organization forced by DPs greatly improve the upgradability and maintainability of applications.

We will use two DPs called MVC (Model View Controller) and Builder. MVC will handle the separation of JavaFX (GUI/View), JDBC (Model), and Logic (Controller). Builder will allow easier implementation of Connection URL to multiple DBs (only MySQL for this assignment).

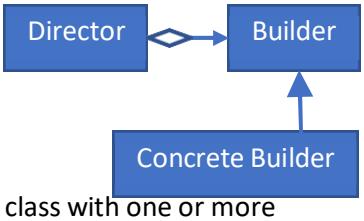
MVC

Model View Controller decouples the application in 3 components. Allowing each component to be theoretically independent of other components. There are variants of MVC which can have more than one of each components in different combinations.



- Model: Can be one class or a group of classes. It represents the underlying data and/or raw functionality. As an example, in our Assignment 1, MySQL is the model in a structural view. Model can update view. This is usually done not in a direct manner, rather in an indirect way usually though the Observer Pattern or through the controller.
- View: Represents the visual representation of the Application. for example, JavaFX, Swing, or a Browser.
- Controller: Its jobs is to connect the view to model. Controller can update both view and model. There are different variations of how controller updates/manipulates other components.

Builder



This DP is used to "separate the construction of complex object from its representation so that the same construction process can create different representations"[1]

The idea here is that the builder will be an interface or an abstract class with one or more concrete classes inheriting from it. These concrete classes are the full implantations of the Builder. For example, in this application the builder is the general JDBC builder and the concrete classes can be MySQL Builder (MySQL is the only one to be implemented), Oracle Builder, and Mongo Builder.

The Director in this case will be JDBC Controller. The user will provide an instance of MySQLBuilder to the controller, MySQLBuilder is subclass of JDBCBuilder. Then the controller will ask builder to generate a URL based on the information it possesses or it was given to by user.

The main advantage will be the fact the director relays on Builder. Meaning we can pass different concrete builders to it and it will work the same way. Allowing for a greater flexibility in the application.

Observer

This DP is a notification system. It is used when multiple objects (Observers) are interested in changes of one Object (Observable). For example, a simple button is observable while all other objects which we add to button as action listeners are observers. Observers get notified of any changes in the observable object.

Observable is usually a class which can accept Observer interface as an argument. Whenever the state of the Observable changes, Observable will notify all Observers of the change. Assume Observer interface has one method called update and Observable class has a list of observers. Each time state of Observable object changes (state depends on implementation), Observable will notify all observers by looping through all observers and calling the update method. The argument/s of update method depends on the implementation.

In JavaFX we use the interface Property to handle the implementation of observer DP for us. For example, look at the example provided the assignment called PropertyExample.java. Every Node in JavaFX has many properties. These properties are width, height, text, color, and many others. Generally to access a property you will use a method named like [property name]Property():Property. Property methods start with the name of the property and end with the word Property. Almost all features in a Node (nodes are button, TextField, TextArea, etc.) have proprieties.

[1] (Design Patterns Elements of Resuable Object-Oriented Software, 1994, p. 97)

Requirements

1. Make sure to run the SQL script attached to the assignment on your MySQL workbench first.
2. Implement JDBCController.
3. Create a sequence diagram of JDBCModel::getAllGlucoseNumbers.
4. Create a sequence diagram of JDBCController:: `loginWith`.

Submission

You will need to submit 1 zip file containing the files below:

1. Submit one zip file of your project.
   [firstName]-[lastName]-[labSection#].zip
2. Submit your diagram/s.

Export Instruction

This is a maven project, to export build your project with maven, file -> open projects from file system- navigate to your unzipped project folder , select folder then finish .
In Package Explorer, left-click on the project -> run as-> 3 Maven build then
Use maven goals ***clean install test*.**

Due Date

Sunday October 24th. However, there will be no late penalties if submitted by October 31th.

Late Penalties

-10% per day, and zero on the fifth day.

Approach

These instructions will explain how to approach your code. Below is the highlight of each step.

1. Model, this component is stand-alone a. The main purpose is to keep all the code related to DB communication.
2. Controller, this class is dependent on Model. However, it can be created in advance with empty methods. Controller will delegate complex works to model. It deals with the builder and act as abstraction between view and model.

Model

Look at the class diagram and check the signature of all methods first. This class is focused on providing functionality of manipulating the DB. Some methods are also explained in sequence diagram.

1. setCredentials Is just a setter.
2. isConnected, make sure the connection is working. You can check this by using methods connection::isClosed and connection::isValid. Do not forget your null check.
3. connectTo, if isConnected call close. Then use the DriveManager::getConnection to create a new

connection. Be.

4. hasValidConnection, simply throws and SQLException if isConnected is false.
    a. All methods that connect to db should call this method first.
5. addGlucoseValue, use QUERY_GLUCOSE_INSERT.
6. getEntryTypes, use QUERY_ENTRYTYPE_SELECT.
7. getAccountInfoFor, use QUERY_ACCOUNT_SELECT. You can use the array COL_NAMES_ACCOUNT to get the column names you need and use them in the getter methods of ResultSet.
8. updateInfo, use QUERY_ACCOUNT_UPDATE.
9. loginWith, use QUERY_VALIDATE.
10. getAllGlucoseNumbers, use QUERY_GLUCOSE_NUMBERS.
    a. Hint, you need a List of Lists. First list has the row and inside of it are the columns.
11. getColumnNames, return a list instance of COL_NAMES_GOLUCOS.
12. Close, if connection is not null call close on it.

Controller

1. This class is very simple for the majority you are just calling the equivalent method on model. Controller will delegate complex works to model. It deals with the builder and act as abstraction between view and model.

Look at the class diagram for details of all methods. This class specifically keeps track of the logged in account and a Boolean value indicating if anyone is logged in. The method findAllGlucoseNumbersForLoggedAccount should only work if login is true.

1. In constructor, initialize all the variables. To initialize an ObservableList you can use FXCollections::observableArrayList.
2. Methods setURLBuilder, setDataBase, addConnectionURLProperty, setCredentials, connect, isConnected, getColumnNamesOfGlucosEntries, updateInfo, addGlucoseValue, and close are straight forward delegation.
3. If any method is throwing a checked exception you will have to wrap in an unchecked exception and throw it out. You can use IllegalStateException. Do not do this for the close method.
4. isLoggedIn is a normal getter.
5. bindInfo, use the method bindBidirectional of property objects to bind them to your variables.
6. addTableListener, add a listener to updateTable.
7. getEntryTypes, if model is connected clear the entryType then add all the values of getEntryTypes() from model to entryType. Finally return the entryType list.
8. loginWith
    a. Use the model to get the account id and update the isLoggedIn.
    b. If logged in, get the account info form model, and update the property values, use the setValue method.
    c. Finally update the enteryTypes, use the available method, do not rewrite it.
9. The method hasValidLogin is identical to isLoggedIn. Only instead of boolean it throws exception if not logged in. It is meant to be used internally as boolean will not provide enough feedback to outside user.
10. setDataBase and addConnectionURLProperty method modify builder object.