Further Programming - Design Choice Report

By Rebecca Watson – s3903758

Assignment 2, Study Period 3, 2022

1. How did you apply MVC design pattern to build this application?

This application applies the Model-View-Controller pattern in the following ways:

All .fxml files containing all GUI code are in the **View** package of the program. They each contain the description of the user interface, which is why they are in the View.

All java class files connected to a .fxml file and declared in code as the controller of that .fxml file (fx:controller=”controller.CreateNewRecordController”) are in the **Controlle**r package of the program.

All other classes and interfaces responsible for creating, updating and deleting the domain object data are in the **Model** package of the program.

2. How does your code adhere to SOLID design principles?

Single Responsibility:

Each class/interface in the Model package is designed to have a single responsibility to fulfil – for example, one interface updates a row in the database, another interface deletes a row in the database; one interface searches for an existing record, another ‘gets’ the record/s and so on.

The User and Record object classes do not perform any calculations or methods for manipulating the data held in that object; they only provide a constructor, getter and setter methods.

Therefore, classes are designed to be modular and easily re-used for other programs.

Open-Closed Principle

Classes are open for extension, particularly in the model – meaning the classes do not have to be re-written to implement new features.

For example, the RecordCollection class implements several interfaces which are independent of each other and easily added to or removed from this collections class. The class RecordCollections is open to extension with additional interfaces, and the interfaces are closed for modification.

Several of the interfaces implemented by UserCollections or RecordCollections have only abstract methods that need implementing in the class they are used in, so new code can be added without touching the existing codebase.

Interface Segregation Principle

Each of the interfaces written in the program do not force classes to implement what they do not need.

This principle is followed to reduce any side effects that code changes might cause, and to give the application the ability to extend as updates are made to it.

All interfaces are written to be as small as possible (to also fit in with Single Responsibility).

3. What other design patterns does your code follow? Why did you choose these design patterns?"

Creational Design Pattern - Singleton

This design pattern is used for the UserCollections and RecordCollections classes, limiting the creation of these classes to only one instance.

This is done to coordinate actions across the application while a single user is logged in at a time.

The constructors of the Singleton classes are private with a synchronized getInstance() method to return the class instance.

Structural Design Pattern – Private Class Data

Access to all class attributes are restricted by setting them as private.

This is done to minimize the visibility of class attributes, preventing undesirable manipulation by other classes.

Making attributes of classes private and declaring constructors of classes (particularly in the Model package) as private or protected enhances encapsulation of the data.

Getters and Setter methods are used for these private attributes where needed.

Behavioural Design Pattern – Mediator

The UserCollections and RecordCollections classes are identified and used as mediators to benefit mutual decoupling.

One Singleton of each of these classes is instantiated and all other object classes/interfaces interact only with one of these mediators.

These two mediator classes are used to encapsulate the communication between other object classes/interfaces.

They are used as intermediary classes to decouple many other classes/interfaces in the Model.