Assignment 2: Justification of Design Patterns

Analyse and justify your design choice in building this desktop-based application by covering the following questions:

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

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

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

1. Model View Control (MVC)

   The model view control methodology is fundamentally easy to follow when using the Scene Builder application to assist.

   The utility forces several of the main components of a MVC design pattern.

   To clarify, nearly all, if not all my views, have been created using the scene builder. The scene builder insists on declaring an applicable controller to go with each view.

   I have kept strictly to this process. I have linked each controller to the relevant data model. See below.

View	Controller	Model
LoginView.fxml	LoginController.java	User.java
ProfileView.fxml	ProfileController.java	Profile.java
WorkSpaceView.fxml	WorkSpaceController.java	WorkSpaceManager.java
WorkSpaceBasketView.fxml	BasketController.java	Basket.java, WorkSpaceManager.java
WorkSpaceTaskView.fxml	TaskViewController.java	WorkSpaceManager.java, Task.java
TaskView.fxml	TaskController.java	WorkSpaceManager.java, Task.java

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

The main design patterns I have adhered to are Singleton, and Façade. I have not strictly utilized the Factory pattern. Its more of a Façade and singleton combined rather than a Factory.

Singleton Façade:- User.java

This is used as a front (Façade) to access and manage all the data relating to the User. It stores the Profile and Photo objects here too. The Profile is used to store all profile information. (Username, password, first name, last name, and default project) The Photo or profile image is also stored within User. The Photo controls the reading and writing of the profile image.

The User is static enabling it to be utilized at any time within the Workspace. It is required often, with only one User logged in at a time, which makes the Singleton methodology ideal.

It is also a Façade in that it groups the classes Profile and Photo together without requiring the programmer to have any amount of detailed knowledge of either one.

Singleton Façade:- WorkSpaceManager.java

Creating this as a singleton is appropriate as the Workspace, as for the User, it only requires a single version. The WorkSpaceManager handles exactly that, the Workspace. It is the front to the data structures of the workspace. There are three basic structures here. The Project, the Basket, and the Task. Each houses its information relating to specific data of each. The database access included.

The WorkSpaceManager accesses the database utilising the forementioned structures. It gives the programmer the ability to retrieve and set any data required without the need for any specialized knowledge. The data and methods surrounding thereof are accessible at anytime during the workspace's existence. And as for the User only one version is required.

Singleton Façade:- TabPaneController.java

This is an interesting one, it did not come about in conceptualization initially. At that point I did not envision requiring this. It came about, as an evolutionary process. Had I known better in the beginning this would have been designed first.

This controls the Tab Pane, which is essentially the Workspace itself. It is where everything is loaded replaced or deleted. It builds the tabs, the scroll panes, and the grid panes all to go with it. The programmer does not have to understand how any of this works. Just give it the column, row positions, and the pane to load and it does it for you. It maintains each Node and stores within the Node itself its current position on the board. This enables accurate manipulation of all the Panes, Tasks and Baskets.

Once again, a single version is used as it enables the same data to be accessed from multiple locations during runtime, without having to pass an unnecessary amount of data around. Only one and the same one is used.

3. I haven't really utilised the factory concept, but it has in a round about way. Not a strict engineering concept of it, but a type, nevertheless. The TabPaneController and FactoryStageLoader come as close as I was able. The FactoryStageLoader is a simple way of loading the fxml screens in one place.