1. My program has a Model, View, Controller, and a local database. I let the controller display the window to the user first. The user can act on the view. Then controller follows the action set by view and controller, call the model to change the database's data, the new data update and store in the corresponding model class, then rebuild the window shown to the user depending on the new model data.

Model: It stores data and gets data from the local database.

Views: It shows user program windows with some buttons to click, area to input and drag.

Controller: When the user acts, it updates the database, view and model.

2.

* Single responsibility Principle: The class in the View package are all fxml files, main used to set and show user windows. The class in Model package store and get data from the database. The class in the dao package only store, get and update data with the local database. The class in the controller only upload changes when the user act.
* Open-closed Principle: The most obvious point is that the change of the task will not affect the column. Same, the change of the column will not affect the project. But only influence the window shown to the user. That means that column class methods are not get polluted with details of task class.
* Liskov Substitution Principle: Both the userDao and workspaceDao method can be called in any situation, and they behave the same. They did not implement any stricter validation and did the same rules to all output parameters as applied by the parent class.
* Interface Segregation Principle: As an example of the connection between dao and model. The model does not need to use all of the dao methods, the model class will call the dao method when need through the interface.
* Dependency Inversion Principle: When the workspace controller wants to change both user and workspace data in the local database, it calls Model class and the Model will call the dao interface to make changes.

3. I use the DTO pattern because data must transfer between class and class.

As an example. The controller can encapsulate data, transfer data between itself, the Model, the View. I also use the DAO pattern, because the data must be stored by a local database. As an example, I have a user value java class in the model package, the Model class connect the user class and Data Access Object Interface UserDao. The UserDaoImpl class implement the UserDao interface, then I can call UserDao’s method to get database data.