Refactoring

# Strategy Pattern for ArrayLists

In the TaskList and ListOfCourses controllers there was code to convert the ArrayList of Courses or Tasks returned by the Database module to a String array to be displayed on the page. This code was transferred into a static class called ArrayConverter using one of two methods: convertCourses or convertTasks. These methods then convert the ArrayList they are passed into an ArrayList of ListItems and return a call to convertListItems. ConvertListItems converts the ArrayList into the needed String array by calling getName() for each of the items in the ArrayList.

The reason for this was to reduce the amount of work done in the controllers without changing the signature of around a dozen methods in the SQLDatabase and StubDatabase classes. The purpose of the controllers is to operate the page they refer to, and by creating the ArrayConverter the controllers do not violate the Single Responsibility Principle.

# Strategy Pattern for Grade Calculation

We implemented the Strategy Pattern again for the Grade Calculation. Initially it was the responsibility of the Task Detail controller to calculate the total grade and remaining weight available for the course being displayed. However, in order to follow the Single Responsibility Principle it was necessary to move this code a static class called Grader.

The structure of Grader is very simple. The controllers can simply pass either of Grader’s two methods: setGrade, and setRemainingWeight to find out the value of either item. Since this is business logic, it made sense to make it the responsibility of the business layer of the application rather than force the controllers to do this work.

# Abstract Database Layer

We implemented an interface called Database. It was written in the vaguest way possible back in iteration one and the controllers used to access the database were using staticDB. Now the controllers recognize what is returned by staticDB as a database only. This was done so that the Presentation layer does not need to depend on the details of the persistence layer. Now it knows that the database methods are available, and trusts that they will not cause the application to crash.

The reason this was done was so that the application could better follow the principles outlined in SOLID. Specifically, this refactor was done for the sake of Dependency Inversion. This will make it easier to test the program, because we are able to easily switch between the Stub database and the persistent SQLite database.