Assignment #5

Design Patterns

CS 3354.251 and 252 Spring 2017
Instructor: Jill Seaman

Due: before class **Wednesday, 4/12/2017** (upload electronic copy by 10:30am, bring paper copy of the UML diagrams to class).

We want to implement a Java class to represent a student and their scores from a class they are enrolled in. The Student class provides at least the following methods:

```
Student(String name);                  //constructor
void addAssignmentScore (double as); //add 1 assignment score
void addExamScore (double es);       //add 1 exam score
```

There is also a Roster class that stores a list of the Students in a particular class. The Roster class provides at least these methods:

```
Roster(String name, String number);   //constructor
String getCourseName();                //getter for name
addStudent (Student student);          //add 1 student
```

See the website for initial versions of the Student and Roster classes, as well as Driver to test the classes. **Do not change the Student or the Roster class except** as specified in the notes below (you will need to add some methods to make them compile with the driver).

1. We need a method to return the Student's class average, but the algorithm to compute the average must be able to be selected at runtime. It also must be possible to add new algorithms to compute the average to the program with only *minor* modifications to the Student class.

   Your task: Using a specific Design Pattern, develop a design for Student that satisfies the above requirements, then implement your design.

   Use the following two algorithms for computing the average in your implementation:
   A. The Assignment average contributes 40%, and the Exam average contributes 60% to the final class average.
   B. Use the same percentages as the first algorithm, but first if there are at least 2 Assignments scores, drop the lowest Assignment score.

Testing The driver creates a Student, adds three assignment scores and two exam scores, and computes their average first using method A then using method B, then using method A again. It calls the method setDropLowestAssign(boolean) to change between methods A and B (the class should use method A by default).

2. We want to extend our design with a class GradeTracker that tracks (stores) the current letter grade of a given Student object (>=90=A, >=80=B, etc.). Whenever the Student object is *changed*, the tracker has to be modified automatically.

   Your task: Using a specific Design Pattern, develop a design for the tracker, then implement your design.

   Testing The driver defines a tracker object to track the student and outputs the letter grade of the student. It then adds an exam score that will change the letter grade of the student, and outputs the tracker's letter grade again to show it was updated automatically.

3. We want to provide access to the averages of the students in the class, without providing access to either the ArrayList of Students or the Student objects.

   We do not want to provide a collection of the averages to the client code (we don't want to require the client to store all the values locally).

   We also want to be able to allow multiple clients to access the averages simultaneously without interference (so adding a "current element" pointer to the Roster will not suffice).

   Your task: Using a specific Design Pattern, develop a design that satisfies the above requirements for accessing the averages, then implement your design. Your design must include the provided AvgDispenser interface, which is called from the driver. (Coding suggestion: return an anonymous class, See Java-Inheritance slides 21 and 22, or return an instance of a private class.).

   Testing The driver creates two students and adds them to a Roster object. It uses an AvgDispenser, obtained from the Roster, for accessing the averages and outputting them to the screen as described above.

4. For problems 1-3, draw the **class diagram** showing how **your** classes implement the Design Pattern, and **label it with the name of the design pattern** you used.

NOTES:

- This assignment may be done with a partner.

- Use the package "assign5" for your classes and put your files in the appropriate directory structure.

- You must add the **GradeTracker** class used in the Driver. You may add other classes and interfaces as needed to implement your design patterns.

- **Allowed Changes to Student** You may add the getAverage() and the setDropLowestAssign(boolean) methods called from the driver, and a field that is used by them to support the required design pattern. You may make Student extend a subclass or implement an interface. You may add code to the existing methods.

- **Allowed Changes to Roster** You may add (only) the getDispenser method called from the Driver.

- **Allowed Changes to Driver** None. Your code must compile and run with the provided Driver (you may of course add code to your copy to do more testing).

- Follow the style guidelines from the class website. **Use javadoc comments for all of your public elements.**

Submit:

A. Please combine your *.java files into a single zip file (assign5_xxxxxx_yyyyyy.zip). The xxxxxx and yyyyyy are your TX State NetIDs (mine is js236, You may have two, one for each partner. If you are working solo, you will only have one). Submit an **electronic copy**, using the Assignments tool on the TRACS website for this class.

B. Submit the **UML diagrams** showing the design of each of the three problems (on paper, bring to class). Put the name of the design pattern you chose for each problem!!