McGill COMP 303 – Software Development
Mini Assignment 8

Due: November 30, 2018 at 23:55 on MyCourses

This mini assignment practices the Strategy and Template Method design patterns.

Part I: Strategy. Implement a generic `Table<T>` class using the Strategy design pattern. A `Table<T>` contains a list of items of type `T` that correspond to each row of a table. The columns of the table are different properties computed from the items, and defined by the user. The `Table<T>` class should be able to print itself in the console in a tabular format.

Example. For example, your table could contain items of a class `Food` with the following accessors: `getName()`, `getCarbs()`, `getFat()`, and `getProteins()`. It should be possible to add the following two columns to your table: *Name* and *Calories*. The *Name* column contains the value returned by `getName()`, and the *Calories* column is computed from the following equation: $cals = 4 \times carbs + 9 \times fat + 4 \times proteins$. After adding some `Food` instances to the table, it should print the following in the console:

```
  Name   Calories
------------------
  Banana  112 cal
     Egg   69 cal
   Bagel  286 cal
```

Requirements.

1. The `Table<T>` class should have at least the following public methods: (1) a constructor that takes as argument at least a list of columns, (2) methods to add/remove items to the table, and (3) a `print` method with no arguments to print the table in the console.

2. In the constructor, the client must indicate the header and how to compute the values for each column. There are several ways to do so. You must decide on a good design.

3. The `Table<T>` class should be able to take any type of items, and compute any value for the columns, as long as it can be computed from the properties of the items.

4. When printing the table, you must make sure that the format is correct. You will lose points if your values are not aligned correctly. You can use the length of the header as the width of each columns.

5. In addition to the `Table<T>` class and any other classes or interfaces supporting it, submit a small `Driver` class containing only a `main` method, that replicates the example above (hint: a banana has 27 g of carbs, 0 g of fat, 1 g of proteins; Egg: C: 0g, F: 5g, P: 6g; Bagel: C: 56g, F: 2g, P: 11g).

Part II: Template Method. Rewrite your code to use the Template Method design pattern instead of the Strategy. You are allowed to change the constructor, but the rest of the public interface of the `Table<T>` class should be the same. You may also change the name of the class to `TableTM<T>`, to avoid a name collision with the first part. The requirements 3 to 5 also apply to this part. For the `Driver` program, you may write a single one for both parts. The `main` method will print twice the same output, but using your two different classes.

Part III: Discussion. Write a small discussion (few hundred words, or about 1/4 to 1/2 of a page) of the advantages and disadvantages of the two designs. DO NOT discuss trivial details such as design X uses interfaces, but design Y uses abstract classes. Your discussion can be in bullet points, with one advantage/disadvantage per point. To help you, here are some questions you can answer to get you started:

- Which design offers the simplest class interface to the user? Which design requires the clients to write the most code to use it?
- How can each design be adapted in the future to add more functionalities? E.g., create new methods to add and remove columns; sort the table according to one column; etc.
- Which design is the most robust (i.e., hard to corrupt/give invalid values)? Which design is the most flexible?
- Are there disadvantages of both the Strategy and Template Method patterns compared to other possible designs?

What to hand in?

Submit your Java files for Part I and II, and a PDF including your answer for Part III.

How it will be graded?

- +10 – The solutions of Parts I and II correctly implements the Strategy and Template Method design patterns.
- +10 – The solutions of Parts I and II shows good general OO design principles.
- +5 – The solutions of Parts I and II conforms to the requirements, and are consistent with each other.
- +5 – The discussion in Part III describes relevant advantages and disadvantages of both designs.