CSE 443 Object Oriented Analysis and Design Homework 1 Report

Ahmed Semih Özmekik
171044039

November 20, 2020

Abstract

Detailed explanation of design choices with class diagrams along with the design patterns used in the homework.

1 Question 1

In this problem, the two main factors are that the user can dynamically change the methods to solve the linear system and develop other methods in the future, which leads us directly to the strategy design pattern.

Since there is no feature that requires differentiation from this pattern in our own design, we directly implemented this design pattern. And we defined an interface called **SystemSolver**. A linear system solver implementing the interface here must take input as a 2-dimensional array (i.e. matrix) with the solve method and fill the result with this input. Thus, we created a design plan that is both dynamic and suitable for the future, with a low cost.

If the customer wants to follow more than one linear equation solving strategy and has the possibility to develop a strategy in the future, the design pattern we are looking for is exactly the strategy design pattern.

We'll start by defining these design patterns to determine our design of the overall scheme.

In the design, we would have restricted the class that will implement its interface to the inputs and outputs of the function at least once. So, we gave the most primitive but powerful 2d double array structure as the input of the solve function to represent the matrix, and as the output 1d a double array gives the solution of the matrix respectively in the matrix.

As stated in the homework report, the user interface was entirely left to us. That's why we set a strict but simple and naive rule. Entries will be performed by file read, and an input file as an example:

```
3, 4, 10
2, 8, 5
```

We think this input interface is a user-friendly format for many reasons. The reason we read from the file is because getting a long linear system from the command line would be both a long and dangerous process. Because when an incorrect input is given it makes the

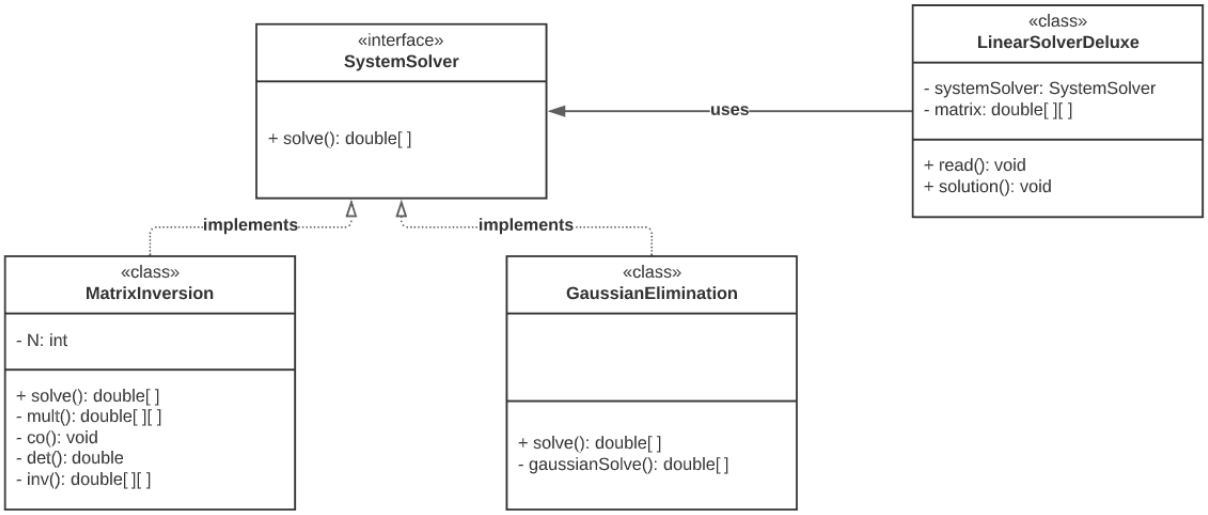

Figure 1: Question 1 Design Class Diagram

correction process difficult, at best the input can be corrected online, but in any case this interface has problems. Instead, we chose to read from the file for an interface where the user can have control at any time and does not have to return to the beginning and return every input when making a mistake.

With this design choice, we chose to separate the coefficients by commas to represent the linear system. For example, the entry above represents:

$$3x + 4y = 18$$
$$2x + 8y = 5$$

As known as:

$$\begin{bmatrix} 3 & 4 & 18 \\ 2 & 8 & 5 \end{bmatrix}$$

Since we almost derive this notation from matrix notation in linear algebra, it will be familiar to the user, and he/she will like to give input in such matrix form.

This interface selection in the design will provide convenience both for the user and the programmer in the parse process (by saving time for the programmer).

2 Question 2

In our design, we first followed a classical observer design pattern and tried to modify the pattern for more than one subject. Because the website we need to create should be responsible for reporting changes in multiple and different content areas. So we need more than a single subject that alerts observers from a single source, that is, a typical observer design pattern implementation.

As we focus on this topic, there are certain things to consider: First, a subscriber can subscribe to multiple content, either one or a combination of them. So we need a design that allows subject combinations. Second, and most importantly, this design needs to be a low-cost design pattern for the future, because different content is likely to come in the future. So we have to concentrate our design in this direction. Hence, when a new content subject comes up, we need to add that subject to our software at the lowest cost.

In this context, our **WebsiteData** class matches subjects with a **subjects** map object. And whichever subscriber (observer) wants to subscribe to which content, they can subscribe to the content they want with a determined string key. When the observer provides both itself and the literal key of the content it wants to register to the **registerObserver()** method, the job is done.

A simple interface has been provided to the programmer. Our main advantage is that when it comes to a new content subject, all we have to do is add it to the Map (in one line operation).

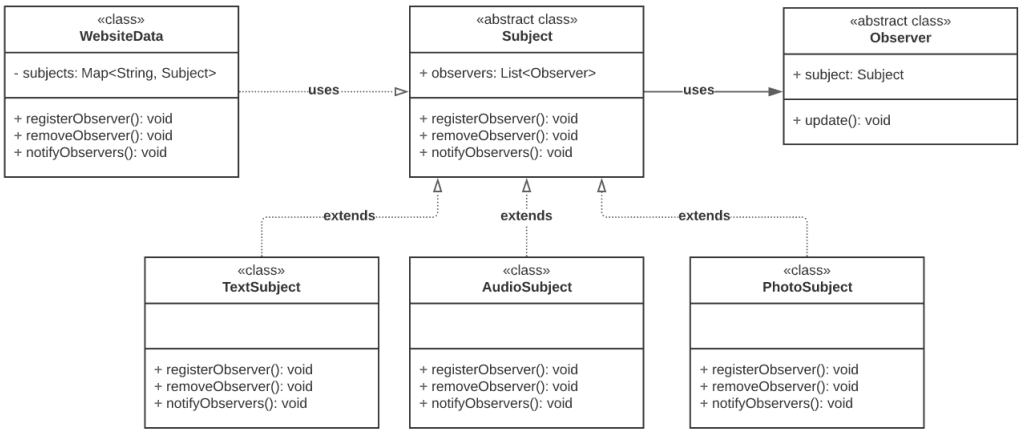

Figure 2: Question 2 Design Class Diagram

3 Question 3

We need a design in which we have to put different accessories (and accessories in different combinations) on our 3 basic suite bases and do this dynamically. At this point, the decorator design pattern comes to mind first. Thus, different accessories (ie decorators)

can be combined at the runtime and equipped in the base suite. The general scheme of our design will be shaped around the decorator design pattern method, and it will gain flexibility.

By creating one Suit interface, we defined 3 concrete Suit classes. Then, we defined 4 different decorator classes by implementing the Decorator design pattern.

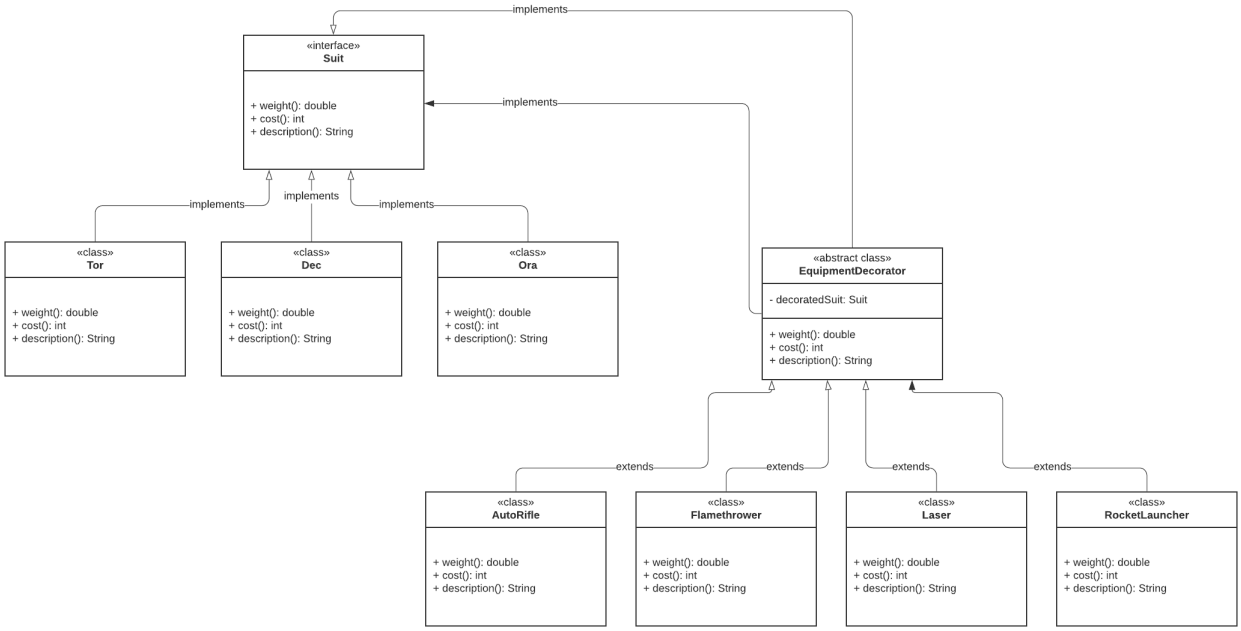

Figure 3: Question 3 Design Class Diagram