OOA&D FIRST HOMEWORK REPORT

Yusuf Patoglu - 151044027

Part1:

In Part 1 it's mentioned that class behavior or its algorithm can be changed at run time so it's obvious that we'll need Strategy Pattern here. Of course this is not the main reason why we chose this pattern. Since we have different algorithms for solving the same task (linear system) strategy pattern is the best choice here.

It will have some advantages like run time algorithm switching and cleaner code since we'll implement our algorithms in different classes.

Linear_System Class:

This is like a wrapper class. It takes the linear system and equations from the user and formats it. It also have functionality to show the system and equations in human readable format.

Matrix Class:

This class has helper functions for basic matrix operations.

Context Class:

Context class has a generic solve function which is for run time.

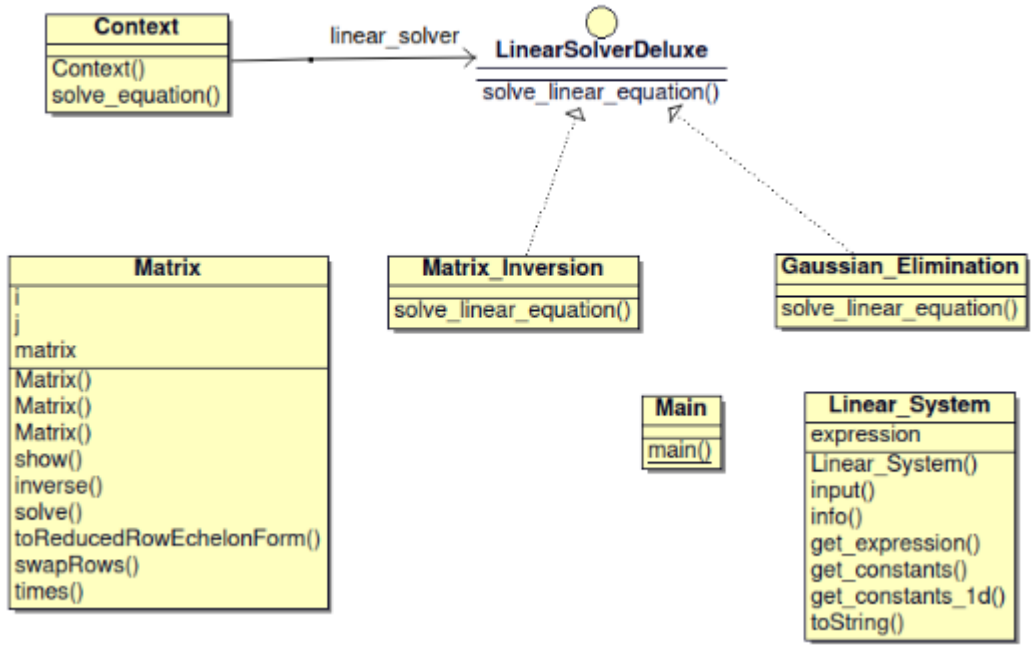
Gaussian_Elimination Class:

In this class Gaussian elimination method is called.

Matrix_Inversion Class:

In this class Matrix Inversion method is called.

Class Diagram:



Part2:

In this project we'll have many users so this must be a one to many relationship like design patttern. Since there are some dependency (subscriber-publisher) and The Observer pattern says that we can add a subscription system so the publisher class and also individual objects can subscribe to or unsubscribe from a stream of events coming from that publisher. So it's best choice to use Observer Pattern Here.

Contents Class:

This class has the simple enumerated types for PHOTO, TEXT and AUDIO.

Observer_Photo Class & Observer_Text Class & Observer_Video Class:

Perform some actions in response to notifications issued by the publisher.

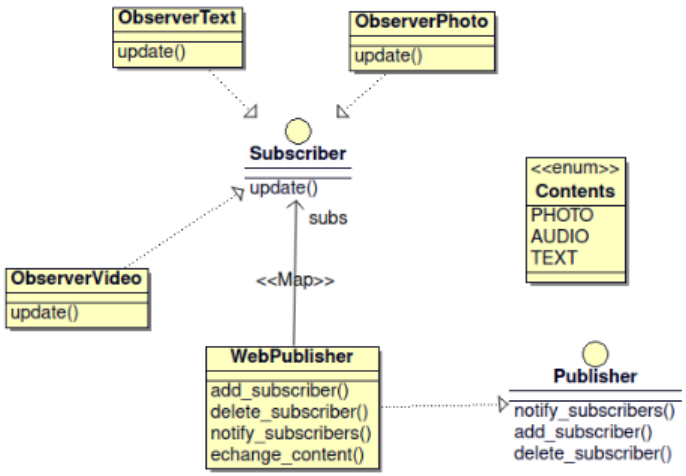
Publisher Class:

The Publisher issues events of interest to other objects. These events occur when the publisher changes its state or executes some behaviors.

Subscriber Class:

The Subscriber interface declares the notification interface.It has single update method

Class Diagram:



Part3:

In this part we could inherit accessories from the armors but if we've done it that way we might experience "class explosion" so we need to find better way for this. Since Decorator pattern can be used to decorate the task of a certain object at run-time and independently of other instances of the same class we can use this pattern.

Armor Class:

It defines an interface to calculate weight and total cost.

Auto Rifle / Flamethrower / Laser Classes:

It implements the operations defined in Armor interface

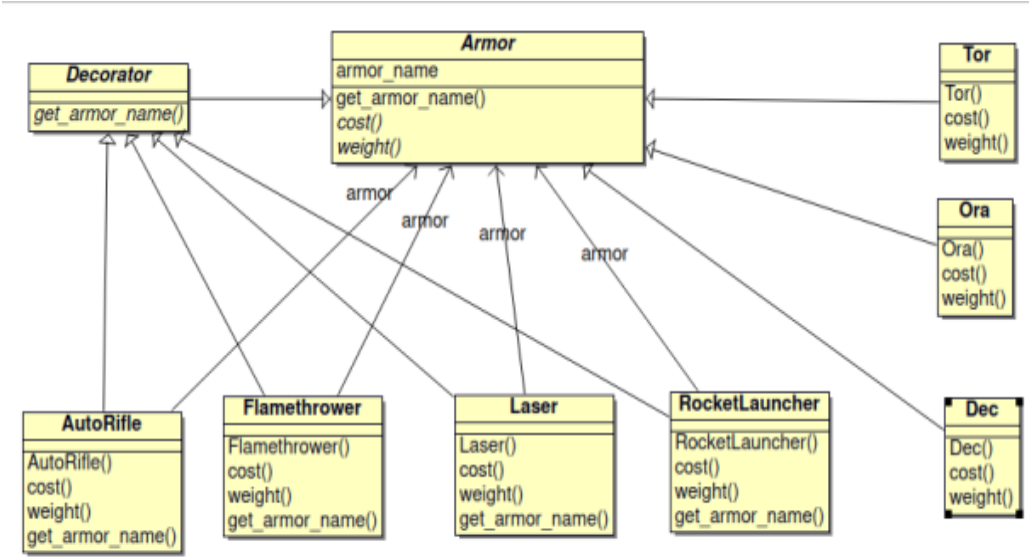
Dec / Tor / Ora Classes:

Classes which extends Armor class.

Decorator Class:

Abstract class that extends the Armor interface

Class Diagram:



P.S. -> For Part1, if you get NaN result, it means that matrix has no valid solution set.

Sources:

- https://rosettacode.org/wiki/Gauss-Jordan_matrix_inversion
- https://refactoring.guru/design-patterns/observer