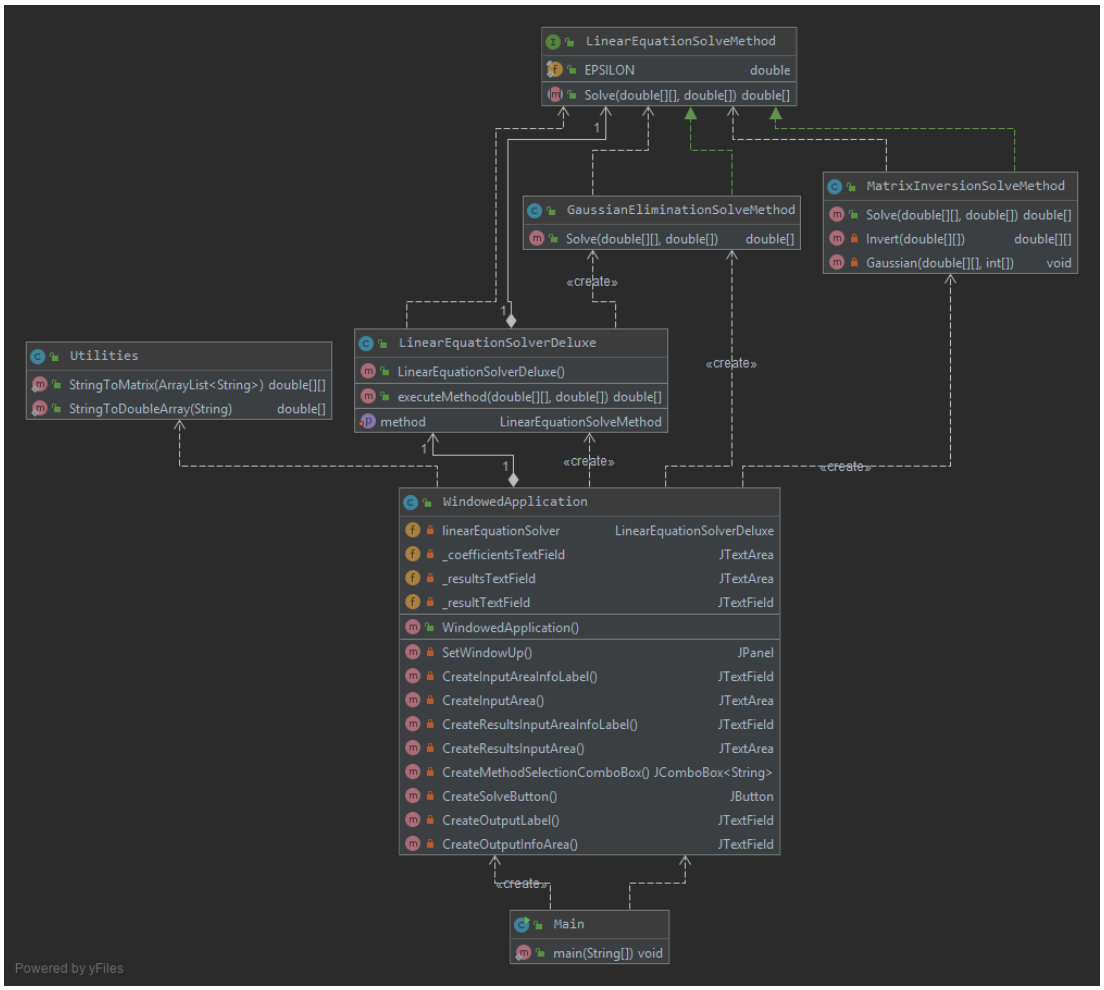
CSE443 – Object Oriented and Design – Homework 1

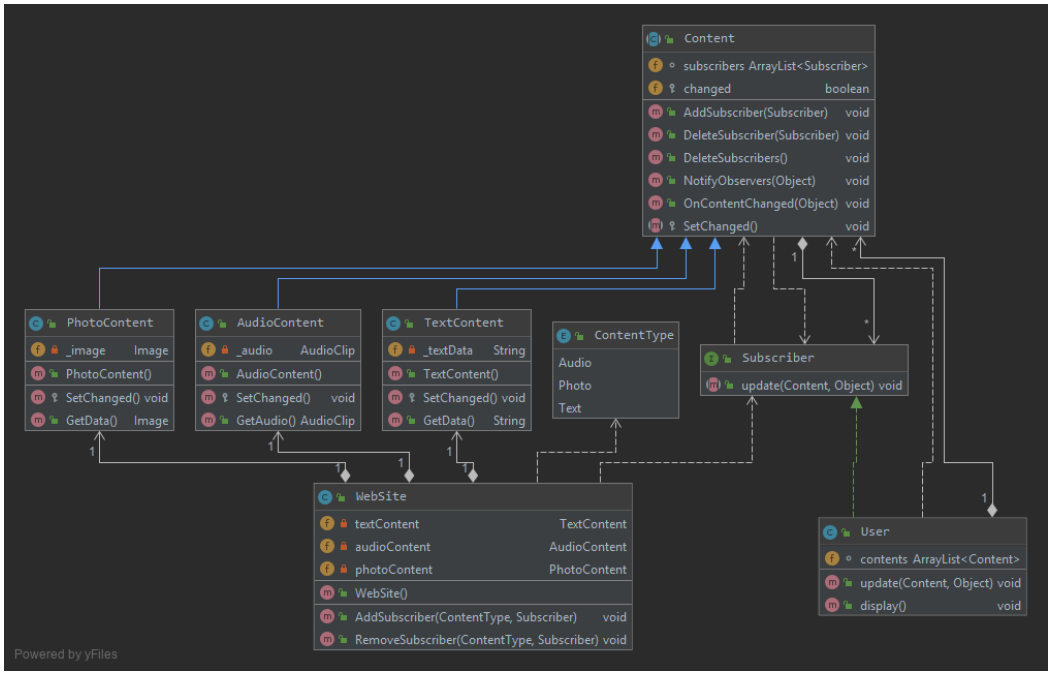
161044042 – Yasir Nacak

Q1.

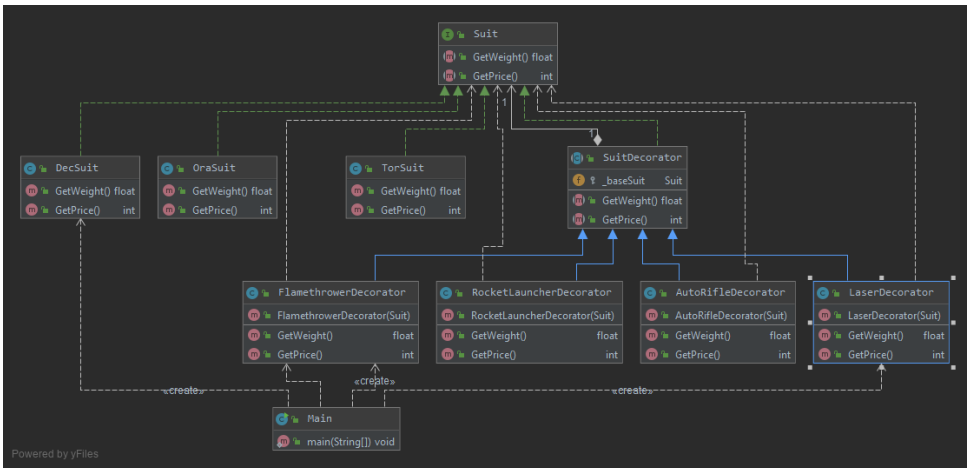


In this problem, I used Strategy Pattern. Using this pattern enabled me to switch between several types of linear equation solving methods. All solving methods implement the Linear Equation Solver Method class. This class provides the base method that every linear equation solving subclass should have. I then created two classes that implement this interface. One of these uses the Gaussian Elimination method and the other uses Matrix Inversion method. Linear Solver Deluxe contains an instance of a solving method. This method can be dynamically changed throughout the lifetime of my program. I also created a windowed application that provides the user a GUI to use this linear equation solver.

Q2.



In this part, I utilized the Observer pattern. This pattern enables subscribers to keep track of changes in the content. I made it so every type of content derives from a base Content class and has the capability of notifying subscribers this class has. I then created a User class that has a method called update. This method gets called whenever a content update itself. The user gets notified this way. Finally, I created a Website class that has all the possible content types my program has. It also has the ability to add/remove subscribers to/from specific types of contents.

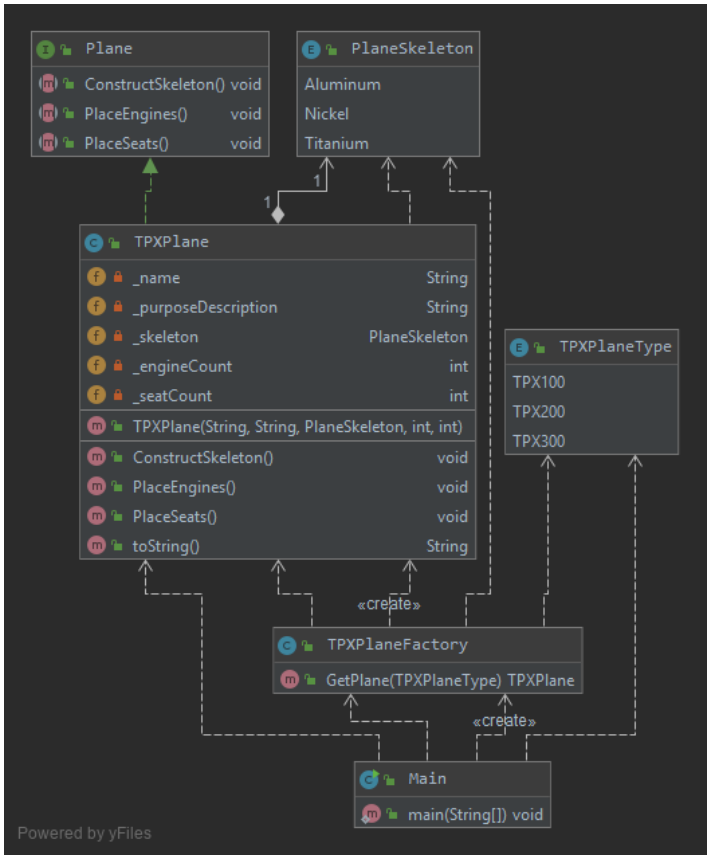


Q3.

As the types of suits (“Dec” “Ora” “Tor”) suggest, I used the Decorator pattern to implement this part. I first implemented a base interface for all types of suits and their decorators. This interface has getter methods for weight and price. After that, I created all three types of suits.

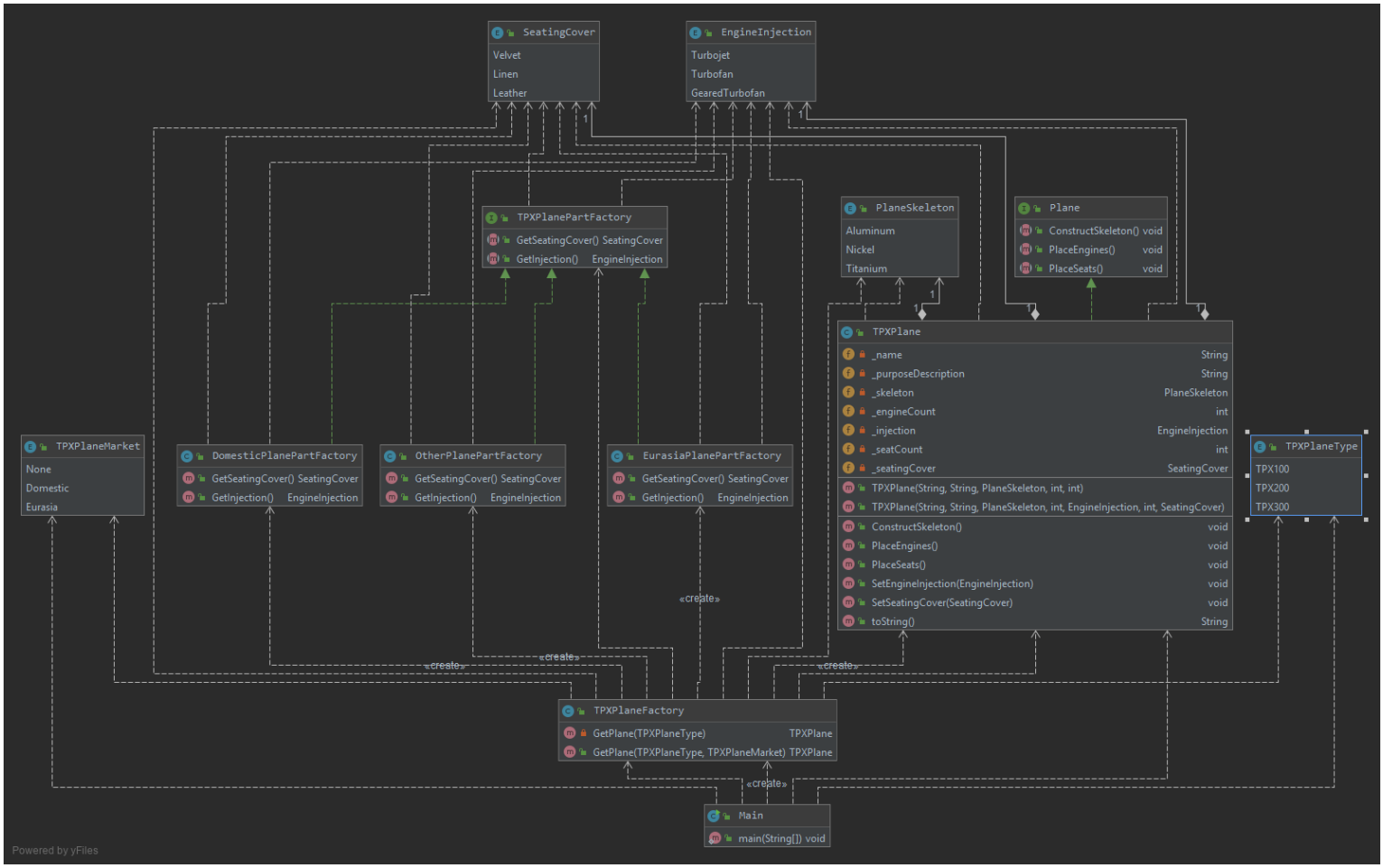
These suits implemented getter methods with values returning their weights and prices. Then I created a base suit decorator that both has and is a suit. The suit that is base decorator has is the one that it decorates. Since this base class also is a suit, it can decorate both suits and other decorators. And finally, I implemented all four types of suit decorators.

Q4 – Part 1.



For this part, I utilized the Factory pattern. I used this pattern to create all types of TPX Planes. I did not create multiple classes for multiple types of TPX Planes because other than their property values, they are the same. Both property type and name wise and also functionality. My factory class takes a TPX Plane model for input and outputs an instance of a TPX Plane with given model name if such exists. And also, since TPX Plane is a specific type of plane, I created an interface for more general type of planes which can also extend the functionality and gives a better understanding of the program.

Q4 – Part 2.



In the final problem, I used the Abstract Factory pattern for creating seating covers and engine injections of planes based on their market. I created an interface for instancing seating covers and engine injections. This interface got implemented by three concrete classes. Each of these concrete classes produced seating covers and engine injections for Domestic, Eurasia and Other market preferences. I also extended my TPX Plane Factory by giving it a method for producing a plane with a market specification. This method creates an instance of one of the three abstract factories and creates seating covers and engine injections from that factory. I did not create interfaces / concrete classes for seating covers and engine injections because in the given specification, there was not functionality need for them.