

Part- 1

Strategy Pattern

We have a job to be done and we need to do it in different ways. In such cases, instead of doing the current job with the if-else blocks by continuously refactoring the related class, adding a new class allows us to do the relevant work in the related class if what we desired. Thus, we will improve our system without making any changes to our existing class.

When we look at the diagram, we can see that there is a relationship between linearContext and linearStrategy type of aggregation (life cycles of related objects are different from each other).

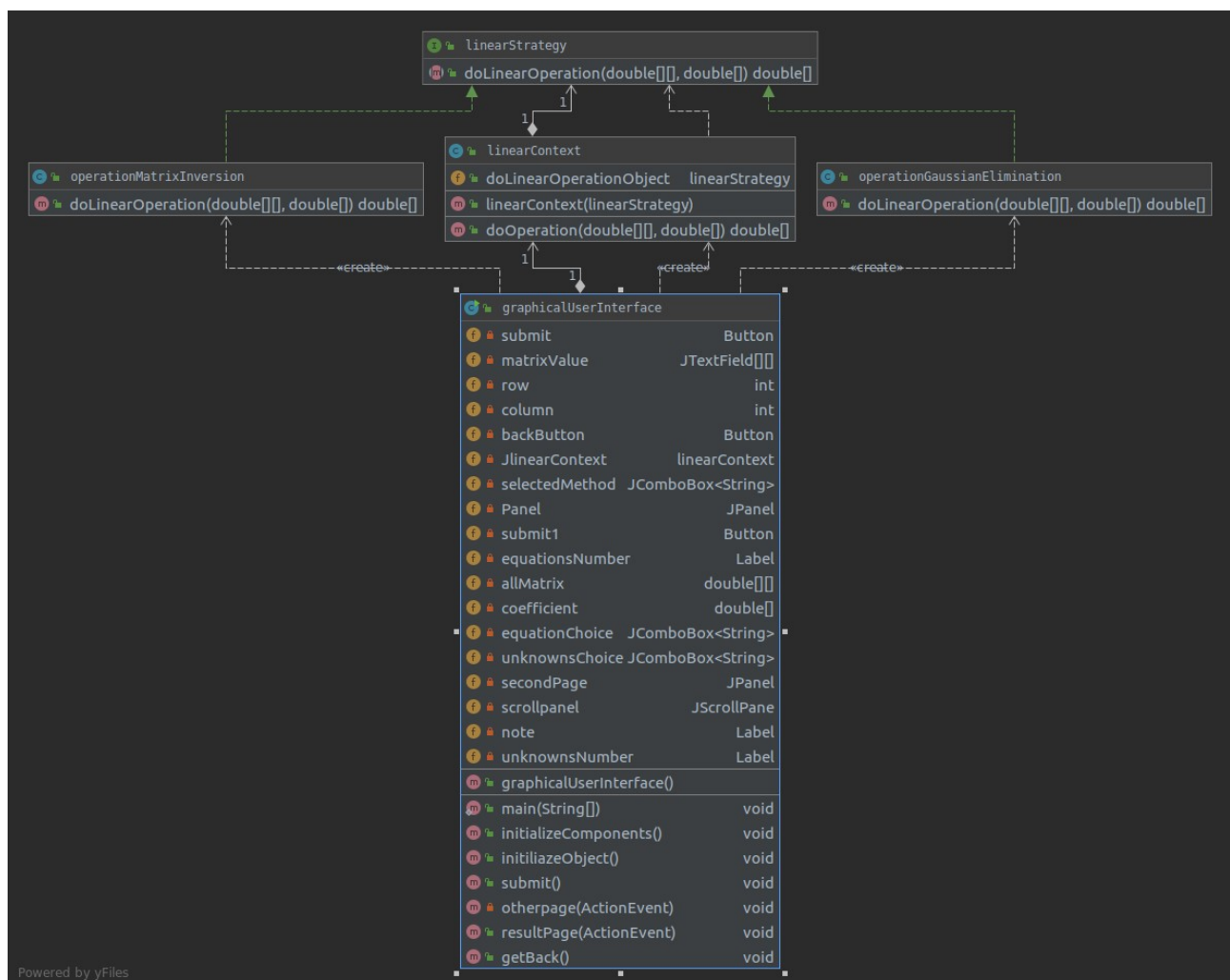
LinearStrategy

By designing an interface, we collect all our common algorithms here.

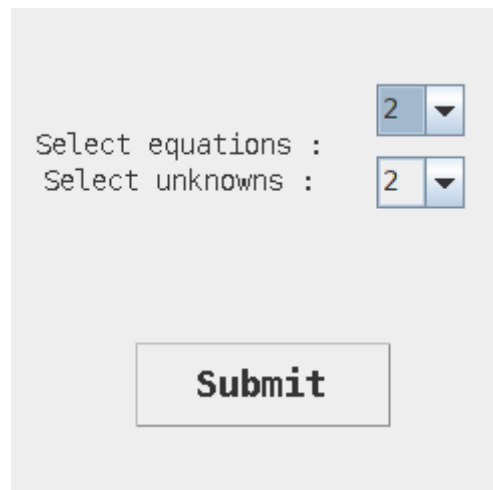
operation(GaussElimination and MatrixInversion)

Our actual class that implements the corresponding algorithm.

In our example, consider the mathematical operations. In this system we can be working with multiple methods. We can do by matrix inversion, gauss elimination .



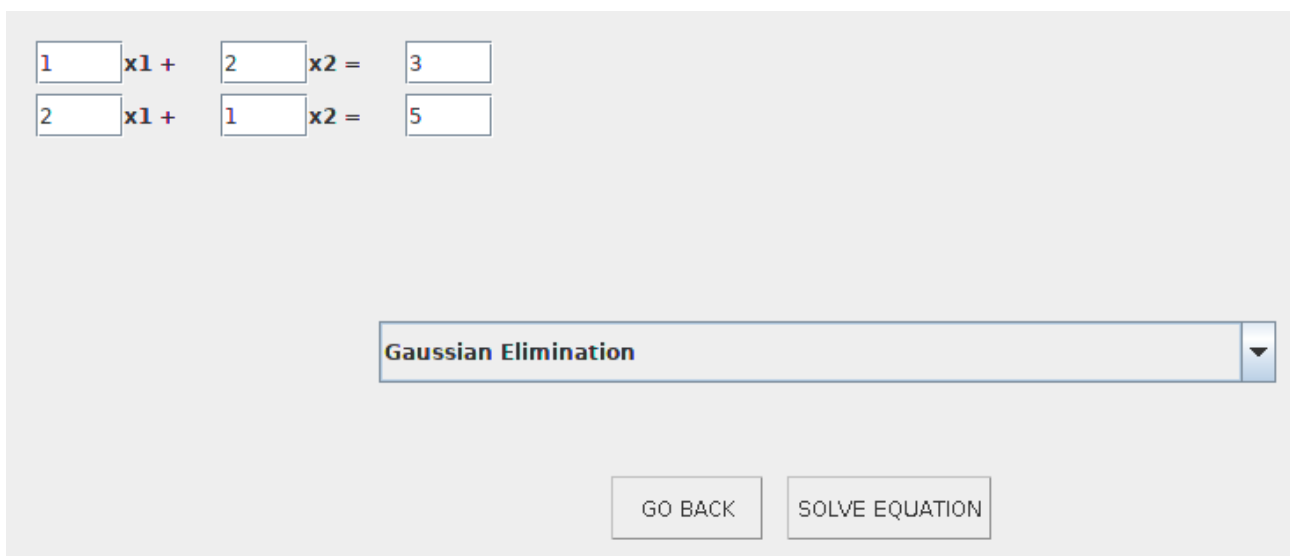
Firstly ,we will select number of unknowns and equation :



Select equations : ▼
Select unknowns : ▼

Submit

Then we will write coefficients of equation and select a operations.



x1 + x2 =
 x1 + x2 =

Gaussian Elimination ▼

Result :

X1 = 2.3333333333333335
X2 = 0.3333333333333333

Part - 2

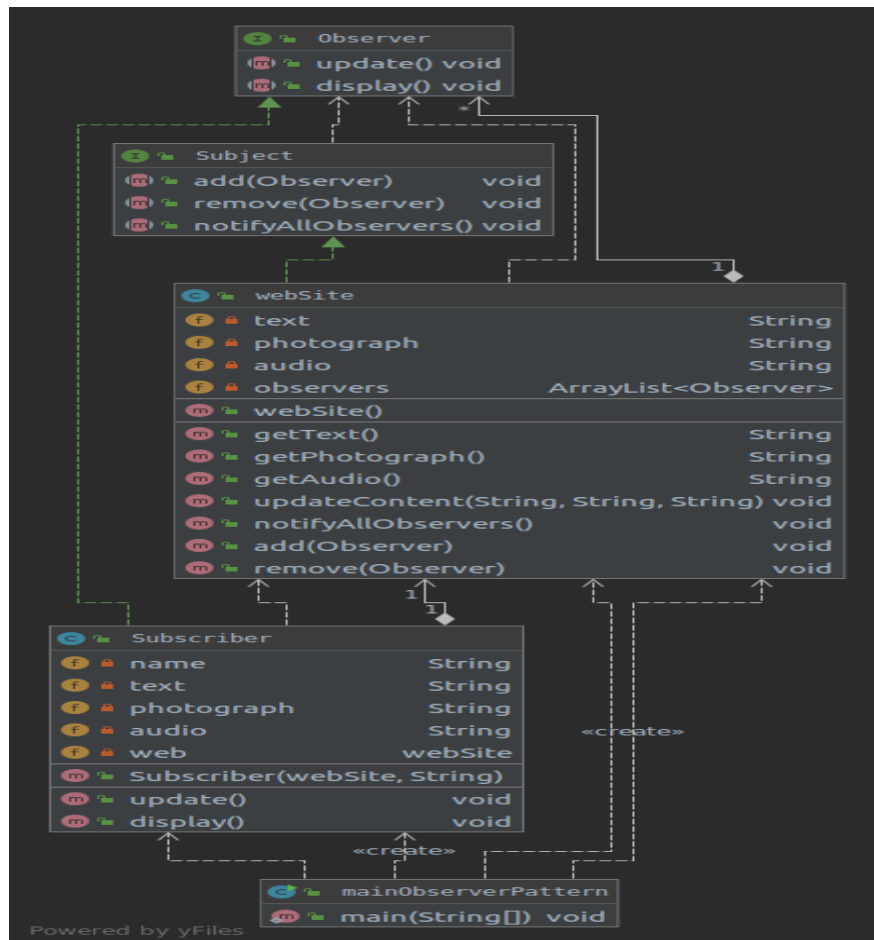
Observer Pattern

We need observer pattern. Because we have to notify subscriber when content is changed. We need a observer list in order to notify.

Solve Problem

In order to implement the Observer design pattern for this problem, there will be 2 interfaces which Subscriber and User were created. The subscriber interface includes the update () method. Classes wishing to subscribe to the website must implement this interface. The user interface has been

created to provide external API. This interface has a method called display (). Observer subscribe to the website and wish to demonstrate using the information from the website must implement these two interfaces.



```

public static void main(String[] args) {

    webSite webSite = new webSite();

    Subscriber trump= new Subscriber(webSite, name: "trump");
    Subscriber hamza = new Subscriber(webSite, name: "hamza");
    Subscriber putin= new Subscriber(webSite, name: "putin");
    Subscriber rte = new Subscriber(webSite, name: "rte");

    String text = "new episode Lacasa de Papel";
    String photograph = "photographs of bank sketches";
    String audio = "Bella ciao";

    webSite.updateContent(text, photograph, audio);

    webSite.remove(hamza);

    text = "new episode The Walking Dead";
    photograph = "photographs of zombies";
    audio = "Sound of Rick";

    webSite.updateContent(text, photograph, audio);
}

```

All subscriber are added :

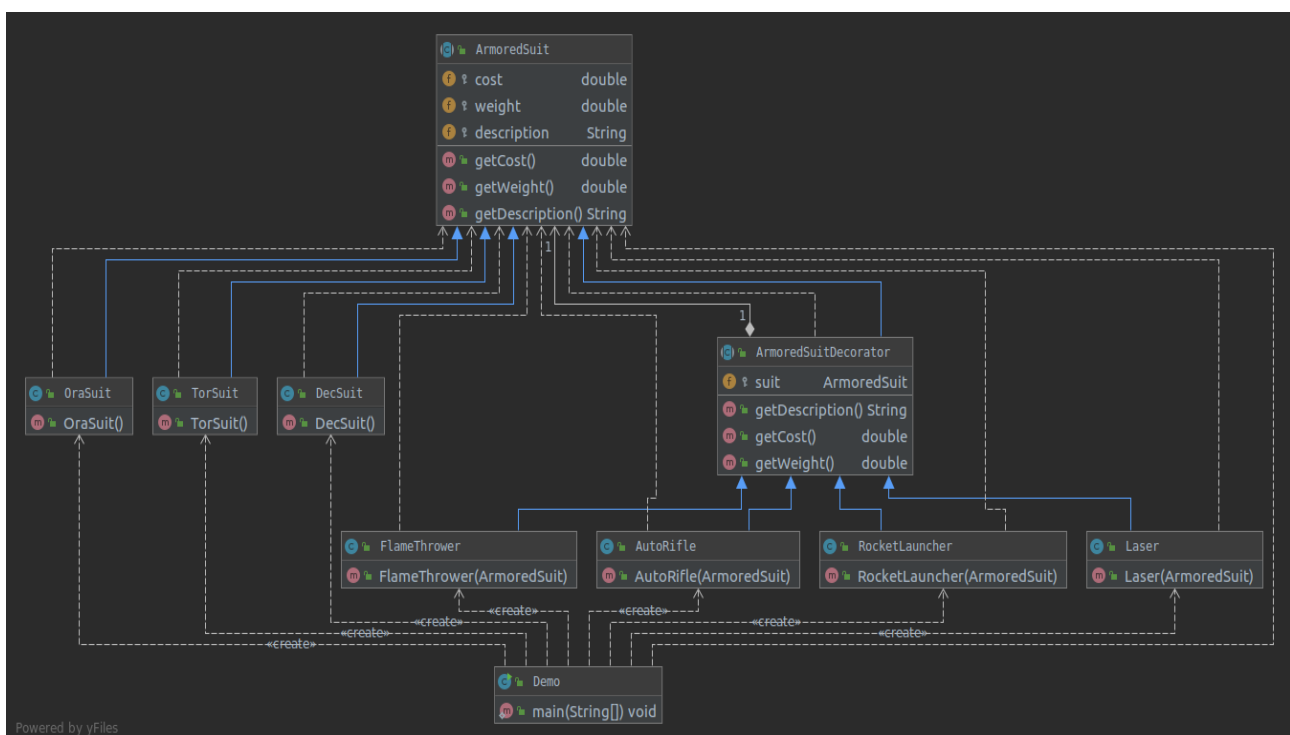
```
-----
Notify : trump , check your update.
Text : new episode Lacasa de Papel
Audio : Bella ciao
Photograph : photographs of bank sketches
-----
Notify : hamza , check your update.
Text : new episode Lacasa de Papel
Audio : Bella ciao
Photograph : photographs of bank sketches
-----
Notify : putin , check your update.
Text : new episode Lacasa de Papel
Audio : Bella ciao
Photograph : photographs of bank sketches
-----
Notify : rte , check your update.
Text : new episode Lacasa de Papel
Audio : Bella ciao
Photograph : photographs of bank sketches
-----
```

Hamza is removed :

```
-----
Notify : trump , check your update.
Text : new episode The Walking Dead
Audio : Sound of Rick
Photograph : photographs of zombies
-----
Notify : putin , check your update.
Text : new episode The Walking Dead
Audio : Sound of Rick
Photograph : photographs of zombies
-----
Notify : rte , check your update.
Text : new episode The Walking Dead
Audio : Sound of Rick
Photograph : photographs of zombies
-----
```

Part-3 Decorator Pattern

- We need decorator pattern. Because, We can set suit object and integrate with component that flamethrower , autorifle, rocketlauncher and laser. So , decorator allows behavior in order to added to any dec, ora and tor.
- Once the program is running and then ,Firstly i selected any dec,ora or tor. Secondly, we are adding component of suits. ArmoredDecorator class keeps reference everytime each component or suit.
- So, i got following class diagram :



I added different component to all suits :

```
ArmoredSuit suit = null;
suit = new DecSuit();
suit = new FlameThrower(suit);
suit = new AutoRifle(suit);
suit = new AutoRifle(suit);
suit = new RocketLauncher(suit);
System.out.println();
System.out.println("Cost and Weight : " + suit.getDescription() + " = "
    + suit.getCost() + "k TL , " + suit.getWeight() + "kg\n");

suit = null;
suit = new OraSuit();
suit = new RocketLauncher(suit);
suit = new Laser(suit);
suit = new RocketLauncher(suit);
System.out.println("Cost and Weight : " + suit.getDescription() + " = "
    + suit.getCost() + "k TL , " + suit.getWeight() + "kg");
suit = null;

suit = new TorSuit();
suit = new AutoRifle(suit);
suit = new AutoRifle(suit);
suit = new RocketLauncher(suit);
System.out.println();
System.out.println("Cost and Weight : " + suit.getDescription() + " = "
    + suit.getCost() + "k TL , " + suit.getWeight() + "kg");
suit = null;
```

Total weight and price of suits , components are calculated.

```
Cost and Weight :Suit : Dec -> + FlameThrower + AutoRifle + AutoRifle + RocketLauncher = 760.0k TL , 37.5kg
Cost and Weight :Suit : Ora -> + RocketLauncher + Laser + RocketLauncher = 2000.0k TL , 50.5kg
Cost and Weight :Suit : Tor -> + AutoRifle + AutoRifle + RocketLauncher = 5210.0k TL , 60.5kg
```

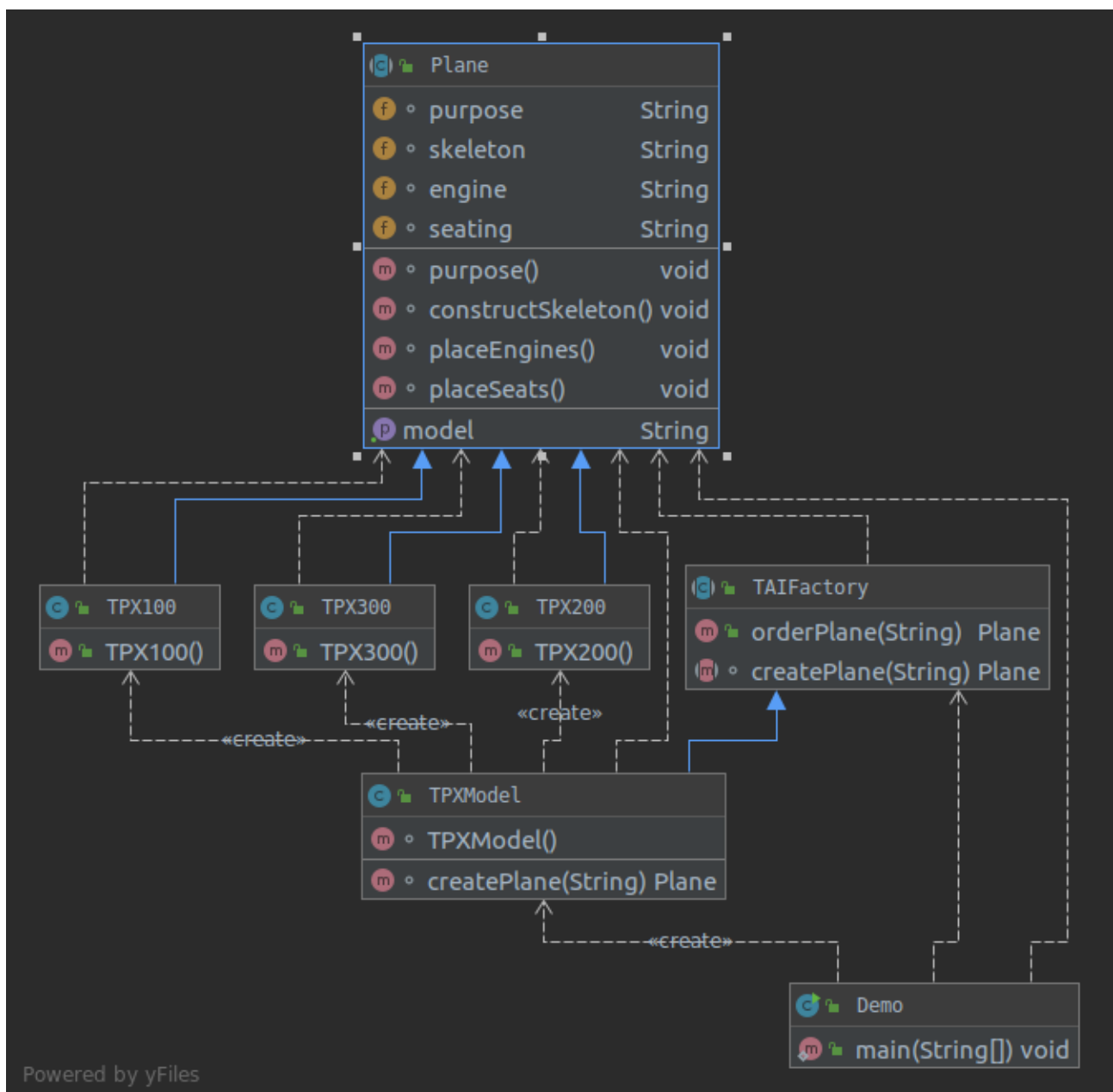
Part – 4 - 1

Factory Pattern

I implemented the orderPlane function in the TAIFactory Abstract class. This function sends the incoming String parameter that is model type of plane to createPlane to get the correct Plane object. Then constructSkeleton (), placeEngines (), and placeSeats () methods are called for the generating plane. Abstract Plane is written for the planes and all necessary operations were performed in the abstract class. The subclass TPX 100,200 and 300 classes are assigned the necessary variables only in the constructors.

Test :

```
public static void main(String[] args) {  
    TAIFactory tpx = new TPXModel();  
    Plane plane = tpx.orderPlane( typeOfModel: "TPX100");  
    plane = tpx.orderPlane( typeOfModel: "TPX200");  
    plane = tpx.orderPlane( typeOfModel: "TPX300");  
}
```



Production of TPX planes :

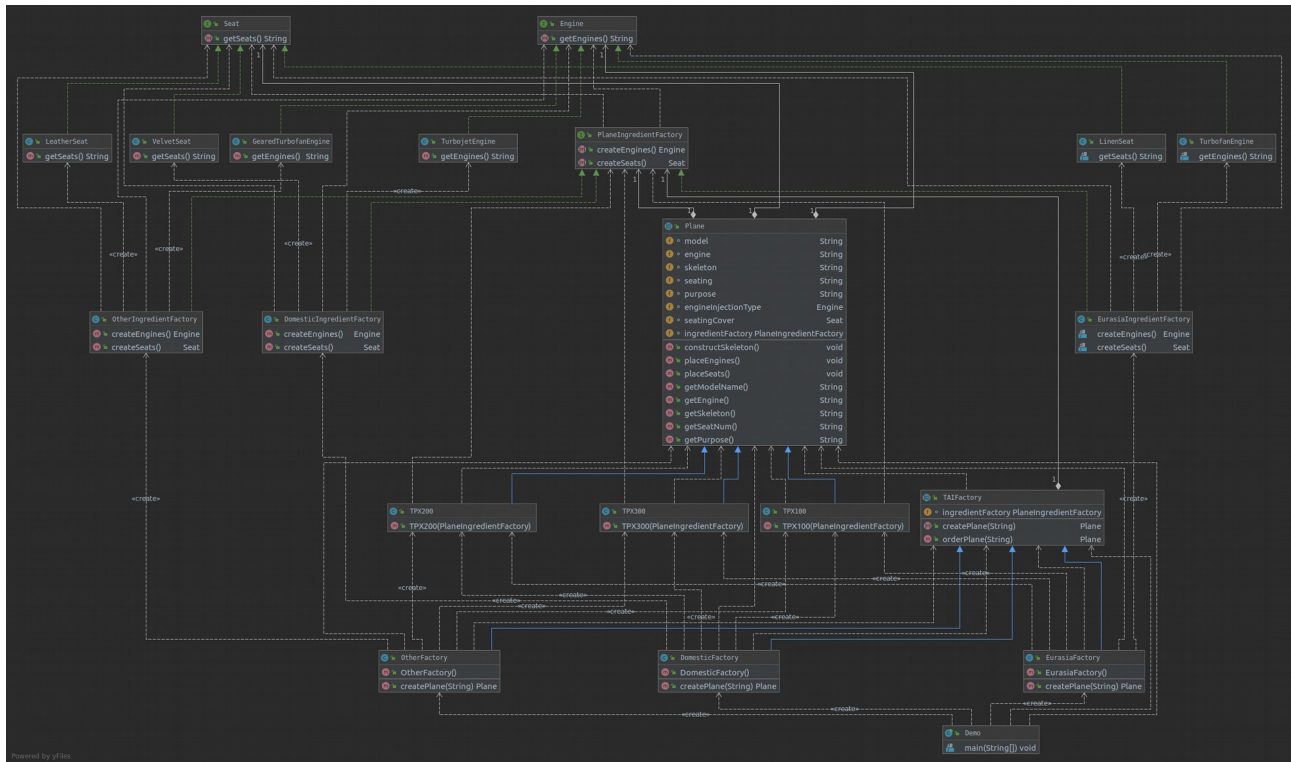
```
---- Plane TPX 100 ----  
Purpose = Domestic flights  
Skeleton = Aluminum alloy  
Engine = Single jet engine  
Seating = 50 seats  
  
---- Plane TPX 200 ----  
Purpose = Domestic and short international flights  
Skeleton = Nickel alloy  
Engine = Twin jet engines  
Seating = 100 seats  
  
---- Plane TPX 300 ----  
Purpose = Transatlantic flights  
Skeleton = Titanium alloy  
Engine = Quadro jet engines  
Seating = 250 seats
```

Part – 4 - 2

Abstract Factory Pattern

I designed and implemented the Abstract Factory Design Pattern which is similar to the Factory Design Pattern. Abstract Factory Design Pattern is one of the creational design patterns. This design pattern provides an interface for creation, without specifying concrete classes of related or dependent objects. This pattern is also called the factories of the factory. It is necessary to create a factory class for each Product subclasses. The factory classes to be created must derive from a super factory class, which is of type interface or abstract.

Ingredient Factory parameter has been added to the constructors since Plane and its subclasses in the previous part will have material changes according to Ingredient Factory and local materials are changed according to this parameter. The material factory will change according to Interface Ingredient was written and Domestic, Eurasia and Other IngredientFactory classes were written and implemented. These classes with the createEngine, createSeat and createSkeleton functions, it will produce different materials according to a factory and send the materials to their factory. Therefore, the TAIFactory abstract class was written and subclasses were created for 3 factory. Subclasses simply assign the object from the parent class by creating the required PlaneIngredientFactory object in their constructors. The createPlane and orderPlane methods are written in the abstract class. I wrote an interface and subclasses for Engine and Seat types.



I combined all component and plane of factory.

```
public static void main(String[] args) {
    TAIFactory store = new DomesticFactory();
    Plane plane = store.orderPlane( model: "TPX100");
    plane = store.orderPlane( model: "TPX200");
    plane = store.orderPlane( model: "TPX300");

    store = new EurasiaFactory();
    plane = store.orderPlane( model: "TPX100");
    plane = store.orderPlane( model: "TPX200");
    plane = store.orderPlane( model: "TPX300");

    store = new OtherFactory();
    plane = store.orderPlane( model: "TPX100");
    plane = store.orderPlane( model: "TPX200");
    plane = store.orderPlane( model: "TPX300");
}
```

All models are customized according to local needs :

---- Plane TPX 100 ----
Purpose : Domestic flights
Skeleton : Aluminum alloy
Engine : Single jet engine and Turbojet
Seats : 50 Velvet

---- Plane TPX 200 ----
Purpose : Domestic and short international flights
Skeleton : Nickel alloy
Engine : Twin jet engines and Turbojet
Seats : 100 Velvet

---- Plane TPX 300 ----
Purpose : Transatlantic flights
Skeleton : Titanium alloy
Engine : Quadro jet engines and Turbojet
Seats : 250 Velvet

---- Plane TPX 100 ----
Purpose : Domestic flights
Skeleton : Aluminum alloy
Engine : Single jet engine and Turbofan
Seats : 50 Linen

---- Plane TPX 200 ----
Purpose : Domestic and short international flights
Skeleton : Nickel alloy
Engine : Twin jet engines and Turbofan
Seats : 100 Linen

---- Plane TPX 300 ----
Purpose : Transatlantic flights
Skeleton : Titanium alloy
Engine : Quadro jet engines and Turbofan
Seats : 250 Linen

---- Plane TPX 100 ----
Purpose : Domestic flights
Skeleton : Aluminum alloy
Engine : Single jet engine and Geared turbofan
Seats : 50 Leather

---- Plane TPX 200 ----
Purpose : Domestic and short international flights
Skeleton : Nickel alloy
Engine : Twin jet engines and Geared turbofan
Seats : 100 Leather

---- Plane TPX 300 ----
Purpose : Transatlantic flights
Skeleton : Titanium alloy
Engine : Quadro jet engines and Geared turbofan
Seats : 250 Leather