CSE 443
OBJECT ORIENTED ANALYSIS AND DESIGN

HOMEWORK 01

REPORT

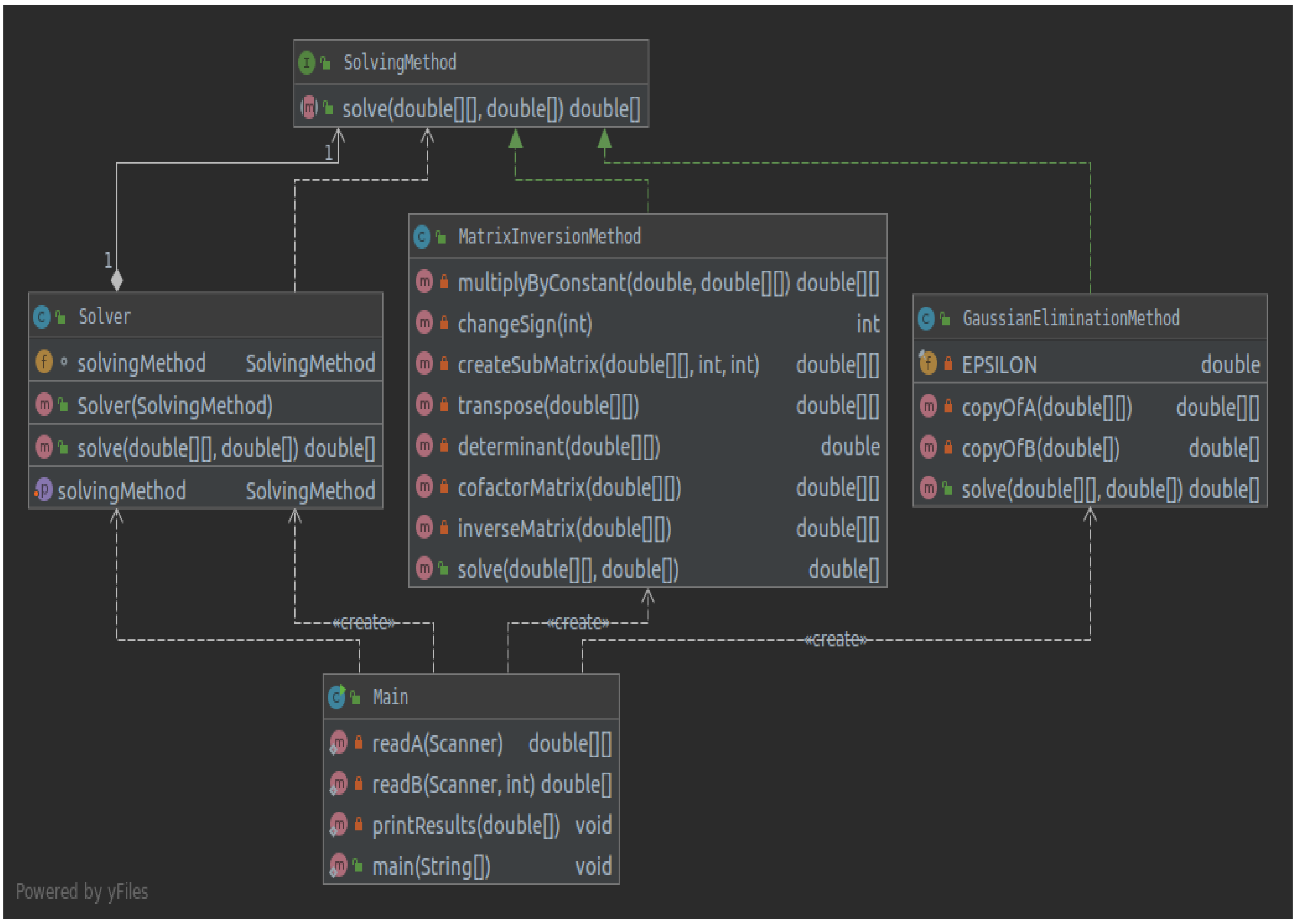
ÖMER ÇEVİK
161044004

1. <u>Part 1</u>

In that part, I used ‘Strategy’ design pattern. I have a Solver concrete class and it has composition relation between SolvingMethod interface. That SolvingMethod interface has a solve() method and there are two types solving methods for this part: Gaussian Elimination and Matrix Inversion. So that means we have GaussianElimination and MatrixInversion concrete classes which implements SolvingMethod interface.

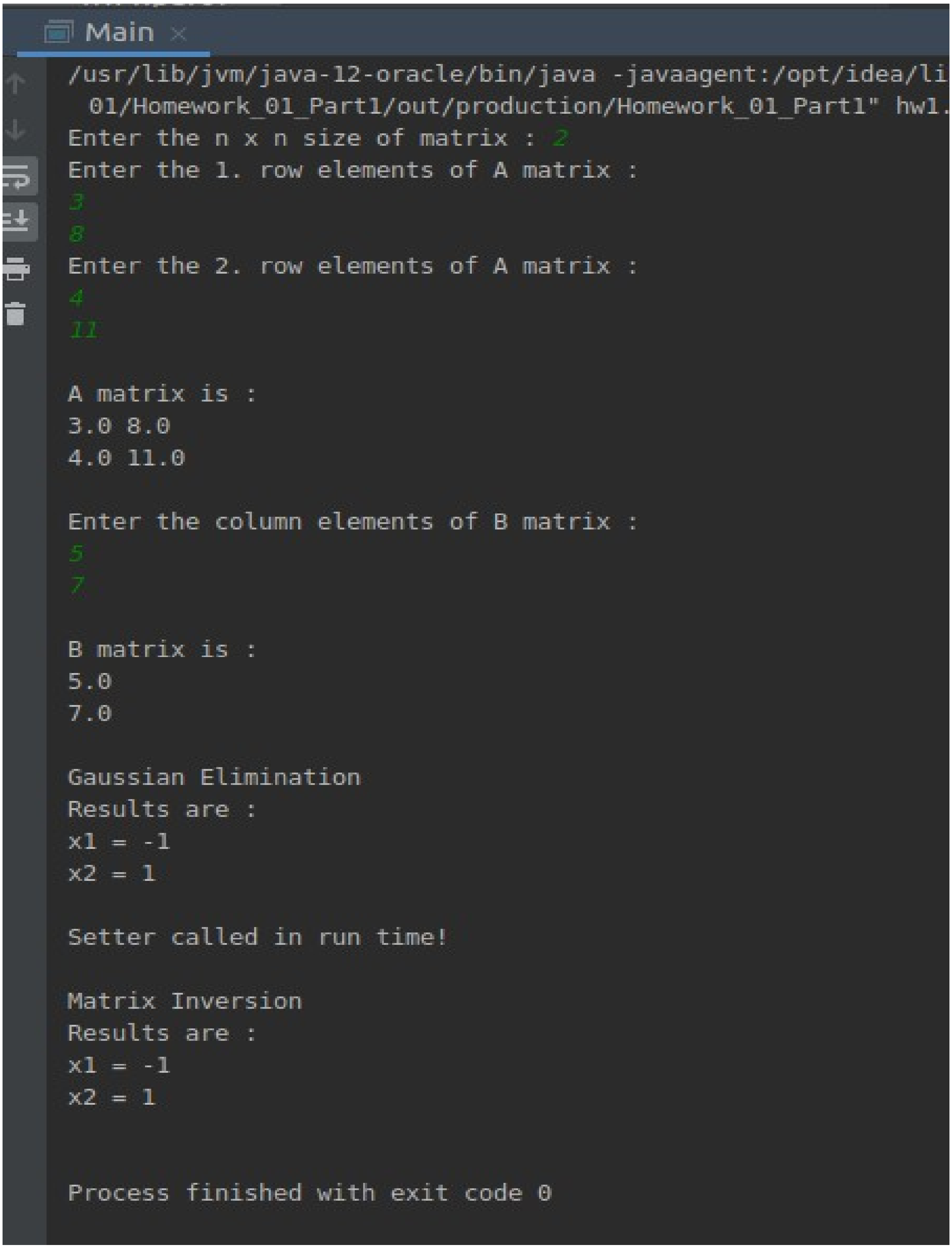
My program firstly starts with user’s n x n matrix size using console. Then user enters the A matrix using console. Then user enters the B matrix usinlg console.
In the main method firstly we call GaussianElimination method and after that we use setter to SolvingMethod and set it to MatrixInversion in the run time. I print each linear solver method’s result to the screen.

- ### <u>Class Diagram For Part 1</u>



SolvingMethod
solve(double[][], double[]) double[]
1
1
MatrixInversionMethod
multiplyByConstant(double, double[][]) double[][]
changeSign(int) int
createSubMatrix(double[][], int, int) double[][]
transpose(double[][]) double[][]
determinant(double[][]) double
cofactorMatrix(double[][]) double[][]
inverseMatrix(double[][]) double[][]
solve(double[][], double[]) double[]
Solver
solvingMethod SolvingMethod
Solver(SolvingMethod)
solve(double[][], double[]) double[]
solvingMethod SolvingMethod
GaussianEliminationMethod
EPSILON double
copyOfA(double[][]) double[][]
copyOfB(double[]) double[]
solve(double[][], double[]) double[]
«create»
«create»
«create»
Main
readA(Scanner) double[][]
readB(Scanner, int) double[]
printResults(double[]) void
main(String[]) void
Powered by yFiles

- **<u>Output Results For Part 1</u>**



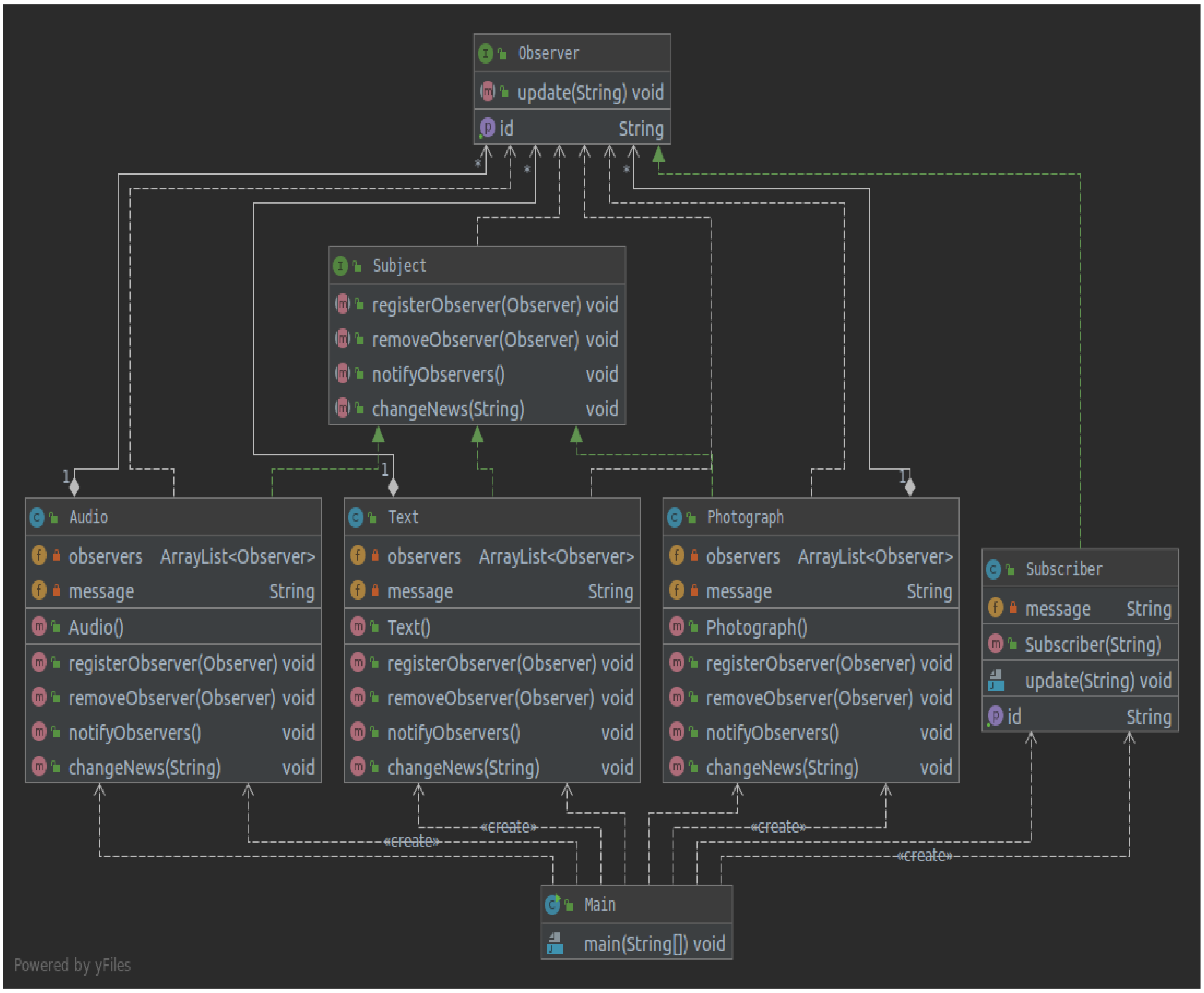
```
/usr/lib/jvm/java-12-oracle/bin/java -javaagent:/opt/idea/li
 01/Homework_01_Part1/out/production/Homework_01_Part1" hw1.
Enter the n x n size of matrix : 2
Enter the 1. row elements of A matrix :
3
8
Enter the 2. row elements of A matrix :
4
11

A matrix is :
3.0 8.0
4.0 11.0

Enter the column elements of B matrix :
5
7

B matrix is :
5.0
7.0

Gaussian Elimination
Results are :
x1 = -1
x2 = 1

Setter called in run time!

Matrix Inversion
Results are :
x1 = -1
x2 = 1


Process finished with exit code 0
```

2. Part 2

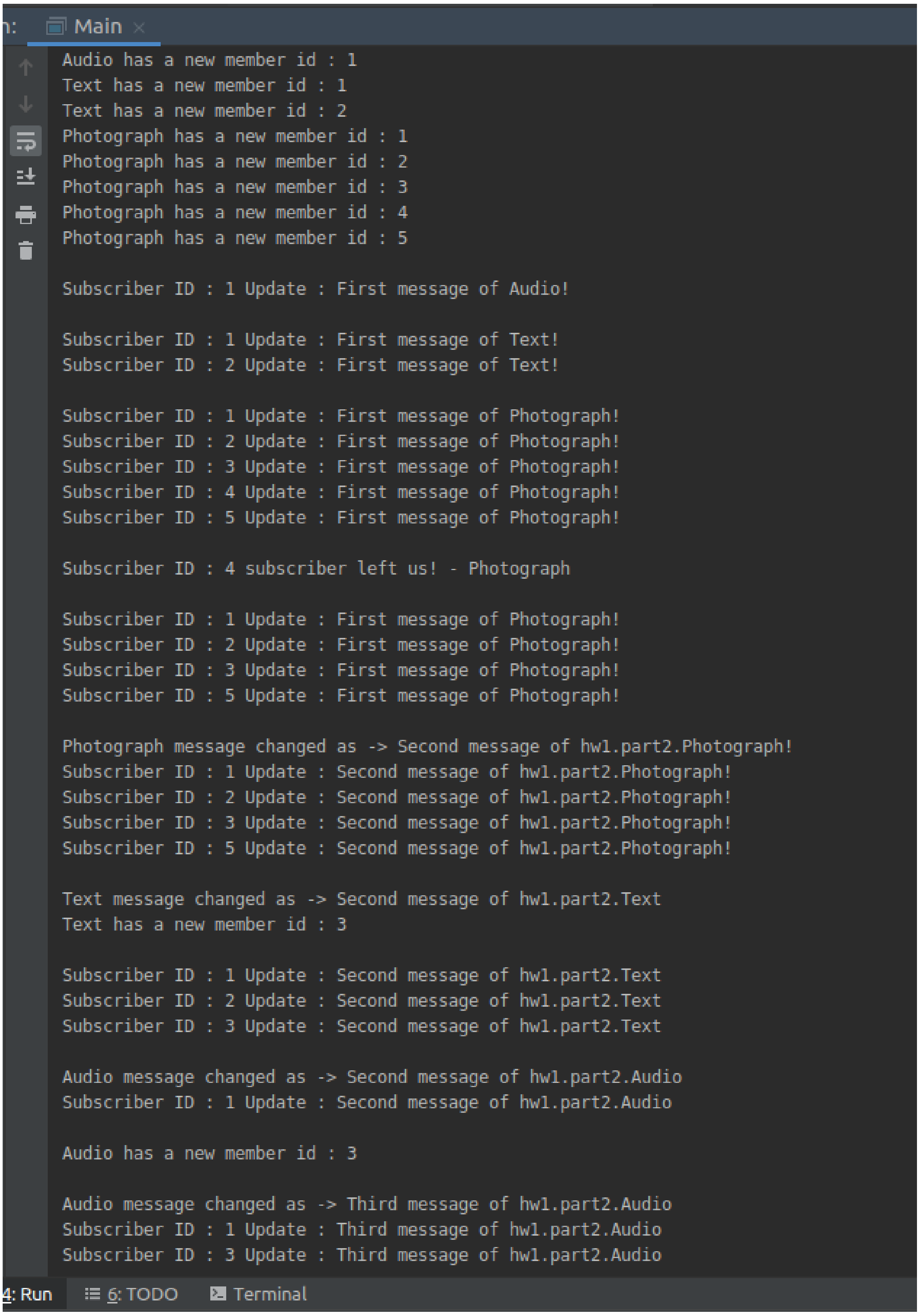
In that part, I used ‘Observer’ design pattern. I have an Observer interface which has update() method and a Subject interface which have registerObserver(), removeObserver(), notifyObservers() and changeNews() methods. In part 2 of homework there are three different Subject objects: Text, Audio and Photograph. Each of these concrete classes implements Subject interface and overrides it’s methods.

Also there is a Subscriber concrete class and it implements Observer interface and overrides update() method of its. Text, Audio and Photograph concrete classes has composition relation between Observer interface using ArrayList of it and has a loosely coupled relation. Each methods of Subject classes are executed on this Observers.

- **Class Diagram For Part 2**



- **<u>Output Results For Part 2</u>**



```
Audio has a new member id : 1
Text has a new member id : 1
Text has a new member id : 2
Photograph has a new member id : 1
Photograph has a new member id : 2
Photograph has a new member id : 3
Photograph has a new member id : 4
Photograph has a new member id : 5

Subscriber ID : 1 Update : First message of Audio!

Subscriber ID : 1 Update : First message of Text!
Subscriber ID : 2 Update : First message of Text!

Subscriber ID : 1 Update : First message of Photograph!
Subscriber ID : 2 Update : First message of Photograph!
Subscriber ID : 3 Update : First message of Photograph!
Subscriber ID : 4 Update : First message of Photograph!
Subscriber ID : 5 Update : First message of Photograph!

Subscriber ID : 4 subscriber left us! - Photograph

Subscriber ID : 1 Update : First message of Photograph!
Subscriber ID : 2 Update : First message of Photograph!
Subscriber ID : 3 Update : First message of Photograph!
Subscriber ID : 5 Update : First message of Photograph!

Photograph message changed as -> Second message of hw1.part2.Photograph!
Subscriber ID : 1 Update : Second message of hw1.part2.Photograph!
Subscriber ID : 2 Update : Second message of hw1.part2.Photograph!
Subscriber ID : 3 Update : Second message of hw1.part2.Photograph!
Subscriber ID : 5 Update : Second message of hw1.part2.Photograph!

Text message changed as -> Second message of hw1.part2.Text
Text has a new member id : 3

Subscriber ID : 1 Update : Second message of hw1.part2.Text
Subscriber ID : 2 Update : Second message of hw1.part2.Text
Subscriber ID : 3 Update : Second message of hw1.part2.Text

Audio message changed as -> Second message of hw1.part2.Audio
Subscriber ID : 1 Update : Second message of hw1.part2.Audio

Audio has a new member id : 3

Audio message changed as -> Third message of hw1.part2.Audio
Subscriber ID : 1 Update : Third message of hw1.part2.Audio
Subscriber ID : 3 Update : Third message of hw1.part2.Audio
```

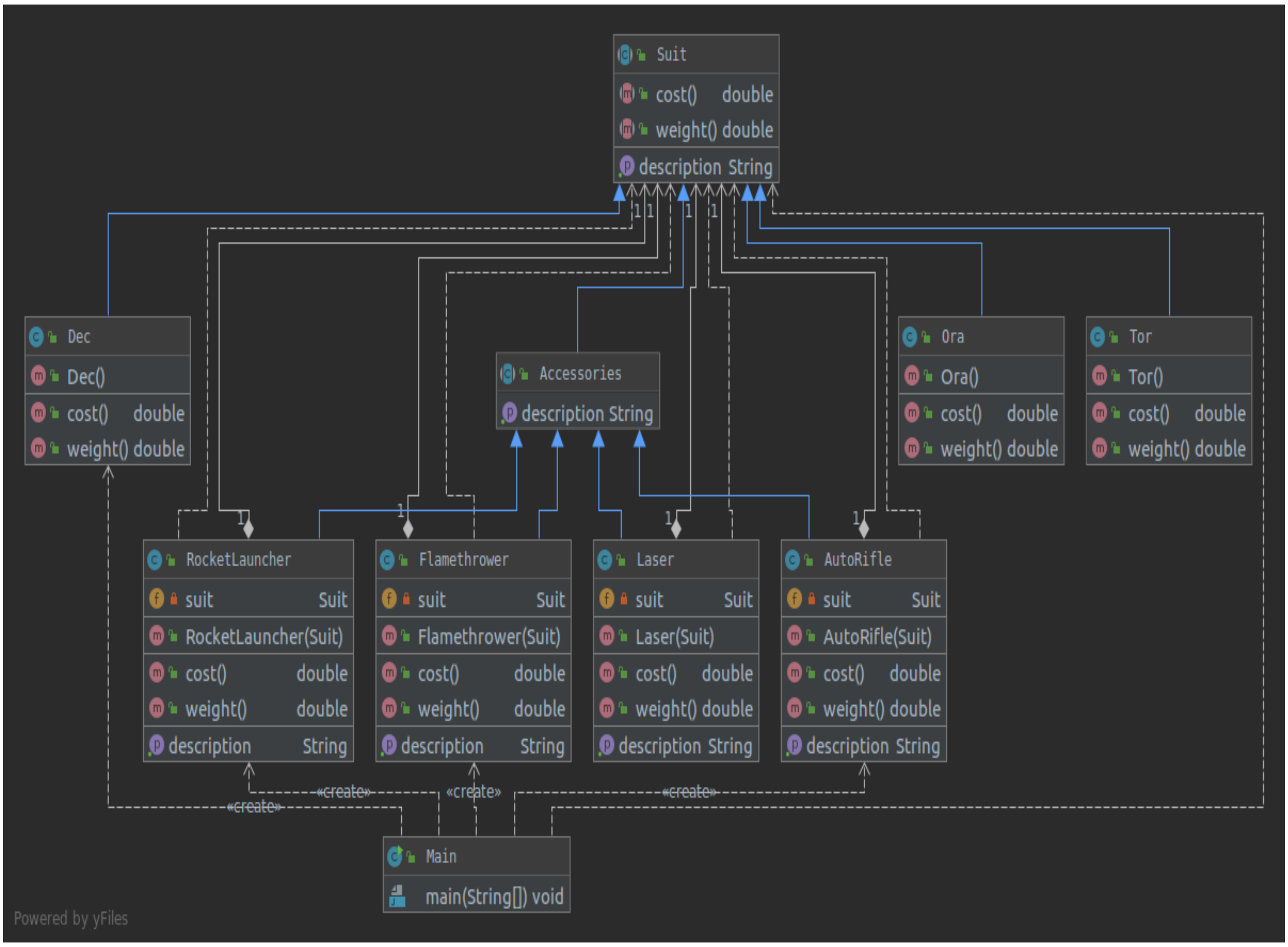
3. Part 3

In that part, I used ‘Decorator’ design pattern. There is a Suit abstract class for to implement each main suit concrete classes and it has description and its getter method, cost() and weight() methods. There are three concrete classes which extends from Suit: Dec, Ora, Tor. Each of these Dec, Ora and Tor concrete classes overrides Suit’s getDescription(), cost() and weight() methods. Also there is another class which extends Suit but as an abstract class: Accessories.

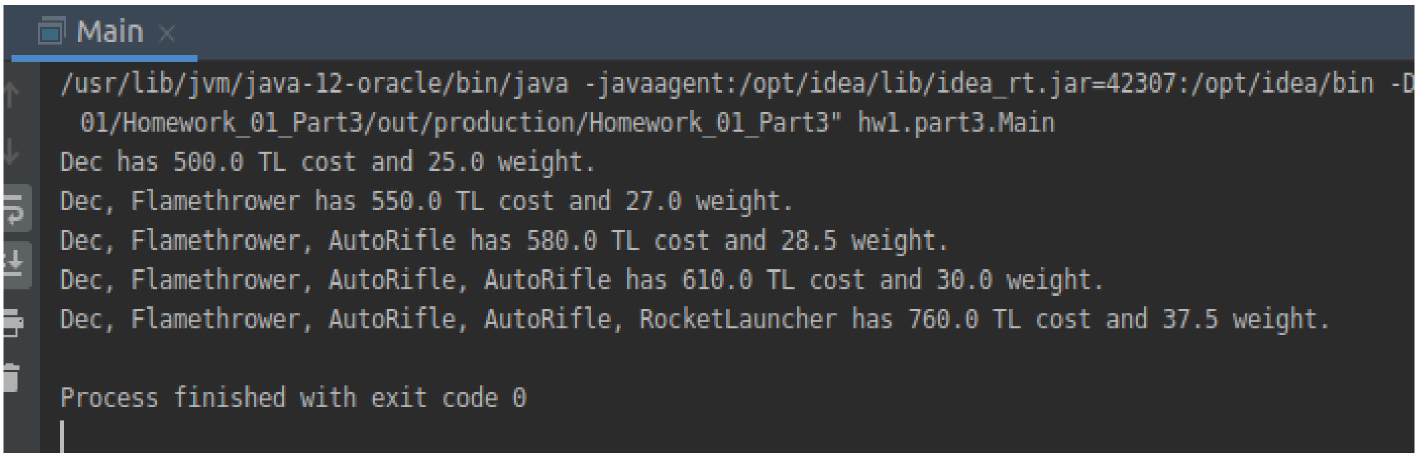
Accessories abstract class is a decorator of Suit. It abstracts getDescription() method of Suit. There are four accessories in this homework: Auto rifle, Flamethrower, Laser and Rocket Launcher. Each of these are implemented as concrete classes and extends Accessories abstract class. Extending Accessories abstract class force them to override cost(), weight() and getDescription() methods.

All of these concrete classes which extends Accessories abstract class has composition relation with Suit interface. Using constructor that applies the relation. Suit object is used for evaluate total weight and cost.

- **Class Diagram For Part 3**



- **Output Results For Part 3**



```
/usr/lib/jvm/java-12-oracle/bin/java -javaagent:/opt/idea/lib/idea_rt.jar=42307:/opt/idea/bin -D
 01/Homework_01_Part3/out/production/Homework_01_Part3" hw1.part3.Main
Dec has 500.0 TL cost and 25.0 weight.
Dec, Flamethrower has 550.0 TL cost and 27.0 weight.
Dec, Flamethrower, AutoRifle has 580.0 TL cost and 28.5 weight.
Dec, Flamethrower, AutoRifle, AutoRifle has 610.0 TL cost and 30.0 weight.
Dec, Flamethrower, AutoRifle, AutoRifle, RocketLauncher has 760.0 TL cost and 37.5 weight.

Process finished with exit code 0
```

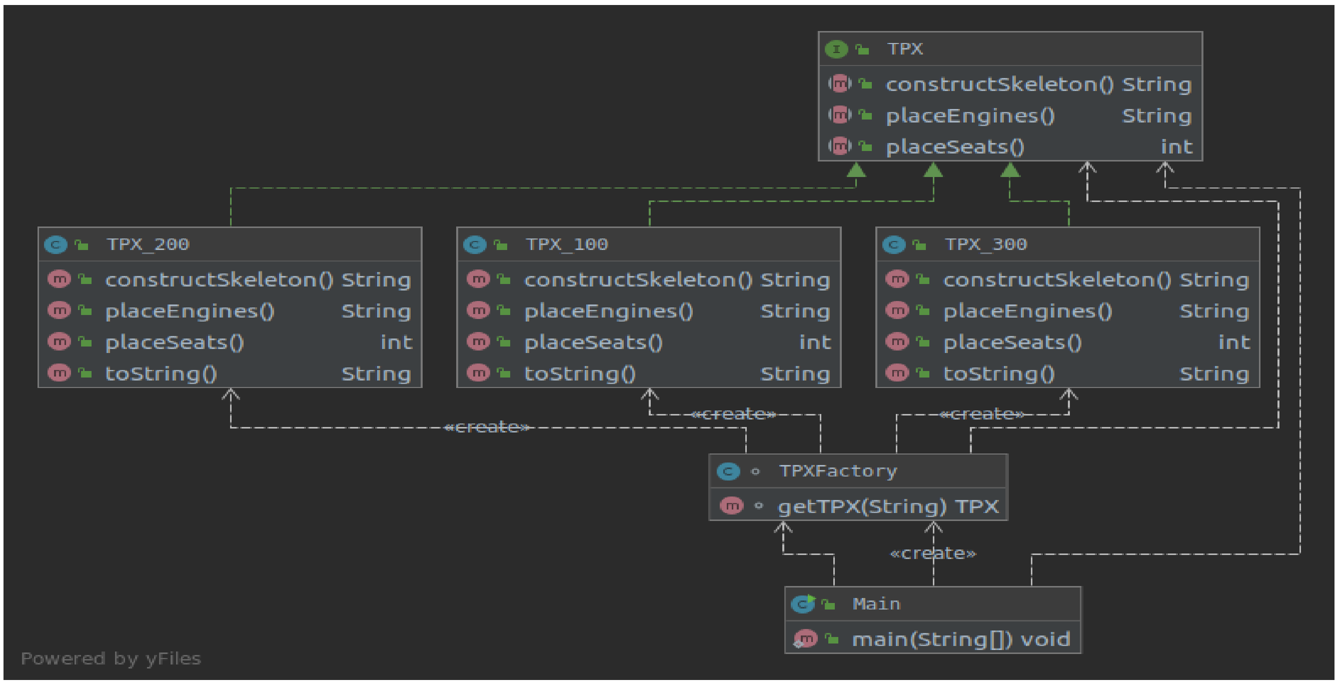
4. Part 4

I implemented that part 4 as two distant projects: Factory and Abstract Factory.

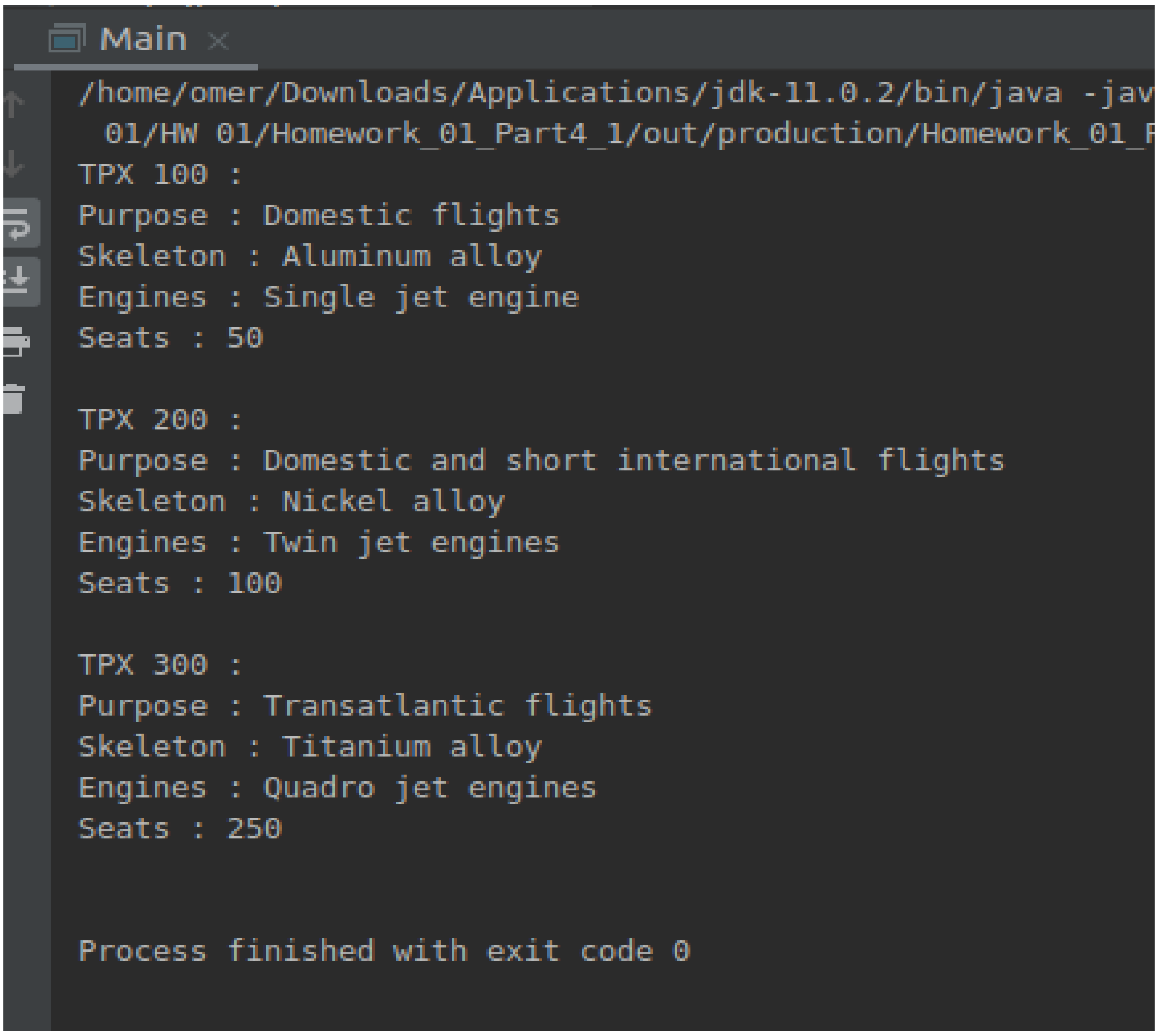
➔ **Factory Method Part**

In that part, I used 'Simple Factory Method' design pattern. In homework, there are TPX type passenger planes. I created as an interface TPX which has constructSkeleton(), placeSeats() and placeEngines() methods. There are also three different TPX models which implements TPX: TPX_100, TPX_200 and TPX_300. Each of these models overrides TPX interface methods and for to print toString() method of Object class. Also there is a TPXFactory concrete class which has getTPX() method to create TPX model and return it. In main method, I create an instance of TPXFactory and using its getTPX() method given as a parameter creating a TPX model and printing the information of it.

- **Class Diagram For Part 4 Factory**



- **<u>Output Results For Part 4 Factory</u>**



```
/home/omer/Downloads/Applications/jdk-11.0.2/bin/java -jav
  01/HW 01/Homework_01_Part4_1/out/production/Homework_01_F
TPX 100 :
Purpose : Domestic flights
Skeleton : Aluminum alloy
Engines : Single jet engine
Seats : 50

TPX 200 :
Purpose : Domestic and short international flights
Skeleton : Nickel alloy
Engines : Twin jet engines
Seats : 100

TPX 300 :
Purpose : Transatlantic flights
Skeleton : Titanium alloy
Engines : Quadro jet engines
Seats : 250


Process finished with exit code 0
```

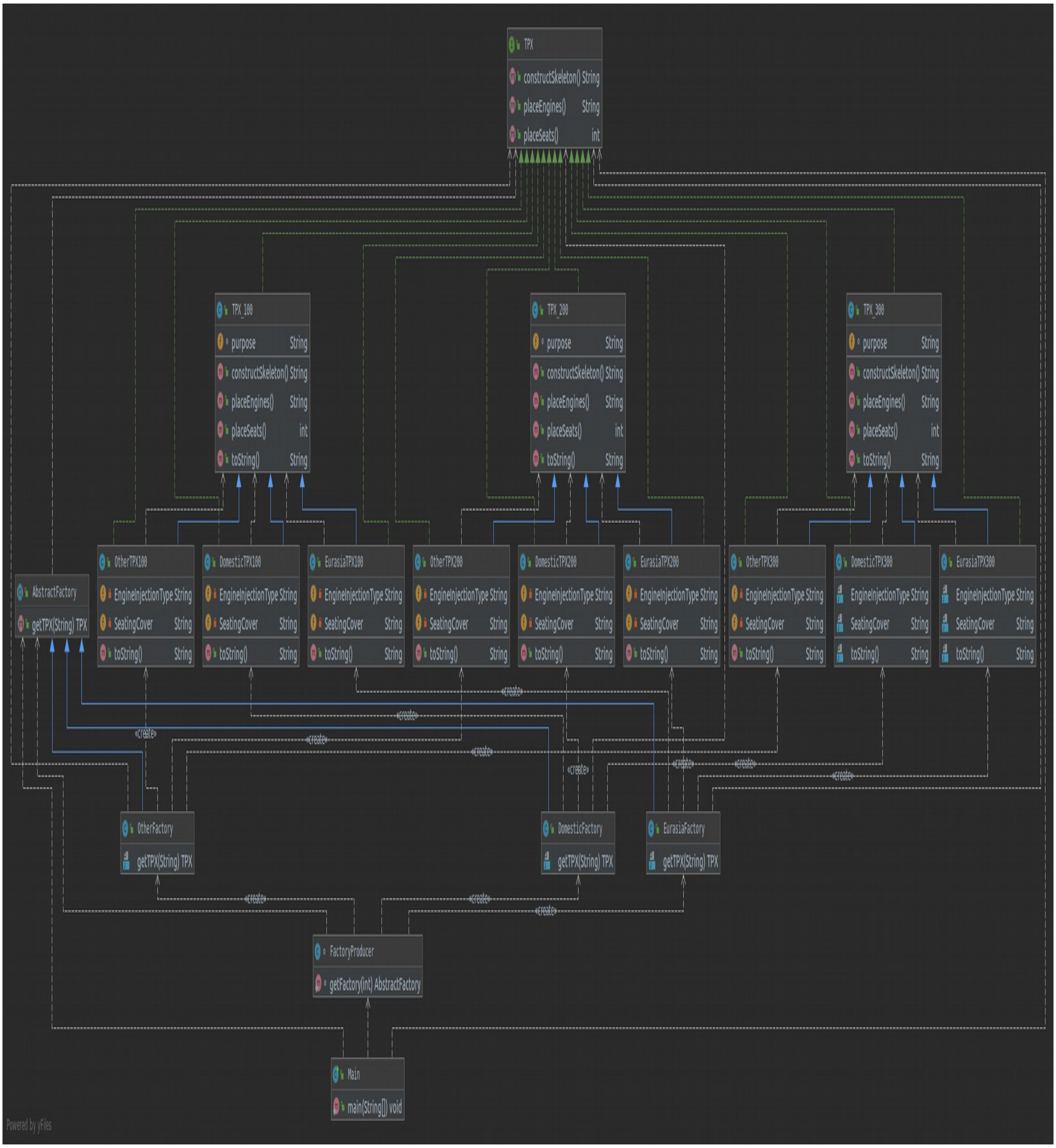
➔ **<u>Abstract Factory Method Part</u>**

In that part, I used ‘Abstract Factory Method’ design pattern. Before that part I have used Simple Factory Method design pattern to abstract TPX models’ creations. For Abstract Factory Method, I abstract TPX factories too.

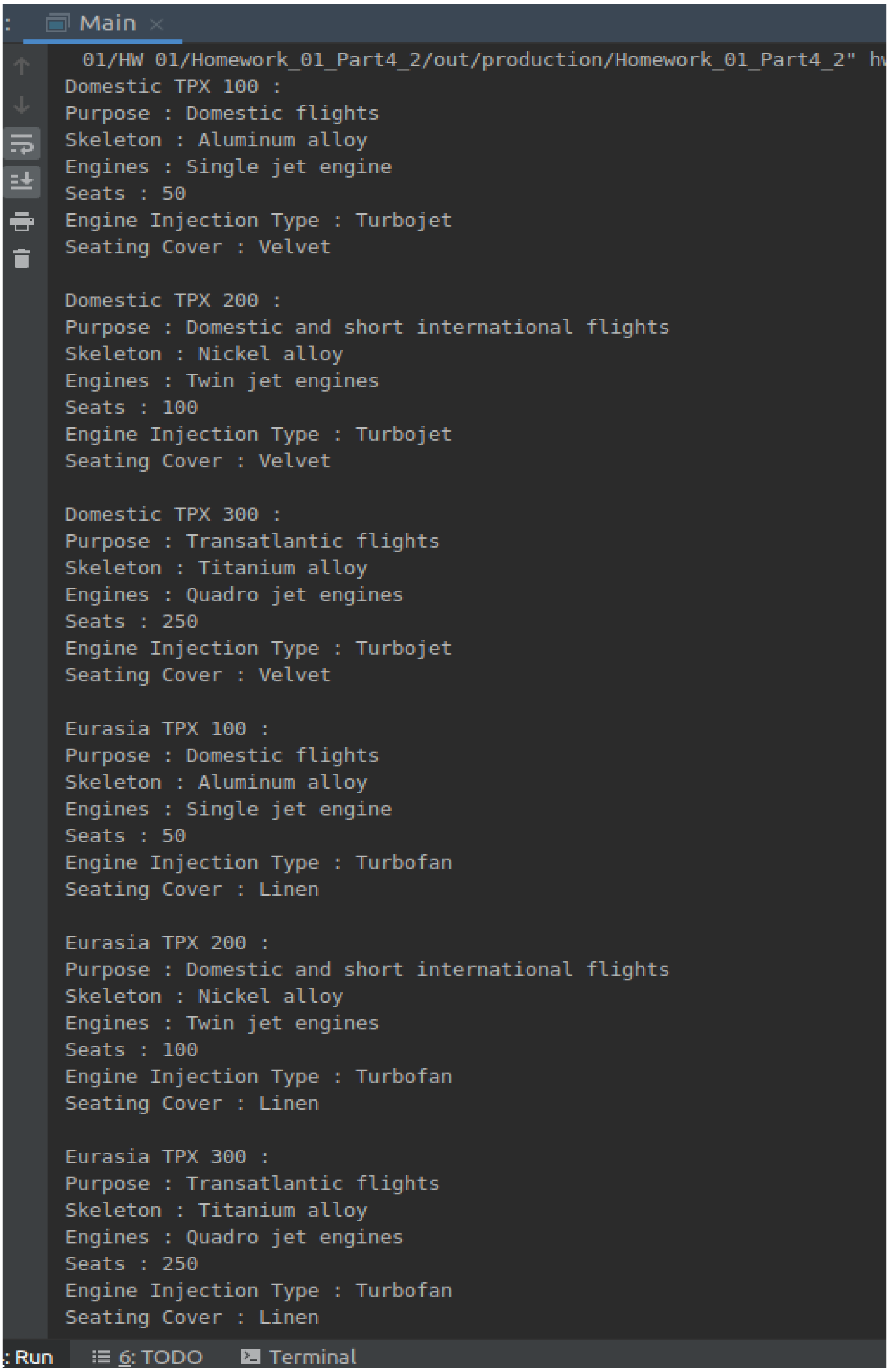
There is a AbstractFactory abstract class of all factories and has a getTPX() method as abstract to create TPX. Also there is a FactoryProducer class and has a getFactory() method to create factories given an integer parameter (0: DomesticFactory, 1: EurasiaFactory, 2: OtherFactory).

There are three main factory concrete classes which extends AbstractFactory abstract class: DomesticFactory, EurasiaFactory and OtherFactory. Each factories has their own TPX models as TPX_100, TPX_200 and TPX_300 which are implements TPX interface and overrides its methods. In main using each factories and results we see in run time change.

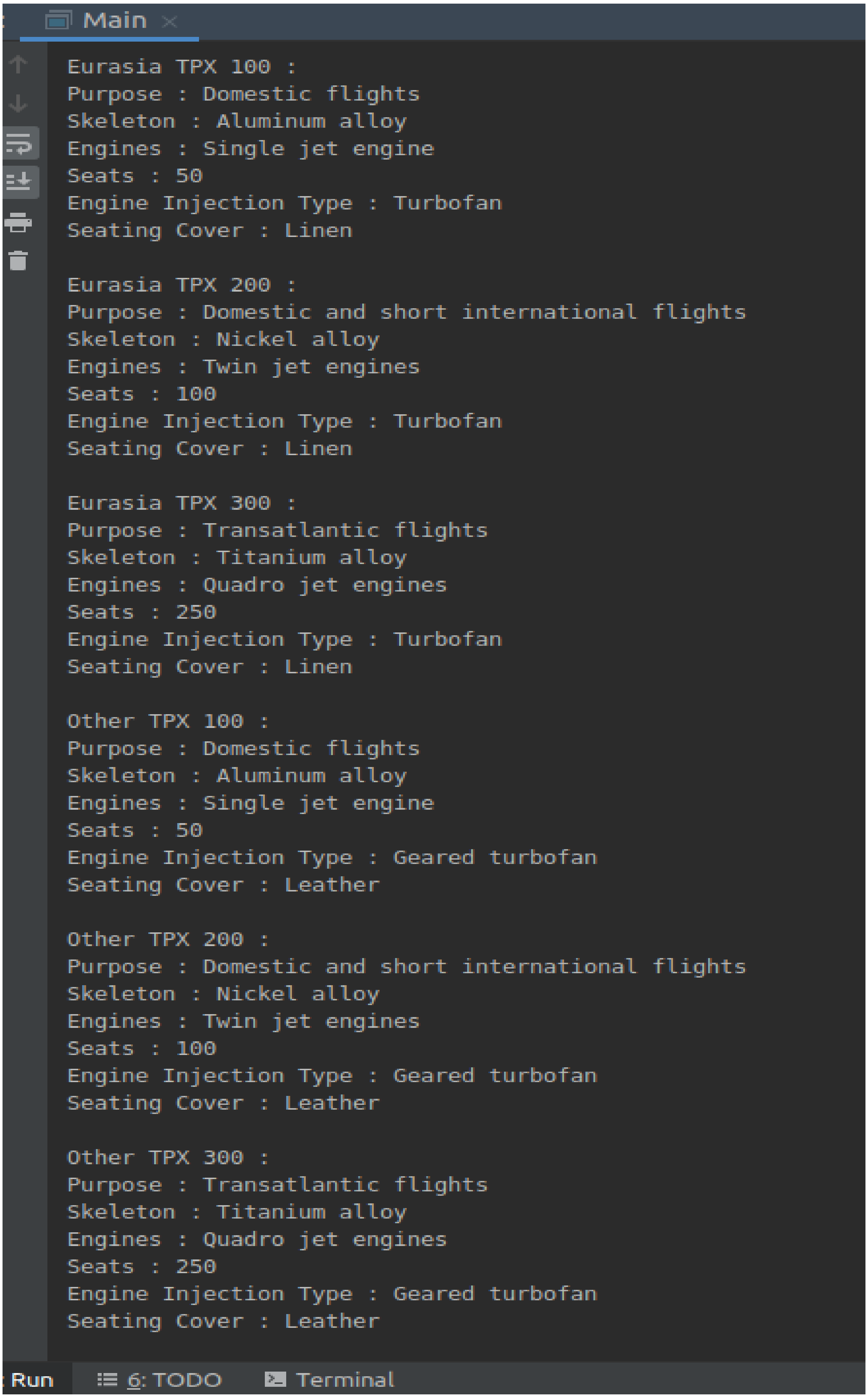
- **<u>Class Diagram For Part 4 Abstract Factory</u>**



- **<u>Output Results For Part 4 Abstract Factory</u>**



```
  01/HW 01/Homework_01_Part4_2/out/production/Homework_01_Part4_2" hw
Domestic TPX 100 :
Purpose : Domestic flights
Skeleton : Aluminum alloy
Engines : Single jet engine
Seats : 50
Engine Injection Type : Turbojet
Seating Cover : Velvet

Domestic TPX 200 :
Purpose : Domestic and short international flights
Skeleton : Nickel alloy
Engines : Twin jet engines
Seats : 100
Engine Injection Type : Turbojet
Seating Cover : Velvet

Domestic TPX 300 :
Purpose : Transatlantic flights
Skeleton : Titanium alloy
Engines : Quadro jet engines
Seats : 250
Engine Injection Type : Turbojet
Seating Cover : Velvet

Eurasia TPX 100 :
Purpose : Domestic flights
Skeleton : Aluminum alloy
Engines : Single jet engine
Seats : 50
Engine Injection Type : Turbofan
Seating Cover : Linen

Eurasia TPX 200 :
Purpose : Domestic and short international flights
Skeleton : Nickel alloy
Engines : Twin jet engines
Seats : 100
Engine Injection Type : Turbofan
Seating Cover : Linen

Eurasia TPX 300 :
Purpose : Transatlantic flights
Skeleton : Titanium alloy
Engines : Quadro jet engines
Seats : 250
Engine Injection Type : Turbofan
Seating Cover : Linen
```



```
Eurasia TPX 100 :
Purpose : Domestic flights
Skeleton : Aluminum alloy
Engines : Single jet engine
Seats : 50
Engine Injection Type : Turbofan
Seating Cover : Linen

Eurasia TPX 200 :
Purpose : Domestic and short international flights
Skeleton : Nickel alloy
Engines : Twin jet engines
Seats : 100
Engine Injection Type : Turbofan
Seating Cover : Linen

Eurasia TPX 300 :
Purpose : Transatlantic flights
Skeleton : Titanium alloy
Engines : Quadro jet engines
Seats : 250
Engine Injection Type : Turbofan
Seating Cover : Linen

Other TPX 100 :
Purpose : Domestic flights
Skeleton : Aluminum alloy
Engines : Single jet engine
Seats : 50
Engine Injection Type : Geared turbofan
Seating Cover : Leather

Other TPX 200 :
Purpose : Domestic and short international flights
Skeleton : Nickel alloy
Engines : Twin jet engines
Seats : 100
Engine Injection Type : Geared turbofan
Seating Cover : Leather

Other TPX 300 :
Purpose : Transatlantic flights
Skeleton : Titanium alloy
Engines : Quadro jet engines
Seats : 250
Engine Injection Type : Geared turbofan
Seating Cover : Leather
```