Gebze Technical University
Computer Engineering

Object Oriented Analysis and Design
CSE 443

HOMEWORK 1 REPORT

Değer MANDAL

161044096

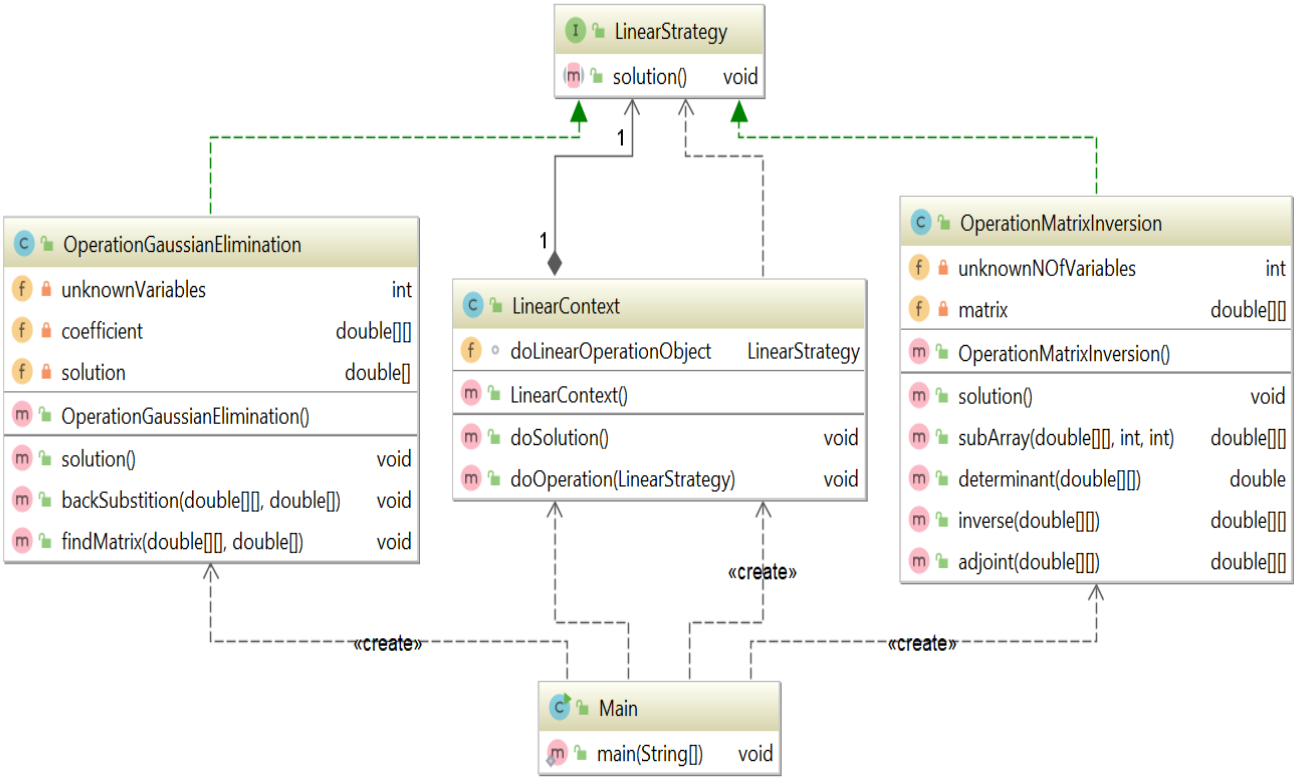
Important: I used “jdk-14.0.2” in my homework.

Part1

Strategy Pattern

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).
I used the strategy pattern for the problem because if we use strategy pattern, we can change solving method dynamically at run time.
LinearContext: LinearContext class has an instance of the LinearStrategy and has doOperation method to create this instance dynamically.
LinearStrategy: By designing an interface, we collect all our common algorithms here. If user wants new solving methods, it can simply added as new class which implements LinearStrategy.
OperationGaussElimination and OperationMatrixInversion: Our actual class that implements the corresponding algorithm. In this system we can be working with multiple methods. We can do by matrix inversion, gauss elimination.

Class Diagram:



Example Output:

```
"C:\Program Files\Java\jdk-14.0.2\bin\java.exe" -Didea.launcher.
Welcome, please select which method will use for the solution!!
#.....................................................#
Please, write 1 for choosing Gaussian Elimination
Please, write 2 for choosing Matrix Inversion
Please, write 3 for exit
Choice:
1
Please, write number of unknown variables:
3
Write equation's coefficients =>
1. coefficents:
1 2 3
2. coefficents:
4 5 6
3. coefficents:
1 0 1
Write solutions:
1. solution:
1
2. solution:
1
3. solution:
1
[ 0.0, -1.0, 1.0 ]
```

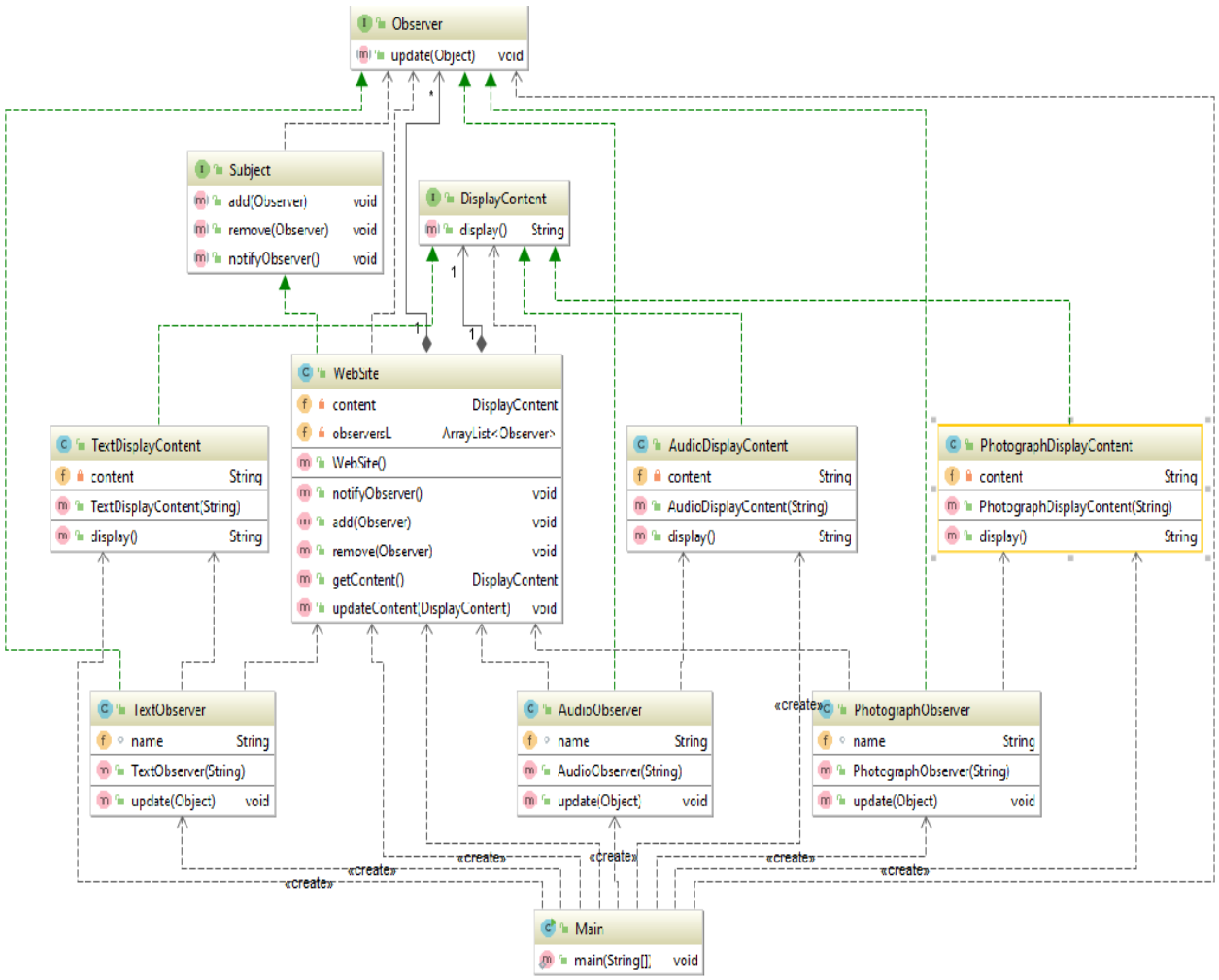
```
#.....................................................#
Please, write 1 for choosing Gaussian Elimination
Please, write 2 for choosing Matrix Inversion
Please, write 3 for exit
Choice:
2
Please, write the dimension of the matrix:
3
Please, write the elements of matrix =>
1. row :
1 2 3
2. row :
4 5 6
3. row :
1 0 1
-0.8333333333333333     -0.3333333333333333     0.5
-0.3333333333333333     -0.3333333333333333     -1.0
0.8333333333333333     0.3333333333333333     0.5
```

```
#.....................................................#
Please, write 1 for choosing Gaussian Elimination
Please, write 2 for choosing Matrix Inversion
Please, write 3 for exit
Choice:
4
Wrong input, please write correct input!!!
#.....................................................#
Please, write 1 for choosing Gaussian Elimination
Please, write 2 for choosing Matrix Inversion
Please, write 3 for exit
Choice:
3
Goodbye!!!
```

Part2
Observer Pattern

We need observer pattern because we have to notify subscriber when content is changed. In order to implement the Observer design pattern for this problem, there will be 2 interfaces which Subject and Observer were created. The observer interface includes the update() method. Classes wishing to subscribe to the website must implement this interface. The subject interface has been created to provide external API. The Subject interface is created to control observers such as removing ,adding notifying observers.
WebSite class has an instance of DisplayContent. DisplayContent interface hold the information about the contents. PhotographDisplayContent, TextDisplayContent and AudioDisplayContent classes implements the DisplayContent interface. DisplayContent instance can change dynamically.

Class Diagram:



Example Output:

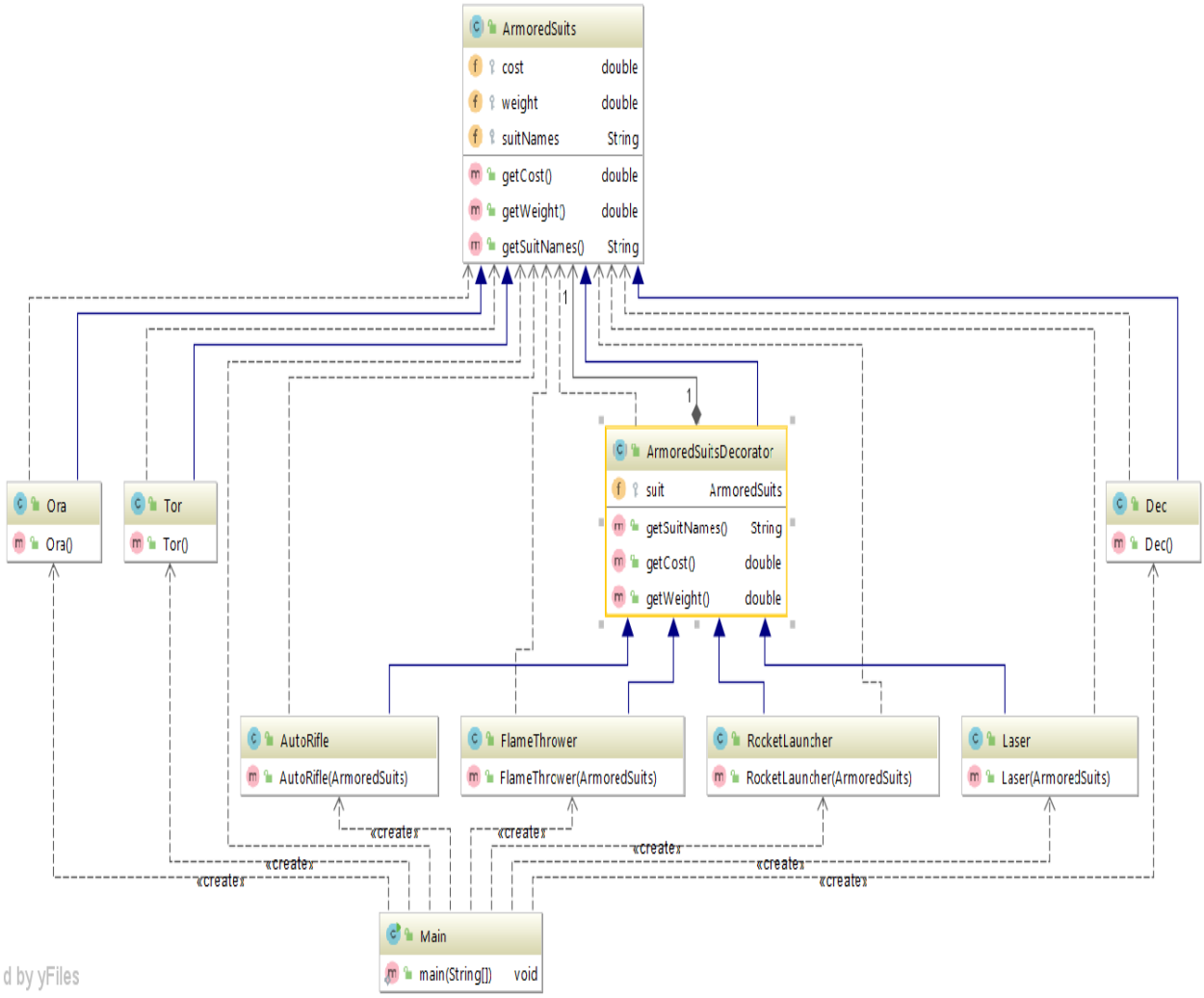
```
WebSite wb = new WebSite();
Observer obs1 = new PhotographObserver( name: "Deger");
Observer obs2 = new AudioObserver( name: "Zeynep");
Observer obs3 = new TextObserver( name: "Seda");
Observer obs4 = new TextObserver( name: "Elif");
Observer obs5 = new PhotographObserver( name: "Seda");
Observer obs6 = new AudioObserver( name: "Seda");
wb.add(obs1);
wb.add(obs2);
wb.add(obs3);
wb.add(obs4);
wb.add(obs5);
wb.add(obs6);
System.out.println("################################################################");
wb.updateContent(new AudioDisplayContent("Audio of GOT"));
wb.updateContent(new TextDisplayContent("Text Of GOT"));
wb.updateContent(new PhotographDisplayContent("Photo of GOT"));
System.out.println("################################################################");
wb.remove(obs6);
wb.remove(obs1);

wb.updateContent(new AudioDisplayContent("Audio of Merlin"));
wb.updateContent(new TextDisplayContent("Text Of Merlin"));
wb.updateContent(new PhotographDisplayContent("Photo of Merlin"));
System.out.println("################################################################");
```

```
"C:\Program Files\Java\jdk-14.0.2\bin\java.exe" -Didea.launcher.port=52925 "-Idea
############################################################
Notify : Zeynep new AUDIO, check your update. ---- Audio: Audio of GOT

Notify : Seda new AUDIO, check your update. ---- Audio: Audio of GOT

Notify : Seda new TEXT, check your update. ---- Text: Text Of GOT

Notify : Elif new TEXT, check your update. ---- Text: Text Of GOT

Notify : Deger new PHOTOGRAPH, check your update. ---- Photograph: Photo of GOT

Notify : Seda new PHOTOGRAPH, check your update. ---- Photograph: Photo of GOT

############################################################
Notify : Zeynep new AUDIO, check your update. ---- Audio: Audio of Merlin

Notify : Seda new TEXT, check your update. ---- Text: Text Of Merlin

Notify : Elif new TEXT, check your update. ---- Text: Text Of Merlin

Notify : Seda new PHOTOGRAPH, check your update. ---- Photograph: Photo of Merlin

############################################################
```

Part3
Decorator Pattern

We need decorator pattern because we can set suit object and integrate with component that flamethrower, autorifle, rocketlauncher and laser. That is to say, additional responsibilities can attach to an object dynamically using decorater pattern. So, decorator allows behavior in order to added to any dec, ora and tor. ArmoredSuitsDecorator class keeps reference everytime each component or suit.

Class Diagram:



Example Output:

```
public static void main(String[] args) {
    ArmoredSuits suit;
    suit = new Dec();
    System.out.println("Total cost and weight: " +  suit.getSuitNames() + "  =  "
            + suit.getCost() + "k TL ,  " + suit.getWeight() + "kg");

    suit = new Ora();
    suit = new FlameThrower(suit);
    suit = new AutoRifle(suit);
    suit = new RocketLauncher(suit);
    suit = new Laser(suit);
    System.out.println("Total cost and weight: " +  suit.getSuitNames() + "  =  "
            + suit.getCost() + "k TL ,  " + suit.getWeight() + "kg");

    suit = new Tor();
    suit = new FlameThrower(suit);
    suit = new AutoRifle(suit);
    suit = new AutoRifle(suit);
    suit = new Laser(suit);
    System.out.println("Total cost and weight: " +  suit.getSuitNames() + "  =  "
            + suit.getCost() + "k TL ,  " + suit.getWeight() + "kg");
```

```
"C:\Program Files\Java\jdk-14.0.2\bin\java.exe" -Didea.launcher.port=53061 "-Didea.launcher.bin.path=
Total cost and weight: Dec  =  500.0k TL ,  25.0kg
Total cost and weight: Ora, FlameThrower, AutoRifle, RocketLauncher, Laser  =  1930.0k TL ,  46.5kg
Total cost and weight: Tor, FlameThrower, AutoRifle, AutoRifle, Laser  =  5310.0k TL ,  60.5kg
```