CSE – 443 OBJECT ORIENTED ANALYSIS AND DESIGN

HOMEWORK 1 REPORT

OĞUZHAN SEZGİN

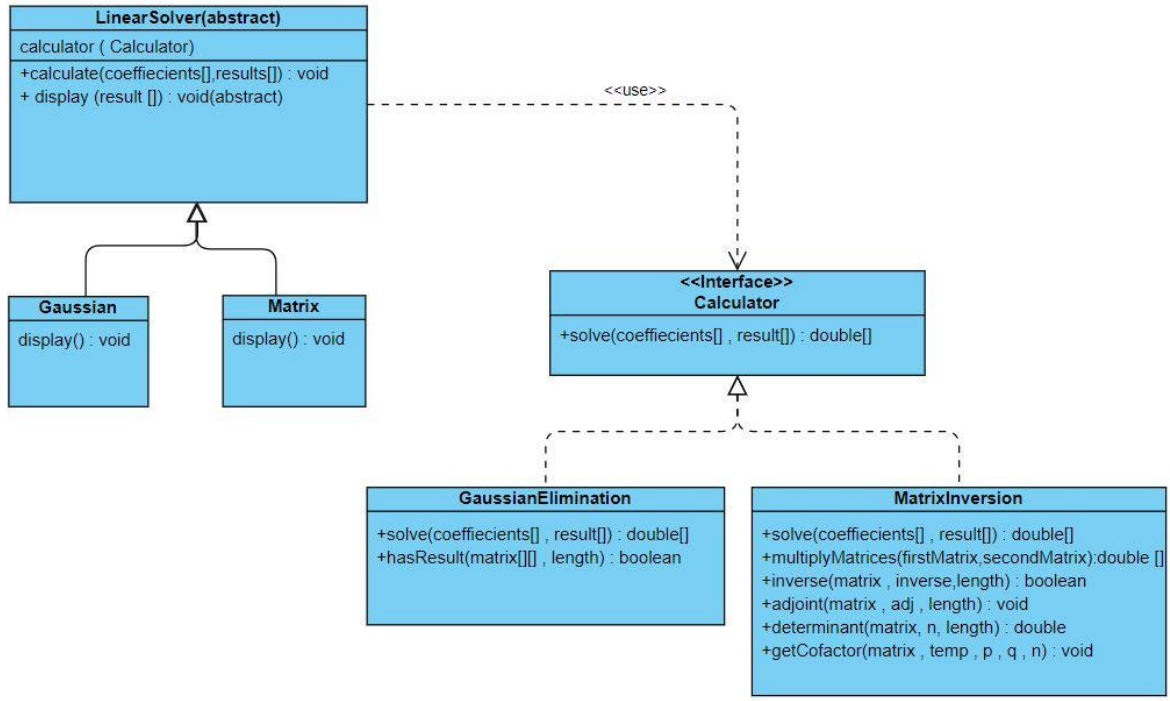
QUESTION 1

In this part of homework 1 use **Strategy Design Pattern**. Because client wants to solve equation in different way and maybe he/she will want to add new ways of solving. So I create a **LinearSolver** abstract class and **Calculator** interface.

Calculator interface has a solve method. **Solve** method include solve implementation of equation. Then I created **GaussianElimination** and **MatrixInversion** class. This classes implements **Calculator** interface so each of these classes implement solve method according to their own solving type. And if client wants to add new solving type ,programmer can create new class and implement **Calculator** interface.

On the other hand **LinearSolver** class has **Calculator** reference and calculate and display methods. "**Calculate**" method firstly call calculator's solve method and then call display method. **Display** method show results of equation. Then I created **Gaussian** and **Matrix** method. This classes implements their display method and assign their **Calculator** reference to a class which implements **Calculator** interface. So if client wants to add new solving type he/she can implement a **Calculator** interface and **LinearSolver** abstract class.

The class diagram showing below :



QUESTION 2

In this part of homework I use **Observer Design Pattern**. Because client wants to create a program which subscribe user to a web site and user notify when made update. So I created a **Subject** interface , **Observer** interface and **Notification** interface and **WebContent** abstract class.

Observer interface has **update** method. This method, record new updates to the derived class which programmer implements(Subscriber class in this example).

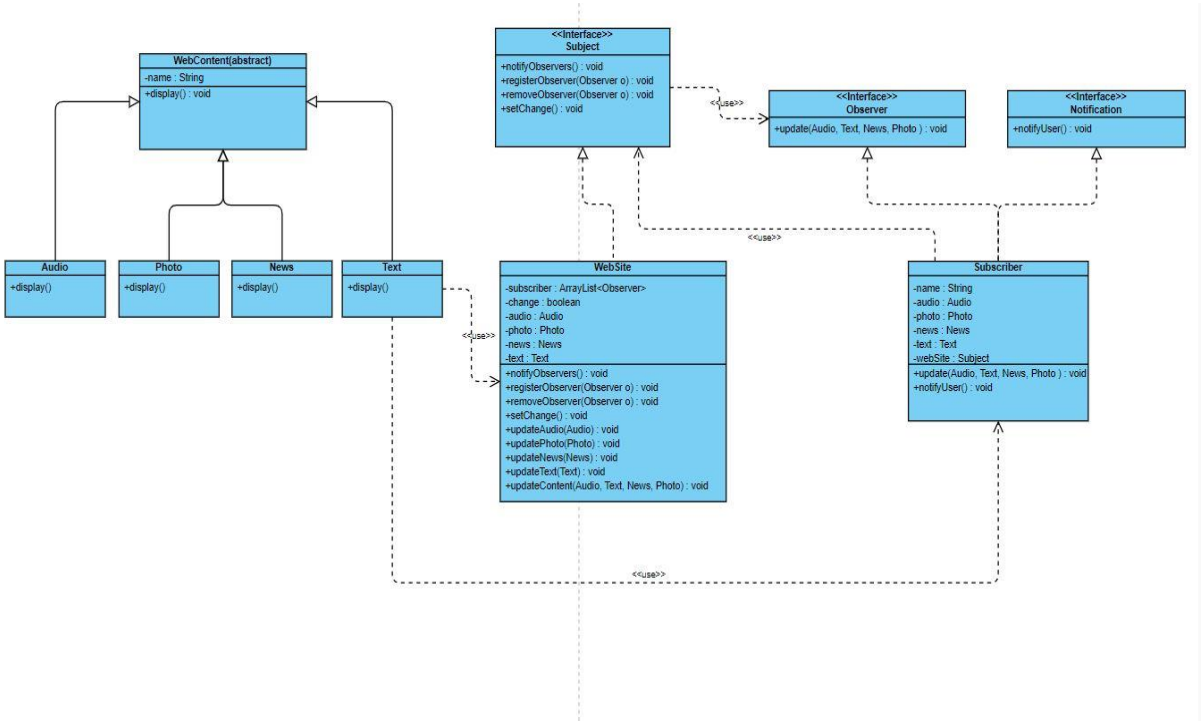
Notification interface has **notifyUser** method. This method notify user (prints website content) when new updates record to the derived class(Subscriber).

WebContent abstract class has a **name** and **display** method. Web contents are derived this class(audio , photo , text , news , etc.). If user wants to add new content she/he derive this class. Display method prints to web contents info. All derived class implement their display method.

Subject interface has **notifyObservers** , **registerObserver** , **removeObserver** and **setChange** methods. **registerObserver** method registers new observer(subscriber) , **removeObserver** method removes observers(Subscriber) , **setChange** method set the change variable which there is in the derived class(WebSite) true when the desired updates made . **notifyObservers** method if the change variable true, call observer's(Subscribers) "update" method and sets the change variable false.

I created **Subscriber** class which implements **Observer** and **Notification** interface. This class implements its own update and **notifyUser** method. Then I crated **WebSite** class. This class implements Subject interface and some update methods which seen on the class diagram .**WebSite** class has Observers **ArrayList** for the subscriber and it stores subscriber on this **Arraylist**.

The class diagram showing below :



QUESTION 3

In this part of homework I use Decorator Design Pattern. Because client wants to add new property to their armor dynamically. So I created to a **Armor** and **EquipmentDecorator** abstract classes. **EquipmentDecorator** derived from the Armor abstract class.

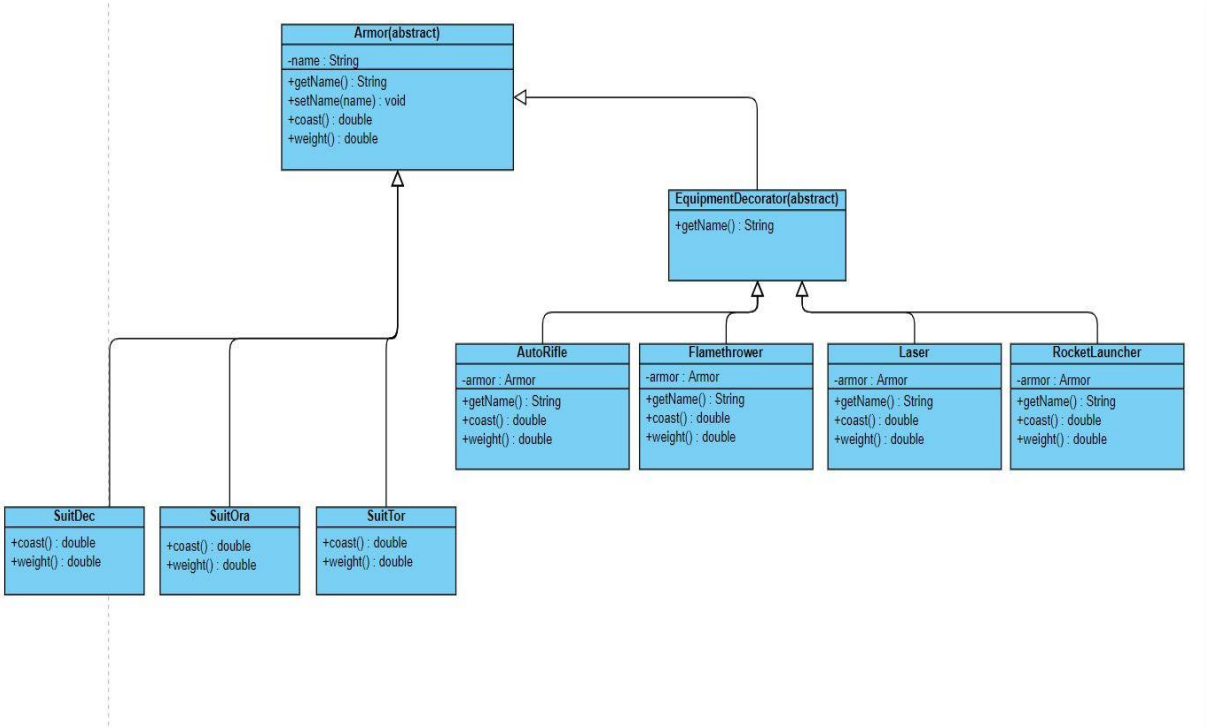
Armor class has name attribute , **getName** method , **setName** method , **cost** method , **weigth** method. **getName** method returns the name variable of Armor class, **setName** method set the name variable of **Armor** class, coast method return the cost of the armor and weight method return the armor weight.

EquipmentDecorator extends to **Armor** class. It override the **getName** method because this method adds and returns class names before itself and own. Armor's equipment extends this class(**Flamethrower**, **AutoRifle**, **RocketLauncher**, **Laser**, etc).If user wants to add new equipment she/he create a class and extends **EquipmentDecorator** class .

I created **Flamethrower**, **AutoRifle**, **RocketLauncher**, **Laser** classes. These classes extends **EquipmentDecorator** class. Theses classes implements cost and weight methods. These methods adds and returns armor cost and weight after itself and own. Also these classes has Armor class reference. In this way these classes can encapsulate other suits or equipment. If user wants to add a equipment to armor he/she create new equipment class(Auto Rifle , etc.) and assign this class's **Armor** reference to **Armor** class which want to add. In this way armor encapsulated to a new equipment.

I created **SuitDec** , **SuitOra** , **SuitTor** classes. This classes extends **Armor** class. These class implements cost and weight method. These methods only return their cost and their weight because it can not encapsulate other suits or armor. So client can not encapsulate a suit with a suit. If user wants create a new suit he/she create a class and extends **Armor** class.

The class diagram showing below :



All of these packages has their own Tester class and Main class(except question 2) packeage.

Tester classes tests all methods of their package class.

Main classes is a user interface classes. User run and use these programs.

Javadoc document is in doc file(index.html).