

**Gebze Technical University  
Computer Engineering**

**Object Oriented Analysis and Design  
CSE443 – 2018 Autumn**

**HOMEWORK 2 REPORT**

**Yunus ÇEVİK  
141044080**

Course Teacher: Erchan Aptoula

Course Assistant: Muhammet Ali Dede

## Answer – 1

- 1.1 Singleton tasarım örüntüsünde adından da anlaşıldığı gibi tek nesne tasarım örüntüsüdür. Bu tasarım örüntüsünde Object sınıfından kalıtım yapılmış olan clone() metodunu kullanarak nesneyi klonlamak mümkündür. Ancak klonlanması tasarım örüntüsüne uymayan bir durumdur tek nesne anlayışını karşılamamaktadır. Ayrıca clone() metodunun kullanılabilmesi için Singleton sınıfının Cloneable interface' inden implements edilmesi ve clone() metodunun override edilerek Object sınıfının clone metodu super.clone() ile çağırılması gerekmektedir. Clone() metodu yeni bir nesne oluşturmaz var olan nesnenin bir kopyasını oluşturur. Aşağıda Cloneable olmayan bir sınıfın objesi oluşturulduktan sonra print() metodunun çağırılması ve clone() metodu ile kopya obje üzerinden print() metodunun çağırılmasını göstereceğim. Ayrıca clone metodundan sonra yeni bir objenin oluşturulmadığı görünmektedir.

```
package singleton;

public class SingletonObject{
    private static SingletonObject instance = null;
    private SingletonObject(){
        System.out.println("new Object");
    }

    public static SingletonObject getInstance(){
        if(instance == null) {
            instance = new SingletonObject();
        }
        return instance;
    }

    public void print(){
        System.out.println("Hello");
    }

    @Override
    public Object clone() throws CloneNotSupportedException{
        return super.clone();
    }
}

public class Main {

    public static void main(String[] args) {
        SingletonObject object = SingletonObject.getInstance();
        object.print();
        try {
            SingletonObject cloneSingleton = (SingletonObject) object.clone();
            cloneSingleton.print();
        } catch (CloneNotSupportedException e) {
            System.out.println(e.getMessage());
        }
    }
}
```

```
"C:\Program Files (x86)\Java\jdk1.8.0_172\bin\java.exe" ...
new Object ➡ Yeni obje oluşturulması
Hello ➡ Oluşturulan objenin print() metodunu çağırması
singleton.SingletonObject ➡ Oluşturulan objenin klonun
                                oluşturulmaya çalışması
Process finished with exit code 0 (Not Cloneable)
```

- 1.2 SingletonObject sınıfı içerisinde Object sınıfından Override edilen clone() metodu içerisinde “CloneNotSupportedException” fırlatarak kullanıcıya Singleton objesinin klonlanamayacağını önlemiş oluruz.

```
@Override
public Object clone() throws CloneNotSupportedException{
    throw new CloneNotSupportedException("Object can not be cloned");
}
```

"C:\Program Files (x86)\Java\jdk1.8.0\_172\bin\java.exe" ...  
new Object  
Hello  
Object can not be cloned    Bir Singleton objesi klonlanmak istediğinde  
                                 "CloneNotSupportedException" fırlatarak bunu  
Process finished with exit code 0 önleyebiliriz.

- 1.3 Eğer SingletonObject sınıfı Cloneable interface yapısından implements edilseydi. Yukarıda belirtilen 1.1’ deki yapıda clone() metodu ile oluşturulan objenin bir klonu alınabilecektir. Böylece ana obje ve klonlanan obje print() metodunu çağırabileceklerdir. Ancak clone() metodu sonunda, klon olan obje yeni bir obje olmayacaktır.

```
package singleton;

public class SingletonObject implements Cloneable{
    .....
    .....
}

@Override
public Object clone() throws CloneNotSupportedException{
    return super.clone();
}
```

"C:\Program Files (x86)\Java\jdk1.8.0\_172\bin\java.exe" ...  
new Object  
Hello    ➡    Singleton Objesi ( new )  
Hello    ➡    Singleton Objesi ( clone )  
  
Process finished with exit code 0

SingletonObject sınıfı Cloneable interface’ inden implements edilse bile yukarıda belirtilen 1.2’ deki yapıda clone() metodunun “CloneNotSupportedException” fırlatmasından dolayı Singleton objesinin klonlanması tamamen önlenmiş olur. Burada “Cloneable” olması çok önemli değildir. “CloneNotSupportedException” ile klonlamanın önüne geçilmiş olunur.

```
package singleton;

public class SingletonObject implements Cloneable{
    .....
    .....
}

@Override
public Object clone() throws CloneNotSupportedException{
    throw new CloneNotSupportedException("Object can not be cloned");
}
```

"C:\Program Files (x86)\Java\jdk1.8.0\_172\bin\java.exe" ...  
new Object  
Hello  
Object can not be cloned    Bir Singleton objesi klonlanmak istediğinde  
                                 "CloneNotSupportedException" fırlatarak bunu  
Process finished with exit code 0 önleyebiliriz.

## Answer – 2

### 1 METHOD

ZırhSan A.Ş projesini tasarlarlarken Decorator Tasarım Desenini kullanarak tasarladım. Bu tasarım deseni, structural tasarım desenlerinden biridir. Bir nesneye dinamik olarak yeni özellikler eklemek için kullanılır. Kalıtım kullanmadan da bir nesnenin görevlerini artırabileceğimizi gösterir.

Not: Bir sınıfın nesnesine runtime zamanında eklenen özellikler, bu sınıftan yaratılmış diğer nesneleri etkilemez.

Decorator tasarım deseninde kalıtım, sadece sınıflar aynı türe sahip olsunlar diye kullanılmaktadır. Nesnenin fonksiyonlarını composition ile sağlıyoruz. Eğer kalıtıma bağlı kalsaydık, compile time zamanında nesnenin davranışı belirlenecekti. Böyle bir durumda ise dinamik olarak yeni özellikler ekleyemeyecektik.

Decorator Tasarım Deseni'nin faydaları:

- loosely-coupled uygulamalar yapmayı sağlar.
- Runtime zamanında(dinamik olarak) bir nesneye yeni özellikler eklenmesini sağlar.
- Özellikleri kalıtım yolu dışında composition ve delegation ile de alınabilmesini sağlar.
- open-closed prensibinin uygulandığı tasarım desendir.

### 2 Add New Weapons

```
// Yeni bir Silah ekleneceği zaman bu sınıf gibi bir sınıf eklenmesi gerekmektedir.
public class SniperRifle extends CustomWeapons {
    ArmoredSuits armoredSuits;

    public SniperRifle(ArmoredSuits armoredSuits) {
        this.armoredSuits = armoredSuits;
    }

    public String getDescription() {
        return armoredSuits.getDescription() + ", SniperRifle";
    }

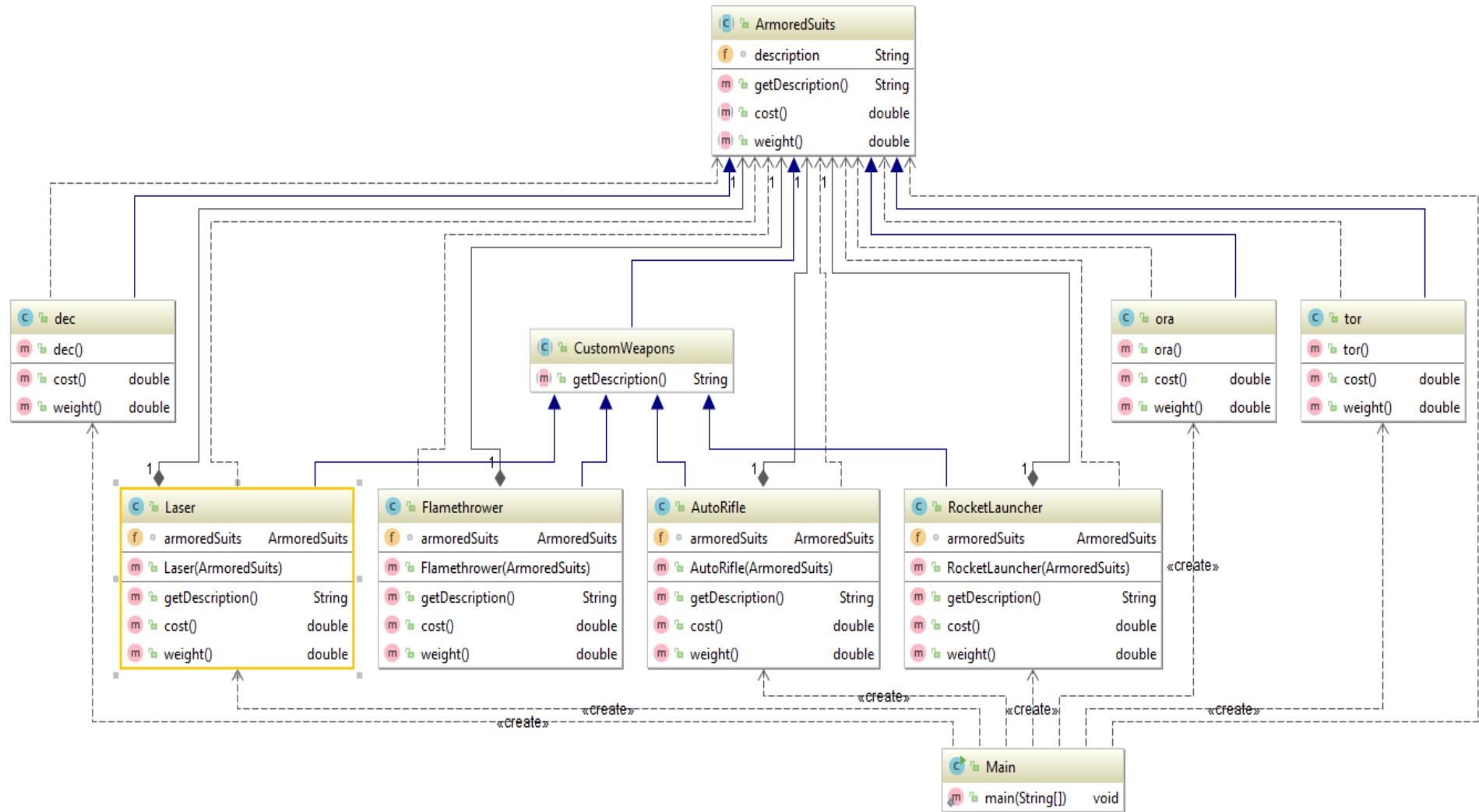
    @Override
    public double cost() {
        return 100 + armoredSuits.cost();
    }
    // Maliyeti bir önceki alınan silahların maliyetinin toplamı ile return edilir.
    }

    @Override
    public double weight() {
        return 5 + armoredSuits.weight();
    }
    // Ağırlığı bir önceki alınan silahların ağırlığının toplamı ile return edilir.
    }
}
```

### 3 Add New ArmoredSuit

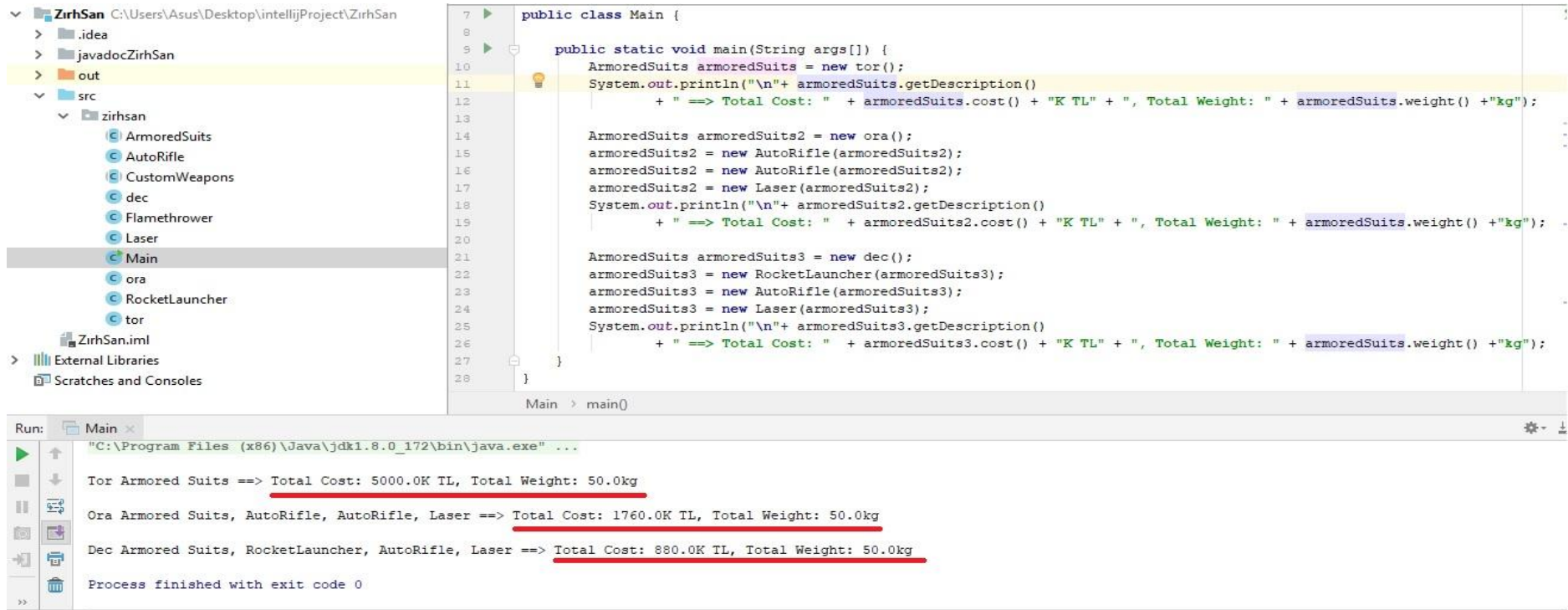
```
// Yeni bir ArmoredSuit ekleneceği zaman bu sınıf gibi bir sınıf eklenmesi ge-  
rekmetedir.  
public class camouflage extends ArmoredSuits {  
  
    public camouflage() {  
        description = "Camouflage Armored Suit";  
        // description kısmına hangi Zırh elbisesini belirlediysek ismi yazılır.  
    }  
  
    public double cost() {  
        return 200;  
        // Maliyeti return edilir.  
    }  
  
    @Override  
    public double weight() {  
        return 15;  
        // Ağırlığı return edilir.  
    }  
}
```

## Class Diagrams



## 4 Result

ZırhSan proje içerisinde bulunan Main.java içerisinde test edilecek şekilde tasarladıktan sonra IntelliJ projesini çalıştırmamız yeterlidir. Aşağıda sonuçların ve Main.java dosyasının içerisindeki ekran görüntüsü mevcuttur.



```
public class Main {  
    public static void main(String args[]) {  
        ArmoredSuits armoredSuits = new tor();  
        System.out.println("\n"+ armoredSuits.getDescription()  
            + " ==> Total Cost: " + armoredSuits.cost() + "K TL" + ", Total Weight: " + armoredSuits.weight() + "kg");  
  
        ArmoredSuits armoredSuits2 = new ora();  
        armoredSuits2 = new AutoRifle(armoredSuits2);  
        armoredSuits2 = new AutoRifle(armoredSuits2);  
        armoredSuits2 = new Laser(armoredSuits2);  
        System.out.println("\n"+ armoredSuits2.getDescription()  
            + " ==> Total Cost: " + armoredSuits2.cost() + "K TL" + ", Total Weight: " + armoredSuits2.weight() + "kg");  
  
        ArmoredSuits armoredSuits3 = new dec();  
        armoredSuits3 = new RocketLauncher(armoredSuits3);  
        armoredSuits3 = new AutoRifle(armoredSuits3);  
        armoredSuits3 = new Laser(armoredSuits3);  
        System.out.println("\n"+ armoredSuits3.getDescription()  
            + " ==> Total Cost: " + armoredSuits3.cost() + "K TL" + ", Total Weight: " + armoredSuits3.weight() + "kg");  
    }  
}
```

Run: Main x

```
"C:\Program Files (x86)\Java\jdk1.8.0_172\bin\java.exe" ...  
Tor Armored Suits ==> Total Cost: 5000.0K TL, Total Weight: 50.0kg  
Ora Armored Suits, AutoRifle, AutoRifle, Laser ==> Total Cost: 1760.0K TL, Total Weight: 50.0kg  
Dec Armored Suits, RocketLauncher, AutoRifle, Laser ==> Total Cost: 880.0K TL, Total Weight: 50.0kg  
Process finished with exit code 0
```

## Answer – 3

### Answer – 3.a

#### 1 METHOD

TAI projesini tasarlarlarken Factory Tasarım Desenini kullanarak tasarladım. Bu tasarım deseni, creational tasarım desenlerinden biridir. Bu tasarım deseni bir nesne yaratmak için arayüz sağlar, fakat hangi sınıftan nesne yaratılacağını, alt sınıfların belirlemesine olanak tanır. Factory Tasarım Deseni, super sınıf ve alt sınıfların olduğu bir uygulamada, alt sınıfların yaratılma işlemini client yani istemci sınıfında yapılmasını engellemek için kullanılır.

**Not:** Üst sınıfta implement edilmemiş bir metod bulunacağı için bu sınıfın türü ya abstract tanımlanmalı ya da interface olmalıdır.

Factory Tasarım Deseni için gereken en önemli özellikler:

- Türü abstract veya interface olan bir süper(base) fabrika sınıfı
- En az bir tane alt fabrika sınıfı
- En az bir tane Product(ürün) sınıfı
- Test sınıfı (Main)

#### 2 Add New Plane Model

```
// TAI' de üretilecek uçak modeli belirlenerek TAIPlanesStore' dan extends edilmelidir.
public class ATAK extends TAIPlanesStore{

    // CreatePlane metodu Uçak modelinin serisine göre Override edilmelidir.
    @Override
    Plane createPlane(String item) {
        if (item.equals("111")) {
            return new ATAK111();
        }
    }
}
```





## 4 Result

TAI proje içerisinde bulunan Main.java içerisini test edilecek şekilde tasarladıktan sonra IntelliJ projesini çalıştırmamız yeterlidir. Aşağıda sonuçların ve Main.java dosyasının içerisinin ekran çıktısı mevcuttur.

```
-----  
--- Making a TPX 100 ---  
Prepare TPX 100  
Creating the skeleton...  
Adding engine...  
Adding seating:  
  
---- Plane Features ----  
Purpose: Domestic flights  
Skeleton: Aluminum alloy  
Engine: Single jet engine  
Seating: 50 seats  
Prepared Plane: TPX 100  
  
-----  
--- Making a TPX 200 ---  
Prepare TPX 200  
Creating the skeleton...  
Adding engine...  
Adding seating:  
  
---- Plane Features ----  
Purpose: Domestic and short international flights  
Skeleton: Nickel alloy  
Engine: Twin jet engines  
Seating: 100 seats  
Prepared Plane: TPX 200  
  
-----  
--- Making a TPX 300 ---  
Prepare TPX 300  
Creating the skeleton...  
Adding engine...  
Adding seating:  
  
---- Plane Features ----  
Purpose: Transatlantic flights  
Skeleton: Titanium alloy  
Engine: Quadro jet engines  
Seating: 250 seats  
Prepared Plane: TPX 300  
  
-----
```

```
public class Main {  
    public static void main(String[] args) {  
        System.out.println("-----");  
        TAIPlanesStore tpx = new TPX();  
        Plane plane;  
        plane = tpx.orderPlane( type: "100");  
        System.out.println("Prepared Plane: " + plane.getModel() + "\n");  
        System.out.println("-----");  
        plane = tpx.orderPlane( type: "200");  
        System.out.println("Prepared Plane: " + plane.getModel() + "\n");  
        System.out.println("-----");  
        plane = tpx.orderPlane( type: "300");  
        System.out.println("Prepared Plane: " + plane.getModel() + "\n");  
        System.out.println("-----");  
    }  
}
```

## Answer – 3.b

### 1 METHOD

TAI' in international market yolunda ilerlemesinden dolayı Factory Tasarım Deseni'nin bir benzeri olan Abstract Factory Tasarım Deseni ile tekrardan tasarladım ve implement ettim. Abstract Factory Tasarım Deseni, creational tasarım desenlerinden biridir. Bu tasarım deseni birbiriyle alakalı veya bağımlı nesnelerin somut sınıflarını belirtmeden, yaratılması için gereken bir arayüz sağlar. Ayrıca bu desene fabrikaların fabrikası(factories of the factory) da denir. Her bir Product alt sınıfları için bir fabrika sınıfı oluşturmak gereklidir. Bu oluşturulacak fabrika sınıfları ise türü interface veya abstract olan bir süper fabrika sınıfından türemelidir.

Abstract Factory Tasarım Deseni için gereken en önemli özellikler:

- Türü abstract, interface veya normal sınıf olan bir süper sınıf
- En az bir tane alt sınıf
- Bir tane süper abstract factory sınıfı
- En az bir tane alt factory sınıfı
- Bir tane Fabrika üretici sınıfı
- Test sınıfı (Main)

### 2 Add New IngredientFactory

```
// Yeni eklenecek Ingredient Factory Plane Ingredient Factory' den implements edilmelidir.  
public class AsiaIngredientFactory implements PlaneIngredientFactory {  
  
    @Override  
    public EngineInjection createEngineInjection() {  
        return new RocketFan();  
    }  
  
    @Override  
    public SeatingCover createSeatingCover() {  
        return new Satin();  
    }  
}
```

### 3 Add New EngineInjection and SeatingCover

```
// Yeni eklenecek motor tipi Engine Injection' dan implements edilmelidir.
public class RocketFan implements EngineInjection {
    @Override
    public String toString() {
        return "RocketFan";
    }
}

// Yeni eklenecek motor tipi Engine Injection' dan implements edilmelidir.
public class Satin implements SeatingCover {
    @Override
    public String toString() {
        return "Satin";
    }
}
```

### 4 Add New Store

```
// Yeni eklenecek store TAIPlanesStore sinifindan extends edilmelidir.
public class AsiaStore extends TAIPlanesStore{

    @Override
    Plane createPlane(String item) {
        Plane plane = null;
        PlaneIngredientFactory planeIngredientFactory =
new AsiaIngredientFactory();
        if (item.equals("111")) {
            plane = new ATAK111(planeIngredientFactory);
            plane.setModel("ATAK111");
        }
        return plane;
    }
}
```

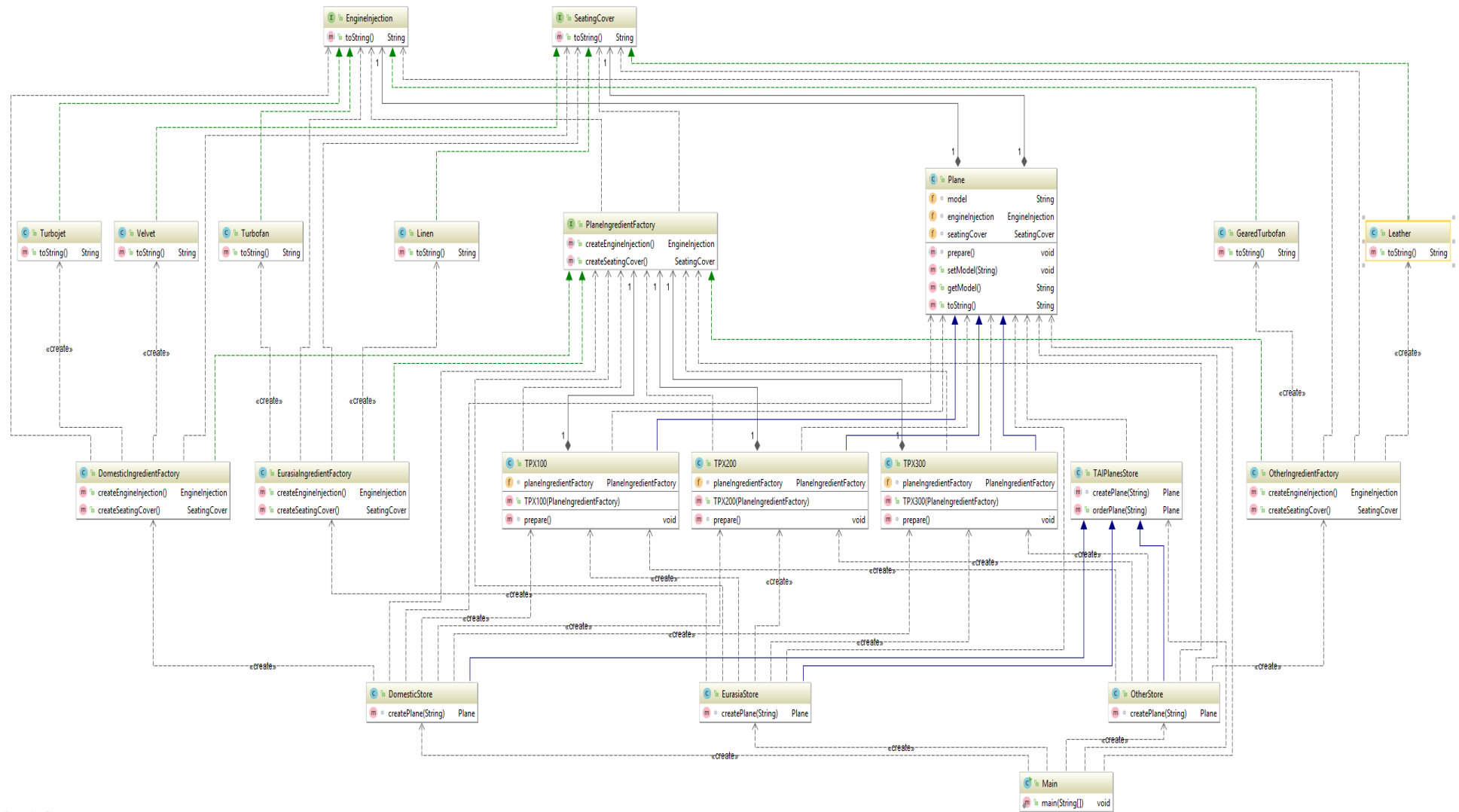
### 5 Add New Plane

```
// Yeni eklenecek uçak modeli Plane sinifindan extends edilir.
public class ATAK111 extends Plane {
    PlaneIngredientFactory planeIngredientFactory;

    public ATAK111 (PlaneIngredientFactory planeIngredientFactory){
        this.planeIngredientFactory = planeIngredientFactory;
    }

    @Override
    void prepare() {
        System.out.println("Preparing " + model);
        engineInjection = planeIngredientFactory.createEngineInjection();
        seatingCover = planeIngredientFactory.createSeatingCover();
    }
}
```

## Class Diagrams



## 6 Result

TAI\_AbstractFactory proje içerisinde bulunan Main.java içerisinde test edilecek şekilde tasarladıktan sonra IntelliJ projesini çalıştırmamız yeterlidir. Aşağıda sonuçların ve Main.java dosyasının içerisinde ekran çıktısı mevcuttur.

```
-----  
--- Making a TPX100 ---  
Preparing TPX100  
  
---- Prepared Plane ----  
---- TPX100 ----  
Turbojet  
Velvet
```

```
-----  
--- Making a TPX200 ---  
Preparing TPX200  
  
---- Prepared Plane ----  
---- TPX200 ----  
Turbofan  
Linen
```

```
-----  
--- Making a TPX300 ---  
Preparing TPX300  
  
---- Prepared Plane ----  
---- TPX300 ----  
Geared Turbofan  
Leather
```

```
-----  
Process finished with exit code 0
```

```
public class Main {  
    public static void main(String[] args) {  
        System.out.println("-----");  
        TAIPlanesStore tpx1 = new DomesticStore();  
        TAIPlanesStore tpx2 = new EurasiaStore();  
        TAIPlanesStore tpx3 = new OtherStore();  
        Plane plane;  
        plane = tpx1.orderPlane( type: "100");  
        System.out.println("\n---- Prepared Plane ----\n" + plane.toString());  
        System.out.println("-----");  
        plane = tpx2.orderPlane( type: "200");  
        System.out.println("\n---- Prepared Plane ----\n" + plane.toString());  
        System.out.println("-----");  
        plane = tpx3.orderPlane( type: "300");  
        System.out.println("\n---- Prepared Plane ----\n" + plane.toString());  
        System.out.println("-----");  
    }  
}
```