Structural - Flyweight Design Pattern

Aynı datanın değişik objeler tarafından kullanılması için bir havuz yaratır ve RAM'i çok efektif kullanılması için tasarlanmıştır.Bellek kullanımı ile ilgili bir sıkıntı yoksa bu pattern den uzak durulması tavsiye edilir.Bir örnekte bir web sitesinde image'leri bir havuzda tutmak için kullanılabilir. 3 State'den oluşur (Extensic state - Instrinsic state - Factory);

- Extensic state yani her instance'da farklılık gösterebilecek alanlar ayrılmalıdır.
- Ayrılmış olan alanlar ilgili metoda parametre olarak gönderilmelidir.
- Nesne oluşturmak için bir factory class oluşturulmalıdır.

Ilk olarak interface create ediliyor;

```
interface Vehicle{
   void assingColor(String color);
   void startEngine();
}
```

Interface'imizi inherit edecek olan class.

- Extensic State her instance'de farklılık gösterecek alanlar. (not shareable)
- Intrinsic State her instance'da aynı olacak alanlar (shareable object)

Birinci Aracımız;

```
class Truck implements Vehicle{
   private final String MAXSPEED; //intrinsic
   private String color; //Extensic
   public Truck(){
      MAXSPEED = "120km/s";
   }
   @Override
   public void assingColor(String color) {
      this.color = color;
   }
   @Override
   public void startEngine() {
      System.out.println(color+ " colored Truck with max speed : " + MAXSPEED);
   }
}
```

Ikinci Aracımız;

```
class Cycle implements Vehicle{
   private final String MAXSPEED;
   private String color;
   public Cycle() {
      MAXSPEED = "20km/s";
   }
   @Override
   public void assingColor(String color) {
      this.color=color;
   }
   @Override
   public void startEngine() {
      System.out.println(color+ " colored Cycle with max speed : " + MAXSPEED);
   }
}
```

Factory ;

```
class VehicleFactory{
   //type of vehicle (truck,cycle or taxi)
   private static HashMap<String,Vehicle> typeOfVehicleCollection = new HashMap<>();
   //vehicle already exist?
   public static Vehicle getVehicle(String vehicleType){
      Vehicle vehicle = null;
      if (VehicleFactory.typeOfVehicleCollection.containsKey(vehicleType)){
         vehicle = VehicleFactory.typeOfVehicleCollection.get(vehicleType);
      }else{
         switch(vehicleType){
            case "Cycle" :
               System.out.println("Cycle is created");
               vehicle = new Cycle();
               break;
            case "Truck" :
               System.out.println("Truck is created");
               vehicle = new Truck();
               break;
            default:throw new IllegalArgumentException("Vehicle type not exist");
         }
         VehicleFactory.typeOfVehicleCollection.put(vehicleType,vehicle);
      }
      return vehicle;
   }
}
```

HashMap içerisinde instance'i create edilmiş olan Vehicle interface'inden inherit edilmiş concrete class'ların listesini tutuyoruz. getVehilce methodumuz vehicle'ın concrete class'larının instance'larının create edilip edilmediğini kontrol ederek eğer instance'ları create edilmemişse create ediyor ve HashMap'imize ekliyor

```
class Test{
   public static void main(String[] args) {
      Vehicle cycle = VehicleFactory.getVehicle("Cycle");
      cycle.assingColor("Blue");
      cycle.startEngine();
      cycle.assingColor("Black");
      cycle.startEngine();
      Vehicle cycleTwo = VehicleFactory.getVehicle("Cycle");
      cycleTwo.assingColor("Blue");
      cycleTwo.startEngine();
      Vehicle truckOne = VehicleFactory.getVehicle("Truck"); //Truck instance'i initialize edildi
      Vehicle truckTwo = VehicleFactory.getVehicle("Truck"); //Truck instance'i zaten yukarıda create edildiği için
bir kez daha initialize edilmedi
   }
}
```

```
"C:\Program Files\Java\jdk1.8.0_231\bin\java.exe" ...
Cycle is created
Blue colored Cycle with max speed : 20km/s
Black colored Cycle with max speed : 20km/s
Blue colored Cycle with max speed : 20km/s
Truck is created

Process finished with exit code 0
```