GEBZE TECHNICAL UNIVERSITY

HW2

REPORT

Barış ŞAHİN

151044016

1-)

A-) 1. What happens if someone tries to clone a Singleton object using the `clone()` method inherited from Object? Does it lead to the creation of a second distinct Singleton object? Justify your answer. **(5 points)**

Object Sınıfına ait clone() metodu protected statüsündedir ve Clonable arayüzünü implement etmediğimiz veya clone fonksiyonunu sarmalayan bir public fonksiyon yazılmadığı sürece kullanılamaz. Yani ekstra bir çaba sarfetmeden ikinci bir obje yaratılamaz.

B-) 2. Clonin Singletons should not be allowed. How can you prevent the cloning of a Singleton object? **(5 points)**

Yukarıda da belirttiğim gibi Clonable implement edilmediği veya clone metodunu sarmalayan public bir metod yazılmadığı sürece klonlanamaz. Bu konu hakkında herhangi bir şey yapmazsak zaten önlenmiş olur.

Let's assume the class Singleton is a subclass of class Parent, that fully implements the Cloneable interface. How would you answer questions 1 and 2 in this case? **(5 points)**

```
/**
 *
 * Parent class that implements cloneable interface
 *
 * */

public class Parent implements Cloneable{
    @Override
    protected Object clone() throws CloneNotSupportedException
    {
        return super.clone();
    }
}
```

C-) 1. What happens if someone tries to clone a Singleton object using the `clone()` method inherited from Object? Does it lead to the creation of a second distinct Singleton object? Justify your answer. **(5 points)**

Cloneable implement edildiyse malesef yeni bir Singleton objesi oluşur.

```
public final class Singleton extends Parent{
    private static Singleton instance;
    public String value;

    private Singleton(String value) {
        // I put a sleep in initialization for observation.
        try {
            Thread.sleep(1000);
        } catch (InterruptedException ex) {
            ex.printStackTrace();
        }
        this.value = value;
        System.out.println("Singleton created(Constructor)");
    }

    public static Singleton getInstance(String value) {
        if (instance == null) {
            System.out.println("Singleton created(getInstance)");
            instance = new Singleton(value);

        }
        return instance;
    }
}
```

```
System.out.println("Illegal cloning experiment >:)");
Singleton molly = (Singleton) singleton.clone();

System.out.println("\nSingleton hashCode:- "
        + singleton.hashCode());
System.out.println("Molly hashCode:- "
        + molly.hashCode());
```

```
"C:\Program Files\Java\jdk1.8.0_141\bin\java.exe" ...
If you see the same value, then singleton was successfully implemented
If you see different values, then 2 singletons were created (Fail)

RESULT:

Singleton created(getInstance)
Singleton created(Constructor)
TestObj1
TestObj1
Illegal cloning experiment >:)
singleton hashCode:- 1163157884
molly hashCode:- 1956725890

Process finished with exit code 0
```

B-) 2. Clonin Singletons should not be allowed. How can you prevent the cloning of a Singleton object? **(5 points)**

Clone metodunu Override ederek bu sorunu çözebiliriz. Yeni halinde exception fırlatabiliriz ya da aynı objeyi geri döndürebiliriz.

```java
public final class Singleton extends Parent{
    private static Singleton instance;
    public String value;

    private Singleton(String value) {
        // I put a sleep in initialization for observation.
        try {
            Thread.sleep( millis: 1000);
        } catch (InterruptedException ex) {
            ex.printStackTrace();
        }
        this.value = value;
        System.out.println("Singleton created(Constructor)");
    }

    @Override
    protected Object clone() throws CloneNotSupportedException {
        throw new CloneNotSupportedException("Its singleton");
    }

    public static Singleton getInstance(String value) {
        if (instance == null) {
            System.out.println("Singleton created(getInstance)");
            instance = new Singleton(value);

        }
        return instance;
    }
}
```

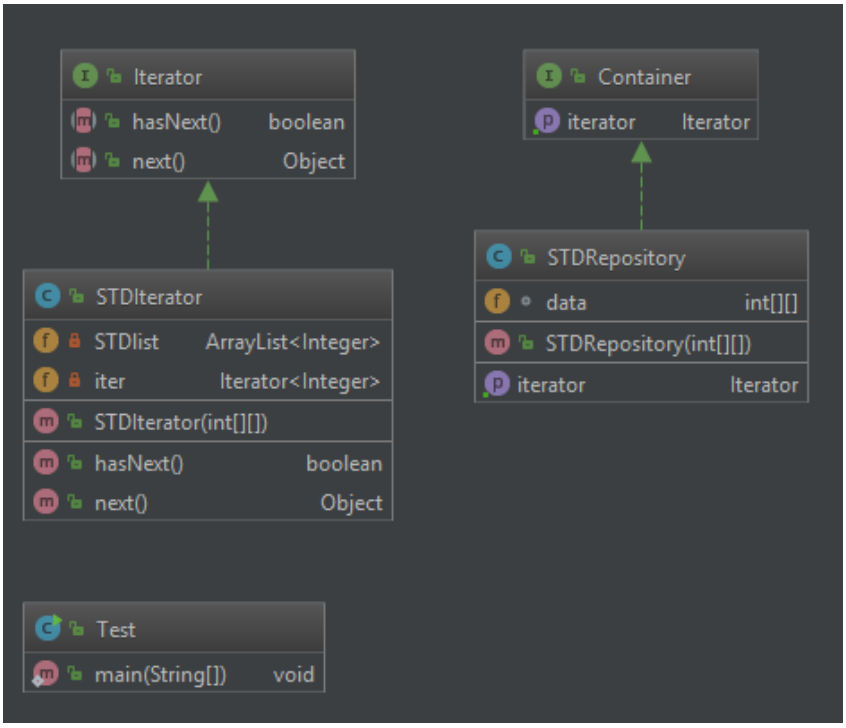
```
If you see the same value, then singleton was successfully implemented
If you see different values, then 2 singletons were created (Fail)

RESULT:

Singleton created(getInstance)
Exception in thread "main" java.lang.CloneNotSupportedException: Its singleton
Singleton created(Constructor)
    at Singleton.clone(Singleton.java:18)
TestObj1
    at Test.main(Test.java:13)
TestObj1
Illegal cloning experiment >:)

Process finished with exit code 1
```

Diğer yöntemin sonuçları Q1/SS klasörü içinde bulunabilir.

İmplementasyondaki son hali aynı objeyi döndürecek şekilde bırakılmıştır.

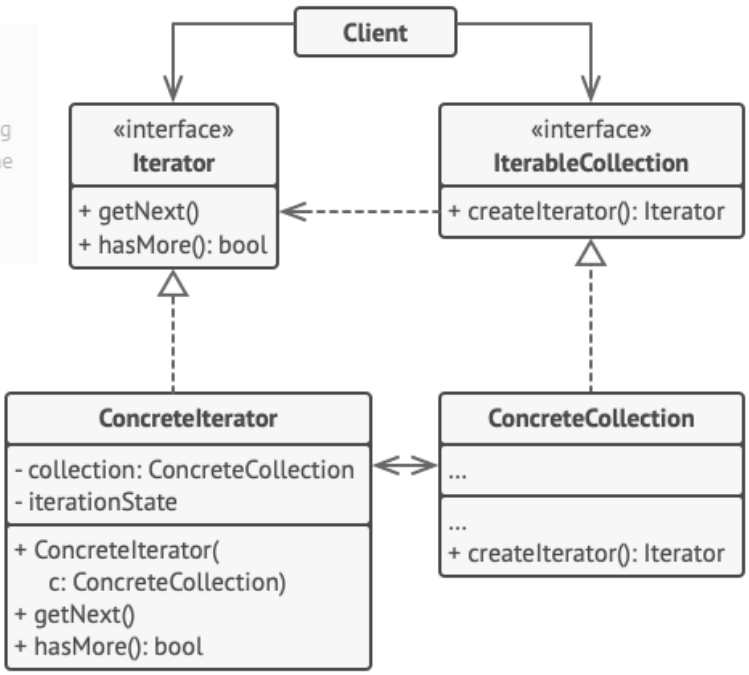
2-)

Sınıf Diyagramı

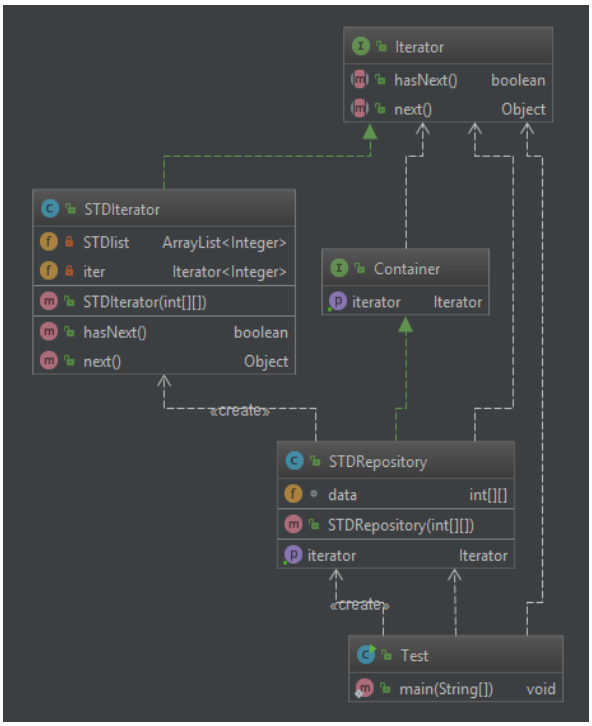


Bizden 2 boyutlu bir listeyi belirtilen şekilde dolaşabilmek için İteratör Tasarım Örüntüsünü implement etmemiz istenmiştir.

Iteratör Tasarım Örüntüsünün



UML Diyagramı



Yukarıdaki sayfada iteratör tasarım örüntüsünün genel diyagramı gösterilmiştir.

Bunu bizim implementastonumuzla eşleştirirsek

Iterator Interface →Iterator Interface
Concrete Iterator → STDIterator
Iterable Collection → Container Interface
Concrete Collection → STDRepository

Bize saat yönü veya saat yönünün tersine olmak üzere iki seçenek verilmişti. Ben saat yönüne göre implement ettim.

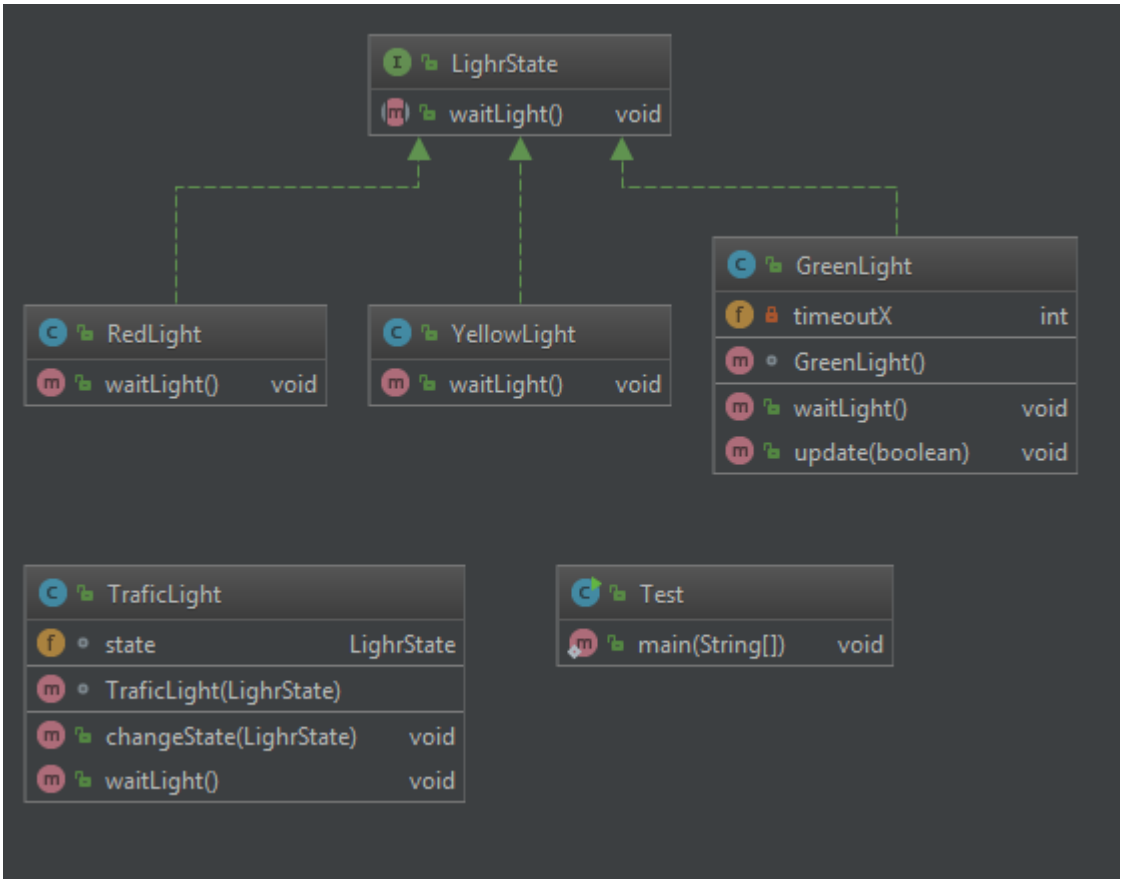
Test ve sonuçları aşşağıda gösterilmiştir

```
STDRepository testRepository = new STDRepository(testdata);
System.out.print("ORDER: ");
for(Iterator iter = testRepository.getIterator(); iter.hasNext();){
    int x = (int)iter.next();
    System.out.print(x + " ");
}
```

```
Satellite Transmits Data
1  2  3  4
5  6  7  8
9  10 11 12
13 14 15 16
ORDER: 1 2 3 4 8 12 16 15 14 13 9 5 6 7 11 10
Process finished with exit code 0
```

3-)

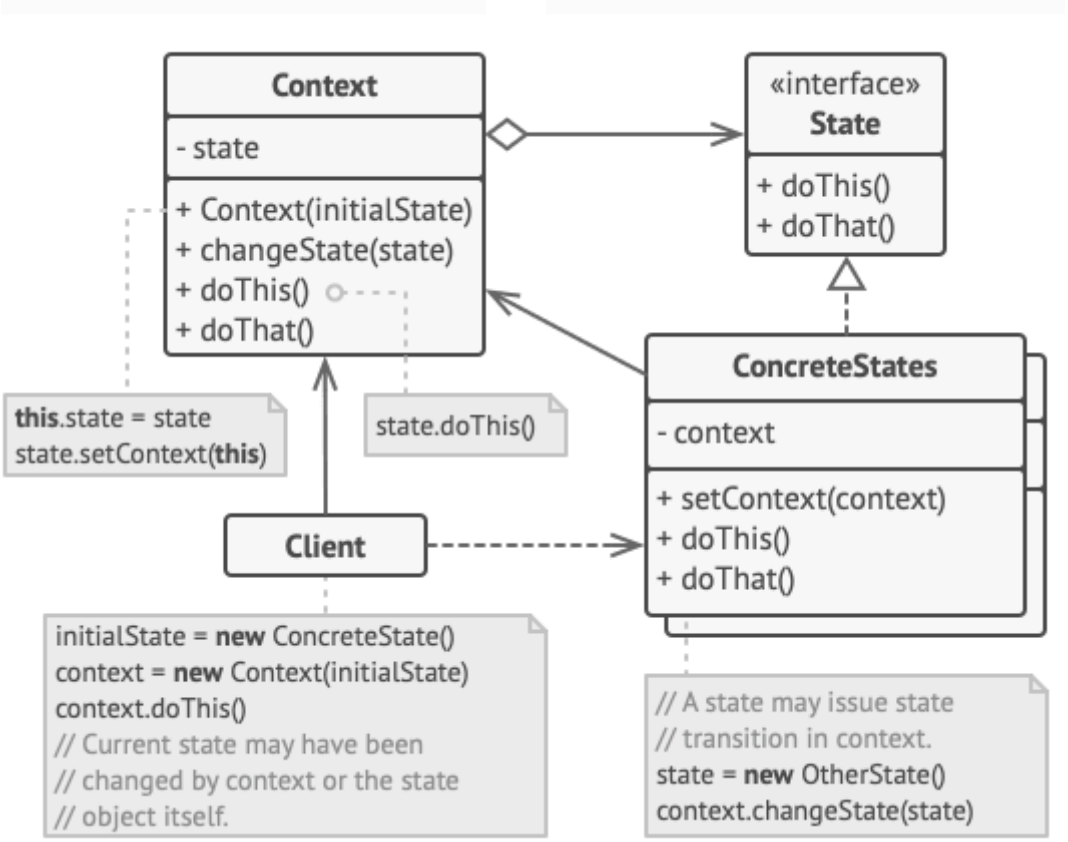
Sınıf Diyagramı



a) RED: switches to GREEN after 15 seconds.
b) YELLOW: switches to RED after 3 seconds.
c) GREEN: switches to YELLOW after 60 seconds (timeout_X).

Bizden yukarıda verilen durumları State Pattern kullanarak implement etmemiz istenmişti.(State Diyagramları Q3/SS klasörünün içinde bulabilirsiniz)

State Tasarım Örüntüsü Genel Diyagramı



Yukarıda State tasarım örüntüsünün genel diyagramı gösterilmiştir.

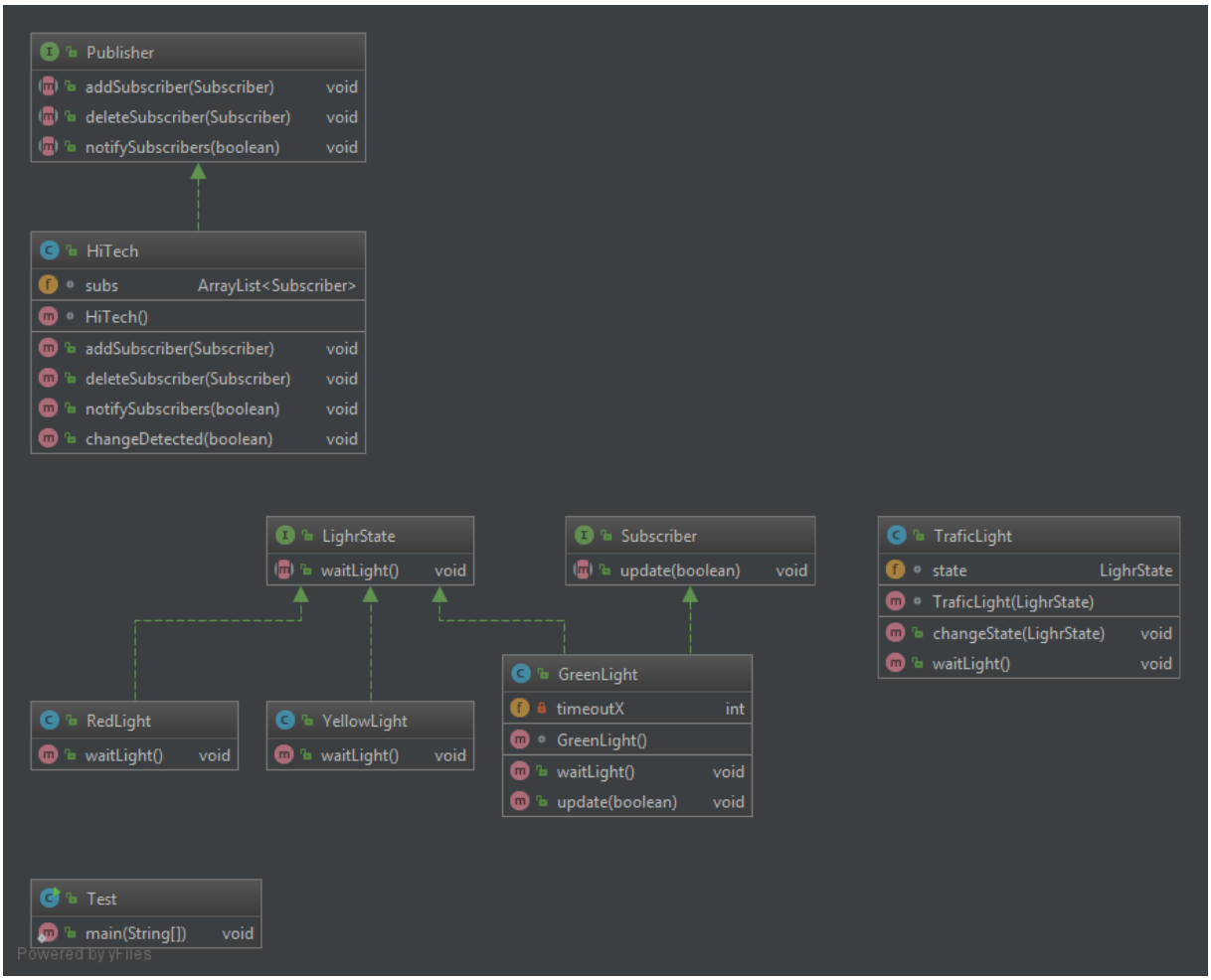
Bunu bizim implementastonumuzla eşleştirirsek

Context →TraficLight
State Interface → LightState Interface
Concrete States → RedLight, YellowLight, GreenLight

İkinci kısımda ise Yeşil Işığın, trafiğin sıkışık olduğu durumlarda daha uzun süre yanması gerektiği ve bunun bize verilen HiTech sınıfını Publisher olarak kabul edip Observer Tasarım Örüntüsü kullanıralarak implement edilmesi gerektiği söylenmiştir.

Sınıf diyagramımız şu şekilde değimiştir

Sınıf Diyagramı V2



HiTech sınıfını publisher ve GreenLight Sınıfını Subscriber olarak tanımladım. Böylelikle HiTech her trafik yoğunluğu değiştiğinde GreenLight a haber verecek ve GreenLight da değşimin ne yönde olduğuna göre bekleme süresini ayarlayabilecek.

Bu programı test ederken 100 defa dönen bir while döngüsü kullandım.Bu döngü içinde verilen sıra ile stateler arasında geçiş yaptım. Her 25 dönüşte bir

Bu programı test ederken 100 defa dönen bir while döngüsü kullandım.Bu döngü içinde verilen sıra ile stateler arasında geçiş yaptım. Her 25 dönüşte bir trafiği yoğunsa normal ,normal ise yoğun olacak şekilde değiştirdim. Testin uzun sürmemesi adına bize verilen sürelerin %1 i kadar beklemektedir.

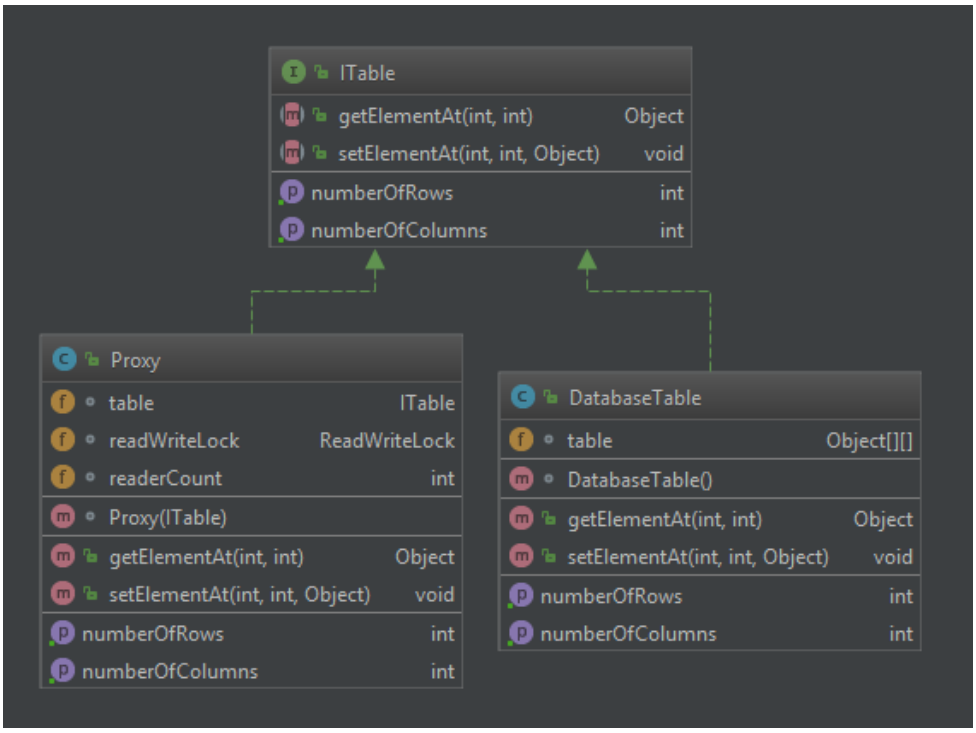
Test sonuçları

```
--------------------------Loop 0 starts--------------------------

Red Light waiting for 15 sec
Red Light waiting for 60 sec
Yellow Light waiting for 3 sec
Traffic has increased substantially
--------------------------Loop 1 starts--------------------------

Red Light waiting for 15 sec
Red Light waiting for 90 sec
Yellow Light waiting for 3 sec
--------------------------Loop 2 starts--------------------------
 --------------------------Loop 24 starts--------------------------

 Red Light waiting for 15 sec
 Red Light waiting for 90 sec
 Yellow Light waiting for 3 sec
 --------------------------Loop 25 starts--------------------------

 Red Light waiting for 15 sec
 Red Light waiting for 90 sec
 Yellow Light waiting for 3 sec
 Traffic runs normaly
 --------------------------Loop 26 starts--------------------------

 Red Light waiting for 15 sec
 Red Light waiting for 60 sec
 Yellow Light waiting for 3 sec
```

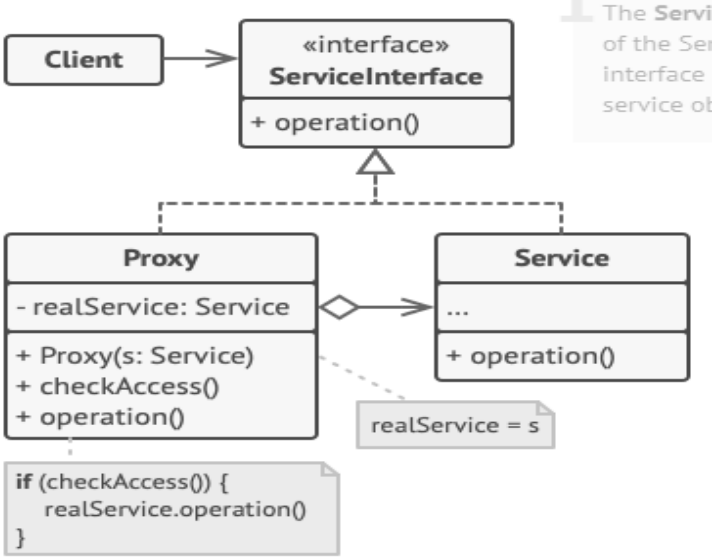
4-)

Bizden senkronizasyon problemi olan eski bir sınıfı Proxy Tasarım örüntüsü kullanarak senkron hale getirmemiz istenmiştir.

Sınıf Diyagramı



Proxy Tasarım Örüntüsü Genel Diyagramı



Yukarıdaki sayfada Proxy tasarım örüntüsünün genel diyagramı gösterilmiştir.

Bunu bizim implementastonumuzla eşleştirirsek

Service Interface →ITable Interface
Service → DatabaseTable
Proxy → Proxy

İkinci kısmında ise Readerlara öncelik vermemiz istenmiştir.

```
@Override
public Object getElementAt(int row, int column) {
    readerCount ++;
    if(readerCount == 1)
        readWriteLock.writeLock().lock();
    readWriteLock.readLock().lock();
    Object returnVal = table.getElementAt(row,column);
    readWriteLock.readLock().unlock();
    readerCount--;
    if(readerCount == 0)
        readWriteLock.writeLock().unlock();
    return returnVal;

}
@Override
public void setElementAt(int row, int column, Object o) {
    readWriteLock.writeLock().lock();
    table.setElementAt(row,column,o);
    readWriteLock.writeLock().unlock();
}
```