Object Oriented Analaysis & Design

HOMEWORK 2 REPORT

Ahmet Yuşa Telli
151044092 | AYTELLİ@GTU.EDU.TR

Part 1:

In this part, we have Singleton Design Pattern.

Let say this is our Cloneable Parent Class:

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

This is simple Singleton class implementation: (this Singleton class from the book without extends)

```
public class Singleton extends CloneableParent {
   private volatile static Singleton unique;

   private Singleton () { }

   public static Singleton getInstance()
   {
      if (unique == null)
      {
         synchronized (Singleton.class)
         {
            if (unique == null){
               unique = new Singleton();
            }
         }
      }
      return unique;
   }
}
```

The object cloning is a way to create exact copy of an object. In Java, clone() method of Object class is used for cloning. The method extends Object class, object of any class including Singleton class can call clone() method.

1. If we want to create a clone:

If we do not override clone() method in Singleton Class, in main function we can create two different Singleton objects. We can see two different hash codes.

2. If we **do not** want to create a clone:

First, we can override clone() method in Singleton class implementation. We do not create a new singleton object. This method must throw “CloneNotSupportedException” :

```
@Override
protected Object clone() throws CloneNotSupportedException{
  throw new CloneNotSupportedException();
}
```

OR

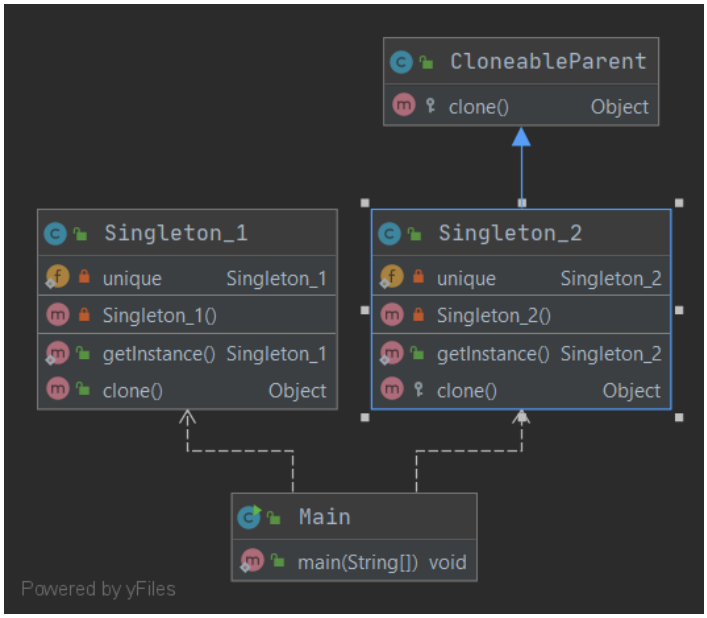
```
@Override
protected Object clone() throws CloneNotSupportedException  {
   return unique;
}
```

For the questions:

1) As we see above, Cloneable super class can create a new singleton class object and we have different singleton objects. It leads to create a second distinct Singleton object.

2) If we do not create a second distinct Singleton object, we must override clone() method in Singleton class implementation. The method throws “CloneNotSupportedExtection”. Or, we can give back the one object which name is ‘unique’ in code.

If singleton class extends from Object. We have to implement clone() method in the class. We can do what we want. We can return new Singleton object, or we can return same object.

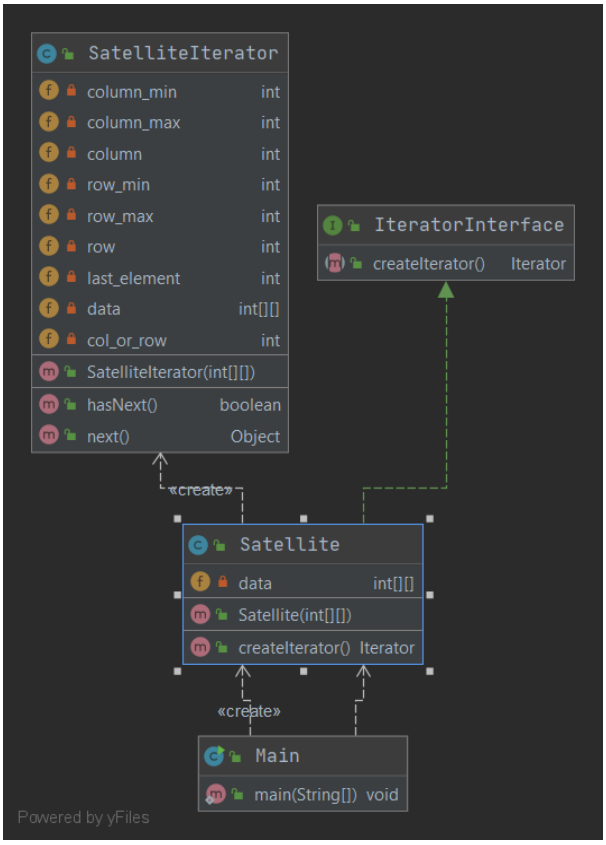


Part 2:

In this part we implemented an Iterator design pattern. We take 2D array data from satellite and print clockwise.

We have a IteratorInterface and it has a createIterator() method. The method returns an Iterator object. We have two classes. One is Satellite and implements from this IteratorInterface. In this class; it takes the data and override createIterator() method. This method returns new SatelliteIterator object.

The second class is SatelliteIterator class, it implements from Iterator. It takes the date from Satellite class. In this class we override hasNext() and next() methods. We move in data using this iterator class.



In the SatelliteIterator class, we have some flags for the clockwise. We are moving on rowmin, rowmax, columnmin and columnmax ways.

Part 3:

In the third part we have two design patterns: State and Observer.

In State design pattern, we have Light interface. It has three methods which are RedOn, GreenOn, YellowOn. We create three classes: RedLight, GreenLight, YellowLight.

In RedLight class, we override GreenOn method and we set the traffic light to Green.

In GreenLight class, we override YellowOn method and we set the traffic light to Yellow.

In YellowLight class, we override RedOn method and we set the traffic light to Red.

These classes take a TrafficLight object. This class implements from Observer. We will talk about later.

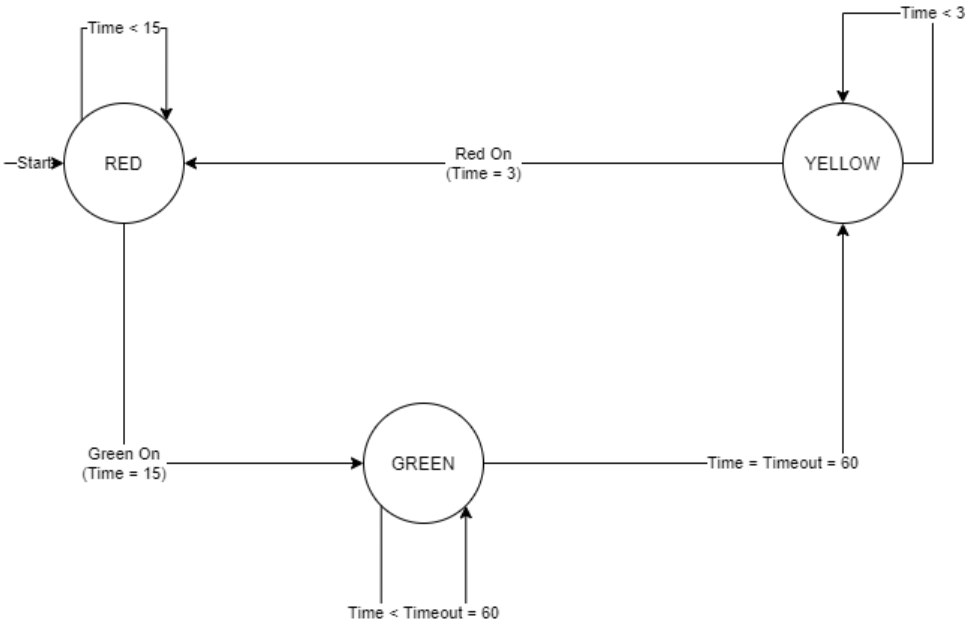
In TrafficLight class, we have objects of classes of states and one current state object. We design the light's seconds in the changeLight() method.

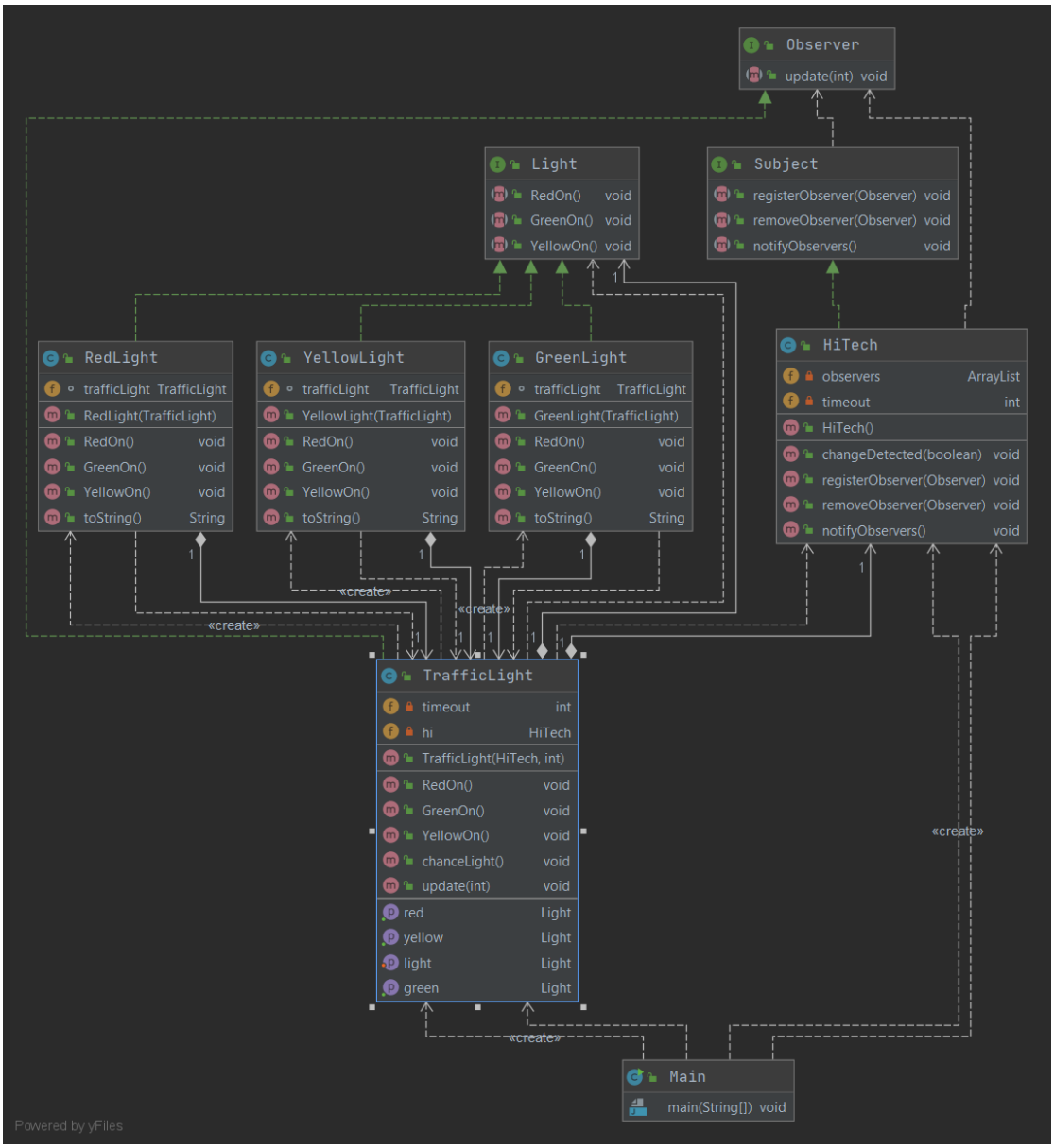
I thought, the red light is on in 15 seconds and the green light is on 60 seconds and the yellow light is on 3 seconds. Total second is 78. One period is 78 seconds. For instance, the yellow light is on at 76^{th} seconds and is off at 78^{th} seconds. The green light is on at 16^{th} seconds and off at 75^{th} seconds.

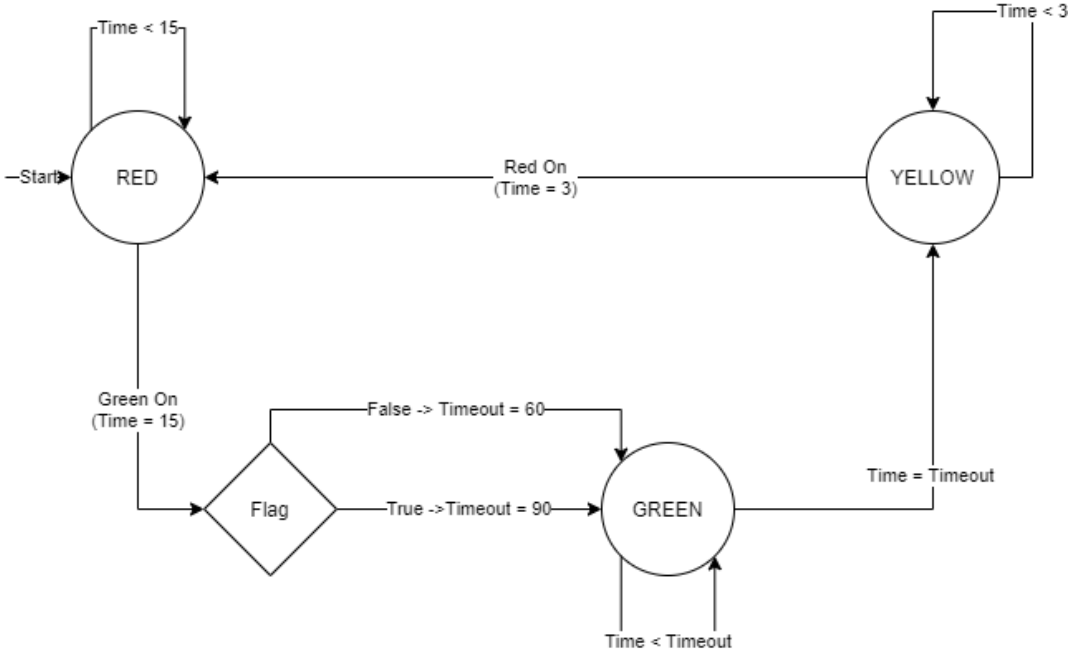
The second design is Observer Pattern. We have Observer and Subject interfaces. In Observer interface we have one update() method. In subject interface, we have registerObserver(), removeObserver() and notifyObservers () methods.

The TrafficLight class implements from Observer. In this class, we override update() method and take the timeout and we have a HiTech instance. We register to observer with this instance.

The HiTech class implements from Subject interface. It has an arraylist for observers. We override Subject's methods. We have a changeDetected() method. We call this method in main, and it chance the timeout then notify observers. Registered observers take the timeout and update the green time.




Observer
update(int) void
Light
RedOn() void
GreenOn() void
YellowOn() void
Subject
registerObserver(Observer) void
removeObserver(Observer) void
notifyObservers() void
RedLight
trafficLight TrafficLight
RedLight(TrafficLight)
RedOn() void
GreenOn() void
YellowOn() void
toString() String
YellowLight
trafficLight TrafficLight
YellowLight(TrafficLight)
RedOn() void
GreenOn() void
YellowOn() void
toString() String
GreenLight
trafficLight TrafficLight
GreenLight(TrafficLight)
RedOn() void
GreenOn() void
YellowOn() void
toString() String
HiTech
observers ArrayList
timeout int
HiTech()
changeDetected(boolean) void
registerObserver(Observer) void
removeObserver(Observer) void
notifyObservers() void
TrafficLight
timeout int
hi HiTech
TrafficLight(HiTech, int)
RedOn() void
GreenOn() void
YellowOn() void
chanceLight() void
update(int) void
red Light
yellow Light
light Light
green Light
Main
main(String[]) void
«create»
Powered by yFiles


Time < 15
Time < 3
Start
RED
Red On
(Time = 3)
YELLOW
Green On
(Time = 15)
False -> Timeout = 60
Flag
True ->Timeout = 90
GREEN
Time = Timeout
Time < Timeout

Part 4:

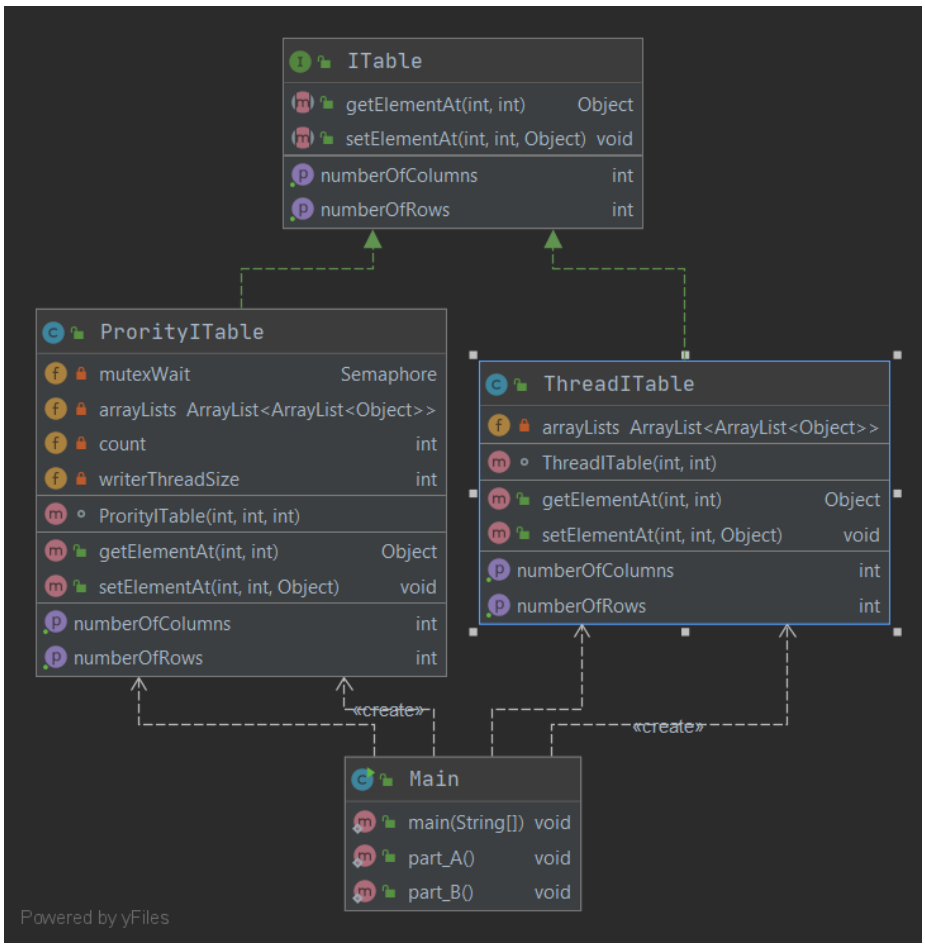
In the last part, we design a Proxy Pattern. In this pattern, we have an interface which name is ITable. It has four methods. Then, we create two classes from this interface.

First class is ThreadITable. We have an array list with two dimensions. Because the methods have row and column values. The array list's type is object. We override the four methods. But we added "synchronized" keyword at the beginning of the methods. Because lots of threads can try to use these methods.

Second class is PriorityITable. For this class, we do not want to getElementAt threads run before setElementAt threads. We want to run setElementAt threads first. We use a Semaphore for this solution. First, we take the setElementat thread size. If getElementAt run first, we check the setElementAt threads count. If all setElementAt threads finish, we let getElementAt threads. If setElementAt threads have not finished yet, we run setElementAt thread before getElementAt threads.

In the test main function, we create 10 threads for ThreadITable class. 5 for setElemetnAt, 5 for getElementAt. All threads want to use row 0 and column 0. All threads start to run in a loop.

10 different thread we use for ProrityITable. 5 for setElemetnAt, 5 for getElementAt. All threads want to use row 0 and column 0. We call first get threads but they do not run, set threads run first.



Ahmet Yuşa Telli 151044092