CSE443 HW2 REPORT

PART-1

Question 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.

Answer 1)

```
public class Singleton {

    private static Singleton uniqueİnstance;
    private Singleton() {};

    public static Singleton getInstance(){

        if(uniqueİnstance==null){
            uniqueİnstance= new Singleton();
        }
        return uniqueİnstance;
    }

    public static void main(String[] args)throws
CloneNotSupportedException{

        Singleton c= Singleton.getInstance();
        c.clone();

    }
}
```

It is assumed that the singleton object is wanted to be copied with clone() method in the main method of the singleton class because If it is tried to be copied in another class, the error "'clone () has protected access in' java.lang.Object" appears. Therefore, it is necessary to override the clone method explicitly and must implement Clonable interface by Singleton. That's why I made the assumption.

If you want to clone the Singleton object as in the main method above, after making the assumption, "java.lang.CloneNotSupportedException: Singleton" error will be received. **So a second Singleton object cannot be created**. Because we dont implement Clonable interface, clone() method generates CloneNotSupportedException. The Cloneable interface must be implemented by the class whose object clone we want create (But we don't want to clone a singleton object anyway, so this rule works in this case and prevents cloning).

Question 2)

Clonin Singletons should not be allowed. How can you prevent the cloning of a Singleton object?

Answer 2) As I said above, if our singleton class does not implement the Cloneable interface, our singleton instances cannot be cloned**. So there is no need to do anything to prevent clone with this form of the sigleton class.**

Let's Assume Singleton class is a subclass of Parent class that fully implements the Cloneable interface. In this case answer1 and answer2 will be like;

Answer1)

Yes In this case, a second singleton object may occur. In such a case, when the clone method is called, no exception will occur. Becase parent class implements Cloneable interface and all methods of it. Thus, the rule "The Cloneable interface must be implemented by the class whose object clone we want create" is ensured in Singleton class by inheritance.

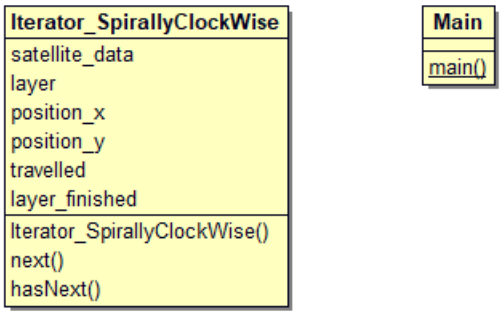
Answer 2)

In order to prevent the singleton object from being cloned, we can override the clone() method and throw an exception such as RunTimeException in the method we override.

PART-2

There is not much to say for this party. I did not write an extra class for getting satellite data. Because data is just an array. So as mentioned in the homework pdf, I used a 2-dimensional array to iterate through the iterator. I am not getting input from the user. I am testing the functionality with different arrays in the Main method. In the homework pdf, the class diagram is not requested for this part, but I still put it (although it is a simple diagram).

Needless to say, I used an **iterator pattern**.



PART-3

A)

Here I sent the trigger, that is the action functions explicitly while switching between states. Just like calling turnCrank (), insertQuart () in the GumbalMachine example in the book. I made the transition between states by calling action functions as after15seconds(), after60seconds(), after3seconds().

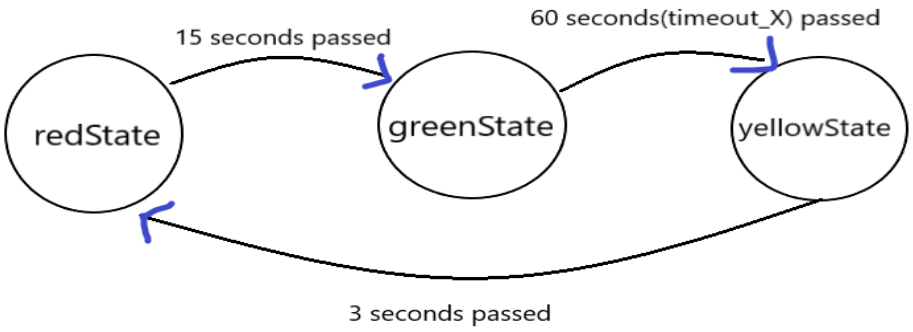
Here, instead of calling the actions explicitly, I could automatically switch the states with a timer, but then the transition between states could not be observed in the main method and the effect of calling wrong actions on states could not be observed. Also user interaction would not be obvious.

There is an interface called **State**. This interface contains the **after3seconds**, **after15seconds** and **after60seconds** methods. There are **redState**, **yellowState** and **greenState** classes that implement this **State** interface. Concrete classes that are yellowState etc. have an instace of **TrafficLight class**. TrafficLight class has all three instance of States and another states that holds current state.

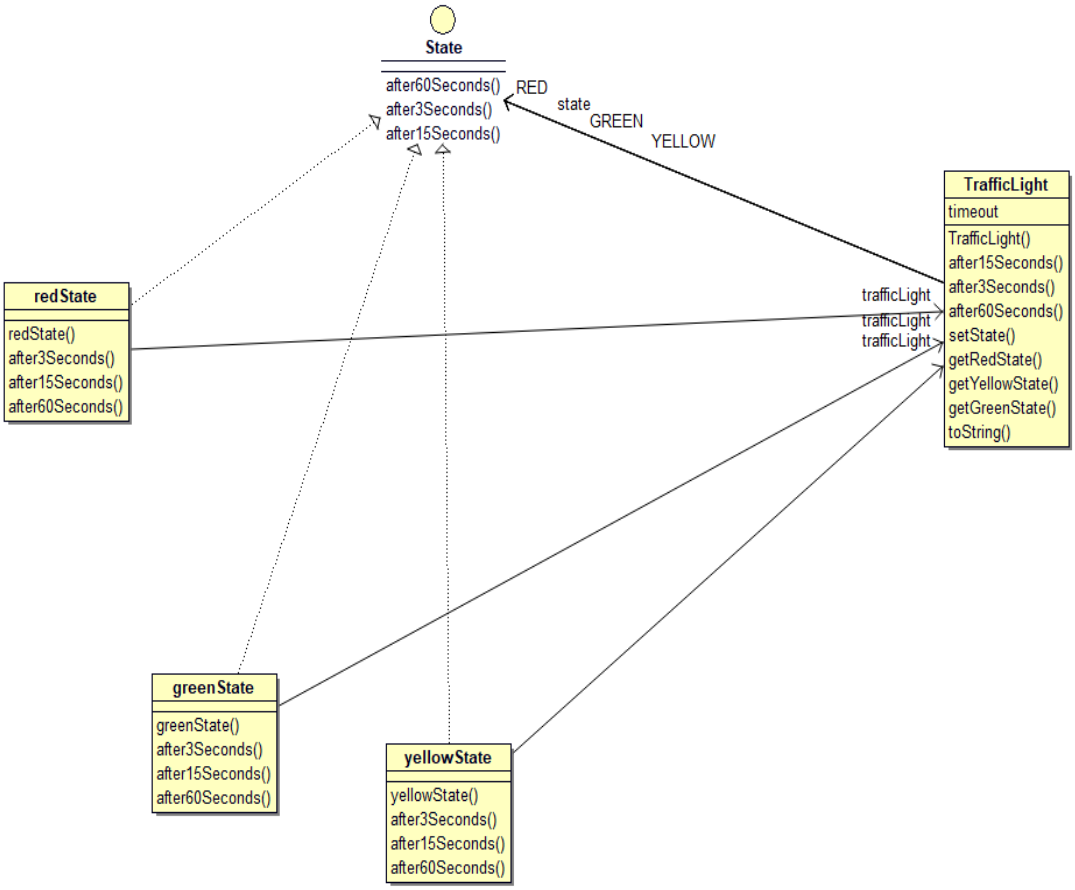
When an action function is called via the TrafficLight object, that function is called via the current state.

My drive class illustrates every state and every transition.

State Diagram-



Class Diagram-



B)

With the changeDetected method of the HiTech class, it is stated that the traffic situation has changed. In this case, HiTech class must be observable and TrafficLight class must be observer in order for HiTech class to report the change to TrafficLight class. So the HiTech class extends Observable, the TrafficLight class implements the Observer.

Since the waiting time of the green light will increase to 90 seconds when there is traffic, a new action function is written because there is now a new action **(after90seconds**).

```
HiTech detectTraffic= new HiTech();
TrafficLight trafficLight = new TrafficLight(detectTraffic);
```

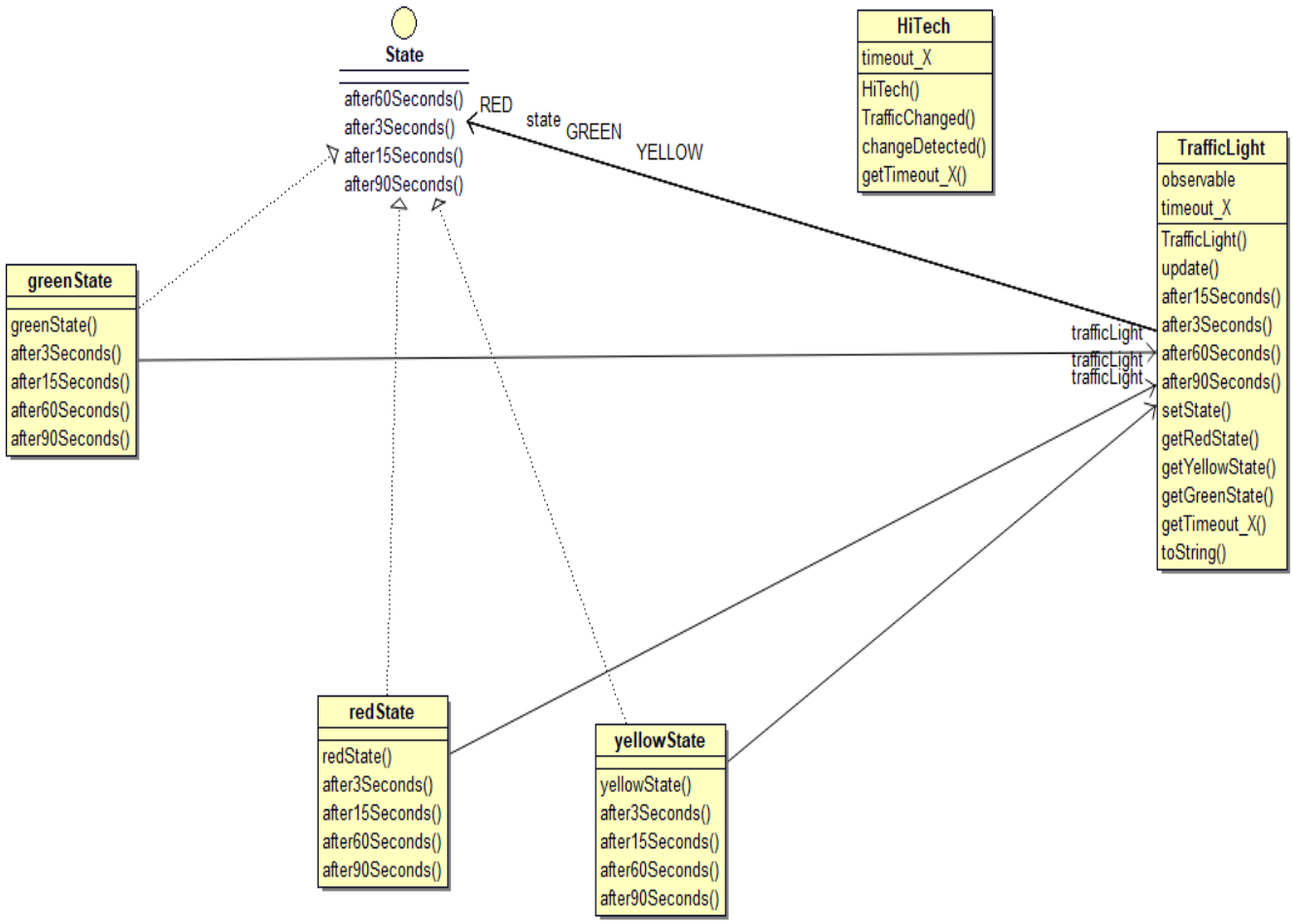
-This is how we subscribe TrafficLight to HiTech to observe traffic changes.

```
detectTraffic.changeDetected(true);
```

-This is how to change traffic situation to high traffic.

In one light state, the correct action function must be called to switch to the other light, otherwise an error message is printed on the terminal.

Class Diagram-



PART-4

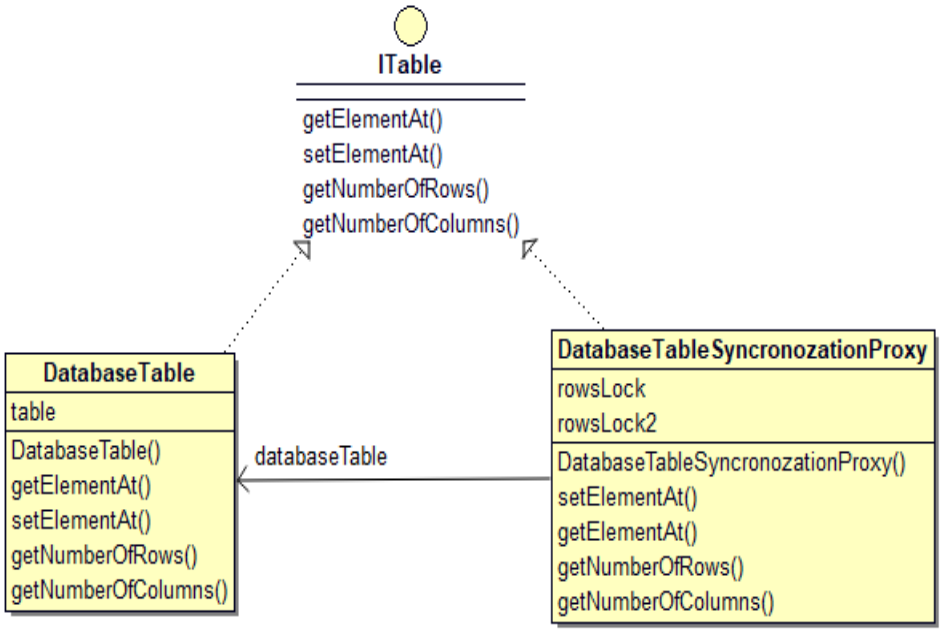
A)

I have a class called DatabaseTableSyncronozationProxy. This class implements the ITable interface. It contains the DatabaseTable class, which we assume is ready for us. When an operation is desired to be performed on the proxy class, this operation is actually called over the DatabaseTable instance that we keep inside. But I used the synchronized keyword in set and get functions. Thus, these functions will be able to work synchronously.

I keep a separate lock object for each database row. I keep these key objects in the synchronized keyword. Thus, if the object is locked, on one method side another method cannot access this row.

In the Main method, two threads are created, each of which is supposed to be a client. Each thread (client) calls functions on the proxy object within itself, for example getElementAt (), setElementAt (). I have deliberately called functions on the same rows on both clients so that they can be clearly seen that they block each other. In the terminal outputs, it can be clearly seen that the get operation is not performed before a write operation is completed on the same row etc.

Class Diagram-



B)

I didnt implement second part of question 4.

İBRAHİM AKBULUT

151044077