# CSE 443
# Object Oriented Analysis Design

# HOMEWORK 2 REPORT

**161044083**

**Galip Tayfun Saygılı**

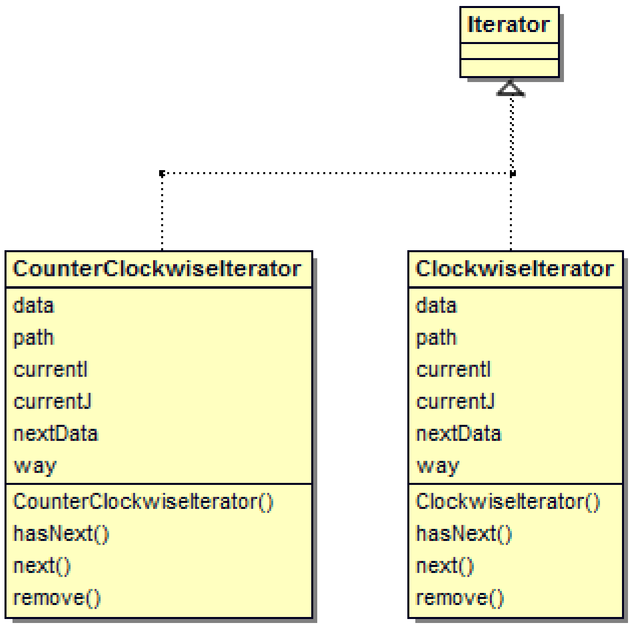
**TA: Erchan Aptoula**

## Part 1

1. If we clone a singleton object with the clone method of Object class, we will get a reference to the same object; moreover, we will have Access to the unique (static) singleton object (After we switch the Access signature of the clone method to public).
2. We may keep the Access protected (We don't switch it; clearly, we may keep using the clone method of the Object class), or we may throw an exception by switching Access to public and appending final keyword to the overriden clone method, so that any singleton object or any object which class is inhetired from Singleton class will not be able to change this situation.
3. If we clone a singleton object with the clone method that fully implemented by the Parent class, then we will get a reference to a new Singleton object, and this is something we would never want, this will destroy the Singleton design pattern requirements.
4. If the parent class did not implement the clone method as final, we may throw an exception by switching Access to public and appending final keyword to the overriden clone method, so that any singleton object or any object which class is inhetired from Singleton class will not be able to change this situation (same precautions with 2).

## Part 2

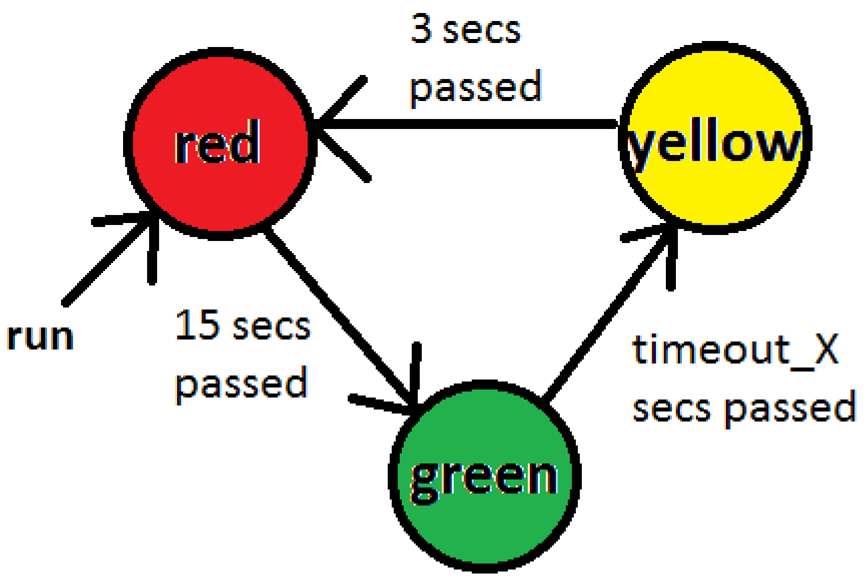
Here i used the iterator design pattern since i need to iterate a data structure.



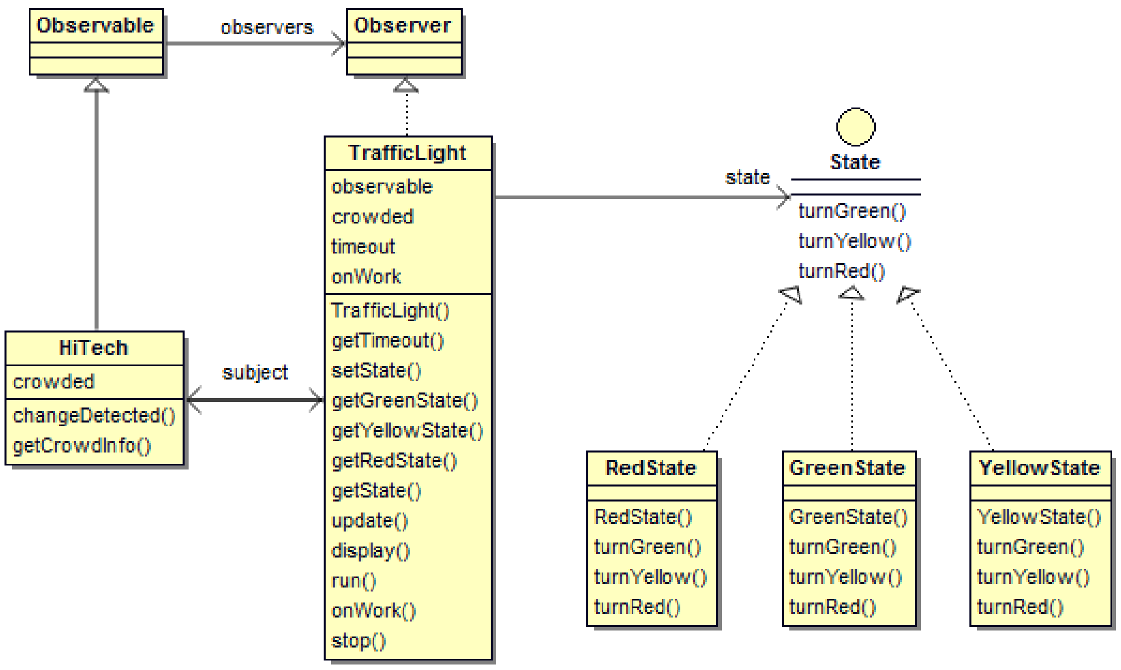
**Class Diagram of Part 2**

## Part 3

I used state diagram in this part and composed the required states in TrafficLight class with has-a relation. And I used observer design pattern to get constant traffic info from mobese (I used 1:10 scale for seconds to switch faster between states).



State Diagram of Part 3



Class Diagram of Part 3

## Part 4

I used Proxy design pattern to be able to give DataBase synchronization ability.

In part a, i just implemented write lock like this:

**Write IF no read or write.**

**Read IF no write.**

Then we cannot read or write while already writing, and we cannot write while reading, we can also read more than one at the same time.

And the output is like this:

```
"C:\Program Files\Java\jdk-11.0.9\bin\java.exe" "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA 2020.2.3\lib\idea_rt.jar=55411:C:\Program Files\JetBrains\IntelliJ IDEA 2020.2.3\bin" -Dfile.encoding=UTF-8 -classpath C:\Users\Tayfun\IdeaProjects\ooad_hw2_part4a\out\production\ooad_hw2_part4 com.company.Main

Server ready
Write thread #0 is waiting.
Write thread #0 has started.
Write thread #1 is waiting.
Read thread #0 is waiting.
Read thread #1 is waiting.
Write thread #2 is waiting.
Write thread #3 is waiting.
Read thread #2 is waiting.
Read thread #3 is waiting.
Read thread #4 is waiting.
Write thread #0 has ended.
Write thread #1 has started.
Write thread #1 has ended.
Read thread #0 has started.
Read thread #3 has started.
Read thread #2 has started.
Read thread #1 has started.
Read thread #4 has started.
Read thread #3 has ended.
Read thread #0 has ended.

Read thread #2 has ended.

Read thread #4 has ended.

Read thread #1 has ended.

response: 2

response: 2

response: 2

response: 2

Write thread #2 has started.

response: 2

Write thread #2 has ended.

Write thread #3 has started.

Write thread #3 has ended.
```

In part b, I added write priority feature to the synchronization that I implemented:

**Write IF no write or read.**

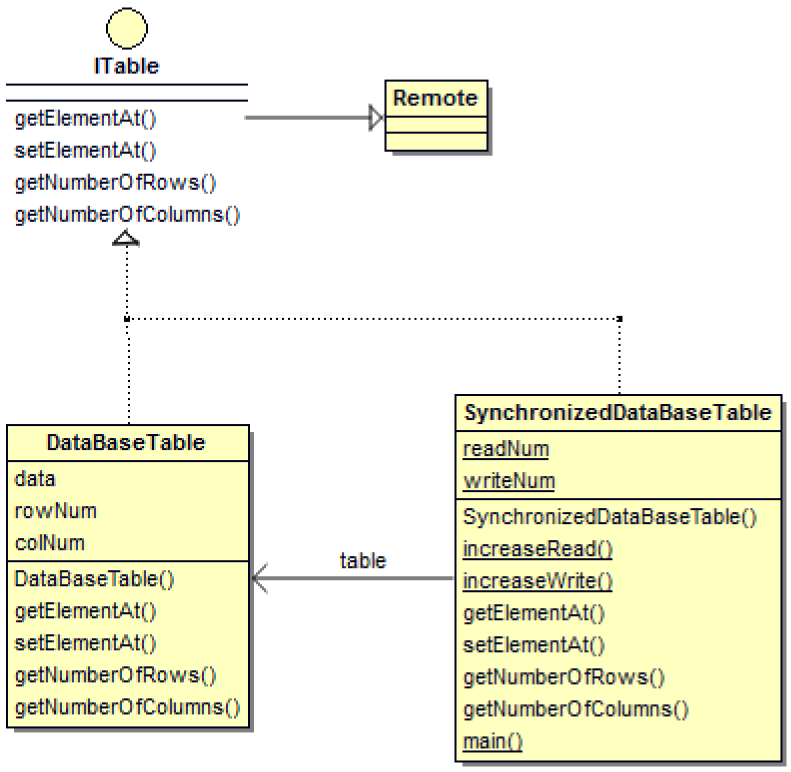
**Read IF no write or no waiting writer.**

This way we give the priority to write by also waiting for waiting writers.

After adding write priority to the synchronization that i implemented, the output is like this:

```
"C:\Program Files\Java\jdk-11.0.9\bin\java.exe" "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA 2020.2.3\lib\idea_rt.jar=55434:C:\Program Files\JetBrains\IntelliJ IDEA 2020.2.3\bin" -Dfile.encoding=UTF-8 -classpath C:\Users\Tayfun\IdeaProjects\ooad_hw2_part4b\out\production\ooad_hw2_part4 com.company.Main

Server ready

Write thread #0 is waiting.

Write thread #0 has started.

Write thread #1 is waiting.

Read thread #0 is waiting.

Write thread #2 is waiting.

Read thread #1 is waiting.

Read thread #2 is waiting.

Read thread #3 is waiting.

Write thread #3 is waiting.
```

```
Read thread #4 is waiting.
Write thread #0 has ended.
Write thread #3 has started.
Write thread #3 has ended.
Write thread #1 has started.
Write thread #1 has ended.
Write thread #2 has started.
Write thread #2 has ended.
Read thread #0 has started.
Read thread #4 has started.
Read thread #1 has started.
Read thread #3 has started.
Read thread #2 has started.
Read thread #0 has ended.
Read thread #4 has ended.
Read thread #2 has ended.
Read thread #3 has ended.
response: 4
Read thread #1 has ended.
response: 4
response: 4
response: 4
response: 4
```



**Class Diagram of Part 4**