Gebze Technical niversity

Computer Engineering

Object Oriented Analysis and Design

CSE443

Homework 2

Can BEYAZNAR

161044038

Question 1

A) 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.

We can create a copy of the object using the Clone () method. However, if we are using the singleton design pattern, duplicating an object is against this design pattern. This leads to distortion of the design pattern. Therefore, if we are using the singleton design pattern, we must prevent duplication of an object.

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

We can solve this problem by using exception. First, if we throw the "CloneNotSupportedException" exception in the clone () method used in the class of the singleton object, the user will not be able to clone the singleton object.

C) 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?

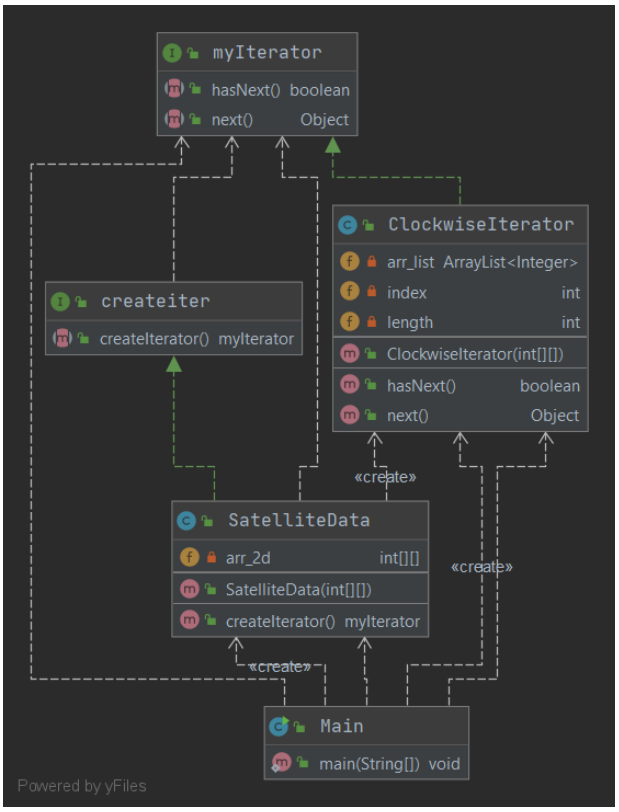
If the singleton object was implemented from the clonnable interface structure, the answer I gave to the first question would not change. This error still exists because an object could be duplicated. So this error needs to be resolved. In the second question, this error can be solved by using the "CloneNotSupportedException" exception.

Question 2

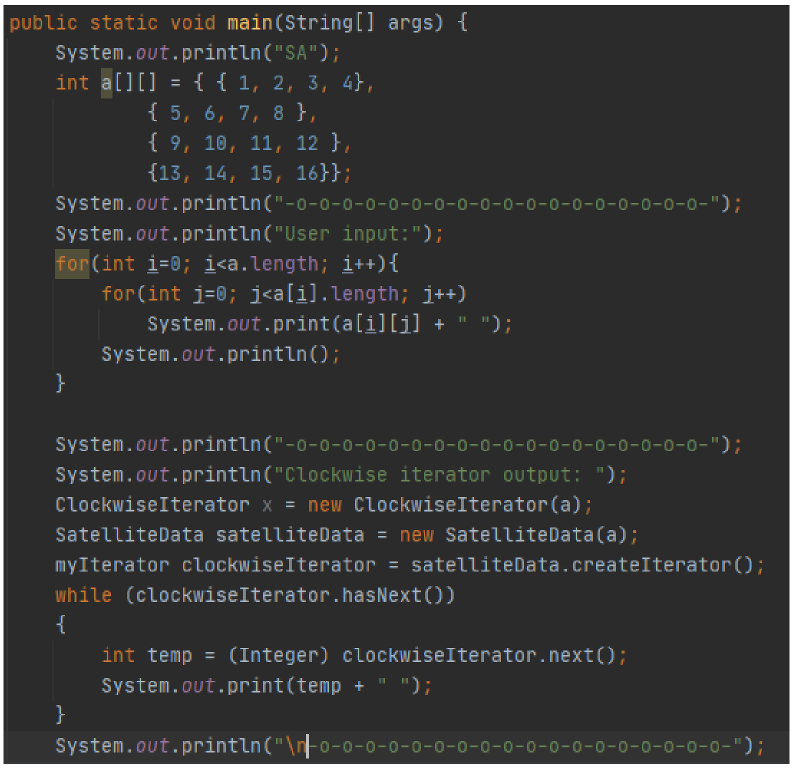
Solution

For this problem, I used the iterator design pattern. Since he wanted one method of spiraling in the assignment, I chose the clockwise one. First, I have to convert the 2 dimensional array coming from the user into a one dimensional array and it should be clockwise. For this, first of all, I have to travel as an iterator. So I am creating myIterator interface, which is my own iterator interface. Then I define the ClockwiseIterator class which implements this interface. So, using this class I will convert the 2 dimensional array from the user to the desired format. Then the user will be able to obtain a clockwise spiral array of this 2D array using the next () and hasNext () methods. But I have to somehow convert the 2D array given by the user into an iterator. For this, I'm creating the createiter interface. Thus, we will be able to convert any array we want to an iterator using this interface. If there is a new iterator method such as ClockwiseIterator in the future, we will be able to implement it more easily. Thus, we reduced the maintenance cost. Next, I created a SatelliteData class that implements the createiter interface. This class takes the 2-dimensional array from the user and calls the createIterator () method, returning the ClockwiseIterator class. Thus, we obtain the spiral clockwise output desired by the user.

The UML diagram of the method I have applied is as follows.

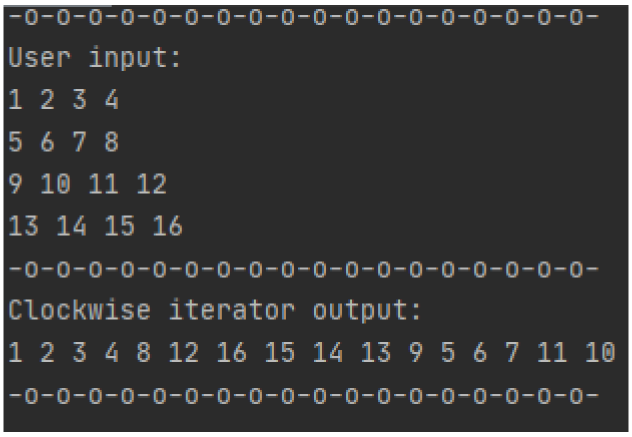


Output



```
public static void main(String[] args) {
    System.out.println("SA");
    int a[][] = { { 1, 2, 3, 4},
            { 5, 6, 7, 8 },
            { 9, 10, 11, 12 },
            {13, 14, 15, 16}};
    System.out.println("-o-o-o-o-o-o-o-o-o-o-o-o-o-o-o-o-o-o-o-");
    System.out.println("User input:");
    for(int i=0; i<a.length; i++){
        for(int j=0; j<a[i].length; j++)
            System.out.print(a[i][j] + " ");
        System.out.println();
    }

    System.out.println("-o-o-o-o-o-o-o-o-o-o-o-o-o-o-o-o-o-o-o-");
    System.out.println("Clockwise iterator output: ");
    ClockwiseIterator x = new ClockwiseIterator(a);
    SatelliteData satelliteData = new SatelliteData(a);
    myIterator clockwiseIterator = satelliteData.createIterator();
    while (clockwiseIterator.hasNext())
    {
        int temp = (Integer) clockwiseIterator.next();
        System.out.print(temp + " ");
    }
    System.out.println("\n-o-o-o-o-o-o-o-o-o-o-o-o-o-o-o-o-o-o-o-");
```



```
-0-0-0-0-0-0-0-0-0-0-0-0-0-0-0-0-0-0-0-
User input:
1 2 3 4
5 6 7 8
9 10 11 12
13 14 15 16
-0-0-0-0-0-0-0-0-0-0-0-0-0-0-0-0-0-0-0-
Clockwise iterator output:
1 2 3 4 8 12 16 15 14 13 9 5 6 7 11 10
-0-0-0-0-0-0-0-0-0-0-0-0-0-0-0-0-0-0-0-
```

Question 3

I divided the solution part of this problem into 2. One was explained in the state design pattern and the other was explained in the observer design pattern. Only the added innovations are explained in the solution part of the observer design pattern.

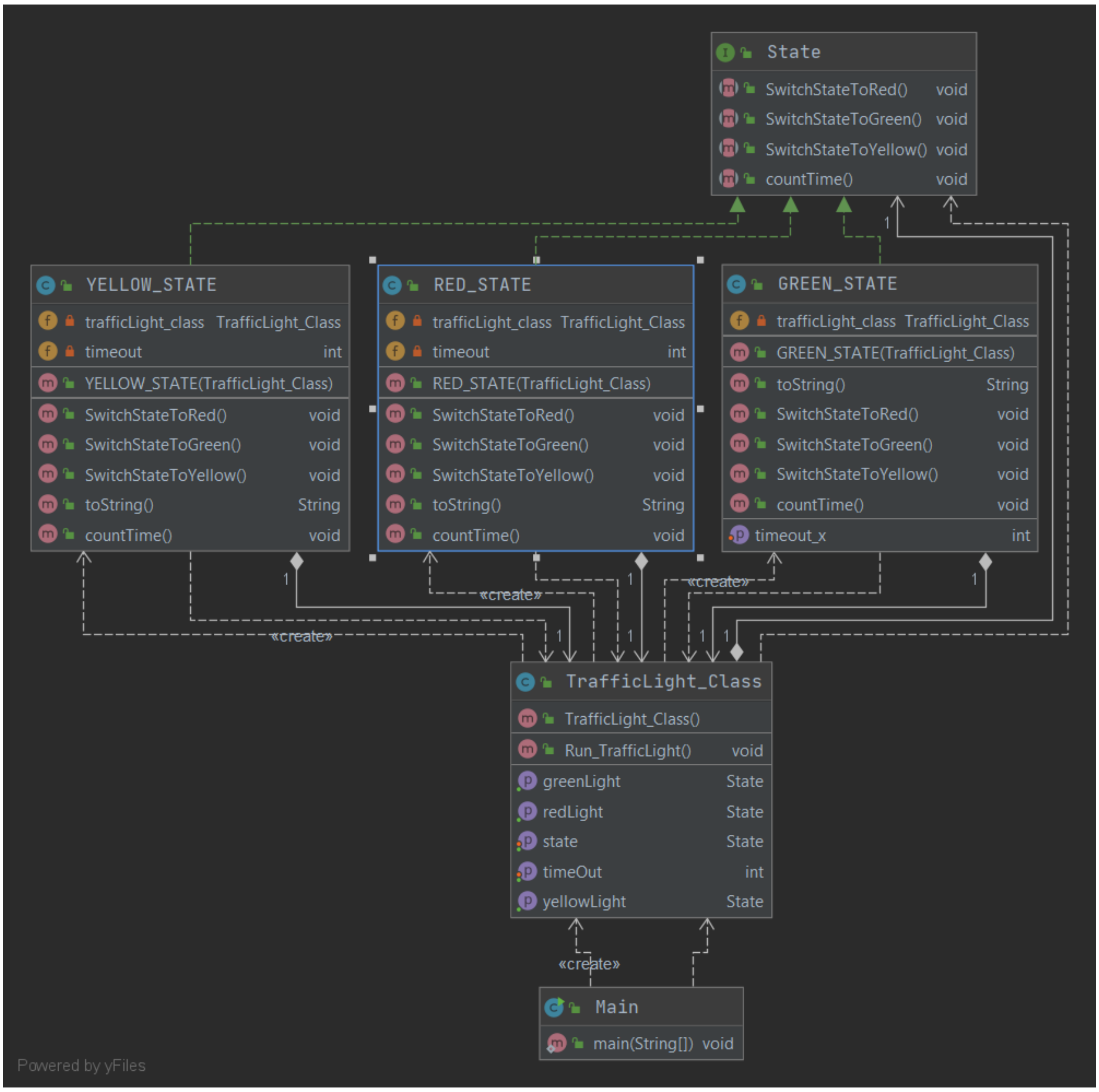
Solution (State Design Pattern)

For the first part of this problem (timeout_x value is fixed) I used the state design pattern. First, I created the State interface to change the traffic light to red, yellow, and green.

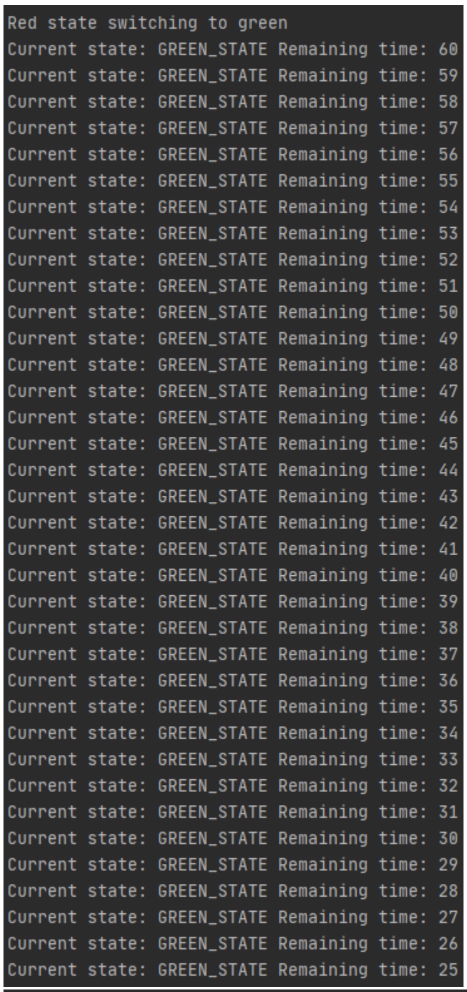
The state interface will contain functions consisting of 3 different states to provide these transformations. These methods are SwitchStateToRed, SwitchStateToGreen, and SwitchStateToYellow. In addition, I wrote another method called countTime () to count the time in these lights. So, using this interface, we can count the time and switch to the traffic light color we want. Then I wrote the class of Green, red and yellow states. These three classes keep the TrafficLight_Class class as a parameter to operate in the traffic light they work on. The TrafficLight_Class class will be explained in the following sections. The only difference between these three classes is related to the SwitchStateToRed, SwitchStateToGreen and SwitchStateToYellow functions. In each of the 3 classes, only one of these methods is implemented inside. the other 2 methods are left blank. For example, the Green light will only turn yellow, so only the SwitchStateToYellow method is filled in.

If we come to the TrafficLight_Class class, this class represents the traffic light. It holds 4 state classes. RedLight, GreenLight, YellowLight. The last class is the state parameter, which represents the current state of the traffic light. Thus, it will be sufficient to assign these parameters to the state parameter when switching to red, yellow and green lights. Here the function simulating the traffic light is the Run_TrafficLight method. In order to avoid code crowds in the Main section, a traffic light has been run in this method. In this method, first the green light turns on and the time starts and then switches to the other light state when the time is over. And it goes like this.

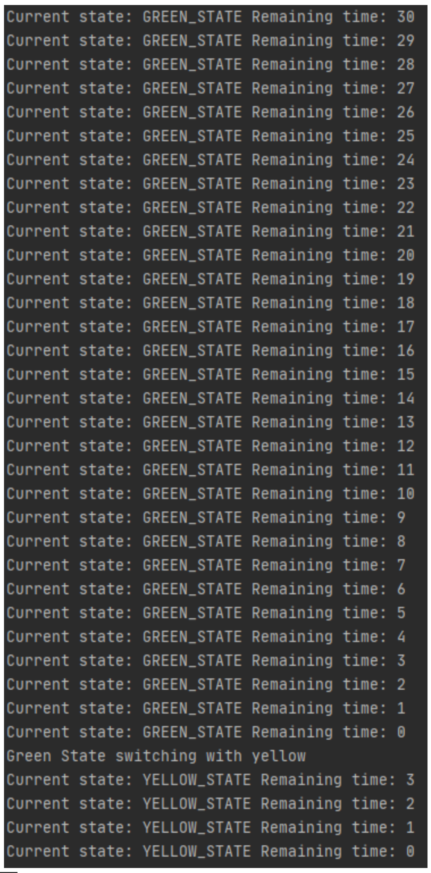
The UML diagram of the method I have applied is as follows.



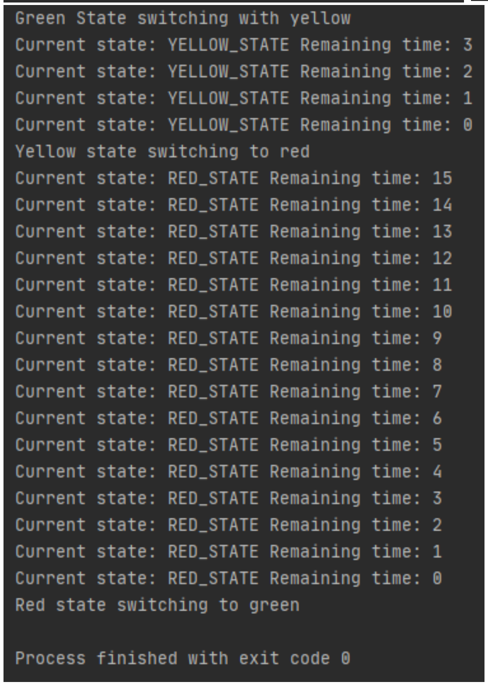
Output



```
Red state switching to green
Current state: GREEN_STATE Remaining time: 60
Current state: GREEN_STATE Remaining time: 59
Current state: GREEN_STATE Remaining time: 58
Current state: GREEN_STATE Remaining time: 57
Current state: GREEN_STATE Remaining time: 56
Current state: GREEN_STATE Remaining time: 55
Current state: GREEN_STATE Remaining time: 54
Current state: GREEN_STATE Remaining time: 53
Current state: GREEN_STATE Remaining time: 52
Current state: GREEN_STATE Remaining time: 51
Current state: GREEN_STATE Remaining time: 50
Current state: GREEN_STATE Remaining time: 49
Current state: GREEN_STATE Remaining time: 48
Current state: GREEN_STATE Remaining time: 47
Current state: GREEN_STATE Remaining time: 46
Current state: GREEN_STATE Remaining time: 45
Current state: GREEN_STATE Remaining time: 44
Current state: GREEN_STATE Remaining time: 43
Current state: GREEN_STATE Remaining time: 42
Current state: GREEN_STATE Remaining time: 41
Current state: GREEN_STATE Remaining time: 40
Current state: GREEN_STATE Remaining time: 39
Current state: GREEN_STATE Remaining time: 38
Current state: GREEN_STATE Remaining time: 37
Current state: GREEN_STATE Remaining time: 36
Current state: GREEN_STATE Remaining time: 35
Current state: GREEN_STATE Remaining time: 34
Current state: GREEN_STATE Remaining time: 33
Current state: GREEN_STATE Remaining time: 32
Current state: GREEN_STATE Remaining time: 31
Current state: GREEN_STATE Remaining time: 30
Current state: GREEN_STATE Remaining time: 29
Current state: GREEN_STATE Remaining time: 28
Current state: GREEN_STATE Remaining time: 27
Current state: GREEN_STATE Remaining time: 26
Current state: GREEN_STATE Remaining time: 25
```



```
Current state: GREEN_STATE Remaining time: 30
Current state: GREEN_STATE Remaining time: 29
Current state: GREEN_STATE Remaining time: 28
Current state: GREEN_STATE Remaining time: 27
Current state: GREEN_STATE Remaining time: 26
Current state: GREEN_STATE Remaining time: 25
Current state: GREEN_STATE Remaining time: 24
Current state: GREEN_STATE Remaining time: 23
Current state: GREEN_STATE Remaining time: 22
Current state: GREEN_STATE Remaining time: 21
Current state: GREEN_STATE Remaining time: 20
Current state: GREEN_STATE Remaining time: 19
Current state: GREEN_STATE Remaining time: 18
Current state: GREEN_STATE Remaining time: 17
Current state: GREEN_STATE Remaining time: 16
Current state: GREEN_STATE Remaining time: 15
Current state: GREEN_STATE Remaining time: 14
Current state: GREEN_STATE Remaining time: 13
Current state: GREEN_STATE Remaining time: 12
Current state: GREEN_STATE Remaining time: 11
Current state: GREEN_STATE Remaining time: 10
Current state: GREEN_STATE Remaining time: 9
Current state: GREEN_STATE Remaining time: 8
Current state: GREEN_STATE Remaining time: 7
Current state: GREEN_STATE Remaining time: 6
Current state: GREEN_STATE Remaining time: 5
Current state: GREEN_STATE Remaining time: 4
Current state: GREEN_STATE Remaining time: 3
Current state: GREEN_STATE Remaining time: 2
Current state: GREEN_STATE Remaining time: 1
Current state: GREEN_STATE Remaining time: 0
Green State switching with yellow
Current state: YELLOW_STATE Remaining time: 3
Current state: YELLOW_STATE Remaining time: 2
Current state: YELLOW_STATE Remaining time: 1
Current state: YELLOW_STATE Remaining time: 0
```



```
Green State switching with yellow
Current state: YELLOW_STATE Remaining time: 3
Current state: YELLOW_STATE Remaining time: 2
Current state: YELLOW_STATE Remaining time: 1
Current state: YELLOW_STATE Remaining time: 0
Yellow state switching to red
Current state: RED_STATE Remaining time: 15
Current state: RED_STATE Remaining time: 14
Current state: RED_STATE Remaining time: 13
Current state: RED_STATE Remaining time: 12
Current state: RED_STATE Remaining time: 11
Current state: RED_STATE Remaining time: 10
Current state: RED_STATE Remaining time: 9
Current state: RED_STATE Remaining time: 8
Current state: RED_STATE Remaining time: 7
Current state: RED_STATE Remaining time: 6
Current state: RED_STATE Remaining time: 5
Current state: RED_STATE Remaining time: 4
Current state: RED_STATE Remaining time: 3
Current state: RED_STATE Remaining time: 2
Current state: RED_STATE Remaining time: 1
Current state: RED_STATE Remaining time: 0
Red state switching to green

Process finished with exit code 0
```

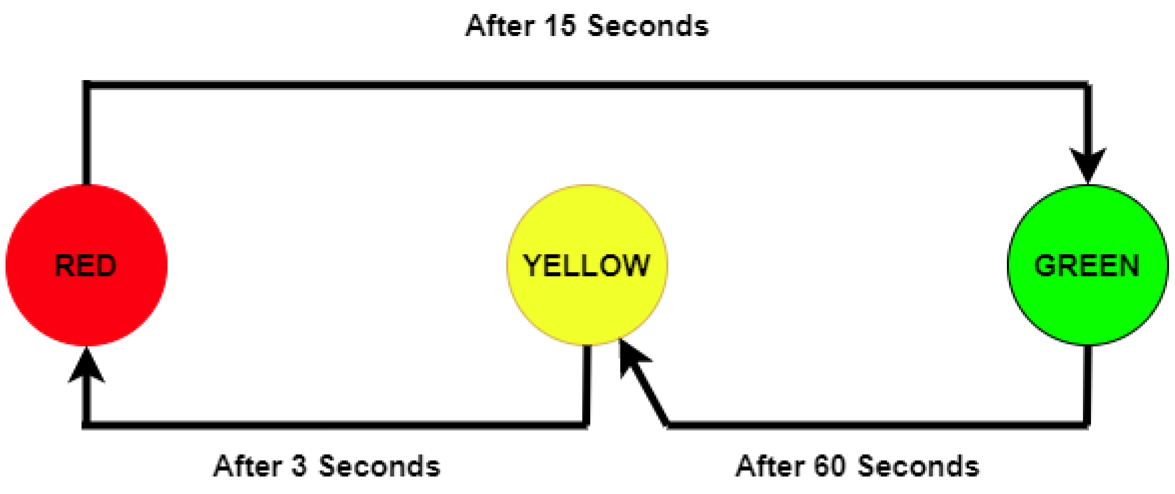
Solution (State Design Pattern)

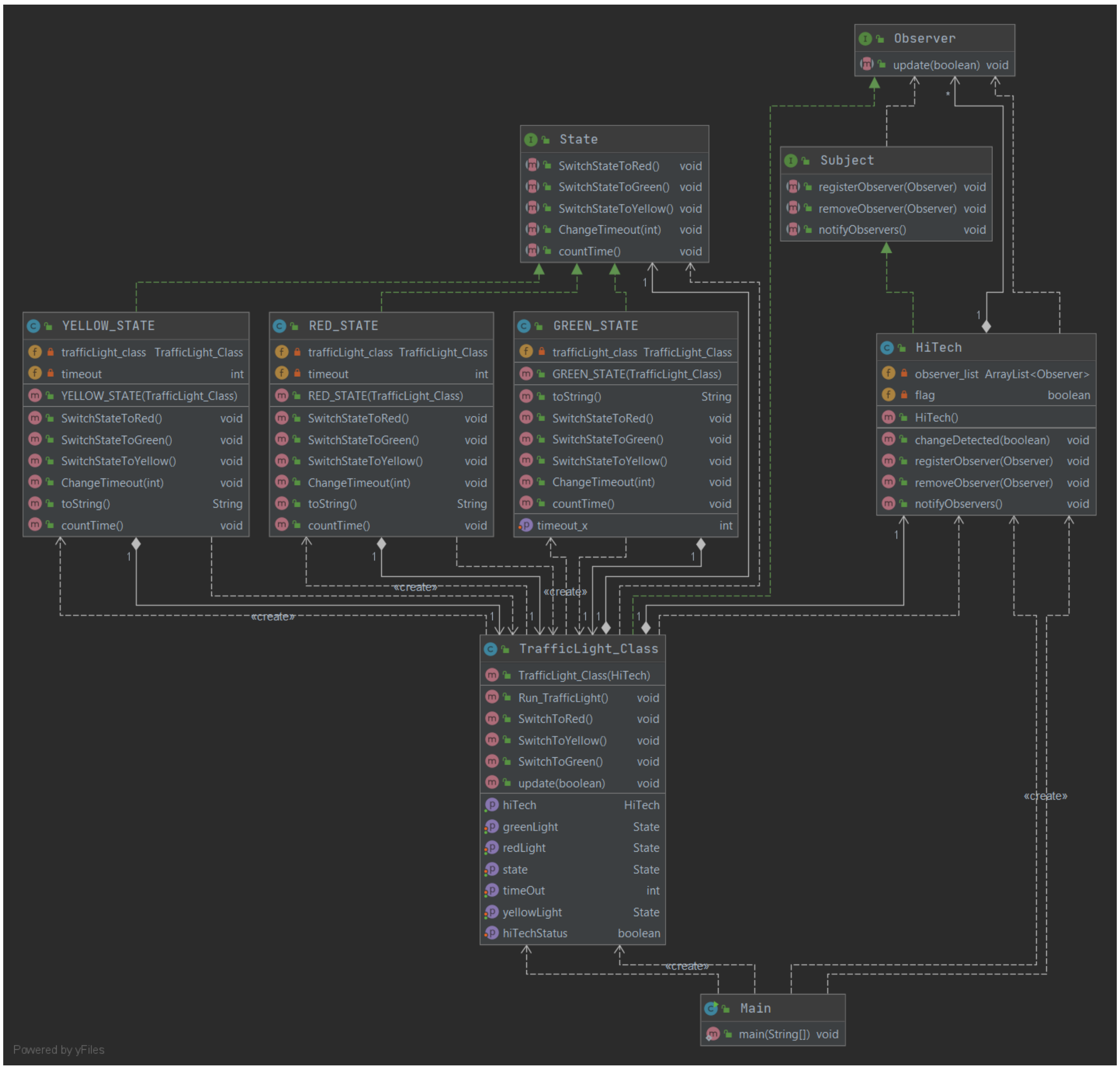
In the problem where the timeout_x value should change according to the traffic situation, I used the observer design pattern. Here I will describe the changes and innovations I have made.

First of all, I created the subject and observer classes required for the Observer design pattern. Then I created the HiTech class that measures the status of traffic. Since this class will be the class that does the updates, it uses the methods of the subject interface. Thus, the TrafficLight_Class class will be aware of the traffic situation as a member of this class. If the changeDetected function is called, I assign the status of the traffic to the flag parameter I have kept in the Hitech class. And I'm calling the notifyObservers method. Thus, in the notifyObservers method, all classes connected to this camera are notified. And it helps in updating values.

Coming to the TrafficLight_Class class, this class additionally uses the method of the Observer interface. Thus, it can follow the traffic situation coming from the camera it follows and change the timeout_x value of the green light. As an additional parameter, a HiTech class camera came to this class. this symbolizes the camera he is following. In the update method, if TrafficStatus is true, timeout_x is 90. If false it becomes 60. There are no other changes in the TrafficLight_Class class. Test simulation of the traffic light is the same. Only in the main section, necessary actions have been taken to show the change of timeout_x according to the traffic situation. Apart from that, no changes were applied to other classes.

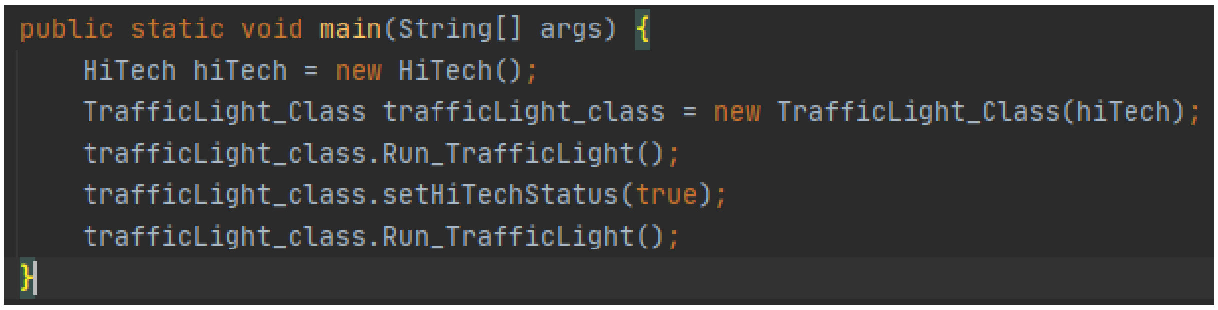
The UML diagram and state diagram of the method I have applied is as follows.



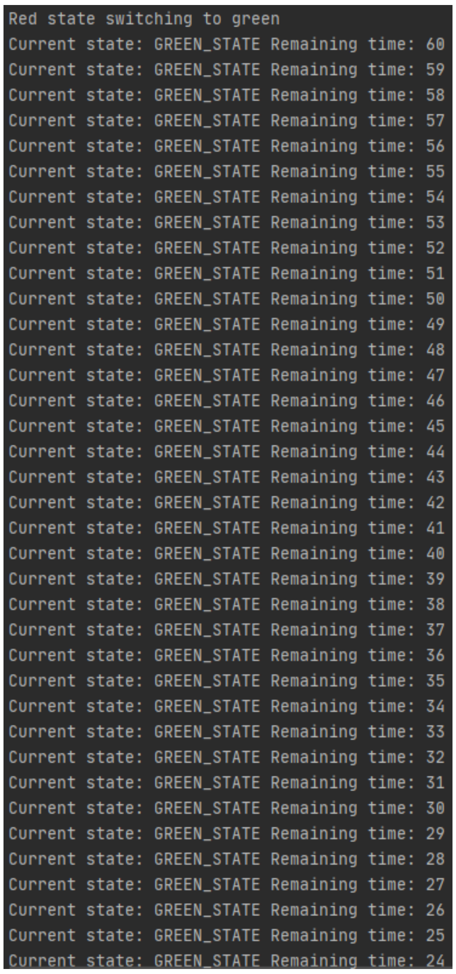


Output

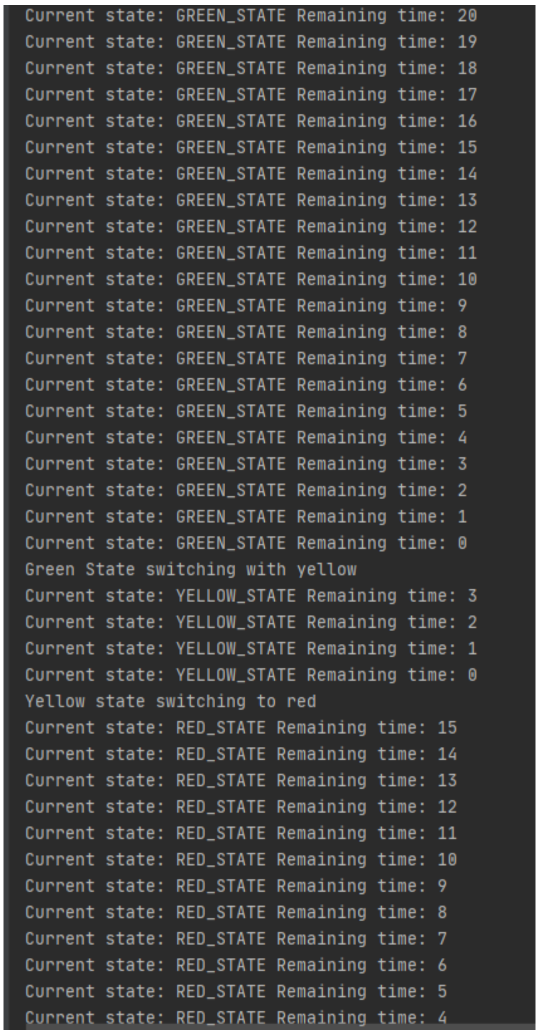
Main



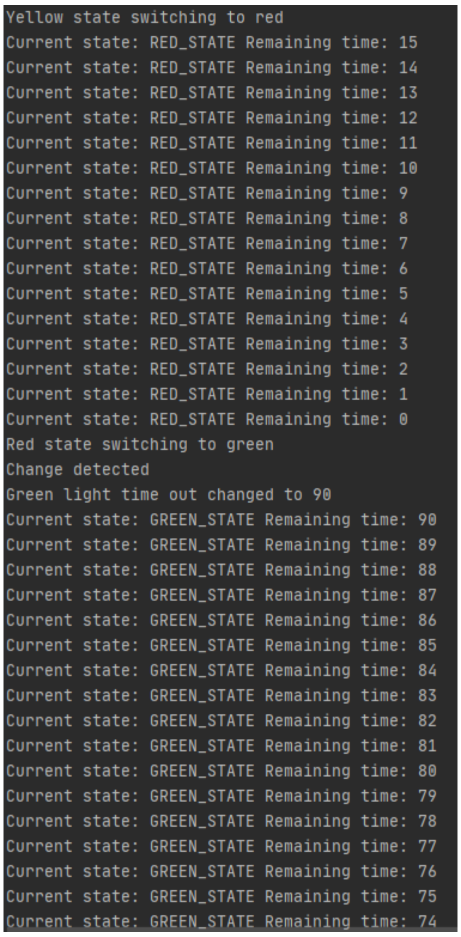
```
public static void main(String[] args) {
    HiTech hiTech = new HiTech();
    TrafficLight_Class trafficLight_class = new TrafficLight_Class(hiTech);
    trafficLight_class.Run_TrafficLight();
    trafficLight_class.setHiTechStatus(true);
    trafficLight_class.Run_TrafficLight();
}
```



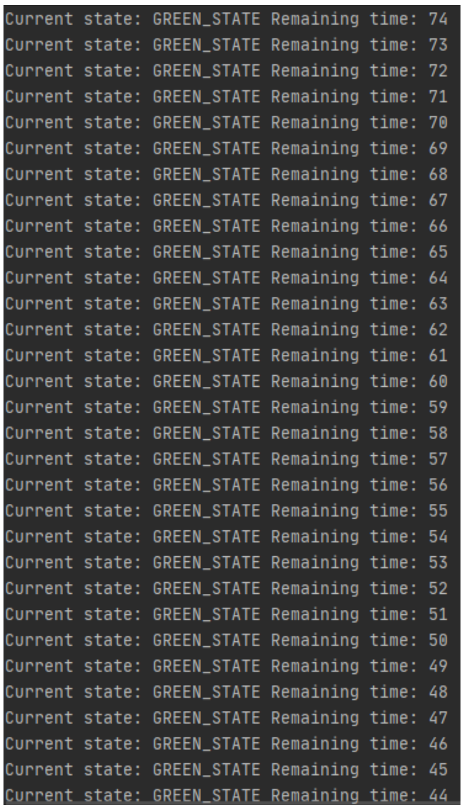
```
Red state switching to green
Current state: GREEN_STATE Remaining time: 60
Current state: GREEN_STATE Remaining time: 59
Current state: GREEN_STATE Remaining time: 58
Current state: GREEN_STATE Remaining time: 57
Current state: GREEN_STATE Remaining time: 56
Current state: GREEN_STATE Remaining time: 55
Current state: GREEN_STATE Remaining time: 54
Current state: GREEN_STATE Remaining time: 53
Current state: GREEN_STATE Remaining time: 52
Current state: GREEN_STATE Remaining time: 51
Current state: GREEN_STATE Remaining time: 50
Current state: GREEN_STATE Remaining time: 49
Current state: GREEN_STATE Remaining time: 48
Current state: GREEN_STATE Remaining time: 47
Current state: GREEN_STATE Remaining time: 46
Current state: GREEN_STATE Remaining time: 45
Current state: GREEN_STATE Remaining time: 44
Current state: GREEN_STATE Remaining time: 43
Current state: GREEN_STATE Remaining time: 42
Current state: GREEN_STATE Remaining time: 41
Current state: GREEN_STATE Remaining time: 40
Current state: GREEN_STATE Remaining time: 39
Current state: GREEN_STATE Remaining time: 38
Current state: GREEN_STATE Remaining time: 37
Current state: GREEN_STATE Remaining time: 36
Current state: GREEN_STATE Remaining time: 35
Current state: GREEN_STATE Remaining time: 34
Current state: GREEN_STATE Remaining time: 33
Current state: GREEN_STATE Remaining time: 32
Current state: GREEN_STATE Remaining time: 31
Current state: GREEN_STATE Remaining time: 30
Current state: GREEN_STATE Remaining time: 29
Current state: GREEN_STATE Remaining time: 28
Current state: GREEN_STATE Remaining time: 27
Current state: GREEN_STATE Remaining time: 26
Current state: GREEN_STATE Remaining time: 25
Current state: GREEN_STATE Remaining time: 24
```



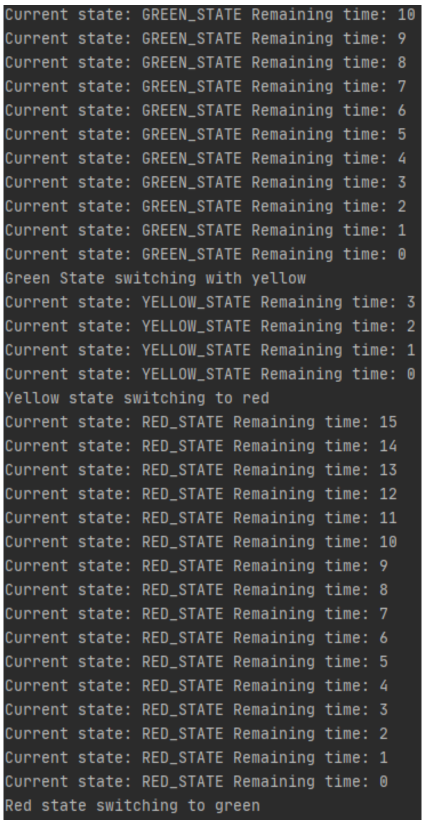
```
Current state: GREEN_STATE Remaining time: 20
Current state: GREEN_STATE Remaining time: 19
Current state: GREEN_STATE Remaining time: 18
Current state: GREEN_STATE Remaining time: 17
Current state: GREEN_STATE Remaining time: 16
Current state: GREEN_STATE Remaining time: 15
Current state: GREEN_STATE Remaining time: 14
Current state: GREEN_STATE Remaining time: 13
Current state: GREEN_STATE Remaining time: 12
Current state: GREEN_STATE Remaining time: 11
Current state: GREEN_STATE Remaining time: 10
Current state: GREEN_STATE Remaining time: 9
Current state: GREEN_STATE Remaining time: 8
Current state: GREEN_STATE Remaining time: 7
Current state: GREEN_STATE Remaining time: 6
Current state: GREEN_STATE Remaining time: 5
Current state: GREEN_STATE Remaining time: 4
Current state: GREEN_STATE Remaining time: 3
Current state: GREEN_STATE Remaining time: 2
Current state: GREEN_STATE Remaining time: 1
Current state: GREEN_STATE Remaining time: 0
Green State switching with yellow
Current state: YELLOW_STATE Remaining time: 3
Current state: YELLOW_STATE Remaining time: 2
Current state: YELLOW_STATE Remaining time: 1
Current state: YELLOW_STATE Remaining time: 0
Yellow state switching to red
Current state: RED_STATE Remaining time: 15
Current state: RED_STATE Remaining time: 14
Current state: RED_STATE Remaining time: 13
Current state: RED_STATE Remaining time: 12
Current state: RED_STATE Remaining time: 11
Current state: RED_STATE Remaining time: 10
Current state: RED_STATE Remaining time: 9
Current state: RED_STATE Remaining time: 8
Current state: RED_STATE Remaining time: 7
Current state: RED_STATE Remaining time: 6
Current state: RED_STATE Remaining time: 5
Current state: RED_STATE Remaining time: 4
```



```
Yellow state switching to red
Current state: RED_STATE Remaining time: 15
Current state: RED_STATE Remaining time: 14
Current state: RED_STATE Remaining time: 13
Current state: RED_STATE Remaining time: 12
Current state: RED_STATE Remaining time: 11
Current state: RED_STATE Remaining time: 10
Current state: RED_STATE Remaining time: 9
Current state: RED_STATE Remaining time: 8
Current state: RED_STATE Remaining time: 7
Current state: RED_STATE Remaining time: 6
Current state: RED_STATE Remaining time: 5
Current state: RED_STATE Remaining time: 4
Current state: RED_STATE Remaining time: 3
Current state: RED_STATE Remaining time: 2
Current state: RED_STATE Remaining time: 1
Current state: RED_STATE Remaining time: 0
Red state switching to green
Change detected
Green light time out changed to 90
Current state: GREEN_STATE Remaining time: 90
Current state: GREEN_STATE Remaining time: 89
Current state: GREEN_STATE Remaining time: 88
Current state: GREEN_STATE Remaining time: 87
Current state: GREEN_STATE Remaining time: 86
Current state: GREEN_STATE Remaining time: 85
Current state: GREEN_STATE Remaining time: 84
Current state: GREEN_STATE Remaining time: 83
Current state: GREEN_STATE Remaining time: 82
Current state: GREEN_STATE Remaining time: 81
Current state: GREEN_STATE Remaining time: 80
Current state: GREEN_STATE Remaining time: 79
Current state: GREEN_STATE Remaining time: 78
Current state: GREEN_STATE Remaining time: 77
Current state: GREEN_STATE Remaining time: 76
Current state: GREEN_STATE Remaining time: 75
Current state: GREEN_STATE Remaining time: 74
```



```
Current state: GREEN_STATE Remaining time: 74
Current state: GREEN_STATE Remaining time: 73
Current state: GREEN_STATE Remaining time: 72
Current state: GREEN_STATE Remaining time: 71
Current state: GREEN_STATE Remaining time: 70
Current state: GREEN_STATE Remaining time: 69
Current state: GREEN_STATE Remaining time: 68
Current state: GREEN_STATE Remaining time: 67
Current state: GREEN_STATE Remaining time: 66
Current state: GREEN_STATE Remaining time: 65
Current state: GREEN_STATE Remaining time: 64
Current state: GREEN_STATE Remaining time: 63
Current state: GREEN_STATE Remaining time: 62
Current state: GREEN_STATE Remaining time: 61
Current state: GREEN_STATE Remaining time: 60
Current state: GREEN_STATE Remaining time: 59
Current state: GREEN_STATE Remaining time: 58
Current state: GREEN_STATE Remaining time: 57
Current state: GREEN_STATE Remaining time: 56
Current state: GREEN_STATE Remaining time: 55
Current state: GREEN_STATE Remaining time: 54
Current state: GREEN_STATE Remaining time: 53
Current state: GREEN_STATE Remaining time: 52
Current state: GREEN_STATE Remaining time: 51
Current state: GREEN_STATE Remaining time: 50
Current state: GREEN_STATE Remaining time: 49
Current state: GREEN_STATE Remaining time: 48
Current state: GREEN_STATE Remaining time: 47
Current state: GREEN_STATE Remaining time: 46
Current state: GREEN_STATE Remaining time: 45
Current state: GREEN_STATE Remaining time: 44
```



```
Current state: GREEN_STATE Remaining time: 10
Current state: GREEN_STATE Remaining time: 9
Current state: GREEN_STATE Remaining time: 8
Current state: GREEN_STATE Remaining time: 7
Current state: GREEN_STATE Remaining time: 6
Current state: GREEN_STATE Remaining time: 5
Current state: GREEN_STATE Remaining time: 4
Current state: GREEN_STATE Remaining time: 3
Current state: GREEN_STATE Remaining time: 2
Current state: GREEN_STATE Remaining time: 1
Current state: GREEN_STATE Remaining time: 0
Green State switching with yellow
Current state: YELLOW_STATE Remaining time: 3
Current state: YELLOW_STATE Remaining time: 2
Current state: YELLOW_STATE Remaining time: 1
Current state: YELLOW_STATE Remaining time: 0
Yellow state switching to red
Current state: RED_STATE Remaining time: 15
Current state: RED_STATE Remaining time: 14
Current state: RED_STATE Remaining time: 13
Current state: RED_STATE Remaining time: 12
Current state: RED_STATE Remaining time: 11
Current state: RED_STATE Remaining time: 10
Current state: RED_STATE Remaining time: 9
Current state: RED_STATE Remaining time: 8
Current state: RED_STATE Remaining time: 7
Current state: RED_STATE Remaining time: 6
Current state: RED_STATE Remaining time: 5
Current state: RED_STATE Remaining time: 4
Current state: RED_STATE Remaining time: 3
Current state: RED_STATE Remaining time: 2
Current state: RED_STATE Remaining time: 1
Current state: RED_STATE Remaining time: 0
Red state switching to green
```

Question 4

In this question, only option A is made. Therefore, only that part will be explained in the solution of the problem.

Solution

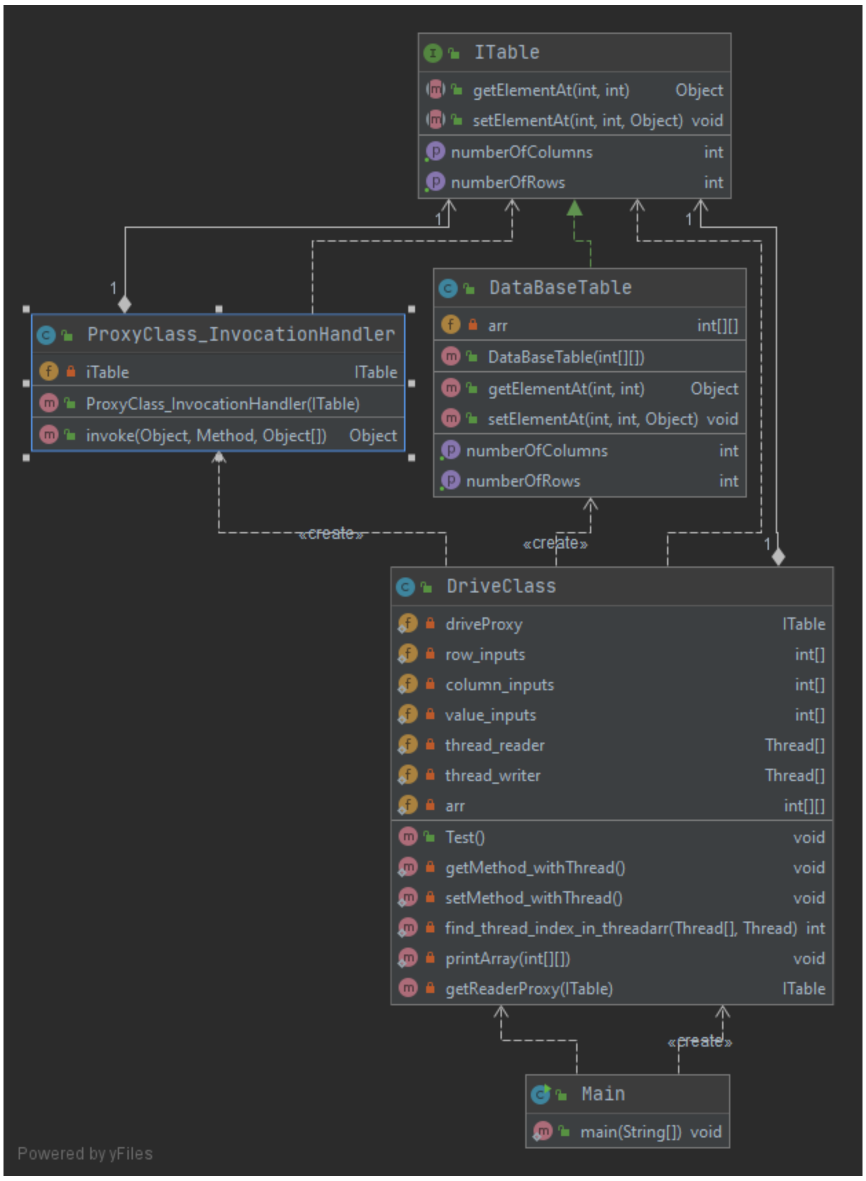
In this question, only option A is made. Therefore, only that part will be explained in the solution of the problem.

In this problem, your company has an old software library that provides the DataBaseTable class, but this class does not provide the ability to allow clients to lock individual table rows. So we have to specify a different method to avoid using this class. This can be solved with the Proxy Design pattern. In other words, we can call the DataBaseTable class through this class by writing a new class derived from the ITable interface. Since there is a reader and writer part in the problem, we should write it by considering these conditions. The reader part can use the getElementAt () method and the writer part can use the setElementAt () method. Otherwise, we have to give an error.

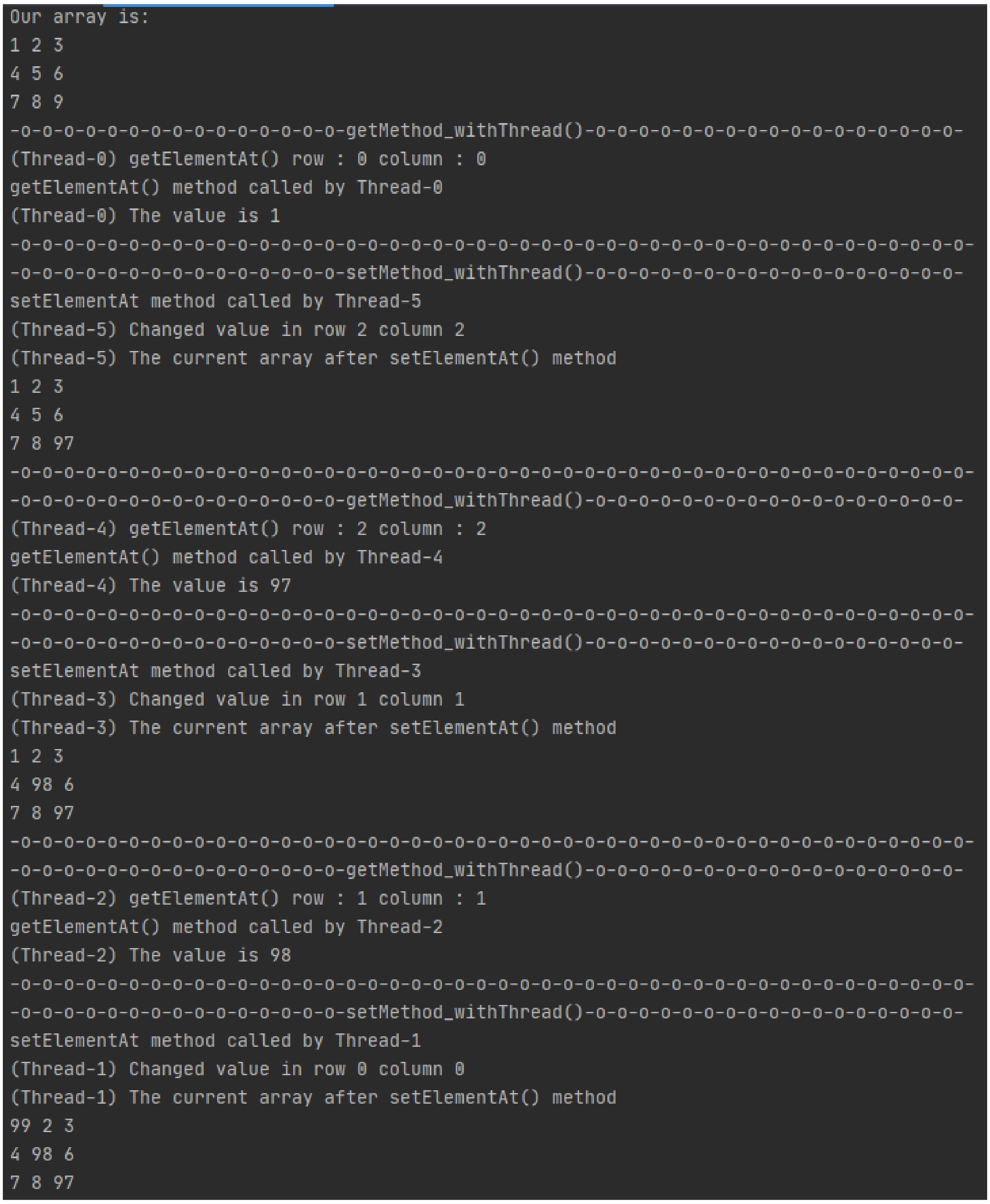
First of all, I implemented the ITable interface given to me in the problem. Then, to make a workable program, I created the DataBaseTable class, which is implemented from ITable, and filled in the methods. And finally I created the ProxyClass_InvocationHandler class. This class only takes a parameter of type ITable. With this parameter, we can call the methods that are filled in the DataBaseTable. This class implements the InvocationHandler interface. The aim here is to override the invoke method and prohibit methods that the user should not use. We will also call getElementAt () and setElementAt () functions in this invoke function. I made the invoke function synchronized because threads would call and we wanted it to run sequentially. After this class, proxy design pattern is completed. Next, do a test using our threads.

For the test, a total of 6 threads are used, with 3 writers and 3 reader threads. These are working synchronously. And they are waiting for each other. To run the test, a sample array is created and the getElementAt () and setElementAt () methods are called. The output of each running thread is written in detail.

The UML diagram of the method I have applied is as follows.



Output



```
Our array is:
1 2 3
4 5 6
7 8 9
-o-o-o-o-o-o-o-o-o-o-o-o-o-o-o-o-getMethod_withThread()-o-o-o-o-o-o-o-o-o-o-o-o-o-o-o-o-o-
(Thread-0) getElementAt() row : 0 column : 0
getElementAt() method called by Thread-0
(Thread-0) The value is 1
-o-o-o-o-o-o-o-o-o-o-o-o-o-o-o-o-o-o-o-o-o-o-o-o-o-o-o-o-o-o-o-o-o-o-o-o-o-o-o-o-o-o-o-o-o-
-o-o-o-o-o-o-o-o-o-o-o-o-o-o-o-o-setMethod_withThread()-o-o-o-o-o-o-o-o-o-o-o-o-o-o-o-o-o-
setElementAt method called by Thread-5
(Thread-5) Changed value in row 2 column 2
(Thread-5) The current array after setElementAt() method
1 2 3
4 5 6
7 8 97
-o-o-o-o-o-o-o-o-o-o-o-o-o-o-o-o-o-o-o-o-o-o-o-o-o-o-o-o-o-o-o-o-o-o-o-o-o-o-o-o-o-o-o-o-o-
-o-o-o-o-o-o-o-o-o-o-o-o-o-o-o-o-getMethod_withThread()-o-o-o-o-o-o-o-o-o-o-o-o-o-o-o-o-o-
(Thread-4) getElementAt() row : 2 column : 2
getElementAt() method called by Thread-4
(Thread-4) The value is 97
-o-o-o-o-o-o-o-o-o-o-o-o-o-o-o-o-o-o-o-o-o-o-o-o-o-o-o-o-o-o-o-o-o-o-o-o-o-o-o-o-o-o-o-o-o-
-o-o-o-o-o-o-o-o-o-o-o-o-o-o-o-o-setMethod_withThread()-o-o-o-o-o-o-o-o-o-o-o-o-o-o-o-o-o-
setElementAt method called by Thread-3
(Thread-3) Changed value in row 1 column 1
(Thread-3) The current array after setElementAt() method
1 2 3
4 98 6
7 8 97
-o-o-o-o-o-o-o-o-o-o-o-o-o-o-o-o-o-o-o-o-o-o-o-o-o-o-o-o-o-o-o-o-o-o-o-o-o-o-o-o-o-o-o-o-o-
-o-o-o-o-o-o-o-o-o-o-o-o-o-o-o-o-getMethod_withThread()-o-o-o-o-o-o-o-o-o-o-o-o-o-o-o-o-o-
(Thread-2) getElementAt() row : 1 column : 1
getElementAt() method called by Thread-2
(Thread-2) The value is 98
-o-o-o-o-o-o-o-o-o-o-o-o-o-o-o-o-o-o-o-o-o-o-o-o-o-o-o-o-o-o-o-o-o-o-o-o-o-o-o-o-o-o-o-o-o-
-o-o-o-o-o-o-o-o-o-o-o-o-o-o-o-o-setMethod_withThread()-o-o-o-o-o-o-o-o-o-o-o-o-o-o-o-o-o-
setElementAt method called by Thread-1
(Thread-1) Changed value in row 0 column 0
(Thread-1) The current array after setElementAt() method
99 2 3
4 98 6
7 8 97
```