CSE443

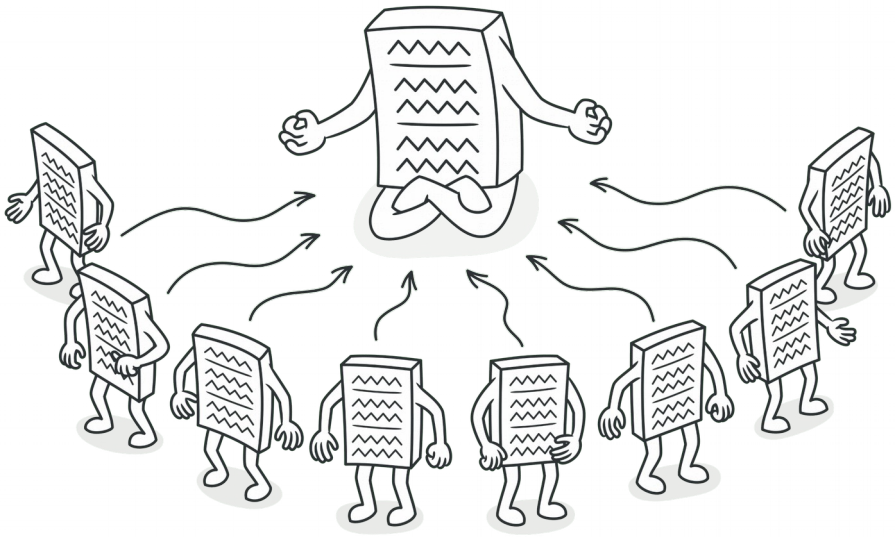
Homework 2 Report

Melih Yabaş

161044072

Part 1

1-)



The singleton pattern is a software design pattern that restricts the instantiation of a class to one "single" instance. In this design pattern, it is possible to clone the object using the clone () method inherited from the Object class but its cloning is a situation that does not fit the design pattern and does not meet the single object understanding.

In addition, in order to use the clone () method, the Singleton class must be implements from the Cloneable interface and the clone () method must be overridden and the Object class must be called with the clone method super.clone ().
The Clone () method does not create a new object, but creates a copy of the existing object.
A new object is not created after Clone method.

2-)

We can use “CloneNotSupportedException” to show the user it is not allowed. This exception does exist into SingletonObject class.

3-)

If the SingletonObject class was fully implemented from the Cloneable interface structure, a clone of the object created with clone () method in 1 above can be obtained.
However, at the end of the clone () method, the object that is a clone will not be a new object.
Even if the SingletonObject class is implanted from the Cloneable interface, the cloning of the Singleton object is completely prevented due to the clone () method throwing "CloneNotSupportedException" in the above 2. It is not very important to be "Cloneable" here.

Part 2

The satellite transmits data as 2D arrays of integers. We need to create an iterator class for these data that will print a 2D array spirally clockwise.
We need need to create an Iterator class for this purpose. MyIterator, implements Iterator interface. It gets a matrix and traverses it spirally. Then it keeps the new shape into an arraylist data structure. OrderSpiral() method does this calculation recursively. It gets start positions, x, y and next direction as inputs.

```
public Object next()
{

    if(hasNext())
        return (d.get(++index));
    else
        throw new ArrayIndexOutOfBoundsException();
}

/***
 * Returns true if the next element exist.
 * @return Boolean value
 */
public boolean hasNext() { return  (index<d.getIndex()-1); }
```

Now, next method shows the next element of the matrix spirally.
Sattellite class gives this iterator to the user. So user can traverse it.

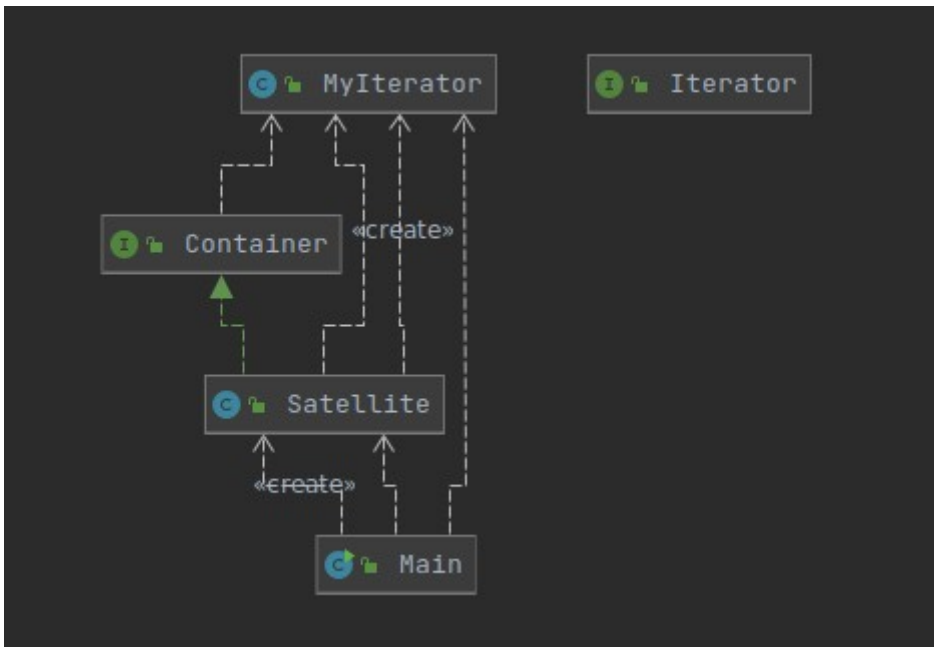
```
melih@melih-SATELLITE-L50-C:~/Desktop/part2$ make
javac Main.java Iterator.java MyIterator.java Container.java Satellit
e.java MyArrayList.java
melih@melih-SATELLITE-L50-C:~/Desktop/part2$ make run
java Main

Two dimensional array:
1 2 3 4 5 6 7
8 9 10 11 12 13 14
15 16 17 18 19 20 21
22 23 24 25 26 27 28
29 30 31 32 33 34 35
36 37 38 39 40 41 42
43 44 45 46 47 48 49


The 2D array is printing spirally clockwise...
1 2 3 4 5 6 7 14 21 28 35 42 49 48 47 46 45 44 43 36 29 22 15 8 9 10
11 12 13 20 27 34 41 40 39 38 37 30 23 16 17 18 19 26 33 32 3mememe
melih@melih-SATELLITE-L50-C:~/Desktop/part2$
```

As you can see, there is a 2 dimensional array. And the program prints it spially using myIterator class.

Class Diagram

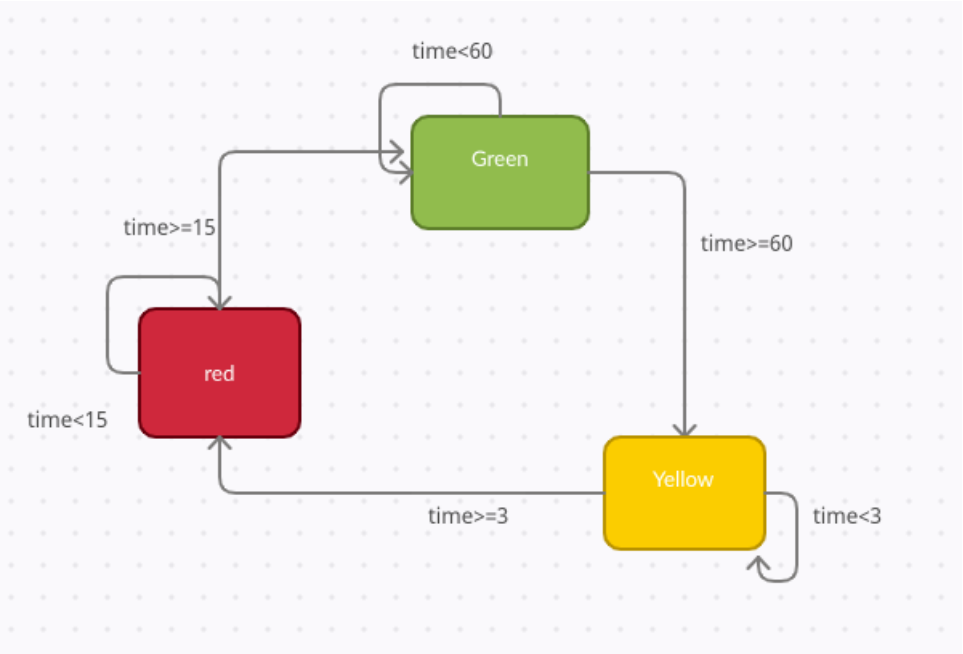


Part 3

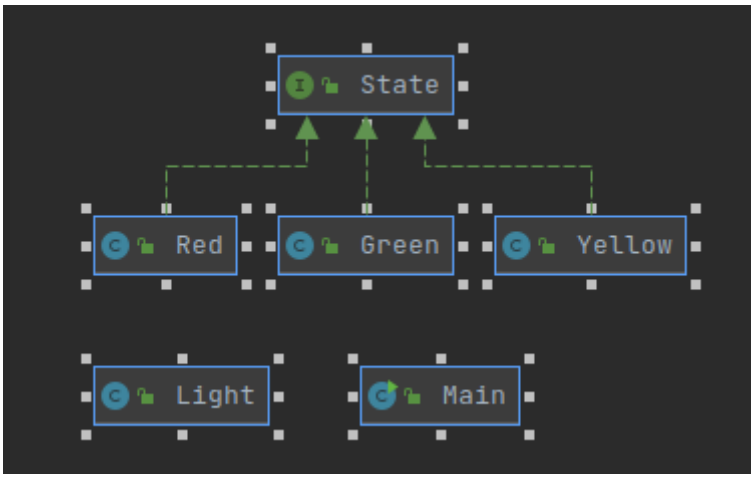
```
public interface State {
    public void red();
    public void green();
    public void yellow();
}
```

We have 3 states. These are: red, green and yellow. First we created a State interface. Each state has its own class and implements this State interface. Each of them keeps its timeout_x and a light object.
RED: switches to GREEN after 15 seconds.
YELLOW: switches to RED after 3 seconds.
GREEN: switches to YELLOW after 60 seconds

State Diagram

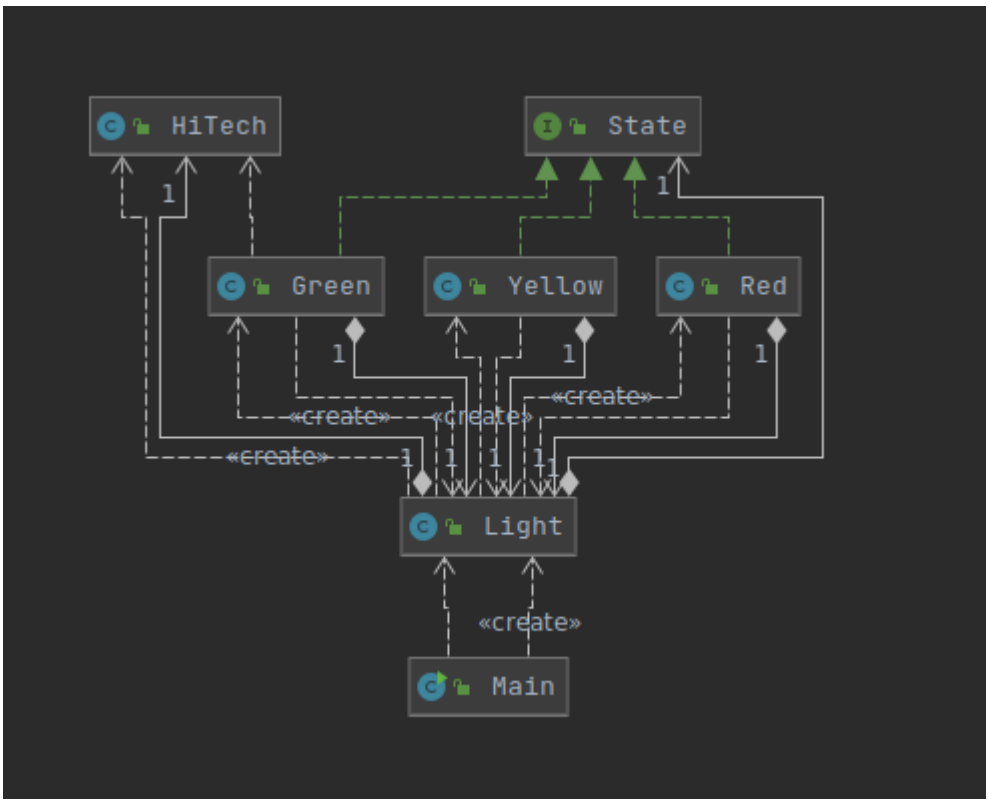


Class Diagram-1



There is a MOBESE camera on top of the traffic light that measures the amount of cars under it. When it detects a lot of traffic timeout_X is increased from 60 to 90 seconds. HiTech class has a method to do that. In this section, we need to add Observer Design Pattern into our system.

Class Diagram-2



Now, our states (red, green and yellow) implements both State and HiTech. These are observers and HiTech is a subject in this design. Each observer has an update method but only Green fills it. When a change detected, HiTech notifies all observers.
Light is out context class. We are using it to operate all system.

A piece of the output of the program

```
86s -Green State
87s -Green State
88s -Green State
89s -Green State
90s -Green State

Switching yellow...

1s -Yellow State
2s -Yellow State
3s -Yellow State

Switcing red...

1s -Red State
2s -Red State
3s -Red State
4s -Red State
5s -Red State
6s -Red State
```

Part 4

a)

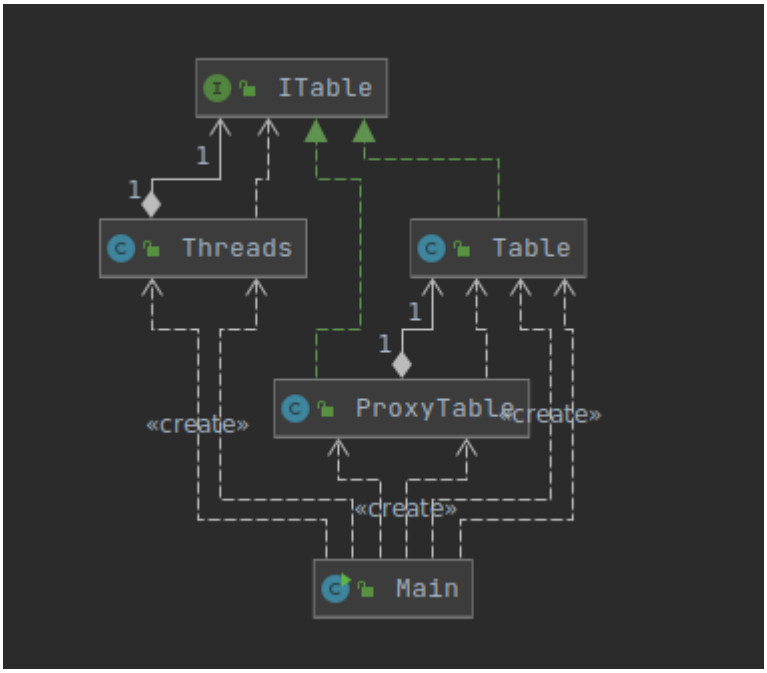
To use DataBase methods correctly, we should create a Proxy Design Pattern and syncronize the methods.

```
public Object getElementAt(int row, int column) {
    synchronized (lockArr[row]) {
        return realTable.getElementAt(row, column);
    }
}

public void setElementAt(Object element, int row, int column) {
    synchronized (lockArr[row]) {
        realTable.setElementAt(element, row, column);
    }
}
```

Now, we can say no reader thread calls getElementAt while a writer thread
is executing setElementAt.
To test this feature, I created a 2d array and, filled it using threads. Threads randomly sets and gets elements from the table.

Class Diagram



b)

In this part, we should design a new synchronization solution that prioritizes writers o DataBaseTable more than readers in terms of table row access.
To do this, a “setting” integer variable keeped in the class. While there are any set operation to do, get operatons going to wait.

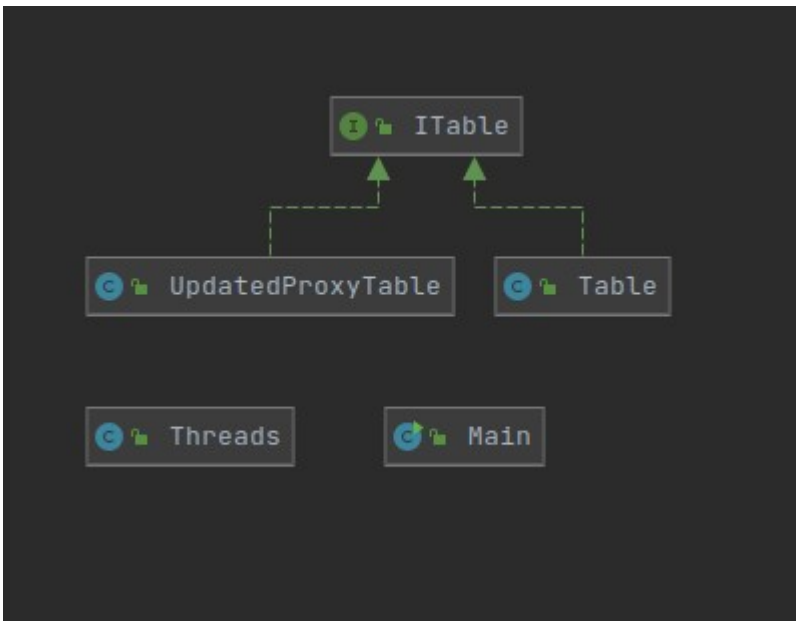
```
public Object getElementAt(int row, int column) {
    synchronized (locks[row]) {
        while(setting>0)
        {
            try{
                wait();

            }
            catch(Exception e)
            {

            }
        }

        return realTable.getElementAt(row, column);
    }
}
```

Class Diagram



Output

```
Thread3 Get Element At 2, 5 Return Value: 86
Thread3 Set Element: 29, Position: 9, 8

Thread1 Get Element At 3, 2 Return Value: 20
Thread1 Set Element: 99, Position: 1, 5

Thread1 Get Element At 1, 5 Return Value: 99

Thread3 Get Element At 9, 8 Return Value: 29
Thread1 Set Element: 79, Position: 5, 8
Thread3 Set Element: 21, Position: 1, 9

Thread1 Get Element At 5, 8 Return Value: 79

Thread3 Get Element At 1, 9 Return Value: 21
Thread3 complete
Thread1 Set Element: 90, Position: 6, 9

Thread1 Get Element At 6, 9 Return Value: 90
```