GEBZE TECHNICAL UNIVERSITY
COMPUTER ENGINEERING

OBJECT ORIENTED ANALYSIS and DESIGN
CSE 443 - 2020 FALL

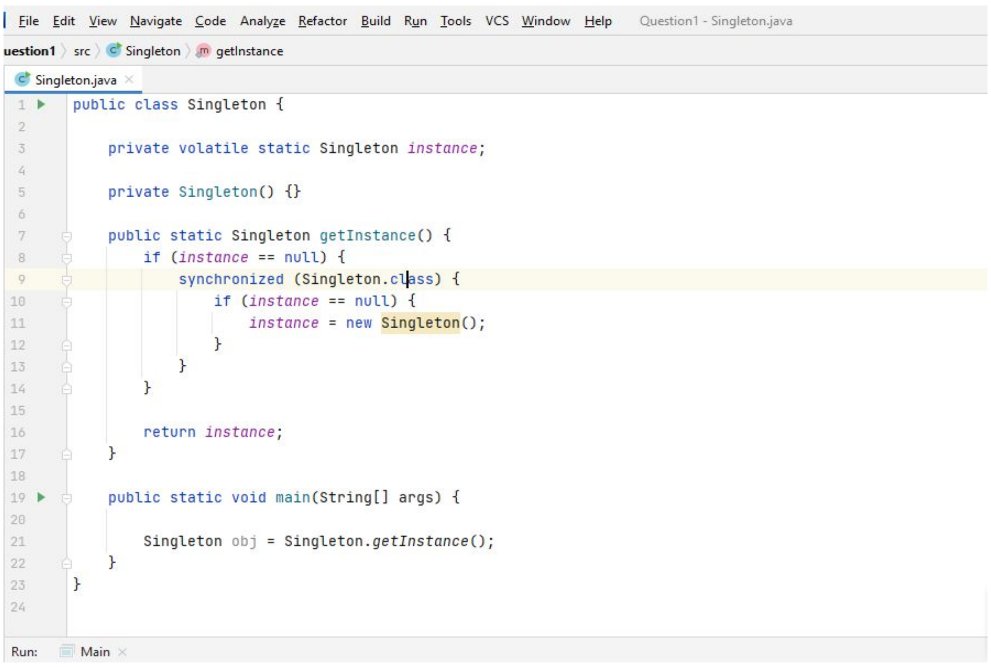
HW2 REPORT

SILA BENGİSU YÜKSEKLİ
1801042877

PART1 - SINGLETON DESIGN PATTERN

I. Explanation of Design Pattern

In some designs, only one object belonging to a class needs to be created. Otherwise, creating a second object will cause an error in such a situation. In such a case, the Singleton Pattern can be used. This design pattern allows only one instance of a class to be created and to open a global access point to it. An implementation of the singleton pattern that I wrote is as following :



File Edit View Navigate Code Analyze Refactor Build Run Tools VCS Window Help Question1 - Singleton.java
uestion1 › src › Singleton › getInstance
Singleton.java
Run: Main

```java
public class Singleton {

    private volatile static Singleton instance;

    private Singleton() {}

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

        return instance;
    }

    public static void main(String[] args) {

        Singleton obj = Singleton.getInstance();
    }
}
```

figure1.1 *Program that carries out Singleton pattern*

II. Questions

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.

- The object class's clone method allows making an identical copy of an object. It creates a new object belonging to the mentioned class and initializes all its fields the same as the contents of the copied object. Additionally, the cloneable interface is an empty interface and is used to specify whether to use the clone method. It is a marker interface. In order to use the clone method, the Cloneable interface must be implemented. Then there are two situations according to what was asked in the question :

- ➔ If the class does not implement the Cloneable interface and the clone method is called on the Singleton object, the CloneNotSupportedException is thrown.

- ➔ If the class implements the Cloneable interface and the clone method is called over the Singleton object, this object is copied. This causes two Singleton objects to be created.

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

- As mentioned in the above, if the class implements the Cloneable interface and calls the clone method, two Singleton objects are created. There are several ways to prevent this situation:

- ➔ This can be avoided by not implementing the Cloneable interface directly. Thus, when the clone method is called, CloneNotSupportedException is thrown.

- ➔ Let's say the Cloneable interface is somehow implemented. Then the clone method can be overridden to throw an exception when it is called. A second option is to make the method return the created unique object reference. Thus, when the clone method is called, it avoids the situation of creating more than one instance, by throwing an exception or returning the same object reference.

3. 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 ?

- The Singleton class will automatically implement the Cloneable interface if it is a subclass that derives from a class that implements the Cloneable interface. In other words, if the clone method is called without taking any precaution, two objects can be created in such a case.

- The way to prevent this is to override the clone method. It can be rewritten to throw the exception, as discussed above, or to return the reference of the unique object. Thus, the production of two objects is prevented.

PART2 - ITERATOR DESIGN PATTERN

I. Explanation of Design Pattern

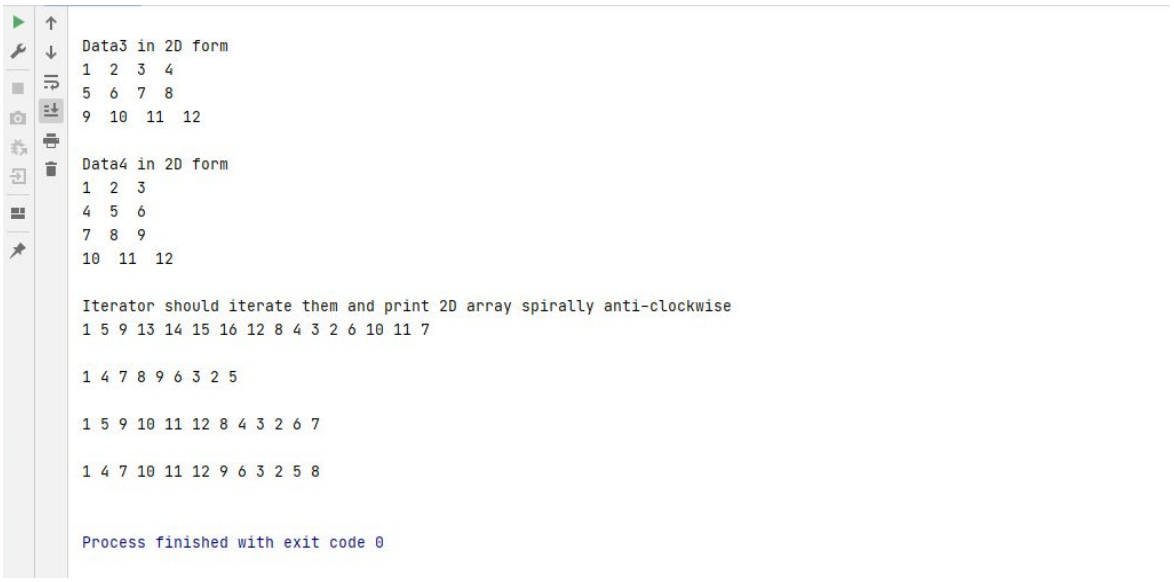
The purpose of this task is to be able to iterate clockwise and anticlockwise on the data received from the satellite and to achieve this in the most efficient way. Data from satellites may be data that needs to be processed differently, or data can be obtained in this way in the future. Therefore, it may be a situation where we have to handle them all differently. The most efficient and cost-effective way to do this is to use the Iterator Design Pattern. In this way, the data can be accessed without revealing its underlying representation. In the future, if it is necessary to read data line by line other than clockwise, all you have to do is write a new iterator class and apply it. We don't need to know anything about the data class. Thus, a program with low maintenance cost is obtained.

II. Explanations of Classes and Interfaces

There are two interfaces called DataOfGokturk3 and DataIterator. DataType1 and DataType2 classes implement the DataOfGokturk3 interface. These represent data of the satellite that is expressed as a two dimensional integer array. Thanks to the createIterator method that the classes have, data can be processed in a way which we want.
Also, there are two classes that implements DataIterator interface which are SpirallyClockwiseIterator and SpirallyAntiClockwiseIterator. First one iterates through the data clockwise and second one does the same thing anticlockwise. Only the algorithm of the SpirallyAntiClockwiseIterator class was implemented as the homework says.
Besides, I wrote a class which is called DataPrinter and it acts like an intermediary class. It holds DataOfGokturk3 objects in a list and provides a method to print data of the satellite.

III. Execution Process

To test methods, I created 4x4 , 3x3, 3x4 and 4x3 matrices. These are representations of satellite data. Then I created different DataOfGokturk3 objects by using these matrices. I gave them to an object of DataPrinter class, then it prints them on the screen by using an iterator according to the data type. An example result of execution is shown in below :



```
Data3 in 2D form
1  2  3  4
5  6  7  8
9  10  11  12

Data4 in 2D form
1  2  3
4  5  6
7  8  9
10  11  12

Iterator should iterate them and print 2D array spirally anti-clockwise
1 5 9 13 14 15 16 12 8 4 3 2 6 10 11 7

1 4 7 8 9 6 3 2 5

1 5 9 10 11 12 8 4 3 2 6 7

1 4 7 10 11 12 9 6 3 2 5 8


Process finished with exit code 0
```

figure2.1 *Example result of the execution*

IV. Class Dependencies and UML diagram

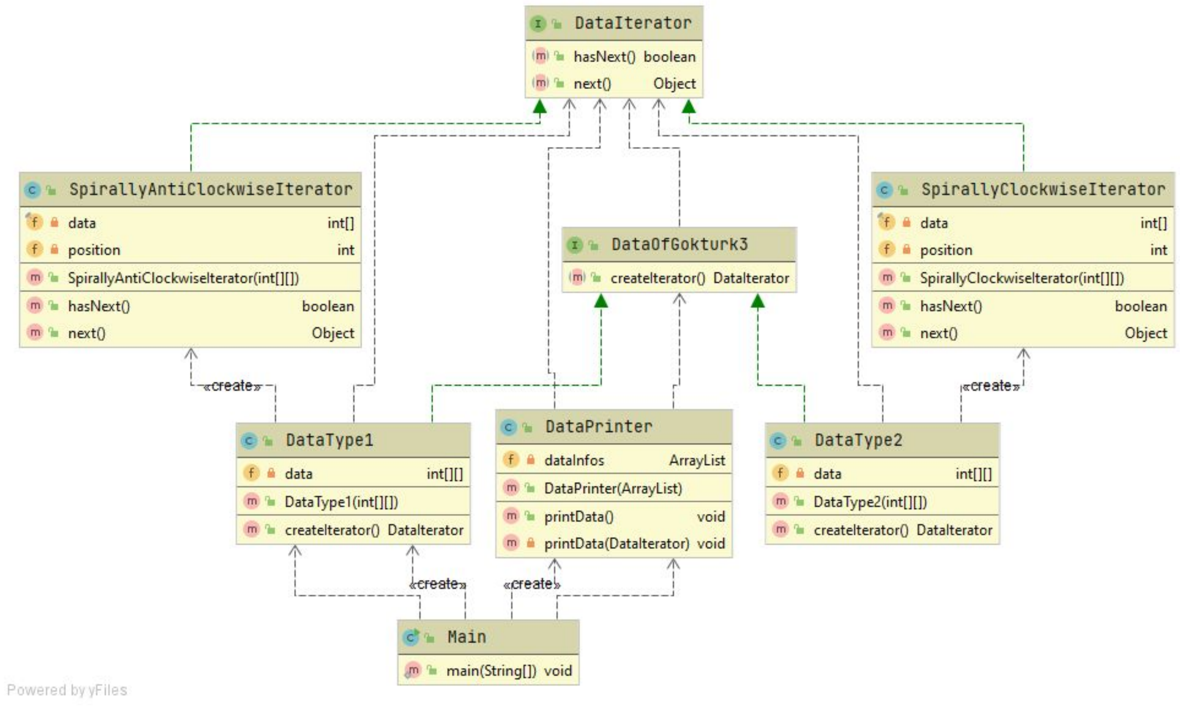


figure2.2 *UML diagram of the program that shows dependencies*

PART3 - STATE / OBSERVER DESIGN PATTERN

A. PART3 - STATE DESIGN PATTERN

I. Explanation of Design Pattern

The purpose of this task is to make a traffic light work. When the traffic light is working, there are three situations: red light, yellow light and green light. These three cases can be written using the state design pattern. The State Design Pattern allows an object to change its behavior when its internal state changes. In this program, the traffic light will change its behavior and activate a different color light when its state changes. Thus, this design offers a good loosely coupled system. If a new color is desired to be added in the future, this task can be easily achieved by writing a state and action for that color. In the traffic light class, a method to switch to that color is written and this task is actually transferred to the state class at the back.

II. State Diagram

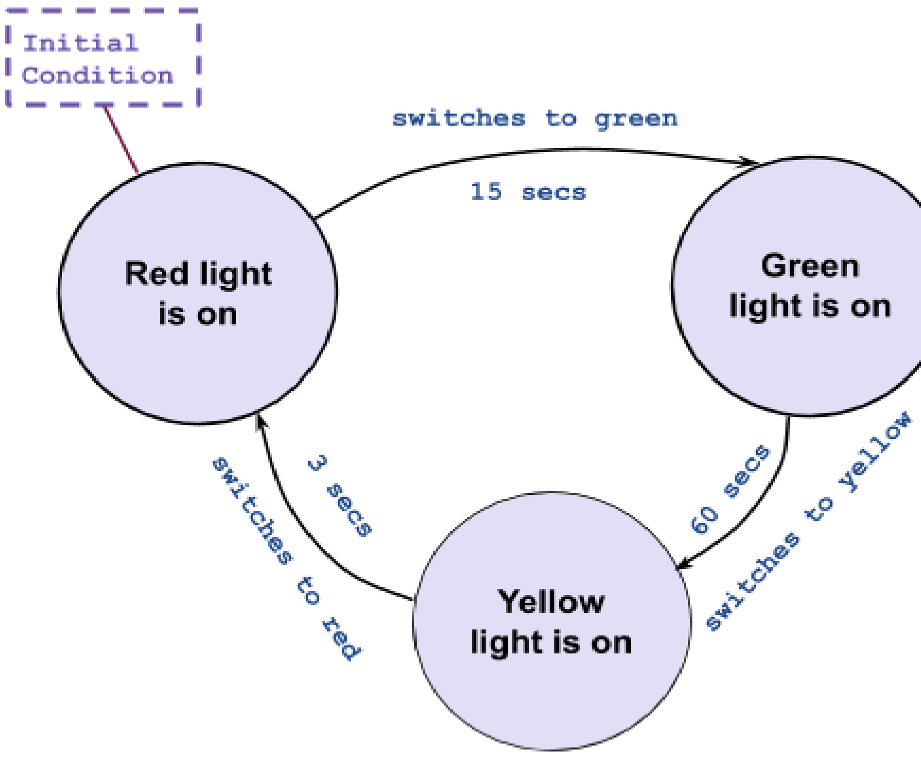


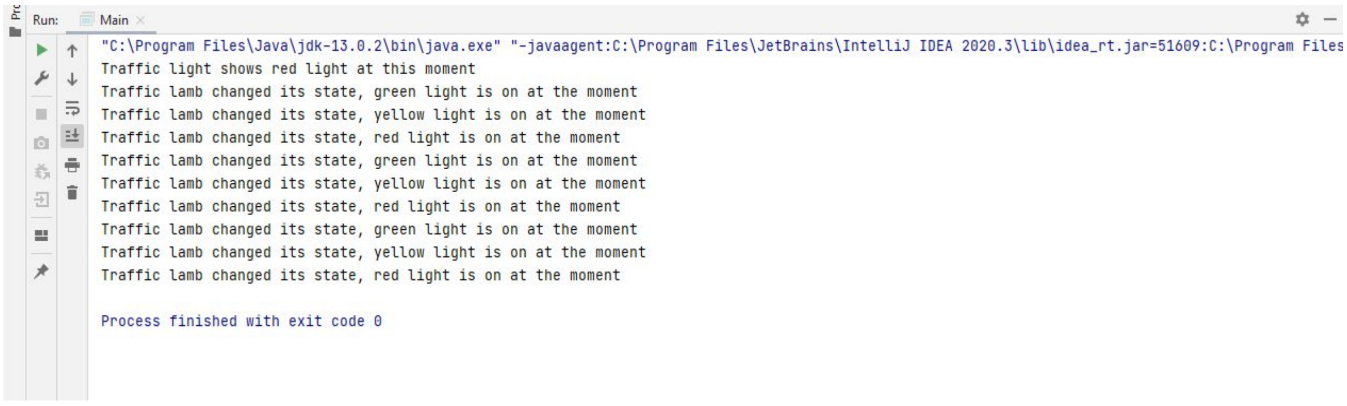
figure3.1.1 *State diagram of the problem*

III. Explanations of Classes and Interfaces

There is a State interface to represent all of the states of the traffic light. These states can be green light, yellow light or red light. So there are also three state classes which represent the cases where the specific light is on. Besides, lamb is represented in the TrafficLight class. This class holds every state that the traffic light will experience and a current state that shows the current light. This class has three methods to switch lights, but actually it delegates its job to state classes.

IV. Execution Process

To test my methods, I created an instance of the TrafficLight class and created a while loop. In this loop TimeUnit.*SECONDS*.sleep method is called to show the elapsed time when the lights change. This loop runs only three times just for the sake of the example.



```
"C:\Program Files\Java\jdk-13.0.2\bin\java.exe" "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA 2020.3\lib\idea_rt.jar=51609:C:\Program Files
Traffic light shows red light at this moment
Traffic lamb changed its state, green light is on at the moment
Traffic lamb changed its state, yellow light is on at the moment
Traffic lamb changed its state, red light is on at the moment
Traffic lamb changed its state, green light is on at the moment
Traffic lamb changed its state, yellow light is on at the moment
Traffic lamb changed its state, red light is on at the moment
Traffic lamb changed its state, green light is on at the moment
Traffic lamb changed its state, yellow light is on at the moment
Traffic lamb changed its state, red light is on at the moment

Process finished with exit code 0
```

figure3.1.2 *Example result of the execution*

V. Class Dependencies and UML diagram

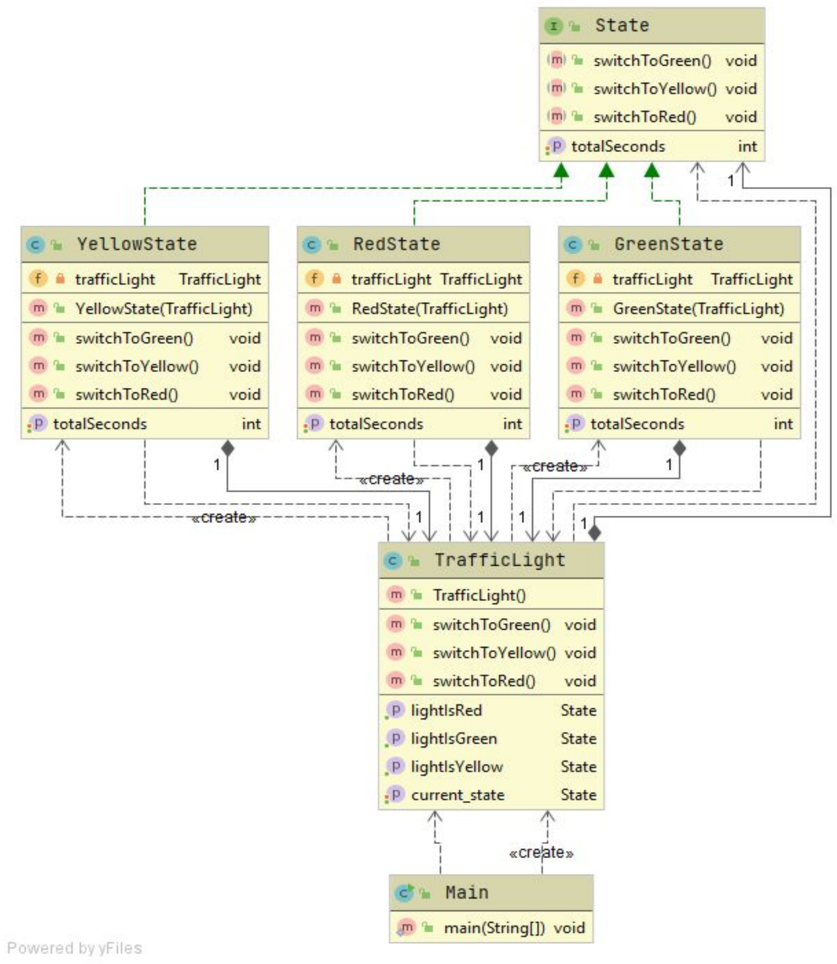


figure3.1.3 *UML diagram of the program that shows dependencies*

B. PART3 - STATE and OBSERVER PATTERN

I. Explanation of Design Pattern

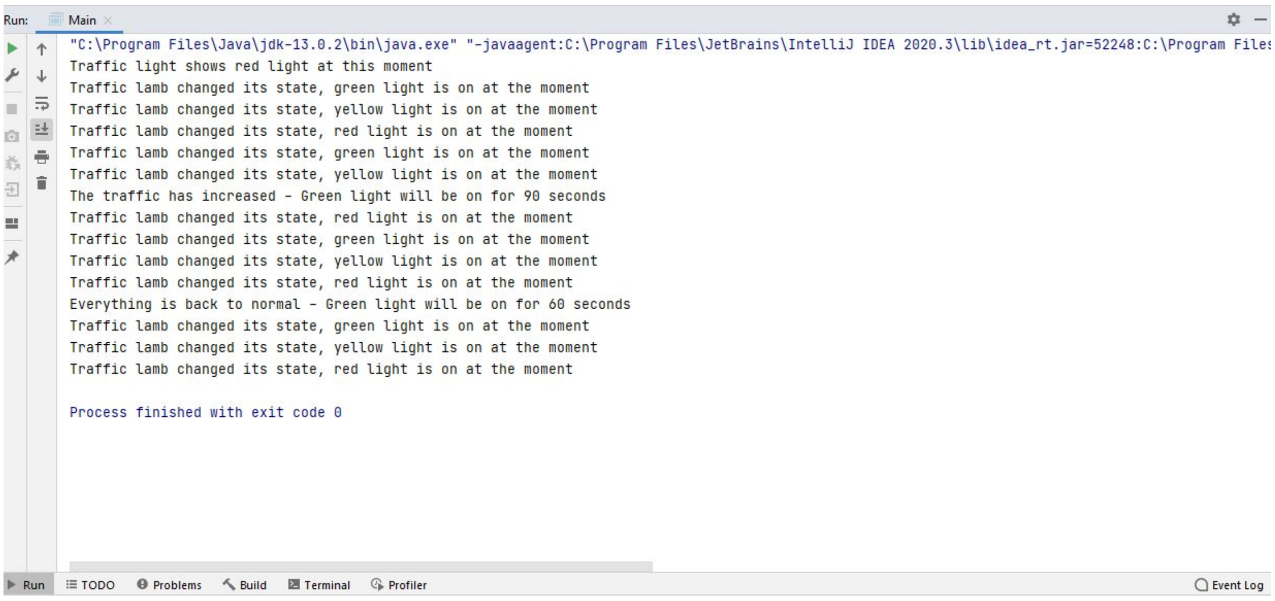
The purpose of this program is to determine the traffic density and adjust the green light duration accordingly. For this, we have a mobese camera software library. The camera mounted above the traffic light, informs whenever any density is observed, and allows the green light to change. This program was implemented by adding the observer design pattern to the state design pattern written in the part1.

II. Explanations of Classes and Interfaces

Subject, Observer interfaces and HiTech class have also been added to the classes written in the part1. HiTech class behaves as a subject by implementing the Subject interface. It has a method which is changeDetected. It takes a flag to determine whether the traffic has increased or not. According to that, it informs all the observers who subscribed to this software. This method is called automatically by the hardware, in our case I called it in my main method to test the program.
TrafficLight class implements the Observer interface and it registers to the HiTech in its constructor so that when an update occurs, traffic lamb will be informed by the subject.

III. Execution Process

To test my methods, I created an instance of the TrafficLight class and created a while loop. In this loop TimeUnit.*SECONDS*.sleep method is called to show the elapsed time when the lights change. This loop runs only four times just for the sake of the example and in the second and third loop, so-called hardware calls the changeDetected method automatically. In our case I do that in the main method and test the transition time between the green and yellow light.



```
"C:\Program Files\Java\jdk-13.0.2\bin\java.exe" "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA 2020.3\lib\idea_rt.jar=52248:C:\Program Files
Traffic light shows red light at this moment
Traffic lamb changed its state, green light is on at the moment
Traffic lamb changed its state, yellow light is on at the moment
Traffic lamb changed its state, red light is on at the moment
Traffic lamb changed its state, green light is on at the moment
Traffic lamb changed its state, yellow light is on at the moment
The traffic has increased - Green light will be on for 90 seconds
Traffic lamb changed its state, red light is on at the moment
Traffic lamb changed its state, green light is on at the moment
Traffic lamb changed its state, yellow light is on at the moment
Traffic lamb changed its state, red light is on at the moment
Everything is back to normal - Green light will be on for 60 seconds
Traffic lamb changed its state, green light is on at the moment
Traffic lamb changed its state, yellow light is on at the moment
Traffic lamb changed its state, red light is on at the moment

Process finished with exit code 0
```

figure3.2.1 *Example result of the execution*

IV. Class Dependencies and UML diagram

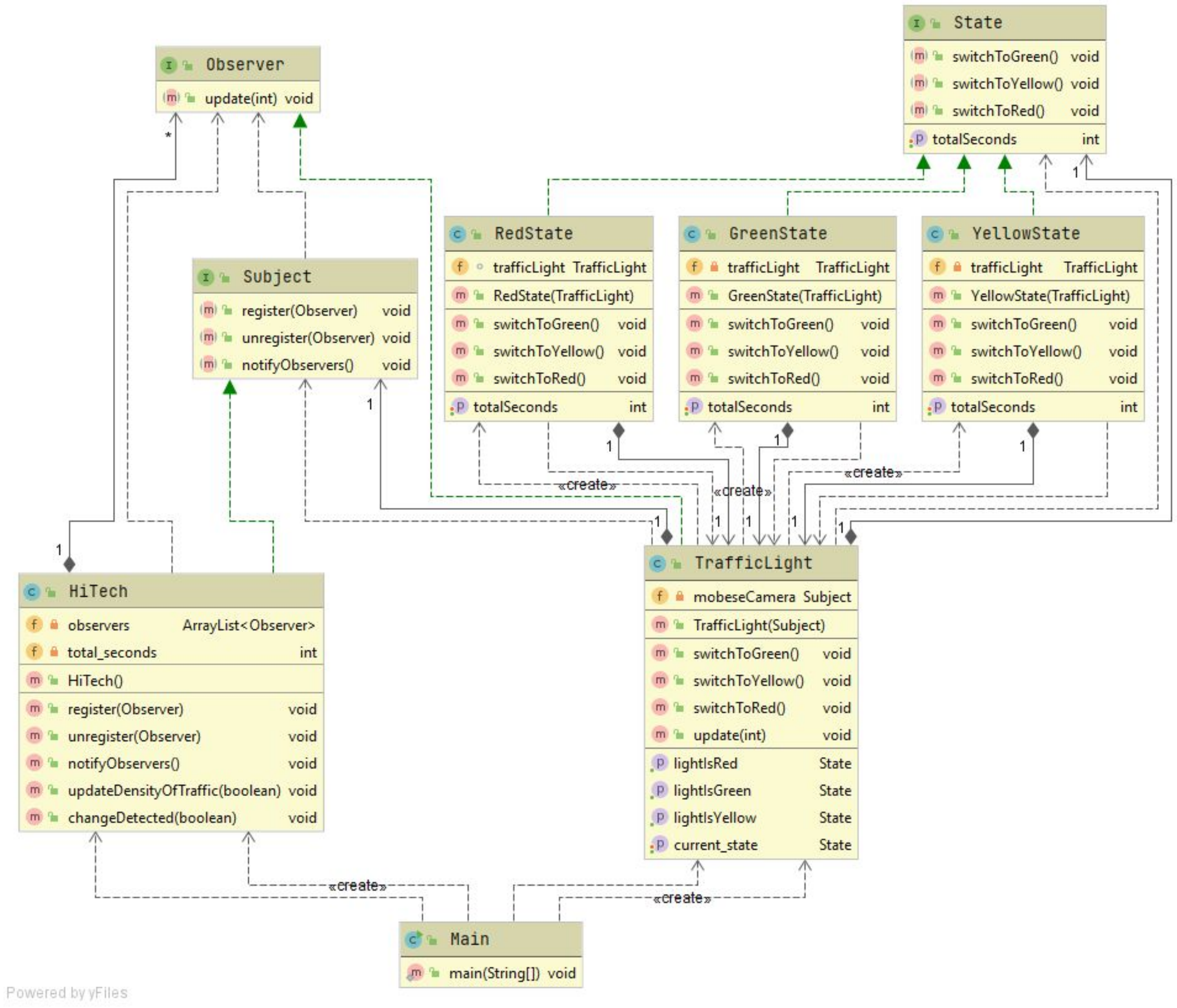


figure3.2.2 *UML diagram of the program that shows dependencies*

PART4 - PROXY DESIGN PATTERN

A. PART4 - PROXY DESIGN PATTERN

I. Explanation of Design Pattern

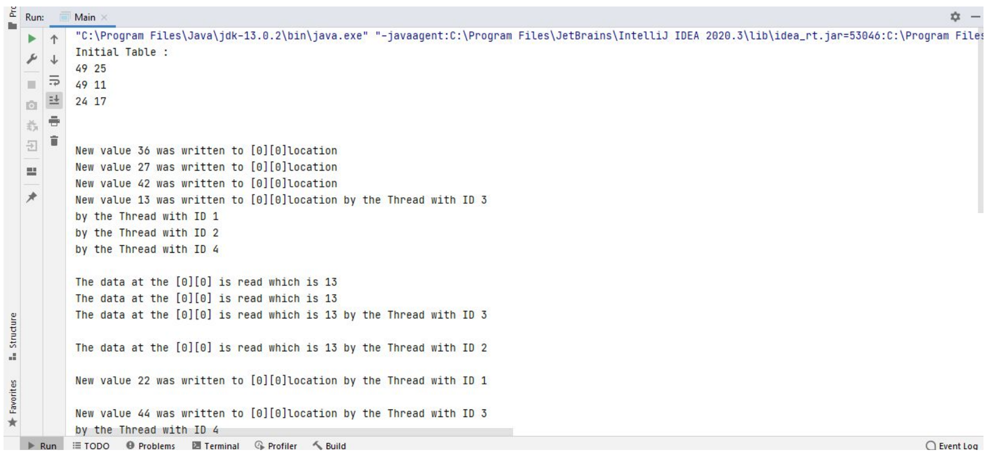
The purpose of this program is to provide synchronization capability to a class that we have and cannot change inside. Proxy design pattern is used to achieve this. Synchronization version of this pattern, which has many types, is used. This pattern acts as an intermediary between client and server, providing synchronization that the server cannot provide. This prevents clients from accessing the same lines at the same time and creating errors.

II. Explanations of Classes and Interfaces

There is an interface which is ITable and DataBaseTable class implements it. As mentioned in the homework, the content of this class cannot be changed and this class has no synchronization capability. So, a class which is called SynchronizationProxy is written to achieve the synchronization. Also, there is a class that has the main method and that is called Client. This class acts like a client that requests something from the server. But in this situation, proxy intervenes between them and provides synchronization to prevent error cases.

III. Execution Process

To test my methods, I created an instance of the DataBaseTable and SynchronizationProxy. There is a method to create threads and start them. In this method every thread has a duty which is writing to and reading from the database. For the sake of example, I created random numbers to set table cells. Additionally, every thread writes to the database if the loop counter is an even number, otherwise they read from the database. Since many threads are running at the same time and reaching the database, I can see if synchronization is achieved. There are information messages printed in the main method that shows the number of the working thread. Since they are not synchronized, text outputs can be complicated.



```
"C:\Program Files\Java\jdk-13.0.2\bin\java.exe" "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA 2020.3\lib\idea_rt.jar=53046:C:\Program Files
Initial Table :
49 25
49 11
24 17


New value 36 was written to [0][0]location
New value 27 was written to [0][0]location
New value 42 was written to [0][0]location
New value 13 was written to [0][0]location by the Thread with ID 3
by the Thread with ID 1
by the Thread with ID 2
by the Thread with ID 4

The data at the [0][0] is read which is 13
The data at the [0][0] is read which is 13
The data at the [0][0] is read which is 13 by the Thread with ID 3

The data at the [0][0] is read which is 13 by the Thread with ID 2

New value 22 was written to [0][0]location by the Thread with ID 1

New value 44 was written to [0][0]location by the Thread with ID 3
by the Thread with ID 4
```

figure4.1.1 *Example result of the execution*

IV. Class Dependencies and UML diagram

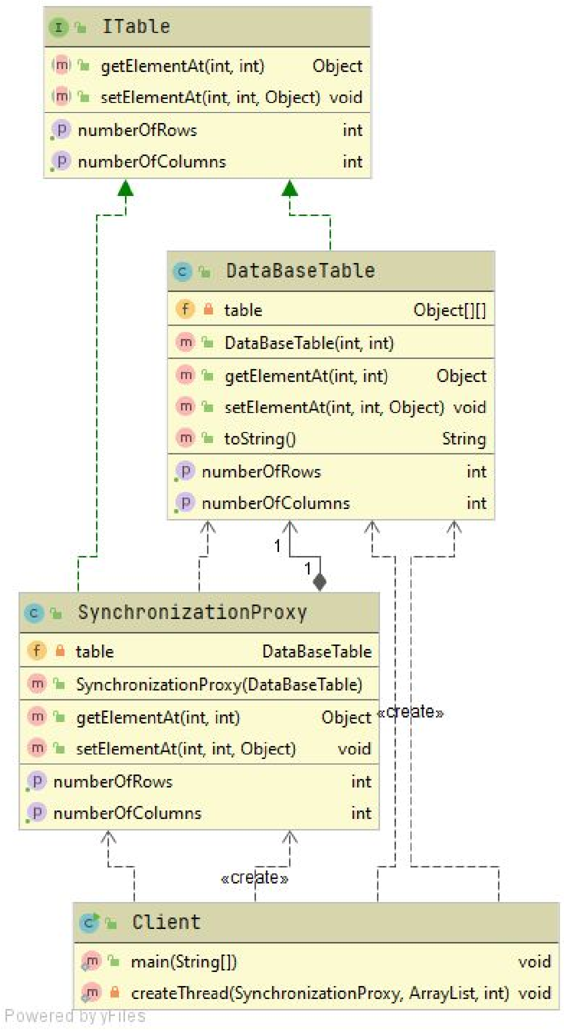

ITable
getElementAt(int, int) Object
setElementAt(int, int, Object) void
numberOfRows int
numberOfColumns int
DataBaseTable
table Object[][]
DataBaseTable(int, int)
getElementAt(int, int) Object
setElementAt(int, int, Object) void
toString() String
numberOfRows int
numberOfColumns int
1
1
SynchronizationProxy
table DataBaseTable
SynchronizationProxy(DataBaseTable)
getElementAt(int, int) Object
setElementAt(int, int, Object) void
numberOfRows int
numberOfColumns int
«create»
«create»
Client
main(String[]) void
createThread(SynchronizationProxy, ArrayList, int) void
Powered by yFiles

figure4.1.2 *UML diagram of the program that shows dependencies*

PART4 - PROXY DESIGN PATTERN

I. Explanation of Design Pattern

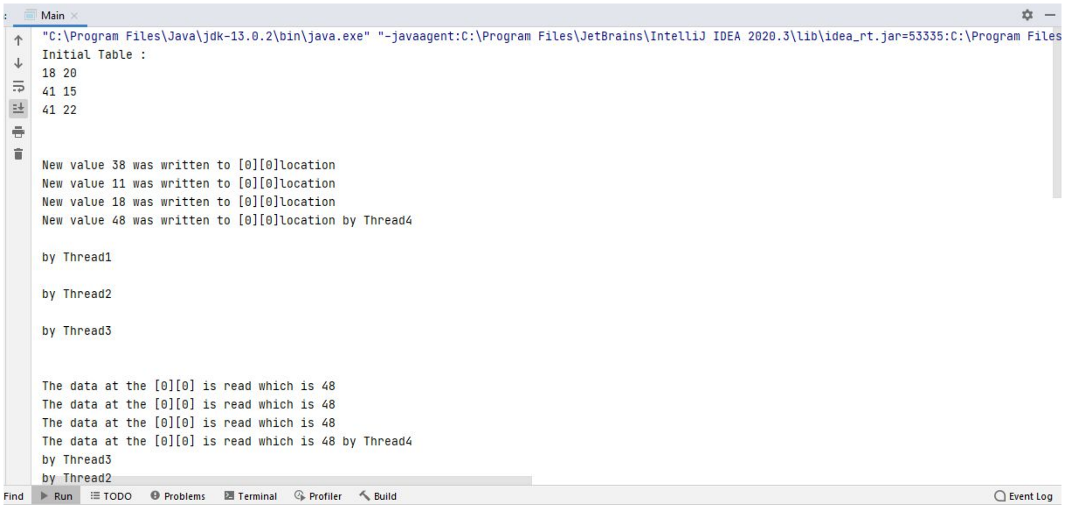
Since the synchronization of the program written in the first part is not very good, a new synchronization has been provided in this program. Writers are given more priority than readers. Again, the proxy pattern and the solution of the classic reader-writer problem are used in this design.

II. Explanations of Classes and Interfaces

All classes and interfaces are the same that is mentioned in the part1. The only difference in this part, writers are prioritized by using lock and Condition objects.

III. Execution Process

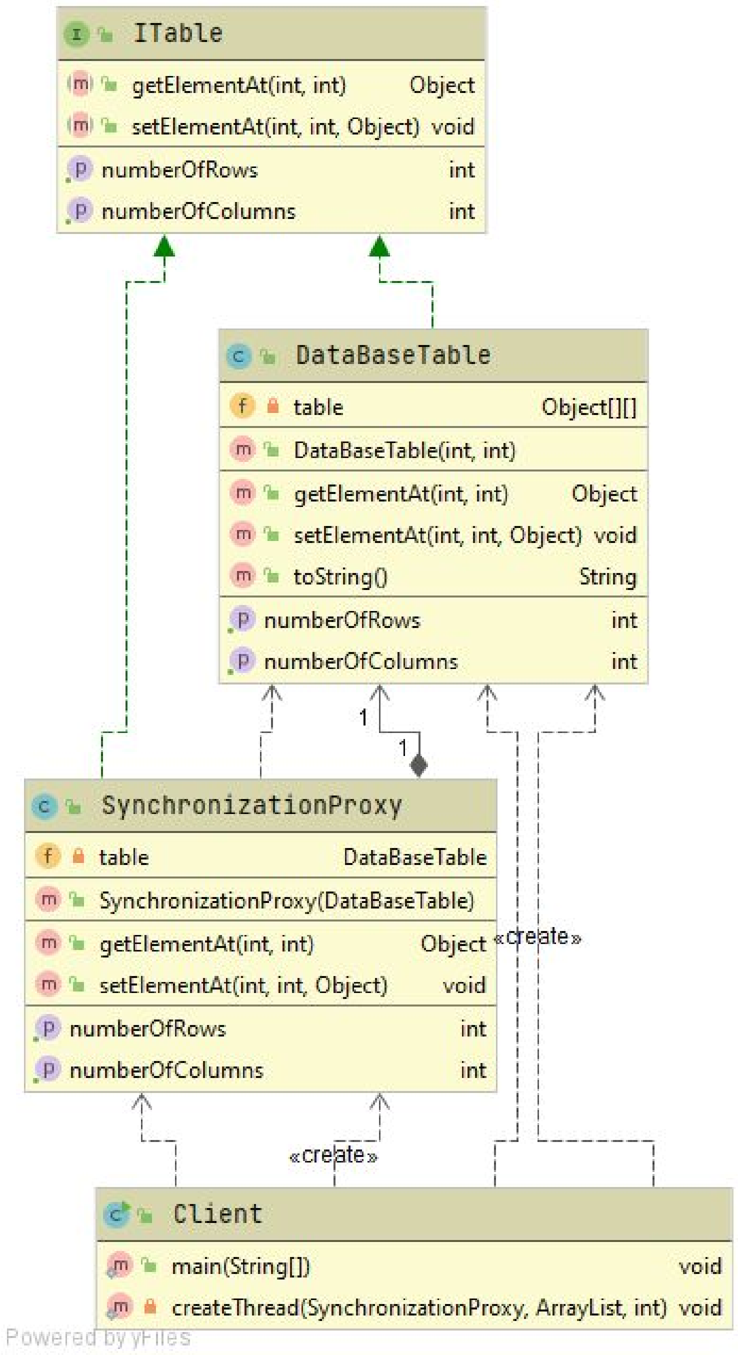
To test my methods, I created an instance of the DataBaseTable and SynchronizationProxy. There is a method to create threads and start them. In this method every thread has a duty which is writing to and reading from the database. For the sake of example, I created random numbers to set table cells. Additionally, every thread writes to the database if the loop counter is an even number, otherwise they read from the database. Since many threads are running at the same time and reaching the database, I can see if synchronization is achieved.There are information messages printed in the main method that shows the number of the working thread. Since they are not synchronized, text outputs can be complicated.



```
"C:\Program Files\Java\jdk-13.0.2\bin\java.exe" "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA 2020.3\lib\idea_rt.jar=53335:C:\Program Files
Initial Table :
18 20
41 15
41 22


New value 38 was written to [0][0]location
New value 11 was written to [0][0]location
New value 18 was written to [0][0]location
New value 48 was written to [0][0]location by Thread4

by Thread1

by Thread2

by Thread3


The data at the [0][0] is read which is 48
The data at the [0][0] is read which is 48
The data at the [0][0] is read which is 48
The data at the [0][0] is read which is 48 by Thread4
by Thread3
by Thread2
```

figure4.2.1 *Example result of the execution*

IV. Class Dependencies and UML diagram



ITable
getElementAt(int, int) Object
setElementAt(int, int, Object) void
numberOfRows int
numberOfColumns int
DataBaseTable
table Object[][]
DataBaseTable(int, int)
getElementAt(int, int) Object
setElementAt(int, int, Object) void
toString() String
numberOfRows int
numberOfColumns int
1
1
SynchronizationProxy
table DataBaseTable
SynchronizationProxy(DataBaseTable)
getElementAt(int, int) Object
setElementAt(int, int, Object) void
numberOfRows int
numberOfColumns int
«create»
«create»
Client
main(String[]) void
createThread(SynchronizationProxy, ArrayList, int) void
Powered by yFiles

figure4.2.2 *UML diagram of the program that shows dependencies*