CSE443 – Object Oriented Analysis and Desing

REPORT – HW2

Furkan ÖZEV

161044036

PART 1:

Question - 1:

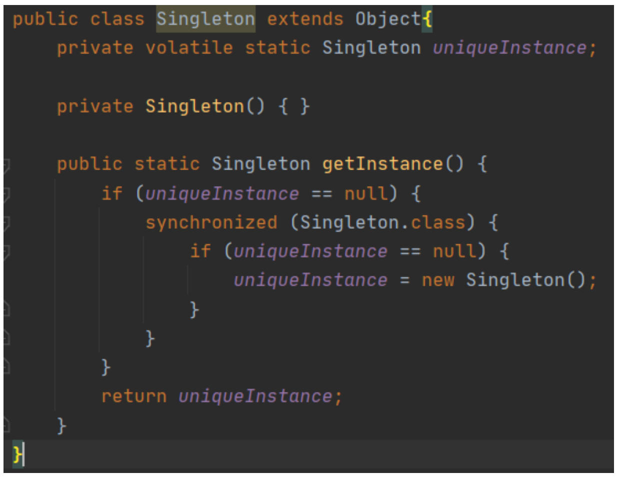
If someone tries to clone a Singleton object using the clone () method inherited from the Object, the code will not compile and the compiler will give an error message.

Error message: “java: clone () has protected access in java.lang.Object”.

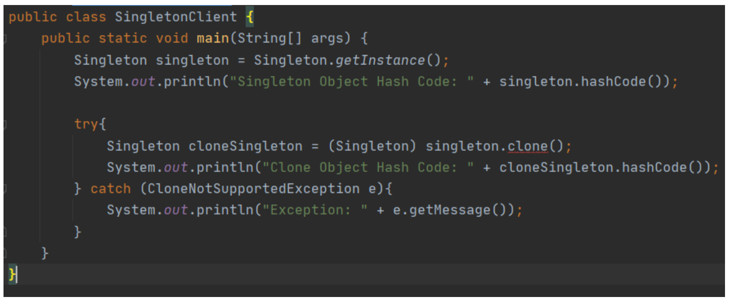
The default of the clone () method is a protected method. Therefore, it is not possible to create a second different Singleton object in this way.

As seen in the example, the error message has been received:

(The code shown in the examples has been added to the file.)

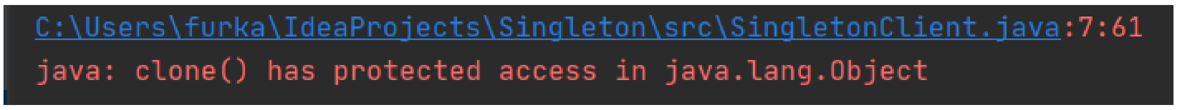


```java
public class Singleton extends Object{
    private volatile static Singleton uniqueInstance;

    private Singleton() { }

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



```java
public class SingletonClient {
    public static void main(String[] args) {
        Singleton singleton = Singleton.getInstance();
        System.out.println("Singleton Object Hash Code: " + singleton.hashCode());

        try{
            Singleton cloneSingleton = (Singleton) singleton.clone();
            System.out.println("Clone Object Hash Code: " + cloneSingleton.hashCode());
        } catch (CloneNotSupportedException e){
            System.out.println("Exception: " + e.getMessage());
        }
    }
}
```

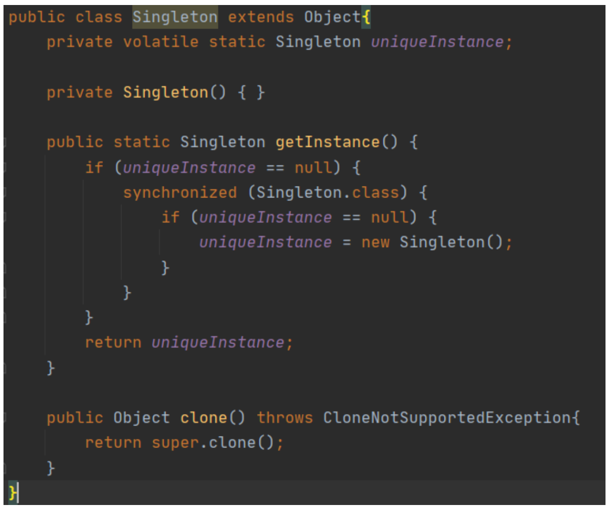
Output:



```
C:\Users\furka\IdeaProjects\Singleton\src\SingletonClient.java:7:61
java: clone() has protected access in java.lang.Object
```

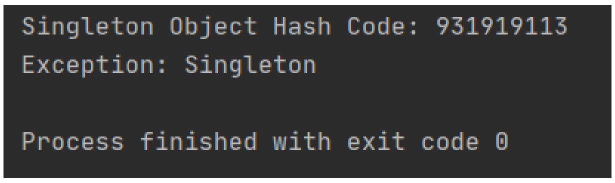
If we override the clone () method and try to clone it using the clone () method of inherited from the Object, the clone () method of the Object class will throw "CloneNotSupportedException" by default.

As seen in the example, an exception was caught:



```java
public class Singleton extends Object{
    private volatile static Singleton uniqueInstance;

    private Singleton() { }

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

    public Object clone() throws CloneNotSupportedException{
        return super.clone();
    }
}
```

Output:



```
Singleton Object Hash Code: 931919113
Exception: Singleton

Process finished with exit code 0
```

Thus, it does not lead to the creation of a second different Singleton object.

Question - 2:

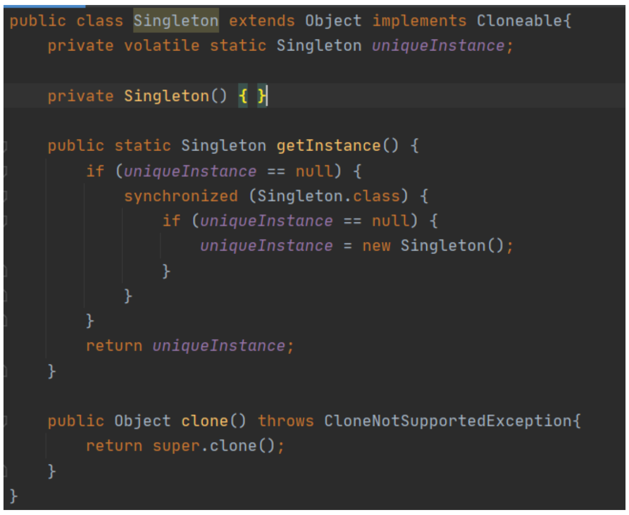
Due to the nature of the singleton design pattern, it should not be cloned. For this, prevention should be taken when necessary.

As I mentioned in the answer to the previous question, it is not possible to clone a Singleton object using the clone () method of inherited from the Object.

Question – 3.1:

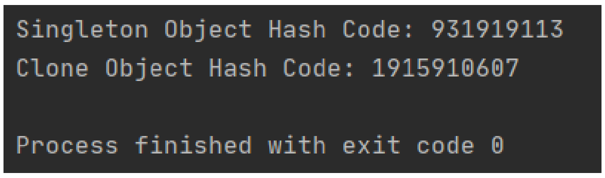
Consider the cases where the Singleton class derives from a class that fully implements the Cloneable interface, or the Singleton class directly implements the Cloneable interface:

As can be seen in the example, the Singleton object can be cloned without any problems.



```java
public class Singleton extends Object implements Cloneable{
    private volatile static Singleton uniqueInstance;

    private Singleton() { }

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

    public Object clone() throws CloneNotSupportedException{
        return super.clone();
    }
}
```

Output:



```
Singleton Object Hash Code: 931919113
Clone Object Hash Code: 1915910607

Process finished with exit code 0
```

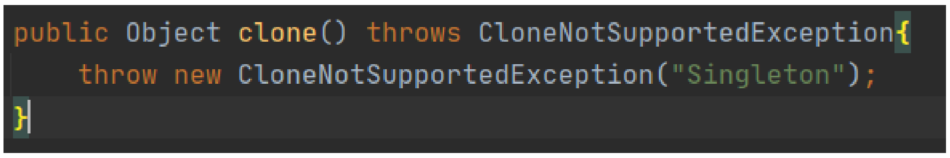
Thus, it leads to the creation of a second different Singleton object. In the ouput, you can see different 2 hash codes.

Question – 3.2:

Consider the cases where the Singleton class derives from a class that fully implements the Cloneable interface, or the Singleton class directly implements the Cloneable interface:

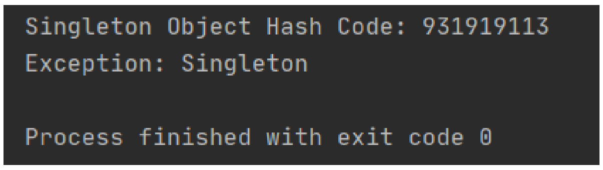
If we throw "CloneNotSupportedException" by overriding the clone () method, we can prevent the Singleton object from being cloned.

As can be seen in the example, the Singleton object can not be cloned, an exception was caught:



```java
public Object clone() throws CloneNotSupportedException{
    throw new CloneNotSupportedException("Singleton");
}
```

Output:



```
Singleton Object Hash Code: 931919113
Exception: Singleton

Process finished with exit code 0
```

PART 2:

As a data analyst, I am tasked with coding two iterator classes for data received from the 2023 Göktürk 3 satellite as 2D integer arrays.

One iterator that will print a 2D array spirally clockwise and another one that will print it spirally anti-clockwise, both starting at the top left element.

Thus, I designed TurkishSpaceAgency project using the **Iterator Design Pattern**.

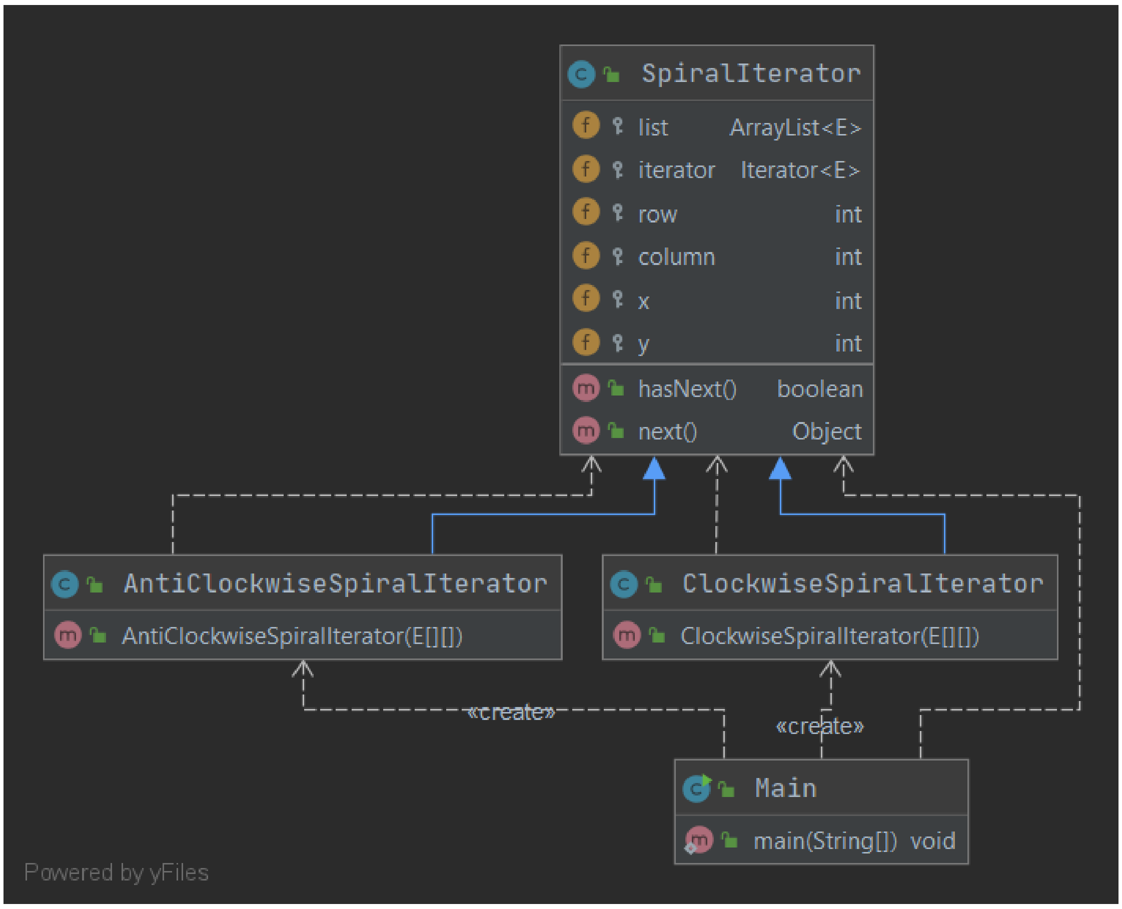
The Iterator Design Pattern provides a way to access the elements of an aggregate object sequentially without exposing its underlying representation.

In order to be suitable for all types of data types, the created iterator classes were made in Generic structure.

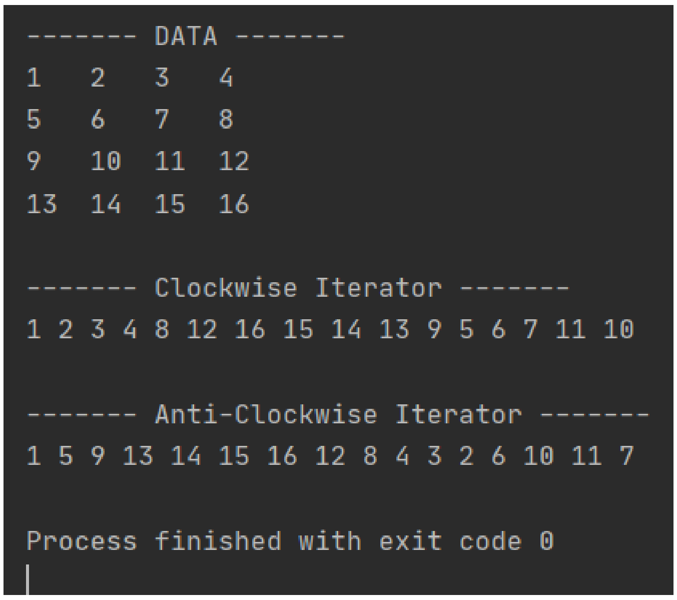
A abstract class named SpiralIterator was created. This class extends from Java's Iterator class. It maintains an array list and an Iterator object as the field. It also holds the dimension information of the 2D array with some variables. Overrides the hasNext () and next () methods inherited from the Iterator class. It calls the methods of the iterator, which it keeps as objects in these methods.

Later, 2 classes were created named ClockwiseSpiralIterator and AntiClockwiseSpiralIterator. These classes extends from SpiralIterator abstract class. It takes the 2D array as a parameter in the constructor of these classes. Creates a new arraylist. While traversing elements in the appropriate direction in a 2D array, these elements are added to the array. Here, 2 different algorithms are used for clockwise and anti-clockwise. Then this arraylist iterator is used as the iterator field. Thus, 2D array can be traversed spirally in any direction.

Class Diagram:



Output:



```
------- DATA -------
1   2   3   4
5   6   7   8
9   10  11  12
13  14  15  16

------- Clockwise Iterator -------
1 2 3 4 8 12 16 15 14 13 9 5 6 7 11 10

------- Anti-Clockwise Iterator -------
1 5 9 13 14 15 16 12 8 4 3 2 6 10 11 7

Process finished with exit code 0
```

PART 3:

In PDF, it was requested to implement the normal state first, then to implement it by adding the camera factor. So, in both cases, I added my codes to the file.

First Desing without HiTech MOBESE -> in **TrafficLight** folder

Second Desing with HiTech MOBESE -> in **HiTechTrafficLight** folder

Main method illustrate its functionality and print every state and every transition on the terminal as text output.

First Design without HiTech MOBESE:

There is a traffic light on the way to campus. It has three states (initially RED):

a) RED: switches to GREEN after 15 seconds.

b) YELLOW: switches to RED after 3 seconds.

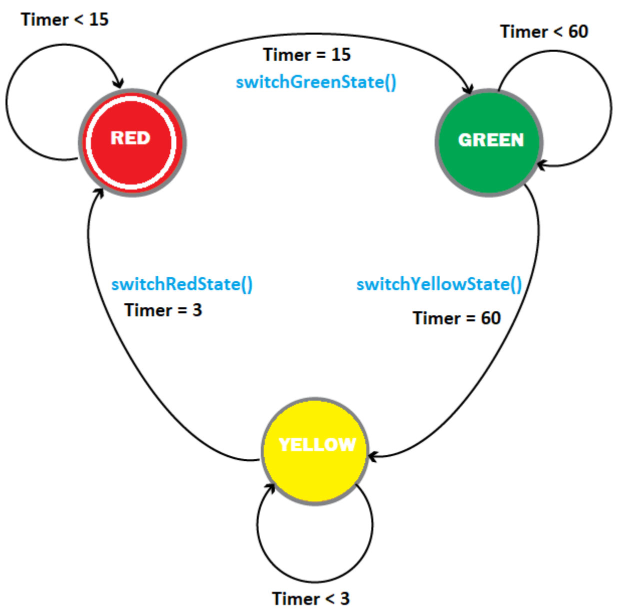
c) GREEN: switches to YELLOW after 60 seconds (timeout_X).

In this example there are multiple states and there are transitions between these states.

Thus, I designed Traffic Light project using the **State Design Pattern**.

The State Design Pattern allows an object to alter its behavior when its internal state changes. The object will appear to change its class.

State Diagram:



A separate class was created for each state. These states override existing methods to do what they need to do according to their situation.

Existing methods: switchRedState (), switchYellowState (), switchGreenState ()

For example, the Red State class prints an error message for the switchYellowState () method, checks the timeout for the switchGreenState () method and can change the state.

Also, the Green State class prints an error message for the switchRedState () method, checks the timeout for the switchYellowState () method and can change the state.

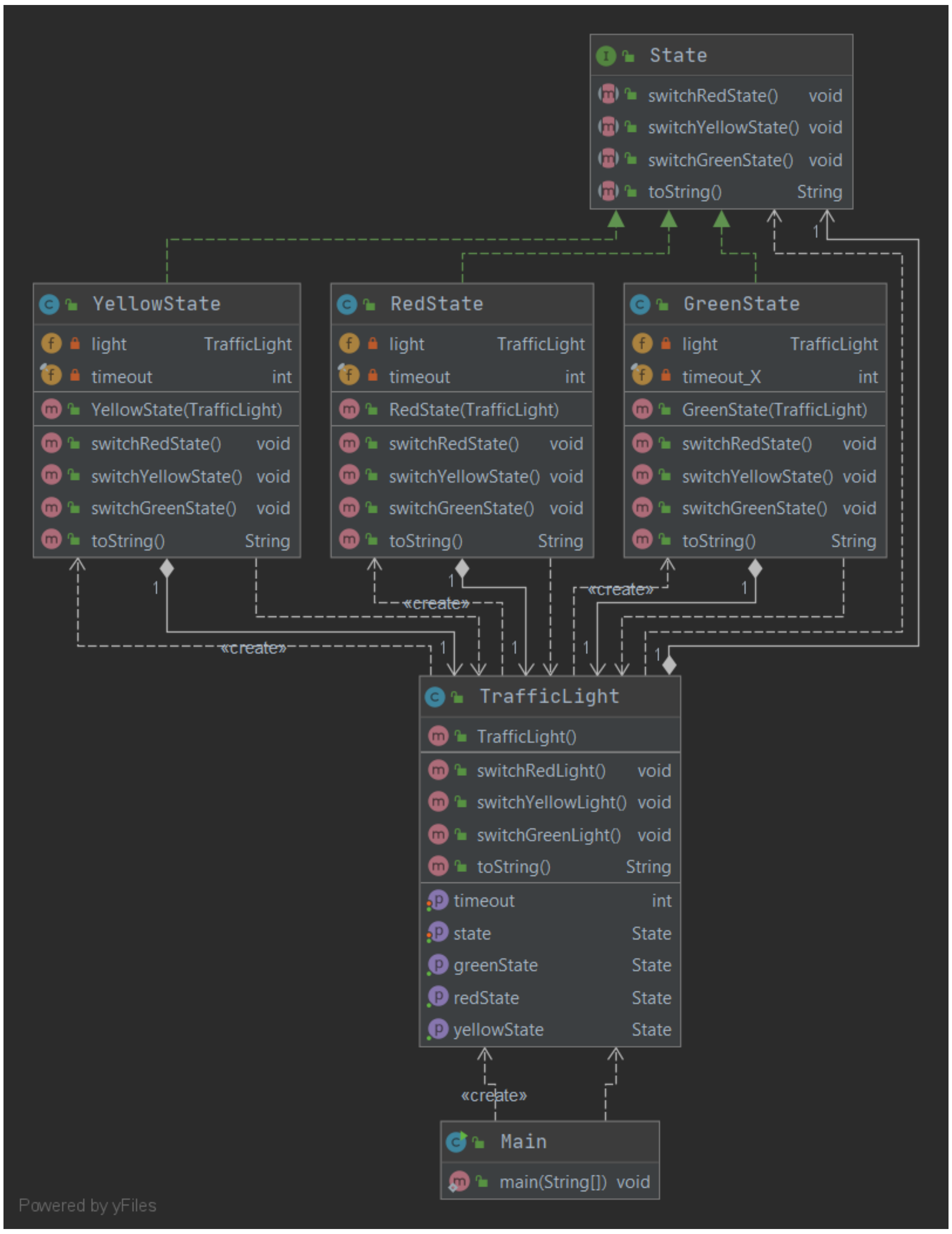
The TrafficLight class stores these stats as fields. It also includes a field to keep the current state and current timeout.

Uses current state's switch methods for switch operations.

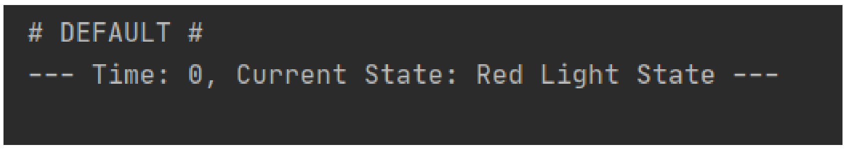
For example, currentState.switchRedState () is called in the switchRedLight () method.

It also includes auxiliary methods such as get and set.

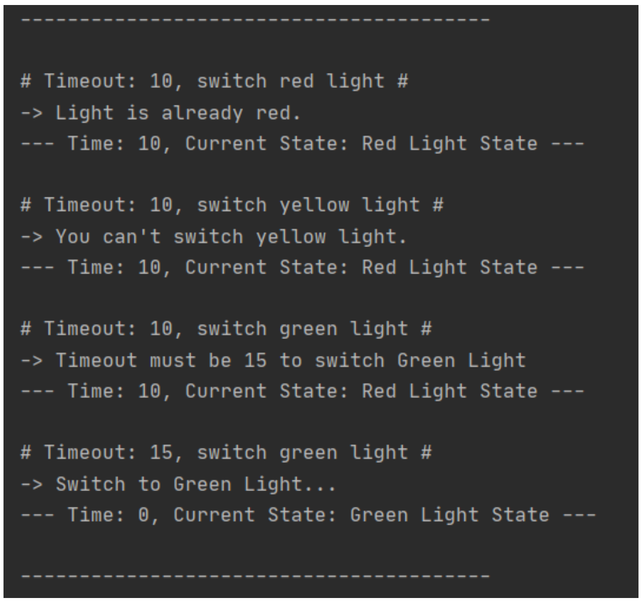
Class Diagram:



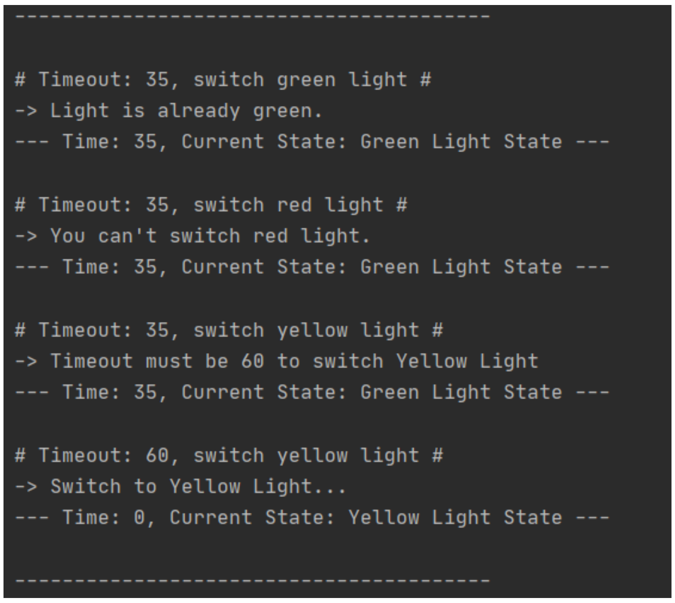
Output:



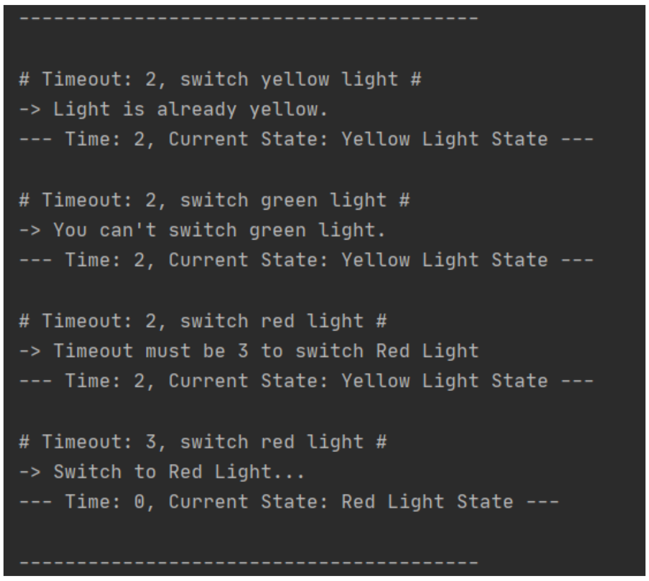
```
# DEFAULT #
--- Time: 0, Current State: Red Light State ---
```



```
----------------------------------------

# Timeout: 10, switch red light #
-> Light is already red.
--- Time: 10, Current State: Red Light State ---

# Timeout: 10, switch yellow light #
-> You can't switch yellow light.
--- Time: 10, Current State: Red Light State ---

# Timeout: 10, switch green light #
-> Timeout must be 15 to switch Green Light
--- Time: 10, Current State: Red Light State ---

# Timeout: 15, switch green light #
-> Switch to Green Light...
--- Time: 0, Current State: Green Light State ---

----------------------------------------
```



```
----------------------------------------

# Timeout: 35, switch green light #
-> Light is already green.
--- Time: 35, Current State: Green Light State ---

# Timeout: 35, switch red light #
-> You can't switch red light.
--- Time: 35, Current State: Green Light State ---

# Timeout: 35, switch yellow light #
-> Timeout must be 60 to switch Yellow Light
--- Time: 35, Current State: Green Light State ---

# Timeout: 60, switch yellow light #
-> Switch to Yellow Light...
--- Time: 0, Current State: Yellow Light State ---

----------------------------------------
```



```
----------------------------------------

# Timeout: 2, switch yellow light #
-> Light is already yellow.
--- Time: 2, Current State: Yellow Light State ---

# Timeout: 2, switch green light #
-> You can't switch green light.
--- Time: 2, Current State: Yellow Light State ---

# Timeout: 2, switch red light #
-> Timeout must be 3 to switch Red Light
--- Time: 2, Current State: Yellow Light State ---

# Timeout: 3, switch red light #
-> Switch to Red Light...
--- Time: 0, Current State: Red Light State ---

----------------------------------------
```

Second Design with HiTech MOBESE:

The states and methods are the same as the first design.

It was noticed that the roads surrounding the campus were sometimes overwhelmed with traffic due to unexpected events. And the situation is getting worse as they have to wait at this traffic light. For this reason, a MOBESE camera was placed above the traffic light, which measures the car density below. The timeout_X increases from 60 seconds to 90 seconds when it detects too much traffic.

More specifically, whenever the camera detects a change of traffic the method changeDetected of the class HiTech (provided by the software library of the camera) is called automagically by the hardware.

if flag is true, it means the traffic has increased substantially, otherwise (if false), everything is normal, so timeout_X returns to its initial value. It is up to you to fill the method.

This new component has been applied to the existing traffic light code and the class diagram has been redrawn.

To get traffic updates for your traffic light, the camera software has been subscribed and the timeout has been set accordingly.

Additionally, For HiTech Traffic Light Project, I used the **Observer Design Pattern** to apply new component with **Singleton Design Pattern**.

The Observer Design Pattern defines a one-to-many dependency between objects so that when one object changes state, all of its dependents are notified and updated automatically.

A class called HiTech was created. This class extends the Observable class of Java. When it detects heavy traffic it notify the observers.

The changeDetected () method takes a boolean as a parameter. This parameter indicates the traffic density. When this method is called, subscribers are awakened with the notify () method and the update () function of the subscribers runs.

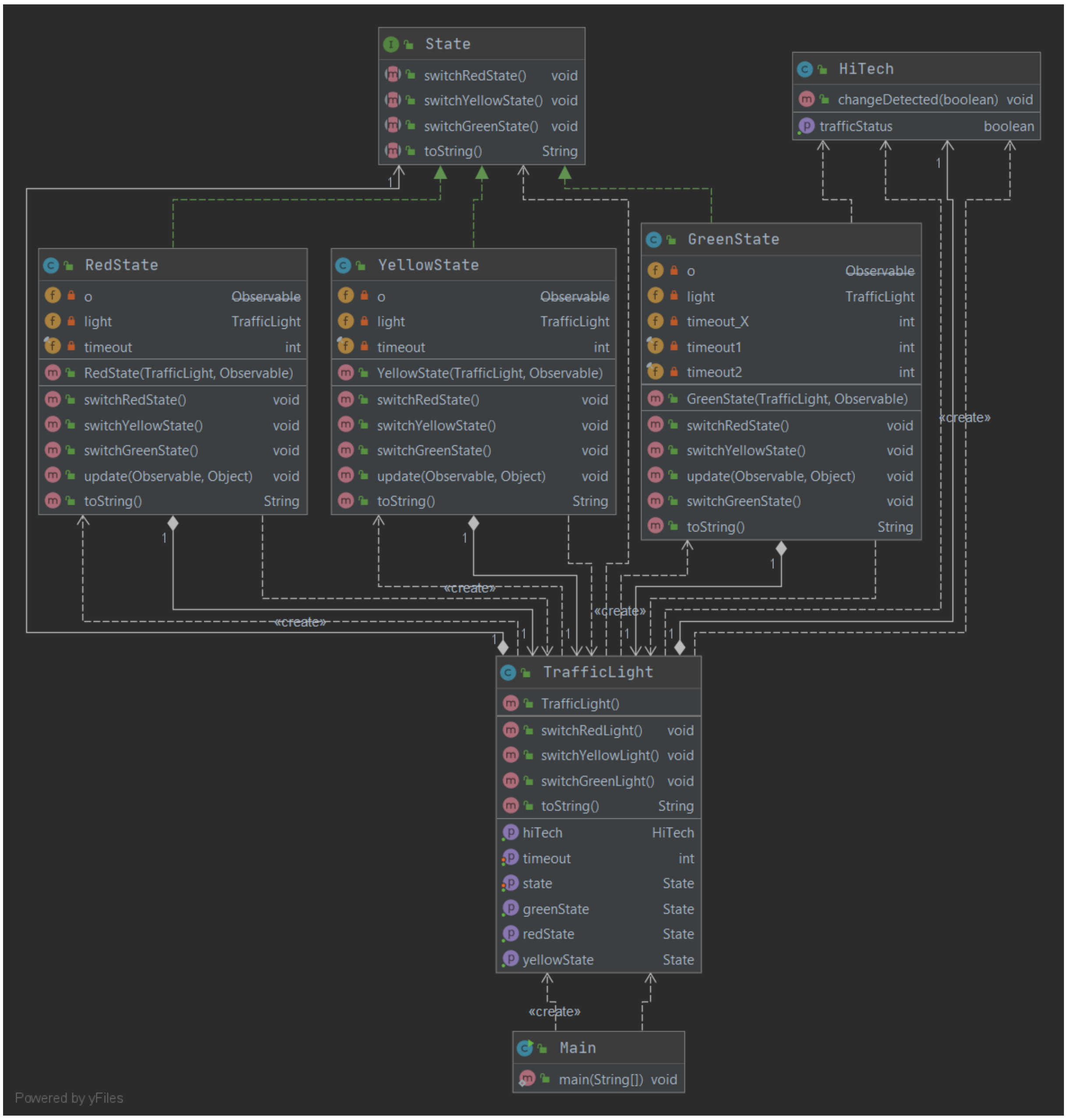
The TrafficLight class keeps the HiTech object as a field like states. It also uses this object as the constructor parameter when creating state objects. The states subscribe using this parameter, using the addObserver () method of the hiTech object.

State classes implement the Observer interface of Java because it is a subscriber.

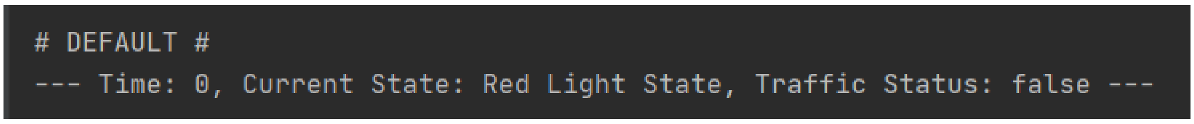
So they override the update () method.

For Red and Yellow States, the update () method doesn't need to do anything, so the method is empty. The function can be implemented if needed in the future. For Green State, the timeout_X will change according to the traffic density. Timeout_X will be determined each time update () method is called.

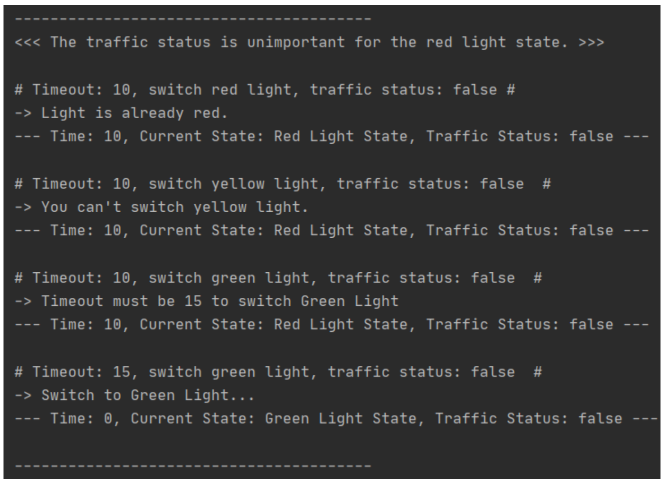
Class Diagram:



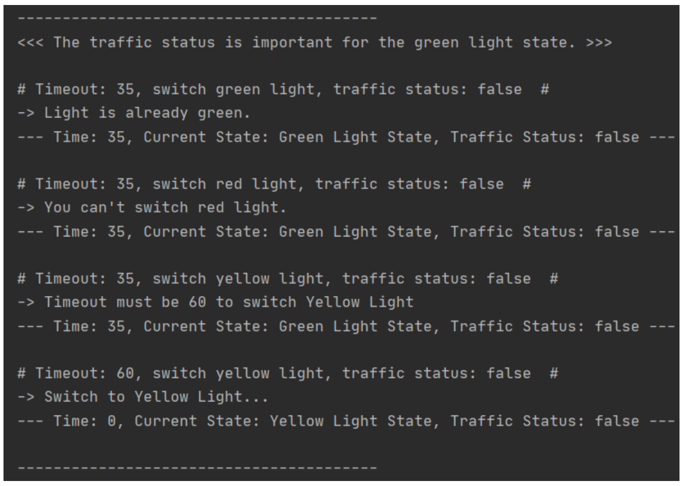
Output:



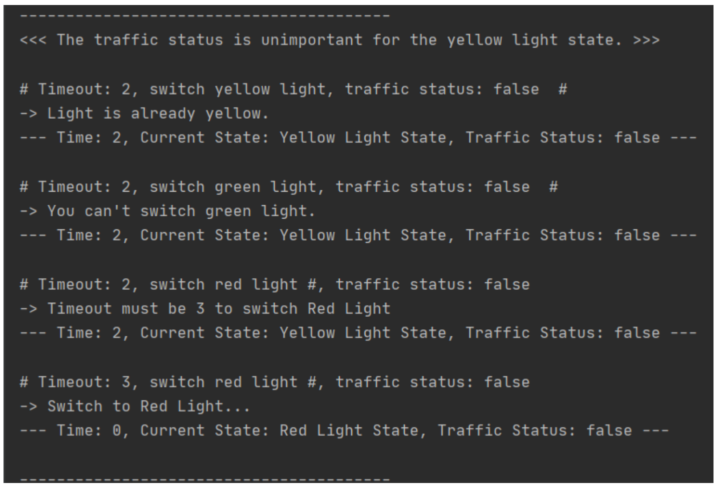
```
# DEFAULT #
--- Time: 0, Current State: Red Light State, Traffic Status: false ---
```



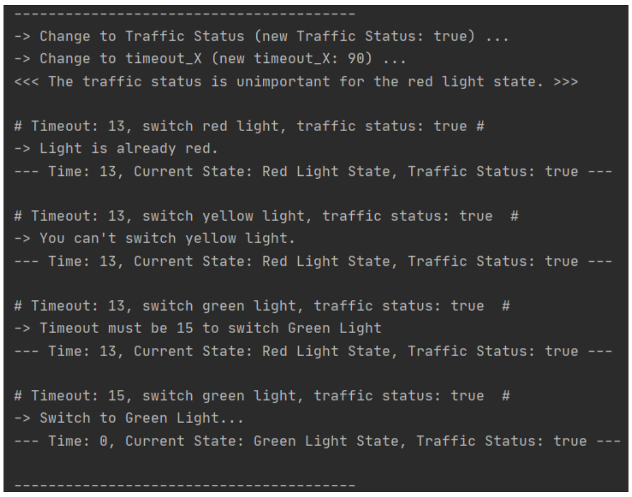
```
-----------------------------------------
<<< The traffic status is unimportant for the red light state. >>>

# Timeout: 10, switch red light, traffic status: false #
-> Light is already red.
--- Time: 10, Current State: Red Light State, Traffic Status: false ---

# Timeout: 10, switch yellow light, traffic status: false  #
-> You can't switch yellow light.
--- Time: 10, Current State: Red Light State, Traffic Status: false ---

# Timeout: 10, switch green light, traffic status: false  #
-> Timeout must be 15 to switch Green Light
--- Time: 10, Current State: Red Light State, Traffic Status: false ---

# Timeout: 15, switch green light, traffic status: false  #
-> Switch to Green Light...
--- Time: 0, Current State: Green Light State, Traffic Status: false ---

-----------------------------------------
```



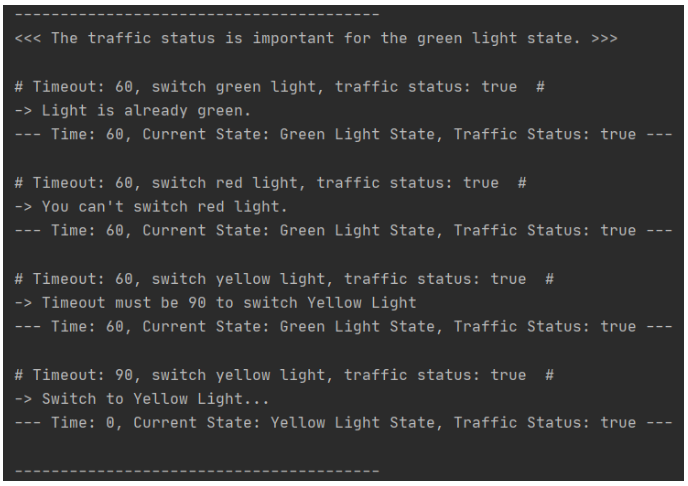
```
-----------------------------------------
<<< The traffic status is important for the green light state. >>>

# Timeout: 35, switch green light, traffic status: false  #
-> Light is already green.
--- Time: 35, Current State: Green Light State, Traffic Status: false ---

# Timeout: 35, switch red light, traffic status: false  #
-> You can't switch red light.
--- Time: 35, Current State: Green Light State, Traffic Status: false ---

# Timeout: 35, switch yellow light, traffic status: false  #
-> Timeout must be 60 to switch Yellow Light
--- Time: 35, Current State: Green Light State, Traffic Status: false ---

# Timeout: 60, switch yellow light, traffic status: false  #
-> Switch to Yellow Light...
--- Time: 0, Current State: Yellow Light State, Traffic Status: false ---

-----------------------------------------
```



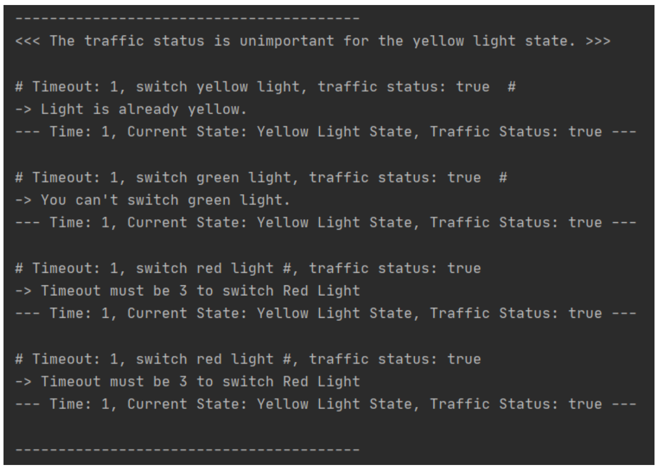
```
-----------------------------------------
<<< The traffic status is unimportant for the yellow light state. >>>

# Timeout: 2, switch yellow light, traffic status: false  #
-> Light is already yellow.
--- Time: 2, Current State: Yellow Light State, Traffic Status: false ---

# Timeout: 2, switch green light, traffic status: false  #
-> You can't switch green light.
--- Time: 2, Current State: Yellow Light State, Traffic Status: false ---

# Timeout: 2, switch red light #, traffic status: false
-> Timeout must be 3 to switch Red Light
--- Time: 2, Current State: Yellow Light State, Traffic Status: false ---

# Timeout: 3, switch red light #, traffic status: false
-> Switch to Red Light...
--- Time: 0, Current State: Red Light State, Traffic Status: false ---

-----------------------------------------
```



```
-----------------------------------------
-> Change to Traffic Status (new Traffic Status: true) ...
-> Change to timeout_X (new timeout_X: 90) ...
<<< The traffic status is unimportant for the red light state. >>>

# Timeout: 13, switch red light, traffic status: true #
-> Light is already red.
--- Time: 13, Current State: Red Light State, Traffic Status: true ---

# Timeout: 13, switch yellow light, traffic status: true  #
-> You can't switch yellow light.
--- Time: 13, Current State: Red Light State, Traffic Status: true ---

# Timeout: 13, switch green light, traffic status: true  #
-> Timeout must be 15 to switch Green Light
--- Time: 13, Current State: Red Light State, Traffic Status: true ---

# Timeout: 15, switch green light, traffic status: true  #
-> Switch to Green Light...
--- Time: 0, Current State: Green Light State, Traffic Status: true ---

-----------------------------------------
```



```
-----------------------------------------
<<< The traffic status is important for the green light state. >>>

# Timeout: 60, switch green light, traffic status: true  #
-> Light is already green.
--- Time: 60, Current State: Green Light State, Traffic Status: true ---

# Timeout: 60, switch red light, traffic status: true  #
-> You can't switch red light.
--- Time: 60, Current State: Green Light State, Traffic Status: true ---

# Timeout: 60, switch yellow light, traffic status: true  #
-> Timeout must be 90 to switch Yellow Light
--- Time: 60, Current State: Green Light State, Traffic Status: true ---

# Timeout: 90, switch yellow light, traffic status: true  #
-> Switch to Yellow Light...
--- Time: 0, Current State: Yellow Light State, Traffic Status: true ---

-----------------------------------------
```



```
-----------------------------------------
<<< The traffic status is unimportant for the yellow light state. >>>

# Timeout: 1, switch yellow light, traffic status: true  #
-> Light is already yellow.
--- Time: 1, Current State: Yellow Light State, Traffic Status: true ---

# Timeout: 1, switch green light, traffic status: true  #
-> You can't switch green light.
--- Time: 1, Current State: Yellow Light State, Traffic Status: true ---

# Timeout: 1, switch red light #, traffic status: true
-> Timeout must be 3 to switch Red Light
--- Time: 1, Current State: Yellow Light State, Traffic Status: true ---

# Timeout: 1, switch red light #, traffic status: true
-> Timeout must be 3 to switch Red Light
--- Time: 1, Current State: Yellow Light State, Traffic Status: true ---

-----------------------------------------
```

PART 4:

In pdf, Company has an old software library providing a class DataBaseTable, but this class does not provide the capability to allow clients to lock individual table rows.

Thus, two clients might end up modifying the same row simultaneously, and we don’t want that happening.

Moreover, we do not have the source code for this class library, but we have the complete documentation and know about the interface ITable implemented by the DataBaseTable class.

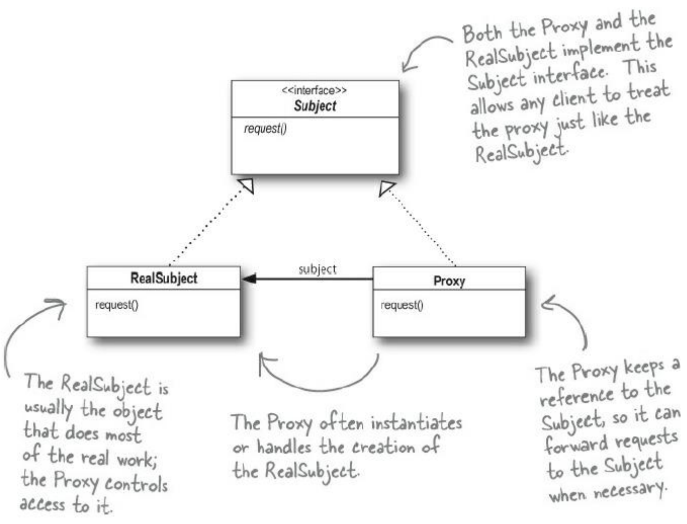
I designed TurkishSpaceAgency project using the **Proxy Design Pattern**.

The Proxy Design Pattern provides a surrogate or placeholder for another object to control access to it.

PART a) :

Proxy design pattern was used in order to equip the DataBaseTable class’ objects with synchronization capability.

No reader thread calls getElementAt while a writer thread is executing setElementAt.



<<interface>>
Subject
request()
RealSubject
request()
subject
Proxy
request()
Both the Proxy and the RealSubject implement the Subject interface. This allows any client to treat the proxy just like the RealSubject.
The RealSubject is usually the object that does most of the real work; the Proxy controls access to it.
The Proxy often instantiates or handles the creation of the RealSubject.
The Proxy keeps a reference to the Subject, so it can forward requests to the Subject when necessary.

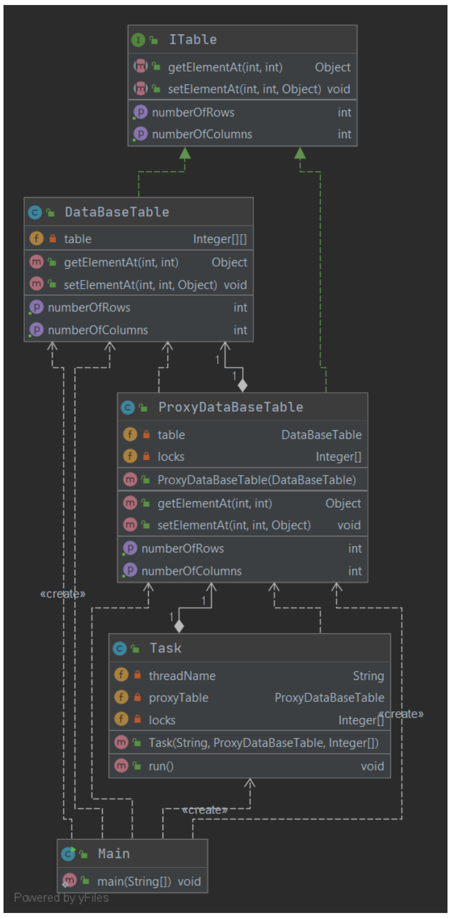
An object is reserved for each line.

These objects are used to synchronize the get and set functions.

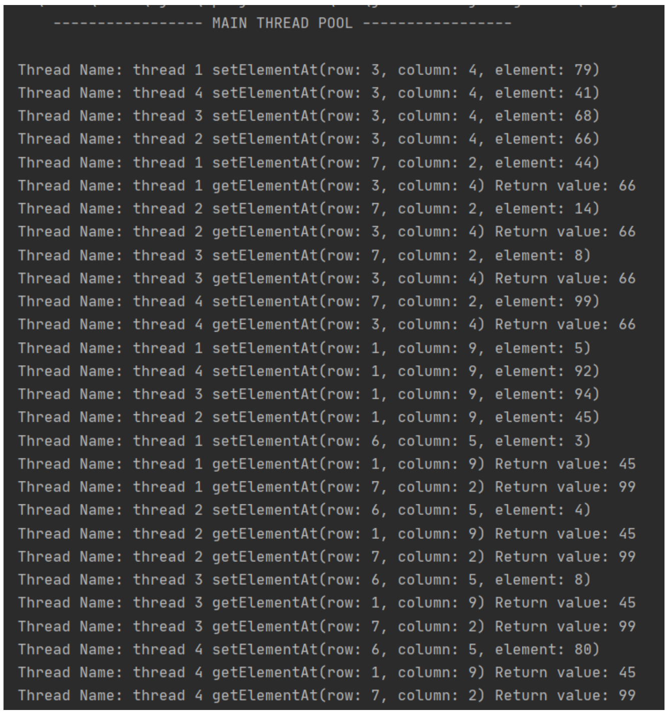
When the get and set functions are called, these objects are used to lock them.

For synchronized, lock row and call database object method with row and column in the functions.

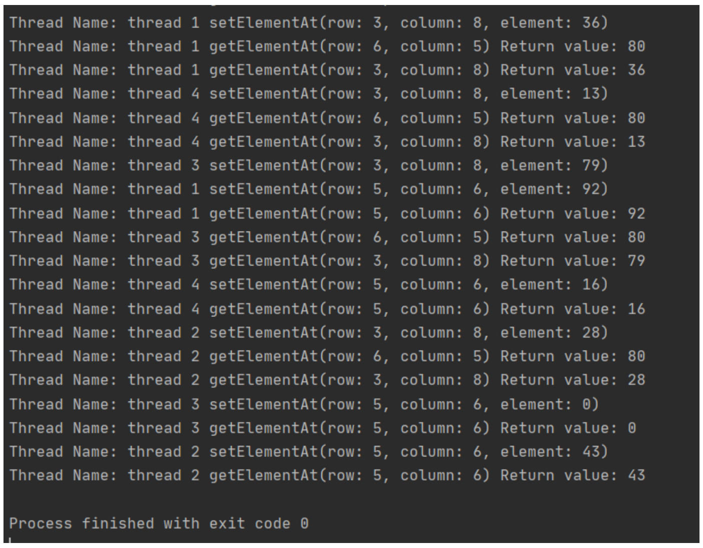
Class Diagram:



Output:



```
    ---------------- MAIN THREAD POOL ----------------

Thread Name: thread 1 setElementAt(row: 3, column: 4, element: 79)
Thread Name: thread 4 setElementAt(row: 3, column: 4, element: 41)
Thread Name: thread 3 setElementAt(row: 3, column: 4, element: 68)
Thread Name: thread 2 setElementAt(row: 3, column: 4, element: 66)
Thread Name: thread 1 setElementAt(row: 7, column: 2, element: 44)
Thread Name: thread 1 getElementAt(row: 3, column: 4) Return value: 66
Thread Name: thread 2 setElementAt(row: 7, column: 2, element: 14)
Thread Name: thread 2 getElementAt(row: 3, column: 4) Return value: 66
Thread Name: thread 3 setElementAt(row: 7, column: 2, element: 8)
Thread Name: thread 3 getElementAt(row: 3, column: 4) Return value: 66
Thread Name: thread 4 setElementAt(row: 7, column: 2, element: 99)
Thread Name: thread 4 getElementAt(row: 3, column: 4) Return value: 66
Thread Name: thread 1 setElementAt(row: 1, column: 9, element: 5)
Thread Name: thread 4 setElementAt(row: 1, column: 9, element: 92)
Thread Name: thread 3 setElementAt(row: 1, column: 9, element: 94)
Thread Name: thread 2 setElementAt(row: 1, column: 9, element: 45)
Thread Name: thread 1 setElementAt(row: 6, column: 5, element: 3)
Thread Name: thread 1 getElementAt(row: 1, column: 9) Return value: 45
Thread Name: thread 1 getElementAt(row: 7, column: 2) Return value: 99
Thread Name: thread 2 setElementAt(row: 6, column: 5, element: 4)
Thread Name: thread 2 getElementAt(row: 1, column: 9) Return value: 45
Thread Name: thread 2 getElementAt(row: 7, column: 2) Return value: 99
Thread Name: thread 3 setElementAt(row: 6, column: 5, element: 8)
Thread Name: thread 3 getElementAt(row: 1, column: 9) Return value: 45
Thread Name: thread 3 getElementAt(row: 7, column: 2) Return value: 99
Thread Name: thread 4 setElementAt(row: 6, column: 5, element: 80)
Thread Name: thread 4 getElementAt(row: 1, column: 9) Return value: 45
Thread Name: thread 4 getElementAt(row: 7, column: 2) Return value: 99
```



```
Thread Name: thread 1 setElementAt(row: 3, column: 8, element: 36)
Thread Name: thread 1 getElementAt(row: 6, column: 5) Return value: 80
Thread Name: thread 1 getElementAt(row: 3, column: 8) Return value: 36
Thread Name: thread 4 setElementAt(row: 3, column: 8, element: 13)
Thread Name: thread 4 getElementAt(row: 6, column: 5) Return value: 80
Thread Name: thread 4 getElementAt(row: 3, column: 8) Return value: 13
Thread Name: thread 3 setElementAt(row: 3, column: 8, element: 79)
Thread Name: thread 1 setElementAt(row: 5, column: 6, element: 92)
Thread Name: thread 1 getElementAt(row: 5, column: 6) Return value: 92
Thread Name: thread 3 getElementAt(row: 6, column: 5) Return value: 80
Thread Name: thread 3 getElementAt(row: 3, column: 8) Return value: 79
Thread Name: thread 4 setElementAt(row: 5, column: 6, element: 16)
Thread Name: thread 4 getElementAt(row: 5, column: 6) Return value: 16
Thread Name: thread 2 setElementAt(row: 3, column: 8, element: 28)
Thread Name: thread 2 getElementAt(row: 6, column: 5) Return value: 80
Thread Name: thread 2 getElementAt(row: 3, column: 8) Return value: 28
Thread Name: thread 3 setElementAt(row: 5, column: 6, element: 0)
Thread Name: thread 3 getElementAt(row: 5, column: 6) Return value: 0
Thread Name: thread 2 setElementAt(row: 5, column: 6, element: 43)
Thread Name: thread 2 getElementAt(row: 5, column: 6) Return value: 43

Process finished with exit code 0
```

PART b) :

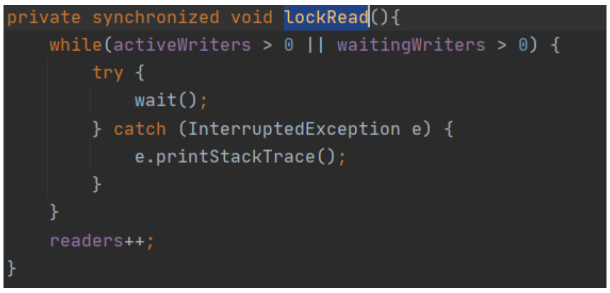
Clients are unhappy with your DataBaseTable synchronization proxy.

There are too many getElementAt calls that keep “locking” the Table’s rows, and this way the clients that need to modify them using setElementAt wait too long to acquire the table lock.

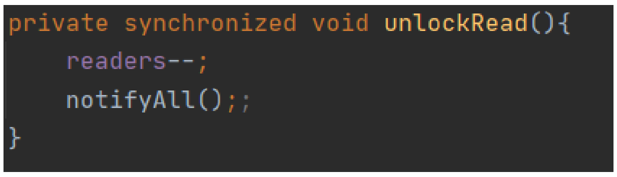
I solved this problem and designed a new synchronization solution that prioritizes writers of DataBaseTable more than readers in terms of table row access.

Writer priorities logic in Java is used.

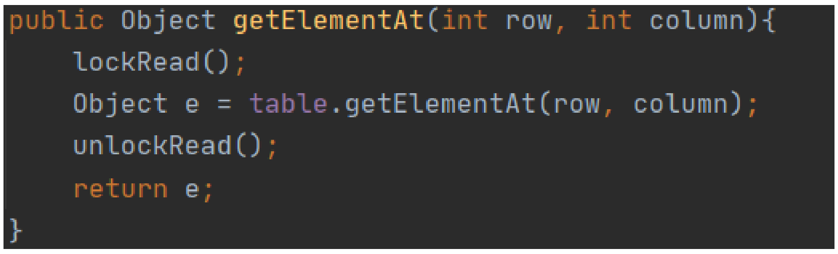
The lockRead () and unlockRead () methods are written. Among these methods, reading from the database is performed.



```java
private synchronized void lockRead(){
    while(activeWriters > 0 || waitingWriters > 0) {
        try {
            wait();
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }
    readers++;
}
```

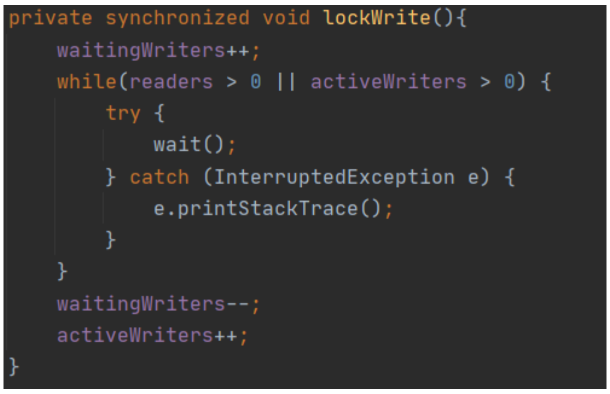


```java
private synchronized void unlockRead(){
    readers--;
    notifyAll();;
}
```

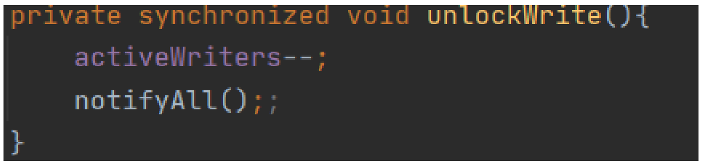


```java
public Object getElementAt(int row, int column){
    lockRead();
    Object e = table.getElementAt(row, column);
    unlockRead();
    return e;
}
```

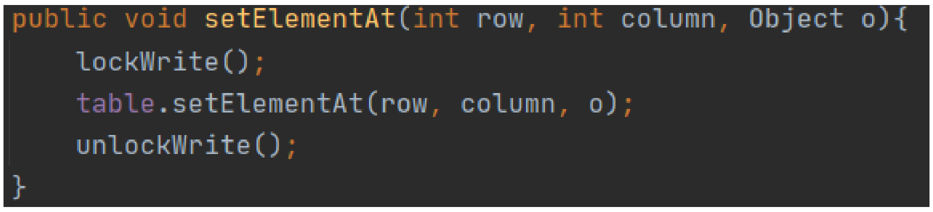
The lockWrite () and unlockWrite () methods are written. Among these methods, writing from the database is performed.



```java
private synchronized void lockWrite(){
    waitingWriters++;
    while(readers > 0 || activeWriters > 0) {
        try {
            wait();
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }
    waitingWriters--;
    activeWriters++;
}
```

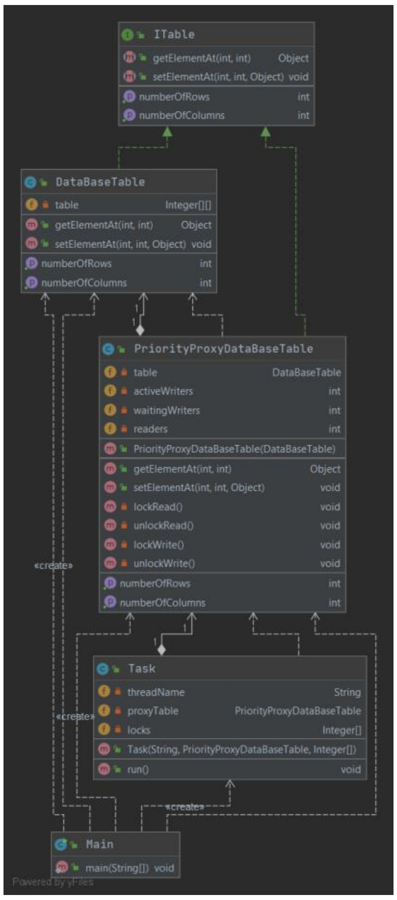


```java
private synchronized void unlockWrite(){
    activeWriters--;
    notifyAll();;
}
```

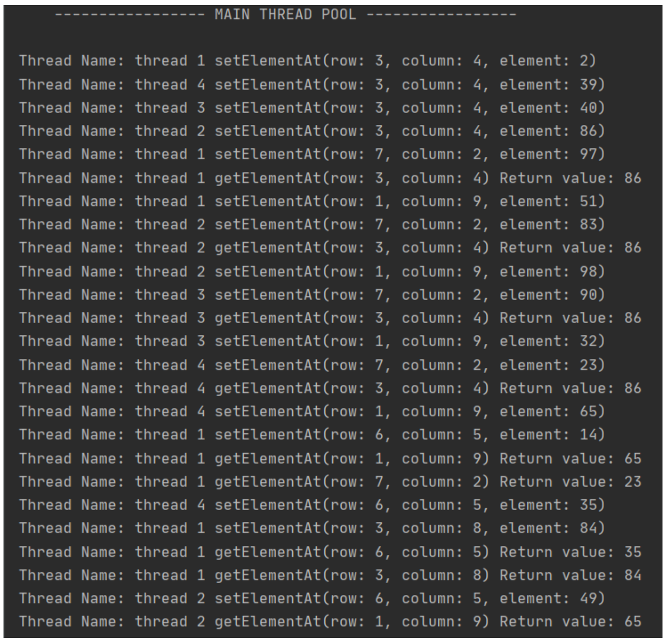


```java
public void setElementAt(int row, int column, Object o){
    lockWrite();
    table.setElementAt(row, column, o);
    unlockWrite();
}
```

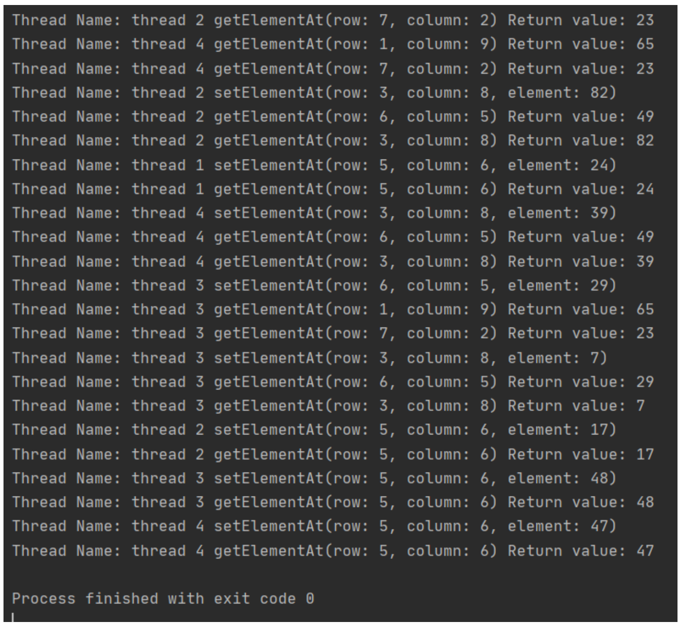
Class Diagram:



Output:



```
    ---------------- MAIN THREAD POOL ----------------

Thread Name: thread 1 setElementAt(row: 3, column: 4, element: 2)
Thread Name: thread 4 setElementAt(row: 3, column: 4, element: 39)
Thread Name: thread 3 setElementAt(row: 3, column: 4, element: 40)
Thread Name: thread 2 setElementAt(row: 3, column: 4, element: 86)
Thread Name: thread 1 setElementAt(row: 7, column: 2, element: 97)
Thread Name: thread 1 getElementAt(row: 3, column: 4) Return value: 86
Thread Name: thread 1 setElementAt(row: 1, column: 9, element: 51)
Thread Name: thread 2 setElementAt(row: 7, column: 2, element: 83)
Thread Name: thread 2 getElementAt(row: 3, column: 4) Return value: 86
Thread Name: thread 2 setElementAt(row: 1, column: 9, element: 98)
Thread Name: thread 3 setElementAt(row: 7, column: 2, element: 90)
Thread Name: thread 3 getElementAt(row: 3, column: 4) Return value: 86
Thread Name: thread 3 setElementAt(row: 1, column: 9, element: 32)
Thread Name: thread 4 setElementAt(row: 7, column: 2, element: 23)
Thread Name: thread 4 getElementAt(row: 3, column: 4) Return value: 86
Thread Name: thread 4 setElementAt(row: 1, column: 9, element: 65)
Thread Name: thread 1 setElementAt(row: 6, column: 5, element: 14)
Thread Name: thread 1 getElementAt(row: 1, column: 9) Return value: 65
Thread Name: thread 1 getElementAt(row: 7, column: 2) Return value: 23
Thread Name: thread 4 setElementAt(row: 6, column: 5, element: 35)
Thread Name: thread 1 setElementAt(row: 3, column: 8, element: 84)
Thread Name: thread 1 getElementAt(row: 6, column: 5) Return value: 35
Thread Name: thread 1 getElementAt(row: 3, column: 8) Return value: 84
Thread Name: thread 2 setElementAt(row: 6, column: 5, element: 49)
Thread Name: thread 2 getElementAt(row: 1, column: 9) Return value: 65
```



```
Thread Name: thread 2 getElementAt(row: 7, column: 2) Return value: 23
Thread Name: thread 4 getElementAt(row: 1, column: 9) Return value: 65
Thread Name: thread 4 getElementAt(row: 7, column: 2) Return value: 23
Thread Name: thread 2 setElementAt(row: 3, column: 8, element: 82)
Thread Name: thread 2 getElementAt(row: 6, column: 5) Return value: 49
Thread Name: thread 2 getElementAt(row: 3, column: 8) Return value: 82
Thread Name: thread 1 setElementAt(row: 5, column: 6, element: 24)
Thread Name: thread 1 getElementAt(row: 5, column: 6) Return value: 24
Thread Name: thread 4 setElementAt(row: 3, column: 8, element: 39)
Thread Name: thread 4 getElementAt(row: 6, column: 5) Return value: 49
Thread Name: thread 4 getElementAt(row: 3, column: 8) Return value: 39
Thread Name: thread 3 setElementAt(row: 6, column: 5, element: 29)
Thread Name: thread 3 getElementAt(row: 1, column: 9) Return value: 65
Thread Name: thread 3 getElementAt(row: 7, column: 2) Return value: 23
Thread Name: thread 3 setElementAt(row: 3, column: 8, element: 7)
Thread Name: thread 3 getElementAt(row: 6, column: 5) Return value: 29
Thread Name: thread 3 getElementAt(row: 3, column: 8) Return value: 7
Thread Name: thread 2 setElementAt(row: 5, column: 6, element: 17)
Thread Name: thread 2 getElementAt(row: 5, column: 6) Return value: 17
Thread Name: thread 3 setElementAt(row: 5, column: 6, element: 48)
Thread Name: thread 3 getElementAt(row: 5, column: 6) Return value: 48
Thread Name: thread 4 setElementAt(row: 5, column: 6, element: 47)
Thread Name: thread 4 getElementAt(row: 5, column: 6) Return value: 47

Process finished with exit code 0
```