CSE433 – Object Oriented Analysis and Design – Homework 2

Gökçe Nur Erer - 171044079

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?

Unless Singleton class doesn't implement Cloneable interface, clone method cannot be called. But if Singleton class implements Cloneable interface and yet has overridden the clone method to call its super class' (Object, if there is no other parent class) clone method, then anyone who calls that clone method can create a second distinct Singleton object.

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

To prevent cloning Singletons one solution could be that when implementing the clone method instead of calling the super class' clone method, CloneNotSupporttedException could be thrown. In that case the Singleton class cannot create clones of its own.

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?

For the first question the answer would be still the same. If the clone method is overridden in the parent and not implemented on the Singleton class, then it is still possible to create copies of the Singleton object. For the second question, to prevent singleton cloning, if the clone method is overridden on singleton class to throw a CloneNotSupportedException then it would prevent the Singleton cloning and it would still enable the parent class to clone if it wants to.

Question 2

For this question's solution Iterator pattern is used. Iterator pattern enables us to iterate through an Iterable object without knowing its internal structure. So for this question the following classes and interfaces are created to implement this design pattern:

1) **Iterator:** This interface represents a skeleton for iterators that can exist. Any class who implements this interface must implement hasNext(), next() and getCurrent() methods.

2) **Iterable**: This interface represents a skeleton for all iterable objects that can exist. Any class who implements this interface must implement iterator() method.

3) **SatelliteData:** This class represents the iterable data that is transmitted from a satellite. It wraps up a 2D array which represents the data itself. It implements the iterator() method since this class implements the iterable interface.

4) **SatelliteDataIterator:** This class represents a concrete implementation of a iterator for a SatelliteData instance. Since it implements the iterator interface it implements next(), hasNext() and getCurrent() methods.

next() Method Implementation:

For the implementation of this iterator a basic logic is followed for the clockwise spiraling.

First of all the corners are found for a full turn of a spiral. That is recorded in a list to be used to evaluate how to move.

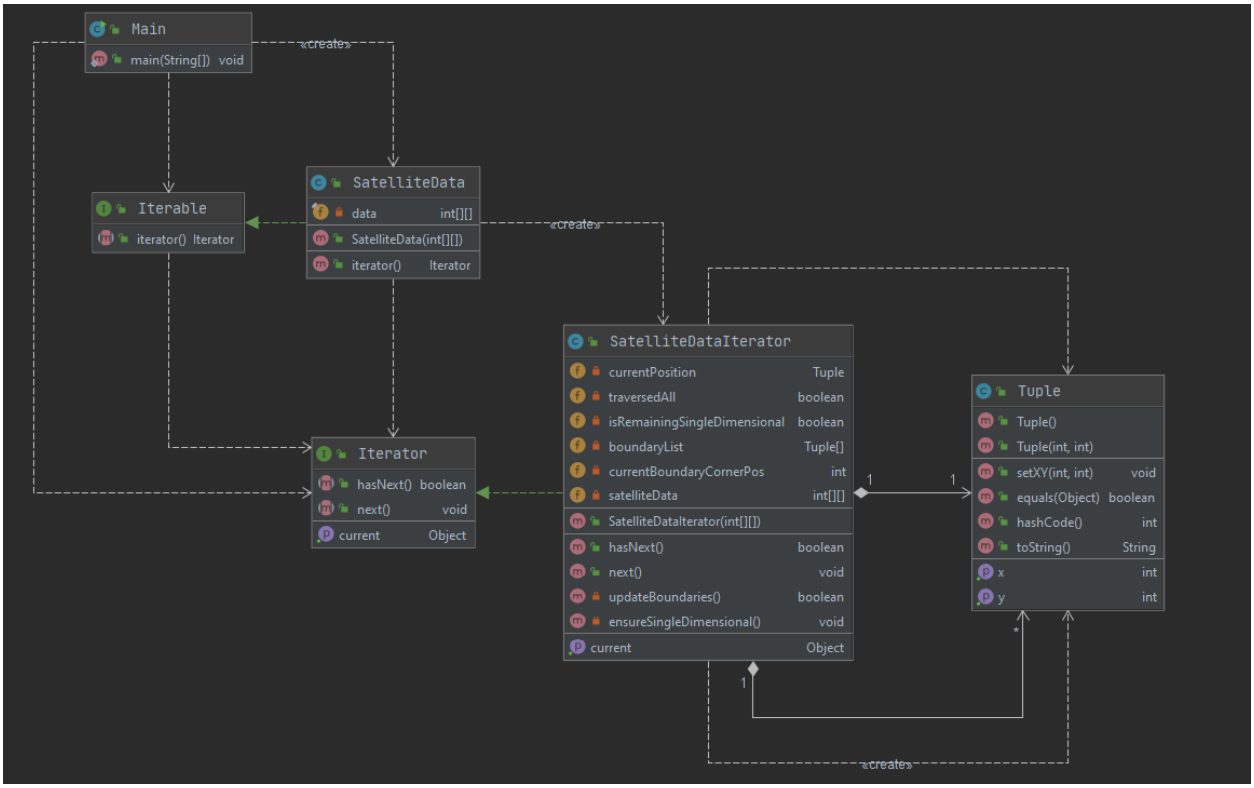
Then these iterating rules are implemented:

- If the last corner that the iterator is coming from is the 0^{th} element of the corner list then the y coordinate of the current position must be increased by one to get the next element.
- If the last corner that the iterator is coming from is the 1^{st} element of the corner list then the x coordinate of the current position must be increased by one to get the next element.
- If the last corner that the iterator is coming from is the 2^{nd} element of the corner list then the y coordinate of the current position must be decreased by one to get the next element.
- If the last corner that the iterator is coming from is the 3^{rd} element of the corner list then the x coordinate of the current position must be decreased by one to get the next element.
- If the current position equals to the start of the corner list then it means a full turn is made so the smaller spiral must be turned. So corner list gets updated.
    - Updating of the corners happens as follows:
        - The x and y coordinates of the first element of the corner list gets increased by one.
        - The x coordinate is increased by one and the y coordinate is decreased by one for the second element in the corner list.
        - The x coordinate is decreased by one and the y coordinate is decreased by one for the third element in the corner list.
        - The x coordinate is decreased by one and the y coordinate is increased by one for the fourth element in the corner list.
        - Then the corner list gets checked if the list became 1D to handle it differently, and corner coordinates are checked if the

coordinates doesn't represent a rectangle/square that has the points starting from left top then right top then right bottom then right top, which means the spiral is done.

hasNext() method uses the info that if all the array is traversed. And this is represented by a Boolean value which is changed on the next() method.

UML Diagram



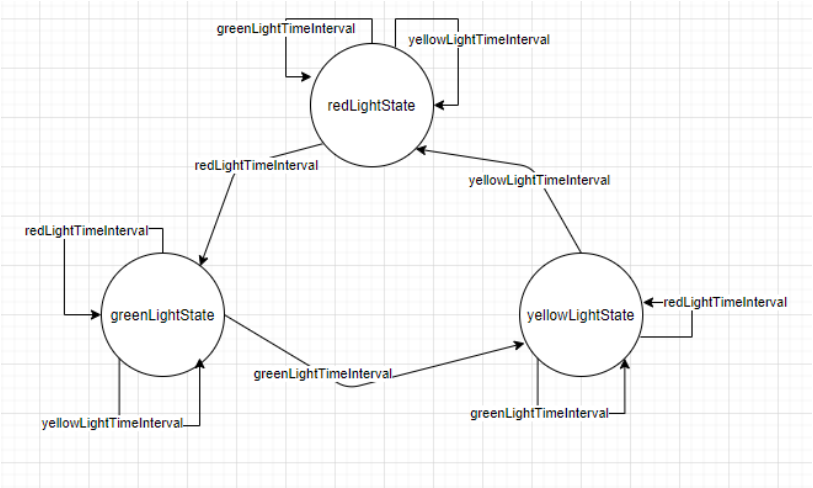
Question 3

a) To solve this question the state pattern is used. State pattern enables us to implement a finite state machine on code. The context doesn't have to worry which state that it will go to but instead the states itself decide on that.

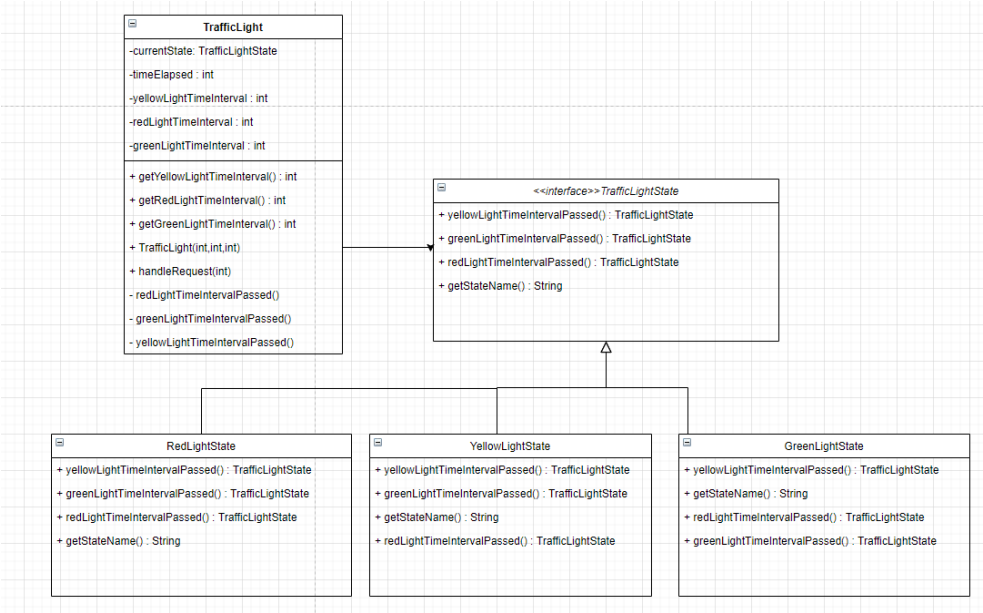
The logic behind the solution is like this:

- First set the initial state as redLightState call handleRequest method periodically. When any kind of timeInterval variable is encountered the method calls the current state's related …LightTimeIntervalPassed() method. And the return value from that method is restored as the new state.

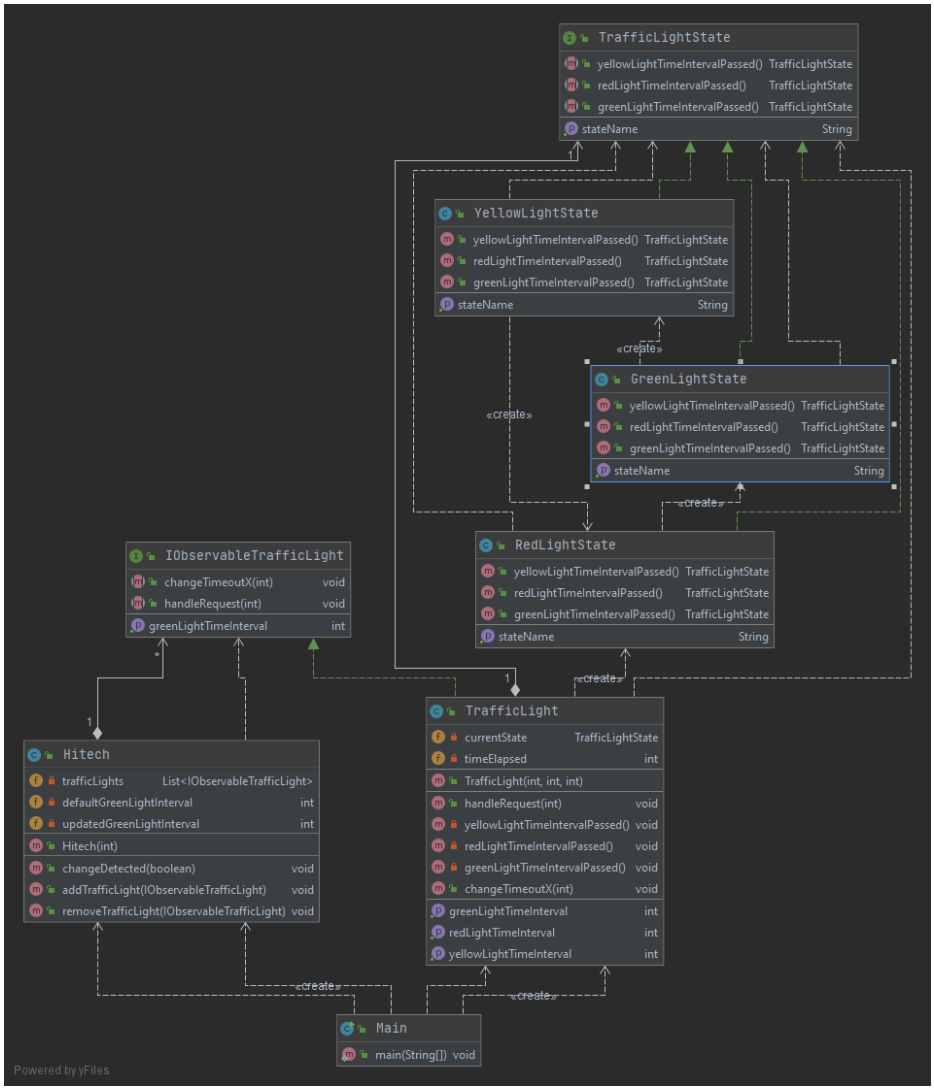
The State Diagram



UML Diagram



b) To implement the Hitech camera with the observer pattern the following strategy is used:
First of all the TrafficLight class got changed that it implements a IObservableTrafficLight interface which has changeTimeoutX(), getGreenLightTimeInterval() and handleRequest() methods.
Then a class called Hitech is added to represent the camera structure. And as it was mentioned in the problem description a method called changeDetected is added. The constructor of this class is assumed that it takes the value of the new updating value of the timeoutX. Also to completely implement the observer pattern addTrafficLight() and removeTrafficLight() methods added as well as a list is created to hold the traffic lights that observe this camera's action. To implement the updating property of the observer pattern the changeDetected method is used. And the method loops between all the observers of the camera and changes their timeoutX (greenLightTimeInterval) to the value that was given in the constructor.



Question 4

a) For the solution for this question the proxy pattern is used. First the ITable interface is created and then DatabaseTable and DatabaseTableProxy classes are created which implement ITable interface. DatabaseTableProxy class holds an instance of the DatabaseTable class so it can call the methods of it to do operations on it.
To implement the synchronization reader thread counts and writer thread counts are calculated when getElementAt and setElementAt are called at the DatabaseTableProxy class. And these methods call the DatabaseTable's related methods only some conditions are met with the reader thread count and writer thread count, else the thread is waited until it gets notified.

b) To implement writer priority to the solution the idea followed was, if a variable called writersWaiting is introduced and checked when reading, the writers would get more priority since the readers are waited until all the writers are done when they execute get and set at the same time.

PS. The UML diagram supplied includes the part b of the question. If the writersWaiting variable is removed it would represent the part a.

UML Diagram