

SE311 – Software Architecture

Smart City Application

Mehmet Aydın KICIRTI -20160602105

Why We chosen below these patterns ?

1. Factory Pattern : According to scenario, in smart city should have 4 type sensor and also these sensors installed everywhere. These Sensors name is Pollution, Temperature, Congestion, Noise . Therefore we think that it should be provided with Factory Pattern and object creation delegation to these sensors. Usage of Factory pattern here will be provided to create different sensors without showing their inner parts. In order to without coupling the client code to concrete classes.
2. Command Pattern : According to the scenario, if there is a malfunctioning sensor in city, the engineer sends a request to reset these sensors. Therefore the Command pattern provided to us if there are a request and response relationship in these between subsystems, think of the best solution for us. In order to directly call reset sensors will increase the dependency, instead of realized to implementation of this pattern and also this pattern encapsulate and forward the reset request to the Engineer which performs as an Invoker in there.

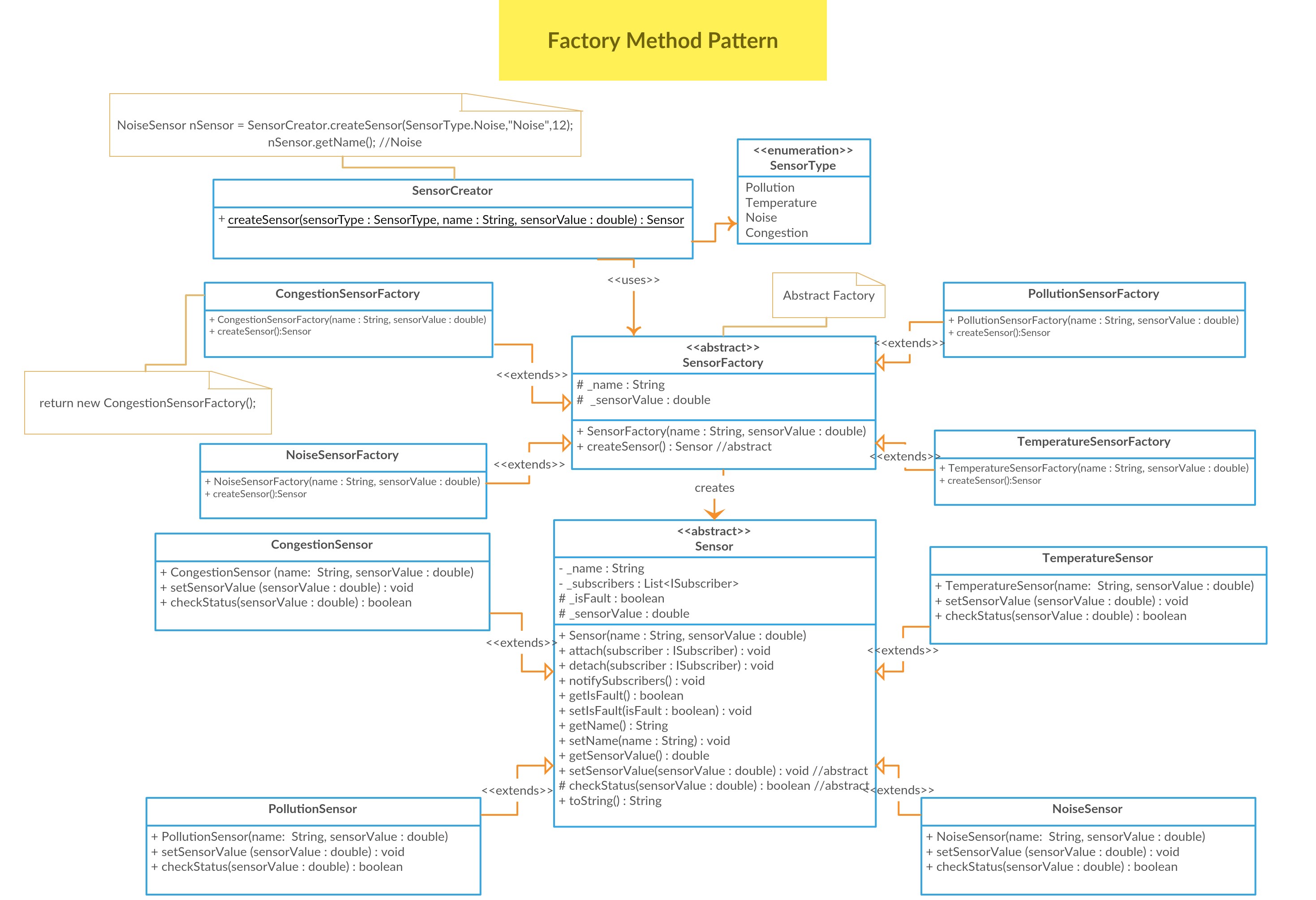
1. Iterator Pattern : According to scenario, you said that “the city is a collection of neighborhoods”,

So we think that the best solution should be implement of iterator pattern and also we can easily get around without any dependency its underlying representation.

1. Composite Pattern : According to the scenario, in our neighborhoods collection includes streets. The street contains apartments and poles. In here, there is an hierarchy so that the best solution to implement of Composite pattern therefore provided that allows clients to process individual objects and compositions uniformly.
2. Singleton Pattern : Scenario says that should be only one Data Monitoring Division in the city. The usage of the singleton would provide only one instance of monitör for object creation. In our case, it creates CityWideMonitor class.
3. Observer Pattern : In the scenario, If it provides the values ​​on the sensors, we should send an information message . In order to solve this situation. Observer pattern can be able to the best choice since we will attach who subscribe to the sensors as an notification.

2.Participant of Each Pattern

- 2.1) Factory Pattern =>



Participants =>

Abstract Factory => SensorFactory

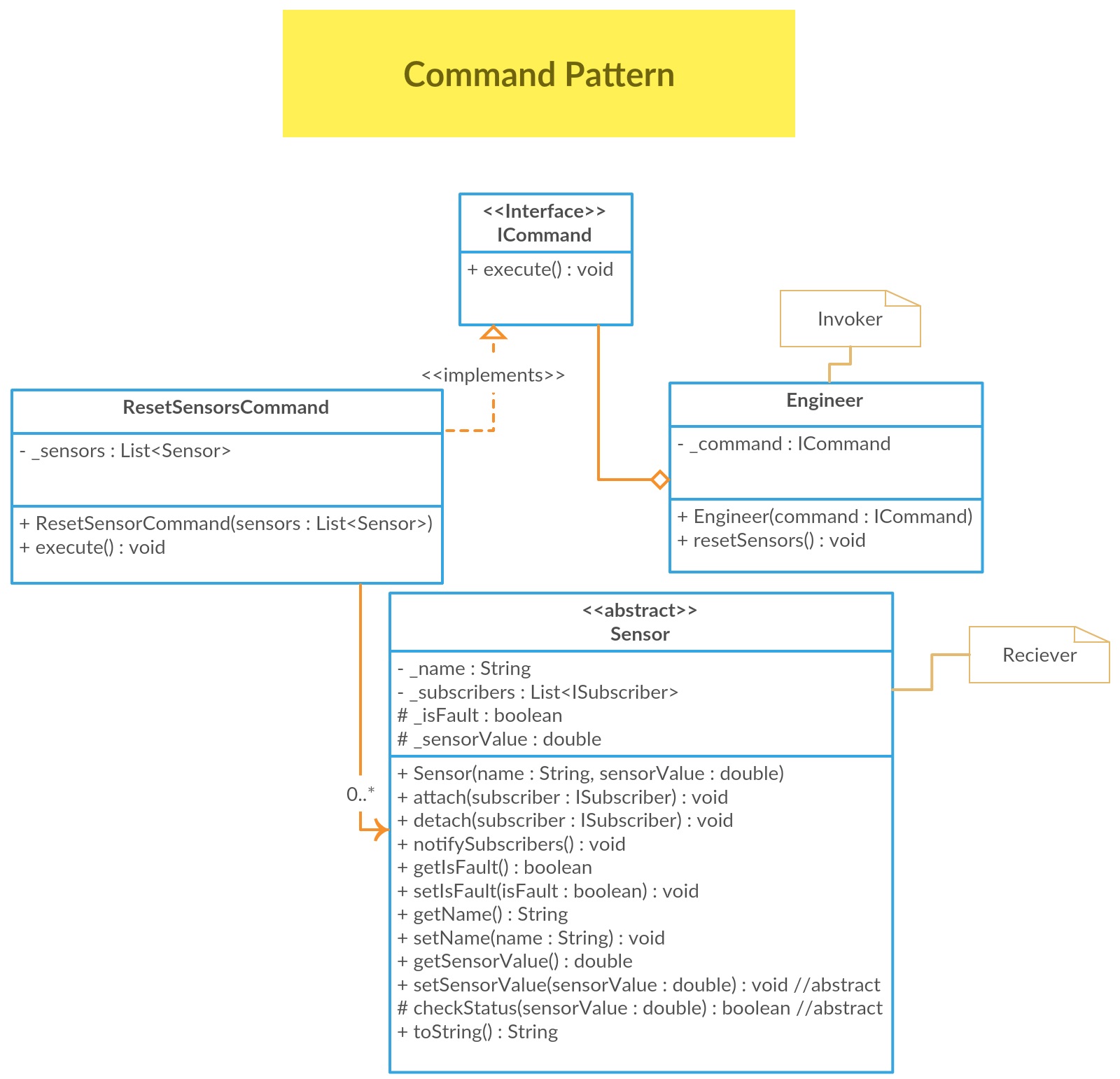
Concrete Factory => CongestionSensorFactory, PollutionSensorFactory,NoiseSensorFactory, TemperatureSensorFactory

AbstractProduct => Sensor

ConcreteProduct => NoiseSensor, CongestionSensor, TemperatureSensor, PollutionSensor

Client => SensorCreator

* 2.2) Command Pattern =>



Participants

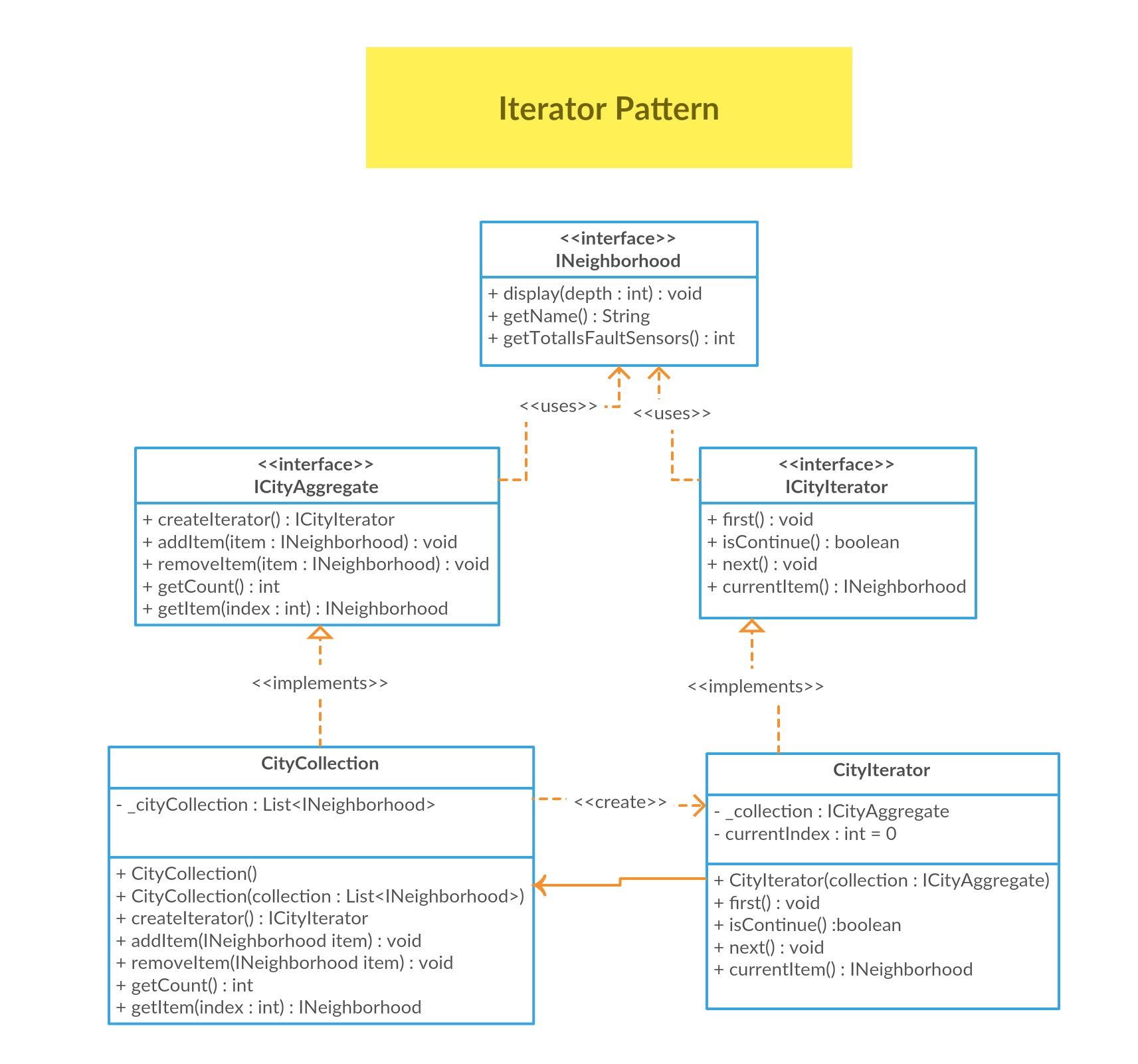
Command => ICommand

ConcreteCommand => ResetSensorsCommand

Invoker => Engineer

Reciever => Sensor

- 2.3) Iterator Pattern



Participants

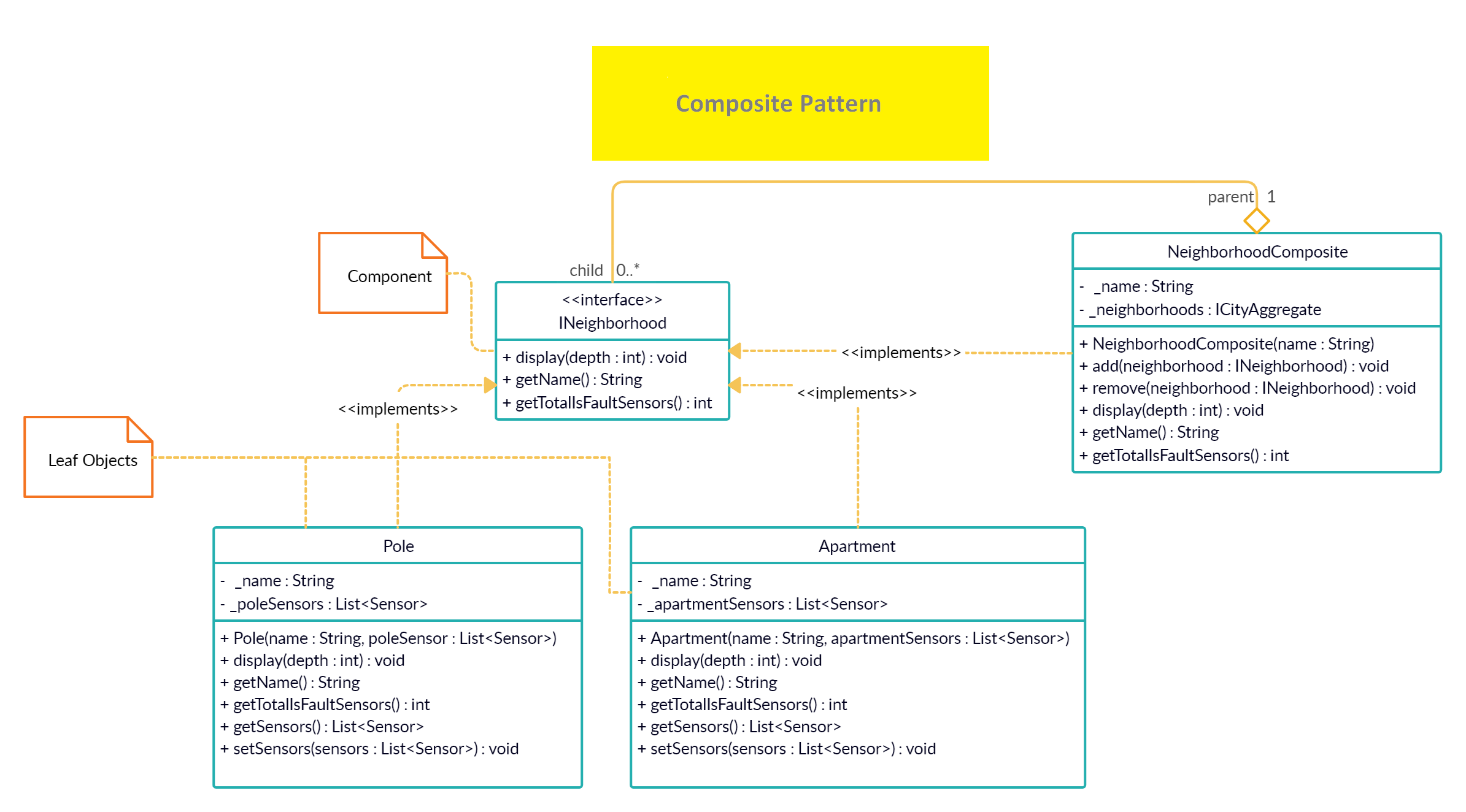
Aggregate => ICityAggregate

Iterator => ICityIterator

ConcereteAggregate => CityCollection

ConcreteIterator=>CityIterator

- 2.4) Composite Pattern



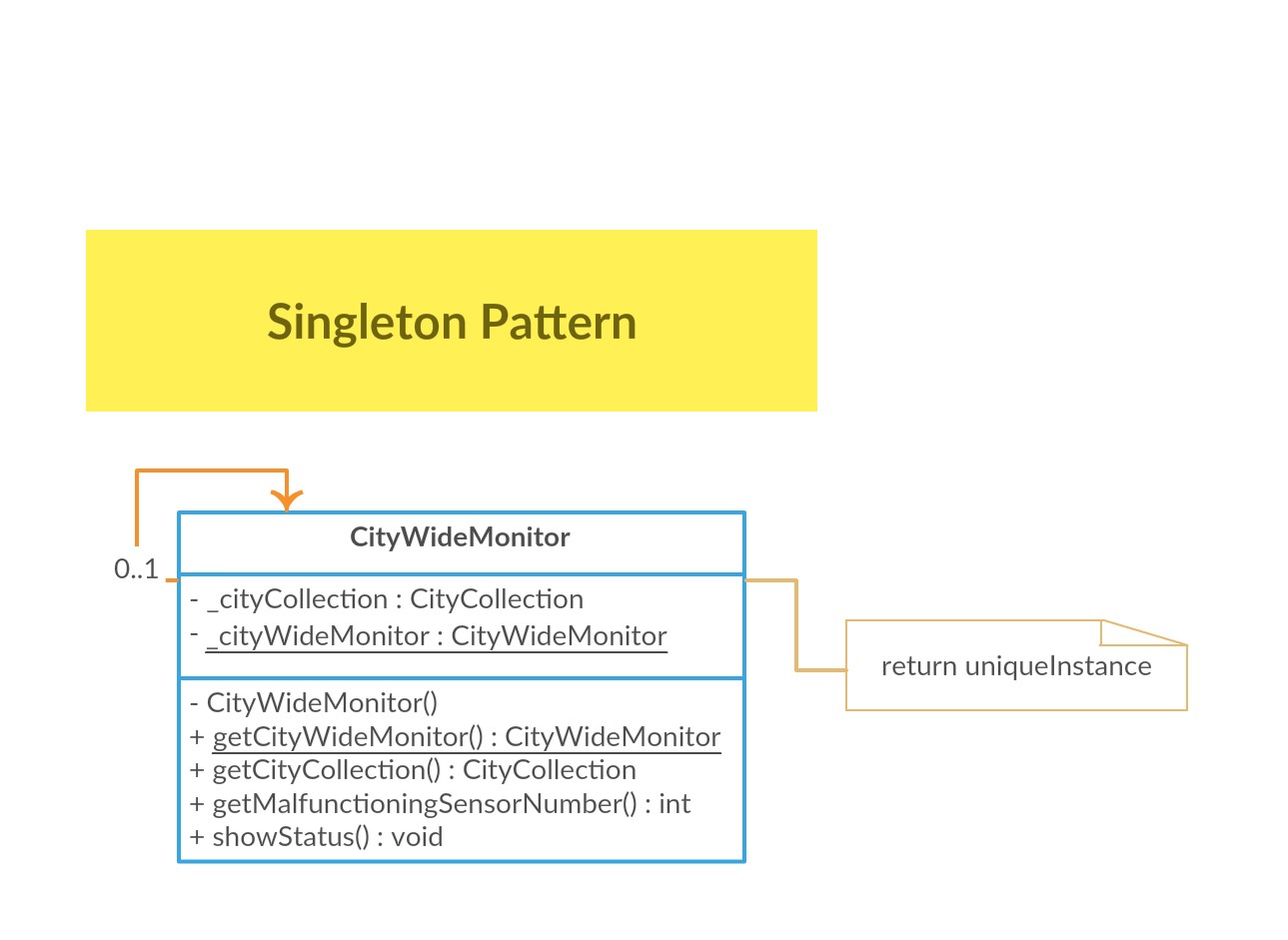
-- Participants

Component => INeighborhood

Composite => NeighborhoodComposite

Leaf => Pole, Apartment

- 2.5) Singleton Pattern



* Participants

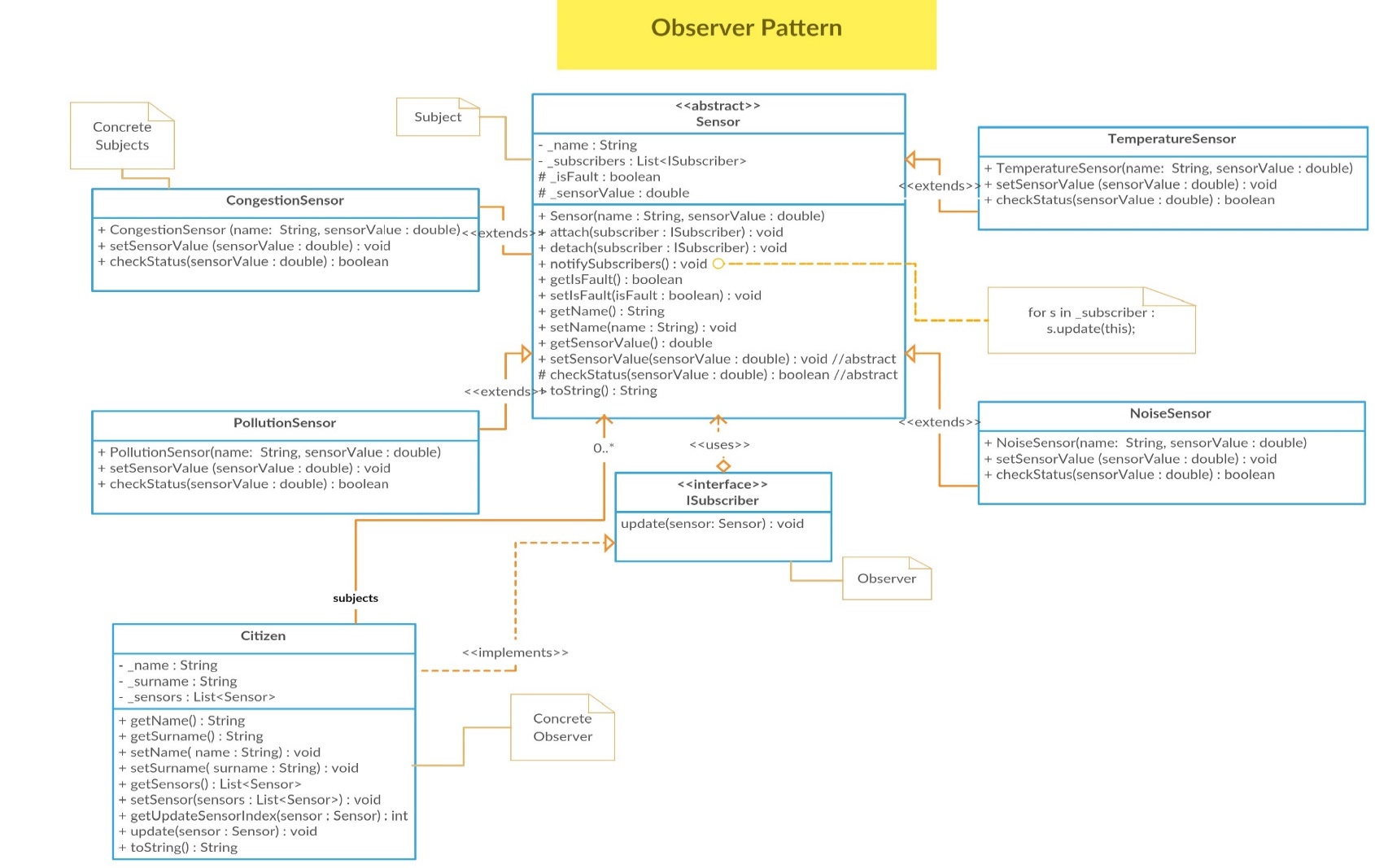
Singleton : CityWideMonitor

Singleton class : CityWideMonitor

If we want to construct a singleton class we need to make first constructor private or protected and also class name instance should be static and again private or protected so that when we want to create instance provide by getCityWideMonitor().This function returns either before we did not create firstly class instantiation or directly returns class instatiation.

And also we can use cityCollection when we were creating a neighorhood with this instance provide addNeighbor into collection.Therefore we can get easily status of city with showStatus() function by the way getMalfunctioningSensorNumber() gets us number of malfunctioning sensor into system. We used with showStatus() function.

- 2.6) Observer Pattern



Participants =>

Subject : Sensor

Observer : ISubscriber

Concerete Subject : CongestionSensor, TemperatureSensor, PollutionSensor, NoiseSensor

Concerete Observer : Citizen

Subject: Sensor

* This is an abstract class for the subject. Sensor class provides the following :
* Attaching and detaching observers to Concrete Subjects. In order to attached an observer there is any state change on sensor value . Citizen knows.
* Notify function is notifySubscribers() to inform status of sensor value changing.
* We use setSensorValue() function. After changed of sensorvalue the belonging to sensors firstly checking status with checkStatus() function if this provide this condition, will notified subscribers.

Observer: ISubscriber

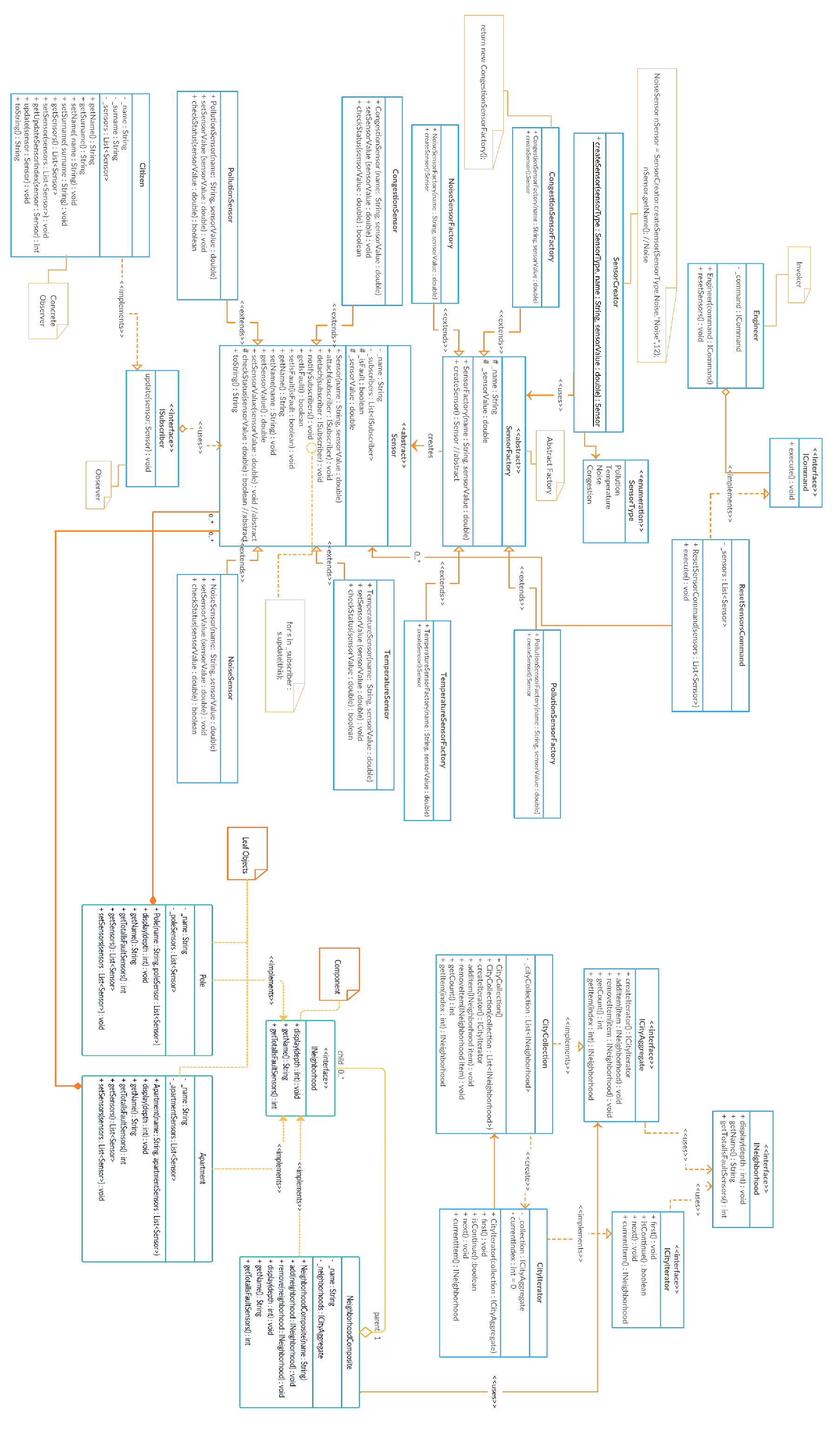
* This is the interface for Observer. As you can see from the name of it. ISubscriber object observes subscribed the sensors. This interface has an important function following in it;
* Update(sensor : Sensor) : Function will work after changing status of the sensorValue if condition is provided. It will update. Observer object is taking current state to the incoming state.

Concrete Subject : CongestionSensor, PollutionSensor, NoiseSensor, TemperatureSensor

* Derived object from Sensor. These sensors has same operations of with it. Will be used as a condition results provided to notify in our scenario.

Concrete Observer : Citizen

* Derived object from ISubscriber. It has also same member of functions and it will work as a Concrete Observer.



Conclusion

In the Smart City Application, we tried to implement 6 patterns. Before drawing the UML, We tried to think about the best choice patterns from the scenario. After that, we started to draw the UML, but this part was so hardest part for us. Especially, especially when we were trying to connect classes in UML. We used Creately to draw the class diagram, then we were adapting to own structures on application. As a result, we aimed to understand the best solution to the use of design patterns.