Marcos Seco Anderson & Diego Dopazo Garcia

E2-REPORT: Management of alerts on the parameters of the tanks of an aquarium

In this exercise we created a system to manage the alerts of an aquarium. To do this first there is a class aquarium, this class stores the tanks, the control devices and the personnel in three lists.

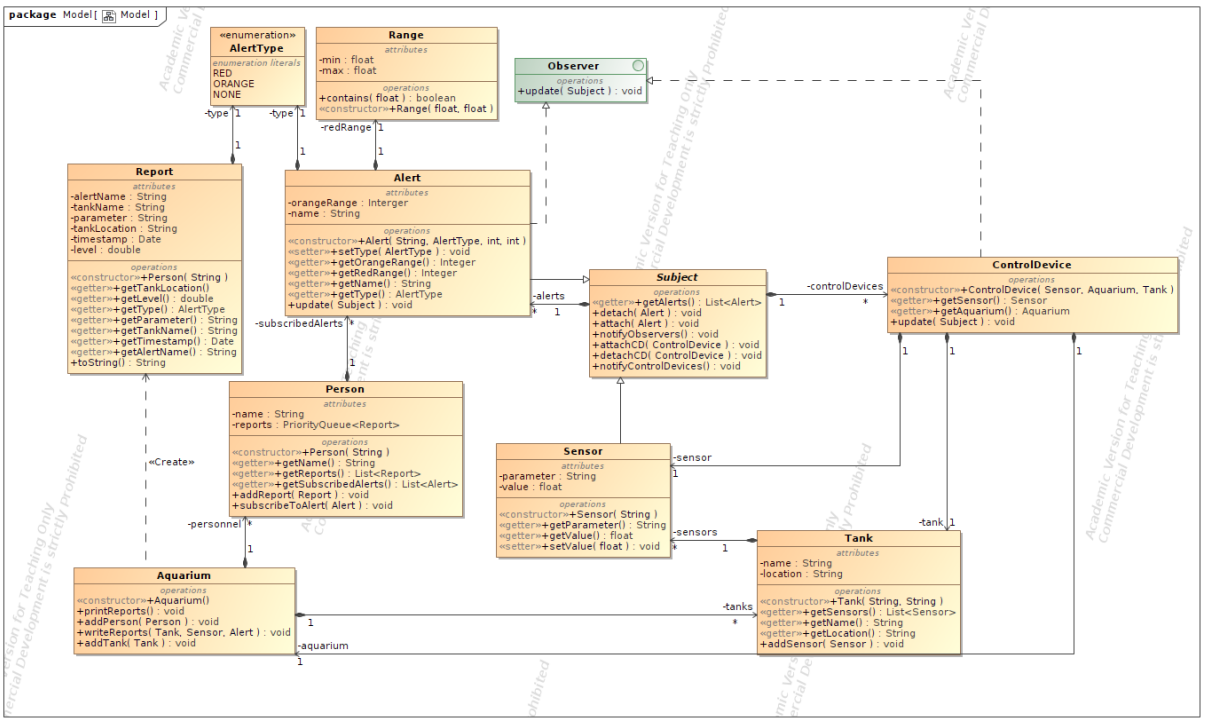
Each tank contains several sensors that are stored in a list of sensors, and each sensor measures a single numeric parameter periodically. To change the value of a parameter there is a method called setValue() that changes the value. We decided to implement the **observer pattern.** This pattern allows objects to be notified when a change occurs in another object that it is observing.

In this code, the Alert class extends the Subject class and implements the Observer interface, which makes it both a subject and an observer. The ControlDevice class also implements the Observer interface and observes the Alert class.

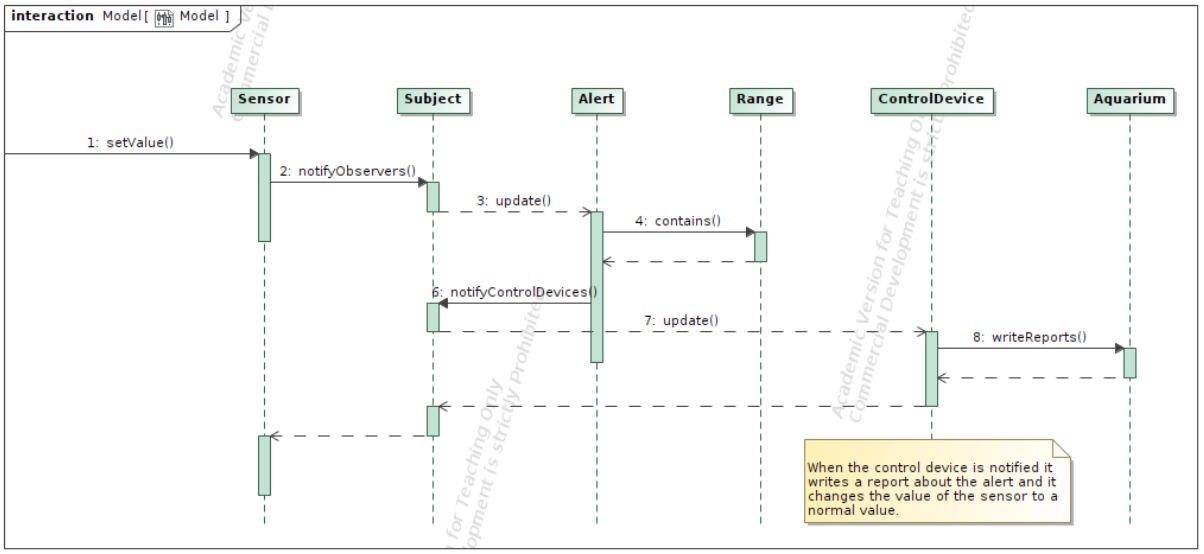
The Alert class has several instance variables, including a name, a redRange, an orangeRange, and an AlertType. It also has methods to access these instance variables, as well as a constructor to initialize them. The update method is implemented from the Observer interface, and it checks the value of a sensor and sets the alert type based on whether the value falls within the red or orange range. Once the alert is fired, the control devices of the alert are notified, a report of the alert is created to the subscribed personnel and the control devices fix the issue by establishing the value of the sensor to a value that does not fire an alert (in our case it sets a random value inside of the normal range).

Finally to print the reports a function printReports() goes through all the personnel and prints each person's report queue.

Here is the Class Diagram of the exercise:



The most important part of the system is how the alerts and the control devices are notified when there are changes, this is all part of a chain reaction that is started when a new value is set to a sensor. The method is called setValue(), and a sequence diagram can help us show how it works:



As we can see when the value of a sensor changes the alerts check if a RED or ORANGE alert should be fired if it should, then it notifies the control devices, where a report is written about the issue and the value is changed back to a value inside of the normal range.

SOLID PRINCIPLES

Single Responsibility Principle (SRP): The Alert class has a single responsibility, which is to represent an alert that can be triggered by a sensor. Similarly, the ControlDevice class has a single responsibility, which is to respond to an alert by generating a report and resetting the value of the sensor.

Open/Closed Principle (OCP): The Alert class is open for extension, in that it can be subclassed and extended to add new behavior, but it is closed for modification, in that its existing behavior cannot be changed without affecting the rest of the system.

Liskov Substitution Principle (LSP): The ControlDevice class correctly implements the Observer interface, which means it can be used as a substitute for any other class that implements the Observer interface.

Interface Segregation Principle (ISP): The Observer interface defines a single method, update(), which means that classes that implement it are only required to implement this single method, rather than a large set of methods that they may not need. This helps to avoid forcing classes to depend on methods that they do not use.

Dependency Inversion Principle (DIP): The ControlDevice class depends on the Observer interface, rather than on a specific concrete implementation of an Observer. This means that the ControlDevice class is not tightly coupled to any specific implementation of an Observer, and can be more easily reused and modified.