Portfolio Assignment

Exam portfolio for the course Software Technology in Cyber-Physical Systems

Mathias Nickolaj Rasmussen

Exam number: 167341815
Software Technology
4th Semester

A portfolio presented for the exam in
Software Technology in Cyber-Physical Systems

The Faculty of Engineering
University of Southern Denmark
Denmark

16/05-2021

Contents

1 Portfolio Assignment - Part 1

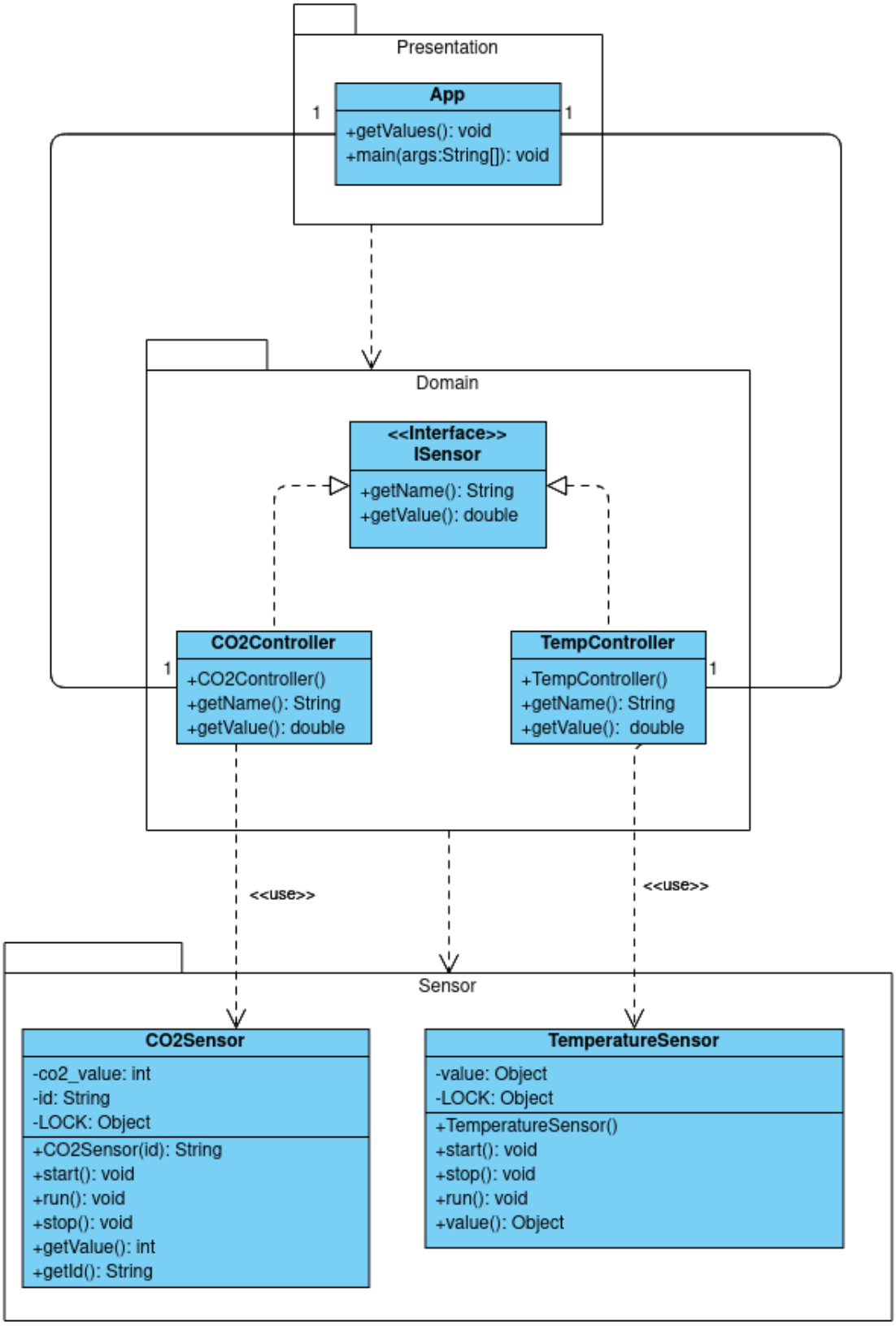

Figure 1: UML class diagram - TestApplication

2 Portfolio Assignment - Part 2

2.1 Application component

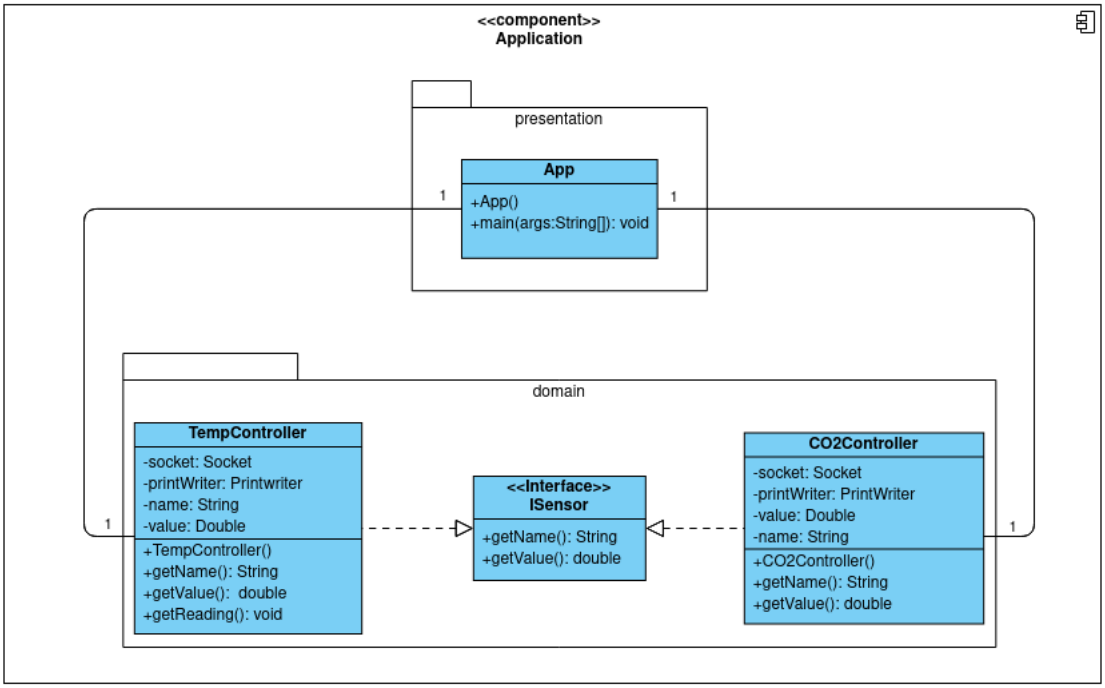

Figure 2: Class Diagram with packages - Application

2.2 Temperature Sensor component

Please see 5

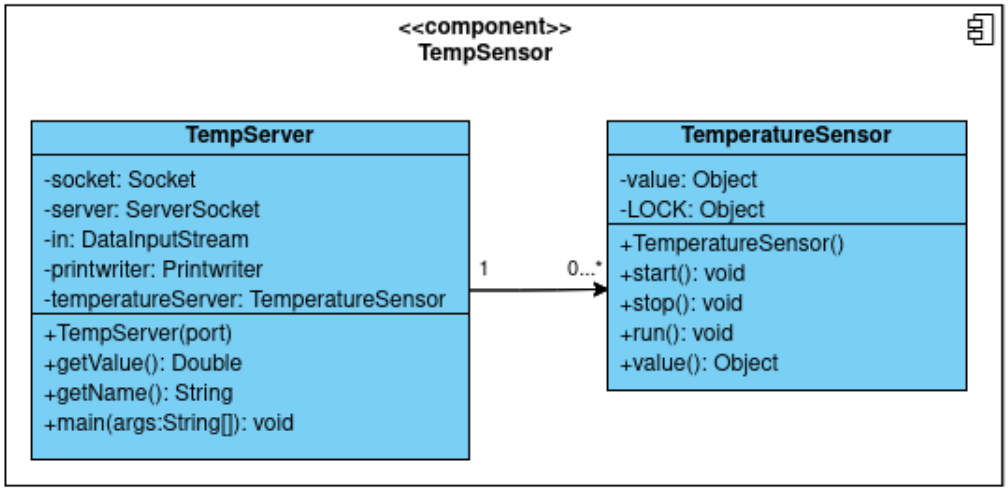

Figure 3: Class Diagram without packages - CO2Sensor

2.3 CO2 Sensor component

Please see 5

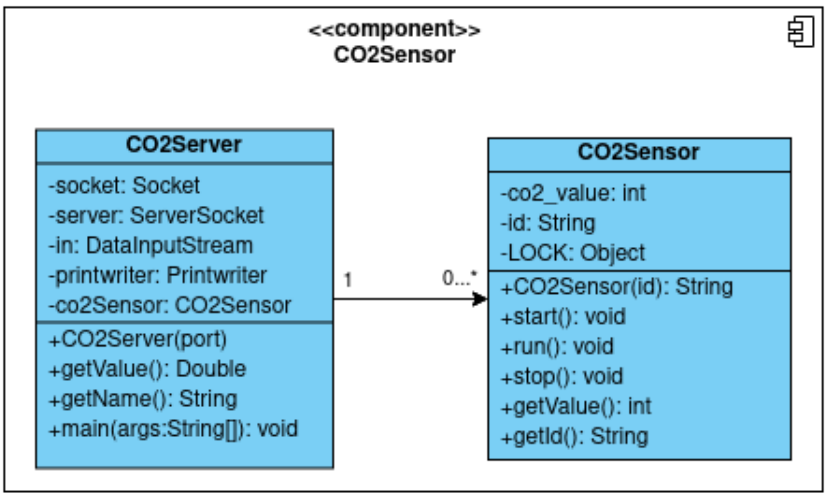

Figure 4: Class Diagram without packages - CO2Sensor

2.4 Socket implementation

Throughout this section **CO2Sensor** will be used as an example to show the server-client implementation. The method used in this assignment is UTF based Socket programming.

2.4.1 Server

The server component implementation for the **CO2Sensor** is shown in Listing 1. Here the constructor for the class **CO2Sensor** is displayed. The constructor takes in an integer as an argument which sets the Socket port. On line 5 a new *ServerSocket* object is instantiated and takes the port, assigned when the constructor is instantiated, as an argument.
The *ServerSocket* is then accepted to establish connection on line 9.
On line 12 to 14 the *PrintWriter* object is instantiated and takes the output stream of the socket as an argument. The *PrintWriter* then prints the name and value from the *getName()* and *getValue()* methods that returns a String name and a Double value generated in the *CO2Sensor* class. The *PrintWriter* is then flushed to clear the output stream.
Then a *DataInputStream* is instantiated with a new *BufferedInputStream* object as argument, that takes the sockets input stream.
From line 17 to 22 a while loop is created. This ensures that if a *ServerSocket* connection is already established on the assigned port, the server will accept, instantiate an *PrintWriter* and have it print the new name and value without establishing a new *ServerSocket* connection.
Lastly the *ServerSocket*, *DataInputStream* and *PrintWriter* is closed.

The constructor is instantiated in a main method in the same class as Listing 1 and are then given port 5001 as argument. This opens a *ServerSocket* connection on URL *http://localhost:5001* which is the same as *http://127.0.0.1:5001*. It is then possible for the subscribed client to reach the *ServerSocket* output on this URL.

```
1     public CO2Server(int port) {
2
3         // starts server and waits for a connection
4         try {
5             server = new ServerSocket(port);
6             System.out.println("Server started");
7             System.out.println("Waiting for a client ...");
8
9             socket = server.accept();
10            System.out.println("Client accepted");
11
12            printWriter = new PrintWriter(socket.getOutputStream());
13            printWriter.println(getName() + "@" + getValue());
14            printWriter.flush();
```

```
15              in = new DataInputStream(new BufferedInputStream(socket.getInputStream())
                    );
16
17              while (socket != null) {
18                  socket = server.accept();
19                  in = new DataInputStream(socket.getInputStream());
20                  printWriter = new PrintWriter(socket.getOutputStream());
21                  printWriter.println(getName() + "@" + getValue());
22                  printWriter.flush();
23              }
24
25              // close connection
26              socket.close();
27              in.close();
28              printWriter.close();
29
30          } catch (IOException i) {
31              System.out.println(i);
32          }
33      }
```

Listing 1: **CO2 WebSocket Application**

2.4.2 Client

In Listing 2 the constructor *CO2Controller()* is shown. *CO2Controller()* implements the request pattern and handles the reading of the Socket.
On line 5 a *Socket* is instantiated with the parameters set to localhost and the port 5001. This means a Socket connection is established at the URL *http://localhost:5001*, which is the same as *http://127.0.0.1:5001*. As mentioned in subsection 2.4.1 this is where the *ServerSocket* is replying requests sent from the *Socket*.
On line 9 a new *PrintWriter* is instantiated and sets the output stream of the socket as an argument.
Next a *InputStreamReader* object is instantiated on line 13 and takes the input stream of the *Socket* as an argument.
Then a *BufferedReader* is instantiated and takes the *InputStreamReader* object as an argument.
From line 16-20 a *String* is created and sat to return the input stream of the *BufferedReader*. The *String* is then added to an array and splitted by the delimiter @. The name and value is retrieved from the *getName()* and *getValue()* methods that are overriden from the interface *ISensor*.
Lastly the *Socket* and *PrintWriter* is closed.

```
1       public CO2Controller() {
2
3           // establish a connection
4           try {
5               socket = new Socket("localhost", 5001);
6               System.out.println("Connected to CO2 server");
7
8               // sends output to the socket
9               printWriter = new PrintWriter(socket.getOutputStream());
10              printWriter.write("hello");
11
12              InputStreamReader inputStreamReader = new InputStreamReader(socket.
                    getInputStream());
13
14              BufferedReader bufferedReader = new BufferedReader(inputStreamReader);
15              String resp = bufferedReader.readLine();
16
17              String[] arr = resp.split("@");
18              name = arr[0];
19              value = Double.valueOf(arr[1]);
20
21              // close the connection
22              socket.close();
23              printWriter.close();
24
25          } catch (IOException i) {
26              System.out.println(i);
```

```
        }
    }
```

Listing 2: **CO2 WebSocket request client**

In Listing 3 the **App** class is shown. The application contains a constructor *App()* that uses the interface *ISensor* to call *getName()* and *getValue()* methods from the controller classes.
The controllers (*TempController, CO2Controller*) are instantiated as an *ISensor* object on line 9 and 10.
The *App()* constructor is then instantiated in the **App** class main method.

```
1  public class App {
2
3      public App() {
4
5          Scanner sc = new Scanner(System.in);
6
7          while (sc.hasNextLine()) {
8
9              int nInt = sc.nextInt();
10             ISensor t;
11             ISensor c;
12
13             switch (nInt) {
14                 // Temperature and CO2 name and value
15                 case (1) -> {
16                     t = new TempController();
17                     c = new CO2Controller();
18                     System.out.println(t.getName() + ": " + t.getValue() + "\n" + c.
                           getName() + ": " + c.getValue());
19                 }
20                 //Temperature name
21                 case (2) -> {
22                     t = new TempController();
23                     System.out.println(t.getName());
24                 }
25                 // Temperature value
26                 case (3) -> {
27                     t = new TempController();
28                     System.out.println(t.getValue());
29                 }
30                 // Temperature name and value
31                 case (4) -> {
32                     t = new TempController();
33                     System.out.println(t.getName() + ": " + t.getValue());
34                 }
35                 // CO2 name
36                 case (5) -> {
37                     c = new CO2Controller();
38                     System.out.println(c.getName());
39                 }
40                 // CO2 value
41                 case (6) -> {
42                     c = new CO2Controller();
43                     System.out.println(c.getValue());
44                 }
45                 // CO2 name and value
46                 case (7) -> {
47                     c = new CO2Controller();
48                     System.out.println(c.getName() + ": " + c.getValue());
49                 }
50                 // Wrong int
51                 case (8), (9) -> {
52                     System.out.println("Sorry that command does not exist");
53                 }
54                 // Exits the program
55                 case (0) -> {
56                     System.out.println("\n" + "Thank you for using this amazing app!
                           See you soon :-)");
```

```
                    sc.close();
                    System.exit(0);
                }
            }
        }
    }

    public static void main(String[] args) {
        System.out.println("""
                Welcome to the application!
                Type 1 for Temperature and CO2 name value
                Type 2 for Temperature name
                Type 3 for Temperature value
                Type 4 for Temperature name and value
                Type 5 for CO2 name
                Type 6 for CO2 value
                Type 7 for CO2 name and value
                Type 0 to exit the program
                """);
        App app = new App();
    }
}
```

Listing 3: **CO2 WebSocket Application**

3 Portfolio Assignment - Part 3

3.1 CO2 publisher

Please see 5

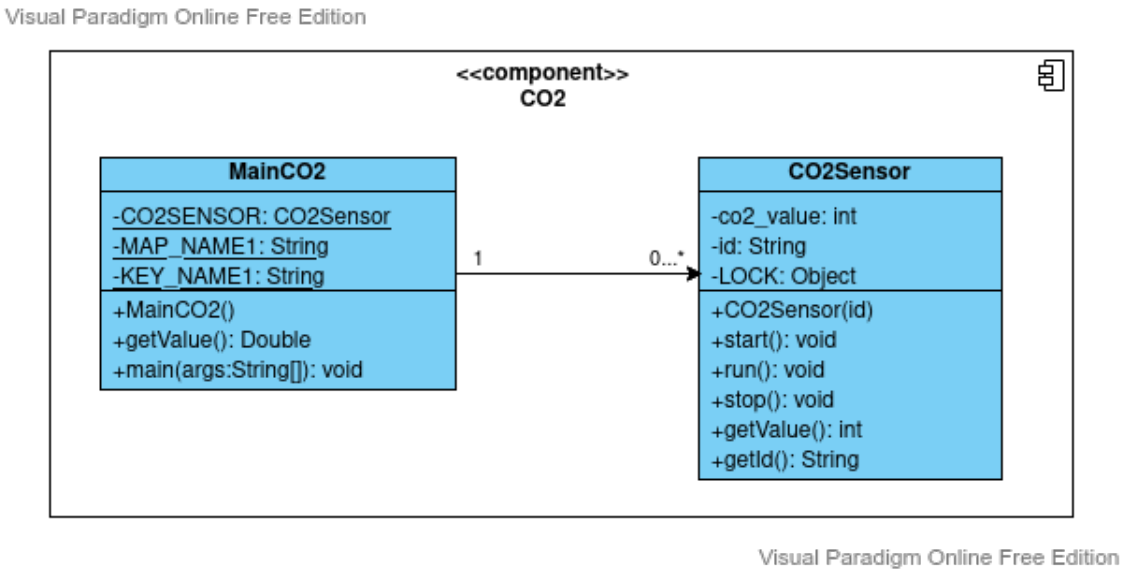

Figure 5: Class diagram - CO2 publisher service

3.2 Temperature publisher

Please see 5

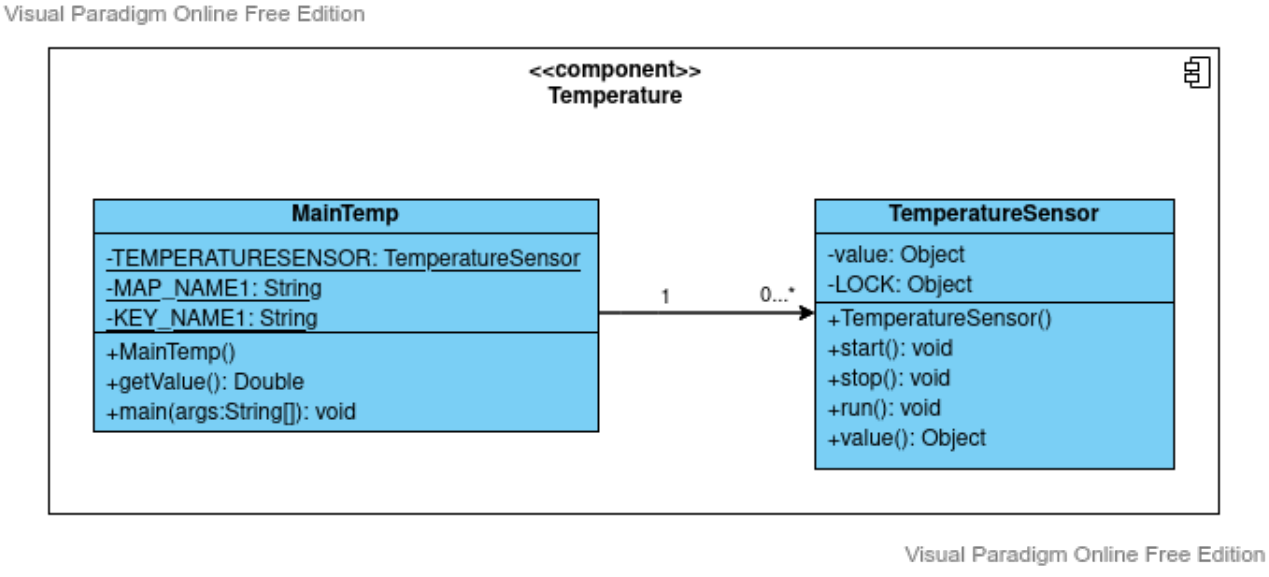

Figure 6: Class diagram - Temperature publisher service

3.3 Client observer

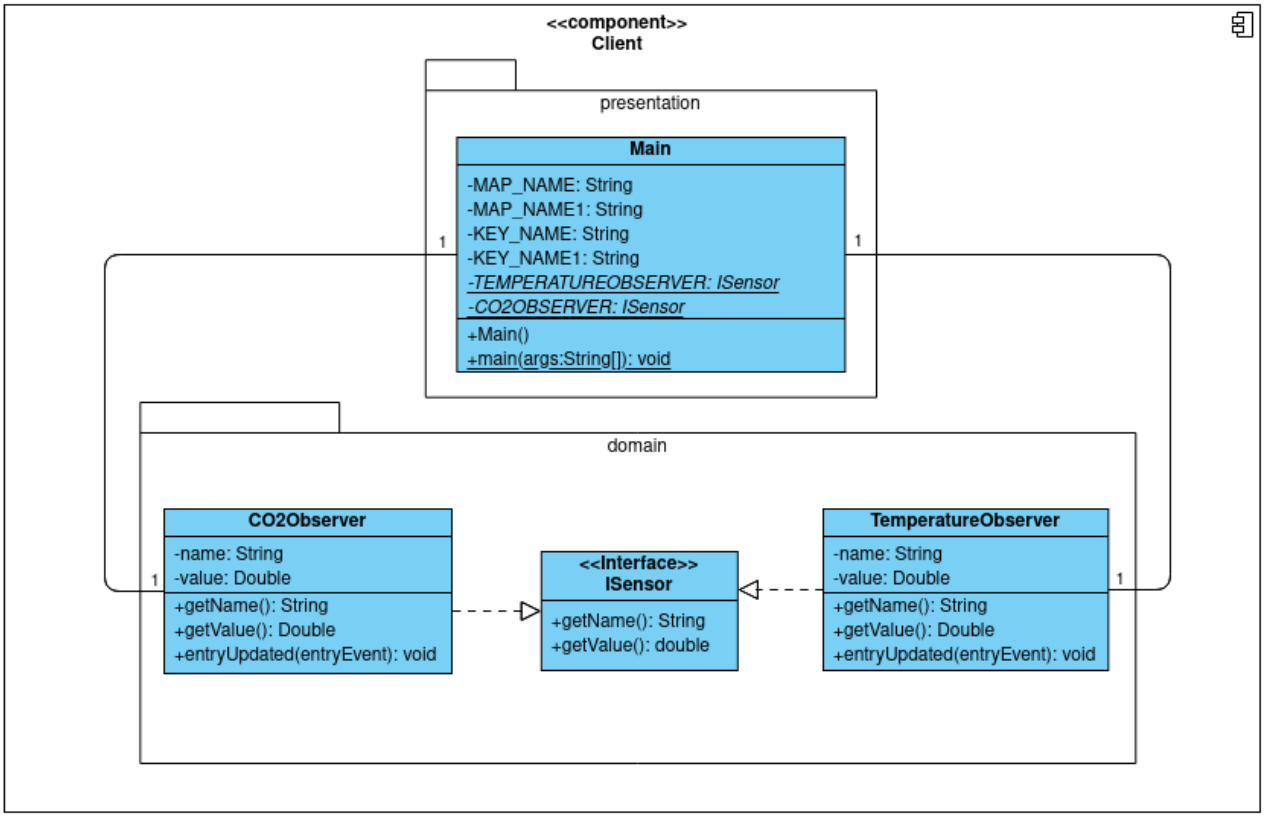

Figure 7: Class diagram - Client observer service

4 Portfolio Assignment - Part 4

4.1 Temperature publisher service

Please see 5

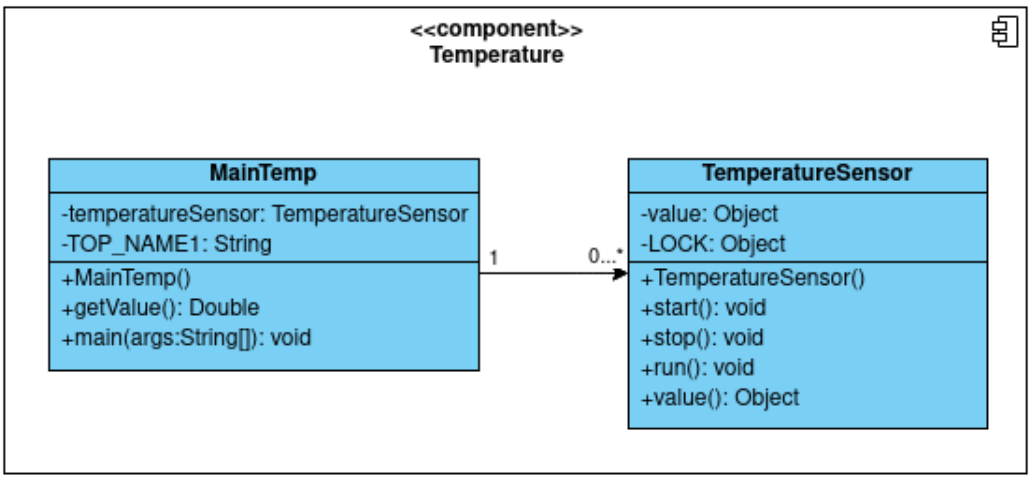

Figure 8: Class diagram - CO2 publisher service

4.2 CO2 publisher service

Please see 5

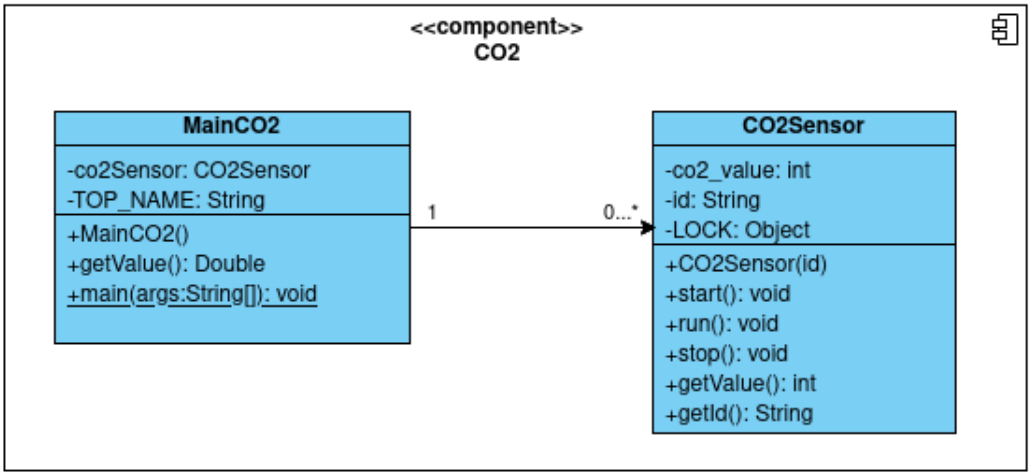

Figure 9: Class diagram - CO2 publisher service

4.3 Client subscriber service

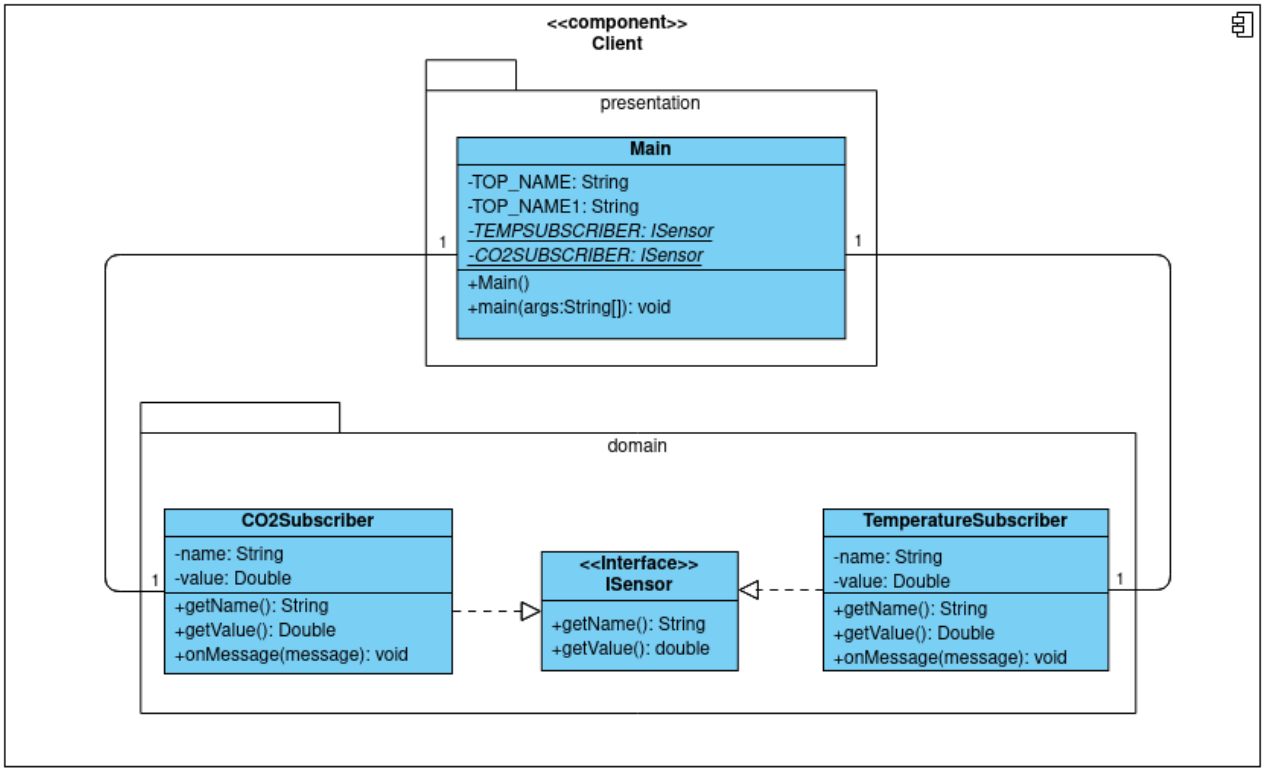

Figure 10: Class diagram - Client subscriber service

5 Arrow Error

The arrow in this Figure is supposed to be an open arrow. Visual Paradigm that was used for modelling, due to the usage in another course, does not seem to have that option.