Software Design Patterns

Practice 2: Design and Responsibility Assignment Patterns (evaluated out of 50 points)

Exercise 1 (15 points)

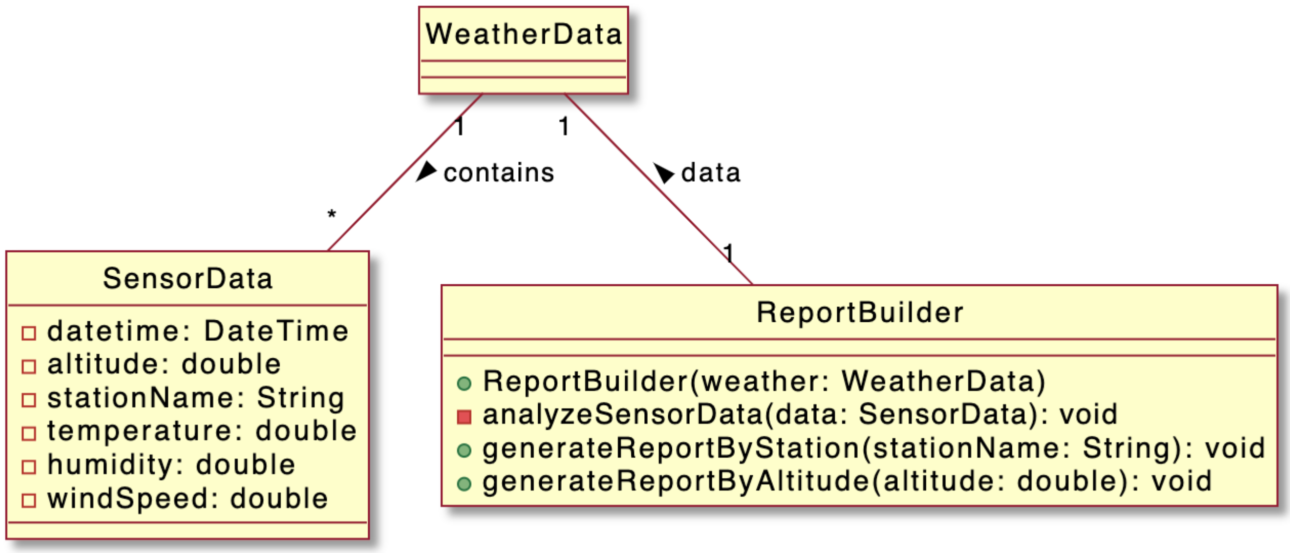
A weather monitoring system collects data from various sensors located in different geographical locations. Each sensor records data about temperature, humidity, and wind speed. The system stores the sensor data in a central repository where it can be accessed for analysis and reporting.

The current design of the system has a `WeatherData` class that holds a list of sensor data. Each sensor data entry contains information like altitude, station name, temperature, humidity, and wind speed. Analysts often need to aggregate these data points to generate reports or perform analysis.

The `ReportBuilder` class is responsible for generating reports based on the sensor data. It can generate a report by selecting the sensor data by station name, or it can make a report by selecting the sensor data where the altitude is higher than the given one. For generating the report the `analyzeSensorData` method should be called with all the required sensor data.

As usual, we should have the minimum coupling between `ReportBuilder`, `WeatherData` and `SensorData`. The internal details of `WeatherData` and `SensorData` should not be exposed to `ReportBuilder` while we want to proportionate different ways to traverse the sensor data.

Class Diagram


WeatherData
1
1
contains
data
*
1
SensorData
datetime: DateTime
altitude: double
stationName: String
temperature: double
humidity: double
windSpeed: double
ReportBuilder
ReportBuilder(weather: WeatherData)
analyzeSensorData(data: SensorData): void
generateReportByStation(stationName: String): void
generateReportByAltitude(altitude: double): void

Integrity Constraints

- SensorData key is datetime and stationName.

Questions

1. (3 points) Justify which design pattern should be applied to generate the reports by station name and also by altitude.
2. (5 points) Show the class diagram with the operations needed to implement this design to be able to generate reports by station name and also by altitude by traversing the available SensorData depending on the type of report.
3. (7 points) Show the complete pseudocode of the classes starting with the operations `ReportBuilder::generateReportByStation` and `ReportBuilder::generateReportByAltitude`.

Exercise 2 (10 points)

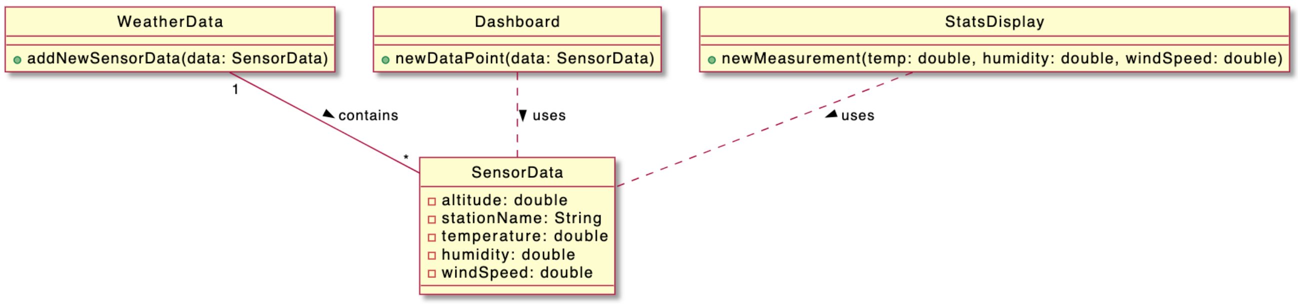
Two new components are added to the weather monitoring system: a dashboard and a statistical display. The dashboard is a graphical user interface that displays the latest weather data from the weather stations. On the other side, the statistical display is a component that analyzes the weather data and displays the minimum, maximum, and average values for each measurement.

You can assume that the `Dashboard` and `StatsDisplay` are already implemented and that they have methods called `newDataPoint` and `newMeasurement`, respectively, that must be called whenever new data is available, that is, when `addNewSensorData` operation from `WeatherData` is called.

As usual, we want to add these two new components with the minimum coupling to the existing system.

Class Diagram

The relationships and initial methods are as follows:



Questions

1. (3 points) Justify which design pattern should be applied to integrate these new system components `Dashboard` and `StatsDisplay`.
2. (3 points) Show the class diagram with the operations needed to complete the design.
3. (4 points) Show the complete pseudocode of the classes with operations or the sequence diagrams to see how `Dashboard.newDataPoint(data: SensorData)` and `StatsDisplay.newMeasurement(data:SensorData)` are called.

Exercise 3 (10 points)

In the statistical display `StatsDisplay` of our weather monitoring system, the scientists have noticed that in some cases they want the absolute mean, maximum and minimum values of the measurements, and in other cases they want to use the mean, maximum and minimum values relative to the last N values, for example the last 5 values recorded (N=5). For its operation, the StatsDisplay should store the temperatures or values required for computing the mean, max and min temperatures inside its own class or associated classes with it.

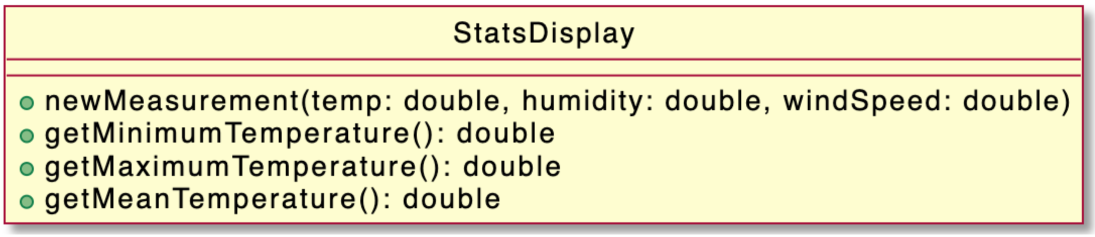
If we have the last 10 values of temperature: 10, 12, 13, 14, 15, 16, 17, 18, 19, 20, the absolute mean is 15.4 and the relative mean of the last N values (when N is 5) is 18.

We are asked to implement a new version of the `StatsDisplay` that allows the scientists to choose between absolute and relative values to the last N values. The scientists also warn us that other possible calculations can be useful in the future, for example exponential decay.

Notice that these features do not depend on the solution of Exercise 2. You only need to assume that the `StatsDisplay` has a method called `newMeasurement` that must be called whenever new data is available. For this exercise only the values for temperature are required, the other values for humidity and wind speed can be ignored.

Class Diagram

The initial design includes the following components:


StatsDisplay
newMeasurement(temp: double, humidity: double, windSpeed: double)
getMinimumTemperature(): double
getMaximumTemperature(): double
getMeanTemperature(): double

Questions

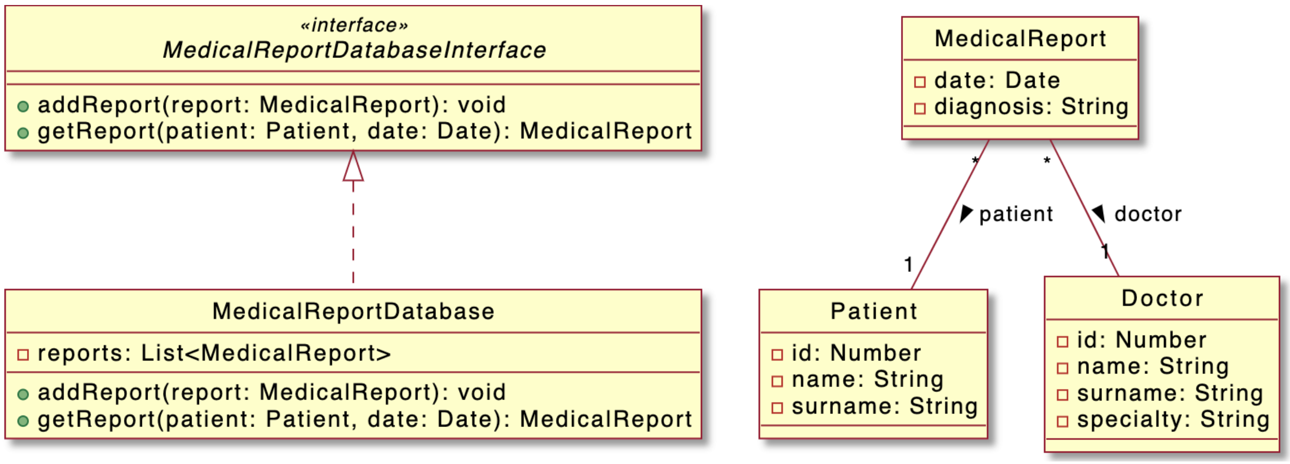
1. (3 points) Justify which design pattern should be applied to allow flexible and interchangeable mean, maximum and minimum calculations in the `StatsDisplay` class.
2. (3 points) Show the class diagram with the operations needed to implement your design.
3. (4 points) Show the complete pseudocode of the classes with the operations or the sequence diagrams of the operations `StatsDisplay:getMinimumTemperature()`, `StatsDisplay:getMaximumTemperature()` and `StatsDisplay:getMeanTemperature()` showing how absolute and relative calculations can be used.

Exercise 4 (10 points)

A hospital has developed a solution to manage the patient's medical history. The system provides an interface to add new reports and get reports identified by patient and date. Medical reports contain the date, patient and doctor data as well as the diagnosis.

We have been hired to integrate this system with another external hospital.

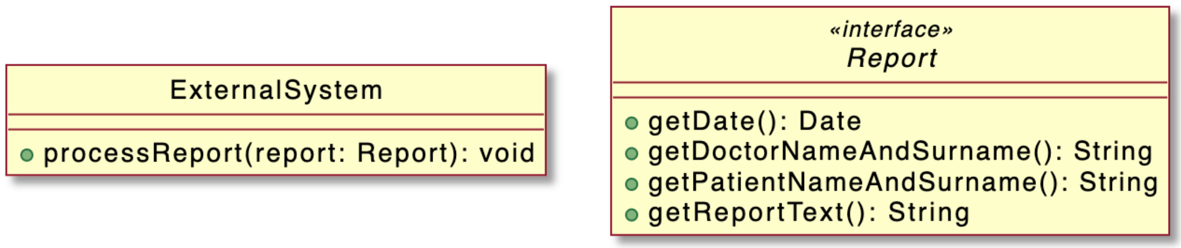
Class Diagram


«interface»
MedicalReportDatabaseInterface
addReport(report: MedicalReport): void
getReport(patient: Patient, date: Date): MedicalReport
MedicalReportDatabase
reports: List<MedicalReport>
addReport(report: MedicalReport): void
getReport(patient: Patient, date: Date): MedicalReport
MedicalReport
date: Date
diagnosis: String
*
*
patient
doctor
1
1
Patient
id: Number
name: String
surname: String
Doctor
id: Number
name: String
surname: String
specialty: String

Integrity Constraints

- MedicalReport key is date + patient.
- Doctor key is id.
- Patient key is id.

In the other hospital they already have a system to manage the reports, and it has the following interface:


ExternalSystem
processReport(report: Report): void
«interface»
Report
getDate(): Date
getDoctorNameAndSurname(): String
getPatientNameAndSurname(): String
getReportText(): String

We want that the original reports of the hospital can be used in the external system as transparently and decoupled as possible.

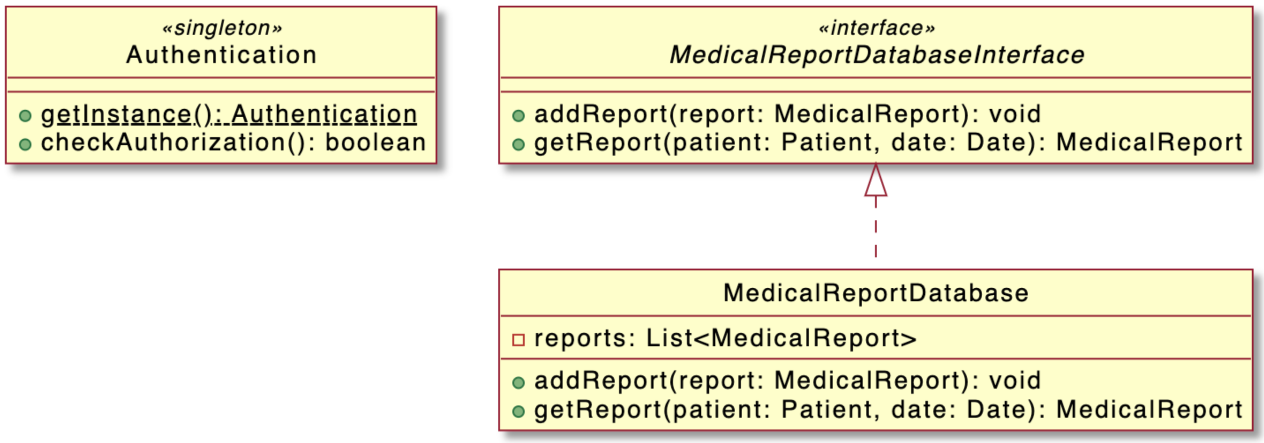
Questions

1. (3 points) Justify which design pattern should be applied to allow the existing medical records to be used in the new external system with minimal changes.
2. (3 points) Show the class diagram with the operations needed to implement this design.
3. (4 points) Show the complete pseudocode of the new classes added in the previous question or the modified existing classes (if any). Notice that you don't need to implement anything in `ExternalSystem`. Your solution only needs to be able to provide classes that can be accepted as input to the `ExternalSystem::processReport()` method.

Exercise 5 (5 points)

After solving the problem of having different interfaces for the reports, the hospital wants to increase the security of the system. They want to ensure that only authorized external users can access and add new reports.

The external system has an interface for accessing the database of medical reports. Also, an `Authentication` singleton is provided to check that the user who is trying to access the report is authorized. This check is done when the `Authentication::checkAuthorization()` is called. If the user is authorized the method returns true, and returns false otherwise.. You can assume that the method `Authentication::checkAuthorization()` can determine the current user on its own.


«singleton»
Authentication
getInstance(): Authentication
checkAuthorization(): boolean
«interface»
MedicalReportDatabaseInterface
addReport(report: MedicalReport): void
getReport(patient: Patient, date: Date): MedicalReport
MedicalReportDatabase
reports: List<MedicalReport>
addReport(report: MedicalReport): void
getReport(patient: Patient, date: Date): MedicalReport

Your solution needs to throw an exception when the user is not valid. We want to implement this security check in the system, but we want to do it in a way that both systems require the minimum changes and stay as decoupled as possible.

Questions:

1. (1 points) Justify which design pattern should be applied to manage the security of the medical report database.
2. (2 points) Show the class diagram with the operations needed to implement this design.
3. (2 points) Show the complete pseudocode of your solution.