Data Access Layer

DataSourceAdapter
- inputFile : String

+ transformInput(inputFile : String) : JSONObject

FileDataListener

+ listen() : void
+ handleEvent(event : Event) : PatientData

DataListener
- dataParser : DataParser
- dataStorage : DataStorage

+ listen() : void
+ handleEvent(event : Event) : PatientData

WebSocketDataListener

+ listen() : void
+ handleEvent(event : Event) : PatientData

DataParser
- dataSourceAdapter : DataSourceAdapter

+ parseData(data : JSONObject) : PatientData

TcpDataListener

+ listen() : void
+ handleEvent(event : Event) : PatientData

1 — 1 (DataSourceAdapter — DataParser)

Data Access Layer

DataSourceAdapter is used to make the data more uniform. It takes in different types of data from Data Listener classes and transforms each input type into a JSON object.

DataParser uses the DataSourceAdapter class, as it transforms the JSON object into the PatientData type for further processing.

Data Listener is an abstract class which is used to handle the event of inputing new data. The listener classes which inherit this class are doing it by using DataParser and DataStorage class instances. DataParser transforms the data from JSON into a PatientData type, which then can be processed further. It uses the DataStorage to have access to the patient's database, so that later it can be added to it.

Data Storage System

DataStorage
- patientsData: List<PatientData>
- dataRetriever : DataRetriever
- patientIdentifier: PatientIdentifier
- alertGenerator: AlertGenerator

+ DataStorage() : void
- storeData(data : PatientData) : void
- retrieveData(patientId : String) : List<PatientData>
- deleteData(patientId : String) : void

1 — 0...* (DataStorage — PatientData)

PatientData
+ patientId : String
- baselineCholesterol : double
- baselineWhiteCells : double
- baselineRedCells : double
- lastSystolicValues : int
- lastDiastolicValues : int
- lastSaturationValues : int
- lastECGValues : double
- timestamp : long

+ PatientData(patientId : String, baselineCholesterol : double, baselineWhiteCells : double, baselineRedCells: int, lastSystolicValues : int, lastDiastolicValues: int, lastECGValues : double, timestamp : long) : void
+ getPatientId() : String
+ getBaselineCholesterol() : double
+ getBaselineWhiteCells() : double
+ getBaselineRedCells() : double
+ getLastSystolicValues() : int
+ getLastDiastolicValues() : int
+ getLastSaturationValues() : int
+ getLastECGValues() : double

StaffDatabase (static)
(...)

+ containsPersonell(staffId : String) : boolean
(...)

DataRetrievier
// we assume that StaffDatabase is a static object

+ DataRetriever() : void
+ verifyPersonnel(staffId : String) : boolean

Data Storage System

PatientData stores all the information about a single patinet's current readings and a respective timestamp. One DataStorage stores the PatientData in a List, hence the caridnality. DataStorage also uses AlertGenerator to evaluateData and generate an alert if needed. It also uses PatientIndentifier to see if the Data is in accordance with the database and if yes store the data in the medical records. It also uses the DataRetriever that is used to check if the medical staff member is in the StaffDatabase (this means that they should be enabled the access to retrieve the data from patients records.

Patient Indentification System

PatientIndentifier uses PatientRecordDatabase to firstly check if the data sent is correct (patient exists and is in the database) and then it can retrieve patient's data. Additionally it creates an IndentityManager that will resolve all the anomalies that arise if the patient is not found in database. Also, the PatientsRecord is stored in PatientRecordDatabase; one database can store an unknown number of patients, hence the cardinality.

Alert Generation System

AlertGenerator
- patientIdentifier: PatientIdentifier

+ AlertGenerator(patientCount: int): void
+ evaluateData(patientData: PatientData): void
- triggerAlert(alert: Alert): void
- fetchPatientData(patientId String): PatientRecord

1 — 0..* (AlertGenerator — Alert)
1 — 1 (AlertGenerator — AlertManager)

Alert
- patientId: String
- condition: String
- timestamp: long

+ Alert(patientId: String, condition :String, timestamp: long): void
+ getPatientId(): String
+ getCondition(): String
+ getTimestamp(): long

AlertManager
// assume there is a connection system that can be used to send messages
+ connectionSystem: ConnectionSystem
(...)

+ alertPersonel(alert: Alert): void
- getStaffId(patientId: String): String
- AlertManager(alert: Alert, staffId: String): void

ConnectionSystem
(...)

+ sendAlert(staffId: String): void
+ sendGlobalAlert(): void
(...)

Alert Generation System

AlertGenerator owns Alert and AlertManager classes. There is only one AlertGenerator and one AlertManager, but the number of Alerts is unknown: from 0 to many because we can generate many Alerts. Moreover, AlertGenerator uses PatientIdentifier to access the PatientRecordDatabase to get the historical medical records. Additionally the AlertManager in order to send alerts needs to use some kind of an external ConnectionSystem that we assume is capable of sendind messages and notifications to the staff.

Patient Identification System

IdentityManager

+ resolveAnomaly(): void

PatientRecord
- patientId: String
- name: String
- dateOfBirth: Date
- medicalHistory: String

+ PatientRecord(patientId: String, name: String, dateOfBirth: Date, medicalHistory: String): void
+ getPatientId(): void
+ getName(): void
+ getDateOfBirth(): void
+ getMedicalHsitory(): void

PatientRecordDatabase (static)
(...)

+ getPatient(patientId String): PatientRecord
(...)

PatientIdentifier
- identityManager: IdentityManager

+ matchPatientId(patientId: String): boolean
+ getPatientRecord(patientId: String): PatientRecord

1 — 1 (IdentityManager — PatientIdentifier)
1 — 0..* (PatientRecord — PatientRecordDatabase)