Table of contents

Alert Generation System

This system is used to efficiently create and distribute alerts in a hospital. The UML diagram below depicts the relationship between the classes:

figure 1

The goal of the Alert Generation System is to alert the appropriate staff in case of an Event trigger. The Design is really simple:

1. The Method evaluateData checks the data against that specific patients thresholds

2. If the evaluateData method returns true, the triggerAlert method is called, which in turn creates an Alert Object

3. The Object contains all the necessary information about the patient, as well as the department the patient is in for more streamlined operations

4. If the Object is created, the AlertManager class will call sendAlert(alert) and send the necessary information to the department specified in the Object With this streamlined operation, time is saved, thus making the response to emergencies easier.

Data Storage System

The Data Storage system is designed to securely handle sensitive Data about patients The UML below shows an example of what it might look like

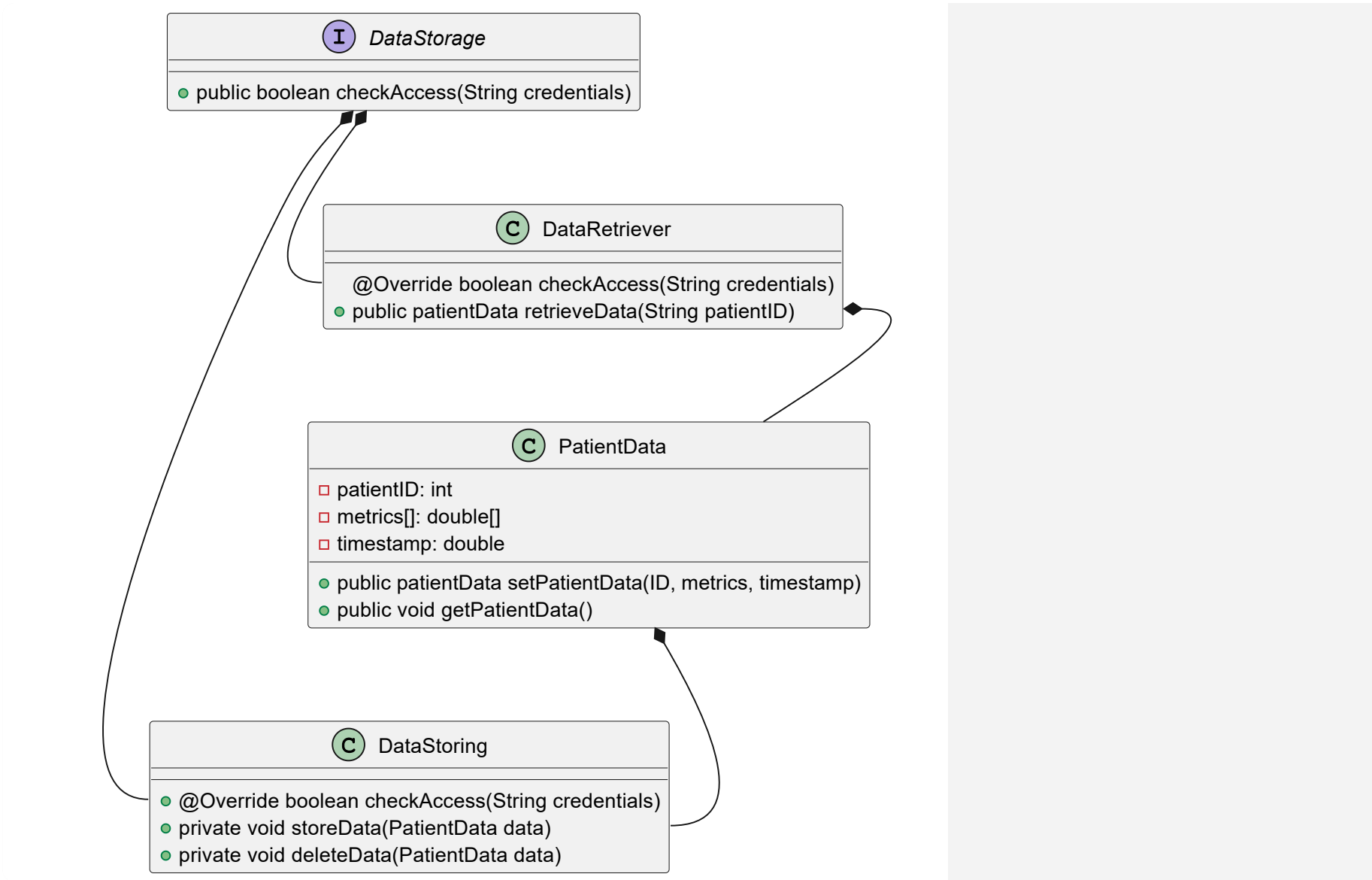

I DataStorage
public boolean checkAccess(String credentials)
C DataRetriever
@Override boolean checkAccess(String credentials)
public patientData retrieveData(String patientID)
C PatientData
patientID: int
metrics[]: double[]
timestamp: double
public patientData setPatientData(ID, metrics, timestamp)
public void getPatientData()
C DataStoring
@Override boolean checkAccess(String credentials)
private void storeData(PatientData data)
private void deleteData(PatientData data)

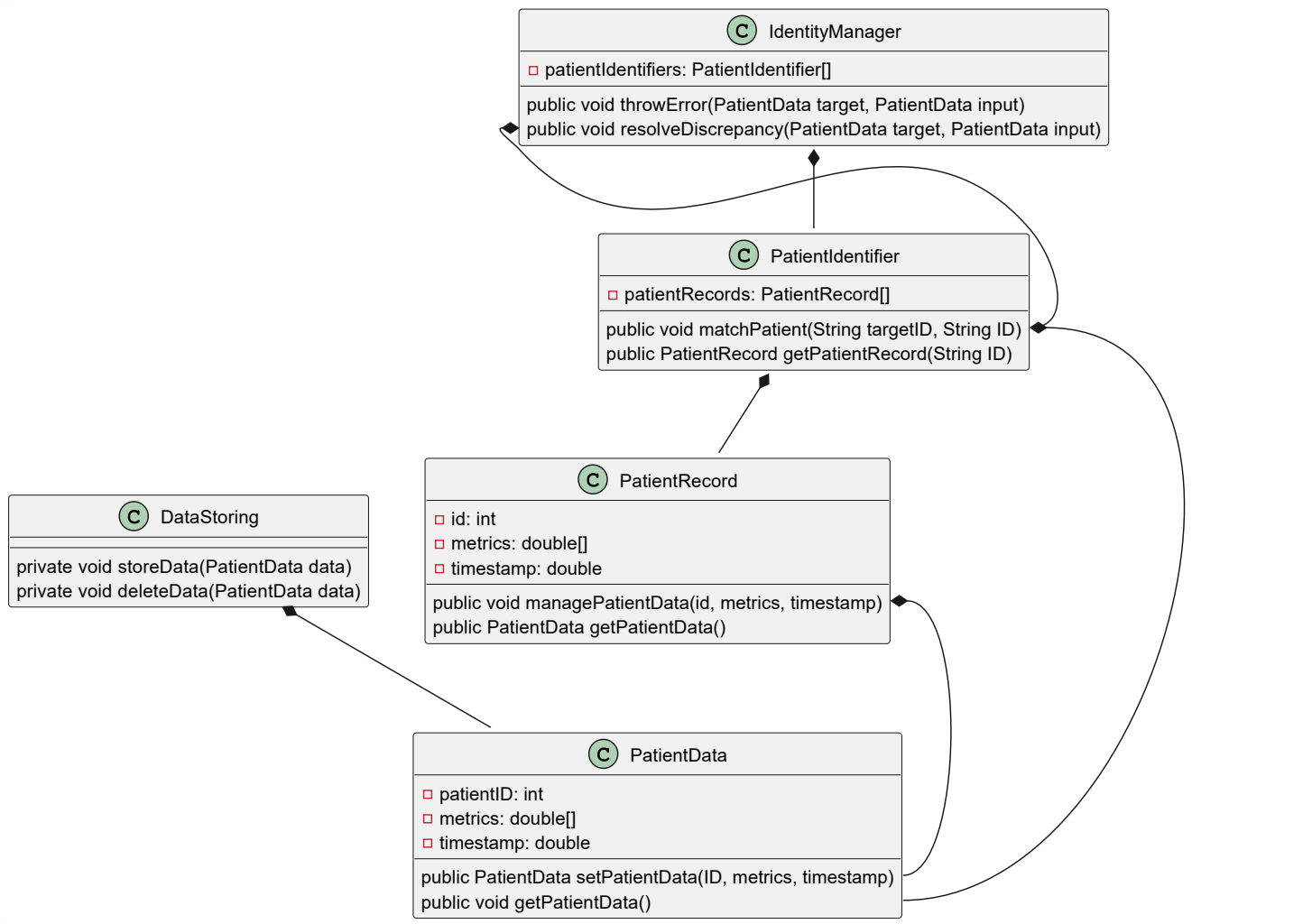
figure 2

In this System we try to handle the data confidentially without sacrificing speed and efficiency. The System is intended to work like follows:

1. In the DataStorage Interface, we create the method checkAccess, this method is used to check if the current user is allowed to access the functionalities
2. The method is later overwritten in the Storing and Retrieving part of the system, to allow for different access levels, for example, we can allow all nurses to retrieve data but not create new data (example)
3. The system allows for storage and deletion of data, if the credentials are met
4. It offers a fair amount of security, since we are always checking if the credentials are met

Patient Identification System

This subsystem ensures proper Handling of the Patient records. This could be an example of how it looks:



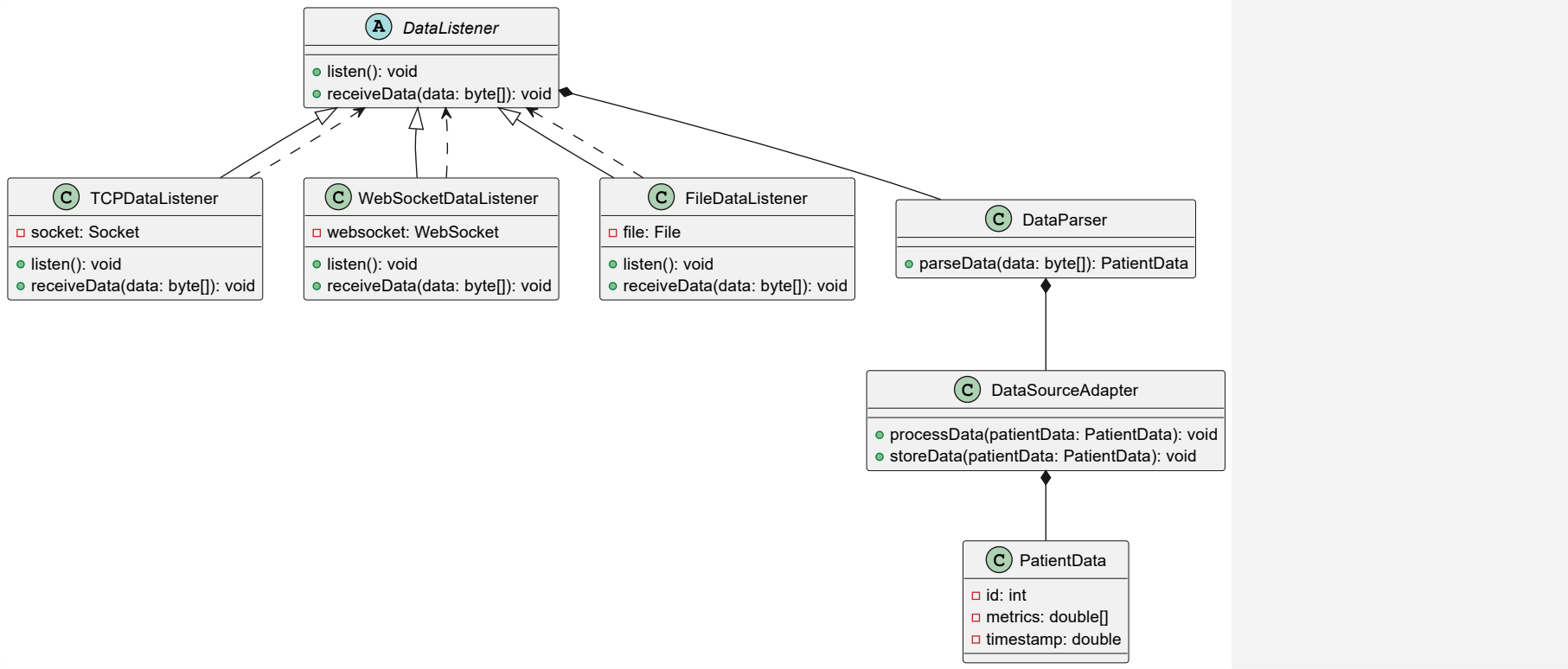
In this system we use some methods from other Classes we created and utilize their respective methods to avoid repetition.

This makes our codebase simpler and better + it follows best practices.

1. PatientIdentifier has a collection of PatientRecord objects, and a method getPatientRecord to retrieve a patient record by ID.
2. IdentityManager has a collection of PatientIdentifier objects, and a method resolveDiscrepancy to handle discrepancies in patient data linkage.
3. PatientData has attributes for patientID, metrics, and timestamp, which make up the object, and a method setPatientData to set the patient data.
4. DataStoring includes methods to store and delete patient data.
5. The relationships between classes are as follows:
    1. DataStorage interface is implemented by both DataStoring and DataRetriever, which means they must provide an implementation for the checkAccess method.
    2. PatientData class is associated with the DataStoring class, indicating that patient data is stored and deleted through the DataStoring class.
    3. DataRetriever class is associated with the PatientData class, indicating that patient data is retrieved through the DataRetriever class.

Data Access Layer

The Data access layer is used for clear and efficient communication between the subsystems. The idea is that the Data is read from different sources, parsed and finally stored in a uniform way



1. The DataListener abstract class defines the interface for listening to data from various sources. It has two methods: listen() and receiveData().

2. The TCPDataListener, WebSocketDataListener, and FileDataListener classes extend the DataListener class and provide specific implementations for listening to data from TCP/IP, WebSocket, and file sources, respectively.

3. The DataParser class is responsible for parsing the raw data received from the DataListener classes into a uniform format, represented by the PatientData class.
4. The DataSourceAdapter class takes the parsed PatientData and processes and stores it in the internal data storage system.
5. The PatientData class represents the standardized data format, with attributes for id, metrics, and timestamp.
6. The relationships between classes are as follows:
    1. DataListener classes send data to the DataParser class.
    2. The DataParser class sends parsed data to the DataSourceAdapter class.
    3. The DataSourceAdapter class stores patient data in the internal data storage system.
    4. The notes at the bottom of the diagram illustrate how the system interacts with external data sources (e.g., signal generator, web portal, legacy system) and internal systems (e.g., data storage).