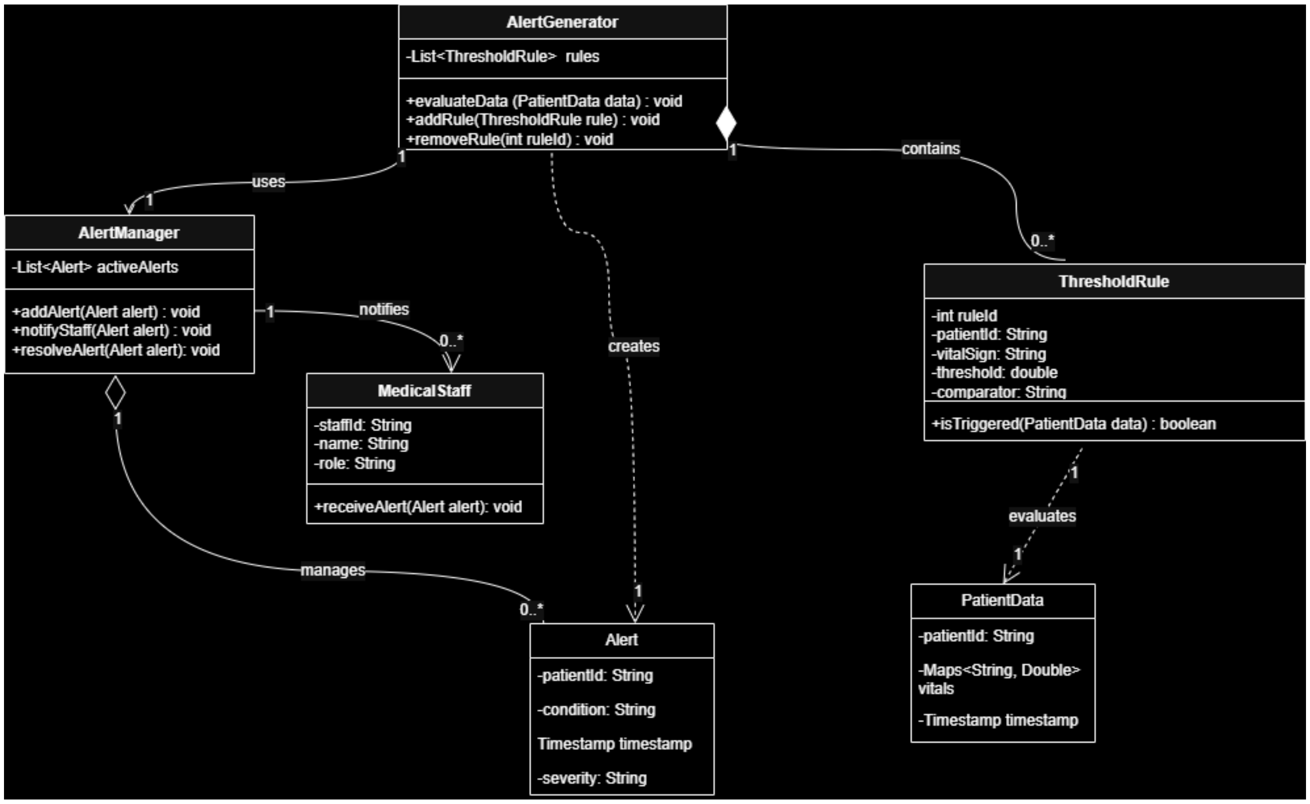
1. Alert Generation System



The UML class diagram designed for the Alert Generation System provides information regarding a healthcare monitoring system, which is made out to generate alerts based on the vital signs of patients. The core of this system is represented by the AlertGenerator class, which maintains a list of ThresholdRule objects and evaluates the incoming PatientData against these rules.

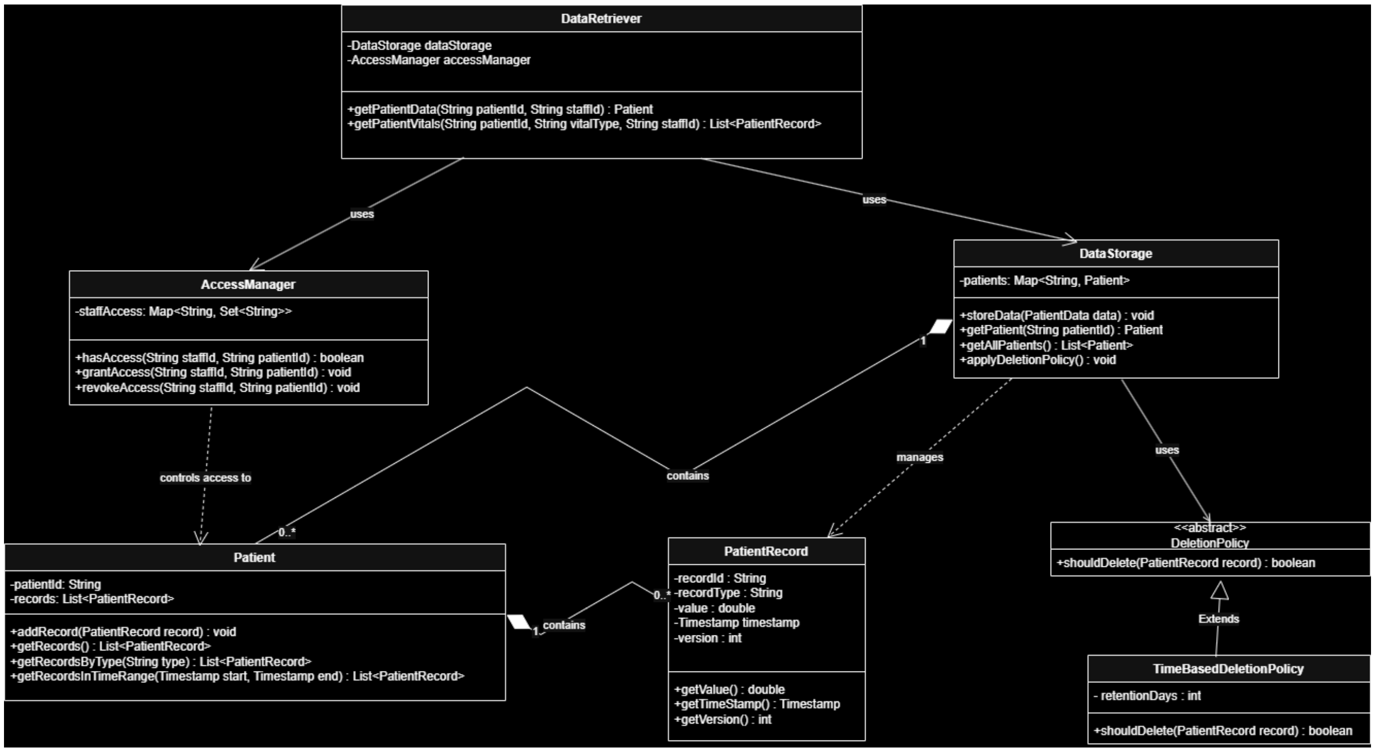
The AlertGenerator allows methods to manage rules (addRule(), removeRule()) and break down patient data. When thresholds are overstepped, it creates Alert objects containing patient ID, condition, timestamp and severity information.

The AlertManager controls the active alerts with methods to add new alerts, notify staff, and manage alerts. It collaborates with MedicalStaff objects that can receive alerts through the receiveAlert() method.

Each ThresholdRule has parameters which define monitoring criteria : rule ID, patient ID, vital sign being monitored, threshold value and comparator. The isTriggered() method examines incoming patient data against these parameters.

This architecture allows personalized patient monitoring with clear separation between data evaluation, alert generation and alert management tasks.

2. Data Storage System


DataRetriever
-DataStorage dataStorage
-AccessManager accessManager
+getPatientData(String patientId, String staffId) : Patient
+getPatientVitals(String patientId, String vitalType, String staffId) : List<PatientRecord>
uses
uses
AccessManager
-staffAccess: Map<String, Set<String>>
+hasAccess(String staffId, String patientId) : boolean
+grantAccess(String staffId, String patientId) : void
+revokeAccess(String staffId, String patientId) : void
DataStorage
-patients: Map<String, Patient>
+storeData(PatientData data) : void
+getPatient(String patientId) : Patient
+getAllPatients() : List<Patient>
+applyDeletionPolicy() : void
1
controls access to
contains
manages
uses
0..*
Patient
-patientId: String
-records: List<PatientRecord>
+addRecord(PatientRecord record) : void
+getRecords() : List<PatientRecord>
+getRecordsByType(String type) : List<PatientRecord>
+getRecordsInTimeRange(Timestamp start, Timestamp end) : List<PatientRecord>
1
contains
0..*
PatientRecord
-recordId : String
-recordType : String
-value : double
-Timestamp timestamp
-version : int
+getValue() : double
+getTimeStamp() : Timestamp
+getVersion() : int
<<abstract>>
DeletionPolicy
+shouldDelete(PatientRecord record) : boolean
Extends
TimeBasedDeletionPolicy
- retentionDays : int
+shouldDelete(PatientRecord record) : boolean

The UML diagram class designed for the Data Storage System shows a comprehensive patient data management system with access control and data retention capabilities.

The DataRetriever class represents the central access point, providing methods to retrieve patient info and vital records.

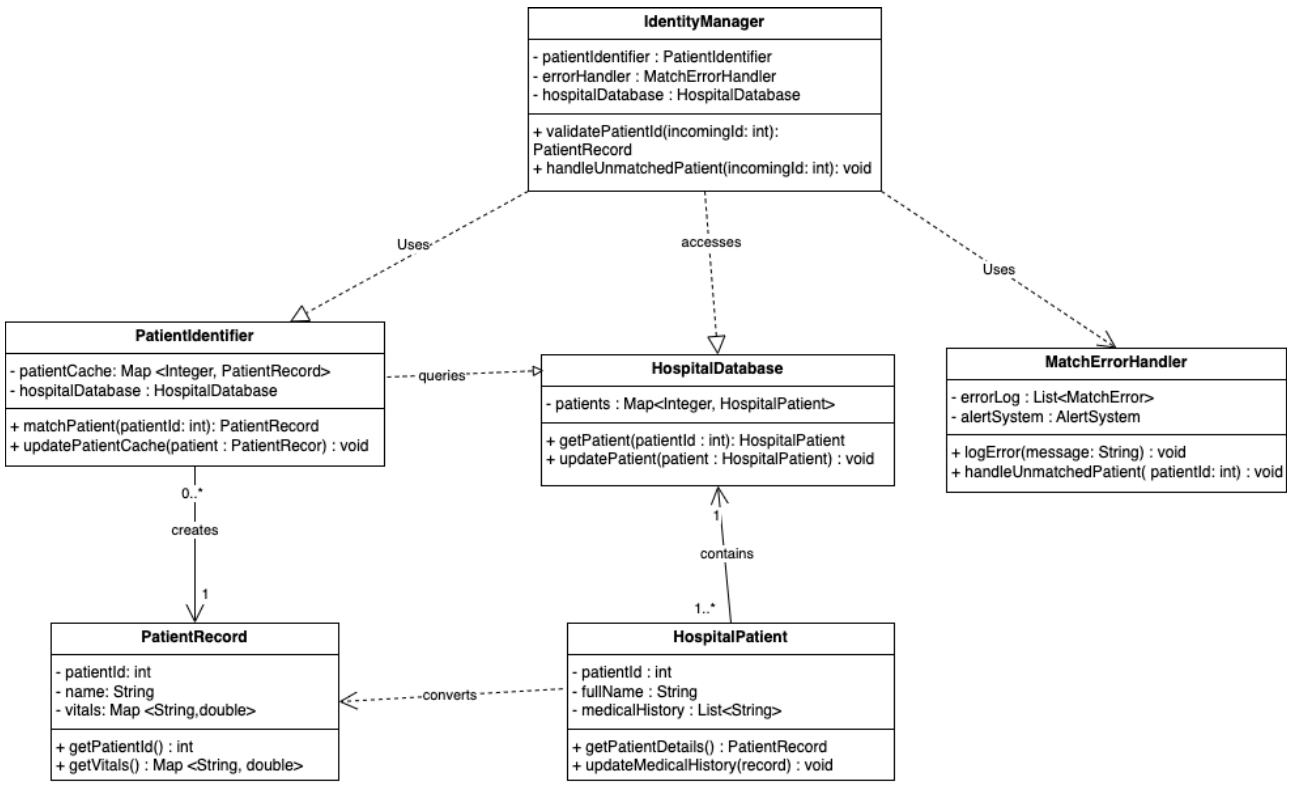
The system uses an AccessManager to control staff access to patient data through methods like hasAccess(), grantAccess() and revokeAccess().

The DataStorage component manages patient records with methods for storing data, retrieving patients and applying deletion policies. Patient information is organized into multiple classes : Patient objects have basic identification and a collection of PatientRecord objects, which store specific measurements with attributes like recordID, type, value, timestamp and version.

The design contains a data retention strategy using the abstract DeletionPolicy class and its concrete implementation TimeBasedDeletionPolicy, which determines when records should be deleted based on retention days.

This architecture addresses healthcare data management challenges including security, access control, data organization and regulatory compliance requirments for data retention.

3. Patient Identification System


IdentityManager
- patientIdentifier : PatientIdentifier
- errorHandler : MatchErrorHandler
- hospitalDatabase : HospitalDatabase
+ validatePatientId(incomingId: int): PatientRecord
+ handleUnmatchedPatient(incomingId: int): void
Uses
accesses
Uses
PatientIdentifier
- patientCache: Map <Integer, PatientRecord>
- hospitalDatabase : HospitalDatabase
+ matchPatient(patientId: int): PatientRecord
+ updatePatientCache(patient : PatientRecor) : void
queries
HospitalDatabase
- patients : Map<Integer, HospitalPatient>
+ getPatient(patientId : int): HospitalPatient
+ updatePatient(patient : HospitalPatient) : void
MatchErrorHandler
- errorLog : List<MatchError>
- alertSystem : AlertSystem
+ logError(message: String) : void
+ handleUnmatchedPatient(patientId: int) : void
0..*
creates
1
1
contains
1..*
PatientRecord
- patientId: int
- name: String
- vitals: Map <String,double>
+ getPatientId() : int
+ getVitals() : Map <String, double>
converts
HospitalPatient
- patientId : int
- fullName : String
- medicalHistory : List<String>
+ getPatientDetails() : PatientRecord
+ updateMedicalHistory(record) : void

This UML class diagram provides information about a patient identity management system for healthcare environments. The IdentityManager is the central coordinator, validating patient IDs and handling unmatched patients.

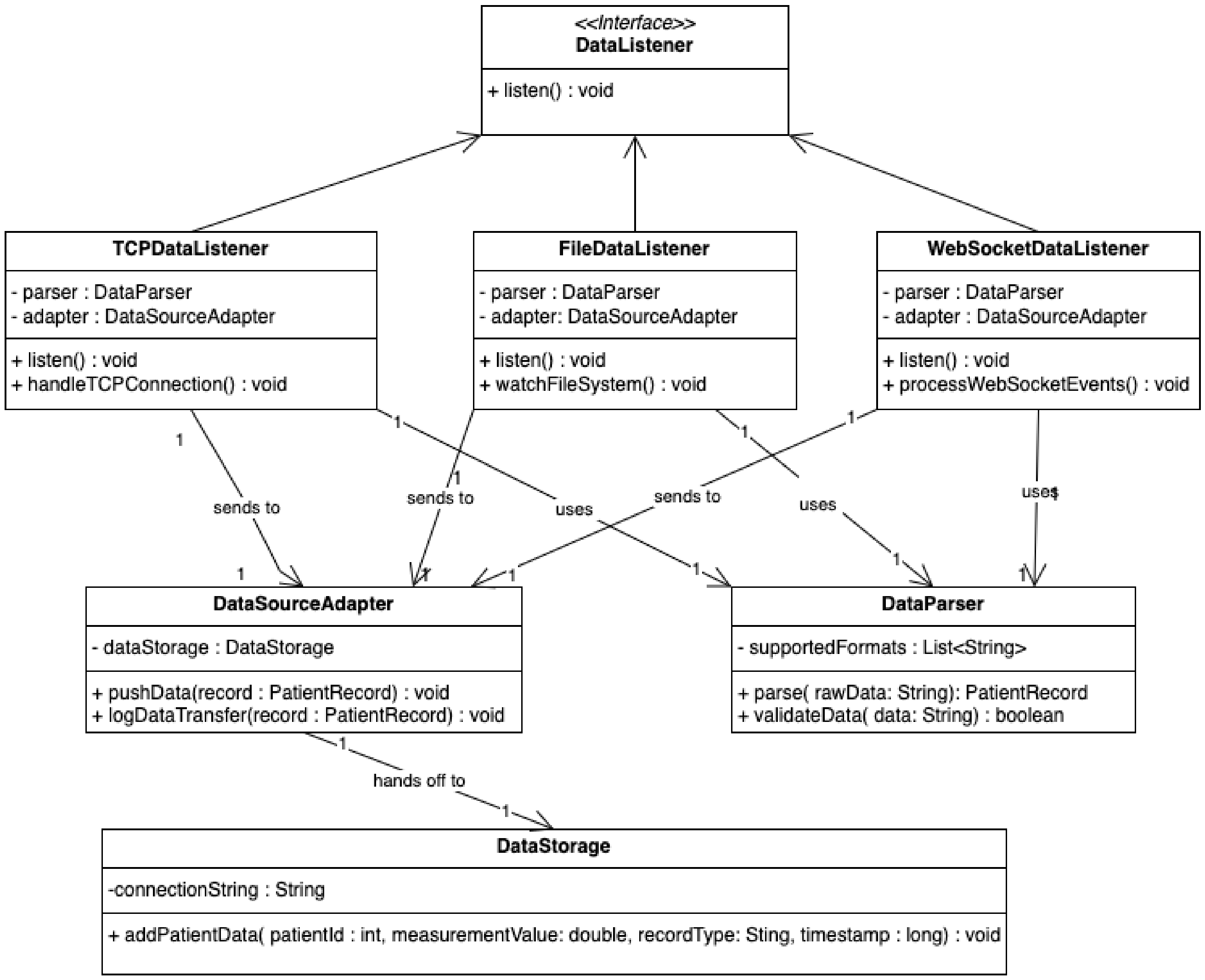
The system relies on 3 key components :

- PatientIdentifier manages a cache of patient records and provides methods to match and update patient information;
- HospitalDatabase stores thorough patient data as HospitalPatient objects;
- MatchErrorHandler processes and logs errors when patients cannot be properly identified.

The diagram shows how patients PatientRecord objects, which contains basic info like ID, name, vitals, relate to the more detailed HospitalPatient records, which include full name and medical history. The conversion relationship between these record types allows for translation between external and internal data representations.

This architecture references the critical challenge of correctly identifying patients across different hospital systems while providing good mechanisms for handling potential matching errors.

4. Data Access Layer



<<Interface>>
DataListener
+ listen() : void
TCPDataListener
- parser : DataParser
- adapter : DataSourceAdapter
+ listen() : void
+ handleTCPConnection() : void
FileDataListener
- parser : DataParser
- adapter: DataSourceAdapter
+ listen() : void
+ watchFileSystem() : void
WebSocketDataListener
- parser : DataParser
- adapter : DataSourceAdapter
+ listen() : void
+ processWebSocketEvents() : void
sends to
sends to
uses
sends to
uses
uses
1
DataSourceAdapter
- dataStorage : DataStorage
+ pushData(record : PatientRecord) : void
+ logDataTransfer(record : PatientRecord) : void
DataParser
- supportedFormats : List<String>
+ parse(rawData: String): PatientRecord
+ validateData(data: String) : boolean
hands off to
DataStorage
-connectionString : String
+ addPatientData(patientId : int, measurementValue: double, recordType: Sting, timestamp : long) : void

This UML class diagram represents a data integration system for patient healthcare records. The system is built around the DataListener Interface, which provides a common structure for 3 different data sources : TCP connections, file systems and WebSockets.

Each listener implementation (TCPDataListener, FileDataListener, WebSocketDataListener) communicates with both a DataParser for processing and validating incoming data and a DataSourceAdapter for storing records. The DataParser supports multiple formats and transforms raw data into structured PatientRecord objects. The DataSourceAdapter handles the transfer of processed records to the DataStorage component, which provides methods with their associated data (Id, value, type, timestamp).

The architecture separates data acquisition, processing and storage concerns, making the system adaptable to different data sources while maintaining constant handling of patient information.