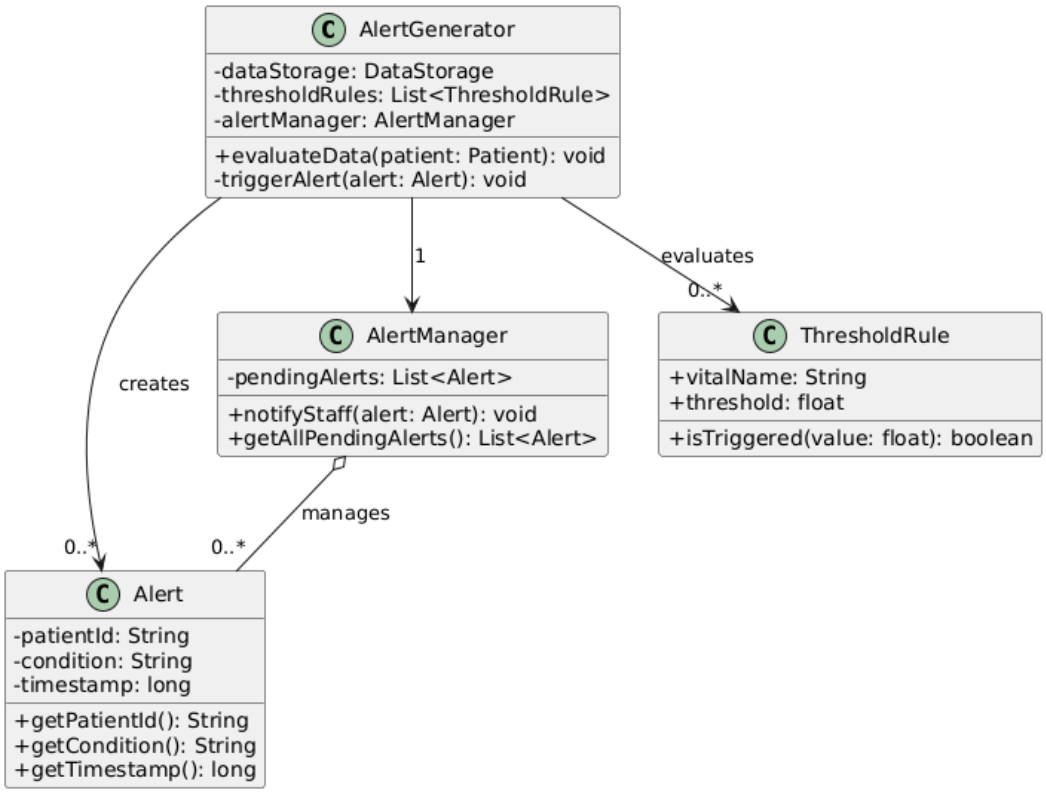
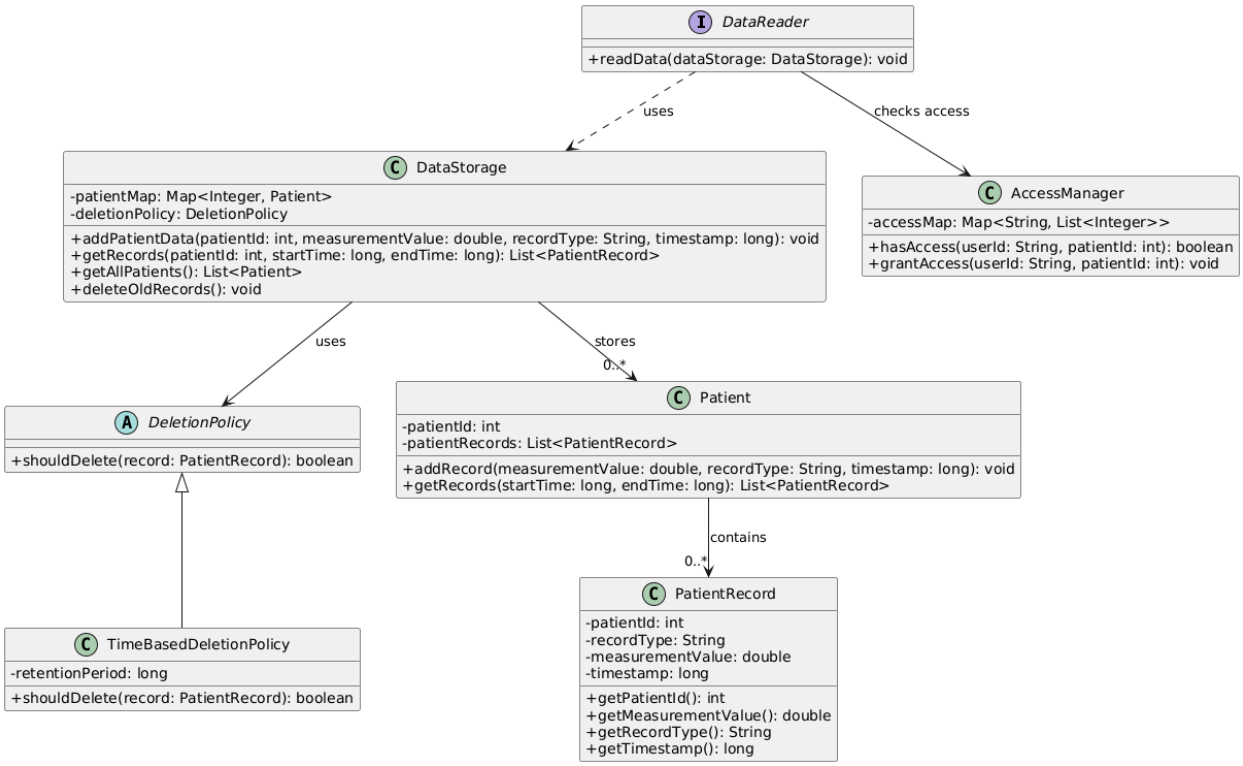
1. Alert Generation System



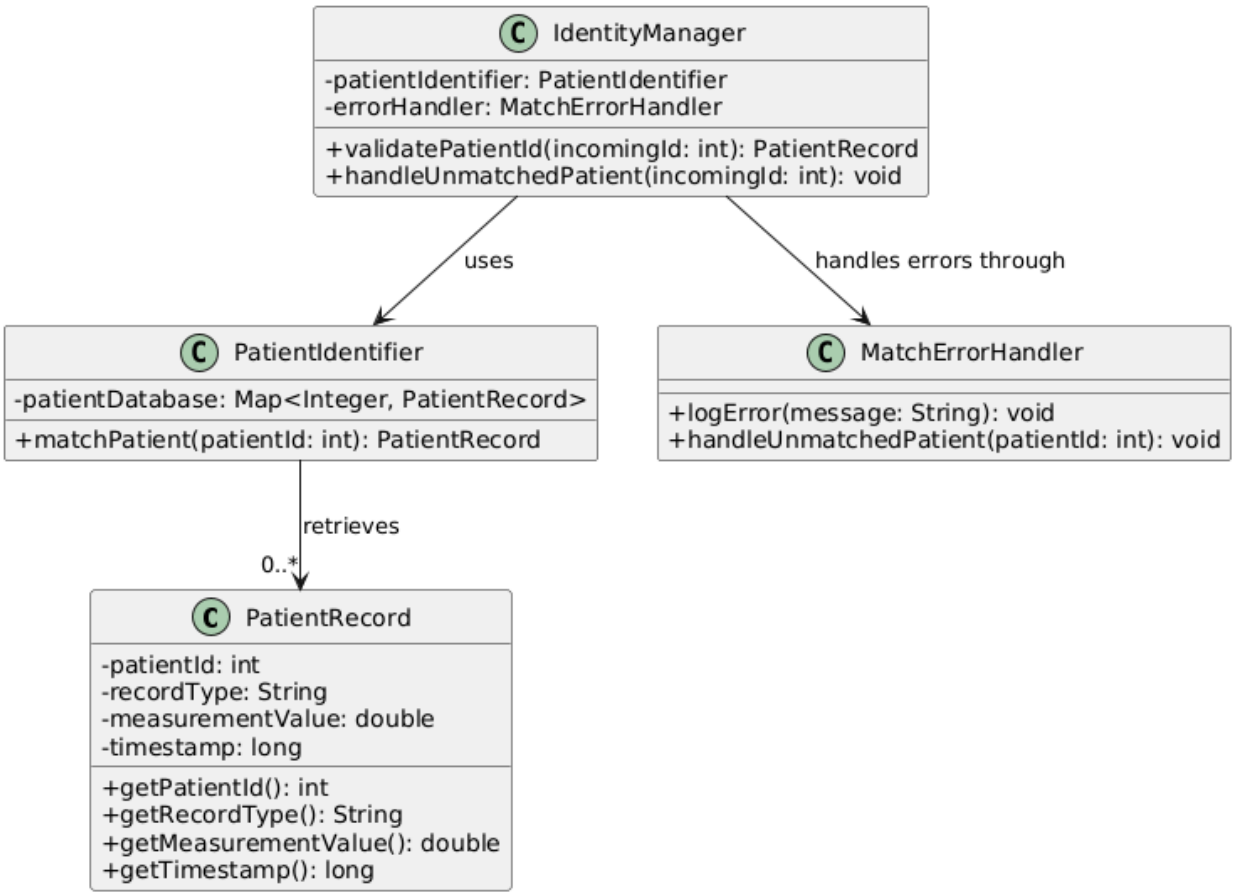
The Alert Generation System monitors patient data in real time and triggers alerts when a vital sign crosses a threshold. The AlertGenerator handles this evaluation process. It works with a list of ThresholdRule objects, where each rule is specific for each patient. For instance, one rule might say, “Trigger an alert if heart rate goes over 130.” Each rule includes the vital sign it’s tracking (like heart rate or blood pressure), the threshold value, and the logic to determine whether an incoming value breaks that threshold. As new data comes in, the AlertGenerator checks it against all applicable rules. If any rule is triggered, an Alert object is created. This alert includes important details such as the patient’s ID, the condition that caused the alert, and the exact time it happened. Once the alert is created it’s passed to the AlertManager. The AlertManager keeps a queue of active alerts. It also provides methods to notify medical staff when something critical occurs, and it allows staff to view the list of current alerts. This setup makes it possible to have personalized alert rules for different patients, maintaining modularity. Moreover, the alert-handling is kept separate from the data-checking, and threshold rules can be added or changed easily.

2. Data Storage System



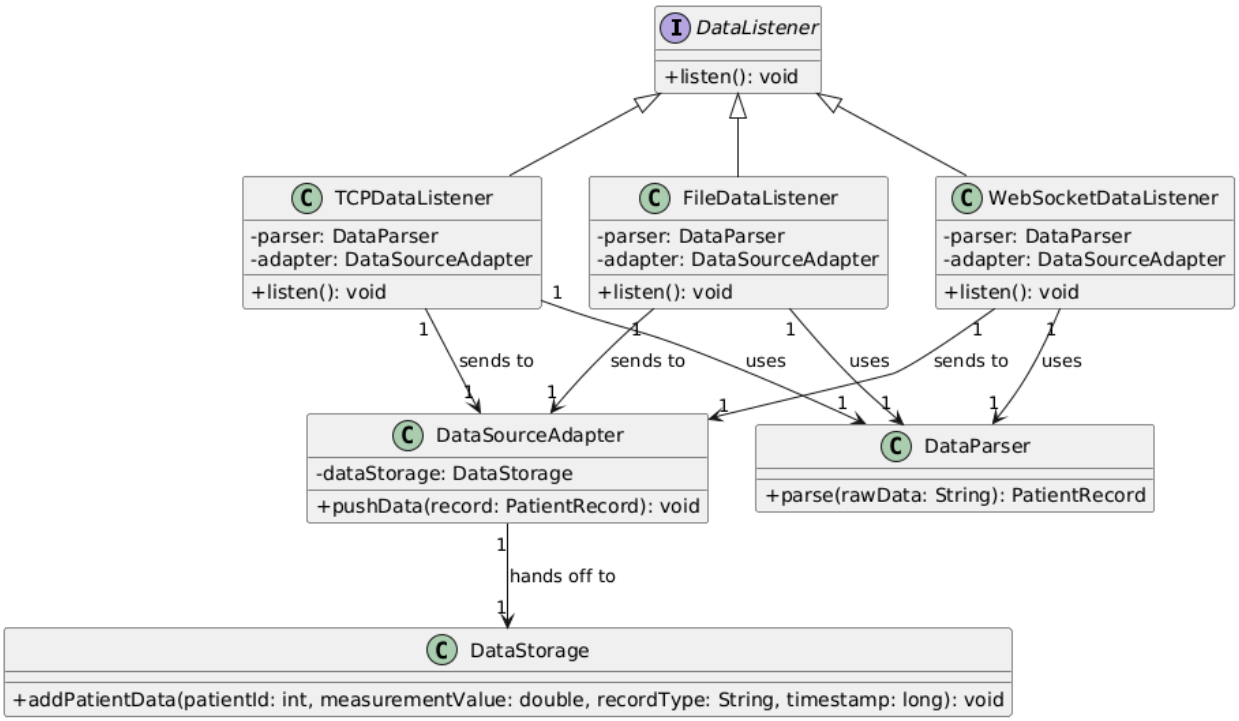
The Data Storage System is responsible for storing, retrieving, and managing all incoming patient data. When the system receives new measurements, the DataStorage class links each data point to the correct Patient using their unique ID. If the patient doesn't exist yet, a new Patient object is created and added to the system. Each Patient contains a list of PatientRecord entries, which hold details such as the record type (e.g., "HeartRate"), the measurement value, and the timestamp of when the measurement was taken. To manage data cleanup, DataStorage relies on a DeletionPolicy. This is an abstract class that defines whether a PatientRecord should be deleted or not. One implementation of this is the TimeBasedDeletionPolicy, which removes records that are older than a specified period. The system checks each record and deletes it if the policy says it's outdated. Access control is handled by the AccessManager, which ensures that only authorized users can retrieve data for a specific patient. It keeps track of which users have permission to view which patients and can give or check access accordingly. External data can be read and imported through the DataReader interface.

3. Patient Identification System



The Patient Identification System is responsible for matching incoming data to the correct patient using their ID. When new data arrives, the IdentityManager takes charge of validating whether the incoming patient ID matches one from the system's database. It does this using the PatientIdentifier, which holds a map of known PatientRecords. Each record contains basic info like the patient ID, the type of record (e.g., "HeartRate"), the measurement value, and the timestamp. If the ID is valid and found in the database, the PatientIdentifier retrieves the matching PatientRecord. This allows the system to link the new data to the correct patient and use their stored information. If the patient ID doesn't match any record in the system, the IdentityManager passes the issue to the MatchErrorHandler. This class handles errors like unmatched or incorrect IDs. It provides methods to log errors and handle unmatched patients in a structured way, so the system can respond properly to unexpected data.

4. Data Access Layer



The Data Access Layer handles incoming data from the external world, such as signal generators. These generators can send data through different channels like TCP, files, or WebSockets. Each type of input has its own class: TCPDataListener, FileDataListener, and WebSocketDataListener. All three implement the DataListener interface, which defines the method listen(), used to start receiving data. Each listener includes a DataParser and a DataSourceAdapter. The DataParser is responsible for taking raw incoming data (such as JSON or CSV) and converting it into a PatientRecord object. This ensures that no matter how the data arrives, it's turned into a standardized format that the rest of the system can understand. Once the data is parsed, it is passed to the DataSourceAdapter, which links the external input and the internal system. The adapter calls the addPatientData() method on the DataStorage class to insert the new record into the system.