Explanation of the UML diagrams of subsystems

1. Alert Generation System

The Alert Generation System monitors incoming data and sends alerts when the vitals cross a patient's threshold. The AlertGenerator class is responsible for checking the incoming PatientData. When a new record comes in, the method "check" in the AlertGenerator is run. It gets the corresponding patient from the DataStorage database and then gets the thresholds from the Patient.
For each incoming record, the system checks whether any threshold is crossed using evaluate() method in the Threshold class. If it returns true, an Alert is created containing the patient ID, condition, and timestamp. This is then passed to the AlertManager, which notifies medical staff via alertStaff().

2. Data Storage System

The Data Storage System handles the storage and retrieval of all patient data. The central interface is DataStorage, which defines the method signatures for adding and getting data. The interface is realized by the class Database, which uses a Map<Integer, Patient> to store all patients. Each Patient holds a list of their records, each with a timestamp, value, and record type (e.g., blood pressure). Using a Map allows fast retrieval of patients by their unique ID, but since we use the DataStorage interface, this setup still allows other ways of storing the data.
All implementations of DataStorage also have a method deleteOldRecords(), this method is used to remove outdated data (i.e. records older than 30 days or another condition) to keep the dataStorage from getting unnecessary large.

The DataRetriever is responsible for getting data from the database, and it can only be used by authorized medical staff. This ensures separation between the storage and access so it stays organized. It also helps with securaty for personal data. DataRetriever can retrieve all records by patient and time by calling the methods defined in the DataStorage interface.

3. Patient Identification System

This system's objective is to match every incoming data point (generated by a generator) to the correct patient in the hospital. The class PatientIdentifier tries to link the incoming PatientRecord to a patient in the HospitalDatabase by ID. If the ID is missing or doesn't match, it will send the record to IdentityManager. IdentityManager tries to still find a match i.e. by looking if the records of a patient in GeneratorDatabases match

the history of a patient in the HospitalDatabase. If no good match is found, the system sends a notification, handled inside IdentityManager. This prevents mislabelling, while not letting information get lost because the record is sent with the notification.

This separation of tasks ensures a clear structure. PatientIdentifier handles the standard case, IdentityManager handles edge cases. They both use the incoming record and the hospital's database, while the IdentityManager also uses the generator's database.

For this UML diagram, I assumed the data storage structure of the hospital and the data generators would be the same: a Map of Patient objects, that each holds a list of records.

4. Data Access Layer

The Data Access Layer is a system to process incoming data the generator. It's designed around the DataListener interface, which has the implementations TCPDataListener, WebSocketDataListener, and FileDataListener. These classes handle different types of input streams, get the data, and then send the data to the DataParser class. The parser reads the incoming data and parses it into PatientData objects. These are then passed to the DataSourceAdapter, which adds the patient records in the database with the addData() method.

The usage of the interface DataListener ensures flexibility and extensibility. You could add a new data source without having to rewrite anything.

General remarks

Security & Privacy Matter
All classes that give or send patient data to humans are:

1. DataRetriever: returns all patient data that is asked for. This can only be used by medical staff and has a method to check whether they are authorized.
2. AlertManager: sends alerts to medical staff when patient vitals cross a threshold.
3. IdentityManager: sends a notification if a record can't be matched with a patient

To ensure the patient's privacy, the connections between the IdentityManager/ AlertManager and the staff should be well secured and the method to check if a user is authorized medical staff in the DataRetriever should be well coded.

In all classes, the instance fields are made private and are only used when necessary.

Multiplicities

With many of the multiplicities, the multiplicity depends on the actual implementation. i.e. an AlertGenerator could technically use more than one DataStorage or AlertManager. But since it's the most logical/easiest, I assumed it's 1.