**patientIdentificationSystemUML**

The CHMS will appropriately annotate patients with alerts so that response to alerts can be given in time after reference vitals are exceeded given a threshold. In a very simplified way, we can visualize the core functionality of this subsystem along with monitoring patients' data, issuing alerts, dispatching them to medical staff, and securely storing health records.

The most essential class in the system is Patient, which contains basic patient information (patientId, name, age). Each patient will have a set of personalized threshold rules defined in ThresholdRule that indicate conditions under which an alert should be sent. Depending on the specific rules, these might involve some health metrics, acceptable limits, and some logic describing when it can be assumed that a threshold has been exceeded.

The AlertGenerator class receives patient data and validates for potential threshold violations. A threshold violation may be when a patient's vitals exceed their predetermined set of limits. Once this condition is triggered, an Alert object is created with the ID identifying the patient, the condition which has been violated, and a timestamp.

The management and dispatching of alerts are handled by the AlertManager, which maintains accountability for handling multiple alerts and for ensuring that the alerts reach their rightful MedicalStaff members. Medical staff, defined by staffId, name, and role, get the alerts and take action based on them.

For keeping a history, Tracking of alerts and storing of patient vitals are covered by DataStorage, ensuring all health records are logged in a secure manner. This enables trend analysis, reporting, and future decisions.

The UML diagram provides a well-structured, scalable architecture for real-time alert processing with strict response norms for health emergencies, thereby aiding in effective clinical patient care and monitoring in the hospital. More modifications, if required, let me know.

**dataStorageSystemUML**

The UML class diagram shows the principal objects and the relationship among them in the CHMS data storage system. This diagram is mainly concerned with the coherently and securely handling of patient data by describing the functions of secure data storing, historical record keeping, data retrieval, and access control.

At the core is the Patient class, which describes individual patients who are the users of the system. Every patient has a unique patientId along with some personal details, such as name and age. The Patient class has a collection called healthRecords (modeled as List<PatientData>) that contains multiple instances of patient records. This 1..\* composition relationship indicates that each patient has one or more health records.

The PatientData class models a single health record with a timestamp, recording the date and time when the data was recorded, a metric that defines the vital sign (for example, heart rate, blood pressure), and a value for storing the measurement. The method getDataDetails() can be used for displaying or exporting the formatted details.

The DataStorage class acts as a home for all patient data. The data are stored in a map using patient identifiers, where each key points to a list of PatientData objects. The class includes operations like storeData() to accept new data points and retrieveData() to permit querying for a given interval of time. It also defines an operation applyDataDeletionPolicy() for managing data retention and deletion of obsolete records in light of data deletion policies.

Concerning data storage, the DataDeletionPolicy class helps define retention rules using an attribute expirationDays and includes the method enforceDataRetention() to cyclically delete outdated records.

On the other hand, the DataRetriever serves as the query interface, allowing medical personnel to retrieve the patient records based on identifiers like ID or time.

Access Control governs the securing of sensitive data. It maintains a list of authorized roles and checks to see if an access attempt was made by a user role entitled to access the data via the verifyAccess() method. The MedicalStaff class represents the health professionals who interact with the system and request data through the DataRetriever following authorization constraints.

Thus, the diagram brings about the organized manner of maintaining the health records of the patients in the health sector in a secure way and its efficiency.

**dataAccesLayerUML**

UML diagram illustrates the patient identification system as part of CHMS. It assures linking every incoming data record to a verified patient. It starts with defining the HospitalPatient class with which one can model an official record of patients from the hospital's database. Characterizing in this class is patientId, name, age, and, moreover, contains a medicalHistory modeled as a List<String>. The method getPatientDetails() brings out the summary of formatted information about the patient's details.

Encapsulation incoming data will refer to the DataRecord class that holds rawPatientId, a map of vitalsData, which means various health metrics, and timestamp indicating time when recorded. Returns this method getRawData() for further processing. It acts as an association between DataRecord and PatientIdentifier denoting that raw data is passed to the identification process.

The class patient identifier is responsible for matching, using hospital patients registered in the hospital. A database is held in a map < int, HospitalPatient >. It has a method matchPatient(), which returns MatchResult to indicate if a match has been found (the isMatch boolean) in addition to an errorMessage if the match has not been found, retrievable through getResult().

The Manager Identity class oversees the identification process-performing verification on data records through probably the method verifyIdentity(). Any error that is raised is passed over to ErrorHandler class. ErrorHandler keeps a log consisting of the List<String> type mismatch records and employs methods such as logMismatch() and resolveIssue() for management of the errors.

The MedicalStaff class assigns medical personnel as the recipient for alerts that will be generated against the identification anomalies through the implementation of its receiveAlert() method. Indeed, this comprehensive design stands as a hallmark for robust matching of patients and error handling-all integral to accurate data integration within a hospital environment.

**alertGenerationSystemUML**

The Data Access Layer in the CHMS provides an interface with an external data source bridging the system internally such that raw patient data can be retrieved and parsed from external channels. Herein, the DataListener interface is used as the data layer interface standardization for data access via the fetchData() method. This interface has been implemented in various specialized classes, namely, TCPDataListener, WebSocketDataListener, and FileDataListener. Each of these concrete listeners deals with data input from their respective sources; the first provides network-based connectivity, the second for real-time data streaming, while the last listens for any data from stored log files.

Once raw data has been fetched from any external source, the fetching is then handed over to the DataParser class. With an attribute formatType to indicate what sort of data is being used (for example, JSON or CSV), the parseData() method in DataParser imports raw string data into a structured PatientData object. This guarantees that data being transmitted is formatted for processing.

The structured data is then forwarded to the DataSourceAdapter, which uses its forwardData() method to bridge the data from being parsed to the system's storage solution. Storage itself is handled by the DataStorage class, which maintains a mapping of patient ids to the list of PatientData objects. DataStorage provides for storage of new data (storeData()), retrieval of data mapped by patient id within a specified time range (retrieveData()), and supporting data freshness through its applyDataDeletionPolicy().

After that, every PatientData record, which contains the relevant timestamp, metric, and value attributes in addition to the method getDataDetails(), is the basic unit of patient information that is actually stored. Eventually, the HealthRecords component will consolidate these records for long-term historical retention allowing retrieval of the full patient history through its retrieveHistoricalRecords() method. This layered architecture presents an extensible and modular framework in favor of flexibility and separation of concerns in the processes of data access.