Alert Generation System

ThresholdRule holds one limit for one vital sign, such as “heart rate > 130 bpm”. Each rule knows how to check a single value through isViolated. It carries no other logic, so rules are easy to add, remove, or store in a database later.
AlertGenerator owns many ThresholdRule objects (shown by the “1..*” aggregation). On every new measurement the generator looks up the matching rule, asks the rule if the value is violated, and if so creates an **Alert**. The generator does not route the alert; it only decides when an alert should exist. This keeps the generator small and easy to test.
Alert is a simple, immutable data object. It records the patientId, the violated condition, and the timestamp. Because the class has no setters, no other part of the system can change an alert once it is created, protecting audit history.
AlertManager is responsible for sending alerts to staff. The generator depends on the manager (dashed arrow) and calls dispatch(alert) whenever an alert is created. How the manager forwards the alert e-mail, dashboard, pager, is hidden inside the manager, so routing rules can change without touching generator code.

Data Storage System

PatientRecord is a single, immutable data point. It stores patientId, the recordType (for example “BloodPressure”), the numeric measurement, and the timestamp. Because the class offers only getters, once a record is created its contents cannot be altered, this protects audit trails.
DataStorage is the central repository. Internally it keeps a Map from patient ID to a list of PatientRecord objects; the aggregation line in the diagram (“1-to-many”) shows this ownership.
addPatientData appends a new record.
getRecords returns every record whose timestamp falls between start and end, enabling real-time dashboards and historical trend graphs.
deleteOlderThan enforces a retention policy by purging data that is older than a caller-supplied interval, meeting the “deletion policies” requirement.
DataRetriever is the only class doctors or analysts touch. It receives a reference to DataStorage in its constructor and exposes a query method that mirrors getRecords. This extra layer lets future versions add access-control checks (for example, role-based filtering) without changing storage internals

Patient Identification System

HospitalPatient is a read-only data object holding the hospital's ground-truth record: database id, full name, dateOfBirth, and an optional medicalNote. Only getters are exposed, so application code cannot alter any official patient data.
PatientIdentifier owns the matching logic. Its single public method match(simId) receives the simulator's integer ID and returns the corresponding HospitalPatient. The implementation might query a hospital database or an in-memory cache. By isolating the lookup here, different matching strategies (exact ID, barcode, RFID, etc.) can be swapped without touching the rest of the system.
IdentityManager orchestrates the process and handles edge cases. It is constructed with one PatientIdentifier and exposes linkOrHandle. If match returns a patient the manager forwards that record to downstream components; if it returns null the manager decides what to do, log an error, raise an alert, or quarantine the data. This concentrates anomaly handling in one place and prevents silent data loss.

Data Access Layer

The Data Access layer pulls bytes or text from the outside world and turns them into clean PatientRecord objects for storage.Six classes cover every responsibility while keeping the rest of the CHMS unaware of how data arrives.
DataListener is a common interface with one method, listen(). It hides network or file details behind a single contract.
TCPDataListener, **WebSocketDataListener**, and **FileDataListener** each bind to a different source, raw TCP socket, WebSocket, or log file tail. All three forward the exact strings they receive to a DataSourceAdapter, shown by dashed dependency arrows.
DataParser converts raw strings into strongly-typed objects. Its setFormat method lets the system switch between CSV, JSON, or any future format without changing listeners.
DataSourceAdapter glues the input side to storage. It owns one DataParser and one reference to DataStoragefrom the previous subsystem. When a listener calls handle(raw), the adapter stores that record via DataStorage.