`Report analysis:`

`com.cardio_generator.outputs` (0%)

Classes in this package output their data through the console, TCP, WebSocket or by writing to files. Because they handle I/O, units aren't tested but are rather checked during the integration stage. In order to test a predefined area, output streams need to be redirected.

`com.cardio_generator.generators` (0%)

Such tests produce data in real time and rely on randomness as well as scheduled events. This makes them different from typical unit tests which don't test them properly without a specific testing framework for scheduling.

`com.cardio_generator.HealthDataSimulator` (0%)

This is the main class for runners. Since it is called directly from an external service, it is left out of unit testing. Testing this aspect works best when using integration or system testing.

`com.data_components` (0%)

It seems that this package has never been opened. There was no evidence that testable logic was executed after my observation.

`com.data_management` (46%)

The core feature of handling and finding patient records was tested as part of the study. Even so, edge scenarios such as improper IDs, applying the retention policy and error situations in patient identity identification were not well discussed.

`com.alerts` (74%)

Sensors measured core alert measures such as heart rate and blood pressure. Combined alerts and nurse-triggered alerts were not fully tested in some complex exercises and could be tested further with sharper scenarios in the future.

First UML:

All information about patients is kept in the DataStorage class. The system is designed according to Singleton to ensure access remains the same everywhere. It uses a Map inside the class to tie patient IDs with their Patient objects. As new information arrives, DataStorage uses addPatientData to update the correct record.

In every Patient object, the list of PatientData stores the details of a single vital reading such as heart rate, oxygen saturation or blood pressure. So, this design is simple to grow and can take on additional kinds of data.

Hospital staff or systems use DataRetriever as the tool to request particular data. It exchanges data with DataStorage to access only the patients' records that suit specific criteria like time range or the type of record. Because we have this clear separation of roles, AlertGenerator can use the latest data to send out timely alerts and reports.

Because of this architecture, components (Patient and PatientData) are strongly linked, whereas connections to services like storage and querying are loose. Concurrent update support, easy scalable monitoring and support for many data or output systems (such as WebSocket and TCP) makes it suitable.

Second:
This class analyzes information about patients and raises alerts if they show signs of serious problems. DataStorage is used by the system to pull patient records for assessment. It looks at previous measurements and, based on set rules, finds and returns Alert objects if someone's heart rate is too high, oxygen saturation too low or vital signs have fallen severely.

PatientId, a description of the alert's meaning and a date and time it was registered are all necessary attributes in each Alert instance. Usually, the class serves as a base for other alert classes like ECGAlert and BloodPressureAlert for special types of logic.

Once new alerts have been created, they go to the AlertManager. The AlertManager logs all received alerts and can direct them to message logging systems, displays on the computer interface or healthcare group alerts. A call to receiveAlert(Alert) occurs when a new alert is ready and dispatchAlerts() sees that they are delivered.

These three classes make sure health is watched over in real time and that staff are notified when they need to intervene.

Third:

This module makes certain that all incoming packets of health information connect properly with the details of a known patient in the hospital's database. Storing patient information like ID, name, date of birth and medical record number is the responsibility of the HospitalPatient class. The class represents all patients who are present in the healthcare system.

Once incoming data arrives, the PatientIdentifier class checks its ID or name against the corresponding HospitalPatient object fields. There is a method match() that determines and explains if the data matches.

On top of the process sits the IdentityManager. It keeps a link to the main list of patients (hospitalDb) and assigns a PatientIdentifier for each patient identity verification through the verifyIdentity() method. If we encounter data that does not clearly fit the patient, because of formatting issues, incomplete information or duplicates, then resolveEdgeCase() is used. Because of this layered method, real-time healthcare systems are reliable and precise, as accurate identification matters greatly for safety and legal reasons.

Fourth:

The Avro ingestion architecture receives data from many sources by building around the DataListener interface. An interface has been introduced with a method called listen() which each data source should follow. It is made more useful by specific classes called TCPDataListener, WebSocketDataListener and FileDataListener. For every subclass, the input type is listened to and raw data strings are retrieved, each time.

Raw data goes straight to the DataParser class when it is received. The job of DataParser is to change unorganized strings into ParsedData objects that are easier to organize. The data format is always consistent regardless of the way the input was created.

The DataSourceAdapter carries data between the way the API is parsed and the storage where it is saved. It takes the output of parsing and sends it to DataStorage, so that concerns are clearly divided. This design ensures the system can be scaled up by using additional input sources while also improving both its testability and reliability.

When data acquisition, parsing and storing are separated, the system remains effective, adjustable and manageable, all key points for healthcare data that needs to stay accurate and immediate.