The Alert Generation System helps detect abnormal heart activity of specified patients and triggers the specific alarm. this system includes three main parts: AlertGenerator, Alert, AlertManager.

The AlertGenerator class checks patient's data to see if anything looks extreme based on limits that were set beforehand. If something does not look write, it triggers an alert. It doesn't know about how alerts are sent, it primarily focuses on checking if something is wrong. This is a great use of Separation of Concern, leading to code being more clean and manageable.

The AlertManager class acts as the main warning router. When it receives an alert from the AlertGenerator, it sends it to a local alert queue and then sends that to all of the subscribers.

The Alert class has a role of a simple object that carries on patientId, condition and timestamp parameters. It is still extremely useful, as it basically lets us know what specifically triggered which patient and which time this was recorded.

This system also has 2 interfaces, PatientDataGenerator and OutputStrategy that essentially help AlertGenerator. PatientDataGenerator povides all the needed information about each patient, and outputstrategy provides the method (as a strategy) for retrieving all that useful information.

Overall the system is clean and follows all the SOLID principles, each class is possible to modify and extend without having to create errors/bugs for other classes since the system is loosely coupled.

The essential goal of Data Storage System is to securely and in organize-able manner store all the essential patient information . This system has 3 main classes which are, PatientData, DataRetriever and DataStorage

For clear assortment each patient has their own unique patientId, the system offers an entire PatientData class that keeps a list of all the patients and their information such as their recordType, measurementType and the timeStamp, overall all the essential information.

This system uses Repository pattern, so it allows querying without having to expose the entire logic behind the storage, the DataRetriever class then provides a very simplified access to the patients' records. It uses DataStorage and has methods such as getAllRecords and getLatestRecord, they also promotes great software design like separation of concerns and encapsulation, encouraging our system to be way more simple and manageable.

The system also supports external data input through a DataReader interface. The CSVReader class implements this interface, and uses the method readData to be able to go through the entire text file. This design makes it easy to extend the classes within the system with other data input types in the future if that will be needed.

Overall, the design ensures secure and structured data handling, it also supports modular extension, and maintains clean interfaces between components.

The Patient Identification System makes sure that data is correctly matched to the patients in the hospital's database. This is used to ensure correct alert triggers, rigth treatment and no mix-up of information. This system plays a crucial role as it is extremely essential to maintain organized environment in the hospital.

The main class of this system is PatientIdentifier, its main task is to match patients with their information from the database provided. It does it by storing an entire map of patient IDs to the HospitalPatient instances and uses the method match() to provide the appropriate information. By using this logic, the system becomes way easier to test and keeps an overall modular look.

For easier access to all the information each patient is represented with HospitalPatient and stores all the needed information such as timstamps, age, records etc. .

The class IdentityManager uses PatientIdentifier to provide a high level method "resolve()" so external systems are also able to use it. Using method handleMisssing() it also handles abnormalities.

An interface PatientDatabase is also included in order to encourage loose coupling and strong cohesion. This interface could later be implemented by database connectors or services that fetch patient data from external systems.

Overall this system seems to follow all the principles and norms in order to pass as a greatly designed and structured system.

The Data Access Layer is designed to flexibly connect the external data sources to the internal components of the Cardiovascular Health Monitoring System (CHMS).

As on of the main parts of the system we have DataListener interface, it defines only one method which is listen() and it starts the data retrieval. Three subclasses: which are,TCPDataListener, WebSocketDataListener, and FileDataListener implement this mthod from the interface. Each subclass is designed to handle a specific source type and can be easily extended if needed, this provides extremely strong modularity and scalability, making sure our design is good.

We also have a DataParser class which is supposed to transform raw available input data into useful objects. It accepts raw string input(CSV) and then converts it into structured objects like for example PatientRecord. This is done to make sure external systems such as AlertGenerators get to get a clean and organized data.

The DataSourceAdapter class is responsible to coordinate the entire process. Once it is started, it listens for any incoming data, then parses it, and eventually passes the result to the storage.

Overall, I believe this design to follow a very strong software design principles, as it includes strategies like separation of concerns, interface-driven development, and extensibility. It allows for future data source types to be added without causing any errors or bugs to already existing system.