Assignment 3 Design Rationale (Yet Khai Bryan Wong)

In this design, I kept the **active model-view-controller design pattern** as it supported the requirements of observing different types of monitors. On the flipside, I removed the abstract factory design pattern as it was unnecessary since no groups of related objects were being instantiated.

A new **design pattern** I implemented was the **iterator pattern** to iterate over a map of instances of the class 'Biometric'. This improves the extensibility of the system by enabling future operations to instantiate multiple iterators over an aggregate of biometric observations and allows biometrics to be removed as they are being iterated over if required. I chose to not implement the iterator pattern over any other aggregate structure in the system as it is unlikely that they would need to utilise these operations. Therefore, traversing these other aggregate structures without an iterator, and simply using a for-each loop is currently sufficient. The disadvantage of the pattern implemented in this system is that it does not entirely hide the implementation of the aggregate structure. This is because maps return a key-value data pair and so their data types need to be known to be iterated over.

The first **design principle** I utilised in the new system design was **Liskov's Substitutability Principle**. This is shown by the abstract 'MonitorTasksManager' and 'MonitorSelector' classes. The benefit is that new types of monitors can be added onto the system simply by subclassing them, and their methods can be executed by existing classes while preserving system correctness.

The second **design principle** I utilised was the **Dependency Inversion Principle**. This is shown by the abstract classes 'MonitorTasks', 'BiometricTextMonitor' and 'BiometricGraphMonitor'. The benefit is that it provides hinge-points in the design so new subclasses can be added without affecting dependencies on the abstract classes.

The first **package level design principle** I used was the **Common Reuse Principle**. Classes were packaged together based on the likelihood that they would be reused together. The benefit of this is that each package has greater cohesion. However, this also grew the system's complexity as there are now many small packages.

The second **package level design principle** I used was **the Acyclic Dependencies Principle**. None of the packages create a cyclic dependency which makes it less likely for the rest of the system to break if there is an issue with a single package. This is supported by the use of the MVC architecture which ensures that the model component never depends on the view or controller components.

The first **refactoring method** I applied was the **rename method**. For example, the name of the 'getName' method in the Practitioner class did not previously demonstrate that it retrieves a practitioner's full name. Therefore, I refactored the method name to 'getFullName' to better reflect its purpose. Throughout development, inconsistencies between the use of 'bloodPres' and 'blood' to refer to blood pressure was also standardised to 'blood'. The benefit is that the code is now more readable and easier to maintain.

The second **refactoring method** I applied was the **move method**. One example of a major issue was that classes were accessing the ObserveStorage class through the DashboardModel class via delegation methods. The DashboardModel class did not use any of these methods at all. Therefore, I removed these unnecessary methods from DashboardModel, and created methods in the classes using the ObserveStorage class to access the ObserveStorage class directly.

The third **refactoring method** I applied was the **self-encapsulate field method** for every class with private attributes to improve system security. The downside is that it involved creating many getter and setter methods which increased the lines of code.

The fourth **refactoring metho**d I also used was the **inlining method** to reduce the number of unnecessary lines of codes such as those in return statements.

FIT3077 - Assignment 3 Class Diagram

Yet Khai Bryan Wong
28762797