Design Rationale

Design Principles

1. The Open Closed Principle - states that software entities such as classes, modules, functions, should be open for extension but closed for modification [1].
    a. In the task2.application package, the MVC model applies this principle. ObservationSubject is an abstract class with processObservation and update abstract methods. Two classes, BloodPressureProcessor and CholesterolProcessor extends the ObservationSubject class. This shows that instead of modifying an existing class, we create another class that extends the ObservationSubject class, this allows us to have classes of different functionality without affecting the existing codes. This allows the code to have maintainability as modification of either CholesterolProcessor or BloodPressureProcessor class will not affect other classes, hence its suitable for debugging.

2. The Common Closure Principle - states that classes that are likely to change together for the same reasons should be grouped together [2].
    a. In this case, any changes to requirements can always trigger certain changes to the component, hence making it harder to change as re-validation will be required for the new changes. Hence, it is better to group any classes that are related together. For instance, we had all controller classes grouped in a Controller package. Initially, there is only the defaultview, which is the table that shows all of the patient's cholesterol and blood pressure view only, but with changes to the requirement (practitioners can choose whether to only view data for blood pressure only, cholesterol only or both), all similar classes (AddPatientController,DefaultViewController,BloodPressureOnlyViewController, CholesterolOnlyViewController, RemovePatientController, etc) are grouped together in the Controller package. Such changes only affect the classes in that package, making updating easier as well as its maintainability. It seems clear that the changes focused into a single package rather than have to dig through a whole bunch of packages and change them all.

Refactoring techniques

1. Rename method - The name of a method does not reveal its purpose [3]
    a. As a new observation had to be added (BloodPressureProcessor), ObservationProcessor was renamed to CholesterolProcessor to make it more clear and specific
2. Extract method - Having a code fragment that can be moved together [4]
    a. When adding the feature to only view blood pressure, the table view had to be reconstructed to show only blood pressure values. The code to add buttons for viewing more of the patients' details were duplicated. Hence, this code was moved to a separate new method and the old code was replaced with a call to the method.
    b. The same situation occurred when implementing the add patients feature to the graph view, which resulted in the addDataToGraph method in AddPatientController.
    c. Therefore, the number of lines of code and code duplication are reduced
3. Extract class - When one class does the work of two, awkwardness results [5]
    a. When implementing the graph view for the high systolic BP monitor, the functionality was initially in the class ViewHighSystolicBPP to fit with its responsibility for being a monitor for high systolic BP
    b. However, the complexity of the code increased as implementation went along. Hence the class ViewHighSystolicBPPGraphs was created to place the fields and methods responsible for the relevant functionality.

References

[1] Despoudis, F., 2017. *Understanding SOLID Principles: Open Closed Principle*. [online] Medium. Available at: <https://codeburst.io/understanding-solid-principles-open-closed-principle-e2b588b6491f>

[2] Bäckhage, E., 2019. *The Common Closure Principle – Ericbackhage.NET*. [online] Ericbackhage.net. Available at: <https://ericbackhage.net/clean-code/the-common-closure-principle/#:~:text=The%20Common%20Closure%20Principle%20(CCP,have%20multiple%20reasons%20to%20change.>

[3] *Week 9 - Refactoring.* [online]. Available at <https://lms.monash.edu/pluginfile.php/10720043/mod_resource/content/2/Week%209%20-%20Refactoring.pdf>

[4] Shvets, A., 2020. *Extract Method*. [online] https://refactoring.guru/. Available at <https://refactoring.guru/extract-method>

[5] Shvets, A., 2020. Extract Class. [online]. https://refactoring.guru/. Available at <https://refactoring.guru/extract-class>