Design Rationale

Team XD:
Loi Chai Lam
Aik Han Ng

Our team had implemented the system with some design patterns and design principles, included Model View controller (MVC), Observer, Dependency Inversion Principle (DIP), Abstract Server, Role principle and Open Closed Principle (OCP).

For the Model we used, we implemented Role principle for the Patient and Practitioner class. The reason is it provides a more flexible solution as one user might have multiple Roles, for example, the Practitioner can also be a Patient in the system. Hence, it becomes more extensible and relatable to the real-world situation (Principles of Object-Oriented Analysis and Design, Week 3 Lecture).

Moreover, we implemented Open Closed Principle and Dependency Inversion Principle in our system. We use abstraction in our system so that all the concrete implementations depend on the interface or abstract class, rather than concrete class, because abstract things change infrequently (Principles of Object-Oriented Design 2, Week 4 Lecture). For example the AbstractDataRetrieval class and AbstractTableModel class. It provides hinge point in our design, where we can easily extend and add functionality to it (Principles of Object-Oriented Design 2, Week 4 Lecture).

As the assignment specification mentioned that we might be required to support an additional web service that supplies the same sort of data, but not necessarily in the same way. Therefore, we used Abstract Server, the AbstractDataRetrieval class as an interface, in the server implementation. The controller depends on the server interface instead of the concrete implementation of the server. In other words, any changes in the concrete server will not propagate to the client and the client can easily use other servers with similar properties. It also provides a hinge point for the server implementation. It can be extended easily to support other similar server implementation without changing the controller (Design Principles and Design Patterns, Week 5 Lecture).

We decided to implement the Model View Controller pattern in the system because it is well-suited for database-driven application (The Model-View-Controller Architectural Pattern, Week 7 Lecture) which is what our application is. We implemented all the logic of the system in Controller class. Our Controller adds action listener to the View’s user interface, while our DataRetrieval resembles the Model, following this handy tutorial by Sylvain Saurel (Saurel, 2016).

The Model is independent of the View and the Controller in our Model View Controller pattern, so that the model can be built and tested independently of the View and Controller.

It provides ease of change for the system (The Model-View-Controller Architectural Pattern, Week 7 Lecture). This is because the UI requirements tend to change more frequently than business logic. Since the Model does not depend on the View, so adding new Views does not affect the Model. The scope of change is restricted to the View only. As for now, we only have one View for our system. However, we can add more Views with the same Model data and show it at the same time in the future, as the Model View controller pattern supports for multiple Views. This provides the extensibility of the View for our system.

However, the Model View Controller pattern introduces extra levels of indirection, and therefore increase the complexity of our system (The Model-View-Controller Architectural Pattern, Week 7 Lecture).

To adhere to the assignment specifications, which states that we should limit network traffic by updating at most once per hour, we decided to implement Observer principle, which updates the data stored every hour, instead of updating it once the object it is listening to changes state. In theory, when a new data being pushed to the server, a method should be called to update all observers (Principles of Object-Oriented Analysis and Design, Week 3 Lecture), but this will increase the network traffic. Thus, the duty of our Observer is to get the refreshed/updated state of the subject every hour. Besides that, we also integrated the Observer with our Model View Controller pattern, which the Controller and View extend from the Observer class. With this implementation, when the Model changes, it will automatically updates the Views and Controller (The Model-View-Controller Architectural Pattern, Week 7 Lecture).

References

[Pag2000] Page-Jones, Meilir, Fundamentals of Object-Oriented Design in UML, Addison-Wesley, 2000 (Ch. 8, 9)
[AIS1977] Alexander, C., Ishikawa, S. and Silverstein, M., A Pattern Language: Towns, Buildings, Construction, Oxford University Press, 1977.
[GHJ1995] Gamma, Erich; Helm, Richard; Johnson, Ralph; and Vlissides, John, Design Patterns: Elements of Reusable Object-Oriented Software, Addison-Wesley, 1995 (Chs. 1, 3, 4, 5).
[Mar2000] Robert C. Martin, Design Principles and Design Patterns, 2000. Available on-line from Object Mentor: http://www.objectmentor.com/resources/articles/Principles_and_Patterns.pdf

[Pag1988] Page-Jones, Meilir, The Practical Guide to Structured Systems Design(2ndEd.), Prentice-Hall, Englewood Cliffs, N.J., 1988. (Chapters 3, 5, 6)
http://www.waysys.com/ws_content_bl_pgssd_ch06.html
[GHJ1995] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides, “Design Patterns: Elements of Reusable Object-Oriented Software”, Addison-Wesley, 1995 (Chs. 1, 3, 4, 5).
[Ree1979] Trygve Reenskaug, “THING-MODEL-VIEW-EDITOR an Example from a planningsystem”, Xerox PARC technical note, May 1979.
http://folk.uio.no/trygver/themes/mvc/mvc-index.html
[Bur1992] Steve Burbeck, “Applications Programming in Smalltalk-80™: How to use Model-View-Controller (MVC)”, University of Illinois in Urbana-Champaign (UIUC) Smalltalk Archive, 1992. http://st-www.cs.uiuc.edu/users/smarch/st-docs/mvc.html
[MS2008] Microsoft Corporation, “Model-View-Controller”, Microsoft Patterns & Practices Developer Center, 2008
http://msdn.microsoft.com/en-us/library/ms978748.aspx
Sylvain Saurel, “Learn to make a MVC application with Swing and Java 8”, Medium, 2016
https://medium.com/@ssaurel/learn-to-make-a-mvc-application-with-swing-and-java-8-3cd24cf7cb10