**RETROSPECTIVE: DESIGN DECISIONS THE HAVE NOT BEEN SUCCESSFUL & ROOM FOR IMPROVEMENT**

Our original design plan involved an abstract class, *Merchandise*, and a subclass, *Medication*, that extended this class. We originally planned on applying the OCP design principle, to allow further expansions in the pharmacy in the future with regards to what they sell. For instance, in the future, if the pharmacy plans on selling bandages, wheelchairs, or crutches, they would be able to add these as subclasses that *extend* the Merchandise class. However, in the beginning of the semester when we were more worried about implementing our big stories, this concept of Merchandise vs Medication got very confusing, especially because they have really similar names. So, we decided to drop the idea of including the OCP design principle and focus more on the implementation of our big stories. Hence, we find this design decision not as successful as we hoped for and something we could improve.

* To improve this in release 2, since we have already implemented the most important design features stories, we can try modifying the code in such a way to incorporate the OCP principle again. This would likely involve renaming a few existing classes & adding an abstract superclass.
* To evaluate this improvement’s success, we can test if by the end of Release 2, it becomes easy to add new classes that inherit the ‘Merchandise’ superclass to mimic non-Medication objects being added to the pharmacy. We can also test if non-medication merchandise can be easily added, removed, increased/decreased in quantity like medication is currently able to do.

Our original design plan involved having a User superclass, and Owner, Pharmacist and Patient as subclasses (they inherit User class) since they are all users of the system. We also originally planned on each user containing methods for the respective features/capabilities they have access to in our system. However, due to the owner and pharmacist having many overlapping features, this design caused a lot of code duplication. And since one of our main aims in Iteration 3 was to reduce duplicated code during refactoring (since it is a code smell), we deleted duplicated methods and moved the methods into one class (ListOfUsers/ListOfOrders/User). However, this design choice resulted in Patient.java, Owner.java and Pharmacist.java turning into data classes. Other classes like Merchandise, Order and Prescription are also data classes. The reason we need/use these classes is because they act as models/encapsulated objects for when we retrieve information from the database.

* As of now, we are not sure how to improve this in release 2, since we might have to change the overall design of the entire program, which will require a lot of thought and work.
* To evaluate this improvement’s success, we can count how many data classes remain at the end of Release 2.

One more thing that wasn’t as successful as we thought is the use of a large if/else if/else block in InitialScreenPanelAdmin.java. This large block is the result of refactoring for large classes and large methods, which is an implicit result of the Admin’s having a lot of capabilities/features. To fix this issue, it would require a lot of additional work and changes that we do not have time for in this iteration.

* To improve this in release 2, we can try categorizing the admin’s features/buttons into categories (such as orders, medications, patients) and create new classes for each of these.
* To evaluate this improvement’s success, at the end of Release 2, we can see if the large if/else if/else block in InitialScreenPanelAdmin.java still exists or it has been spread to different classes.

The last minor thing that wasn’t as successful as we thought is duplicated code. In the front end, since we are checking if input fields are empty or are invalid in almost all methods for when a button is clicked, it becomes sort of repetitive for each button. We didn’t have time to fix this because it would cause a chain of changes that would be needed to be altered as well.

* To improve this in release 2, we can try to include a one-stop new method that does all the input validation, and this method would be invoked in all button-clicking methods.
* To evaluate this improvement’s success, at the end of Release 2, we can see if a new method has been created that is invoked to check/verify the input fields for all buttons before sending the input to the logic layer.