CSCC01 Team Code Review

Team Name: Ctrl-Alt-Elite
Team Number: 11
Date: November 26th, 2018

Table of Contents

Table of Contents **2**
- Code Review Videos 3
- Code Review (Sprint 3 & 4)
  - Jun 4
  - Leo 4
  - Vishwa 4
  - Tayyab 5
  - Angela 5
- Code Review (Sprint 5 & 6)
  - Jun 6
  - Leo 6
  - Vishwa 6
  - Tayyab 7
  - Angela 7

Code Review Videos

Code Review - Debriefing Meeting (Sprint 3&4)
https://www.youtube.com/watch?v=k5RTcFJyUJk

Code Review - Group Review Session (Sprint 3&4)
https://www.youtube.com/watch?v=_ej1VAiWz4A

Code Review (Sprint 3 & 4)

Jun

For this code review, I reviewed the Command classes under model.database.api. I didn't have good design in the beginning, just put everything that communicates with MySQL database server in one single class QueryOnDatabase. However, it got messy very soon. I decide to using the abstract class Command that contains a abstract function handle() and let other specific Command classes (CreateCommand, SelectCommand etc.) to extend the Command class.

In this way, not only the code's readability is improved, but also it follows the single responsibility principle.

Leo

For this code review, I review the Uploading CSV Files feature on the U3-T2-Update-CreateTemplateModelImpl.createUsingFile branch, which I implemented. At first, I assumed that this would be a simple task to just update the existing function calls, which created a ExcelFile object to parse the template information from the Excel files, to use the CSVFile object and functions. However, I then realized, with input from Jun and Vishwa, that I would instead create an interface where both ExcelFIle and CSVFIle implement. Therefore, I created an interface called TemplateFileInterface with all same functions with respect to both file formats, where I refactored the implementations to TemplateFileExcelImpl and TemplateFileCsvImpl.

This improved the design of the code because it removed the need for large blocks of code after checking the type of file the user has uploaded and call their specific functions. Instead, it just checks initially the file type to instantiate either a TemplateFileExcelImpl or TemplateFileCsvImpl and stores it in a TemplateFileInterface. This allowed for the same functions to be called, which makes it easier to extend if needed in the future.

Vishwa

For this code review, I review the code for CreateTemplate user story which was initially thought to have a model interface like the presenter that helps communicate between presenter and the backend. After implementing the interface, I learned it was not going to be very good from a design point of view because having an interface for the model for each user story means duplicated code where the same code will go in different model interfaces if something is needed from the database like fetching template names.

Therefore, after discussing it with my team members, I decided to avoid the interfaces for the model layer and instead have Usecases that can solve the purpose - getting data to and from the database using the entities. An advantage of usecases was that they did not know anything about the external agencies, view or framework. All they want is data into so they can start working. Moreover, we were able to reuse the usecases where needed without much duplicated code. Therefore, I believe thinking about a particular implementation from a design perspective right after developing really helps reveal some of the issues that can come up later in the project due to changes which are inevitable.

Tayyab

For this code review, I chose to look at the code for the User Interface classes that I implemented. After reviewing the code, I believe that it is best if I break down the giant constructors for the UI classes because as of right now the constructors for them are way too long. Breaking down the code helps improve design because it increases modularity. I noticed that many calls in the constructors are repeated in most, if not all the other UI classes so they would be better suited to a class such as UIHelpers.

In addition, I noticed that there are many branches for features that have been already been developed and shipped a long time ago, so they should be deleted.

Angela

For this specific code review, I have chosen the code implemented in my branch U7-ReportsRunningSQL. It was decided for the beginning that we were going to follow the MVP pattern. Every time a new functionality needs to be implemented, we try to keep them consistent with already existent code.

During the first read of my code, I made some style improvements and spotted a couple of duplication, for which I implemented new methods and integrate them accordingly. I reviewed the unit tests, to make sure all the main capabilities were tested and I was not missing anything important.

Made sure my code was well commented, specially each method and class to understand the behaviour of the code. Reviews and spotted any redundancies in my code and thought about any other way of coding the same functionality.
Scanned the code one more time to reaffirm that none of the already implemented methods and classes could be reused as part of my code.

Code Review (Sprint 5 & 6)

Jun

I chose to do code review for the UI classes, after reading the UI classes. I realize that there are a lot of duplicated code that we can simplify. For example, in view.ui.register, I refactor out the duplicate code out in registerOfficer and registerTeqStaff by have them extend the abstract class registerNewUser. By doing this, not only improve the readability of code, but also reduce the redundant code. Moreover, when you have more type of user, you can just create a new class for new type of user without modifying any of existing code make code easy to extends.

Also, when I review the code in Login class, I moved the main function out to make a new class Run. By doing this, we follow the single responsibility principle.

Leo

When it comes to review code for this sprint, I had to choose a less efficient solution to keep the implementation simple. This applied to the Edit and Update from Table task which required updating the database for each individual attribute of each row. Originally, I wanted to store all of the changes the user had done to a row and update it all at the same time, but this required having a lot of nested loops with complicated conditions on when to actually update the database; however, I chose to update each attribute in each row individually as that required much less logic. The reason I originally planned to do it all at once is because that only required fetching all the rows and columns from the database once. However, with the updating of each value individually would required fetching from the database every single iteration. At the end of the day, having a less than preferred efficiency wise solution makes the code much more readable.

Vishwa

During this sprint, I was developing the code to handle exceptions that arise when uploading to templates using user-uploaded files. I had two different exception classes that inherit from the same class - InsertException. Initially, I had them both differently implemented despite of similar requirements of the exceptions in the program. I had checks for both exceptions in similar places and was completely unaware of it during the development.

After later inspection of the code and feedback from my team members, I figured it would be better to develop methods that can be used regardless of the type of the exception. For example, I added the getMessage() method that just returns me the cause of the error. Therefore, I made the exceptions inherit from InsertException which was more of a generic exception of the other two and override methods to fulfill the requirements for each exception class. This helped me achieve polymorphism and design the interface with a much more generic class. Moreover, this helped me to debug issues much easily as I could handle both types of InsertExceptions using the InsertException class.

Tayyab

In this code review, I focused specifically on the user story 10 - resolving conflicts. While developing this user story, I made a giant action listener for when the resolve button in the UI is clicked (i.e when the user wants to resolve a certain conflict that they've selected) because a lot of code was required. This caused a lot of headaches because the

After the lecture about code smells, I realized that it was better to move this giant action listener to a different class altogether because it's more than 100 lines of code on its own. Doing this made the Resolve Conflicts UI class much more readable and easy to work with because the class is smaller.

Angela

For this specific code review, I have chosen the code implemented in my branch U4-Search account. Feature to be able to search in the database for an specific account and show its characteristics.

During the first read of my code, I tried to reduce the amount of warnings and replace some of my code with simpler pieces to make it more readable. I reviewed the unit tests to make sure I tested the most important methods.

I decided to delete some pieces of my code when I realized that some useful classes and methods were implemented in other user stories and would be useful for me to use them instead of creating another instance

In summary, this time I had less warnings and style changes to make as I had paid more attention while coding at the beginning and remembered the fixes and changes I made during my first code review.