Code Review Summary

TEAM

BRAW

Bekzod Tursunov **|** **R**adu Laudat **|** **A**lvin Tang **|** **W**esley Ma

"Infinite Possibilities with BRAW"

Table of Contents

Table of Contents

Code Review Guidelines

Code Review Summary
- Alvin’s Review
- Wesley’s Review
- Bekzod’s Review
- Radu‘s Review

Code Review Guidelines

Developers should look at these:

1. How the code is documented
2. Naming conventions
3. Code readability
4. Follow SOLID principles
5. Could a design pattern be used, and if it was used, was it implemented correctly?
6. Unit tests for a majority of functions
7. Remove unneeded code

Code Review Summary

Alvin's Review

- Looking at the pull request "Manual conflicts gui #22", for the constructor, *ManualConflictResolvePanel*, its name is actually inaccurate as the class extends a JFrame, not a JPanel. However, in general, most naming conventions (guideline 2) are followed.
- From the code smells lecture, long methods and large classes are bad, as it makes the code hard to read (guideline 3). In general, the repository holds some methods and classes that are very long. An example such as the one in the pull request "ManualConflictsGUI -> master #21", the *toJSON* method in the *TrendChartScheme* class has too many lines. It could have been broken down by using private helper methods (i.e. *getEntryCountForCurrentYear*). Another example is in Radu's pull request "Manual conflicts gui #22", where the constructor *ManualConflictResolvePanel* was very long, and hard to read. Yet extending JFrame to make GUIs would usually have large code blocks involved so I wouldn't see anyway around breaking the code up into methods. However, the code could still be broken up into groups (i.e. group for labels, fields differently for easy readability) in the same method.
- Looking at the pull request "Manual conflicts gui #22", the general code doesn't seem to follow SOLID design (guideline 4). The way Radu wrote his code to handle conflicts made it such that it was hard to extend, if more conflicts were to be added. For example, he uses a method *checkForAutomaticConflicts* which calls different methods to check specific types of automatic conflicts. Instead, what he should have done was to make an interface such as *AutomaticConflict*, then have specific classes for each automatic conflict that inherits from that interface. Then use a design pattern such as the Observer pattern to append all the conflicts, and then it will be easier to generate an action to check all conflicts by using one method in the Observable (guideline 5).
- Unneeded code was removed by Bekzod in the pull request "D5 bug fixes #23". As well, unneeded code was usually caught in pull request reviews. For example, Bekzod recommended that I use String.join to list years with commas (in the class *GenerateReports*) instead of creating a private method to do the same thing. Therefore, in general, guideline 7 was followed.

Wesley's Review

- Lots of copy-pasted code in the system for conflict checking. This is from the code within the ManualConflictGUI. Specifically, the ExitButtonListener class reimplements much of the code already present. This made it difficult to debug.
- The GUI, in general, should adhere to MVC better. We have lots of logic within the GUI classes. Additionally, these classes can be better if we used inheritance for the window for adding charts.
- Since the AddChartWindow can add multiple types of charts now, need to refactor so it follows SOLID and we don't need an if statement.
- Lots of unneeded comments, especially in the XLSXReader and ChartScheme class
- However, we should still work on having more Javadoc to at least explain the logic or results of more complex methods
- Many methods in our code have methods that are over 10 lines long (eg. GUI classes, ChartScheme.toJson()), though not sure if we have the time to completely refactor. This affects code readability.
- Although most of the functional code is covered, still need tests for the specific ChartScheme types, and conflict check methods
- Also need to cover conflicIt checking, query, and template selection in the acceptance test document.
- Another issue is within the ServiceReceivedTemplate; This class should not have the GUI itself, the functionality to add the template, as well as the entire template data in a single class. This instead should have a generic template adder window class, a separate listener, and the template data in its own class following the template design pattern.

Although we won't be able to fix all the issues, the D5BugFixes branch should address many issues related to code design, and hopefully, we update our tests by Monday.

Bekzod's Review

Code Review Pull Request #27: Table Bugfix

1) How the code is documented
   a) The code is simple enough to not need comments, but complex code requires more in depth comments
2) Naming conventions
   a) Encouraged to follow camelCase patterns, with clear and concise names for every specific class/variable/method
   b) Found these to be followed consistently
3) Code readability
   a) The code is encouraged to be more readable and if it is confusing, we gave constructive criticism to improve the wording, until it is easy to read.
   b) Given comments where the code could be improved.
4) Follow SOLID principles
   a) Our code is enforced to be Single Responsibility, Open/Close Responsibility, Liskov Substitution Principle, Interface Segregation and Dependency Injection.
   b) Found code to be following SOLID principles where required.
5) Could a design pattern be used, and if it was used, was it implemented correctly?
   a) Following design patterns are encouraged to write scalable and efficient code,
      i) Examples are: Microservices, Builder, Model View Controller, etc.
   b) Found the code to not require any design patterns for this merge request
6) Unit tests for a majority of functions
   a) We have been writing unit tests for our major functions, and use travis ci to ensure CI across branches
7) Remove unneeded code
   a) Encouraged to remove unneeded comments and old code if seen as navigating through different files.

Radu‘s Review

Upon inspecting a few commits made in the TemplatesInGUI branch at varying periods in time:

- Some variable or class names should be more specific to what they are corresponding to. There are a few examples of names in the code which don’t make intuitive sense as to what they are. You have to get an idea of what it means by analyzing the code itself

- Instance variables should always be private to a class, but in certain cases it can be seen that our design has forbidden us from doing so. Steps to improve could be adding methods to get data without compromising class and information integrity through the use of public variables.

-The idea of spaghetti code makes itself present in a few of the classes, a strong example of this would be a few characteristics in the incorporation of templates. Future sprints should aim to clean this up, but the overall java swing convention limits our ability to totally avoid this from occurring.

- Documentation should take a greater importance in all further commits. It seems as if we have ignored docstrings as well as a few comments per file in most of the commits during this project. More effort should be put into this in order to make understanding the code much easier for developers extending this project in the future.