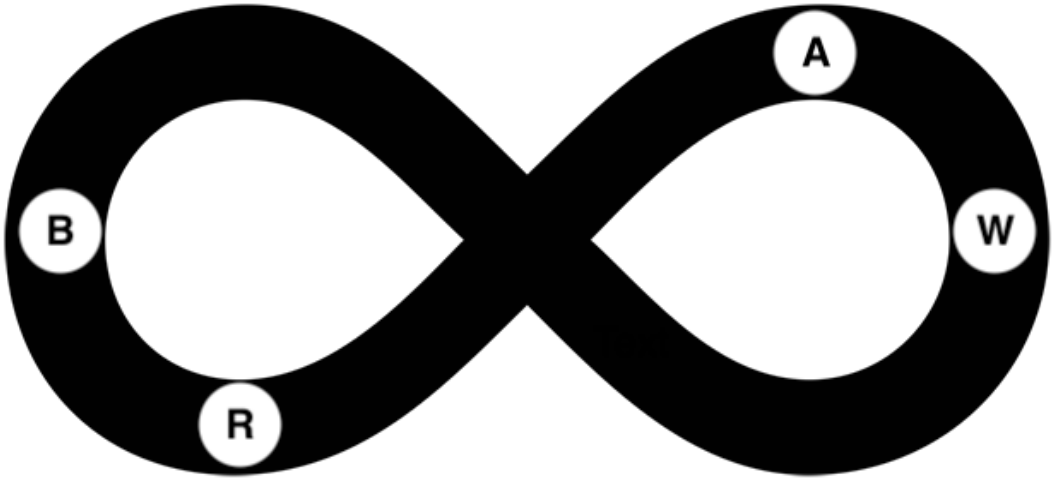
Code Review Summary

TEAM

BRAW

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



"Infinite Possibilities with BRAW"

Table of Contents

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 Wesley's code from the GraphCreation branch in pull request #5, I can see that he is not following the 6th principle about comments. In a file like DistributionChartScheme.java, I can see that he hasn't put any comments down which makes it very hard to understand the code and what it does. Luckily, he has followed the 2nd principle about naming conventions which gave me an idea of what the code did, which was a class of a scheme of a distribution chart. He has also followed the 5th principle of testing as I can see him testing the serialized JSON of a ChartScheme compared to what is expected. However, he could do more testing as it only relies on only two test files. For example, another unit test could include test files with different values in cells. Finally, I would like to add the fact that Wesley added some documentation in the README.md file about how to execute the code, but it should have been added by Bekzod earlier in the ChartJsapp branch pull request #4. As well, in general, the whole team should provide better documentation on how each class connects with each other, since no UML diagram was developed. Yet, as we are pressed for time, it is no wonder this guideline is not fully followed. In conclusion, Wesley, for the most part, follows the guidelines listed above, but there are some discrepancies as well which include the whole team.

Wesley's Review

Looking at Alvin's changes based off the GraphCreation branch, in the GraphReportsGUI branch, I can see that he has followed most if not all of the relevant guidelines. An improvement that can be made is to use the MVC (Model View Controller) design pattern with GenerateGraph GUI. The current code has the button functionality within the same class as the Java Swing window itself. By having the functionality in a separate class, it would also better adhere to the Single Responsibility Principle. As a comment about the team's code as a whole, we should be more vigilant in increasing code coverage within unit/integration tests, as well as adding acceptance test documentation.

Bekzod's Review

Code Review Pull Request #2: Column Aggregator merge

1. How the code is documented
    a. We found that generally 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
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. Early in the code, we were unsure if we will reuse the code, so this check was less enforced
    b. Later on (approximately during Deliverable 3), we have been strict on not including duplicate code, and not reinventing code/methods that have been already written for us (natively in java)
    c. If a piece of code can be improved, suggestions were given to the controversial code.

Radu‘s Review

Upon inspecting a few commits made in the chartjsapp branch at varying periods in time:
- Documentation can definitely be an asset worthy to the team. During introductory computer science courses, we are instructed that all code should be given a doc string as well as some comments explaining developed functionality in the code. We are lacking this at the moment, so this can definitely be something we can improve on.
- 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.
- Certain variable names should be more specific to what they are storing, some can be misinterpreted as holding something else.
-The idea of spaghetti code makes itself present in a few of the classes, a strong example of this would be a few characteristics of the GUI class. Future sprints should aim to clean this up, but the overall java swing convention limits our ability to totally avoid this from occurring.

Code Review Debriefing Meeting Video

View the video here: https://www.youtube.com/watch?v=GFLT08cjsqg