HOA 23b Assignment 2

1.

The worst code smell, that we immediately identified in our project, was Blob Code. Some classes were too long and had many methods and attributes, while some methods were long and had many if statements. In many cases, just by looking at the code, some of the if statements seemed pointless and easy to avoid or change. This led us to look further into the code using a metric tool.

We used the MetricsTree IntelliJ plugin for this assignment to confirm our suspicions. We ran the Project Metrics program and went through the metrics of all methods and classes. We identified metrics that were red or orange, according to the MetricsTree coloring scheme, and confirmed the indication by manually checking the method, and the value of the metric. Most of them were, indeed, tied to Blob Code.

Some of the worst metrics, and the thresholds which we used to identify them, were:

- Methods:
    - Lines Of Code > 30
    - McCabe Cyclomatic Complexity > 5
- Classes:
    - Weighted Methods Per Class > 30
    - Response For A Class > 60
    - Number Of Methods > 15

Of course, some classes naturally required to be longer than others. That is because they were more complex, encapsulated more related business logic or had to do with multiple endpoints which simply made no sense elsewhere. As a result, we felt that setting relative, rather than absolute, goals was more appropriate. Based on the scores, but also on the complexity of our methods and classes, we decided to aim for 25-50% improvement in each metric, which is generally achievable with a few class extractions and a couple method extractions.

If we had too large of a decrease after one class extraction, that would likely mean that the extracted class would be more likely to have bad metrics itself. As a result, we made sure that after each refactor operation the new class(es) would be underneath the set thresholds.

2.

We refactored the following methods and classes. Additionally listed are the refactoring operations used in the process, as well as the improvement seen in the metrics.

Methods:

VotingService.castElectionVote() - Method Extraction:

- Lines Of Code 41 → 20
- McCabe Cyclomatic Complexity 7 → 2

AssociationService.verifyCouncilMember() - Method Extraction:

- Lines Of Code 25 → 18
- McCabe Cyclomatic Complexity 6 → 4

AssociationService.verifyCandidate() - Method Extraction:

- Lines Of Code 42 → 30
- McCabe Cyclomatic Complexity 9 → 5

AssociationService.processElection() - Method Extraction:

- Lines Of Code 31 → 30
- McCabe Cyclomatic Complexity 6 → 4

AssociationService.joinAssociation() - Method Extraction:

- Number of Parameters 7 → 3
- McCabe Cyclomatic Complexity 4 → 3
- Lines Of Code 22 → 21

Classes:

VotingService - Class Extraction:

- Weighted Methods Per Class 67 → 56
- Response For A Class 121 → 71
- Number Of Attributes 9 → 4
- Number Of Methods 18 → 13

AssociationController - Class Extraction:

- Number Of Methods 17 → 13
- Response For A Class 73 → 57
- Weighted Methods Per Class 41 → 33

Association - Introduce Parameter Object:

- Weighted Methods Per Class 22 → 16
- Response For A Class 28 → 17
- Number Of Added Methods 20 → 14
- Number Of Operations 34 → 28

Membership - Introduce Parameter Object:

- Weighted Methods Per Class 24 → 13
- Response For A Class 28 → 17
- Number Of Attributes 12 → 8
- Number Of Added Methods 22 → 11
- Number Of Operations 36 → 25

Activity - Introduce Parameter Object:

- Weighted Methods Per Class 17 → 10
- Response For A Class 20 → 14
- Number Of Methods 17 → 10
- Number Of Accessor Methods 9 → 4
- Number Of Attributes 9 → 6