Continuous Integration Report

Cohort 1, Team 3

Ari Kikezos

Arya Enkhnasan

Ben Green

Calum Wright

Lilac Graham

Skylar Garrett

Tools Used

The coverage testing tool used was a tool called Jacoco, a Java coverage testing tool. When the project's tests run, the coverage testing software places a flag at each line, each decision branch, each function, and each file, and when the process ends generates a report identifying which flags were hit where. If every flag was hit and the program didn't fail, the coverage score is 100%. Jacoco then produces a report in a HTML tree.

Another version of such testing tools that was used at the start of the project was Intellij IDEA's "Run with Coverage", which performed the same function. This allowed developers to get a sense of their tests' progress before adding their tests to a pull request.

GitHub Actions Infrastructure

The intention of the GitHub Actions Infrastructure was to disallow direct modifications to main and to check any pull request made from a different branch in the repository. The two settings we set were pass percentage overall and pass percentage of newly added files. It is the convention that the pass percentage of the newly added files is higher than the other so that the trend of lines covered increases with increased software functionality.

The intended GitHub Actions Pipeline was as follows:

1. The team agrees to not push directly to the main branch without good reason and team agreement, allowing the main branch to remain flexible but controlled. Despite this, the GitHub Actions Pipeline would still run on any direct push, just to be safe.
2. Any pull request from another branch in the repository into the main branch would be subject to the same testing process.
3. The pipeline would then prepare a Ubuntu Linux testing environment. This was chosen mostly arbitrarily due to the fact that our project runs on Java, a sandboxed programming language, whose main design is focused at being platform independent. Any discrepancies in implementation were sorted before this step.
4. The pipeline then prepares a Java 17 runtime environment with which to run the test suite.
5. Using this, it then runs the Jacoco coverage testing process.
6. Finally, ensuring that the report summarises that the minimum coverage overall and the minimum coverage of changed files were within acceptable limits (40% and 60% covered, respectively), the build would not fail.
7. A copy of the coverage report would be added to the pull request.

This would have allowed developers to get a good sense of how easy their programming approach was to test on each major addition to the program.

Unfortunately, due to various complicating factors and miscommunications, CI was implemented incorrectly (specifically, builds would succeed when they shouldn't have) and the mistake was caught too late. The pipeline was instead performing steps 1 to 4 in the above list and then failing quietly. The one pitfall this did protect against was pushing actually non-functional code to the main branch, avoiding some rebasing scenarios.

Testing that was performed successfully and by other means is highlighted in the testing document, Test2.