For the final deliverable, I decided to write and test (via TDD) a simple Hangman program. The program should take in user input for a word to guess, clear the screen to prevent another player from seeing the word, and then take in input for either a letter guess or a guess of the entire word that the first player entered. If the letters or word guessed are incorrect, the Hangman will be drawn, adding a part for each incorrect guess until all six wrong guesses have been exhausted, which marks the end of the game and a loss for the user. However, if the user correctly guesses the word, they win the game and the game ends. By writing this program while using TDD, I was able to write methods that thoroughly check base, edge, and corner cases and deal with problematic input appropriately. I chose this project because I have never written a Hangman program before, and wanted to test out TDD to see how it would affect my coding.

TDD on Hangman

CS 1632 – FINAL DELIVERABLE

Therese Dachille

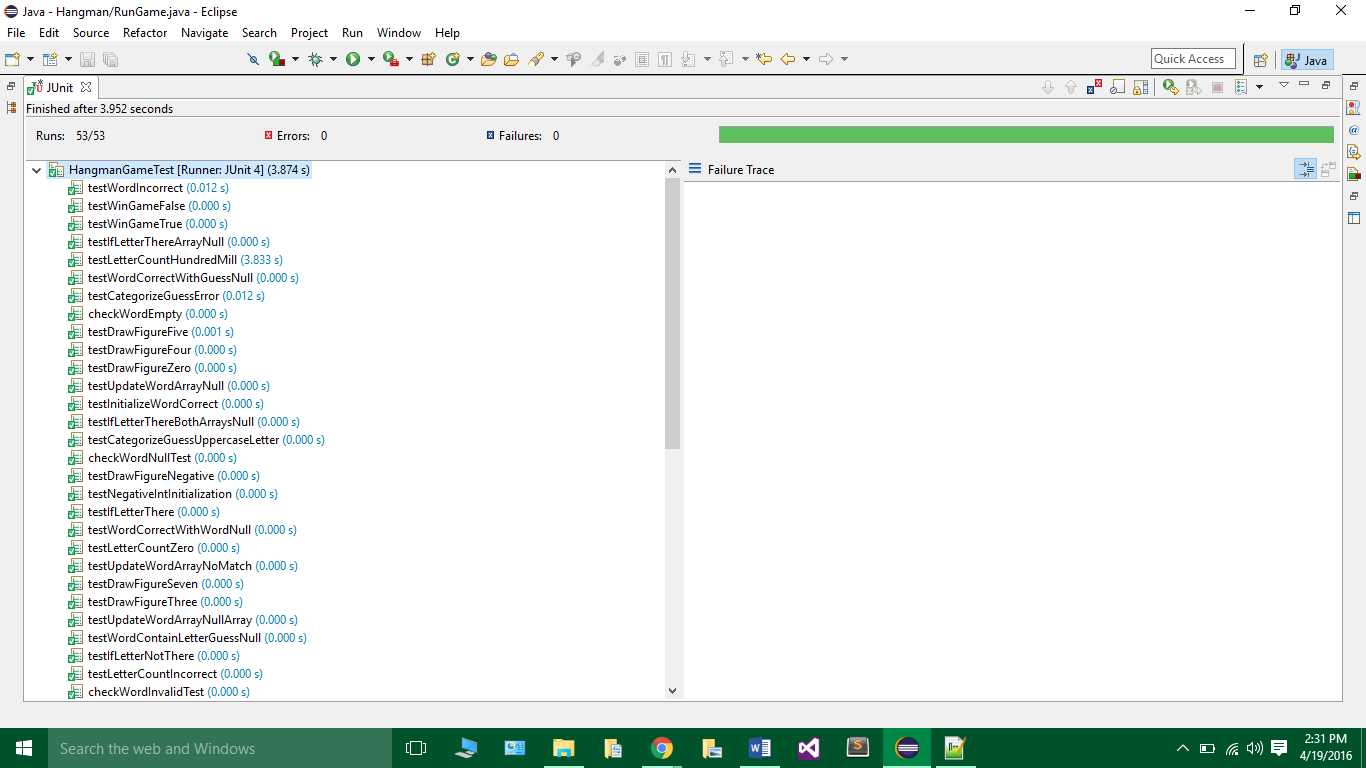
https://github.com/tcd12/CS1632FinalDeliverable

I implemented the program by creating a HangmanGame class that includes most of the functionality of the program and is made up entirely of non-void methods. I did this in light of doing TDD, since methods must be testable. Because of this, I was able to easily test this class with 99.8% code coverage. With a total of 53 tests, each method has around five tests checking its base, edge, and corner cases. I found that TDD made it easy to write a functioning program with minimal revision after running it for the first time, once my classes and tests were complete. Implementing TDD helped me be more thoughtful in my coding and encouraged me to write much more thorough code than I usually do.

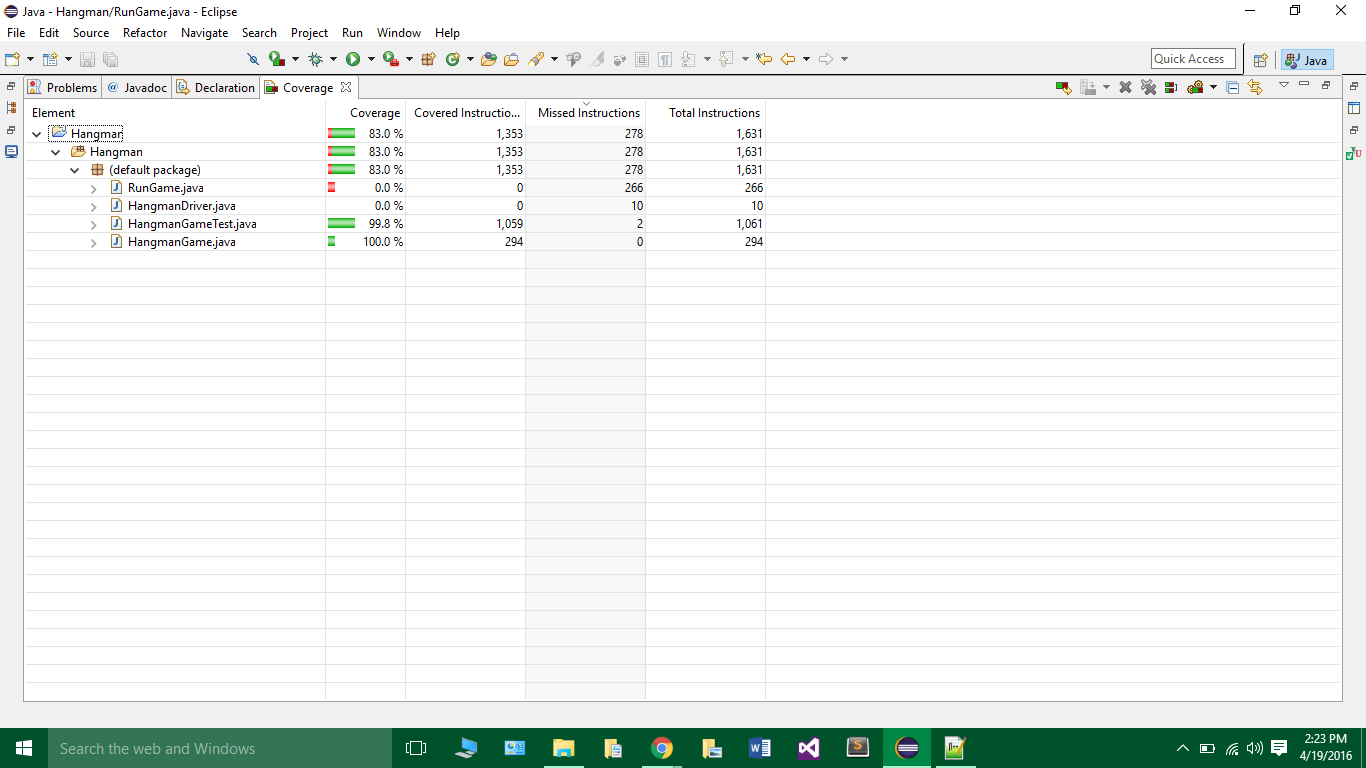
Some issues I faced involved making sure my methods were thoroughly-testable, and figuring out what to return from some methods. It also took some time to get used to coming up with cases to test, but it became much easier over time. At first, it felt strange testing seemingly-obvious cases, but then I realized that that is part of the main motivation of using TDD –to make sure even the most basic functions of methods work properly. Another issue I faced was with a test counting word length, where I would have liked to test Java’s Max Int, but could not due to memory limitations. However overall, I encountered little issues with writing tests for my code, as this is something we have done for multiple projects now, and I feel comfortable with TDD both conceptually and in practice.

To improve the quality of my Hangman program, I would want to use exploratory/black-box and manual testing to further check all base and edge cases, as these can never be overly-tested. I am sure that my program would benefit from more testing of this kind, to exploit any bugs that differ from the requirements of Hangman.

All in all, my implementation of Hangman is a quality program, with over 80% code coverage. TDD guarantees that the current functionality works properly, and also that code can easily be modified on the off-chance that a bug is found. Due to this impressive assurance of quality, as well as the user-friendly yet appealingly-minimal interface, I would highly recommend that this program be released immediately.



Passing tests



Code coverage