501 A1. Refactoring, and Source Control

Chris Wozniak

October 9, 2015

Why refactor?

The reason for refactoring this assignment, the CPSC233 Assignment 1, was for a few reasons.

1 Old Code

This was the first assignment done in java, and as such it was not up to par with what I'm capable of writing today.

2 No "meta-work"

By metawork, I'm referring to unit testing, source control, in-line or method level comments. The assignment was built and had proper output so it was considered complete.

What was refactored?

The commit log shows versions from the top down, but I will have to start from the initial commit, follow along to determine which version matches the code provided.

1.0 Initial commit/bug fix

This was my initial commit of the origional assignment. There were some memory leaks and a few other stylistic things I had to change just to get myself prepared for the refactoriings.

1.2 Merging similar classes

Here we had two classes: Attacker, and Defender which both had similar functionality, but with different names. Creating a new class called "Fighter" it had all of the properties of both attacker, and fighter. This would make it that much easeier to work with the two objects, since they were both fighters, one merely in an attacking state, the other in a defending state.

1.2.1 Merging methods

Once the classes were merged together, in fighter, the methods got streamlined. Turning "attack()" and "defend()" into "action()" since having both was a redundancy.

1.3 New Simulation class

The driver for the class was performing multiple roles: setting up the objects, getting user input, running the simulation, and formatting the output. It made for an extremely long method that needed to be broken up. In order to organize the many methods that would need to be created, I created a new class called "Simulator" which took most of main's work, and organized them into appropiate methods.

1.4 Exceptions

Now that the classes were defined, I created member Exceptions, to better organize the code. This way, when/if an exception is thrown it will carry with it an appropiate name so that it would make sense within the context of the code.

1.5 Mistake

After making the commit, and working through the code, I realized that the program in its entirety was basically self-automated. As such, once the methods were created there was no need to worry about exceptions being thrown, so long as the objects were created properly. So The next refactor was reverting what I did, then implementing unchecked exceptions so that the program would become much cleaner, and easier to read without losing functionality.

Remark. *Its important to note here that there is a branch in my log. This is because of a mistake in deleting my unit tests instead of deleting my exceptions. As such, I had to create a new branch to save my new content, then checkout the master branch, and do a hard revision backwards to reaquire my unit tests, then merge the two branches together.*

2.0 Clean release

After my unit tests were reconstructed, and tested good with the new exceptions, I made a final commit for a clean print for the assignment submission.

Well-defined example

A well defined example for the refactorings used would have to be the major transition from a long method in "Manager.java" to the class "Simulator.java" where the entire class can be used as an example, since it all came from the original main method.

Unit Testing

Once again, the code as it was created did an okay job handling user input error, with some modifications brought in around 1.2-1.4 to avoid pitfalls (ie. using Scanner's nestInt() without checking for an int first)

With that out of the way, the big trouble next were the unchecked exceptions which were implemented and tested in 1.4-1.5. They check for NullPointerException, and InvalidArgumentException after that. The point of the code is to get the input from the user, then run a simulation out of the users hands, so if we error check, and guaretee that the input is okay, the program will not have a problem running.

Positive or Negative gain?

The codes solution was a minimal one being a CPSC233 project, and as such much of the limitations I ran into turned out to be less of a refactoring problem, and instead had problems maintaining the exact same output at the first iteration. This refactoring begs for more work to be done on the program, as its just too barebones to refactor any further until more substance is applied to the program itself.