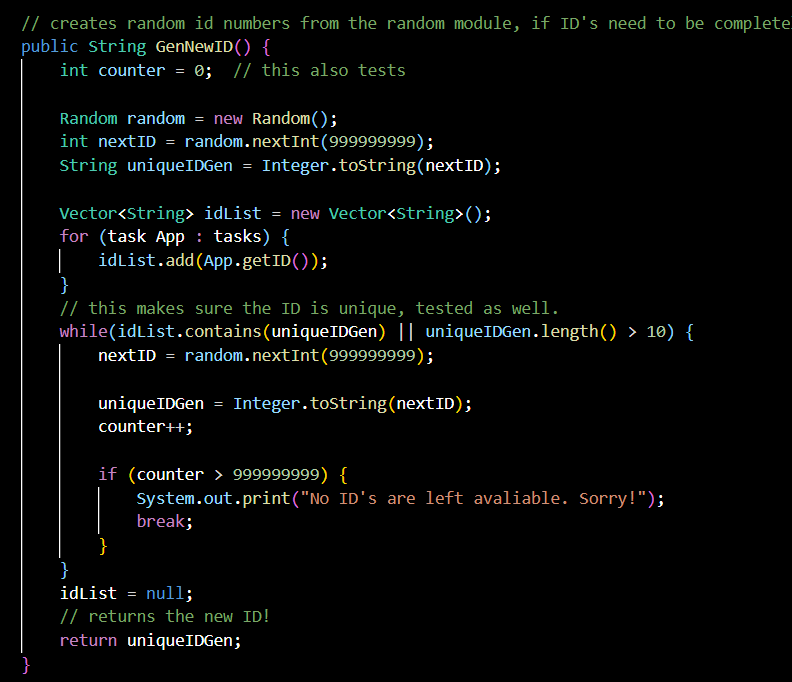
Project 2 Reflection

During the last 5 weeks, I have been continuously working on different software building and testing assignments, culminating into the Project 1 submission, where all files needed to be combined and tested. Over the course of this class, my approach to building and testing software has been completely aligned to the requirements necessary, where they could have been achieved. Examples of this in my code are my coverage results, where worked diligently to align my software with the 80% coverage requirements, having to completely rewrite my original code and tests to align to the requirements. A good code example of this is:



As my Junit testing originally did far better when I used the generator and vectors instead of Array Lists, I was forced to change the entirety of my preexisting code to replicate this, as my coverage tests were abysmal originally across the task assignments and the contact assignments.

Through this, I also ensured my code was technically sound for a final time. Taking another look through my old work, I was able to properly identify what needed to be altered to align to requirements in a better way, and I was able to rewrite software to be more efficient in testing. Specific code chunks that support this are my tests;

void testGetters() {

Contact con = new Contact("02345678", "Will", "McLaughlin", "3838838332", "762 Dake St");

Assertions.*assertEquals*(con.getID(), "123456789");

Assertions.*assertEquals*(con.getFullName(), "William McLau");

Assertions.*assertEquals*(con.getPhoneNumber(), "7046345345");

Assertions.*assertEquals*(con.getAddress(), "762 Fake St");

I wrote every test again to make it more functional, as well as neaten the code to make it actually readable and more functional in coverage testing. This also included testing with the boundary and incorrect values. Null tests and format tests were included in the contact files, as well as the task files to improve testing efficiency and soundness. But still overall, the biggest change I made was adding vectors. This allowed my code to test so much more smoothly, getting my wanted results nearly immediately.

Over the course of this class, I used functional testing and unit testing the most over my development of software. Functional testing is when a tester is testing the software for their requirements by running tests, not just taking the code at face value. Functional testing is used in every type of project, as everyone needs to actually test their code and see if it works. Unit testing is where individual blocks or units of code are being tested to make sure they work, which is all I did with the J Unit tests that were ran. Unit testing could be used in any type of project as well, however, it is best used in projects where features or more modular, and can be broken down easily to be tested separately. Testing techniques I did not use at all would be things like automated tests, and regression tests. Automated tests are tests written that automatically occur to ensure the software is up to standard and working. This type of testing could be used in any type of code or testing, as there are no real special requirements. Bigger projects would be mor ideal for it to be used, however. This was not used in the slightest by me in my J Unit testing. Regression testing is a technique in which the software is tested as soon as a new feature is added, to make sure that the new feature doesn’t cause issues. This could be used in just about any project, no matter the size or use. In my code, I wrote all the features and tests at once, and then fixed any issues, as the project was not too big.

The mindset needed to test software is quite different than the mindset needed to create software, as you become far more routine with your testing and method and detail driven. In my time spent building tests, since the requirements for all three aspects of the final project were similar, I found myself always falling back on being careful, and re using what had already proven to be successful for testing. Writing several tests that tested each aspect in a different random way was a way I employed caution in my tests. My end goal was to find consistency in my software that I was able to universally apply to the interactions of code.

Bias was not really an issue that I needed to limit in my code I do not think. Early on in building the tests I decided that everything should align with the requirements no matter what, and that if it was in the requirements to be tested it would be written. However, I imagine bias could become quite destructive to the quality of your code if it is allowed to creep in. Just realizing your code is very fallible I found is always a great way to remove bias from your code. Realizing that no matter what you do, the computer is always correct. If the code does not run it is simply wrong, and if it doesn’t test well, the code you wrote is also simply wrong.

Finally, discipline is something that presents itself when writing all types of code, tests, or structural. Sometimes I find coding to be like a battle, where I’m just sticking in there until I finally figure out the bug or the issue in the test or framework. It is important to never cut a corner when testing or writing code, because the code is all interconnected. One issue with one line or one unit test can cause the entire thing to come crashing down, and this becomes even mor important when you have an actual coding job. To avoid technical debt in the future, I will have to find patterns in my testing and building of code that work consistently, as well as learn from my peers with more experience in the code base what is actually working, and what is not. Note taking and consistency with learning will be very important factors in my success of avoiding technical debt.