Project 2

By: Vidall Tyric Williams

For the appointment, contact, and task features I made sure to closely examine the requirements and implement them in a way that followed modern day Java practices. To this end, the approach I used was able to align perfectly with the presented software requirements. The requirements for all three features required the id be unchangeable after it was first declared. In order to accomplish this I added a check within the code. In the contact.java file the check is used on line 13 and the function is created on line 58. It throws an error if the id is being changed after it has already been set in the constructor. This however is just a preventative backup measure. The first way to prevent the id from being changed is to simply remove the set function that usually accompanies java class variables. Another common requirement was to ensure that the strings could only go to a certain length. This is seen in every constructor and setter that I wrote. There are if statements checking that the string length limit cannot be exceeded and if it is to throw an error.

Using the coverage percentage as a basis for the effectiveness of my Junit tests I would personally say that the tests I wrote were highly effective. They properly tested the null and string length checks as well as assessed whether the id could be changed after its been set. The only check that was difficult to change into a test was the function that allowed each of the changeable attributes of each class instance to be changed through user input. The difficulty arose from the test not being able to run after a test function had been completed.

I ensured my code was technically sound by thoroughly going over the code and making sure that each and every requirement was met and had proper tests built around it. This meant providing code for not only the basic level of requirements but also addressing edge cases and ensuring the code can run fully on its own. An example of this is when it is required that the code used a built-in data structure to store each of the different feature classes. I made sure that even if one was provided outside of the code at a later date that there was an array list that could also provide that functionality for testing. I made sure my code was efficient by shortening when I could such as in line 13 on contact where I used a ternary operator instead of an if statement. I also used the built in null checker since it was reported that it was the recommended way to check for null.

The focus of the class is testing and because of this there were a number of different testing types I was able to employ that ensured my code met the requirements of the assignment. These included unit, integration, white box, system, and acceptance testing. There were also forms of testing that weren’t accessible to me because of their primary use cases such as black box testing and some non functional tests.

Unit testing is used to test each individual code unit to make sure they are returning expected outputs versus their inputs. This form of testing is more than likely to be the most used form of testing. In my case it was without question. There are a multitude of different errors that can occur at runtime for any size project and the only way to safely manage that is with unit testing. Making sure that the smaller parts that make up the whole can do their job is the most effective and important way to ensure that all the other tests run smoothly and the code is problem free.

Integration testing is when the different modules, units, or pieces are tested together to see if they run as well together as they do apart. Integration testing is almost a guaranteed requirement in any testing situation because of this and passing the integration tests are an important step when testing the system.

System testing is when all of the code is tested together to make sure that it functions correctly. It is a 100% requirement that any shipped code works well with the rest of its parts or there isn’t much reason to make the code in the first place. System testing is therefore essential to any team’s testing suite.

Acceptance testing is when the code is tested to make sure it meets the requirements of the person, people, or entity that requested them in the first place. Acceptance testing is used frequently in every field as the person who commissioned the code more than likely had some idea of exactly what they wanted the project to look and act like. This is also a chance for features to be removed or added based on where the project itself is and what the commissioner wants.

Black box testing is when you get someone with no real knowledge of the code base to test your code. This form of testing is especially useful when the product will be used by people. One of the biggest users is the game industry. Any number of hundreds or thousands of bugs are possible within a videogame so having users interact with the product before it ships is a must. On the other hand, white box testing is the primary form of testing used by coders and is what I use. It is the opposite of black box testing meaning that the tester has complete knowledge of the codebase and if present they also know of the design documents and overall planning of the application.

Non-functional tests are used to test the parts of the code that are not generally related to the main functionality of the application. These includes things like the security, speed, and compatibility across other platforms and devices.

The mindset I approached this project with was a very open one. I tried to google ways I was implementing the requirements even if I was able to get them correct the first time. I wanted to make sure that I was implementing each requirement in the best way possible. This led to a very natural respect for the complexity of the code and how it relates to each other. Each step was to ensure that the end user experience was good not that I was getting a good grade or doing something my way. This also naturally helped lessen the impact of my bias. I found numerous mistakes in my own code but because I approached from a viewpoint of coding for the user I was able to get the job done instead of being stubborn and being afraid of making a mistake. This approach is also essential to my way of eliminating technical debt. I code for the end user experience because at the end of the day theyre the ones funding the project. Any concerns or misfires when they are using the application in a real life scenario can lead to thousand or even hundreds of thousands of dollars lost and sometimes even hours. Its not my job in the field to be stubborn or try and prove something. Its my job to make a product that does what its supposed to do as efficiently and quickly as possible.