Tatiana Case

August 20th, 2023

CS-305

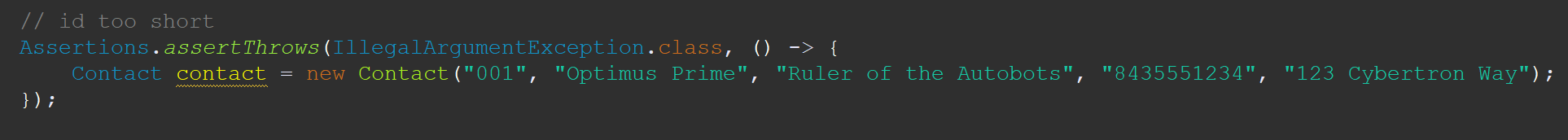
Project Two

Summary and Reflections Report

I made sure that my testing strategy met the needs of the product by creating a test for each of the possible outcomes. For example, in my TaskServiceTest I have a text for null called “testUpdateTaskName.” This test also includes a written description of its purpose and a name. I have also included tests to update task description, delete contact, and add contact. To test the required parameters, in the TaskTest java file, I have included tests for the max task ID length of 10 characters, max task name length of 20 characters, max task description of 50 characters, task name cannot be null, and task description cannot be null. These were all the requirements necessary for this program. I applied this same process with the contact service program. Unit Testing Approach:

There are a few things that I consider when determining how effective my JUnit tests are. The first thing I consider is code coverage. I ensure that my tests cover as much of the code as necessary. The second thing I consider is the number of tests. I need to ensure that I have a decent variety of tests which cover all possible scenarios. The third thing I consider is the quality of the tests. I need to ensure that all my tests are coded correctly and easy to follow. When it comes to code coverage, I believe 80-90% is a good amount of coverage to aim for.

To ensure that my code was technically sound, I ran tests with inputs that were purposefully incorrect to ensure that the proper outcome was executed. For example, In ContactTest, I inputted an ID that was 3 characters when the minimum characters in an ID should be 13. If the code is written correctly, this should throw an exception. When the program threw an exception, I knew that my code was technically sound.



I made sure that my code was efficient by keeping it simple and not adding any redundant lines of code. This allows me to keep my code organized and keep trac of all the requirements that need to be included. For example, the code that I’ve added below is clear and concise. By using variables that clearly describe what I am testing while keeping my code as simple as possible, it fulfills the requisite criteria while remaining efficient.

A screen shot of a computer program

Description automatically generated

There were many testing techniques I could have applied to the milestones, but I chose to go with the ones that I am most comfortable with. I focused mainly on functional testing to determine whether the functional requirements were being met. For example, in the task service milestone, we were to write code that could add tasks with a unique ID, delete tasks per task ID, and update the name and descriptions of these tasks. After creating code to complete these tasks, I was able to test the code by adding and deleting tasks and by updating the names and descriptions of tasks. If the code could not complete those actions, I would be able to go back and determine where the errors in my code were and adjust as necessary. Concerning usability and maintainability, I made sure that the code was structured to properly define functions and processes. This allows the code to be easy to read and easy to manipulate in the future if necessary. to perform in its specific block to increase efficiency and usability. Unit testing was also performed to validate if the unit was running accurately or not and if the requirements were achieved.

The techniques that I have not used are techniques that require all classes to be integrated before implementing. These include integration testing, acceptance testing, and performance testing. I will perform integration tests to see how well the classes work together after all the classes have been integrated. Once I am satisfied with the integration of all classes, I will perform acceptance tests to test its acceptability. Finally, I will conduct performance tests to ensure that the program is performing to the best of its ability.

Functional testing, in my opinion, is the best way to test code in the early stages of development. It allows the programmer to check for functionality and correctness as they go. It also allows for trial an error. A programmer can go back and add or delete code when necessary to ensure that the code is working and that it was created using best practices. This type of testing can be applied to any project or program. For example, in another course I was tasked to code a world clock that showed the current time for whatever country the user requested. When I thought I was done with the program, I was able to act as the user and input different countries. If the time displayed was incorrect, I went back into the code and changed lines wherever necessary. I was able to do this until the code was working and successful.

My mindset when approaching a program is that of a surgeon. I go through requirements like a fine-tooth comb and make sure to have all of them isolated before beginning any work. I also make a list of the importance of those requirements, ensuring that the most critical are completed first then working my way down the list. Then, I complete in-depth tests for each part of the program. For example, I conducted tests simulating user inputs to ensure consistent outputs and program behavior. By following testing plans and adhering to software requirements, I believe I was able to limit bias in the review of my code. By writing and testing my own code, it would be easy to have bias considering my style of coding and my comfort with my own processes. So, objective validation and external input are crucial to impartial testing. Discipline and commitment to quality is of the upmost importance in software engineering. Coding lazily and cutting corners runs the risk of compromising the program’s stability and functionality. While developing my project, I meticulously tested code components, even when I felt like giving up. Having the patience and fortitude to persevere when the code is not functioning properly is the difference between a good coder and a great coder.