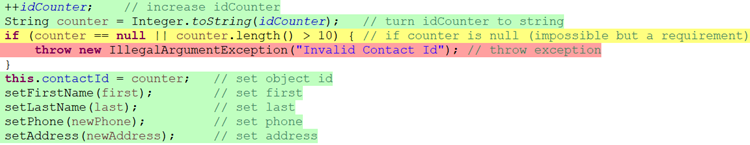
Brian Engel

Project Two

My testing approach was designed mainly on the requirements of the software. While I did test for the basic functionality of the classes while writing them, the Junit tests were all designed around the requirements. If you look at the tests for any of the classes, you will see that the test names all reflect what requirement I am testing for in the test. A few examples of the test functions are: void testUpdateTaskInvalidDescription() from TaskServiceTest.java, void testConstructorWithLongDescription()from TaskTest.java, void testAddContact() from ContactServiceTest.java, and void testFirstNameNotMoreThan10() from ContactTest.java.

For the contact services Junit tests I couldn’t get quite to 100% coverage because instead of asking for a contact id I automatically assigned one. I started the id at the number 1000000000 and for every contact added it added one to the number and turned it into a string. I have a line to throw an exception if it is null which I think is impossible the way I set it up, or if the string is longer than ten characters, but I would have to create 9 billion contacts to get to 11 characters.





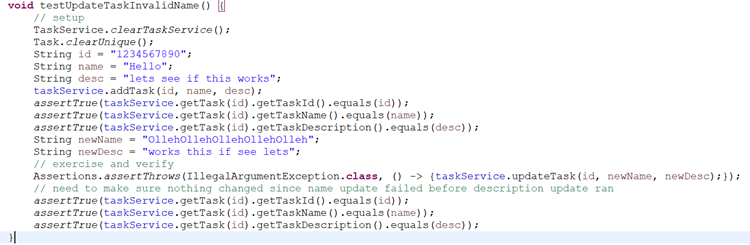
In task service I decided to not use a counter to assign a task id and that simplified thing quite a bit. I got 100% coverage for both Task and TaskServices classes.



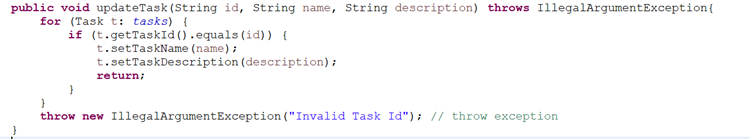
To ensure that my classes were technically sound I ran a ton of different test cases on all the functions. While most of the time everything ran as intended, using the Junit tests helped me catch an unexpected outcome in one of my classes. In the update task function in task service I had originally tried to use this code:



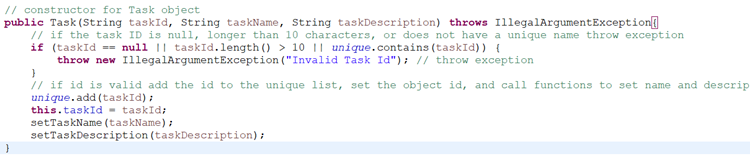
I thought it was pretty clever and got to reuse some of my code. I was running tests on this function and it worked fine most of the time, but when you would enter an invalid name, the constructor that is called in addTask would throw an exception on setTaskName and never run setTaskDescription, and you would end up with a Task with an id and no name or description.

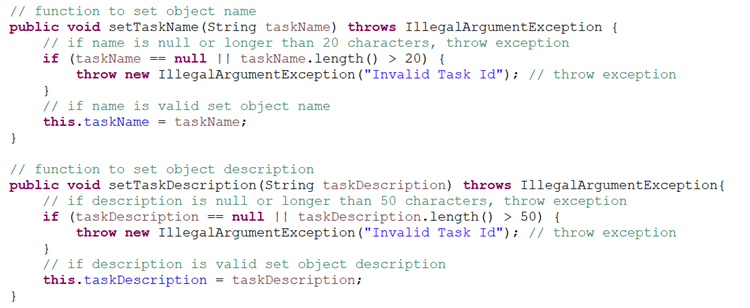


I had to switch to the code below which is probably what I should have done in the first place. Now if there is an invalid name it keeps the old name and description, and if there is an invalid description it changes the name to the name argument, but keeps the old description.



I tried to make my code as efficient as possible by reusing as much code as possible. The above example is a time it didn’t work out, but my constructors are a good example of when it did work.





As you can see instead of assigning this.taskName and this.taskDescription I call setTaskName and setTaskDescription. These are also used in the code above for updateTask. Doing it this way makes sure every time these are called they properly validate the arguments, save a lot of lines, and make sure the code is consistent.

In each of the milestones, the main type of testing I did was functional testing, using unit testing in J unit. I would create test cases from the requirements and write a specific unit test for each test case. This ensured that the output of each function matched what the requirement was. I suppose I also used a very small amount of integration testing as well, since some of the functions that I wrote call other functions in them. This occurs most frequently in the way I designed the constructors. They set the id since it has to be unique, but then call a separate function to set everything else, such as name, date, and description. Also, integration testing was also used in between classes. An example would be that the TaskService class integrates the Task class into it by having an array list of Task objects. If the Task class didn’t work, neither would the TaskService class. One other type of testing I did would be regression testing. Every time I made a change to anything, I would run the entire test suite again to make sure everything was still working.

There are a bunch of types of testing I didn’t use as well. I didn’t do any sort of performance, stress, or load testing. To do this you would have to test the application with large amounts of data that you would encounter in use. This would show if the system were stable, ran fast enough, and how much data it would take to break it. There was also no usability testing, since there is no user interface to interact with. There was no acceptance testing, since no stakeholders passed off on it, and no Alpha or Beta testing since the application isn’t finished and definitely not ready to be tested by users.

The practical use of these techniques really just depends on where you are in the project. Functional testing and unit testing are going to be most useful when you are first writing code and making sure it passes the use cases. Integration testing is for when you start putting classes and packages together to make sure they work with each other. Regression testing is most useful when you change code and have to make sure you didn’t break anything while doing it. All of the performance testing would happen after you have a working application. Usability testing could be included with alpha and beta testing, and honestly anyone working with the user interface could give feedback that would count as usability testing. In alpha testing, the goal is to find and fix as many bugs as possible before it gets passed on to a group of users and outside testers in beta testing. This is real world conditions but on a limited scale and done to discover any issues with different systems and environments. After this, if everything works correctly, there is only acceptance testing, which is done by the stakeholders to make sure the software meets all of their requirements.

The mindset that I took to testing on this project was not only to make sure that the software requirements were fulfilled, but to try and break my app with invalid inputs wherever I could enter them. I guess you could say I used 0 caution in testing. For just about every input in the program I had a test for a valid input, a too long input, and a null input. Since they were all strings I didn’t really have to worry about type too much, which made it a bit easier. The one thing that got pointed out was that my date input was accepting a date in dd/MM/yyyy for todays date when it shouldn’t have been, and it was something that I had totally overlooked in testing, because I didn’t realize there was a lenient option for parsing a date that I had to turn off. (Thanks for pointing that out by the way. I ended up learning even more about parsing dates because of it.)

I don’t think I had any bias in reviewing my own code, as opposed to reviewing someone else’s code. I tried my hardest to think of ways to make my code fail, and to make sure it was the highest quality I could write. Honestly, since I wrote the code, I knew certain things that might make it fail and I made it fail if I could until I got them to work correctly. The updateTask function from above is an example of that. I would rather break my own code than for it to fail later in the process and waste a lot more time and money.

It is extremely important to be committed to quality in your coding and testing. As a tester, that means testing as much of the code as possible and as many inputs as possible to make sure the code is bug free. Most importantly, it means making sure the software requirements are fulfilled. As a developer, commitment to quality is making sure your code is efficient, reusable, and as uncomplicated as possible with comments to help others follow the code, and bug free. This can be seen in my code in the constructor calling the setters. By reusing code as much as possible, it lowers the chance of introducing a bug, and if there is a bug you only have to fix it in one place. In the testing, you can see this by the different inputs I have for every single function, and the distinct testing function names to let you know exactly what the function is testing for.