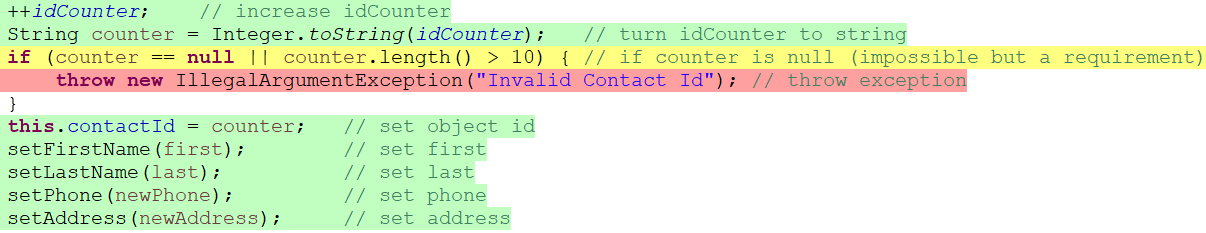
Brian Engel

Module Four Journal

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.

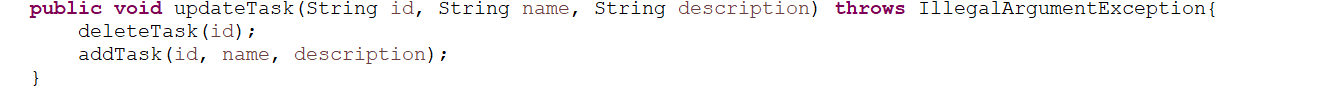




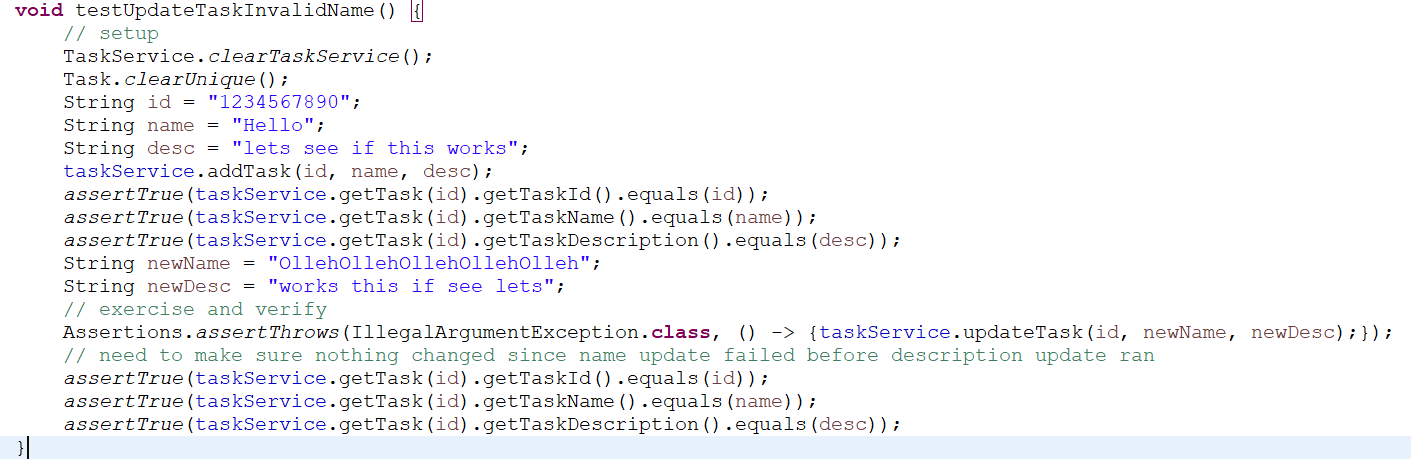
In task service I decided to not use a counter to assign a task id and that simplified thing quite a bit. I got 100% coverage for both Task and TaskServices classes.



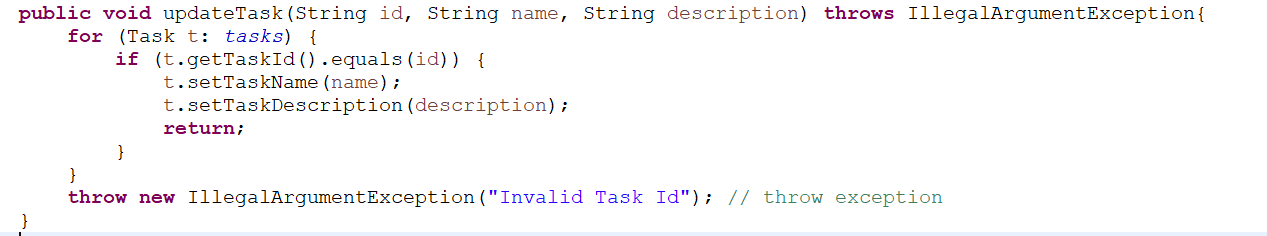
To ensure that my classes were technically sound I ran a ton of different test cases on all the functions. While most of the time everything ran as intended, using the Junit tests helped me catch an unexpected outcome in one of my classes. In the update task function in task service I had originally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



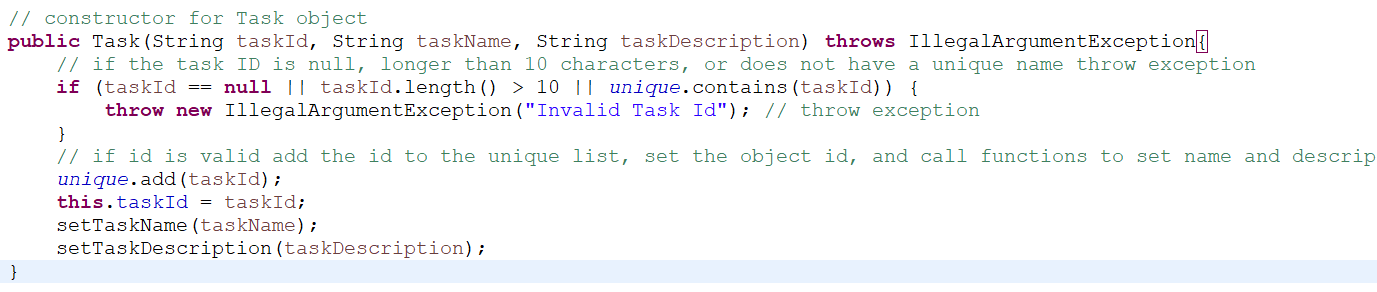
I thought it was pretty clever and got to reuse some of my code. I was running tests on this function and it worked fine most of the time, but when you would enter an invalid name, the constructor that is called in addTask would throw an exception on setTaskName and never run setTaskDescription, and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

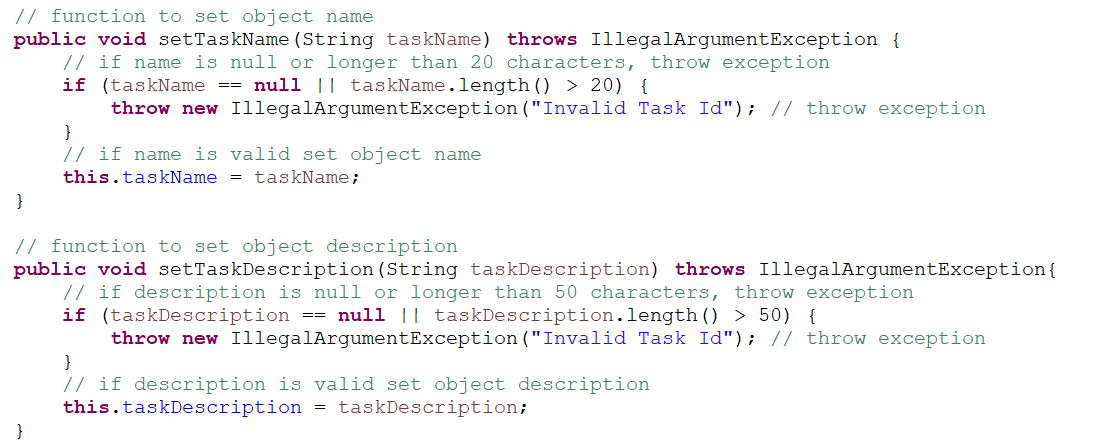


I had to switch to the code below which is probably what I should have done in the first place. Now if there is an invalid name it keeps the old name and description, and if there is an invalid description it changes the name to the name argument, but keeps the old description.



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.