Lab: Advanced JUnit

Using the features of JUnit

Introduction

You should now have gained the knowledge and skills necessary to utilize JUnit testing in a significant manner in your projects. In today's lab, you are going to be building a series of JUnit unit tests for a small calculating system that we provide for you. There are two files you will need: Simple.java and Complex.java. You will need to import them into Intellij and set up the project structure to match the package statements.

Dependencies

Make sure to include the following dependencies in your project

- JUnit Engine
- JUnit Params

In both files there are implementations for some different types of mathematical calculations. You are going to implement JUnit tests to verify that these methods are working properly. There may be errors in some of them, or they may be working properly. If there are errors in any of the methods, you will need to correct the errors in the source files after catching the errors in your tests. Be sure to consider the equivalence classes of each operation and the edge cases you will need to test to ensure you can catch any errors.

You are allowed to make changes to the source files as necessary to make your tests pass; however you cannot change the functionality of the methods. Your test cases must adhere to the requirements in the next section, and all of your test cases must pass in order to receive full credit. You will turn in both of your test files and your modified source files (if you made any changes). You will also submit a screenshot of your test cases running and passing.

Requirements

To receive full credit for today's lab, your solution must meet the following requirements:

1. At least one test for each method
    a. Tests must be nontrivial (i.e., your test must actually test the methods. Your tests must be able to fail if the method contains a bug.)
2. Test files must parallel the structure of the project, i.e. a test file for each source file

3. You must use display names or display name generators to label your tests. For the **Complex.java** test file you must use **nested tests with display names** to test the different methods.
    a. You need to have at least 3 non-trivial tests for each function in Complex.java. One should be a normal test, and 2 should be edge cases.
    Note: A parameterized test method with multiple sets of inputs counts as multiple tests.
    Note: An edge case is generally described as a situation that occurs at an extreme parameter. This happens usually at minimums and maximums, or at points in an algorithm or piece of code where we expect output to change in a significant way. Since we are testing math equations general edge cases will occur at places such as 0, negative numbers, or extremely large scale numbers. If you can think of other bounds where behavior might change you are more than welcome to test for these.
4. You must test for Exceptions in every method that throws an Exception
    a. **Note:** Any method than can divide by zero may throw ArithmeticException
5. You must create one parameterized test for each method in both source files
    a. Include a comment before each parameterized test that shows how your inputs cover the equivalence classes for that operation
6. There are a few bugs in the code that you will find through unit testing. When you find these bugs, fix them and put a comment next to your code explaining what was wrong and how you changed it.
7. All your tests must pass when run

Notes

In order to get parameter testing to work you must either have a dependency on `junit-jupiter-params` or you need to have dependency version 5.5.0-M1 (or later) as your normal org.junit.jupiter dependency.

Submission

Electronically submit

- SimpleTest.java
- ComplexText.java
- your modified code files
    - Simple.java
    - Complex.java
- a screenshot showing the results of running your tests.

Example Screenshot: (You will have more tests than this)