Homework #1

Due September 11th, 2020

TDD & JUnit

1. Introduction

This assignment helps you practice Test-Driven Development (TDD). The problem you are trying to solve here is one that you should be able to solve in a prior course on Data Structures. However, here you need to use the TDD methodology we have covered.

Note that the 3 laws of TDD are:

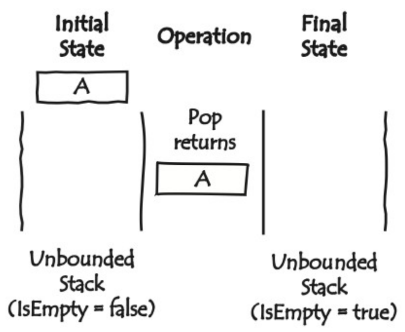
1. You may not write production code until you have written a failing unit test

2. You may not write more of a unit test than is sufficient to fail

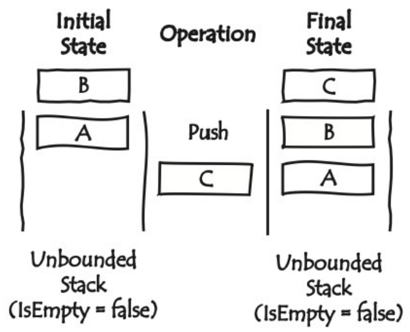
3. You may not write more production code than is sufficient to pass the currently failing test

NOTE:: do not write the functional code in your head before you write the test.

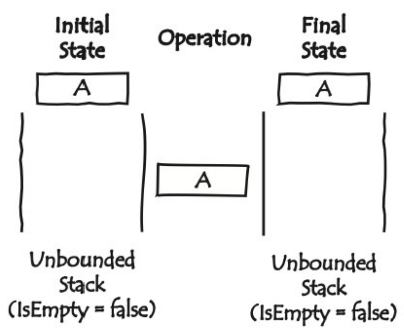
In this assignment, you will use JUnit and Java to implement a **<u>bounded stack of size 5</u>**, namely MyStack, using an integer array. Recall the three basic operations of a traditional stack are push, pop, and top as shown below:


Initial State
Operation
Final State
A
Pop returns
A
Unbounded Stack (IsEmpty = false)
Unbounded Stack (IsEmpty = true)

(a) Pop


Initial State
Operation
Final State
B
A
Push
C
C
B
A
Unbounded Stack (IsEmpty = false)
Unbounded Stack (IsEmpty = false)

(b) Push


Initial State
Operation
Final State
A
A
A
Unbounded Stack (IsEmpty = false)
Unbounded Stack (IsEmpty = false)

(c) Top

Homework #1

Due September 11th, 2020

2. Get Started with Basic Test

You should start the assignment by creating a JUnit test class and name it “BasicTest”. This class is for testing/implementing basic operations of a bounded stack. Note that the data structure you should use is array. The following test cases are required, although you do not have to follow the exact order when you implement them. Also, you can have more test cases than what are listed here.

1. Create a MyStack and verify that IsEmpty is true.

2. Push a number on the MyStack and verify that IsEmpty is false.

3. Push a number, Pop the number, and verify that IsEmpty is true.

4. Push a number, remember what it is; Pop the number, and verify that the two numbers are equal.

5. Push three numbers, remember what they are; Pop each one, and verify that they are removed in the correct order.

6. Pop a MyStack that has no elements.

7. Push a number and then call Top. Verify that IsEmpty is false.

8. Push a number, remember what it is; and then call Top. Verify that the number that is returned is the same as the one that was pushed.

Homework #1

Due September 11th, 2020

When you implement these tests the MyStack class should also be implemented along the way. **To indicate the order in which you implement the tests and features label each method (in both test and functional code) using STEP 1, STEP 2, etc, and include a short description about the method.** Listing 1 shows a sample based on the class demo.

Listing 1: Sample test code with STEP ID and comments.

```java
/**  === STEP 1 ===
      * Test 1: test initialization
      * When an account is created, its balance should be 0.
      */

    @Test
    public void test_settlement(){
        BankAccount ba = new BankAccount();
        assertEquals(0, c.settlement());
    }
```

In addition, you should also document your refactor activities in a text file refactor.txt. Each line of this file should look like the following:

Refactor after STEP 2: “description of changes.”

Note that you do not have to do refactor in every single “red-green-refactor" cycle, but for the ones that you do improve you should document the improvement in this file.

Homework #1
Due September 11th, 2020

3. Advanced Test

Now that you have already finished implementing a bounded stack using TDD, you can write more tests for some special cases:

1. Call Top on a MyStack with no elements.

2. Push a number to a MyStack that is full (hint: you may throw an exception here).

3. A self-defined, new test case (it can be anything, e.g., performance test).

4. Aggregating Tests in Suite

Now that you have two test classes, let’s combine them and make a test suite. When you run this test suite all test cases in both classes should be executed automatically (hint: to do this, you need to use the @RunWith annotation as seen in Step 9 of the JUnit guide and demo code).

5. Submission

This assignment is due at 11:59pm on 9/11/2020. Late submissions are subject to a 20% penalty. Late submissions will not be accepted after 3 days past the deadline (11:59pm on 9/14).

Submit the following files as a package to Blackboard:

1. BasicTest.java

2. AdvancedTest.java

3. TestSuite.java

4. MyStack.java

5. refactor.txt

Homework #1

Due September 11th, 2020

6. Grading

10pts	BasicTest Code Correctness
10pts	AdvancedTest Code Correctness
10pts	MyStack Code Correctness
10pts	Correctly followed TDD
5pts	Test Suite
5pts	Code Readability

TOTAL: 50 pts