TDD Using Java

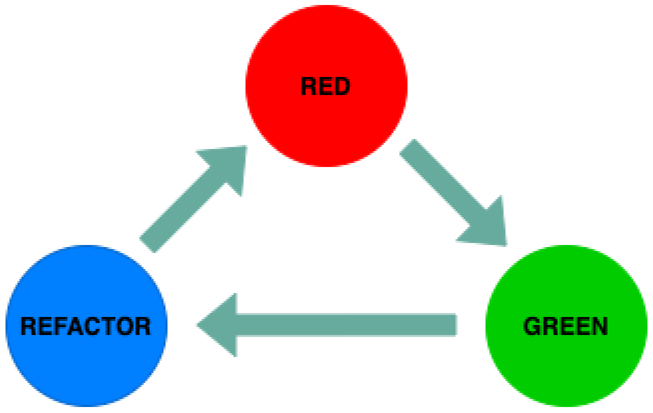
Worksheet

Introduction

Welcome to the beginning of your journey with us into learning more about Test Driven Development in Java!

This guide will walk you step-by-step through how we use TDD in order to develop projects in Java :)

What Is TDD?



1. Test the simplest thing you can
2. Write the simplest code you can to make the test pass
3. If the code quality can be improved, refactor

Tips:

Keep in mind that the idea behind TDD is to do the minimum amount of work to make the tests pass and repeat the process until all of the functionality is implemented.

If you only had tests that assert on the value being true then you can simply make the test pass by making the method return true with no other logic.

The Luhn Algorithm?

Luhn Algorithm, or Modulus 10 Algorithm, is a mathematical formula that helps to determine whether or not a correct identification number has been provided.

This will be explained with a Visa number: 4003600000000001

1. Multiply the digits from the back, starting from the second-to-last. Skip one number, double the next until you're now on the first number.

400360000000014

1 x 2 + 0 x 2 + 0 x 2 + 0 x 2 + 0 x 2 + 6 x 2 + 0 x 2 + 4 x 2

2. Now, we have product digits for all doubled digits, e.g. the product digit of 1 × 2 is 2. However, the product digit of 6 × 2 will be 1 + 2 (derived from its product '12') because 12 is higher than 10. Add all the derived product digits together.

= 2 + 0 + 0 + 0 + 0 + 1 + 2 + 0 + 8 = 13

3. The sum derived should be added to the sum of the digits that weren't doubled.

13 + 4 + 0 + 0 + 0 + 0 + 0 + 3 + 0 = 20

4. If the last digit in the total is 0, then the set of numbers is valid.

The last digit in 20 is 0. So, the Visa number is correct.

Requirement 1: Check if the credit card number is valid

Our first requirement is to check if the credit card number consists only of numbers, no letters or special characters.

You will need to implement both the test and the code that does this.

In the test class you will find a method called "validCreditCardNumberTest", this is the test that you will want to modify for this task.

[JAVA TEST]

```
void validCreditCardNumberTest(){
    assertTrue({call to method to validate credit card number});
}
```

What test cases can you think of that we should be testing here?

Requirement 2: If the credit card number length is valid return true from method

Using what you have learned in requirements one, you should be able to implement the following test. Once again think about the scenarios that you would like to cover.

[JAVA TEST]

```
void isCreditCardLengthValidTest(){
        {Your assertions here}
}
```

Requirement 3: If the credit card number length is invalid return false from method

[JAVA TEST]

```
void isValidCardNumberLengthReturnsFalseWhenLengthIsInvalid(){
```

    {Your assertions here}

}

Requirement 4: If the credit card number length is invalid from method then throw checkInvalidCardLengthThrowsException

Tip: look up how to assert on exceptions for assertJ

You will want to implement a luhnCheck method, that will throw this exception using the methods you tested in steps 1-3.

[JAVA TEST]

void checkInvalidCardLengthThrowsException(){

        {Your assertions here}

}

Requirement 5: If the credit card number length is valid, ensure that checkValidCardLengthDoesNotThrowException is not thrown

Tip: look up how to assert on exceptions not being thrown for assertJ

[JAVA TEST]

void checkValidCardLengthDoesNotThrowException(){

    {Your assertions here}

```
}
```

Requirement 6: Implement the Luhn Check

Tip: you will want to write a test case for each size credit card number, 14, 15 and 16 digits.

[JAVA TEST]

```
void checkLuhnSumReturnsCorrectValue(){

    {Your assertions here}

}
```

Once you have completed this, do you think we can make the test cleaner in any way? If you do, raise your hand and one of us can check with you. If you don't think there is then explain to us why you don't think there is.

There are 3 more requirements left. Try them out by yourself, applying what you have done so far :)

Requirement 7: Validate National insurance number

DO NOT USE A REGEX FOR THIS TASK!

Given a String check if it is a valid national insurance number (NINO), you can find out more about what a valid national insurance is here https://www.gov.uk/hmrc-internal-manuals/national-insurance-manual/nim39110.

You can assume that you will be passed the whole national insurance number and nothing else, there will be no spaces in the number.

Your method should return true or false depending on if the String provided is a valid NINO.

Requirement 8: Given a String that is a valid national insurance number, mask the 6 numbers but leave the letters.

Given a valid national insurance number, mask the 6 numbers but leave the rest. By masking we mean replacing these numbers with the character "*". For example the national insurance number: QQ123456A would become: QQ******A. You should not mask if the national insurance number is invalid. You should return the masked String.

Requirement 9: Mask the national insurance number in a given String

The previous method you implemented for requirement 8, says to expect the String to only be the national insurance number. We have new requirements that state that the String will now be a full sentence or paragraph and all national insurance numbers in that String must be masked. For example if you have the String, "Hi my name is Bob and my national insurance number is: QQ123456A", this should return a String "Hi my name is Bob and my national insurance number is: QQ******A"

We have created a repo that contains the ways that we would have implemented the code to solve the above problems, if you are interested in this please let someone in the Zoom know your GitHub email address and we will add you.

And that's all folks! We hope you enjoyed your Capital One workshop experience :)