Test Driven Development (TDD) - Individual Exercise

The purpose of this exercise is to provide you with the opportunity to practice the art of developing high-quality software using the process of Test Driven Development (TDD).

Learning Objectives

After completing this exercise, students will understand:

- How to use the Red, Green, Refactor approach to design and develop functional, high-quality code.
- How to use code katas to practice and refine development skills.

Evaluation Criteria & Functional Requirements

Your code will be evaluated based on the following criteria:

- The project must not have any build errors.
- Unit tests pass as expected.
- There is appropriate code coverage to verify the application code is functioning as expected.
- Katas have been completed as assigned.
- Code has been developed using TDD practices. You are expected to:
    - Write a test.
    - **Add and commit the test with the message "Add test method " followed by the test method name**. For instance, "Add test method result_should_be_Fizz_when_number_is_3".
    - Write enough code to make the test pass.
    - **Once the test is passing, add and commit the code with the message "Add passing code for test method " followed by the test method name.** For instance, "Add passing code for test method result_should_be_Fizz_when_number_is_3".
    - Repeat these steps until all tests have been implemented and are passing.
- **Code that does not have an associated unit test will be considered incomplete**.
- Good test method names are provided that clearly state what is being tested.

Remember, when using a test-driven approach to development, you **do not write all of the tests first** and then write all of the code to make the tests pass. TDD follows the following process:

- Write a single test method based on the requirement you are working on.
- Run the tests, and verify that the test fails. (Red)
- Write enough code to make the test pass. (Green)
- Refactor if necessary, and verify that the code still passes. (Refactor)
- Repeat previous steps until all of the requirements have been satisfied.

For this exercise, you will be completing several code katas. Code katas are a great way to practice your development skills, and are extremely valuable to developers who want to become better practitioners of TDD.

The requirements for each kata are provided below. Your job will be to implement a ***test-driven approach*** to solving these problems.

Empty classes and test files have already been created for you.

FizzBuzz Kata

Part 1

FizzBuzz is a program that interviewers will often have developers they are interviewing with write or discuss during the interview process. We will use this opportunity to practice solving this problem.

Your job will be to create a method called `fizzBuzz`, that will accept an `int`, and returns a `String`. The string that is returned is based on the following requirements:

- If the number is divisible by 3, convert the number to the string, "Fizz".
- If the number is divisible by 5, convert the number to the string, "Buzz".
- If the number is divisible by 3 AND 5, convert the number to the string, "FizzBuzz"
- For all other numbers between 1 and 100 (inclusive), simply convert the number to a string.
- Any number that is not between 1 and 100 (inclusive), convert the number to an empty string.

The following illustrates these requirements:

Method Call	Expected Result
`fizzBuzz(1)`	`"1"`
`fizzBuzz(3)`	`"Fizz"`
`fizzBuzz(5)`	`"Buzz"`
`fizzBuzz(15)`	`"FizzBuzz"`
`fizzBuzz(22)`	`"22"`
`fizzBuzz(0)`	`""`

Part 2

A new requirement has been added by the interviewer to see how well you adapt to change. The new requirement is as follows:

- If the number is divisible by 3, OR contains a 3, convert the number to the string, "Fizz".
- If the number is divisible by 5, OR contains a 5, convert the number to the string, "Buzz".
- If the number is divisible by 3 AND 5, OR contains a 3 AND 5, convert the number to the string, "FizzBuzz"

For instance:

Method Call	Expected Result
`fizzBuzz(3)`	`"Fizz"`
`fizzBuzz(13)`	`"Fizz"`
`fizzBuzz(35)`	`"FizzBuzz"`
`fizzBuzz(5)`	`"Buzz"`

Method Call	Expected Result
`fizzBuzz(51)`	`"Buzz"`
`fizzBuzz(53)`	`"FizzBuzz"`

Are there any additional cases we might be missing?

Potter Kata

Once upon a time there was a series of 5 books about a very English hero called Harry. (At least when this Kata was invented, there were only 5. Since then they have multiplied.) Children all over the world thought he was fantastic, and of course, so did the publisher. In a gesture of immense generosity to mankind, (and to increase sales), they set up the following pricing model to take advantage of Harry's magical powers.

One copy of any of the five books costs 8$. If, however, you buy two different books from the series, you get a 5% discount on those two books. If you buy 3 different books, you get a 10% discount. With 4 different books, you get a 20% discount. If you go the whole hog, and buy all 5, you get a ***huge*** 25% discount.

Note that if you buy, say, four books, of which 3 are different titles, you get a 10% discount on the 3 that form part of a set, but the fourth book still costs 8$.

Your mission is to write a method called `getCost(int[] books)` that will calculate the price of any conceivable shopping basket, giving as big a discount as possible. The array of books that is passed to the method indicates the number of copies of each book the customer is purchasing, and the method will return a `double` that represents the cost for all of the books provided.

For instance, if the customer is ordering 2 copies of the first book, 2 copies of the second book, 2 copies of the third book, 1 copy of the fourth book, and 1 copy of the fifth book, the method will be called as follows:

```
getCost(new [] { 2, 2, 2, 1, 1 })
```

The result for the method call will be $51.20.

Hint

On the surface, this Kata looks easy but there is certainly a level of complexity. When you calculate the above basket, it isn't `5 * 8 * 0.75 + 3 * 8 * 0.9`. It is actually `4 * 8 * 0.8 + 4 * 8 * 0.8`. The trick is to write code intelligent enough to identify that two sets of four books is cheaper than a set of five and a set of three.

Prime Factors Kata (OPTIONAL)

Factorize a positive integer number into its prime factors.

Use the TDD approach to write tests that call a single method `List<int> factorize(int n)`. Given a positive integer `n`, return its prime factors as a `List` of type `int`.

Note 1 is always omitted from the result set.

For instance:

Value of `n`	Expected Values
2	[2]
3	[3]
4	[2, 2]
6	[2, 3]
7	[7]
8	[2, 2, 2]
9	[3, 3]
10	[2, 5]

Roman Numerals Kata (Challenge)

Step 1

The Romans were a clever bunch. They conquered most of Europe and ruled it for hundreds of years. One thing they never discovered was the number zero. This made writing and dating extensive histories of their exploits slightly more challenging, but the system of numbers they came up with is still in use today. For example the BBC uses Roman numerals to date their programs.

The Romans wrote numbers using letters - I, V, X, L, C, D, M. (notice these letters have lots of straight lines and are hence easy to hack into stone tablets).

The kata says you should write a method to convert from normal numbers to Roman Numerals: eg `String convertToRomanNumeral(int n)`.

For example:

Number	Roman Numeral
1	I
10	X
7	VII

There is no need to be able to convert numbers larger than about 3000. (The Romans themselves didn't tend to go any higher.)

Note that you can't write numerals like "IM" for 999. Wikipedia says: Modern Roman numerals ... are written by expressing each digit separately starting with the left most digit and skipping any digit with a value of zero. To see this in practice, consider the ... example of 1990. In Roman numerals 1990 is rendered: 1000=M, 900=CM, 90=XC; resulting in MCMXC. 2008 is written as 2000=MM, 8=VIII; or MMVIII.

Hint

Is there a data structure that we learned that can store all of the numeral letters and their values? (I, V, D, M etc.)

This resource on how roman numerals work may also provide additional insight and value as you work through the kata.

Step 2

Write a method `int convertToDigit(String romanNumeral)` that converts a Roman Numeral to its equivalent integer value.

For instance:

Roman Numeral	Number
XXIV	24
III	3
MCMXCVII	1998

Getting Started

- Import the tdd-exercises project into Eclipse.
- Once you've decided which kata you would like to start with, open the corresponding file in the test/java/com/techelevator directory. For instance, if you are going to start with the FizzBuzz kata, open the `KataFizzBuzzTests.java` file.
- Add a test method to the class for the first scenario you are testing.
- Once you have the test method written, commit the code with the message "Add test method ", followed by the name of the method you are testing. For instance, "Add test method result_should_be_Fizz_when_number_is_3".
- After committing the test code to git, open the class you are testing. For instance, to continue with our example above, open the `KataFizzBuzz.java` file in main/java/com/techelevator.
- Add **only** the code necessary to get the test you wrote in the previous step to pass.
- Now, commit the passing code to git with the message "Add passing code for test method " followed by the test method name. For instance, "Add passing code for test method result_should_be_Fizz_when_number_is_3".
- Repeat these steps until all tests have been implemented and are passing for all katas.

Tips and Tricks

- Katas are used by many companies during the hiring process when working to find new developers to join their teams. As such, it will be to your advantage to practice writing code with code katas on a consistent basis.
- Often times when teams assign katas for candidates, they are looking to see your thought process over time. That being said, they will often be curious to see your commit history when working through exercises. Avoid the urge to write all of the code up front and then adding a massive commit. Instead, write a little code, commit your code. Write a little more code. Commit. This will allow a reviewer to see

how your code has developed over time, and will help reviewers to understand your approach to problem solving.

- Katas are a great way to practice your skills, learn new programming languages, and keep your existing skills fresh. There are a number of katas published online that you are encouraged to work through. Some of these include the KataCatalogue from Coding Dojo, this curated list of awesome katas on GitHub, and katas available from CodeKata.com.
- Of course, there are more modern ways to complete katas as well, though most of them will not provide you the opportunity to hone your craft of TDD. However, some notable ones include CodeWars, LeetCode, and exercism.io.
