Test Driven Development (TDD) - Pair 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). Specifically, you will be practicing this methodology using pair programming.

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.
- How to use the technique of "ping-pong" programming when working in a pair.

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.
- **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 these exercises, you and your pair partner will be practicing TDD using an approach referred to as "ping-pong" programming. The process is similar to the process identified above, but has a few differences as well.

- One of the pair partners will write a single test method based on the requirement that is being worked on.
- The tests will be run and should fail. (Red)
- The other pair partner will then write enough code to get the test to pass. (Green)
- Both partners will then review the code (and the tests) to determine if any code refactoring needs to take place. (Refactor)
- The steps will be repeated until all of the requirements have been satisfied.

For this exercise, you and your pair partner can decide to use one of the following workflows.

Write Tests and Code on the Same Machine

With this approach, you and your partner will share the same machine. One person will take over the keyboard to write the test, while the other developer watches. Once the test is written, the keyboard will be handed over to the other developer and she or he will write the code to make the test pass. This approach will likely cause the least amount of complexity and friction to use.

Collaborate Using a BitBucket Repository

With this approach, each developer will work on different machines. One developer will write a single test method, commit the changes, and then push to BitBucket. The other developer will then pull in the change from BitBucket, write enough code to get the test to pass, and then push to BitBucket. This approach requires a bit more complexity, and developers need to be very diligent and disciplined in order to avoid merge conflicts, however it can be a great way to practice collaborative development skills with remote teams.

Which method you choose to use is at your discretion, and in fact, it is probably best to choose one method for one of the katas and then trying a different approach on another one.

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.

String Calculator Kata

Create a simple `StringCalculator` class with a method `int add(String numbers)`.

Step 1

The `numbers` variable that is passed to the `add` method accepts a comma-delimited list of numbers that should be added together. The string may contain 0, 1, or 2 numbers. The value returned by the `add` method will be the sum of the numbers provided in the string.

If an empty string is provided, 0 should be returned, if only number is provided in the string, then the number should be returned, and if two numbers are provided, the sum of the two numbers will be returned.

For example:

numbers	value returned
""	0
"1"	1
"1,2"	3

Remember, you should write the test methods first, then write the code to get the test to pass.

Hint

Begin with the simplest test case using an empty string and move to 1 then 2 numbers.

Step 2

Adding two numbers together is great, but it would be even better if we could add an unlimited amount of numbers together.

Modify the `add` method to handle an unknown amount of numbers.

numbers	value returned
"5,7,12"	24
"6,2,1,8"	17

Step 3

Allow the `add` method to handle new line characters between numbers (instead of commas).

numbers	value returned
"5\n3,2"	10
"3\n5\n2,4"	14

Note: The input `"1,\n"` is not valid. A comma will not end the line. You do not need to code for it.

Step 4 (Bonus)

Add support for input delimeters. To change a delimiter, the beginning of the string will contain a separate line that is prefixed with `//`. For example, "//delimeter\n[numbers,...]". The input delimiter is optional.

numbers	value returned	remarks
"//;\n1;2"	3	delimited by ";"
"//!\n4!9"	13	delimited by "!"
"3\n5\n2,6"	14	default delimiters
"3,5,9"	17	default delimiters

Numbers to Words Kata (Challenge)

It occurs now and then in real life that people want to write about money, especially about a certain amount of money. If it comes to checks or contracts for example some nations have laws that state that you should write out the amount in words in addition to the amount in numbers to avoid fraud and mistakes. So if you want to transfer $745 to someone via check, you have to fill out two fields:

1. 745 (amount in numbers)
2. seven hundred and forty-five (amount in words)

Step 1

Write a `NumbersToWords` class that can convert whole numbers into words.

Test Cases to Consider

- **1 digit numbers** (zero, three, seven)
- **2 digit numbers** (ten, fourteen, twenty-six)
- **3 digit numbers** (two hundred and nine, three hundred, four hundred and ninety-eight)
- **4 digit numbers** (three thousand and four, five thousand and twenty-six, seven thousand and one hundred and eleven)
- **5 digit numbers** (forty thousand, eighty-seven thousand and six hundred and fifty-four)
- **6 digit numbers** (five hundred thousand, eight hundred and three thousand and three hundred and eight, nine hundred and ninety-nine thousand and nine-hundred and ninety-nine)

Step 2

Write a `WordsToNumbers` class that converts words into numbers.

Getting Started

- Import the tdd-exercises-pair project into Eclipse.
- Once you've decided which kata you would like to start with, create a test class in the test/java/com/techelevator directory.
- 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.
- After committing the test code to git, create the class you are testing in the main/java/com/techelevator directory.
- 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.
- 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.
