TDD Manifesto

Test Driven Development in every home

Exercises

Katas

Greatness comes from practicing. Solving Kata exercises is an excellent way to deepen the fundamentals of Test Driven Development. Kata exercises are short (15-30mins) long coding exercises which can be solved in any language. We highly recommend for TDD beginners to solve multiple Kata exercises before applying TDD in real life projects. Here in this section, we have collected the descriptions of some popular Kata exercises, and some exercises from real-life applications. Have fun with them!

How to solve the Katas with TDD

- Read the “Getting started” and “A clean test” section to familiarize yourself with the fundamentals of TDD (TODO: add link to the sections)
- Pick your favorite programming language
- Pick a unit testing framework for your language. Some popular choices per languages:
    - **Jest** – Javascript/Typescript:

    - **NUnit** – C#
    - **JUnit** – Java
    - **pytest** – Python

- Try not to read ahead of the being solved exercise
- Solve only one requirement at a time

Kata 1 – FizzBuzz

FizzBuzz is one of the most famous coding exercises for beginners. It is a simple exercise but an excellent one to start learning the TDD flow with.

Requirements

1. Write a "fizzBuzz" method that accepts a number as input and returns it as a String.

Notes:

- start with the minimal failing solution
- keep the three rules in mind and always write just sufficient enough code
- do not forget to refactor your code after each passing test
- write your assertions relating to the exact requirements

2. For multiples of three return "Fizz" instead of the number

3. For the multiples of five return "Buzz"

4. For numbers that are multiples of both three and five return "FizzBuzz".

Kata 2 – String calculator

Create a simple calculator that takes a `String` and returns a `integer`

Signature (pseudo code):

```
int Add(string numbers)
```

Requirements

1. The method can take up to two numbers, separated by commas, and will return their sum as a result. So the inputs can be: "", "1", "1,2". For an empty string, it will return 0.

Notes:

- start with the simplest case (empty string) and extend it with the more advanced cases ("1" and "1,2") step by step

- keep the three rules in mind and always write just sufficient enough code
- do not forget to refactor your code after each passing test

2. Allow the `add` method to handle an unknown number of arguments

3. Allow the `add` method to handle newlines as separators, instead of comas

   - "1,2\n3" should return "6"
   - "2,\n3" is invalid, but no need to clarify it with the program

4. Add validation to not to allow a separator at the end

   - For example "1,2," should return an error (or throw an exception)

5. Allow the `add` method to handle different delimiters

   - To change the delimiter, the beginning of the input will contain a separate line that looks like this:

```
//[delimiter]\n[numbers]
```

   - "//;\n1;3" should return "4"
   - "//|\n1|2|3" should return "6"
   - "//sep\n2sep5" should return "7"
   - "//|\n1|2,3" is invalid and should return an error (or throw an exception) with the message "'|' expected but ',' found at position 3."

STOP HERE if you are a beginner. Continue if you could finish the steps (1-5.) within 30 minutes.

6. Calling `add` with negative numbers will return the message "Negative number(s) not allowed: <negativeNumbers>"

   - "1,-2" is invalid and should return the message "Negative number(s) not allowed: -2"
   - "2,-4,-9" is invalid and should return the message "Negative number(s) not allowed: -4, -9"

7. Calling `add` with multiple errors will return all error messages separated by newlines.

   - "//|\n1|2,-3" is invalid and return the message "Negative number(s) not allowed: -3\n'|' expected but ',' found at position 3."

8. Numbers bigger than 1000 should be ignored, so adding 2 + 1001 = 2

Kata 3 – Password input field validation

Create a function that can be used as a validator for the password field of a user registration form. The validation function takes a string as an input and returns a validation result. The validation result should contain a boolean indicating if the password is valid or not, and also a field with the possible validation errors.

Requirements

1. The password must be at least 8 characters long. If it is not met, then the following error message should be returned: “Password must be at least 8 characters”

2. The password must contain at least 2 numbers. If it is not met, then the following error message should be returned: “The password must contain at least 2 numbers”

3. The validation function should handle multiple validation errors.

- For example, “somepassword” should an error message: “Password must be at least 8 characters\nThe password must contain at least 2 numbers”

4. The password must contain at least one capital letter. If it is not met, then the following error message should be returned: “password must contain at least one capital letter”

5. The password must contain at least one special character. If it is not met, then the following error message should be returned: “password must contain at least one special character”

Kata 4 – Search functionality

Implement a city search functionality. The function takes a string (search text) as input and returns the found cities which corresponds to the search text.

Prerequisites

Create a collection of strings that will act as a database for the city names.

City names: Paris, Budapest, Skopje, Rotterdam, Valencia, Vancouver, Amsterdam, Vienna, Sydney, New York City, London, Bangkok, Hong Kong, Dubai, Rome, Istanbul

Requirements

1. If the search text is fewer than 2 characters, then should return no results. (It is an optimization feature of the search functionality.)

2. If the search text is equal to or more than 2 characters, then it should return all the city names starting with the exact search text.

- For example for search text “Va”, the function should return Valencia and Vancouver

3. The search functionality should be case insensitive

4. The search functionality should work also when the search text is just a part of a city name

   - For example “ape” should return “Budapest” city

5. If the search text is a “*” (asterisk), then it should return all the city names.

Kata 5: Point of sale kata

Create a simple app for scanning bar codes to sell products.

Requirements

1. Barcode ‘12345’ should display price ‘$7.25’

2. Barcode ‘23456’ should display price ‘$12.50’

3. Barcode ‘99999’ should display ‘Error: barcode not found’

4. Empty barcode should display ‘Error: empty barcode’

5. Introduce a concept of `total` command where it is possible to scan multiple items. The command would display the sum of the scanned product prices

Kata 6: Banking kata

Note: This is an advanced example where the solution requires knowledge of using a mocking framework. The possible solution can also have an elaborated design. Solve it only if you feel comfortable with mocking frameworks and designing your code.

Create a simple bank application with features of depositing, withdrawing, and printing account statements.

Constraints

1. Start with a class with the following structure

```
public class Account {
  public void deposit(int amount)
  public void withdraw(int amount)
  public void printStatement()
}
```

2. You are not allowed to add any other public methods in this class

3. Use Strings and Integers for dates and amounts (keep it simple)

4. Don't worry about the spacing in the statement printed in the console

Requirements

1. Deposit into Account

2. Withdraw from an Account

3. Print the Account statement to the console

```
DATE       | AMOUNT  | BALANCE
10/04/2014 | 500.00  | 1400.00
02/04/2014 | -100.00 | 900.00
01/04/2014 | 1000.00 | 1000.00
```