Code Craft Kick-Start

Exercises & Links

Day #1

TDD

Exercise #1

The Mars Rover Kata (simplified)

Test-drive code to drive a rover over the idealised surface of Mars, represented as a 2-D grid of x, y coordinates (e.g., [4, 7]).

The rover can be "dropped" on to the grid at a specific set of coordinates, facing in one of four directions: North, East, South or West.

Sequences of instructions are sent to the rover telling it where to go. This sequence is a string of characters, each representing a single instruction:

- R = turn right
- L = turn left
- F = move on square forward
- B = move one square back

The rover ignores invalid instructions.

How to tackle this exercise:

1. *Make a test list for your rover based on these requirements*
2. *Work through your test list on test case at a time, remembering to Red-Green-Refactor with each test*
3. *When refactoring, don't forget that test code has to be maintained, too!*

Links

The complete Mars Rover TDD kata is described at https://kata-log.rocks/mars-rover-kata

Refactoring
Exercise #2

Mars Rover refactored

Carefully refactor your Mars Rover code to remove code smells and make it as easy to understand and easy to change as you can.

If your code isn't where you'd like to start (e.g., if it's already perfect!), clone our .NET solution files using the link below.

How to tackle this exercise:

1. *Use version control, and commit whenever your tests pass.*
2. *If you mess up, revert the code to the previous working version.*
3. *Make sure you're seeing your tests pass frequently.*
4. *Try to use automated refactorings and other IDE shortcuts where possible.*
5. *If you're using a stripped-down editor with limited refactoring support, be extra disciplined about taking baby steps, performing one refactoring (Rename, Extract Method, Introduce Parameter, etc) at a time and making sure you run the tests whenever you believe they should pass.*

Links

Source code - https://github.com/jasongorman/java_refactoring

Design Principles

Exercise #3

CD Warehouse

Test-drive some code that satisfies the following requirements:

```
Customers can buy CDs, searching on the title and the
artist. Their credit card payment is processed by an external provider.
Record labels send batches of CDs to the warehouse. Keep a
stock count of how many copies of each title are in the warehouse.

Customers can leave reviews for CDs they've bought through the
warehouse, which gives each title an integer rating from 1- 10 and
the text of their review if they want to say more.
```

How to tackle this exercise:

1. *Identify the user actions we need to test, and make a Test List for each of them to drive your TDD*
2. *Run the requirements through a tag cloud generator to build a customer word list to use as inspiration for names in your code*
3. *At every refactoring step, carefully review your code to ensure it follows the principles of Simple Design, Modular Design and that it has a strong conceptual correlation with the customer's language*
4. *Don't forget to apply the disciplines of TDD and refactoring that we've learned today!*

Links

Tag Cloud Generator - https://www.wordclouds.com/