zenika-formation-tdd

Labs

Prerequisite

Tools

- Java JDK (http://www.oracle.com/technetwork/java/javase/downloads/index.html)
- Maven 3.3.x (https://maven.apache.org/download.cgi)
- Eclipse STS (http://spring.io/tools/sts/all)
  - Includes Maven and some configured plugins (m2e, Git, ...).
- Firefox (https://www.mozilla.org/fr/firefox/new/)
  - Chrome may be used

Configuration

Encoding

Make sure UTF-8 is the default encoding (General->Workspace::Text file encoding)

Automatic static imports

- `Window → Preferences`
- `Java → Editor → Content Assist → Favorites`
- New Type... :
- `org.junit.Assert`
- `org.hamcrest.Matchers`
- `org.mockito.Mockito`
- `org.mockito.Matchers`
- ...

Advices

`Alt` + `Shift` + `X` + `T` : Run unit tests.

Lab 1 : Fizz buzz

1. Goal

Learn TDD technics with a simple exercise : [FizzBuzz (https://en.wikipedia.org/wiki/Fizz_buzz)](https://en.wikipedia.org/wiki/Fizz_buzz).

2. Steps

- Import the Maven project.
- Check that the test is failing and write code to fix it.
- Use TDD to solve Fizzbuzz. Check TODO.md file to know what to implement.
- Don't limit to write tests first but use some TDD principles `Fake it` , `Triangulation` , `Obvious Method` and refactoring `Extract Method` , `Hide Method` .

Lab 2 : Tennis

1. Goal

Improve your TDD skills by solving a more complex exercise and discover AssertJ.

2. Steps

- Import the Maven project.
- Read README.md
- Solve the kata using TDD.
- Be mindful of the steps order
- Rollback if a step is too hard to implement.

Lab 3 : Legacy code & refactoring - Add new functionality with tests

1. Goals

Learn :

- Add tests on existing untested code.
- Break dependencies.
- Add a new functionality using TDD.

2. Steps

- Assumption : Code is known well enough to add tests.
- With the provided code, follow these steps :
  - Break dependency with the `System` class.
  - Add tests.
  - Check all tests pass.
  - Refactor using SOLID principles (especially SRP).
  - Using TDD, add a functionality that allows the use of multiple hashing algorithms.
- Information : There is a `main` method within `SecurityManagerTest`.

Lab 4 : Legacy code & refactoring - Sprout Method

1. Goals

Learn to :

- Add a new functionality into untested code.
- Use the *Sprout method* pattern.

2. Steps

- Assumption : Impossible to add tests to the existing code (not enough time, too hard).
- With the provided code, follow these steps :
  - After data validation, add a method call to dispatch user information to a third party application.
  - Do not modify any code outside of this call.
  - Using TDD, code the call to the third party application (using mocks or stubs).
- Information : There is a `main` method within `SecurityManagerTest` .

Lab 5 : Legacy code & refactoring - Wrap Class

1. Goals

Learn to :

- Add new functionalities into untested code.
- Use the *Wrap class* pattern.

2. Steps

- Assumption : Impossible to had tests to the existing code (not enough time, too hard).
- With the provided code, follow these steps :
    - Create a new class or method for the new functionality.
    - Do not modify any existing code.
    - Using TDD, code the call to the third party application (using mocks or stubs).
- Information : There is a `main` method within `SecurityManagerTest`.
- Context : the method to extend is an instance method (non static method).

Lab 6 : BDD with Cucumber

1. Goals

Learn to :

- Add a new functionality using Cucumber.

2. Steps

- Read `kata-potter-enonce.md` .
- Define the first use case into the `basket_price.feature` file.
- Code the associated steps into the `BasketPriceStepDefs` class (`@Given` , `@When` , `@Then`).
- Check that the test fails.
- Implement the code using TDD.
  - A functionality may involve multiple classes / methods. Thus, multiple TDD iterations are needed to pass BDD test(double cycle).