Applicaiton répondeur en TDD

Julien Fredon

Version 1.0, 2023-01-30

Table of Contents

2. Objectifs du TP

- travailler sur un projet en utilisant la méthode Test Driven Development
- identifier et différencier ce qui relève du métier de ce qui relève du *plumbing*

3. Rappel des trois lois du TDD :

1. **Loi no 1** : Écrivez un test qui échoue avant d'écrire le code de production correspondant.
2. **Loi no 2** : Écrivez une seule assertion à la fois, qui fait échouer le test ou qui échoue à la compilation.
3. **Loi no 3** : Écrivez le minimum de code de production pour que l'assertion du test actuellement en échec soit satisfaite.

4. Programme Ohce

4.1. Description de l'application

ohce est une application s'exécutant en mode console qui répond l'inverse de ce que l'utilisateur saisit dans le terminal.

Même si cela semble une application stupide, ohce sait faire une chose ou deux.

Lorsque vous démarrez ohce en lui passant votre nom en paramètre de la ligne de commande, il vous salue différemment selon l'heure, mais uniquement en espagnol :

Entre 20h et 5h59, ohce vous accueillera en disant :
¡Buenas noches <votre nom>!

Entre 6h et 11h59 heures, ohce vous accueillera en disant :
¡Buenos días <votre nom>!

Entre 12h et 19h59 heures, ohce vous accueillera en disant :
¡Buenas tardes <votre nom>!

Lorsque vous saisissez un palindrome sur la ligne de commande, ohce l'aime et après avoir répondu en l'inversant, il ajoute :
¡Bonita palabra!

ohce sait quand s'arrêter, il suffit d'écrire *Stop!* et il vous saluera d'un
Adios <votre nom>. et se terminera.

Les données saisies par l'utilisateur ne seront pas stockées en base de données MySQL.

4.2. Exemple d'exécution

En considérant le programme lancé à 10h…

```
$ php ohce.php Madeline
> ¡Buenos días Madeline!
$ hola
> aloh
$ ressasser
> ressasser
> ¡Bonita palabra!
$ stop
> pots
$ Stop!
> Adios Madeline.
```

5. Travail à réaliser

5.1. Analyse

Au lieu de débuter par la création du fichier `ohce.php` puis développer le code qui va récupérer une saisie utilisateur et dérouler ainsi notre développement, nous allons tout d'abord analyser le programme à partir de sa description et faire la distinction entre le métier et le *plumbing*.

Cette étape va nous permettre de nous représenter mentalement les potentiels composants ou briques logicielles qui vont composer notre programme et nous allons débuter par ces dernières.
Nous complèterons le programme petit à petit en faisant interragir nos briques et nous terminerons pas la saisie utilisateur.

5.2. Première brique : la salutation

Une partie des besoins exprimés concernent ce qu'on pourrait nommer des salutations. Nous allons donc développer en suivant la méthodologie TDD, un composant `Greetings`.

Développez, au fur-et-à-mesure les tests suivants.
Vous devez faire passer le test le plus vite possible et vous ne pouvez développer le code du programme uniquement si un test le nécessite et que les précédents passent tous.

- should_return_buenas_noches_madeline_between_20h_and_6h
- should_return_buenos_dias_madeline_between_6h_and_12h
- should_return_buenas_tardes_madeline_between_12h_and_20h
- should_return_buenas_noches_prosper_between_20h_and_6h
- should_return_buenos_dias_prosper_between_6h_and_12h
- should_say_adios_madeline
- should_say_adios_prosper

Vous pouvez compléter cette liste avec des tests qui permettent de vérifier les cas particuliers (exemple quelle salutaion lorsqu'il est 19h 59 minutes et 59 secondes) .

5.3. Deuxième brique : le moteur de réponse

L'autre élément et peut être le plus important c'est le composant qui gère les réponses, nous nommerons ce composant `AnsweringMachine` puisque sa responsabilité sera de formaliser une réponse en fonction d'un mot ou d'une expression.

Créer une nouvelle classe de test avant de développer notre classe. Les tests à réaliser seront :

- should_return_aloh_to_hola
- should_return_girafe_to_efarig
- should_return_ressasser_and_bonita_palabra_to_ressasser

5.4. Le composant de commande (sorte de contrôleur)

Nous allons à présent développer la commande, ce sera le composant responsable des traitements et des appels aux autres composants.

Comme notre programme fonctionne en ligne de commande, nous ne parlons pas de contrôleur dans ces cas là mais de commande.

Tout comme le contrôleur, la commande n'est plus véritablement du code métier, il s'agit plus de *plumbing*, du code pour faire dialoguer les autres composants et recevoir les saisies utilisateurs et à cause de cela nous ne ferons pas de tests dans un premier temps (les parties importantes de notre programme étant déjà testées).

Créez la nouvelle classe `Ohce` et créer également un fichier `ohce.php` à la racine du projet. Ce fichier permettra de lancer notre application ainsi :

ohce.php

```
<?php

use Ohce\Ohce;
use Ohce\Greetings;
use Ohce\AnsweringMachine;

require_once "vendor/autoload.php";

if($argc <2){
    echo "usage: php ohce.php <username>\n";
    exit(1);
}

$username = $argv[1];
$greeting = new Greetings($username, new DateTimeImmutable('now'));
$answeringMachine = new AnsweringMachine();

Ohce::runFromCli($greeting, $answeringMachine);
exit(0);
```

6. Évolution de notre programme

À présent, nous devons également pouvoir exécuter le programme en lui précisant un fichier contenant la liste de mots à traiter.

Le fichier `ohce.php` peut donc prendre un paramètre supplémentaire : le nom de fichier qui contiendra la liste de mots et le programme sera lancé avec la commande :
`php ohce.php Madeline input.txt`

ohce.php

```php
<?php

use Ohce\Ohce;
use Ohce\Greetings;
use Ohce\AnsweringMachine;

require_once "vendor/autoload.php";

if($argc <2){
    echo "usage: php ohce.php <username>\n";
    exit(1);
}

if($argc === 2){
    $username = $argv[1];
    $greeting = new Greetings($username, new DateTimeImmutable('now'));
    $answeringMachine = new AnsweringMachine();

    Ohce::runFromCli($greeting, $answeringMachine);
    exit(0);
}
if ($argc === 3) {
    $username = $argv[1];
    $fileName = $argv[2];
    $greeting = new Greetings($username, new DateTimeImmutable('now'));
    $answeringMachine = new AnsweringMachine();

    Ohce::runFromFile($greeting, $answeringMachine, $fileName);
    exit(0);
}
```

Développez la nouvelle méthode `runFromFile` de la classe `Ohce`.

6.1. Refactoring

6.1.1. Suppression du code dupliqué

La classe `Ohce` contient à présent deux méthodes `runFromCli` et `runFromFile` qui se ressemblent.

Remodelez cette classe pour ne faire plus qu'une méthode `run`. Vous devrez développer deux nouveaux composants représentant les modes de saisies (par l'utilisateur et par fichier) et vous devrez peut-être regarder ce que l'on nomme les **Générateurs** et modifier le fichier `ohce.php` mais vous ne devriez pas avoir à modifier le cœur de l'application, c.à.d. les classes métiers.

6.1.2. Meilleure organisation de nos fichiers

Vos classes n'ont pas les mêmes responsabilités mais elles sont toutes rangées dans le même

répertoire, cela pourrait être gênant si le programme se développe encore.

Réusinez également votre code afin de mieux ranger vos classes :

- déplacez la classe `Ohce` dans le namespace `Ohce\Command`,
- déplacez les classe `AnsweringMachine` et `Greetings` dans le namespace `Ohce\Service`,