AVIGNON UNIVERSITÉ

Rapport TP6

Tester une implémentation

Étudiant
Michel Marie LAMAH

19 avril 2023

Master Informatique
Intelligence Artificielle

UE Génie logiciel avancé
ECUE Techniques de test

Responsables
Daniel SALAS
Emmanuel FERREIRA

UFR
SCIENCES
TECHNOLOGIES
SANTÉ

AVIGNON UNIVERSITÉ
CERI

CENTRE
D'**ENSEIGNEMENT**
ET DE **RECHERCHE**
EN **INFORMATIQUE**
ceri.univ-avignon.fr

Sommaire

1 Introduction

Dans cet TP[1], il sera question de tester une implémentation fournie.
Pour réaliser ce travail, j'utiliserai :

- **IDE** : Intellij IDEA [1]
- **Dépôt de repository Git** : Github [2]
- **JUnit** : pour les tests unitaires

Toutes les modifications réalisées ici se trouvent sur la branche **TP06** de mon repository [3].

Mais si vous souhaitez voir toutes les modifications effectuées depuis le TP1 jusqu'à maintenant vous pouvez voir la branche master [2].

2 Configuration Pom

Pour la réalisation de ce TP, il a été demandé de d'ajouter la dépendance ci-dessous :

```
<dependency>
    <groupId>org.apache.commons</groupId>
    <artifactId>commons-collections4</artifactId>
    <version>4.0</version>
</dependency>
```

Listing 1. Dépendance à ajouter dans le fichier pom

La raison derrière cette modification est du au faite que l'implémentation fournie utilise des dictionnaires notamment **Map** et **HMap** pour créer ses données et **commons-collections4** est justement la bibliothèque en Java qui traite des structures de données.

3 Implémentation fournie

Ci-dessous le code de l'implémentation fournie.

```
<dependency>
    <groupId>org.apache.commons</groupId>
    <artifactId>commons-collections4</artifactId>
    <version>4.0</version>
</dependency>
```

Listing 2. Code de l'implémentation

```
@Override
public Pokemon createPokemon(int index, int cp, int hp, int dust, int candy) {
   String name;
   if(!index2name.containsKey(index)) {
      name = index2name.get(0);
   } else {
      name = index2name.get(index);
   }
   int attack;
   int defense;
   int stamina;
   double iv;
   if(index < 0) {
      attack = 1000;
      defense = 1000;
      stamina = 1000;
```

1. https://www.jetbrains.com/fr-fr/idea/
2. https://github.com/

```
17      iv = 0;
18    } else {
19      attack = RocketPokemonFactory.generateRandomStat();
20      defense = RocketPokemonFactory.generateRandomStat();
21      stamina = RocketPokemonFactory.generateRandomStat();
22      iv = 1;
23    }
24    return new Pokemon(index, name, attack, defense, stamina, cp, hp, dust, candy, iv);
25  }
```

Listing 3. Code de l'implémentation

Cette implémentation utilise le dictionnaire ci-dessous pour générer la métadonnée **name**.

```
1  static {
2    Map<Integer, String> aMap = new HashMap<Integer, String>();
3     aMap.put(-1, "Ash's Pikachu");
4     aMap.put(0, "MISSINGNO");
5     aMap.put(1, "Bulbasaur");
6     //TODO : Gotta map them all !
7     index2name = UnmodifiableMap.unmodifiableMap(aMap);
8  }
```

Listing 4. Le dictionnaire de nom utilisé

Passant au crible cette implémentation à travers mes tests plusieurs observations ont été faites :

- **Points positifs**
  - Mes tests passent pour les index corrects (entre 0 et 150), voir Ligne 18 - 23 du code LISTING 3
  - Cette implémentation génère aléatoirement les méta-données sans passer par **PokemonMetadataProvider** donc diminue les dépendances, voir Ligne 6 - 8 du code LISTING 3
- **Points négatifs**
  - Mes tests ne passent pas pour les index incorrects (inférieur à 0 et supérieur à 150) mais aussi pour les index n'existant pas.
  - Certes le fait de générer les méta-données directement dans l'implémentation permet de diminuer les dépendances par contre dans le cas où plusieurs implémentations utiliseraient ce code, on aurait beaucoup de code dupliqué raison pour laquelle dans mon implémentation j'utilisais le **PokemonMetadatProvider**

4 Proposition d'amélioration

Conformément aux problèmes relevés précédemment, il est claire que l'amélioration consistera à gérer les cas des index incorrects.

Mais bien avant celà, je vous présente l'implémentation que j'avais fait qui gérait tout ces cas en utilisant PokemonMetadataProvider qui obtenir les métadonnées.

```
1  @Override
2  public Pokemon createPokemon(int index, int cp, int hp, int dust, int candy){
3     PokemonMetadataProvider metadataProvider = new PokemonMetadataProvider();
4     String name;
5     int attack;
6     int defense;
7     int stamina;
8     double iv;
9
10    PokemonMetadata metadata;
```

```
11
12      try {
13          metadata = metadataProvider.getPokemonMetadata(index);
14          name = metadata.getName();
15          attack = metadata.getAttack();
16          defense = metadata.getDefense();
17          stamina = metadata.getStamina();
18          iv = (index == 0)?0.56:1;
19
20          return new Pokemon(index,name,attack,defense,stamina,cp,hp,dust,candy,iv);
21      } catch (PokedexException e) {
22          throw new RuntimeException(e.getMessage());
23      }
24  }
```

Listing 5. Mon implémentation de PokemonFactory

Dans cette implémentation, j'utilisais une implémentation de MetaDataProvider pour récupérer les métadonnées et c'est cette implémentation qui gérait les cas des index incorrects, ci-dessous son code.

```
1   @Override
2   public PokemonMetadata getPokemonMetadata(int index) throws PokedexException {
3       if (index < 0 || index > 150){
4           throw new PokedexException("Invalid ID");
5       }else{
6           for (PokemonMetadata pokemonMetadata:metadataList) {
7               if(pokemonMetadata.getIndex() == index){
8                   return pokemonMetadata;
9               }
10          }
11      }
12      throw new PokedexException("Ce Pokemon n'existe pas");
13  }
```

Listing 6. Mon implémentation de MetaDataProvider

Maintenant en ce qui concerne l'amélioration de l'implémentation fournit je vais rester dans leur idée tout en gérant les cas des index incorrects.

```
1   @Override
2   public Pokemon createPokemon(int index, int cp, int hp, int dust, int candy) {
3      String name;
4      int attack;
5      int defense;
6      int stamina;
7      double iv;
8      try {
9         if(index < 0 || index > 150){
10           throw new PokedexException("Index Invalid");
11        }else if(!index2name.containsKey(index)){
12           throw new PokedexException("Ce pokemon n'existe pas");
13        }
14        name = index2name.get(index);
15        attack = RocketPokemonFactory.generateRandomStat();
16        defense = RocketPokemonFactory.generateRandomStat();
17        stamina = RocketPokemonFactory.generateRandomStat();
18        iv = 1;
19     } catch (PokedexException e) {
20        throw new RuntimeException(e.getMessage());
21     }
22     return new Pokemon(index, name, attack, defense, stamina, cp, hp, dust, candy, iv);
```

```
23 }
```

Listing 7. Mon implémentation de MetaDataProvider

Avec cette implémentation, mes tests unitaires passaient.

5 Conclusion

Si jusque là, on ne faisait que tester ce que nous codons, dans cet tp, il était question de tester un code développer par une autre personne et de l'améliorer afin que nos testes puissent passer un peu comme le développement orienté test.

Ce tp marque la fin des tps de techniques de tests, à vrai dire avant je ne pensais vraiment pas à tout ce qui est test mais à travers ces 6 tps, j'ai pu savoir comment ça pouvait aider à minimiser les risques et des dépenses inutiles (même si ici nous n'avons fait aucune dépense).

A travers ces TPs, j'ai aussi pu apprendre beaucoup de chose sur les outils de tests comme JUnit et Mockito, de qualimétrie comme checkstyle et Jacoco mais aussi de couverture de code comme CodeCov tout en passant par l'outil d'intégration continue comme CircleCI.

Références

[1] Daniel SALAS. *TP6 : Tester une implémentation*. Université Avignon. 2023. url : https://github.com/Youkoulanda/ceri-m1-techniques-de-test/blob/master/TPs/TP6.md.

[2] LAMAH Michel Marie. *TP6 : Documentation [master]*. Université Avignon. 2023. url : https://github.com/MichelMarieLamah13/ceri-m1-techniques-de-test/tree/master.

[3] LAMAH Michel Marie. *TP6 : Tester une implémentation [TP06]*. Université Avignon. 2023. url : https://github.com/MichelMarieLamah13/ceri-m1-techniques-de-test/tree/TP06.