AVIGNON UNIVERSITÉ

Rapport tp6

Inès El Mahi

07/04/2024

Informatique
Master1 Intelligence Artificielle
ECUE Techniques de test

Responsables
Bonnefoy Ludovic

UFR
SCIENCES
TECHNOLOGIES
SANTÉ

AVIGNON UNIVERSITÉ
CERI

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

Sommaire

1 Objectif du tp

Dans notre TP, nous avons reçu une version de l'interface IPokemonFactory de la part de la Team Rocket. Notre but est de l'intégrer à notre propre code, puis de la tester à l'aide de nos tests existants,on ne se contente pas de vérifier si elle respecte les règles de base, mais on examine aussi la qualité du code, ses performances et sa facilité à être maintenu. En comprenant ses avantages et ses faiblesses, nous pourrons décider si nous l'adoptons dans notre projet.

2 Implémentation de RocketPokemonFactory

```java
package fr.univavignon.pokedex.imp;

import java.util.HashMap;
import java.util.Map;
import java.util.Random;

import org.apache.commons.collections4.map.UnmodifiableMap;

import fr.univavignon.pokedex.api.IPokemonFactory;
import fr.univavignon.pokedex.api.Pokemon;

public class RocketPokemonFactory implements IPokemonFactory {

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

   private static int generateRandomStat() {
      int total = 0;
      for(int i=0; i < 1000000; i++)
      {
         Random rn = new Random();
          int r = rn.nextInt(2);
          total = total + r;
      }
      return total / 10000;
   }

   @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;
```

```
46      double iv;
47      if(index < 0) {
48         attack = 1000;
49         defense = 1000;
50         stamina = 1000;
51         iv = 0;
52      } else {
53         attack = RocketPokemonFactory.generateRandomStat();
54         defense = RocketPokemonFactory.generateRandomStat();
55         stamina = RocketPokemonFactory.generateRandomStat();
56         iv = 1;
57      }
58      return new Pokemon(index, name, attack, defense, stamina, cp, hp, dust, candy,
            iv);
59   }
60 }
```

Listing 1. Implémentation de RocketPokemonFactory

2.1 Explication de l'implémentation

- **Importations :**
  - Les classes nécessaires sont importées, telles que 'Map', 'HashMap', 'Random', 'UnmodifiableMap', etc.
- **Initialisation de la carte 'index2name' :**
  - Une carte associant les index des Pokémon à leurs noms est initialisée, mais tous les noms ne sont pas encore mappés.
- **Méthode 'generateRandomStat()' :**
  - Cette méthode génère aléatoirement une statistique pour les attributs d'un Pokémon.
- **Méthode 'createPokemon()' :**
  - Cette méthode crée une instance de 'Pokemon' en fonction des paramètres passés. Elle utilise la carte 'index2name' pour obtenir le nom du Pokémon correspondant à l'index donné. Elle génère ensuite aléatoirement les statistiques d'attaque, de défense et de stamina si l'index est positif, sinon elle attribue des valeurs par défaut très élevées pour ces statistiques.

3 Configuration avant test

Tout d'abord j'ai pris l'implémentation de RocketPokemonFactory et je l'ai poser au bon endroit avec mes autres classes d'implémentations. Puis j'ai ajouter la dépendence **commons-collections4** de Apache dans mon fichier **pom.xml** comme suit :

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

    </dependencies>
```

(a) Ajout de la dépendance commons-collections4

4 Test de RocketPokemonFactory

J'ai tester sur **RockePokemonFactory** avec la classe **RocketPokemonFactoryTest** qui est exactement la meme que que la mienne **IPokemonFactoryTest**, juste que **RocketPokemonFactoryTest** test directement sur **RocketPokemonFactory** :

```java
package fr.univavignon.pokedex.api;

import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.Test;
import static org.junit.jupiter.api.Assertions.*;

public class RocketPokemonFactoryTest{

    private RocketPokemonFactory pokemonFactory;
    private Pokemon bulbizarre;
    private Pokemon aquali;

    @BeforeEach
    void setUp() {
        pokemonFactory = new RocketPokemonFactory();

        bulbizarre = new Pokemon(1, "Bulbasaur", 126, 100, 90, 600, 100, 4000, 4, 1.0);
        aquali = new Pokemon(133, "Vaporeon", 186, 168, 260, 2729, 202, 5000, 4, 1.0);
    }

    @Test
    void testCreationPokemon() {
        Pokemon bulbasaur = pokemonFactory.createPokemon(1, 600, 100, 4000, 4);
        assertNotNull(bulbasaur, "Bulbasaur ne devrait pas être null");
        assertEquals(1, bulbasaur.getIndex(), "L'index de Bulbasaur devrait
            correspondre");
        assertEquals("Bulbasaur", bulbasaur.getName(), "Le nom de Bulbasaur devrait
            correspondre");
        assertEquals(600, bulbasaur.getCp(), "Les CP de Bulbasaur devraient
            correspondre");
        assertEquals(100, bulbasaur.getHp(), "Les HP de Bulbasaur devraient
            correspondre");
        assertEquals(1.0, bulbasaur.getIv(), "L'IV de Bulbasaur devrait correspondre");
        assertEquals(4000, bulbasaur.getDust(), "La poussière du Pokémon devrait être
            de 4000");

        Pokemon vaporeon = pokemonFactory.createPokemon(133, 2729, 202, 5000, 4);
        assertNotNull(vaporeon, "Vaporeon ne devrait pas être null");
        assertEquals(133, vaporeon.getIndex(), "L'index de Vaporeon devrait
            correspondre");
        assertEquals("Vaporeon", vaporeon.getName(), "Le nom de Vaporeon devrait
            correspondre");
        assertEquals(2729, vaporeon.getCp(), "Les CP de Vaporeon devraient
            correspondre");
        assertEquals(202, vaporeon.getHp(), "Les HP de Vaporeon devraient
            correspondre");
        assertEquals(1.0, vaporeon.getIv(), "L'IV de Vaporeon devrait être parfait");
        assertEquals(5000, vaporeon.getDust(), "La poussière du Pokémon devrait être
            de 5000");
    }

```

```
42      @Test
43      void testConsistencyOfPokemonMetadata() {
44          PokemonMetadata expectedMetadata = new PokemonMetadata(1, "Bulbasaur", 126,
                100, 90);
45          Pokemon createdPokemon = pokemonFactory.createPokemon(1, 600, 100, 4000, 4);
46          assertEquals(expectedMetadata.getIndex(), createdPokemon.getIndex(), "L'index
                des métadonnées devrait correspondre");
47          assertEquals(expectedMetadata.getName(), createdPokemon.getName(), "Le nom
                devrait correspondre");
48          assertEquals(expectedMetadata.getAttack(), createdPokemon.getAttack(),
                "L'attaque devrait correspondre");
49          assertEquals(expectedMetadata.getDefense(), createdPokemon.getDefense(), "La
                défense devrait correspondre");
50          assertEquals(expectedMetadata.getStamina(), createdPokemon.getStamina(),
                "L'endurance devrait correspondre");
51      }
52
53
54      @Test
55      void testPokemonCreationWithInvalidValues() {
56          assertThrows(IllegalArgumentException.class, () ->
                pokemonFactory.createPokemon(1, -1, -1, -1, -1),
57                  "Devrait lancer IllegalArgumentException pour des valeurs invalides");
58      }
59
60
61      @Test
62      void testCreationDePokemonRenvoieNull() {
63          assertNull(pokemonFactory.createPokemon(-2, 10000, 10000, 10000, 10000), "La
                création d'un Pokemon avec des paramètres invalides devrait renvoyer null");
64      }
65
66
67      @Test
68      void testPokemonCandyGetter() {
69          assertEquals(4, bulbizarre.getCandy(), "Le nombre de candies de Bulbasaur
                devrait être correct");
70          assertEquals(4, aquali.getCandy(), "Le nombre de candies de Vaporeon devrait
                être correct");
71      }
72  }
```

Listing 2. Implémentation de RocketPokemonFactoryTest

4.0.1 Explication des tests

En lançant mes test j'ai eu deux erreurs majeurs qui ce sont révéler :

Erreur de création de Pokémon avec des paramètres invalides (testCreationDePokemonRenvoieNull) :
Montre que le test s'attend à ce que la méthode de création d'un Pokémon retourne null lorsque des paramètres invalides sont fournis. Cependant, la méthode a retourné une instance de Pokemon au lieu de null.

```
org.opentest4j.AssertionFailedError: La création d'un Pokemon avec des paramètres invalides devrait renvoyer nul
Expected :null
Actual   :fr.univavignon.pokedex.api.Pokemon@770c2e6b
<Click to see difference>

 <4 internal lines>
    at fr.univavignon.pokedex.api.RocketPokemonFactoryTest.testCreationDePokemonRenvoieNull(RocketPokemonFactory
    at java.base/java.util.ArrayList.forEach(ArrayList.java:1596) <9 internal lines>
    at java.base/java.util.ArrayList.forEach(ArrayList.java:1596) <27 internal lines>
```

(a)

Erreur de lancement d'exception pour la création de Pokémon avec des valeurs invalides (testPokemonCreationWithInvalidValues) :

Cela indique que lorsqu'un test essaie de créer un Pokémon avec des valeurs invalides, il s'attend à ce qu'une IllegalArgumentException soit lancée. Cependant, aucune exception n'a été lancée.

```
ntest4j.AssertionFailedError: Devrait lancer IllegalArgumentException pour des valeurs invalides ==> Expected ja
ernal lines>
fr.univavignon.pokedex.api.RocketPokemonFactoryTest.testPokemonCreationWithInvalidValues(RocketPokemonFactoryTes
java.base/java.util.ArrayList.forEach(ArrayList.java:1596) <9 internal lines>
java.base/java.util.ArrayList.forEach(ArrayList.java:1596) <27 internal lines>
```

(a)

4.1 Résolution des problèmes :

Pour résoudre ces problèmes, nous devrions :

Corriger la méthode generateRandomStat :
en remplaçant l'implémentation actuelle de la méthode generateRandomStat pour qu'elle génère un nombre aléatoire valide entre 0 et DEFAULT_STAT_VALUE.

```
Random rn = new Random();
   int total = 0;
   for (int i = 0; i < 1000000; i++) {
       total += rn.nextInt(2);
   }
   return total / 10000;
}
```

Ajouter une validation dans la méthode createPokemon :

```
public Pokemon createPokemon(final int index, final int cp, final int hp, final int
    dust, final int candy) {
   if (index < 0 || cp < 0 || hp < 0 || dust < 0 || candy < 0) {
       return null;
   }
}
```

Lancer une exception pour des valeurs vraiment invalides :

```
if (cp <= MINIMUM_VALID_CP || hp <= MINIMUM_VALID_HP) {
   throw new IllegalArgumentException("Les valeurs de CP et de HP sont trop basses.");
}
```

5 Test de la qualité avec checkstyle

Le rapport de qualité généré par Checkstyle pour la classe RocketPokemonFactory identifie plusieurs erreurs, notamment des problèmes de documentation manquante, des espaces manquants, des nombres magiques non définis comme des constantes, des lignes de code trop longues et des paramètres non déclarés comme finaux. Ces erreurs sont des points à corriger pour améliorer la lisibilité, la maintenabilité et la qualité globale du code.

- **Ligne 14, colonne 5 :** Commentaire Javadoc manquant.
- **Ligne 26, colonne 9 :** Il manque une espace après 'for'.
- **Ligne 26, colonne 9 :** Il manque une espace autour de 'for'.
- **Ligne 26, colonne 18 :** Il manque une espace après '='.
- **Ligne 26, colonne 18 :** Il manque une espace avant '='.
- **Ligne 26, colonne 26 :** '1000000' devrait être défini comme une constante.
- **Ligne 27, colonne 9 :** Accolade à la colonne 9 devrait être sur la ligne précédente.
- **Ligne 32, colonne 24 :** '10000' devrait être défini comme une constante.
- **Ligne 35, colonne 5 :** La classe 'RocketPokemonFactory' semble être conçue pour extension, mais la méthode 'createPokemon' n'a pas de Javadoc qui explique comment le faire en toute sécurité. Si la classe n'est pas conçue pour extension, envisagez de rendre la classe 'RocketPokemonFactory' finale ou de rendre la méthode 'createPokemon' static/final/abstract/empty, d'ajouter les annotations permises pour la méthode.
- **Ligne 36 :** La ligne excède 80 caractères (trouvé 82).
- **Ligne 36, colonne 34 :** Le paramètre index devrait être final.
- **Ligne 36, colonne 45 :** Le paramètre cp devrait être final.
- **Ligne 36, colonne 53 :** Le paramètre hp devrait être final.
- **Ligne 36, colonne 61 :** Le paramètre dust devrait être final.
- **Ligne 36, colonne 71 :** Le paramètre candy devrait être final.
- **Ligne 38, colonne 9 :** Il manque une espace après 'if'.
- **Ligne 38, colonne 9 :** Il manque une espace autour de 'if'.
- **Ligne 47, colonne 9 :** Il manque une espace après 'if'.
- **Ligne 47, colonne 9 :** Il manque une espace autour de 'if'.
- **Ligne 48, colonne 22 :** '1000' devrait être défini comme une constante.
- **Ligne 49, colonne 23 :** '1000' devrait être défini comme une constante.
- **Ligne 50, colonne 23 :** '1000' devrait être défini comme une constante.
- **Ligne 58 :** La ligne excède 80 caractères (trouvé 91).

6 Correction envisagées pour la classe RocketPokemonFactory

La classe RocketPokemonFactory nécessite des améliorations pour garantir la qualité et la lisibilité du code. Cela inclut l'ajout de commentaires manquants, la correction des erreurs d'espacement, la définition des nombres magiques comme des constantes, et le respect de la limite de 80 caractères par ligne. De plus, la méthode generateRandomStat doit être révisée pour une génération efficace de nombres aléatoires, et la documentation de la méthode createPokemon doit être complétée pour une meilleure compréhension de son fonctionnement.

7 Conclusion

En conclusion, l'analyse de la classe RocketPokemonFactory révèle plusieurs aspects à améliorer pour garantir sa qualité et sa maintenabilité. En corrigeant les erreurs de style, en ajoutant une documentation adéquate et en optimisant certaines parties du code,

nous pouvons rendre cette classe plus robuste et plus facile à comprendre. L'application de bonnes pratiques, ainsi que l'utilisation d'outils d'analyse statique comme Checkstyle, peuvent grandement contribuer à l'amélioration de la qualité globale du code.