CHEN Marc-Yong
Taleb Aslan

Rapport
JAVA
Projet Dictionnaire

Exercice 1 : Présentation générale

1.1

```
class DictionaryTest {

}
```

Création de la classe test DictionaryTest.

1.2

```
public class Dictionary {
    1 usage
    protected String name;
    1 usage
    protected Map<String, String> translate;
    no usages
    public Dictionary(String n)
    {
        this.name = n;
        translate = new HashMap<>();
    }
}
```

Création de la classe Dictionary avec comme attribut un String name et un Hashmap translate qui stocke toutes les traductions qui contient comme type de clé String et type de valeur String avec un constructeur qui prend en paramètre un String pour name et permet d'initialiser translate.

1.3

```java
public String getName()
{
    return this.name;
}
```

Méthode getName qui renvoie le nom du dictionnaire.

1.4

```java
class DictionaryTest {
    2 usages
    Dictionary DictionaryUnderTest;

    no usages
    @BeforeEach
    void setup()
    {
        DictionaryUnderTest = new Dictionary( n: "DictionaryUnderTest");
    }
    no usages
    @Test
    void testName()
    {
        assertEquals( expected: "DictionaryUnderTest", DictionaryUnderTest.getName());
    }
}
```

On crée un objet Dictionary “DictionaryUnderTest” que l’on initialise dans la méthode setup, ensuite on fait un Test “testName” qui teste si le nom du dictionnaire est bien le bon.

1.5

```java
@Test
void testIsEmpty()
{
    assertEquals( expected: true, DictionaryUnderTest.translate.isEmpty());
}
```

Nouveau test qui vérifie si notre dictionnaire est bien vide. Pour cela on regarde si notre Hashmap translate est vide en utilisant la méthode de base isEmpty() qui renvoie un booléen true si le Hashmap est vide sinon il renvoie false.

Exercice 2 : Ajout des traductions

2.1

```java
@Test
void testTranslation()
{
    String word = "Bonjour";
    String translation = "Hello";
    DictionaryUnderTest.addTranslation(word, translation);
    assertEquals(translation, DictionaryUnderTest.getTranslation(word));
}
```

On a deux String, le premier “word” est le mot à traduire et “translation” est la traduction. On utilise la méthode addTranslation(String, String) qui ajoute une nouvelle traduction dans notre Hashmap. On utilise la méthode getTranslation(String) qui va prendre en paramètre un String qui sera la clé dans notre Hashmap est va retourner la valeur de la clé correspondante, qui est la traduction du mot. On fait ensuite un assertEquals avec translation et getTranslation(word).

2.2

```java
public void addTranslation(String fr, String en)
{
    translate.put(fr, en);
}
```

```java
public String getTranslation(String word)
{
    if(translate.containsKey(word)) return translate.get(word);
    return null;
}
```

- Pour la méthode void addTranslation(String, String), on ajoute dans le Hashmap translate, le mot français comme clé et le mot anglais comme valeur.
- Pour la méthode String getTranslation(String), on prend en paramètre le mot dont on recherche la traduction, on cherche donc ce mot comme clé dans translate et on retourne la valeur de cette clé avec get(String) qui est la traduction du mot si elle existe sinon on renvoie null.

Exercice 3 : Traductions multiples

3.1

```java
@Test
void testMultipleTranslations()
{
    String word = "Bonjour";
    ArrayList <String> translations = new ArrayList<>();
    translations.add("Hello");
    translations.add("Hi");
    DictionaryUnderTest.addTranslation(word, translations);
    assertEquals(translations, DictionaryUnderTest.getMultipleTranslations(word));
}
```

On a toujours notre String “word” qui est le mot à traduire mais on a maintenant une Arraylist “translations” qui contient les multiples traductions du mot. On ajoute cette traduction avec addTranslation que l’on a modifié en conséquence en changeant le type addTranslation(String, Arraylist), de même pour l’attribut translate dans la classe Dictionnaire on passe d’un map<String, String> à map <String, ArrayList<String>>. On fait ensuite un assertEquals avec translation et getMultipleTranslations(word).

3.2

```
public ArrayList getMultipleTranslations(String word)
{
    return  translate.get(word);
}
```

Même principe que pour getTranslation sauf que l’on renvoie une ArrayList.

3.3

```
protected Map<String, ArrayList<String>> translate;
```

Changement de type dans le Hashmap :
Map<String, String> → Map<String, ArrayList<String>>

```
public void addTranslation(String fr, ArrayList en)
{
    translate.put(fr, en);
}
```

Changement de type dans les paramètres :
addTranslation(String, String) → addTranslation(String, ArrayList)

```java
public ArrayList getTranslation(String word)
{
    if (translation_fr_to_en.containsKey(word)) return translation_fr_to_en.get(word);
    return null;
}
```

Changement de type dans le retour de la méthode :
String getTranslation(String) → ArrayList getTranslation(String)

Exercice 4 : Traductions multiples bidirectionnelle

4.1 : test

```java
@Test
void testBidirectionality()
{
    String word_fr = "Bonjour";
    String word_fr2 = "Salut";
    String word_en = "Hello";
    String word_en2 = "Hi";
    ArrayList <String> translations_fr_en = new ArrayList<>();
    translations_fr_en.add(word_en);
    translations_fr_en.add(word_en2);
    ArrayList<String> translations_en_fr = new ArrayList<>();
    translations_en_fr.add(word_fr);
    translations_en_fr.add(word_fr2);
    DictionaryUnderTest.addTranslation(word_fr, translations_fr_en);
    DictionaryUnderTest.addTranslation(word_fr2, translations_fr_en);
    assertEquals(translations_fr_en, DictionaryUnderTest.getMultipleTranslations(word_fr));
    assertEquals(translations_en_fr, DictionaryUnderTest.getMultipleTranslations(word_en));
```

On teste la bidirectionnalité de notre dictionnaire, c'est à dire que la traduction s'effectue dans les deux sens par exemple, si on veut traduire "Bonjour" par "Hello" et "Hi", on retrouve dans notre dictionnaire la traduction de "Bonjour" par "Hello" et "Hi" et les traductions de "Hello" et "Hi", par "Bonjour".
Dans notre test on va traduire les mots "Bonjour" et "Salut" par "Hello" et "Hi", on a 4 String pour chaque mot, et une ArrayList qui contient les traductions. On utilise la méthode addTranslation pour ajouter les traductions dans notre dictionnaire, on a également une autre ArrayList qui contient les mots traduit en français. On teste la bidirectionnalité en faisant deux assertEquals, le premier qui vérifie la traduction du français vers l'anglais et le deuxième de l'anglais vers le français.

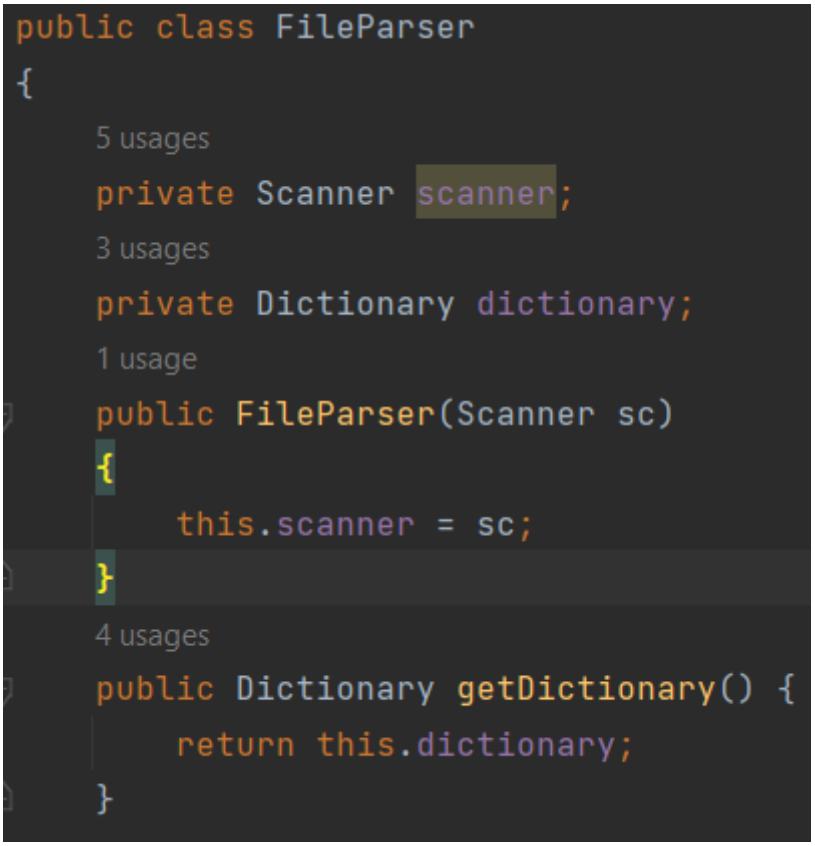
4.2

```
public void addTranslation(String fr, ArrayList en)
{
    if(!translation_fr_to_en.containsKey(fr))  translation_fr_to_en.put(fr, en);
    ArrayList<String> french = new ArrayList<>();
    french.add(fr);
    for(int i  = 0; i < en.size(); i++)
    {
        if(!translation_en_to_fr.containsKey(en.get(i)))
        {
                translation_en_to_fr.put((String) en.get(i), french);
        }
        else if (!translation_en_to_fr.get(en.get(i)).contains(fr))
        {
            translation_en_to_fr.get(en.get(i)).add(fr);
        }
    }
}
```

Modification de la méthode addTranslation

On modifie cette méthode afin de rajouter la bidirectionnalité. Pour ce faire, on va utiliser la deuxième Hashmap que l'on a créé, on fait une boucle sur le nombre d'éléments de notre ArrayList qui contient les mots en anglais et pour chaque mot on regarde si la Hashmap "translation_en_to_fr" contient déjà ce mot comme clé et sinon on l'ajoute avec le mot en français dans le Hashmap. Si le mot existe déjà alors on regarde la ArrayList associée à ce mot et on vérifie si le mot en français est dans cette arrayList ou non, si il ne le contient pas, on l'ajoute à l'arrayList. Maintenant quand on va ajouter une traduction d'un mot français vers une liste de mot anglais, chaque mot anglais aura la traduction vers le mot français.

Exercice 5 : Chargement de fichiers



```
public class FileParser
{
    5 usages
    private Scanner scanner;
    3 usages
    private Dictionary dictionary;
    1 usage
    public FileParser(Scanner sc)
    {
        this.scanner = sc;
    }
    4 usages
    public Dictionary getDictionary() {
        return this.dictionary;
    }
```

Class FileParser, qui possède comme attribut un Scanner et un dictionnaire avec un constructeur qui prend un scanner en paramètre.

```java
public void fillDictionary()
{
    String line;
    String list[];
    boolean first_word = true;
    dictionary = new Dictionary(scanner.nextLine());

    while(scanner.hasNextLine())
    {
        ArrayList <String> traduction = new ArrayList<>();
        line = scanner.nextLine();
        list = line.split( regex: ";");
        if (list.length >= 2)
        {
            String word = list[0];
            for (String trad : list)
            {
                if (!first_word)  traduction.add(trad);
                else first_word = false;
            }
            getDictionary().addTranslation(word, traduction);
            first_word = true;
        }

    }
    scanner.close();
}
```

Méthode fillDictionary, on récupère la première ligne de notre fichier txt qui est le nom du dictionnaire. Ensuite tant que le scanner n'est pas arriver à la fin du fichier, pour chaque ligne on stocke dans un tableau tous les String séparés par un ";". Pour chaque ligne après la première, le premier mot est le mot à traduire et les suivants sont les traductions. On vérifie que la list de mot est supérieur à 2 et on stocke tous les mots à par le premier dans une arrayList "traduction" et le premier mot dans un String word, ensuite on utilise addTranslation (word, traduction). Quand on arrive à la fin du fichier, on ferme le scanner.

```
class FileParserTest {
    6 usages
    FileParser fileParserUnderTest;
    3 usages
    @Mock
    Scanner mockScanner;
    no usages
    @BeforeEach
    public void setup() throws IOException {
        MockitoAnnotations.initMocks( testClass: this);
        fileParserUnderTest = new FileParser(mockScanner);
    }
```

Notre classe FileParserTest, on utilise un mock pour simuler un Scanner “mockScanner”. Dans notre méthode setup, on initialise notre mock et notre FileParser.

```
@Test
void testFillDictionary()
{
    Mockito.when(mockScanner.nextLine())
            .thenReturn( value: "Larousse")
            .thenReturn( value: "Bonjour;Hello")
            .thenReturn( value: "Salut;Hello;Hi");
    Mockito.when(mockScanner.hasNextLine())
            .thenReturn( value: true)
            .thenReturn( value: true)
            .thenReturn( value: false);

    fileParserUnderTest.fillDictionary();
    ArrayList<String> traduction_bonjour = new ArrayList<>();
    traduction_bonjour.add("Hello");

    ArrayList <String> traduction_salut = new ArrayList<>();
    traduction_salut.add("Hello");
    traduction_salut.add("Hi");
    assertEquals( expected: "Larousse", fileParserUnderTest.getDictionary().getName());
    assertFalse(fileParserUnderTest.getDictionary().isEmpty());
    assertEquals(traduction_bonjour, fileParserUnderTest.getDictionary().getMultipleTranslations( word: "Bonjour"));
    assertEquals(traduction_salut, fileParserUnderTest.getDictionary().getMultipleTranslations( word: "Salut"));
}
```

Dans le test, on simule la méthode nextLine() de notre mockScanner pour quel renvoie les données comme si elles étaient dans un fichier.

Ici les 3 premiers appels de cet fonction seront simulés, le premier renvoie le nom du dictionnaire ici “Larousse”, le deuxième et troisième renvoie des traductions de mot à ajouter dans le dictionnaire.
On fait de même pour la fonction hasNextLine() qui va renvoyer false au bout du troisième appel simulant la fin du fichier. On utilise la méthode fillDictionary, et on teste si le nom de notre dictionnaire est le bon, qu’il n’est pas vide ou bien que les traductions ce sont bien ajoutées dans le dictionnaire.