Génie Logiciel - Tests & Mocks

TAYLOR MATT / SOCHAJ YOANN

12.04.2021

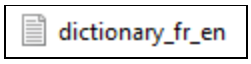
Sommaire:

I. Présentation du projet et guide d'utilisation

Ce projet consiste à créer un **dictionnaire bilingue**; l'objectif est de permettre des traductions dans un sens ou dans l'autre (ie: français vers anglais et anglais vers français).

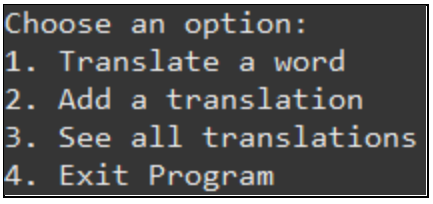
A chaque ajout de méthode nous utilisons des **tests JUnit** pour vérifier le bon fonctionnement de celles-ci et également tester les cas limites.

Le remplissage du dictionnaire peut se faire par l'intermédiaire d'un fichier où la première ligne contient le nom du dictionnaire (ie: Collins) et les lignes suivantes contiennent un caractère "**;**" séparant les traductions (ie: navire;ship).
Le fichier texte d'exemple s'appelle ***dictionary_fr_en.txt*** et se trouve dans le dossier source.



Voici un guide d'utilisation pour notre programme qui permet aussi de mettre en avant les différents ajouts que nous avons implémentés:

Au lancement du programme vous avez le choix entre **4 options**:



1. <u>Translate a word:</u>
Vous devez maintenant choisir entre une traduction français vers anglais ou l'inverse. Ensuite vous entrez le mot que vous voulez traduire dans la console, deux cas sont possibles:
- votre mot est dans le dictionnaire, la traduction est affichée.
- votre mot n'est pas le dictionnaire, un message correspondant est affiché.

```
Choose an option:
1. Translate a word
2. Add a translation
3. See all translations
4. Exit Program
1
1. French to English
2. English to French
1
Enter the word in French to translate :
navire
navire in English is [ship, boat]
```

2. <u>Add a translation:</u>
Entrez le mot d'abord en anglais puis la traduction en français.
- si le dictionnaire ne "connaît" pas cette traduction il va être ajouté dans le fichier.

```
20 arbre;tree
21 jambon;ham
22 ecran;screen
23 bois;wood
24 boite;box
25 biere;beer
```

- sinon un message d'erreur est affiché.

```
Enter the English word:
gold
Enter the French word:
or
The dictionary already knows this translation.
```

```
Choose an option:
1. Translate a word
2. Add a translation
3. See all translations
4. Exit Program
2
Enter the English word:
beer
Enter the French word:
biere
The translation: biere -> beer has been added to the dictionary.
```

- si on souhaite rajouter une deuxième traduction pour le même mot (traduction multiple) il va être rajouté à la suite séparé d'une virgule:

je souhaite ajouter la traduction **journey** pour le mot **voyage:**

```
8 voyage;trip
```

```
Enter the English word:
journey
Enter the French word:
voyage
The translation: voyage -> journey has been added to the dictionary.
```

la traduction est ajouté dans le fichier et la ligne correspondante est modifiée en conséquence:

```
8 voyage;trip,journey
```

3. <u>See all translations:</u>

Permet simplement d'afficher toutes les traductions dans la base de données.

```
Francais: [jambon] Anglais: [ham]
Francais: [or] Anglais: [gold]
Francais: [agent] Anglais: [officer]
Francais: [biere] Anglais: [beer]
Francais: [vert] Anglais: [green]
Francais: [chapeau] Anglais: [hat]
Francais: [argent] Anglais: [silver]
Francais: [erreur] Anglais: [mistake]
Francais: [voyage] Anglais: [trip, journey]
Francais: [chien] Anglais: [dog]
Francais: [arbre] Anglais: [tree]
Francais: [souris] Anglais: [mouse]
Francais: [bas] Anglais: [low]
Francais: [sauter] Anglais: [jump]
Francais: [journal] Anglais: [newspaper]
```

4. <u>Exit program:</u>

Quitte le programme.

II. Les différentes étapes du projet

Exercice 1:

Le premier test consiste à vérifier le **Getter** que nous avons implémenté pour renvoyer le nom du dictionnaire.

Dans l'image ci-dessous nous créons un dictionnaire en précisant le nom dans le constructeur puis nous utilisons la méthode ***assertEquals*** pour vérifier l'unicité des 2 paramètres (*"Collins"* et ***d1.getName()***).

```
@Test
public void returnCorrectName() {
    Dictionary d1 = new Dictionary("Collins");
    assertEquals("Collins", d1.getName());
}
```

Exercice 2:

Lors de cet exercice nous avons ajouté un HashMap en attribut du dictionnaire. Il va permettre de stocker les traductions uniques pour l'instant (c'est-à-dire un mot a une seule traduction).

```
public class Dictionary {
    private String name;
    public HashMap<String, String> translation;
```

Les tests réalisés sont les suivants:

- ***addTranslation()*** permet de vérifier que la traduction a bien été ajoutée dans le dictionnaire (HashMap) pour cela on regarde la taille du HashMap.

```
@Test
public void addTranslation() {
    Dictionary d1 = new Dictionary("Collins");
    d1.addTranslation("chapeau", "hat");
    assertEquals(1, d1.translation.size());
}
```

- ***returnCorrectTranslation()*** permet de vérifier notre méthode ***getTranslation(String word)*** qui nous retourne la traduction de “word”.

```
//Retrieve the translation from the Hash Map
public String getTranslation(String word) {
    return translation.get(word);
}
```

Le test est le suivant: on vérifie que la méthode retourne “hat” pour la traduction du mot “chapeau” après l’avoir ajouté dans notre dictionnaire.

```
@Test
public void returnCorrectTranslation() {
    Dictionary d1 = new Dictionary("Collins");
    d1.addTranslation("chapeau", "hat");
    assertEquals("hat", d1.getTranslation("chapeau"));
}
```

Exercice 3:

Dans cet exercice nous implémentons les traductions multiples en sens unique (c'est-à-dire un mot peut avoir plusieurs traductions).
Voici le test correspondant:

```
@Test
public void returnMultipleTranslations() {
    Dictionary d1 = new Dictionary("Collins");
    ArrayList<String> navireTranslations = new ArrayList<String>(2);
    navireTranslations.add("ship");
    navireTranslations.add("boat");
    d1.addTranslation("navire", navireTranslations);
    assertEquals(navireTranslations, d1.getMultipleTranslations("navire"));
}
```

Pour passer ce test nous avons eu besoin de modifier notre code. Nous utilisons plus un *HashMap<String, String>* mais ***HashMap<String, ArrayList<String>>*** ce qui permet d’avoir plusieurs traductions pour un seul mot. Les méthodes ***addTranslation()*** et ***getTranslation()*** ont aussi été modifiées pour s’adapter au nouveau HashMap:

- ***addTranslation()*** prend en paramètre une ArrayList<String> et non un String.

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

- ***getTranslation()*** retourne maintenant une ArrayList<String> et non un String.

```
public ArrayList<String> getTranslation(String word) {
    return translation.get(word);
}
```

Voici notre méthode ***getMultipleTranslations(String word)***:

Elle parcourt les clés du HashMap. Si la **clé** est égale au mot que l'on souhaite traduire alors on ajoute les traductions avec notre méthode ***getTranslation()*** dans l'***ArrayList<String> result*** que l'on va retourner à la fin de la méthode.

```
//Retrieve the (Array) list of translations for the same word
public ArrayList<String> getMultipleTranslations(String word){
    ArrayList<String> result = new ArrayList<String>();
    for(String i : translation.keySet()) {
        if(i == word) {
            result.addAll(this.getTranslation(word));
        }
    }
    return result;
}
```

Exercice 4:

Nous implémentons maintenant les traductions multiples bidirectionnelle. Cet ajout requiert une modification importante de notre classe Dictionary: l'ajout d'une **deuxième HashMap**: une va gérer les traductions français vers anglais et l'autre l'inverse.

```
public class Dictionary {
    private String name; //attribut pour le nom du dictionnaire
    public HashMap<ArrayList<String>, ArrayList<String>> translationFR_EN;
    public HashMap<ArrayList<String>, ArrayList<String>> translationEN_FR;
```

De multiples méthodes ont dû être modifiées pour s'adapter à ce changement:
addTranslation() met la traduction dans les deux HashMap.
getTranslation() sait si le mot passé en paramètre est francais ou anglais et renvoie la traduction associée.

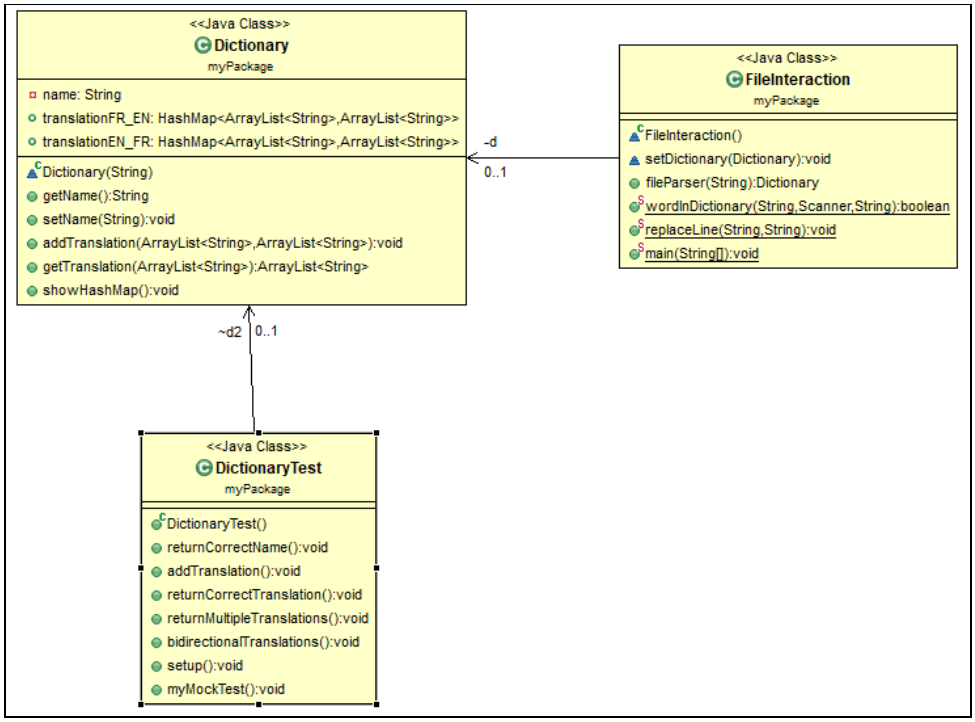
Voici le test sur les traductions bidirectionnelle:

```java
@Test
public void bidirectionalTranslations() {
    Dictionary d1 = new Dictionary("Collins");
    ArrayList<String> sensTranslations = new ArrayList<String>(2);
    ArrayList<String> sens = new ArrayList<String>(1);
    ArrayList<String> meaning = new ArrayList<String>(1);
    sensTranslations.add("direction");
    sensTranslations.add("meaning");
    sens.add("sens");
    meaning.add("meaning");
    d1.addTranslation(sens, sensTranslations);
    assertEquals(sensTranslations, d1.getTranslation(sens));
    assertEquals(sens, d1.getTranslation(sensTranslations));
}
```

Exercice 5:

C'est ici que le plus intéressant du projet est fait; nous mettons en place une base de données pour le dictionnaire sous forme de fichier texte.
Pour intégrer ce parseur dans notre architecture logicielle existante nous avons créé une nouvelle classe **FileInteraction** qui comme son nom l'indique va gérer tout ce qui concerne le fichier. Il va modifier un objet Dictionary en conséquence.

Voici le diagramme de classe de notre projet:



Nous avons décidés de ne pas utiliser de mocks pour le fichier mais plutôt pour le dictionnaire: (tous ces tests sont dans la classe **DictionaryTest**).

```
d2 = mock(Dictionary.class);
```

```
when(d2.getTranslation(chapeau)).thenReturn(chapeauTranslation);
```

Nous avons implémenté plusieurs méthodes dans la classe **FileInteraction** pour faciliter la compréhension:

- ***fileParser()*** qui retourne un dictionnaire à partir d'un fichier texte.

```
public Dictionary fileParser(String filePath) {
```

- ***wordInDictionary()*** renvoie vrai si le mot est dans le dictionnaire, faux sinon.

```
public static boolean wordInDictionary(String word, Scanner dictionary, String language) {
```

- ***replaceLine()*** remplace une partie d'un fichier texte.

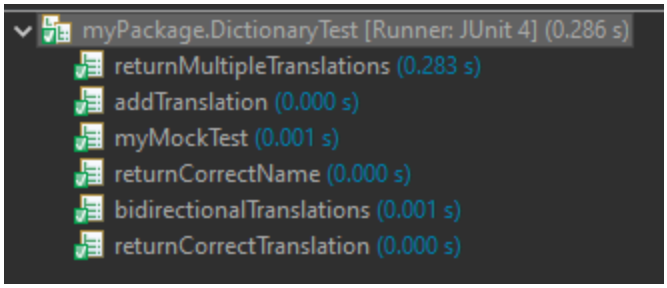
```
public static void replaceLine(String word, String filler) {
```

- ***main()*** qui va gérer toutes les méthodes au-dessus.

```
public static void main(String[] args) throws FileNotFoundException {
```

Toutes ces méthodes sont détaillées de manière excessive dans notre code si vous êtes intéressés par celles-ci.

Tous nos tests:



III. Problèmes rencontrés

Nous n'avons pas rencontré de problèmes majeurs, cependant nous avons évidemment eu quelques difficultés au cours du projet surtout pour ce qui concerne la modification du fichier texte.
Par exemple, si le mot français existe déjà et que nous voulons ajouter une deuxième traduction pour le même mot, accéder à la bonne ligne et la modifier en conséquence est un peu tordu mais nous nous en sommes sortis.

Il a aussi fallu faire pleins de tests pour vérifier tous les cas limites:
- lecture et écriture dans le fichier pour éviter toutes les exceptions possibles
- si l'utilisateur veut la traduction d'un mot qui n'est pas dans le dictionnaire, etc..

IV. Conclusion

Nous avons adorés travailler sur ce mini-projet de Test & Mocks. Il nous a permis de mettre en pratique et approfondir les différentes notions que nous avons vu dans l'unité d'enseignement Génie Logiciel.

Nous sommes maintenant davantage conscients de l'importance des tests JUnit dans le développement d'un programme pour tester tous les cas limites existants.