Projet 3: poèmes de promesses futures

- Échéance : soumettre le projet avant le 15 mars à 10h40
- Coder dans un style fonctionnel est nécessaire pour réussir
- Personne de contact : Cédric Libert

Introduction

On va encore améliorer notre générateur de poèmes pour qu'il nous permette de générer des œuvres plus sérieuses : des quatrains. Un quatrain est un poème (ou un morceau de poème) constitué de quatre vers. Pour cette exercice, on va créer uniquement des quatrain en rimes embrassées, c'est-à-dire dont les vers 1 et 4 riment et dont les vers 2 et 3 riment (structure ABBA). Par exemple :

Je lui demandai :
est-elle morte il y a cent ans ?
et je pleurai trois jours amèrement.
Personne ne le savait.

Évidemment, pour que ces quatrains soient plus créatifs, on va chercher à créer les deux vers intérieurs à partir d'un corpus particulier, et les deux vers extérieurs à partir d'un autre corpus. On utilisera aussi deux corpus de secours, au cas où les deux premiers corpus sont occupés, inaccessibles, inexistants... Ici on travaille en local, mais imaginez que ces corpus pourraient être des ressources distantes présentes sur des serveurs différents (qui pourraient être temporairement inaccessibles) et qu'il ne serait pas efficace de les concaténer directement avant le traitement. Idem pour le dictionnaire, on prévoie un dictionnaire de secours.

Cela nous permet, par ailleurs d'utiliser des Future et des Promise.

1 Objectifs d'apprentissage

Dans ce projet, vous utiliserez :

- Future
- Promise
- onSuccess
- onFailure
- recover ou recoverWith
- success (sur une Promise)
- (p :Promise).future
- Await.ready
- ...les Future dans des « for comprehension »

2 Modifications périphériques

2.1 À importer

```
1 import scala.concurrent._
```

```
2 import scala.concurrent.duration._
3 import ExecutionContext.Implicits.global
```

2.2 Modification de `DeuxVers.ecrire`

On aimerait que la méthode `ecrire` renvoie une liste de phrases (en réalité, elle en renverra deux). Ou une exception si rien n'a pu être généré. Plus de Try donc, on va gérer les erreurs ailleurs avec les Future. Notez que vous devez aussi modifier la signature de la méthode abstraite `Poeme.ecrire`.

2.3 Modification de `extraire_phrases`

Cette fonction doit être modifiée très légèrement pour renvoyer un résultat de type `Future[List[Phrase]]`.

3 Modifications de la fonction `main`

3.1 Le quatrain promis

L'idée derrière la `Promise`, c'est de déclarer une variable qui n'a pas encore de valeur, et de lui assigner une valeur à un moment donné de l'exécution. Notre `Promise` ici, ça sera notre quatrain.

Une fois le quatrain calculé (sous la forme d'une chaine de caractères), il faudra l'assigner correctement à cette `Promise`.

Attention : le programme ne devra s'arrêter que quand le `Future` associé à ce quatrain sera terminé. Si vous ne faites pas ça, il est possible que votre programme s'arrête avant que vos poèmes ne soient générés. Et vu qu'il s'agit, en réalité, de threads, il sont arrêtés si le processus dans lequel ils ont été créés est arrêté.

3.2 Extraction de phrases

Vous allez devoir extraire les phrases depuis deux corpus différents. Il est évidemment intelligent de paralléliser cette action. Heureusement, `extraire_phrase` renvoie un `Future`.

Attention ! faites en sorte que, en cas de souci dans la fonction d'extraction de phrases (le corpus n'existe pas par exemple), un corpus de secours et un dictionnaire de secours soient utilisés.

3.3 Créer deux fois deux vers

Pour chacune des listes de phrases extraites, il faut générer deux phrases qui riment. Attention, n'hésitez pas à faire ça dans un `Future`.

3.4 Si ça fonctionne

Si vous parvenez à créer deux fois deux vers, mettez-les correctement dans la `Promise`

3.5 ...sinon

Sinon mettez dans le quatrain le message d'erreur.

3.6 Finalement

Trouvez comment afficher le contenu du quatrain.

3.7 Évaluer ses pairs

Vous devrez évaluer le projet de trois autres étudiants. Cette évaluation fait partie de l'exercice et est obligatoire pour le réussir. Les critères d'évaluation sont les suivants :

- Compile
- Sans warning
- s'exécute, est correct (fait ce qui est demandé, gère les cas d'erreurs demandés), se termine
- les concepts et fonctions suivants sont utilisés : Future, Promise, onSuccess, onFailure, recover ou recoverWith, success (sur une Promise), Await.ready, Future dans un for comprehension

En plus de cette lettre vous ajouterez un **feedback**. Cette seconde partie est vraiment destinée à dire à vos condisciples comment, d'après vous, ils pourraient s'améliorer et ce que vous trouvez intéressant dans ce qu'ils ont fait. Fournissez un feedback complet et précis. C'est l'une des parties les plus importantes de l'exercice. Ce feedback aura deux partie :

- commentaires concernant tous les critères définis ci-dessus (ça ne compile pas, pourquoi ? ça ne s'exécute pas correctement, qu'est-ce que ça fait à la place de ce qui est demandé ? D'où vient le problèmes ? Quels concepts n'ont pas été implémentés ? ...)
- suggestions et commentaires : commentaires négatifs **et** positifs, questions (pourquoi tu as fait ceci ou cela ?), suggestions.

Veillez, dans les deux parties, à utiliser des **termes précis et objectifs** pour évaluer la qualité (contre exemple : bien, super, excellent ne sont pas des termes précis et objectifs) ainsi que pour décrire le programme (utilisez les termes qui conviennent en Scala : classe, case class, object, application partielle, fonction, argument,...).

3.8 Dernières consignes importantes

- envoyez vos fichiers **en UTF8 sans BOM**
- mettez des **chemins relatifs** pour les fichiers, et considérez que le dictionnaire et le corpus sont dans le même dossier que le fichier de code. Ça facilitera la correction.
- envoyez du code qui compile
- essayez de nettoyer votre code pour ne pas avoir de warning (évitez les "val" dans les for comprehension, les opératuers postfixes (a toList -> a.toList)). Par contre si vous implémentez filter et pas filterWith vous aurez un warning. Celui-là n'est pas trop grave.
- **N'indiquez pas de package**. Ça facilitera la correction.
- Pas besoin de faire de l'interaction avec l'utilisateur.