Travail pratique #3

Ce travail doit être fait individuellement.

Aucun retard permis

Notions mises en pratique : respect d'un ensemble de spécifications, expressions lambda, et streams.

1. Description générale

Votre travail consiste à implémenter un petit module de recherche et d'analyse de corpus dans un dictionnaire donné, en utilisant les streams et les expressions lambda.

Fichiers fournis :

1)	Classe `EntreeDictionnaire`	Classe qui modélise une entrée de dictionnaire qui contient le mot défini, la catégorie grammaticale de ce mot, et la définition de ce mot. À ne pas modifier.
3)	Fichier texte **echantillon.txt**	Exemple de fichier texte contenant des entrées de dictionnaire, pouvant être passé en paramètre du constructeur de la classe `Dictionnaire`.

Note : La classe `EntreeDictionnaire` fournie sera utilisée telle quelle dans les tests de votre programme. Il est donc important de ne pas la modifier.

1.1 DÉTAILS ET CONTRAINTES D'IMPLÉMENTATION

Vous devez implémenter une (seule) classe nommée `Dictionnaire`. Cette classe contient un dictionnaire c.-à-d. une liste d'entrées de dictionnaire (où chaque entrée de dictionnaire contient le mot défini, sa catégorie grammaticale, et sa définition), et fournit des méthodes permettant de faire des recherches spécifiques dans ce dictionnaire, et d'en analyser le corpus. Cette classe doit contenir les membres décrits dans les sous-sections ci-dessous.

IMPORTANT :

- **Le nom et le type de l'attribut, les noms de méthodes, les types retournés ainsi que l'ordre et le type des paramètres mentionnés dans les sections suivantes doivent être respectés à la lettre sinon vous risquez de perdre plusieurs points dans les tests.**
- **Les recherches et tris effectué(e)s sur des chaines de caractères ne doivent jamais tenir compte de la casse.**
- **L'ordonnancement des chaines de caractères se fait toujours selon l'ordre naturel des chaines de caractères, mais sans tenir compte de la casse.**

1.1.1 Constantes de classe

Vous pouvez déclarer autant de constantes de classe que vous le jugez pertinent.

1.1.2 Attribut d'instance

Nom de l'attribut	type	Description
`dictionnaire`	`List<EntreeDictionnaire>`	La liste des entrées de ce dictionnaire.

- Vous DEVEZ respecter le nom et le type donnés de l'attribut d'instance.
- Vous ne DEVEZ PAS ajouter d'autres attributs d'instance ou de classe.

1.1.3 Constructeur

Paramètres	type	Description
`cheminFic`	`String`	Le chemin du fichier qui contient la liste des entrées de dictionnaire que contiendra ce dictionnaire. On suppose que ce fichier, s'il existe, est valide (qu'il est bien formé selon le format expliqué ci-dessous). Le fichier peut aussi exister, mais ne contenir aucune entrée de dictionnaire.

Ce constructeur lit chacune des entrées de dictionnaire contenues dans le fichier donné en paramètre, et construit la liste des entrées de dictionnaires (attribut `dictionnaire`). Le fichier est formaté de la façon suivante (supposez qu'il est valide, c.-à-d. formaté correctement, et qu'il n'y manque aucune donnée) : chaque ligne du fichier (sauf les lignes blanches ou les lignes de commentaires) contient les informations sur une entrée de dictionnaire : (1) le **mot** défini, (2) la **catégorie grammaticale** de ce mot, et (3) la **définition** de ce mot. Chaque information est séparée par le caractère `'\t'` (TAB). Par exemple une ligne d'un tel fichier pourrait ressembler à la chaine suivante :

```
"JUPITÉRIEN\tadj.\tQui appartient ou se rattache à la planète Jupiter."
```

Précisions sur le format d'un fichier valide :

- La casse des informations dans le fichier peut varier.
- Le fichier peut contenir **des lignes blanches ou des lignes de commentaires** qui doivent être ignorées lors de la lecture des entrées du dictionnaire. Plus précisément, si l'on appelle la méthode `trim()` sur une ligne blanche, cela retourne une chaine vide, et si l'on appelle la méthode `trim()` sur une ligne de commentaires, cela retourne une chaine dont le premier caractère est le caractère `'#'`.
- Pour les lignes qui ne sont pas blanches ou de commentaires, il n'y a aucune espace superflue en début ou en fin de ligne.
- Il est possible que le fichier contienne des **doublons** (lignes identiques, peu importe la casse).
- Les lignes ne sont pas nécessairement triées en ordre lexicographique (contrairement à un vrai dictionnaire).
- Si un mot possède `n` définitions, il y aura donc `n` entrées de dictionnaire pour ce mot.
- Notez aussi qu'un mot peut avoir différentes catégories grammaticales. Par exemple, le mot « uniforme » est un nom, et aussi un adjectif (correspond à différentes entrées de dictionnaire).
- Les catégories grammaticales possibles sont : `n.` (nom), `v.` (verbe), `adj.` (adjectif), et `adv.` (adverbe).

Le fichier **echantillon.txt**, fourni avec l'énoncé de ce TP, est un exemple de fichier valide. Il contient quelques entrées de dictionnaires, quelques lignes blanches, et de commentaires. Vous pourrez l'utiliser pour tester vos méthodes. Évidemment, vous pouvez ajouter certaines entrées bidon dans ce fichier, pour tester des cas particuliers. Notez que ce ne sera pas nécessairement ce fichier qui sera utilisé pour les tests de votre TP.

Donc, ce constructeur construit une `EntreeDictionnaire` pour chacune des lignes du fichier `cheminFic` (sauf les lignes blanches ou de commentaires), et les ajoute à la liste `dictionnaire`.

DE PLUS :

1. L'ordre des entrées de dictionnaire, dans la liste `dictionnaire` créée, doit respecter l'ordre des entrées de dictionnaire dans le fichier donné en paramètre.
2. Le constructeur doit éliminer les doublons des entrées de dictionnaire, s'il y en a. Il est important que les doublons retirés soient bien ceux qui viennent après dans la liste des entrées de dictionnaire. Par exemple, soit la liste qui contient les entrées de dictionnaires suivantes : `[ e1, e2, e3, e1, e1, e3, e4]`. Il est important que ce soit les doublons bleus qui soient retirés de la liste, et non les doublons orange.
3. Si le fichier reçu en paramètre ne peut pas être lu (est `null`, est inexistant sur le disque, etc.) ou s'il existe, mais qu'il est vide (ou ne contient que des lignes blanches ou de commentaires), alors la liste créée demeure vide (la liste `dictionnaire` n'est pas `null`, et a une longueur égale à 0).

CONTRAINTES D'IMPLÉMENTATION

- **VOUS DEVEZ** utiliser un stream contenant les lignes du fichier donné en paramètre, et des expressions lambda comme paramètre(s) des méthodes appelées sur ce stream.
- **VOUS DEVEZ** utiliser la méthode `map` de `Stream` pour transformer les `String` en `EntreeDictionnaire` dans le stream.

1.1.4 Méthodes d'instance publiques

Nom méthode : `nombreEntreesDictionnaire`
Type retour : `long`
Paramètre : Aucun

Cette méthode retourne le nombre total d'entrées dans ce dictionnaire.

CONTRAINTE D'IMPLÉMENTATION

VOUS DEVEZ utiliser un stream et des expressions lambda comme paramètre(s) des méthodes appelées sur ce stream (s'il y a lieu).

Nom méthode : `nombreDeMots`
Type retour : `long`

Paramètre	type	Description
`categorieGramm`	`String`	Catégorie grammaticale des mots recherchés.

Cette méthode retourne le nombre de mots **distincts** (sans tenir compte de la casse) de la catégorie grammaticale `categorieGramm` donnée (sans tenir compte de la casse), parmi toutes les entrées de ce dictionnaire. Attention, un mot de la catégorie grammaticale recherchée peut avoir plusieurs définitions, qui correspondent à plusieurs entrées de dictionnaire. Il ne faut cependant compter ce mot qu'une seule fois. Si aucun mot de la catégorie grammaticale donnée n'est trouvé, cette méthode retourne 0.

CONTRAINTE D'IMPLÉMENTATION

VOUS DEVEZ utiliser un stream et des expressions lambda comme paramètre(s) des méthodes appelées sur ce stream.

Nom méthode : `obtenirEntreesDictionnaire`
Type retour : `List<EntreeDictionnaire>`

Paramètre	type	Description
`mot`	`String`	Les entrées de dictionnaire recherchées doivent définir ce `mot` (en plus d'avoir une catégorie grammaticale, qui se trouve dans `categoriesGramm`).
`categoriesGramm`	`String[]`	Liste des catégories grammaticales des entrées de dictionnaires recherchées.

Cette méthode retourne une liste contenant toutes les entrées de dictionnaire définissant le `mot` donné, et dont la catégorie grammaticale se trouve dans le tableau `categoriesGramm` donné. Si aucune entrée de dictionnaire n'est trouvée, cette méthode retourne une liste vide.

Si le paramètre `mot` est `null`, cette méthode lève une `NullPointerException`. Si le paramètre `categoriesGramm` est `null`, la méthode ne tient pas compte de la catégorie grammaticale du mot recherché. Autrement dit, on recherche alors toutes les entrées de dictionnaire dont le mot défini est `mot`, peu importe sa catégorie grammaticale. Notez que si le tableau `categoriesGramm` est de longueur 0, cette méthode retourne automatiquement une liste vide.

De plus, **la liste des entrées de dictionnaire retournée doit être triée** sur la définition des entrées, et si plusieurs entrées de dictionnaire ont la même définition (sans tenir compte de la casse), celles-ci doivent être triées entre elles sur leur catégorie grammaticale.

⇨ RAPPEL : les recherches et tris sur des chaines de caractères ne doivent jamais tenir compte de la casse.

CONTRAINTES D'IMPLÉMENTATION

- **VOUS DEVEZ** utiliser un stream et des expressions lambda comme paramètre(s) des méthodes appelées sur ce stream.
- Pour obtenir une liste (`List<T>`) à partir d'un `stream<T>`, **UTILISEZ** la méthode terminale `collect`, sur ce stream, de cette manière : `stream.collect(Collectors.toList())`. Il vous faudra importer la classe `java.util.stream.Collectors;`

Nom méthode : `obtenirEntreesDictionnaire`
Type retour : `List<EntreeDictionnaire>`

Paramètre	type	Description
`expression`	`String`	Les entrées de dictionnaire recherchées doivent contenir cette `expression` dans leur définition.

Cette méthode retourne une liste contenant toutes les entrées de dictionnaire dont la définition contient l'expression donnée (sans tenir compte de la casse). Si aucune entrée de dictionnaire n'est trouvée, cette méthode retourne une liste vide. Par exemple, les chaines suivantes contiennent l'expression `"AbS"` : "**abs**olution", "**ABS**TRACTION", "toto**aBs**toto", "toto**AbS**".

Notez qu'une chaine vide est contenue dans toute chaine non `null`.

Si le paramètre `expression` est `null`, cette méthode lève une `NullPointerException`.

De plus, **la liste des entrées de dictionnaire retournée doit être triée** sur le mot des entrées. Si plusieurs entrées de dictionnaire ont le même mot (sans tenir compte de la casse), celles-ci doivent alors être triées entre

elles sur leur définition (sans tenir compte de la casse), et si plusieurs entrées de dictionnaire ont la même définition (sans tenir compte de la casse), celles-ci doivent être triées entre elles selon leur catégorie grammaticale (sans tenir compte de la casse).

⇨ RAPPEL : les recherches et tris sur des chaines de caractères ne doivent jamais tenir compte de la casse.

CONTRAINTES D'IMPLÉMENTATION

- **VOUS DEVEZ** utiliser un stream et des expressions lambda comme paramètre(s) des méthodes appelées sur ce stream.
- Pour obtenir une liste (`List<T>`) à partir d'un `stream<T>`, **UTILISEZ** la méthode terminale `collect`, sur ce stream (comme indiqué plus haut).

Nom méthode : `trouverMotsCorrespondants`
Type retour : `String[]`

Paramètre	type	Description
`patron`	`String`	Le patron auquel doivent correspondre les mots retournés.

Cette méthode retourne un tableau de longueur minimale `n` contenant les `n` **mots distincts** (sans doublons) qui correspondent (sans tenir compte de la casse) au `patron` donné en paramètre, parmi les mots de toutes les entrées de ce dictionnaire. Un mot correspond à un patron s'il contient le même nombre de caractères que le patron, et si le caractère à la position `i` dans le mot est égal (sans tenir compte de la casse) au caractère à la position `i` dans le patron ou sinon, s'il correspond à un point `'.'` dans le patron. Par exemple, soit le patron `"al.o...e."`. Les deux mots suivants sont des mots correspondants à ce patron : `"ALGONKIEN"`, et `"ALÉOUTIEN"`. On voit que les caractères bleus correspondent à des caractères identiques (en ignorant la casse) dans le patron, aux mêmes positions. Les caractères noirs correspondent à des points dans le patron. Et les mots correspondants ont la même longueur que le patron. Au patron `"......."` correspondent tous les mots de 7 lettres. Au patron `"A...."` correspondent tous les mots de 5 lettres commençant par la lettre `'a'` (sans tenir compte de la casse), au patron `"b..o."` correspondent des mots comme `"BACON"`, `"BISON"`, `"BUTOR"`, etc.

De plus :
- Le tableau retourné doit être trié selon l'ordre naturel des chaines de caractères (de manière lexicographique), mais sans tenir compte de la casse.
- Tous les mots dans le tableau retourné doivent être en minuscules.

Note : n'oubliez pas que la méthode `distinct` sur un stream de `String` utilise la méthode `equals` de la classe `String`, et non `equalsIgnoreCase`.

CONTRAINTES D'IMPLÉMENTATION

- **VOUS DEVEZ** utiliser un stream et des expressions lambda comme paramètre(s) des méthodes appelées sur ce stream.
- **VOUS DEVEZ** utiliser la méthode `toArray` de `Stream` pour obtenir le tableau à retourner.

Nom méthode : `trouverMotsParLongueur`
Type retour : `String[]`

Paramètre	type	Description
`nbrMots`	`int`	Le nombre maximum de mots retournés.

Cette méthode retourne un tableau de longueur minimale contenant les `nbrMots` mots **distincts** (sans tenir compte de la casse) les plus longs ou les plus courts (selon leur nombre de caractères), parmi les mots de toutes les entrées de ce dictionnaire. Plus précisément :

- Si `nbrMots` est négatif, on veut obtenir les `|nbrMots|` les plus courts.
- Si `nbrMots` est positif, on veut obtenir les `nbrMots` les plus longs.
- Si `nbrMots` est égal à 0, évidemment, le tableau retourné sera de longueur 0.
- Si `|nbrMots|` est plus grand que le nombre de mots distincts `n` dans ce dictionnaire, le tableau retourné contient **tous** les mots distincts de ce dictionnaire (et le tableau retourné est de longueur `n`).
- Tous les mots dans le tableau retourné doivent être en minuscules.

Algorithme **à respecter** pour trouver les mots les plus courts :

1) Ordonner d'abord les mots **distincts** en ordre croissant de leur longueur. Puis, pour les mots de même longueur, les ordonner entre eux en ordre lexicographique croissant.
2) Retourner un tableau contenant les `|nbrMots|` premiers mots (ou tous les mots si `|nbrMots|` est plus grand que le nombre de mots distincts dans ce dictionnaire).

Algorithme **à respecter** pour trouver les mots les plus longs :

1) Ordonner d'abord les mots **distincts** en ordre décroissant de leur longueur. Puis, pour les mots de même longueur, les ordonner entre eux en ordre lexicographique croissant.
2) Retourner un tableau contenant les `nbrMots` premier mots (ou tous les mots si `nbrMots` est plus grand que le nombre de mots distincts dans ce dictionnaire).

Par exemple :

```
Dictionnaire d = new Dictionnaire("echantillon.txt"); //fichier fourni

System.out.println(Arrays.asList(d.trouverMotsParLongueur(6)));
```

Affiche (les mots de longueurs différentes sont de différentes couleurs) **:**

```
[insatisfaction, algorithmique, décomposition, effectivement, intronisation, visualisation]
```

```
System.out.println(Arrays.asList(d.trouverMotsParLongueur(-6)));
```

Affiche (les mots de longueurs différentes sont de différentes couleurs) **:**

```
[mot, yoyo, brume, céder, effet, gorge]
```

CONTRAINTES D'IMPLÉMENTATION

- **VOUS DEVEZ** utiliser un stream et des expressions lambda comme paramètre(s) des méthodes appelées sur ce stream.
- **VOUS DEVEZ** utiliser la méthode `toArray` de `Stream` pour obtenir le tableau à retourner.

NOTE : La méthode `limit` de l'interface `Stream` pourrait être fort utile ici…

Nom méthode : `moyenneLongueurMots`
Type retour : `double`
Paramètre : Aucun

Cette méthode retourne la moyenne de la longueur des mots **distincts** (sans tenir compte de la casse) de toutes les entrées de ce dictionnaire.

Par exemple, soit un mini dictionnaire ne contenant que les 3 entrées suivantes :

```
MAISON      n.    Bâtiment destiné à servir d'habitation à l'homme.
ÉMINENT     adj.  Élevé par rapport au niveau environnant.
Maison      n.    Habitation pour l'humain.
```

Les deux mots **distincts** sont "MAISON" de longueur 6, et "ÉMINENT", de longueur 7. La moyenne des longueurs est donc `(6 + 7) / 2 = 6.5`

Note : Si ce dictionnaire est vide, la moyenne retournée est 0.

CONTRAINTES D'IMPLÉMENTATION

- **VOUS DEVEZ** utiliser un stream, et des expressions lambda comme paramètre(s) des méthodes appelées sur ce stream.
- **VOUS DEVEZ** utiliser la méthode `average` de `IntStream` pour obtenir le résultat retourné.

Nom méthode : `frequence`
Type retour : `double`

Paramètre	type	Description
`c`	`char`	Le caractère dont on veut la fréquence sur le corpus formé par les définitions de toutes les entrées de ce dictionnaire.

Cette méthode retourne la fréquence (en pourcentage) du caractère donné en paramètre (sans tenir compte de la casse) dans le corpus formé par les définitions de toutes les entrées de ce dictionnaire. Cette méthode n'élimine pas les doublons de définition, s'il y en a. La fréquence du caractère `c` se calcule comme suit :

```
Fréquence(c) = nombre total de caractères c / nombre total de caractères * 100
```

Par exemple, soit un mini dictionnaire ne contenant que les 2 entrées suivantes :

```
MAISON      n.    Bâtiment destiné à servir d'habitation à l'homme.
ÉMINENT     adj.  Élevé par rapport au niveau environnant.
```

Trouvons la fréquence du caractère `'e'` : la première définition contient 49 caractères dont 4 `'e'`, et la seconde définition contient 40 caractères dont 3 `'e'`.

```
Fréquence('e') = (4 + 3) / (49 + 40) * 100 = 7 / 89 * 100 = 7.8651685 %
```

Note : Si l'on ne trouve aucun caractère recherché, la fréquence retournée est 0.

CONTRAINTE D'IMPLÉMENTATION

- **VOUS DEVEZ** utiliser au moins un stream, et des expressions lambda comme paramètre(s) des méthodes appelées sur ce stream.

2. Précisions sur les spécifications

- À moins d'indication contraire, lorsqu'on demande de trier, c'est toujours en ordre croissant.
- Deux objets de type `EntreeDictionnaire e1` et `e2` sont considérées égales si `e1.equals(e2)` retourne `true`.
- **À moins d'indication contraire, les recherches, comparaisons, et tris effectué(e)s sur les chaines de caractères ne doivent jamais tenir compte de la casse.**
- **Vous devez utiliser des expressions lambda pour tous les paramètres d'un type interface fonctionnelle.**
- Réutilisez votre code autant que possible. Vous pouvez (et devriez) faire des **méthodes privées** pour bien **structurer/modulariser votre code** (séparation fonctionnelle).
- Toute méthode qui n'est pas décrite dans cet énoncé DOIT être `private`.
- **Le non-respect des contraintes d'implémentation peut engendrer des pénalités sévères, car ce sont ces contraintes qui assurent que vous utilisiez la matière qu'on veut noter.**
- N'oubliez pas d'écrire les commentaires de documentation (**Javadoc**) pour documenter toutes vos méthodes et votre classe.
- Utilisez des constantes autant que possible. Celles-ci doivent être déclarées et initialisées au niveau de la classe (constantes de classe) : `public (ou private) static final`...
- Toute **variable globale est INTERDITE** (sauf l'attribut d'instance, et les constantes de classe).
- Votre classe doit se trouver dans le **paquetage par défaut**.
- Il ne doit y avoir aucun affichage dans vos méthodes (pas de `System.out`)
- Vous DEVEZ respecter **le style Java**.
- Votre code doit compiler et s'exécuter avec le **JDK 8.**
- Vous devez respecter le **principe d'encapsulation** des données.
- Votre fichier doit être encondé en **UTF-8**.

Le non-respect de toute spécification ou consigne se trouvant dans l'énoncé de ce TP est susceptible d'engendrer une perte de points.

Si quelque chose est ambigu, obscure, s'il manque de l'information, si vous ne comprenez pas les spécifications, si vous avez des doutes… vous avez la responsabilité de vous informer auprès de votre enseignante.

3. Détails sur la correction

3.1 LA QUALITÉ DU CODE (40 POINTS)

Concernant les critères de correction du code, lisez attentivement le document "**Critères généraux de correction du code Java"**. De plus, votre code sera noté sur le respect des conventions de style Java vues en classe dont un résumé se trouve dans le document **"Conventions de style Java"**. Ces deux documents peuvent être téléchargés dans la section TRAVAUX PRATIQUES (ET BOITES DE REMISE) de la page Moodle du cours.

Votre code sera corrigé sur la totalité ou une partie des critères de correction indiqués ci-dessus, ainsi que sur le respect des spécifications/consignes mentionnées dans ce document.

3.2 L'EXÉCUTION (60 POINTS)

Un travail qui ne compile pas se verra attribuer la note 0 pour l'exécution.

Votre code sera testé en tout ou en partie. Les points seront calculés sur les tests effectués. Ceci signifie que si votre code fonctionne dans un cas x et que ce cas x n'est pas testé, vous n'obtiendrez pas de points pour ce cas. Il est donc dans votre intérêt de vous assurer du bon fonctionnement de votre programme dans tous les cas possible..

4. Date et modalités de remise

4.1 REMISE

Date de remise : AU PLUS TARD le **27 juillet 2022** à 23h59. **AUCUN RETARD ACCEPTÉ**

Le fichier à remettre : `Dictionnaire.java`
(**NE PAS remettre votre fichier dans une archive zip, rar…**).

Remise via Moodle uniquement.

Vous devez remettre (téléverser) votre fichier sur le site du cours (Moodle). Vous trouverez la boîte de remise dans la section **TRAVAUX PRATIQUES (ET BOITES DE REMISE)**.

Assurez-vous de remettre votre travail avant la date limite, sur Moodle, car après, vous n'aurez plus accès à la boite de remise, et vous ne pourrez plus remettre votre travail.

4.2 POLITIQUE CONCERNANT LES RETARDS

AUCUN RETARD PERMIS

4.3 REMARQUES GÉNÉRALES

- Aucun travail reçu par couriel ne sera accepté. PLUS PRÉCISÉMENT, **un travail remis par courriel sera considéré comme étant non remis**.
- **Vous avez la responsabilité de conserver des copies de sauvegarde** de votre travail (sur disque externe, Dropbox, Google Drive, etc.). La perte d'un travail due à un vol, un accident, un bris… n'est pas une raison valable pour obtenir une extension pour la remise de votre travail.
- **N'oubliez pas d'écrire votre nom complet et votre code permanent (entre autres) dans l'entête de votre classe à remettre).**

N'attendez pas à la dernière minute pour commencer le travail, vous allez fort probablement rencontrer des problèmes inattendus !

Le règlement sur le plagiat sera appliqué sans exception. Vous devez ainsi vous assurer de ne pas échanger du code avec des collègues ni de laisser sans surveillance votre travail au laboratoire. Vous devez également récupérer rapidement toutes vos impressions de programme au laboratoire.

BON TRAVAIL