Mini Projet – Outil de Recherche, Déduplication et Comparaison de Noms

Nom : Mohamed Ramzi Haddad
Classes : 1AINFO
École : Ecole Nationale d'Ingénieurs de Tunis
Module : Mini Projet Programmation Orientée Objets
Version : 1
Date : 9/4/2025

🎯 Objectif

Développer une application en ligne de commande (CLI) permettant d'effectuer trois types de traitements sur des fichiers CSV contenant des **noms complets** (chaînes de caractères sur une seule colonne) :

1. **Recherche** d'un nom dans une liste.
2. **Comparaison** de deux listes pour détecter les correspondances.
3. **Déduplication** d'une liste.

Chaque traitement repose sur une **configuration modulaire** combinant un prétraitement, une structure d'indexation, une mesure de comparaison et un seuil ou un nombre de résultats.

⚙️ Fonctionnement Général

- L'utilisateur **choisit d'abord la fonctionnalité** à exécuter (recherche, comparaison, déduplication).
- Ensuite, il **fournit les chemins des fichiers** nécessaires au traitement.
- L'outil applique la **configuration active**, qui peut être personnalisée ou laissée avec ses **valeurs par défaut**.

🔧 Configuration

La configuration inclut les éléments suivants :

1. **Prétraitement** : ex. normalisation (minuscule, accents...), encodage phonétique (Soundex, Metaphone...).
2. **Indexation** : ex. dictionnaire, trie, arbre.
3. **Mesure de comparaison** : ex. distance de Levenshtein, Jaro-Winkler, égalité exacte.
4. **Paramètre de résultat** :
    - Soit un **seuil** (valeur minimale/maximale selon la mesure),
    - Soit un **nombre maximal de résultats** à afficher.

⚠️ Acceptation d'un résultat :

- Pour les **mesures de similarité**, une valeur est acceptée si **elle est supérieure ou égale au seuil**.
- Pour les **distances**, une valeur est acceptée si **elle est inférieure ou égale au seuil**.

🧩 Contraintes sur la configuration :

- Chaque point de variabilité doit proposer **au moins 2 à 3 variantes**, **implémentées via polymorphisme**.
- Une **valeur par défaut** doit être définie pour chaque paramètre.

📋 Fonctionnalités Principales

1. Recherche d'un nom

- L'utilisateur saisit le **nom complet** à rechercher.
- Il indique le **chemin du fichier CSV** contenant la liste.
- Le système affiche les correspondances selon la configuration active.

2. Comparaison de deux listes

- L'utilisateur fournit les **chemins des deux fichiers CSV**.
- Le système affiche les couples de noms jugés similaires.

3. Déduplication d'une liste

- L'utilisateur fournit le **chemin du fichier à traiter**.
- Le système détecte les doublons internes.

4. Configuration

- 4.1 Choisir un **prétraitement**
- 4.2 Choisir une **structure d'index**
- 4.3 Choisir une **mesure de comparaison**
- 4.4 Définir un **seuil** ou un **nombre de résultats**

🧭 Exemple de Menu

```
1. Effectuer une recherche
   Saisir le nom à rechercher
   Fournir le fichier CSV
2. Comparer deux listes
   Fournir le premier fichier
   Fournir le second fichier
3. Dédupliquer une liste
   Fournir le fichier à traiter
4. Configurer les paramètres
   1 Choisir les prétraitements
   2 Choisir une structure d'index
   3 Choisir une mesure de comparaison
   4 Définir le seuil ou nombre de résultats
5. Quitter
```

🧪 Exigences Techniques

- Interface en ligne de commande simple avec **menu numéroté clair**.
- Architecture orientée objet avec **utilisation du polymorphisme** pour chaque stratégie.
- Code **modulaire et extensible**, facilement adaptable.
- Bonne **performance** sur des fichiers de taille moyenne à grande.
- Gestion **automatique des valeurs par défaut** pour chaque traitement.
- Il faut utiliser Github tout au long du projet et utiliser l'email officiel de l'enit.

🏅 Critères d'Évaluation

1. **Modélisation** :
    - La structuration des classes et des relations entre elles.
    - Utilisation appropriée de la POO (encapsulation, héritage, polymorphisme).
2. **Généricité et Extensibilité** :
    - Le code doit être conçu pour permettre facilement l'ajout de nouvelles fonctionnalités ou de nouveaux algorithmes.
    - Gestion flexible des encodages/structures des indexes/mesures.
3. **Polymorphisme** : Le polymorphisme doit être utilisé dans
    - les structures de données des indexes
    - Les encodeurs (bonus)
    - Les mesures de similarité et de distance
4. **Bonnes pratiques de POO** :
    - Utilisation d'interface ou de classes abstraites pour définir des comportements génériques.
    - Code propre, lisible et bien commenté.
5. **Analyse des résultats** :
    - Temps de recherche, d'indexation, de déduplication en fonction des structures de données et des algorithmes.
    - Qualité de recherche en fonction des mesures et des encodages.
    - Complexité des algorithmes.
6. **Gestion des opérations/processus de développement** :
    - Utilisation de Github et traçabilité des interventions.
    - Tests unitaires pour valider les fonctionnalités (bonus).