Feuille de TP n° 2 : Java, Mots et Tableaux de Mots

Même si cela n'est pas mentionné explicitement, vous devez tester vos fonctions à l'aide d'une méthode `main` adaptée.

Exercice 1 : La classe `Chaine`

Récupérez le code de la classe `Chaine`. Il s'agit de la classe que vous avez implémentée lors du TP1. Au cas où vous n'auriez pas sauvegardé votre code, une classe `Chaine` avec quelques fonctionnalités est disponible à l'adresse :

`http ://www.pps.jussieu.fr/ lengrand/Work/Teaching/Chaine.java`.

Au cours du TP, vous avez le droit d'enrichir cette classe en y ajoutant de nouvelles méthodes.

Exercice 2 : La classe `Mot`, retour sur la notion d'héritage

On veut spécialiser la classe `Chaine` en une classe plus spécifique représentant des mots.

1. Écrire une classe `Mot` héritant de la classe `Chaine`. Définir dans cette classe les constructeurs `Mot()`, `Mot(int taille, char c)` et `Mot(String s)`.
2. Vérifier que les méthodes définies dans la classe `Chaine` peuvent être utilisées sans être redéfinies dans la classe `Mot`.
3. Écrire deux méthodes `char tete_mot()` et `Mot queue_mot()`
4. Écrire une méthode `int nbOcc(char c)` qui rend le nombre d'occurrences du caractère `c` dans le mot courant.
5. Écrire une méthode récursive `boolean estFacteur(Mot m)` qui rend `true` si `m` est un facteur du mot courant.
6. Écrire une méthode `boolean estSousMot(Mot m)` qui rend `true` si `m` est un sous-mot du mot courant.
7. Écrire une méthode `Mot miroir()` qui rend le mot miroir du mot courant (c'est à dire le même mot à l'envers).
8. Écrire une méthode `boolean palindrome()` qui rend `true` si le mot courant est un palindrome (c'est à dire s'il peut être lu indifféremment dans les deux sens).

Par la suite, on va vouloir considérer des opérations plus complexes sur les mots comme par exemple `prefixes` permettant d'obtenir la liste de tous les préfixes du mot courant. Pour cela, on a besoin de manipuler des listes de mots.

Exercice 3 : La classe `TabMots`

1. Écrire une classe `TabMots` permettant de stocker et de manipuler un tableau de `Mots`. L'unique champs de cette classe sera `private`. Qu'est-ce que cela signifie ? Faire ce qu'il faut pour permettre la lecture et la mise à jour de ce champ.
2. Écrire un constructeur `TabMots(int n)` permettant de construire un tableau de taille `n`. Chaque case du tableau contiendra le mot vide.
3. Écrire une méthode `int length()` qui rend la longueur du tableau.
4. Écrire une méthode `String toString()` qui crée un objet de type `String` à partir du tableau stocké.

Exercice 4 : Quelques opérations plus complexes

1. Écrire les méthodes `TabMots prefixes()`, `TabMots suffixes()` et `TabMots facteurs()`, qui construisent la liste de tous les préfixes, suffixes et facteurs du mot courant. Dans quelle classe allez vous écrire ces méthodes ?
2. Écrire une méthode `TabMots sousMots()` qui construit la liste de tous les sous-mots du mot courant.

3. On appelle *mélange* (ou *shuffle*) de deux mots le fait de construire un mot nouveau en mélangeant les lettres de deux mots existants, tout en conservant l'ordre des lettres. Par exemple, en mélangeant les mots *abc* et *dd*, on peut obtenir les mots *abcdd*, *adbdc*, *abddc*, etc. Écrire une méthode `TabMots shuffle(Mot m)` qui calcule la liste de tous les mots que l'on peut obtenir par mélange du mot courant et du mot `m`.
4. Écrire les méthodes `int nbOccFacteur(Mot m)` et `int nbOccSousMot(Mot m)` qui calculent le nombre d'occurrences du mot `m` respectivement comme facteur et comme sous-mot du mot courant.

Exercice 5 : Réflexion

Le but des prochaines séances de TP sera d'écrire une classe Java `AutomateDet` pour représenter et manipuler les automates déterministes. Les automates ayant une structure relativement complexe, cela nécessitera très certainement de définir plusieurs classes contenant chacune un ou plusieurs champs. Proposez une architecture adaptée pour manipuler les automates déterministes.