Feuille de TD n° 3 : Rappels de Java – mots et listes chaînées

Toutes les méthodes mentionnées dans le sujet sont des méthodes dynamiques.

Exercice 1 : Listes chaînées (rappel d'IF122). Les mots finis sur un alphabet sont des séquences de caractères. Pour ce TD, nous souhaitons représenter de telles séquences à l'aide de listes chaînées.

1. Écrire une classe `Liste` permettant de représenter des listes simplement chaînées contenant des données de type `Object`.
2. Écrire les constructeurs `Liste()` et `Liste(Object o, Liste l)` permettant de construire une liste vide, et une liste de premier élément `o` et d'éléments suivants `l`
3. Écrire la méthode `boolean estVide()` qui teste si une liste ne contient aucun élément, et les méthodes `Object tete()` et `Liste queue()` qui renvoient respectivement le premier élément de la liste, et la liste privée de son premier élément.
4. Écrire les méthodes suivantes : `int longueur()` qui renvoie la longueur de la liste, `Object element(int i)` qui renvoie le i-ème élément de la liste, et `Liste sousListe(int i, int j)` qui renvoie la portion de la liste située entre les indices `i` et `j` de la liste courante.
5. Écrire les méthodes `int indiceDe(Object o)` qui renvoie l'indice de première occurrence de l'objet `o` s'il apparait dans la liste, et -1 sinon, et `boolean equals(Liste l)` qui teste si la liste courante et la liste `l` sont identiques.
6. Écrire la méthode `String toString()` qui convertit la liste en chaîne de caractères prête à être affichée.

Exercice 2 : Manipulations simples sur les mots. On veut spécialiser la classe `Liste` en une classe plus spécifique représentant des mots.

1. Écrire une classe `Mot` héritant de la classe `Liste`, dont les données sont toutes du type `Character`. Définir dans cette classe les constructeurs `Mot()` et `Mot(char c, Mot l)`, et définir une constante `EPSILON` représentant le mot vide.
2. Écrire une méthode `int nbOcc(char c)` qui rend le nombre d'occurrences du caractère `c` dans le mot courant.
3. Écrire les méthodes `boolean estFacteur(Mot m)` et `boolean estSousMot(Mot m)` qui rendent `true` si `m` est un facteur ou un sous-mot du mot courant.
4. Écrire une méthode `Mot miroir()` qui rend le mot miroir du mot courant (c'est à dire le même mot à l'envers).
5. 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).

Exercice 3 : Manipulations (un peu moins simples) sur les mots.

1. Écrire les méthodes `Liste prefixes()`, `Liste suffixes()` et `Liste facteurs()`, qui construisent la liste de tous les préfixes, suffixes et facteurs du mot courant.
2. Écrire une méthode `Liste 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 `Liste shuffle(Mot m)` qui calcule l'ensemble 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.
5. Écrire les méthodes `boolean commuteAvec(Mot m)` et `boolean estConjugue(Mot m)` qui rendent `true` si le mot courant commute avec `m` ou est conjugué à `m` respectivement.