Université de Nice–Sophia Antipolis — Mercredi 14 avril 2004
E.S.I.N.S.A. 2ème Année

Contrôle de TP Informatique

Documents non autorisés — Durée : 2h

Attention : la qualité des commentaires (avec notamment, la présence des antécédents et conséquents avant chaque méthode), les noms donnés aux variables, l'utilisation à bon escient des majuscules et la bonne indentation rentreront pour une part **importante** dans l'appréciation du travail.
Les sections 1 et 2 sont indépendantes ; *les exercices qui les composent ne le sont pas tous.* Cependant, si vous n'avez pas réussi à écrire la méthode demandée à l'un des exercices, vous pouvez faire les autres exercices en considérant que vous disposez quand même de ladite méthode. En plus des méthodes explicitement demandées, vous pouvez définir toutes les méthodes que vous estimez nécessaires pour répondre à une question.
Le sujet comporte un verso.

Avant de commencer

Vous mettrez votre solution dans votre compte sur `djinn`.

Plus grand préfixe commun

Un *préfixe* d'une chaîne de caractères `s` est toute sous-chaîne de `s` qui débute au premier caractère de `s`, y compris la chaîne `s` elle-même.

Le *plus grand préfixe commun* (pgpc) à un ensemble de mots est défini comme la chaîne de caractères la plus longue possible telle qu'elle soit un préfixe de tous les mots de l'ensemble.

> **Exemple 1 :** Soit l'ensemble de mots `retarder, retard, renard`, le *pgpc* de cet ensemble est : `re`.
> **Exemple 2 :** Soit l'ensemble de mots `retarder, retard, retardement`, le *pgpc* de cet ensemble est : `retard`.
> **Exemple 3 :** Soit l'ensemble de mots `avance, retard`, le *pgpc* de cet ensemble est : (la chaîne vide).

Dans la suite, vous représenterez un mot par un objet de type `String`. Vous vous limiterez à des mots composés uniquement de lettres minuscules non accentuées.

1 Liste de mots

Dans cette section, vous allez manipuler un ensemble de mots à l'aide d'une *liste*. Vous utiliserez les classes du *package* **liste** de l'API locale pour vos listes.

1) Définissez une classe `Test` contenant la méthode `main` dans laquelle vous construirez une liste de 3 mots ayant un plus grand préfixe commun de 2 lettres minimum.

2) Vous allez maintenant programmer la recherche du plus grand préfixe commun dans une liste de mots. Pour cela vous utiliserez les deux méthodes qui suivent.

2.a) Définissez dans `Test` la méthode `pgpc` dont voici l'en-tête :

```
/**
 * Rôle : retourne le plus grand préfixe commun de deux mots.
 *
 * Antécédent : mot1 et mot2 sont initialisés.
 *
 * @return une chaîne de caractères qui est le plus grand préfixe
 *         commun à mot1 et mot2.
 */
private static String pgpc(String mot1, String mot2)
```

2.b) À présent, définissez dans `Test` la méthode dont voici l'en-tête :

```
/**
 * Rôle : retourne le plus grand préfixe commun d'une liste de
 *        mots.
 *
 * Antécédent : la liste l est initialisée et ses éléments sont
 *              tous du type String.
 *
 * @param l une liste de mots.
 * @return une chaîne de caractères qui est le plus grand
 *         préfixe commun aux mots de l.
 */
public static String pgpc(Liste l)
```

2.c) Ajoutez dans `main` l'affichage du *pgpc* de la liste définie dans l'exercice 1.

2 Des mots dans un arbre de lettres

Vous allez maintenant manipuler un ensemble de mots grâce à un *arbre de lettres* modélisé par la classe abstraite `ArbreLettre` de l'API. Dans un tel arbre, chaque nœud a pour valeur une lettre, à l'exception de la racine qui porte une valeur particulière inutilisée. En plus de sa `valeur`, un nœud dispose d'un attribut `niveau` qui correspond à la position de la lettre du nœud dans un mot. Enfin, un nœud peut aussi correspondre à la dernière lettre d'un mot (notez que ce n'est pas équivalent à être une feuille) auquel cas l'attribut `finMot` permet de le signaler. La figure 1 (page suivante) illustre par un exemple le concept d'arbre de lettres.

3) Familiarisez-vous avec la classe `ArbreLettre` grâce à la documentation de l'API locale. Définissez ensuite une classe `MonArbreLettre` qui hérite de `ArbreLettre` et que vous munirez de deux constructeurs : un sans argument, et un autre avec deux arguments permettant d'initialiser le `niveau` et la `valeur` d'un nœud. Ce dernier constructeur peut émettre une exception `CaractèreInvalideException` si la `valeur` passée en argument n'est pas une lettre minuscule non accentuée.

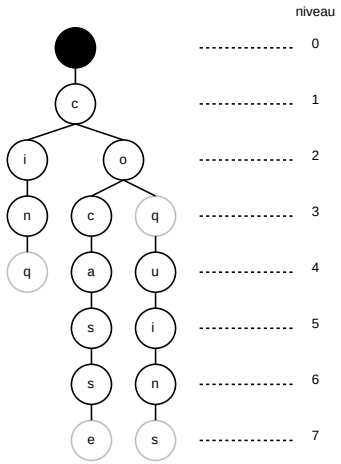

FIG. 1 – L'arbre de lettres correspondant à l'ensemble de mots : `cinq, coq, coquins, cocasse`. La racine est représentée par un nœud rempli de noir ; les nœuds marquant une fin de mot sont cerclés en gris ; les nœuds de même niveau sont alignés horizontalement.

4) Ajoutez dans `MonArbreLettre` les deux méthodes dont les en-têtes sont :

4.a) `public boolean racine()` qui indique si le nœud courant est la racine de l'arbre de lettres.

4.b) `public boolean existeFils(char lettre)` qui indique si le nœud courant possède un fils correspondant à la lettre `lettre` .

5) Donnez maintenant une définition à la méthode `ajouter` héritée de `ArbreLettre` . Cette méthode est destinée à être appelée sur la racine d'un arbre de lettres dans lequel elle ajoute le mot passé en paramètre. Attention, la méthode peut émettre une exception (*cf.* la documentation).

6) Dans la méthode `main` de `Test` , construire l'arbre de lettres (appelez-le `adel`) contenant les mêmes mots que la liste de l'exercice 1 (ou de nouveaux mots respectant les mêmes contraintes si vous n'avez pas fait cet exercice). Affichez l'arbre obtenu en utilisant la méthode `toString` de `ArbreLettre` .

7) Dans `MonArbreLettre` , ajoutez la méthode publique et sans arguments `pgpc` retournant une chaîne de caractères qui est le plus grand préfixe commun des mots de l'arbre courant. Dans `main` , utilisez cette méthode pour afficher le *pgpc* de l'arbre `adel` précédemment construit.

8) Parcours préfixé d'un arbre de lettres.

8.a) Dans `MonArbreLettre` , ajoutez la méthode `parcoursPréfixe` qui réalise un parcours préfixé de l'arbre de lettres. Cette méthode prend en argument un objet du type `Opération` et appelle la méthode `traiterNoeud` de cet objet sur tous les nœuds parcourus.

8.b) Le fichier `AfficherArbre.java` contient le squelette d'une classe implémentant l'interface `Opération` . Modifiez-le afin que, conjugué avec un parcours préfixe, les mots de l'arbre parcouru soient affichés dans l'ordre alphabétique.

8.c) Dans `main` , effectuez un parcours préfixe de l'arbre `adel` qui affiche les mots qu'il contient dans l'ordre alphabétique.

Bon courage et à bientôt pour les projets.