Programmation Orientée Objet
TP 2 : chaînes de caractères et encapsulation des données.

https ://dpt-info.u-strasbg.fr/~kvanhoey

Introduction

Le code écrit pour les exercices de ce TP devra être remis sur la page Moodle de ce module d'ici le début de la prochaine séance. Le code devra être clair et commenté. Vous devrez également rédiger un rapport contenant les réponses aux questions posées, ainsi que tout ce qui vous semblera pertinent : remarques sur votre code, stratégie d'implémentation et difficultés rencontrées.

1 Classe « String »

Le but de cet exercice est d'explorer la classe `java.lang.String` en testant ses différentes méthodes sur des chaînes et d'autres valeurs lues au clavier. Il s'agit d'écrire la classe exécutable `TestChaines` dont la méthode principale effectue les opérations suivantes :

1. Lire une chaîne $s1$ et afficher sa longueur (méthode *length*) ;
2. Lire deux chaînes $s1$ et $s2$ et afficher les résultats renvoyés par les expressions :

```
s1 == s2
s1.equals(s2)
s1.compareTo(s2)
s1.compareToIgnoreCase(s2)
```

3. Entre autres, essayer les couples « **abcd** » et « **abcd** », puis « **abcd** » et « **AbcD** ».
4. Lire deux chaînes $s1$ et $s2$ et afficher la réponse à la question : « ces deux chaînes commencent-elles par le même caractère ? ». Utiliser la méthode d'instance charAt ;
5. Lire deux chaînes $s1$ et $s2$ et afficher la réponse aux questions :
   – $s1$ commence-t-elle par s2
   – $s1$ finit-elle par s2
   – $s1$ contient-elle $s2$?
   Remarque : référez-vous aux méthodes *startsWith*, *endsWith* et *contains*
6. Lire deux chaînes $s1$ et $s2$ et concaténer $s2$ à s1. Utilisez l'opérateur + ou la méthode *concat*.
7. Lire une chaîne $s1$ et remplacer toutes les instances de caractère **o** par **O** (méthode *replace*).
8. Lire deux chaînes $s1$ et $s2$ et si $s1$ contient $s2$, renvoyer $s1$ privée de $s2$ sinon renvoyer $s1$. Utiliser les méthodes *indexOf* et *substring*.
9. Lire une chaîne représentant un nom de ville et l'afficher entièrement en majuscules (méthode *toUpperCase*).

10. Lire une chaîne représentant un nom de ville, lui enlever les éventuels blancs en début et à la fin (méthode *trim*).
11. Lire une chaîne $s1$ comportant plusieurs noms de villes séparés par un et un seul « » (blanc). Décomposer la chaîne pour accéder directement à chacun des noms de ville grâce à la méthode *split*. Afficher un nom de ville par ligne. Exemple d'appel :

```
TestChaines "Paris Lyon Marseille Strasbourg"
```

2 Classe « StringBuffer »

Les objets de type String ne sont pas modifiables. Leur manipulation conduit à la création de nouvelles chaînes et de petits changements dans le contenu de la chaîne peuvent être coûteux. Dans les programmes manipulant intensivement les chaînes, la perte de temps et la place mémoire sont importantes. En particulier l'opérateur $+=$ est très inefficace. Par exemple, $str+='A'$ est implémenté comme $str = str+'A'$. Cela signifie qu'une nouvelle chaîne est créée dont la valeur est le résultat de $str+'A'$ et str référence une nouvelle chaîne.

Faire le test suivant :

```
lire une chaine str
afficher str+'A' ou str.concat('A')
afficher str
```

La classe `StringBuffer` permet de manipuler les chaînes en modifiant les objets. Effectuer les opérations suivantes sur la classe `StringBuffer` :

1. Utiliser les 3 constructeurs de la classe :

```
public StringBuffer() (longueur = 0)
public StringBuffer(int capacity) (longueur = 0)
public StringBuffer(String str)
```

2. Utiliser les méthodes pour modifier votre chaîne : *append*, *insert*, *setCharAt*, *replace*, *reverse*, ...
   Noter que le $3^{\text{ème}}$ argument de la méthode replace est de type `String` ; `StringBuffer` n'étant pas une sous-classe de `String`, on ne peut transmettre un `StringBuffer` en 3ème argument.
3. Faire le même test que pour une instance de `String` pour vérifier que l'instance de `StringBuffer` est modifiable.

3 Parseur de fichiers

Le but de cet exercice est d'écrire un parseur de fichiers qui va convertir un fichier texte en un autre, tel que le nouveau fichier contienne le même texte mais dont tout les mots débutent par une majuscule. Pour cela, vous aurez à manipuler des chaînes de caractères, la lecture/écriture de fichiers étant fournie.

Vous trouverez dans l'archive TP2_Parseur.zip un répertoire contenant un fichier texte à parser, une classe `Parser` munie d'un *main* qui lit/écrit des fichiers. Ce code utilise une classe `Word` (représentant un mot) que vous aurez à écrire et dont la documentation est fournie dans le répertoire *doc/*. Implémentez cette classe de façon à ce que cet algorithme puisse fonctionner.

Quelques consignes et contraintes :

- Faites en sorte qu'une instance de `Word` contienne dès sa construction et de façon définitive et inchangeable une chaîne de caractères représentant un mot.
- Pour la méthode *countNbWords*, vous pouvez vous aider de la classe *StringTokenizer* de l'API Java.
- Pour la méthode *toFirstUpperCase*, vous pouvez vous aider de la classe *StringBuffer* de l'API Java.
- Les plus motivés d'entre vous pourront remplacer le tableau renvoyé par la méthode *cutPhrase2Words* par un `Vector`.