Programmation Orientée Objet
M2 CCI 2008–2009
hoenen@icps.u-strasbg.fr

Université de Strasbourg
Département d'informatique
23 janvier 2009

TP4 : String et conteneurs

1 Chaînes de caractères

La classe `String` Cette classe représente en Java une séquence de caractères. On peut créer un String simplement à l'aide d'une chaîne de caractères constante (à gauche), ou à partir d'un tableau de caractères (à droite) :

```
String mStr = "message";
```

```
char[] mTab = {'m','e','s','s','a','g','e'};
String mStr = new String(mTab);
```

Le langage Java supporte l'opérateur de concaténation `+` entre les Strings en appelant la méthode `append` (des classes `StringBuilder` ou `StringBuffer`), ainsi que la conversion de n'importe quel type d'objet vers un String à l'aide de la méthode `toString` (classe `Object`). Les String en Java sont des constantes, et ne peuvent par conséquent pas être modifiées après leur initialisation (les méthodes qui rajoutent en éliminent des caractères dans une chaîne ne modifient pas cette chaîne mais en renvoie une nouvelle).

1. Écrire un petit programme pour tester les méthodes suivantes de manipulation de `String` à partir de tableaux de caratères :
   - une méthode qui prend une chaîne de caractère et renvoie son inverse,
   - une méthode qui prend deux chaînes de caractères et renvoie la plus grande des deux dans l'ordre lexicographique,
   - une méthode qui prend une chaîne de caractère et détecte si, oui ou non, il s'agit d'un palindrome,
   - une méthode qui prend en entrée une phrase (une chaîne de caractère qui peut contenir des espaces) et renvoie un tableau contenant les mots de cette phrase (un mot est une suite de caractères ne contenant aucun espace).

La classe `StringBuffer` Cette deuxième classe permet également de représenter une séquence de caractères, mais cette fois cette séquence est modifiable. Un `StringBuffer` possède une capacité (le nombre de caractères qui peuvent y être stockés) qui vaut par défaut 16 mais qui peut aussi être spécifiée à la création de l'objet et peut être agrandie (automatiquement) en cas de dépassement.

2. Les méthodes précédentes peuvent-elles être implantées de manière simplifiée en utilisant des objets de type `StringBuffer`, ou en faisant appel à des méthodes existantes dans l'API Java dans les classes `String` et `StringBuffer` ? Faire les modifications lorsque c'est effectivement le cas.

2 Tris sur tableaux d'entiers

Faire un petit programme java pour tester les différentes méthodes ci-dessous et afficher le résultat.

1. Le *tri par sélection* consiste rechercher à partir du premier élément, le plus petit élément d'un tableau pour le placer en tête. Puis de recommencer à partir du deuxième élément du tableau. Écrire une méthode qui réalise le tri dans l'ordre croissant sur un tableau d'entiers par cette méthode.

2. Le *tri à bulle* consiste à parcourir tous les éléments d'un tableau et si l'élément $i+1$ est inférieur à l'élément i, alors intervertir ces deux valeurs. On recommence tant que le tableau n'a pas été parcouru sans aucune permutation. Écrire une méthode trie un tableau d'entiers dans l'ordre croissant en utilisant la méthode *à bulle*.

La classe `Arrays` Cette classe du paquetage `java.util` contient uniquement des méthodes statiques qui permettent la manipulation de tableaux. Il y a notamment des méthodes pour le tri (`sort`), pour le remplissage d'un tableau (`fill`), pour la recherche binaire (`binarySearch`) et pour l'affichage d'un tableau (`toString`).

3. Effectuer le tri dans l'ordre croissant et l'affichage du tableau trié en utilisant les méthodes statiques de la classe `Arrays`.

Performances La classe `java.lang.System` contient notamment des méthodes permettant d'estimer le temps entre le 1er janvier 1970 et le moment où la méthode est appelée. C'est par exemple le cas pour la méthode `currentTimeMillis` qui retourne cette estimation en millisecondes. On peut donc calculer le temps qui s'est écoulé durant l'exécution d'une partie du code en appelant cette méthode avant et après le bloc de code à *bencher* et en calculant la différence entre les deux valeurs reçues.

4. Comparer les temps des différents algorithmes de tri sur un même exemple, un tableau de 1000 entiers générés aléatoirement et dont la valeur varie entre 1 et 1000. Une fois le tableau généré, le copier à l'aide de la méthode `clone()` de la classe `Object` pour pouvoir passer un objet identique en argument des différentes méthodes de tri.

3 Tableaux dynamiques

Interface `Comparable` Une classe qui implémente l'interface `Comparable` doit fournir le code pour la méthode `int compareTo(T o)`. Cette méthode prend un objet de type `T` et renvoie un entier négatif, nul ou positif si l'objet depuis lequel la méthode est appelée est plus petit, égale ou plus grand que l'objet passé en paramètre. Idéalement, il faudrait que si `x.compareTo(y)==0` renvoie la même valeur que `x.equals(y)`.

1. Étendre la classe `Personne` pour lui permettre d'implémenter l'interface `Comparable`.

Classe `Vector<T>` Un objet de type `Vector<T>` correspond à un tableau contenant des objets de type `T`. À la différence des tableaux classiques, la taille d'un `Vector` peut être modifiée après sa création, notamment de manière automatique en cas de dépassement de capacité lors de l'ajout d'un élément supplémentaire (méthodes `add` ou `insertElementAt`).

2. On veut maintenant gérer une liste de personnes à l'aide d'un `Vector`. Construire une telle liste.

Parcours d'un conteneur Un *itérateur* est un objet qui permet de parcourir de manière séquentielle les éléments d'un conteneur. La classe `Vector` hérite de la classe `AbstractList` qui implémente des interfaces `Itarator` et `ListIterator`, et possède des méthodes `iterator()` et `listIterator()` pour renvoyer un itérateur sur les éléments contenus. Les itérateurs de type `ListIterator` fournissent plus de fonctionnalités que ceux du type `Iterator` : la liste peut être parcourue dans les deux sens et on peut ajouter et supprimer des éléments parcourus.

3. Afficher la liste dans l'ordre croissant en utilisant la méthode `sort` de la classe `Arrays`.
4. Afficher le plus grand élément de la liste en utilisant un itérateur.
5. Écrire une méthode qui ajouter un élément à la bonne place dans une liste triée dans l'ordre croissant.