Licence Informatique **UE INF203**
Université Bordeaux 1 **Programmation par objets**
2006-2007 **Travaux Dirigés**
feuille 2

Se placer dans le répertoire `programmation-par-objets\src`
Télécharger les fichiers qui accompagnent le td à l'aide de la commande
`wget http://dept-info.labri.fr/ENSEIGNEMENT/poo/src/fichiers-2.tar.gz` puis décompacter cette archive au moyen de la commande `tar -xvzf fichiers-2.tar.gz`. Se placer dans le répertoire td02

Tableaux, tableaux de tableaux

Exercice 1. Soit $E = \{0, 1, \ldots, n\}$.
Une relation binaire R sur E est un sous-ensemble de $E \times E$ constitué des couples (i, j) tels que iRj.

On représente une relation binaire R par une matrice carrée de booléens de dimension $n + 1$, notée `relation`, telle que :

`relation`$[i][j]$ a la valeur *true* si iRj, $false$ sinon.

On testera les fonctions au fur et à mesure de leur écriture à l'aide de la classe TestRelation fournie.

1. Une relation R est réflexive si et seulement si $\forall x \in E, xRx$.
   Écrire une fonction `public static boolean reflexive(boolean[][] relation)` qui retourne *true* ssi la relation passée en paramètre est réflexive.
2. Cette question ne sera pas traitée en Td.
   Une relation R est symétrique si et seulement si $\forall x, y \in E$, $xRy \Rightarrow yRx$.
   Écrire une fonction `symetrique` qui retourne *true* ssi la relation passée en paramètre est symétrique.
3. Cette question ne sera pas traitée en Td.
   Une relation R est transitive si et seulement si $\forall x, y, z \in E$, xRy et $yRz \Rightarrow xRz$.
   Écrire une fonction `transitive` qui retourne *true* ssi la relation passée en paramètre est transitive.
4. R et S étant 2 relations binaires sur E, la composée V de ces 2 relations est la relation définie par : $xVz \Leftrightarrow \exists\, y \in E \;/\; xRy$ et ySz
   Écrire une fonction `composée` qui retourne la composée des 2 relations passées en paramètres.
5. L'image de `i` pour la relation R est définie par : $image_R(i) = \{j \in E / iRj\}$.
   Écrire une fonction `image` qui retourne l'image de `i` pour cette relation.

Exercice 2. Écrire un programme `Combinaisons` qui calcule et affiche les n+1 premières lignes du triangle de Pascal. Le programme comportera une fonction de calcul du triangle et une fonction d'affichage du tableau.
Par exemple `Combinaisons 3` calcule et affiche :

```
1
1   1
1   2   1
1   3   3   1
```

`Rappel` : $\forall n \in N,\ C_n^0 = C_n^n = 1$ $\forall (n, p) \in N^2,\ 1 < p < n,\ C_n^p = C_{n-1}^{p-1} + C_{n-1}^p$

Passage de paramètre par valeur, par référence

Exercice 3.

1. Étudier le code des classes `ValeurReference` et `Entier` fournies. Compiler et exécuter la classe `ValeurReference`. Expliquer le résultat obtenu.
2. La classe `Entier` est mal écrite. Elle viole plusieurs principes vus en cours (encapsulation des données..). Les classes `Entier2` et `ValeurReference2` en tiennent compte. Étudier le code de ces classes.

Chaînes de caractères : classe String

Exercice 4. Dans cet exercice les seules méthodes de la classe String que l'on utilisera sont les méthodes d'instance **charAt** et **length**

Écrire un programme **Chaines** comportant les méthodes suivantes (on complétera le programme fourni) :

1. Une fonction booléenne **chainesEgales** qui retourne true si les deux chaînes fournies en paramètre, sont égales et false sinon.
2. Une fonction **extraire(String s, int début, int fin)** qui retourne la sous-chaine extraite de s de l'indice **debut** à l'indice **fin-1**.
3. Une fonction booléenne **estSousMot(String u, String v)** qui retourne true si **u** est un sous mot de **v**, false sinon.
   Rappel : Soit le mot $v = v_0 v_1 \dots v_{n-1}$ où les v_i sont les n lettres de v. Le mot u de longueur p est un sous mot du mot v s'il existe un p-uplet $(i_0, \dots, i_{p-1})$ tel que
   $0 \leq i_0 < i_1 < \dots < i_p - 1 < n$ *et* $u = v_{i_0} v_{i_1} \dots v_{i_{p-1}}$.
   Intuitivement, on obtient u en supprimant certaines lettres de v. Exemple : $u = accb$ est sous-mot de $v = bbabcbbcaba$ car $u = v_2 v_4 v_7 v_9$.

Les mots utilisés et les indices seront les arguments du programme.

Exercice 5. Consulter la documentation de la classe String et plus particulièrement celle des méthodes d'instance **equals**, **indexOf** et **substring**.
Écrire un programme **Chaines2** correspondant au programme **Chaines** mais utilisant au maximum les méthodes de la classe String (on complétera le programme Chaines2 fourni).

Les exercices qui suivent ne sont pas traités dans cette séance.
Ils sont à faire pour avant le prochain td.

Chaînes de caractères : classe StringBuffer

Exercice 6. On considère le programme StringDemo dont les sources sont fournis. À l'aide de la documentation de la classe StringBuffer, analysez le source de ce programme.

Que fait le programme ?
Exécuter ce programme. Expliquer le résultat obtenu.

Écrire une classe

Exercice 7. On considère les classes `Entier2` et `ValeurReference2` definies précédemment.

1. Ajouter à la classe `Entier2` une variable de classe (static) qui compte le nombre d'objets construits. On modifiera le constructeur en conséquence.
   Écrire, dans la classe `Entier2`, une fonction `nombre` qui retourne le nombre d'objets construits et, dans la classe `ValeurReference2`, un appel à cettte fonction.
2. Écrire une fonction `rangeValeurs` qui modifie trois objets de type Entier2 x, y et z de telle manière que leurs valeurs se retrouvent en ordre croissant.
   Par exemple, si les valeurs de x, y et z sont dans cet ordre 7, 3 et 6, après l'appel à `rangeValeurs` elles devront être 3, 6, 7.