Télécom Saint-Étienne 2A
Année universitaire 2012-2013
Java INFO 4.1
TP n°2 - 3h00

Les enseignants du module : J. Fayolle, C. Gravier, F. Laforest, J. Subercaze

OBJECTIFS DU TP:

L'objectif de ce TP est de vous familiariser avec la manipulation des collections en Java Vous allez notamment apprendre à :

- Utiliser les différents types de collections ainsi que les `Maps`
- Définir des comparateurs entre éléments
- Utiliser les outils d'analyse de mémoire fournis par le JDK 7

1 Exercice 1 : Collections, comparateurs

1.1 Partie A : prise en main

Dans cet exercice, vous devez créer une liste de `Double` (avec l'implémentation de votre choix), puis y ajouter 10 éléments générés aléatoirement. Pour cela utilisez la méthode `Math.random();`. Affichez la collection générée.

Ensuite lisez la documentation de la classe `Collections` pour trouver la méthode qui retourne le plus grand élément d'une collection. Utilisez la pour afficher le plus grand chiffre de votre collection.

1.2 Partie B : objets et comparateur

Définissez un objet `Rectangle` qui possède deux attributs, longueur et largeur qui sont des `double`. Ecrivez dans cette classe, une méthode statique qui permet de générer aléatoirement un rectangle. La signature de cette méthode est la suivante :

```
public static Rectangle genereRectangle()
```

Essayez maintenant de retourner le plus grand rectangle en utilisant la même méthode que dans la partie précédente, que se passe-t-il ?

Pour pouvoir comparer les différents rectangles entre eux, il faut spécifier comment les comparer. Pour cela on implémente l'interface `Comparable` en changeant la déclaration de la classe en :

```
public class Rectangle implements Comparable<Rectangle> {
```

Une erreur s'affiche alors, cliquez sur la croix rouge à gauche de la ligne et choisissez "Add unimplemented methods". Une méthode `public int compareTo(Rectangle o) {` est automatiquement rajoutée en bas de votre classe.

Cette méthode permet de comparer deux rectangles entre eux, le rectangle courant, que l'on identifie par `this` et le rectangle auquel il est comparé qui est passé en paramètre, `Rectangle o` par défaut.

Ecrivez le comparateur qui compare les rectangles selon leur surface, la méthode renvoie 1 si `this` est plus petit que `o`, 0 et -1 pour l'égalité et plus petit. Affichez ensuite le rectangle le plus petit.

On souhaite maintenant ajouter une couleur au rectangle, qui sera soit rouge, vert ou bleu (Utilisez une `Enum` pour définir les couleurs). On définit arbitrairement l'ordre suivant entre les

couleurs : *rouge* < *vert* < *bleu*. Cet ordre est utilisé en cas d'égalité entre la surface des rectangles. Modifiez le comparateur pour intégrer cet ordre secondaire, puis testez votre programme en entrant des valeurs adéquates pour les rectangles de votre liste.

1.3 Partie C : Maps

Une utilisation classique des `Maps` est celle de dictionnaire, c'est-à-dire qu'à un mot on associe sa définition ou sa traduction dans une autre langue. Dans un premier temps vous réaliserez un dictionnaire bilingue français → anglais, puis un dictionnaire multilingue.

Ouvrez le fichier `Exercice1c.java` que vous devrez compléter dans cet exercice.

1. A l'aide d'une `Map` (implémentation au choix), écrivez un programme qui permet de
   (a) Entrez un mot puis sa traduction dans le dictionnaire. Utilisez la méthode `String lireUneligne()` qui vous est fournie.
   (b) Affichez le contenu du dictionnaire.
2. Créez une nouvelle fonction qui permet de rechercher un mot dans le dictionnaire et qui affiche la traduction si le mot est présent.
3. Pour créer un dictionnaire multilingue, comment allez-vous procéder ? Proposez votre solution à l'enseignant, puis implémentez un exemple simple (entrez un mot avec ses traduction en anglais et en allemand).

2 Exercice 2 : occupation mémoire

La gestion de la mémoire en JAVA est assurée par la machine virtuelle (JVM) qui exécute vos programmes. La JVM Oracle fournit des outils pour analyser la gestion de la mémoire. Dans ce TP sur les collections, nous sommes amener à manipuler de grand nombre d'objets, c'est l'occasion idéale pour découvrir les outils d'analyse de mémoire. Dans cette partie vous allez apprendre à prendre en main l'outil `VisualVM`.

Dans cet exercice, on souhaite mesurer l'occupation en mémoire des différentes implémentations de l'interface `List`.

Pour ce faire nous allons utiliser les trois implémentations vues dans le cours précédent :

- `ArrayList`
- `LinkedList`
- `Vector`

Ouvrez le fichier `Exercice2.java` et modifiez-le afin que votre programme réalise les fonctions suivantes :

- Instancier trois listes d'entiers en utilisant les trois types d'implémentation de `List`.
- Remplir chacune de ces listes de la manière suivante : de $i = 0$ à $i < LONGUEUR\_LISTE$ ajouter i dans la liste. De cette manière toutes les listes contiennent les mêmes entiers.
- Utiliser la méthode `pause()` fournie pour que votre programme s'arrête :
  - Avant le remplissage de chaque liste (même de la première)
  - Après le remplissage de la dernière liste

La JVM stocke les objets créés par les programmes dans une zone mémoire appellée *heap*, dont la taille est variable entre des valeurs *min* et *max*. Le dépassement de mémoire par rapport à la *heap* allouée à un moment t peut occasionner le démarrage du $garbage - collector$, ce que l'on souhaite éviter dans notre cas pour visualiser correctement l'occupation mémoire. Nous allons donc lancer notre programme avec une *heap* constante à 512Mo. Pour cela :

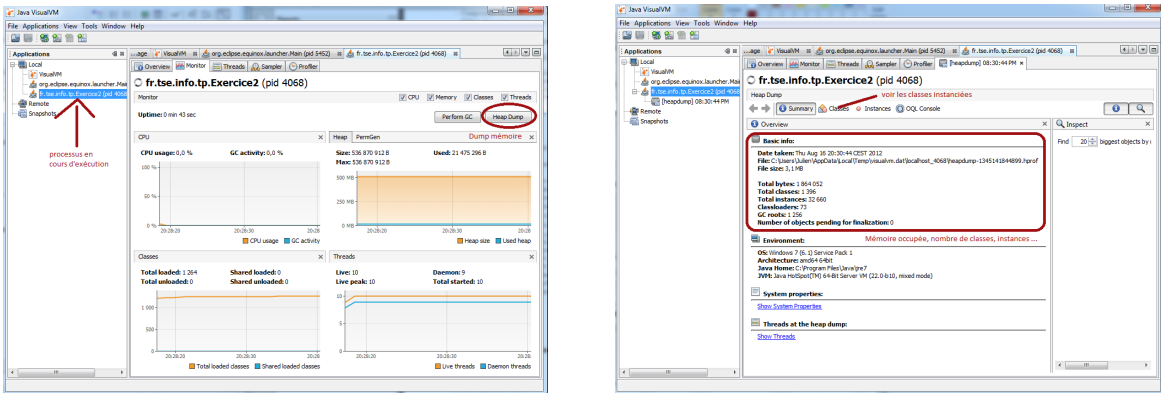
1. Clic-droit sur votre fichier `Exercice2.java` dans le "Package Explorer", puis choisissez "Run As" puis "Run configurations"
2. Dans l'onglet "Arguments", dans la zone "VM arguments", indiquez les valeurs suivantes : "-Xms512M -Xmx512M"

3. Puis cliquez sur "Run"

Votre programme est maintenant en pause avant le remplissage de la première liste. Pour démarrer l'outil VisualVM effectuez les opérations suivantes :

1. Démarrer l'explorateur windows
2. Se placer dans le répertoire C:\Program Files\Java\jdk1.7.X_XX\bin
3. Lancer *jvisualvm.exe*

Les deux captures d'écran suivantes vous indiquent comment utiliser VisualVM pour trouver les informations importantes. Le bouton "Heap Dump" permet de créer une copie de la mémoire à un instant t pour pouvoir en analyser le contenu, c'est cet outil que nous allons utiliser dans cet exercice.



(a) (b)

FIG. 1 – *Utilisation de Visual VM*

A faire : Pour les tailles de liste suivantes : 100, 1000, 10000, 100000, reporter la valeur "total bytes" dans un tableur. Calculer l'occupation mémoire pour chacune des trois implémentations (ArrayList, LinkedList, Vector) et calculer le nombre d'octets utilisé pour stocker un élément dans chaque liste. Quelle est celle qui occupe le moins de place ? Quelles conclusions tirez-vous de ces mesures ?

3 Exercice 3 (optionnel): aller plus loin

En complément à la mesure de l'occupation mémoire, le temps d'exécution pour les différentes opérations :

1. Ajouter
2. Accéder au i-ème élément
3. Rechercher une élément
4. Supprimer un élément

est déterminant pour choisir quelle implémentation utiliser selon les besoins de votre application.

Dans cet exercice, écrivez une procédure de test pour évaluer les performances des quatre opérations puis implémentez la.

Astuce : Pour mesurer le temps écoulé entre deux opérations, on fait appel deux fois à la méthode `System.currentTimeMillis();` qui donne le temps actuel en millisecondes, puis on soustrait les valeurs pour en déduire le temps écoulé.

Encore plus loin : Via Maven, importer la librairie Guava qui implémente aussi le framework Collections, puis comparer les performances avec celle fournit dans le JDK. Pour aller encore plus loin, comparer avec l'implémentation fournie par Apache Commons Collections.