TP 3 : Structures récursives

Objectif du TP

L'objectif de cette séance est de développer des structures de données récursives (majoritairement des *listes*) et des algorithmes les manipulant.

Recommandations

Immédiatement après chaque séance de TD/TP, chaque étudiant ou groupe d'étudiants enverra un mail à l'enseignant responsable du groupe un compte rendu de l'activité réalisée pendant la séance au format PDF dans lequel les réponses aux questions auront été saisies. Il est fortement conseillé de faire des copies d'écran de vos programmes pour aller plus vite.

Le sujet du mail devra obligatoirement respecter le format suivant : **[S3T ou S3D ou S3A][M313][Gx][Seance_y] Nom1 / Nom2** ou *x* représente le numéro de groupe et *y* le numéro de la séance (par exemple : mailto:denis.pallez@unice.fr?subject=[S3A][M313][G1][Seance2]Turing). Le fichier PDF devra également être renommé de la façon suivante : **<NomEtudiant1_NomEtudiant2>_TP<N° du TP>_<date de la séance de TP en anglais>.pdf** (par ex : Pallez_TP1_20141306.pdf).

Vous ne pouvez pas faire plus de 2 séances avec le même binôme et vous ne pouvez pas faire plus de la moitié des séances seul. Tout manquement à ces règles pourra entrainer des retraits de points sur la note de contrôle continu.

Exercice 1 Listes chainées type1

a) Implémenter la première structure de données `Liste` vu en cours ;
b) Programmer la méthode `longueur()` qui calcule la longueur de la liste ;
c) Créer une classe de test et tester le calcul de la longueur avec une liste à 0, 1 et plusieurs éléments ;
d) Créer la version fonctionnelle récursive de l'ajout d'une valeur à la fin de la liste.

Exercice 2 Listes chainées type 2

a) Récupérer le fichier `ListeEntier.java` qui implémente le type `Liste` d'entiers;
b) Créer la méthode récursive `estDans(int v)` qui précise si la valeur v se trouve dans la collection ;
c) Ecrire une procédure `vider()` qui vide la liste ;
d) Créer la méthode `finalize()` de la classe `Liste` ;
e) Créer une nouvelle classe de `Test` (initialiser un générateur de nombre aléatoire dans une méthode `@BeforeClass`) et implémentez les axiomes du TAD `Collection` vu en cours ;
f) Créer une méthode récursive `inverse()` fonctionnelle et procédurale qui inverse la liste ;

Exercice 3 Crible d'Ératosthène (Wikipédia)

Le crible d'Ératosthène est un procédé qui permet de trouver tous les nombres premiers inférieurs à un entier naturel *n* donné. Le principe est le suivant :

1. Une collection *c* contenant tous les nombres compris entre 2 et le nombre *n* est créée ;
2. Pour tous les nombres *i* contenus dans c, supprimer l'ensemble des nombres *j* situés après *i* qui sont multiples de *i* dans c ;
3. Une fois terminé, la collection *c* contient l'ensemble des nombres premiers compris entre 2 et *n*.

Ecrire une méthode récursive fonctionnelle et procédurale qui permet de calculer les nombres premiers en utilisant le crible d'Ératosthène.

Exercice 4 Liste Générique

a) Dorénavant, on souhaite créer des listes de types quelconques. Pour cela, nous allons utiliser les patrons de classe et remplacer le type entier par un type quelconque que nous appellerons `T`. Créez une nouvelle classe `Liste` basée sur `ListeEntier` (copier-coller) en remplaçant `int` par `T`. Pour que cela fonctionne, Java demande à paramétrer la classe avec ce type `T` juste après le nom `class Liste <T>`. Réécrivez la classe pour qu'elle puisse gérer n'importe quel type de données. Indications : sous Eclipse, il ne doit pas y avoir de soulignements oranges avec comme commentaires `Liste is a raw type …`.
b) Faites les tests sur cette classe.
c) Créez une classe `Personne` qui modélisera une personne avec un nom et une taille. Avec le menu `Source > Generate Getters and Setters`, générer automatiquement les accesseurs.
d) Créer une liste de personnes de noms quelconques et de tailles différentes.

Exercice 5 Liste ordonnée

a) Créer un nouveau type de données `OrderedListe<T>` qui hérite de `Liste<T>` et qui maintient en permanence une liste ordonnée de valeurs.
b) Créer la classe de test correspondante en testant sur des entiers.

Exercice 6 Liste doublement chainée

a) Comment faut-il modifier les classes `Cellule` et `Liste` pour pouvoir gérer des listes doublement chainées (on connait l'élément suivant *mais* aussi le précédent) ?
b) Créer une nouvelle classe `DoubleLinkedListe<T>` qui hérite de `Liste<T>` et qui implémente les listes doublement chainées.

Exercice 7 Fonctionnelle et procédurale

Créer les versions fonctionnelles des méthodes `addHead`, `addTail`, `delHead` et `delTail` dans la classe `Liste<T>`.

Pour aller plus loin

Exercice 8 Itérateur de liste

Un itérateur est un objet qui permet de parcourir de manière transparente toute collection d'éléments sans savoir comment la collection est implémentée. L'intérêt de définir un itérateur est de récupérer les éléments de la liste les uns après les autres.

La notion d'itérateur existe déjà en Java avec deux interfaces Iterator et Iterable. On souhaite écrire le code ci-contre en sachant que `lint` est une liste d'entiers sur laquelle on peut itérer (qui implémente `Iterable`). Comme la liste est itérable, elle doit être capable de renvoyer un itérateur sur notre liste. Pour cela, il est nécessaire de définir une nouvelle classe d'itérateur sur une liste (`ListeIterator`) qui implémentera `Iterator`. Cette dernière classe possède trois méthodes `hasNext()`, `next()` et `remove()`.

```
public void test() {
    Iterator<Integer> it = lint.iterator() ;
    while (it.hasNext()) {
        System.out.print(it.next()+",");
    }
    System.out.println("");
    for (Integer i:lint) {
        System.out.print(i+",") ;
    }
```

a) Nous devons tout d'abord définir une nouvelle classe `ListeIterator` qui implémente `Iterator` et qui redéfinit au moins les 2 méthodes `next()` et `hasNext()`. Pour définir ces méthodes, nous aurons besoin d'une référence sur l'élément courant référencé par l'itérateur.

b) Plutôt que de modifier la classe `Liste`, nous allons créer une nouvelle classe `IteratedListe<T>` qui héritera de `Liste` et qui implémentera `Iterable`.

Sources pour ce TP

Claire David, Structures de données et objets Java 2005-2006.

UKO, Tutoriel Java, Structures Chainées