Haute École Bruxelles-Brabant
École Supérieure d'Informatique
Bachelor en Informatique

2018 – 2019
DEV2 – JAVL

DEV2 – JAVL – Laboratoire Java

TD 4 – Listes

Résumé

Ce TD a pour but de vous familiariser avec les listes et les façons d'incrémenter une variable.

Mise en place

Pour coder ce TD, vous allez utiliser le dépôt créé lors du 1er TD.

1. Vérifiez que la version du dépôt sur le PC est synchronisée avec le dépôt sur GITLAB [a] : faites un *clone* s'il n'est pas encore présent sur le PC ou un *pull* si votre PC contient une vieille version du dépôt.
2. Créez un nouveau projet NETBEANS. (name = `td04-listes`, location = le chemin vers `DEV2-JAVL`, Group id = g12345, Version = 1.0, package = `g12345.dev2.td04` [b]).

a. Dans le cas contraire, il vous sera difficile d'y sauver votre travail plus tard.
b. Remplacez bien sûr 12345 par votre identifiant.

1 Listes

Dans ce TD nous vous demandons d'écrire une classe `TdListes`. Sauf mention explicite, chaque algorithme à produire sera implémenté sous forme de méthode statique de cette classe `TdListes`.

Pour chaque exercice, on vous demande d'écrire une *javadoc* de la méthode. On ne peut trop vous conseiller de vous référer à la documentation de `List` et de `Collections` afin d'y trouver des méthodes à utiliser au sein de votre code.

Version du document
14 février 2020

Exercice 1 — Mise en bouche

Écrivez une méthode principale qui :

- ▷ crée une liste vide (une `ArrayList<Integer>` [a]) ;
- ▷ y place les éléments 42 et 54, dans cet ordre ;
- ▷ remplace le 54 par la valeur 44 ;
- ▷ insère un 43 entre les deux valeurs de la liste ;
- ▷ supprime le dernier élément de la liste ;
- ▷ supprime la valeur 42 ;
- ▷ et, enfin, vide la liste.

Utilisez le débogueur [b] pour vérifier que chaque étape fonctionne correctement.

a. Pourquoi `Integer` et pas tout simplement `int` ?

b. Pour en savoir plus, on vous propose de lire ce petit document (`www-etud.iro.umontreal.ca/~gottif/bdeb/A09/DebogageNetbeans.pdf`) écrit par Fabrizio Gotti de l'université de Montréal.

Exercice 2 — Somme d'une liste (cf. exercice 3 du syllabus d'algorithmique)

Écrire un algorithme qui calcule la somme des éléments d'une liste d'entiers.

```
int sum(List<Integer> liste)
```

Exercice 3 — Une classe instanciable : `Nombres`

Pour pratiquer l'orienté-objet en parallèle avec la notion de liste, nous vous demandons maintenant d'écrire une classe instanciable `Nombres` comportant un attribut privé : une liste d'entiers que vous pouvez appeler `nombres`.

Le constructeur de cette classe initialisera la liste à une nouvelle liste (`ArrayList`) vide. Écrivez également trois méthodes :

```
void add(Integer val) // ajouter un élément à la liste
void remove(Integer val) // supprimer un élément de la liste s'il existe
Integer sum() // calculer la somme des éléments de la liste
```

Pour écrire les deux premières méthodes, utilisez les méthodes existantes de l'interface `List` ; pour la troisième méthode, faites appel à la méthode `TdListes.sum(...)` déjà écrite.

Voici un exemple de test pour votre méthode `sum` :

```
@Test
public void testSum_tousStrictementPositifs() {
  Nombres instance = new Nombres();
  instance.add(4);
  instance.add(12);
  instance.add(52);
  Integer expResult = 68;
  Integer result = instance.sum();
  assertEquals(expResult, result);
}
```

Exercice 4 — Concaténation de deux listes (cf. exercice 5 du syllabus d'algorithmique)

Écrire un algorithme qui reçoit deux listes et ajoute à la suite de la première les éléments de la seconde. La seconde liste n'est pas modifiée par cette opération. (Vous pouvez ne considérer des listes d'entiers pour le moment. Vous verrez ultérieurieurement comment écrire une méthode qui accepte n'importe des listes d'objets de type arbitraire.)

Ajouter ensuite une méthode à la classe `Nombres` permettant de concaténer une liste d'entiers à la liste courante.

Exercice 5 — Les extrêmes (cf. exercice 7 du syllabus d'algorithmique)

Écrire un algorithme qui supprime le minimum et le maximum des éléments d'une liste d'entiers.

Aide : que doit-il se passer si il y a plusieurs maxima ou plusieurs minima ? Pour simplifier, vous pouvez ne supprimer qu'une seule occurrence dans ces cas.

Voici un test unitaire :

```
@Test
public void testSupprimerMinEtMax() {
  List<Integer> liste = new ArrayList<>(List.of(1, 4, 2, 0));
  List<Integer> expected = List.of(1, 2);
  TdListes.supprimerMinEtMax(liste);
  assertEquals(liste, expected);
}
```

Pour `expected` on n'utilise pas le constructeur explicite `new ArrayList<>()`, mais on l'a utilisé pour `liste`, pourquoi ? Pour le comprendre, lisez la documentation de la méthode `List.of`.

Ajouter ensuite une méthode à la classe `Nombres` permettant d'obtenir cet effet sur la liste courante.

Exercice 6 — Éliminer les doublons d'une liste (cf. exercice 9 du syllabus d'algorithmique)

Soit une liste triée d'entiers avec de possibles redondances. Écrire un algorithme qui enlève les redondances de la liste.

Exemple : Si la liste est $(1, 3, 3, 7, 8, 8, 8)$, le résultat est $(1, 3, 7, 8)$.

On vous en demande 2 versions.

- ▷ Faites l'exercice en créant une nouvelle liste (la liste de départ reste inchangée)
- ▷ Refaites l'exercice en modifiant la liste de départ (pas de nouvelle liste)

Écrivez des tests unitaires avec les assertions adéquates permettant de tester non-seulement la suppression des doublons, mais aussi la contrainte de non-modification de la liste de départ (dans la première version).

Ajouter ensuite une méthode à la classe `Nombres` permettant d'éliminer les doublons de la liste courante. Laquelle des deux versions allez-vous utiliser, et pourquoi ?

Exercice 7 — Fusion de deux listes (cf. exercice 8 du syllabus d'algorithmique)

Soient deux listes triées d'entiers (redondances possibles). Écrire un algorithme qui les fusionne. Le résultat est une liste encore triée contenant tous les entiers des deux listes de départ (qu'on laisse inchangées).

Exemple : Si les deux listes sont $(1, 3, 7, 7)$ et $(3, 9)$, le résultat est $(1, 3, 3, 7, 7, 9)$.

Exercice 8 — Le plus tôt

Écrivez une méthode qui reçoit une liste de `Moment` (cf. TD2) et retourne le moment de la liste qui est le plus tôt dans la journée.

Pour intégrer la classe Moment à votre projet courant, assurez-vous que le projet du TD2 est bien fait avec Maven [1] et est ouvert dans Netbeans et cliquez-droit sur `Dependencies`,

1. Pour le voir au premier coup d'oeil : un projet de type `ant` aura des `Libraries`, tandis qu'avec Maven il s'agit de `Dependencies`

puis `Add Dependency...` et choisissez l'onglet `Open Projects`. Sélectionnez le projet du TD2.

2 Réflexion sur les post- et pré-incrémentation

On peut incrémenter une variable de plusieurs façons, par exemple :

```
i = i + 1;
i += 1;
i++; // post-incrémentation
++i; // pré-incrémentation
```

Exercice 9 Valeur des incrémentations

Toutes ces instructions modifient la valeur de `i` ; mais ce sont également des expressions (sans le point-virgule), c'est-à-dire qu'elles ont une valeur. Trois d'entre elles ont une valeur qui correspond à la *nouvelle valeur* de `i`. Quelle est l'expression dont la valeur est autre, et quelle est la valeur ?

Exercice 10

Qu'affiche le code suivant ?

```
int i;
System.out.println(List.of(i = 42, i++, --i));
```

Exercice 11

Que fait le morceau de code suivant dans les cas ci-dessous :

```
int[] tab = new int[5];
int i = 0;
while (i < tab.length) {
  // assignation
}
```

où l'assignation est :

1. `tab[i++] = i;`
2. `tab[i] = i++;`
3. `tab[++i] = i;`
4. `tab[i] = ++i;`

Exercice 12 Déverminage

Le code suivant devrait afficher les coordonnées entières de `[0 0]` jusque `[2 4]`. Pouvez-vous le corriger ?

```
int nbLignes = 3;
int nbCols = 5;
boolean stop = false;
int lg = 0, col = 0;
while (! stop) {
  System.out.printf("[%d %d]\n", lg, col);
  stop = ! (col++ < nbCols) && ! (++lg < nbLignes);
  col %= nbCols;
}
```

Notez bien : les codes précédents sont là pour lancer la réflexion sur le sens des opérateurs de pré- et post-incrémentation. Ce ne sont pas des exemples à suivre. N'oubliez pas qu'en pratique, un code plus court n'est pas forcément plus lisible ! Un code très court mais inhabituel risque d'être plus difficile à comprendre qu'un code plus long mais idiomatique.