Simplon : Java

Nicolas Rousset

Sujet : 1er août 2016

1 I. Sujets de la journée

Les principaux objectifs de cette semaine sont les suivants :

1.1 I.1 Pré-requis

Important

Toutes les ressources pour cette formation peuvent être trouvées à l'adresse :
https://github.com/Aenori/cours/tree/master/Cours_Simplon_Java_2016_08_01_05

1. Créez un projet eclipse java pour cette semaine (le nom n'a pas d'importance)
2. A l'intérieur de ce projet créez deux packages edu.simplon.data et edu.simplon.testunitaire
3. Eclipse va créer des répertoires correspondant sur votre ordinateur, copiez dans ces répertoires les fichiers suivants :

```
DataJour1.java dans edu.simplon.data
Item.java      dans edu.simplon.data
TestCaseWithIntrospection.java    dans edu.simplon.testunitaire
TestItemFunctionSimplonJour1.java dans edu.simplon.testunitaire
```

Vous pouvez obtenir l'adresse de votre workspace en allant dans file > switch workspace.
A noter que si vous avez mal nommer vos packages, il y aura une erreur, car le nom du package est inclus dans les fichiers, et il doit donc être cohérent par rapport à la structure du projet.

Vous pouvez maintenant lancer les tests unitaires en cliquant sur le projet puis "run as -> junit test".
Pour l'instant ceux-ci indiqueront tous qu'ils sont en état de "failure".
Il s'agit d'un usage un peu spécifique des tests unitaires à but pédagogique afin que chacun d'entre vous puisse tester son code.
A noter que normalement java provoque une erreur de compilation si on utilise

une méthode qui n'existe pas, on utilise ici un méthode assez particulière appelée "introspection" pour contourner ce problème.

Créer un package edu.simplon.datascience dans lequel vous allez créez une classe FonctionRecherche, en la dotant d'une fonction **main** que nous utiliserons aujourd'hui.
Lancez les tests unitaires, normalement il y en a un qui doit s'afficher en ok.

1.2 Exercice 1 : codage d'une fonction "afficher les éléments d'un tableau"

NB : le test unitaire sur cette fonction l'appelle sans vérifier le résultat

Codez une fonction **affichez_tous_les_elements** dans la classe FonctionRecherche, qui prend en entrée un tableau de Item (type Item[]), affiche tous les éléments de ce tableau et ne renvoit rien.
Si vous utilisez directement la fonction d'affichage sur les objets Item, est-ce que le résultats paraît satifaisant ? (normalement vous avez quelque chose du genre edu.simplon.data.Item@75ab4586)

1.3 Exercice 2 : codage d'une méthode toString

Ecrivez une méthode **toString** dans la classe Item, qui ne prend pas d'argument et renvoit une chaine de caractère représentant une item, de la forme :

```
id : {id}, nom : {nom}, prix : {prix}
```

Où les propriétés entre accolades sont remplacées par les attributs correspondant. Par exemple pour un item d'id 1, de nom "Truc" et de prix 5, cela donnerait :

```
id : 1, nom : Truc, prix : 5.0
```

1.4 Exercice 3 : éléments de prix minimums et maximums d'une liste

Ecrivez une fonction **prix_min_et_max_liste** dans la classe FonctionRecherche, qui prend en argument un tableau de Item, et qui renvoit un tableau de 2 double, le premier étant le prix minimum dand le tableau, le deuxième le prix maximum dans le tableau.

Quel est l'intérêt d'avoir une seule fonction qui fournit les deux plutôt que 2 fonctions ?

Testez le temps pris par votre fonction avec un code du type :

```
Item[][] all_item_tabs = new Item[][]{
    DataJour1.getItemTab10(),
    DataJour1.getItemTab100(),
```

```
    DataJour1.getItemTab1000(),
    DataJour1.getItemTab10000(),
    DataJour1.getItemTab100000() };
    DataJour1.getItemTab1000000() };

int nb_item = 10;
for( int i = 0; i < all_item_tabs.length ; ++i )
{
    long debut = System.currentTimeMillis();
    double[] result = prix_min_et_max_liste( all_item_tabs[i] );
   System.out.println( "Appel avec " + nb_item + " resultats : " + result[0] + " / " + result[1] +
    nb_item *= 10;
}
```

1.5 Exercice 4 : trouver l'ensemble des éléments compris dans une fourchette de prix

Ecrire une fonction **extraction_dans_une_fourchette_de_prix** dans la classe FonctionRecherche, qui prend en argument un tableau de Item et deux valeurs de type double, et renvoit l'ensemble des items du tableau dont le prix est compris entre ces deux valeurs sous forme d'une List.

Par exemple :

```
List<Item> liste1 = extraction_dans_une_fourchette_de_prix( DataJour1.getItemTab100000(), 5, 2
```

Renvoit une liste contenant tous les items du tableau **DataJour1.getItemTab100000()** dont le prix est compris entre 5 et 20 (bornes incluses)

1.6 Exercice 5 : détecter si il existe au moins deux éléments ayant des prix identiques

Ecrire une fonction **possede_deux_prix_identiques** qui prend en argument un tableau d'Item, et qui renvoit True si celui-ci contient au moins deux Item avec des prix identiques, et False sinon.

1.7 Exercice 6 : trouver l'ensemble des éléments ayant un prix identiques sur la liste

Ecrire une fonction **trouvez_items_avec_prix_identiques** qui prend en argument un tableau d'Item, et renvoie une List qui contient tous les éléments qui possèdent au moins un autre élément de même prix.
L'ordre de ces éléments n'a pas d'importance.

Les éléments ne doivent être présents qu'une seule fois.
Vous pouvez tester sur l'ensemble de 100 éléments, qui contient un total de 24 éléments de prix identiques.

2 Partie II : rendre le code plus efficace

Avez-vous une idée comment on pourrait accélérez toutes ces manipulations sur les prix ??

2.1 Exercice 7 : codage d'une recherche dichotomique

Implémentez un algorithme de recherche dichotomique dans la fonction **recherche_dichotomique**, qui prend en argument un tableau d'item trié par id et un entier correspond à l'id d'un Item, et renvoit l'élément correspondant.
La seule boucle que vous êtes autorisé à utiliser est une boucle for avec 30 itérations :

```
for( int i = 0; i < 30 ; ++i )
```

2.2 Exercice 8 à 11 : réécrire les exercices 3 à 6 de façon efficace sur une liste triée par prix

Les fonctions prendront les mêmes entrées et les mêmes sorties, seul leur nom changera, vous y ajouterez un suffixe "_trie".
Les noms des fonctions seront donc :

- Exercice 8 => prix_min_et_max_liste_trie, même consigne que l'exercice 3, sauf que vous supposez que le tableau est trie

- Exercice_9 => extraction_dans_une_fourchette_de_prix_trie
- Exercice_10 => possede_deux_prix_identiques_trie
- Exercice_11 => trouvez_items_avec_prix_identiques_trie

3 Exercice 12 et + : implémentation de tris

Pour ceux qui ont fini le reste, on peut s'intéresser aux algorithmes de tri. Il s'agit d'un sujet important en informatique, mais très largement couvert, et les implémentations existentes sont extrêmement performantes.
Le tri est également un des problèmes pour lesquels il existe le plus d'algorithmes différents.

Il n'y a pas de tests unitaires, mais vous pouvez comparer les résultats avec la fonction de tri implémenté dans le code.

3.1 Exercice 12 : tri par permutation

Il s'agit d'un tri "en place", à savoir donc que l'on modifie l'ordre des éléments dans le tableau donné en entrée, mais on ne renvoit pas de nouveau tableau.
Il consiste à parcourir le tableau complètement plusieurs fois, et à intervertir deux éléments qui ne sont pas dans le même ordre.
On s'arrête lorsque l'on a fait un parcours complet du tableau sans trouver deux éléments consécutifs qui ne soit pas bien ordonnés.

3.2 Exercice 13 : tri par sélection

Dans ce cas on cherche le plus grand élément du tableau, et l'on le permute avec le dernier élément, ensuite on recommence en ignorant le dernier élément, etc …