Travaux pratiques scala 2 : programmation fonctionnelle

Jonathan Lejeune

Objectifs

Ce sujet de travaux pratiques vous permettra de vous exercer sur la syntaxe scala et à la programmation fonctionnelle.

Exercice 1 – C'est les soldes ! !

Le package de travail de cet exercice est `datacloud.scala.tpfonctionnel.catalogue` **.**. Cet exercice utilise les types `Catalogue` et `CatalogueWithNonMutable` programmés dans le tp précédent. Il sera éventuellement necessaire d'étendre la visibilité de l'attribut exclusivement aux instance des classes filles afin que ces dernières puissent y accéder directement.

Question 1

Écrire un trait scala `CatalogueSolde` qui étend le trait `Catalogue` et qui offre la méthode `solde`. Cette méthode prend en paramètre un entier compris entre 0 et 100 et représente le pourcentage de rabais à appliquer sur l'ensemble des produits du catalogue.

Question 2

Écrire une classe `CatalogueSoldeWithFor` qui étend la classe `CatalogueWithNonMutable` et qui implante le trait `CatalogueSolde`. La méthode `solde` sera codée à l'aide d'une boucle for classique.

Question 3

Tester votre classe avec la classe Junit suivante fournie dans les ressources de TP :

`datacloud.scala.tpfonctionnel.catalogue.test.CatSoldeForTest`

Question 4

Écrire une classe `CatalogueSoldeWithNamedFunction` qui étend la classe `CatalogueWithNonMutabl` et qui implante le trait `CatalogueSolde`. La méthode `solde` sera codée avec l'appel à la méthode `mapValues` de la classe `Map` avec un paramétrage d'une fonction nommée. Cette fonction nommée sera la suivante :

```
def diminution(a:Double, percent:Int):Double = a * ((100.0 - percent) / 100.0
```

Le prototype de la méthode `mapValues` est le suivant :

```
def mapValues[C](f: (B) => C): Map[A, C]
```

Elle produit une nouvelle map à partir de la map appelante en appliquant sur chaque valeur la fonction `f`.

Question 5

Tester votre classe avec la classe Junit suivante fournie dans les ressources de TP :

`datacloud.scala.tpfonctionnel.catalogue.test.CatSoldeNamedFTest`

Question 6

Écrire une classe `CatalogueSoldeWithAnoFunction` qui étend la classe `CatalogueWithNonMutable` et qui implante le trait `CatalogueSolde`. La méthode `solde` sera codée avec l'appel à la méthode `mapValues` de la classe `Map` avec un paramétrage d'une fonction anonyme.

Question 7

Tester votre classe avec la classe Junit suivante fournie dans les ressources de TP :

`datacloud.scala.tpfonctionnel.catalogue.test.CatSoldeAnoFTest`

Dans la suite du TP, le package de travail de l'ensemble des exercices est `datacloud.scala.tpfonctionnel`.

Exercice 2 – Nombres premiers

Le crible d'Ératosthène est une méthode qui permet de trouver tous les nombres premiers inférieurs à un certain entier naturel N donné. Pour rappel un nombre premier est un nombre entier qui n'est divisible que par 1 ou par lui-même. Exemples : 2 3 5 7 11 .... Le principe de l'algorithme est de procéder par élimination. Il s'agit de supprimer d'une liste triée des entiers de 2 à N tous les éléments qui sont multiples d'un entier inférieur. Ainsi il ne restera que les entiers qui ne sont multiples d'aucun entier, et qui sont donc les nombres premiers.

Question 1

Créer un `object` scala `Premiers` et y coder la fonction `premiers` qui pour un entier `n` donné renvoie la liste des nombres premiers inférieurs ou égaux à `n`. L'idée de cette fonction est d'appliquer le crible d'Eratosthène en initialisant une liste d'entiers de 2 à `n` (utilisation de `range`) et d'appliquer successivement des filtres (méthode `filter`). Au filtre i il faudra enlever de la liste les nombres qui sont divisible par i.

Question 2

Écrire une autre version de la fonction appelé`premierWithRec` en utilisant une fonction imbriquée récursive dont l'algorithme est le suivant :

```
FONCTION f( liste entiers l )
 SI 1er element au carre > dernier element
 ALORS resultat l
 SINON
 resultat = concatenation du 1er element avec
  f( l sans le 1er element et tous les elements non multiples du 1er element )
 FIN SI
FIN FONCTION
```

Question 3

Tester avec la classe Junit suivante fournie dans les ressources de TP :

`datacloud.scala.tpfonctionnel.test.PremiersTest`

Exercice 3 – Letter Count

Question 1

Créer un `object` scala `Counters` et y coder la fonction `nbLetters` qui pour une liste de chaîne de caractères donnée calcule le nombre de lettres (caractère espace non compris) qu'elle contient. **Il vous est interdit d'utiliser une boucle.** Pour ceci il faut :

1. transformer la liste en liste de mots grâce à la méthode `flatMap`
2. transformer la liste de mots en remplaçant chaque mot par son nombre de lettres grâce à la méthode `map`
3. faire la somme de chaque élément grâce à la méthode `reduce`

Question 2

Tester avec la classe Junit suivante fournie dans les ressources de TP :

`datacloud.scala.tpfonctionnel.test.CountersTest`

Exercice 4 – Moyenne

Question 1

Créer un `object` scala `Statistics` et y coder la fonction `average` qui pour une liste de notes pondérées donnée calcule la moyenne de cette liste. Un élément d'une telle liste est un tuple de deux valeurs où le premier élément représente la note et le deuxième son coefficient respectif. Nous souhaitons utiliser les fonctions `map` et `reduce` qui calculeront un tuple de deux valeurs :

- la première valeur sera égale à la somme pondérée des notes
- la deuxième valeur sera égale à la somme des coefficients.

Le résultat de la fonction sera la division entre ces deux valeurs. **Il vous est interdit d'utiliser une boucle.**

Question 2

Tester avec la classe Junit suivante fournie dans les ressources de TP :

`datacloud.scala.tpfonctionnel.test.StatisticsTest`

Exercice 5 – Fonction d'ordre supérieur

Question 1

Créer un `object` scala `MySorting` et y coder les fonctions suivantes :

- `isSorted` qui permet de tester si un tableau est trié ou pas. Elle possède une variable de type `A` et prend comme paramètre :
  - ◇ un tableau d'élément de type `A`
  - ◇ une fonction qui prend deux `A` en argument et qui renvoi un booléen. Elle définit un ordre entre des éléments de type `A` en renvoyant vrai si le premier argument est inférieur ou égal au deuxième argument.

- `ascending` qui définit une fonction d'ordre ascendant sur un type générique donné (noté `T`). La fonction renvoyée prendra deux éléments a et b de type `T` et renverra un vrai si $a \leq b$, faux sinon.
- `descending` qui définit une fonction d'ordre ascendant sur un type générique donné (noté `T`). La fonction renvoyée prendra deux éléments a et b de type `T` et renverra un vrai si $b \leq a$, faux sinon.

Pour les méthodes `ascending` et `descending` l'existence d'une politique de comparaison sur des éléments de type `T` est nécessaire. En java nous utiliserions un `Comparator<T>`. En scala, il existe le trait `scala.math.Ordering[T]` qui étend et enrichi l'interface `Comparator<T>` de java (la méthode `compare` classique existe donc). Il est donc nécessaire que les méthodes `ascending` et `descending` prennent en argument un `Ordering[T]`. En scala, il est très courant de passer ce type d'argument implicitement ce qui évite à l'appelant de fournir à chaque appel une référence vers l'instance de type `Ordering[T]`. On notera que l'objet compagnon `Ordering` contient déjà des instances d'`Ordering` pour les types les plus courant (`IntOrdering`, `LongOrdering`, `StringOrdering`, `DoubleOrdering`, ...).

Question 2

Tester avec la classe Junit suivante fournie dans les ressources de TP :

`datacloud.scala.tpfonctionnel.test.MySortingTest`

Exercice 6 – Des fonctions en folies !

Question 1

Créer un `object` scala `FunctionParty` et y coder les fonctions suivantes : Donnez le code des fonction suivantes :

- `curryfie[A,B,C](f: (A, B)=>C): A =>B =>C` qui renvoie une fonction qui est la version curryfiée de la fonction passée en paramètre
- `decurryfie[A,B,C](f: A =>B =>C):(A, B)=>C` qui renvoie une fonction qui est la version non curryfiée de la fonction passée en paramètre
- `compose[A,B,C](f: B =>C, g: A =>B): A =>C` qui renvoie une fonction qui fait la composition des deux fonctions passées en paramètre
- `axplusb(a:Int,b:Int):Int=>Int` qui renvoie une fonction f tel que $f(x) = ax+b$. Cette fonction se basera exclusivement sur la fonction `curryfie` et `compose`.

Question 2

Tester avec la classe Junit suivante fournie dans les ressources de TP :

`datacloud.scala.tpfonctionnel.test.FunctionPartyTest`