TD 2 : Premiers programmes en SCALA

Université Paris Diderot – Master 2 SRI/LC/LP

25 octobre 2011

1 Environnement de développement pour Scala

1.1 Installation de Scala

Sur `lucien` La dernière version du compilation SCALA est installée sur `lucien` ici :

`/usr/local/src/scala-2.8.0.final`

Pour utiliser le compilateur en ligne de commande, vous devez mettre à jour votre variable d'environnement $PATH :

```
~% export PATH=$PATH:/usr/local/src/scala-2.8.0.final/bin
```

Après avoir relancé votre shell, vous devriez alors obtenir :

```
~% scala
Welcome to Scala version 2.8.0.final (Java HotSpot(TM) 64-Bit Server VM, Java 1.6.0_20).
Type in expressions to have them evaluated.
Type help for more information.
scala>
```

qui est la boucle interactive de SCALA.

Sur votre machine personnelle En fonction de votre environnement de travail et votre système d'exploitation plusieurs solutions s'offrent à vous pour installer SCALA. Plus d'informations ici :

`http://www.scala-lang.org/downloads`

Si vous rencontrez des difficultés, posez des questions sur la mailing-list du cours.

La distribution standard de Scala Voici les commandes principales de la distribution de SCALA :

Commande	Description
`scala`	la boucle interactive.
`scalac`	le compilateur.
`fsc`	le serveur de compilation.
`scaladoc`	le générateur de documentation de code.
`scalap`	le créateur de paquet.
`sbaz`	le gestionnaire de paquet.

1.2 Emacs

Pour les utilisateurs d'EMACS, il suffit de rajouter les lignes suivantes dans votre fichier `.emacs` pour activer, entre autres choses, la coloration syntaxique :

```
(add-to-list 'load-path "/usr/local/src/scala-2.8.0.final/misc/scala-tool-support/emacs")
(require 'scala-mode-auto)
(add-hook 'scala-mode-hook '(lambda () (yas/minor-mode-on)))
```

Pour utiliser efficacement ce mode EMACS, consultez :

`http://www.scala-lang.org/node/354`

1.3 Eclipse

Il existe un *plugin* SCALA pour ECLIPSE. Celui-ci est installé sur `lucien`.

`http://www.scala-ide.org/`

La procédure d'installation est décrite en détails sur ce site :

`https://www.assembla.com/wiki/show/scala-ide/Requirements_and_Installation`

1.4 Outils essentiels

Comme pour l'apprentissage de tout langage de programmation, il faut consulter régulièrement la documentation de la bibliothèque standard pour se l'approprier progressivement :

`http://www.scala-lang.org/api/current/index.html`

Par ailleurs, le langage SCALA est spécifié très précisément dans le document suivant :

`http://www.scala-lang.org/sites/default/files/linuxsoft_archives/docu/files/ScalaReference.pdf`

Pour finir, le site des créateurs du langage fourmille de renseignements sur SCALA :

`http://www.scala-lang.org`

2 Bonjour le monde !

Exercice 1 *Voici un classique « Hello World ! » écrit en* SCALA *:*

```
01 object HelloWorld extends Application {
02   override def main (args :Array[String]) {
03     println ("Hello World !")
04   }
05 }
```

1. Compilez et exécutez ce programme.

2. Sachant que `args.length` renvoie la taille du table `args`, modifiez ce programme pour qu'il affiche `Hello World!` si aucun argument n'est passé en ligne de commande et `Hello X_1 X_2 ... X_n!` si `X_1`, ..., `X_n` sont les arguments passés en ligne de commande. Écrivez plusieurs versions de ce programme, en utilisant les boucles while, les expressions conditionnelles, les notations « `for x ← ...` ». Quelle est la solution la plus concise ? □

3 Tours de Hanoï

Exercice 2 *On souhaite implémenter un plateau de jeu pour le jeu de réflexion "Tours de Hanoï". L'utilisateur du programme pourra indiquer le nombre de tours et disques dont il souhaite disposer, et jouer jusqu'à fin de partie, détecté automatiquement le programme.*

Au début normal d'une partie, toutes les tours sont vides sauf une, appelée tour de départ, sur laquelle sont empilées les disques, toutes de diamètre différents. Le but est de déplacer tous les disques, un à un de tour en tour, jusqu'à ce qu'elles soient toutes sur la tour désignée comme tour d'arrivée. Cependant, un disque d'un certain diamètre ne doit jamais se retrouver sur un autre de diamètre inférieur à aucune étape de la partie.

1. Consultez l'API de la bibliothèque standard de Scala *pour prendre connaissance des méthodes proposées par les types* `Array`*,* `IndexedSeq`*, et* `List`*.*

2. Implémentez le plateau de jeu en utilisant un objet de type `Array` *dont les indices représenteront les disques, et les valeurs les tours lesquelles se trouvent les disques.*

3. Implémentez maintenant le plateau de jeu en utilisant un objet de type `IndexedSeq` *avec la même sémantique d'indices et de valeurs. N'oubliez pas d'utiliser méthodes maintenant disponibles.*

4. Implémentez maintenant le plateau de jeu en utilisant un objet de type `Vector` *dont les indices représentent les tours, et dont les valeurs représentent l'ensemble des disques posés sur les tours. Utilisez des objets de type* `List` *pour représenter ces valeurs.*

5. Restructurez l'ensemble de vos implémentations à l'aide d'une classe abstraite ne spécifiant aucune implémentation, et d'une sous-classe par implémentation.

□

4 Table de hachage

Exercice 3 *On souhaite implémenter une structure de données qui associe des images de type* `V` *à des clés de type* `K` *en s'appuyant sur une fonction de hachage définie sur les clés de type* `K`*. On vous fournit le squelette de code suivant :*

```
01 abstract class HashTable {
02   type K
03   type V
04
05   def hash (k :K) :Int
06   def add (k:K, v:V)
07   def find (k :K) :Option[V]
08 }
09
10 abstract class SimpleHashTable extends HashTable {
11   type Bucket = List[Pair[K, V]]
12   type Table = Array[Bucket]
13
14   def alloc_data (size :Int) :Table = { /*? */ }
15   def initial_size :Int
16   var data = alloc_data (initial_size)
17   var count = 0
18   def add_in (data :Table, k :K, v :V) { /*? */}
19   def rehash () { /*? */ }
20   def add (k :K, v :V) { /*? */ }
21   def find (k :K) :Option[V] = { /*? */ }
22 }
23
24 object Test extends Application {
25   override def main (args :Array[String]) = {
26     val t = new SimpleHashTable {
27       type K = String;
28       type V = String;
29       def hash (k :K) = k.hashCode ()
30       def initial_size = 31
31     }
32     t.add ("Luke", "Darth");
33     t.add ("Leila", "Darth");
34     t.add ("George W", "George");
35     println (t.find ("Abama"));
36     println (t.find ("Leila"));
37   }
38 }
```

1. Consultez l'API de la bibliothèque standard de SCALA *pour prendre connaissance des méthodes proposées par les types* `List`, `Pair`, `Option` *et* `Array`.

2. Donnez une implantation stupide de chacune de ces méthodes pour que ce programme compile.

3. Implantez la fonction `alloc_data` *qui créé un objet de type* `Table` *dont tous les éléments ont été initialisés à* `Nil`.

4. Implantez la fonction `add_in (table : Table, k : K, v : V)` *qui calcule la valeur de hachage de* `k` *et insère le couple* `(k, v)` *dans la liste des couples de l'élément* `k % table.length` *de* `table`.

5. Implantez la fonction `rehash ()` *qui alloue une nouvelle table* `new_data` *de taille* `2 * data.length + 1` *et y réinsère tous les éléments de* `data` *avant de remplacer* `data` *par* `new_data`

6. Implantez la fonction `add (k : K, v : V)` qui insère un nouveau couple `(k, v)` dans la table et vérifie ensuite que le nombre d'éléments de la table est inférieur à `data.length / 2`. Dans le cas contraire, on utilise la méthode `rehash` pour s'assurer que cet invariant est maintenu.

7. Implantez la fonction `find (k : K)` qui renvoit `None` si aucune image n'est associée à `k` et `Some (v)` si `(k, v)` est la dernière association de `k` insérée dans la table.

8. Bonus. Implémentez une nouvelle classe de table de hachage qui la résolution de collision dite du coucou comme décrite ici :

`http://en.wikipedia.org/wiki/Cuckoo_hashing`

□

5 Modélisation objet d'une bibliothèque

Exercice 4 *On modélise une application devant servir à l'inventaire d'une bibliothèque. Elle devra traiter des documents de nature diverse : des livres, des dictionnaires, et autres types de documents qu'on ne connaît pas encore précisément mais qu'il faudra certainement ajouter un jour (articles, bandes dessinées...). Tous les documents possèdent un numéro d'enregistrement et un titre. À chaque livre est associé, un auteur et un nombre de pages, les dictionnaires ont, eux, pour attributs supplémentaires une langue et un nombre de tomes. On veut manipuler tous les articles de la bibliothèque au travers de la même représentation : celle d'un document.*

1. Après avoir dessiné le diagramme de classe UML de cette modélisation objet, définissez en SCALA *les classes `Document`, `Book` et `Dictionary`. (Essayez de rester le plus abstrait possible.)*

2. Définissez une classe `Library` réduite à une méthode main permettant de tester les classes précédentes (ainsi que les suivantes).

3. On souhaite autoriser des extensions futures de fonctionnalité des objets de la hiérarchie des documents sans que cela ne nécesse de modification dans le code des classes existantes. La nouvelle fonctionnalité est une méthode `keywords () : Set[String]` qui produit un ensemble de mot-clés à associer au document. Pour autoriser ces extensions futures, on modifie (pour la dernière fois !) les classes `Document`, `Book` et `Dictionary` en rajoutant une méthode « `def accept[Result] (v : Visitor[Result]) : Result` » où la classe `Visitor` est définie comme suit :

```
01 abstract class Visitor[Result] {
02   def visitBook (b :Book) :Result
03   def visitDictionary (d :Dictionary) :Result
04 }
```

Écrivez le corps des méthodes `accept` des classes `Dictionary` et `Book` puis implémentez un objet `Visitor` particulier qui récolte les mots-clés des documents présents dans leur titre et leur nom d'auteur (si il existe).

4. Implémentez la fonctionnalité précédente à l'aide d'une analyse de motif (pattern matching). Comparez ces deux approches. En particulier, comment faire évoluer ces implémentations lorsque l'on rajoute un nouveau type de document ?

□