1

Les collections

Main Author(s): to be fixed: B. Pottier, Université de Brest, Bernard.Pottier@univ-brest.fr, catherine dezan

1.1 Introduction :organisation hiérarchique des collections

La figure suivante présente une partie de l'organisation hiérarchique des collections.

```
Object ()
  Collection ()
      Bag ('contents')
      SequenceableCollection ()
          ArrayedCollection ()
             Array ()
             CharacterArray ()
                  String ()
                    Symbol ()
                  Text ('string' 'runs')
             IntegerArray ()
                 ByteArray ()
                 WordArray ()
          Interval ('start' 'stop' 'step')
          OrderedCollection ('firstIndex' 'lastIndex')
             SortedCollection ('sortBlock')
      Set ('tally')
          Dictionary ()
```

1.2 TP

1.2.1 La classe Collection

On trouve dans cette classe, les messages (les énumérateurs) compréhensibles par toutes les sous-classes.
On retiendra les messages suivants :

- les énumérateurs correspondant aux messages : collect:, do:, detect:, reject:, select, inject: into:.
- les messages de conversion : asArray, asBag, asSet, asSortedCollection.
- le message size donne la taille de la collection

1.2.2 La classe SequenceableCollection

Un ordre est défini entre les élements de la collection. Les objets de cette classe comprennent les messages first, last.

- La classe Array est une collection d'éléments de taille fixe. La création d'un object de cette classe peut se faire grâce au message new:.

  Ex : Array new: 12 crée un tableau de taille 12 dont les éléments sont initialisés à nil.

  Les accès aux éléments du tableau sont possibles avec la méthode at:. La modification d'un élément du tableau est faite par la méthode at: put:.

- La classe Interval ne possède pas d'ordre explicite mais implicite. La création d'un intervalle se fait avec les messages to: ou to: by: selon le pas de l'intervalle choisi.

  Ex: 1 to: 10 crée un intervalle avec tous les éléments de 1 à 10.
  1 to: 10 by: 2 crée un intervalle de 1 à 10 avec uniquement les éléments impairs.

- La classe OrderedCollection définit des collections de taille dynamique. La création est faite par le message new, puis l'extension par l'utilisation du message add: ou addAll:.

  Ex: OrderedCollection new add:1; add:2 définit une collection avec les éléments 1 et 2.

  Le message addAll: permet de définir une collection à partir de celle passée en paramètre.

  Ex: OrderedCollection new addAll: #(#(1 4) 5 6) crée une instance de la classe OrderedCollection contenant les éléments #(1 4), 5 et 6. La suppression d'un élément dans un tableau se fait par la méthode remove:.

- La classe SortedCollection est une sous-classe de la classe OrderedCollection. Les éléments de cette classe peuvent être triés selon un certain critère définit dans un bloc en utilisant la méthode sortBlock:.
  Ex :

```
| aCol |
aCol:= SortedCollection new.
aCol add:2; add:8; add:1.
        ''la collection obtenue est triee dans l'ordre croissant''
aCol sortBlock: [ :elem1 :elem2 | elem1 ¿ elem2].
        ''on trie la collection dans l'ordre decroissant''
```

Exercices

Exercise: 1. Créer un tableau de taille 5, trier ses élements dans l'ordre décroissant et donner le tableau résultat.

Exercise: 2. soit matrice := #(#(1 0 0) #(1 2 0) #(1 2 3)), faire la somme des éléments se trouvant sur la diagonale de cette matrice.

Exercise: 3. Définir par un bloc avec un paramètre (représentant la taille des tableaux)les tableaux de la forme suivantes :

```
#(#(1))                    pour taille=1
#(#(1) #(1 2))             pour taille=2
#(#(1) #(1 2) #(1 2 3))    pour taille=3
#(#(1) #(1 2) #(1 2 3) #(1 2 3 4))    pour taille=4
....
```

1.2.3 La classe Set

Un ensemble est une collection dont les éléments n'ont pas d'ordre, pas de clés d'accès. Les doublons n'y sont donc pas autorisés.

Les méthodes suivantes sont disponibles sur un ensemble :

- add: unElement méthode d'ajout de l'élément unElement (si il n'appartient pas déja à l'ensemble)

- remove: unElement ifAbsent: unBloc supprime l'élément unElement s'il est présent et évalue le bloc unBloc si l'élément est absent.
- includes: unElement teste si l'élément unElement appartient à l'ensemble (renvoie `true` si il s'y trouve, `false` sinon).
- occurrencesOf: unElement donne le nombre d'occurrences de l'élement unElement (0 ou 1)

Exemples.

```
| aSet |
"Construction d'un ensemble vide"
aSet := Set new.                "Set ()."
"Ajouts"
aSet add: 1; add: 1@3.
aSet.                           "Set (1 1@3)"

"Suppression"
aSet remove:3 ifAbsent:[aSet add: 4/5].
aSet.                           "Set (1@3 (4/5) 1)"

"Appartenance"
aSet includes:1@3.              "true"

"Transformation d'un Array en Set"
#(2 7 t 7 9) asSet              "Set (2 #t 7 9)"
#(3 6 6) asSet occurrencesOf: 6 "1"
```

Exercices

Exercise: 4. A partir d'un premier tableau, définir un nouveau tableau où les doublons du tableau initial sont supprimés. On peut utiliser la méthode asArray pour transformer un receveur en tableau.

Exercise: 5. Construire un ensemble contenant tous les nombres de 1 à 100.

Exercise: 6. Transformer l'ensemble précédent en remplaçant chaque nombre par son reste de la division entière par 5 (modulo 5). Quelle est la nature et la taille de cette nouvelle collection? Refaire la même opération directement sur un intervalle de (1..100).

1.2.4 La classe Dictionary

Les dictionnaires sont des ensembles dont les éléments sont des instances de la classe *Association*. Une association est un couple d'objets, le premier élément étant une clé, le deuxième représentant une valeur.

Exemple : 2->3 est une association dont la clef est 2 et la valeur 3.

Quelques méthodes disponibles sur les dictionnaires :

- at:k renvoie la valeur référencée par la clé k
- at:k put:o installe l'objet o comme valeur à la clé k
- add:$uneAssociation$ ajoute au dictionnaire, l'argument $uneAssociation$ (qui est une association)
- associations renvoie une collection ordonnée des associations du dictionnaire
- keys renvoie un set contenant toutes les clés

- values renvoie une collection ordonnée comportant toutes les valeurs
- includesKey:key teste si la clé key appartient au dictionnaire
- associationAt:key renvoie l'association dont la clef est key
- keyAtValue:$uneValeur$ renvoie une clef associée à la valeur $uneValeur$ passée en paramètre
- keysAndValuesDo:$unBloc$ exécute le bloc $unBloc$ pour chaque association clé-valeur du dictionnaire.

Exercices

Exercise: 7. Créer un ensemble avec les éléments appartenant à l'intervalle (1 .. 100) et dont le reste de la division entière par 5 est égal à 0. Construire l'association avec 0 comme clé et cet ensemble comme valeur.

Exercise: 8. Créer un dictionnaire où les clés sont les restes de la division par 5 des éléments appartenant à l'intervalle (1 .. 100) et les valeurs sont les sous-ensembles constitués d'élements de l'intervalle qui ont le même reste pour la division par 5 (reste représenté par la clé).

Exercise: 9. Transformer le dictionnaire précédent en un nouveau dictionnaire òu les sous-ensembles associés aux différentes clés ont été remplacés par la somme de leurs éléments.

1.2.5 La classe Bag

Une instance de la classe Bag peut être considéré comme un Set qui autorise les doublons. Un Bag ne duplique pas physiquement les doublons, mais maintient le nombre d'occurrences de chaque élément. Un Bag a un contenu qui est un dictionnaire associant chaque élément à son nombre d'occurrences.

Bag hérite des méthodes classiques de la classe Collection.

Quelques methodes spécifiques de Bag :

- occurrencesOf:$unElement$ donne le nombre d'occurrences de l'élement $unElement$
- add:$unElement$ withOccurrences:$nbOccurrences$ ajoute le nombre d'occurrences $nbOccurrences$ de l'objet $unElement$
- remove:$unElement$ ifAbsent:$unBloc$ supprime une occurrence l'élément $unElement$ s'il est présent et évalue le bloc $unBloc$ si l'élément est absent
- removeAllOccurrencesOf:$unElement$ ifAbsent:$unBloc$ supprime toutes les occurrences de l'élément $unElement$ s'il est présent et évalue le bloc $unBloc$ si l'élément est absent
- valuesAndCountsDo:$unBloc$ permet d'itérer le bloc $unBloc$ sur des couples (objet nb_occ_de_objet) en supposant que le bloc $unBloc$ a deux arguments (analogie avec la méthode keysAndValuesDo: des dictionnaires).

Exercices

Exercise: 10. Récupérer dans un Bag les restes de la division par 2 des nombres compris entre 1 et 100 000. Inspecter l'objet ainsi obtenu.

Exercise: 11. Faire la somme des éléments se trouvant dans le Bag. Faire la même opération sur le tableau correspondant à ce bag. Quel est le code qui offre le temps d'exécution le plus intéressant? Pour connaitre le temps d'exécution d'un code, utilisez l'expression Time millisecondsToRun: suivi d'un paramètre étant un bloc sans variable comportant le code à évaluer. Par exemple Time millisecondsToRun: [(1 to: 1000) asBag].

Exercise: 12. Généraliser le test précédent sur une collection de tableaux ayant des valeurs constantes et sur une collection de bag comportant les mêmes éléments afin de mieux comparer les temps d'exécution pour l'opération de la somme.

1.3 Exercices (TD)

1.3.1 La classe SequenceableCollection

Exercise: 13. Créer en utilisant un bloc avec un paramètre (le paramètre caractérisant la taille des matrices générées) les matrices carrés suivantes :

```
#(#(1))            pour valeur=1
#(#(1 2) #(1 2))          pour valeur=2
#(#(1 2 3) #(1 2 3) #(1 2 3) ) pour valeur=3
```

Exercise: 14. Définir à partir d'une matrice (tableau de lignes), un nouveau tableau où les éléments sont les sommes des lignes du tableau initial.

Exercise: 15. Soit le tableau #('toto' 'lulu' 'guillaume' 'luc' 'laurent'), définir un nouveau tableau où les noms sont triés par ordre décroissant de taille.

1.3.2 La classe Set

Exercise: 16. Convertir le tableau #(1 2 3 4 2 4) en set. Quelle est la taille du résultat?

Exercise: 17. Comment afficher tous les éléments d'un set dans le Transcript?

Exercise: 18. Créer un bloc à deux paramètres — deux sets — qui renvoie un set correspondant à l'intersection des deux sets

Exercise: 19. Créer un bloc à deux paramètres — un set et un objet quelconque — Si l'objet appartient à l'ensemble il est supprimé de celui-ci. Si il n'y appartient pas, le message "l'objet monObjet n'appartient pas a l'ensemble" est affiché sur le Transcript.

1.3.3 La classe Bag

Exercise: 20. Créer un bag et y ajouter tous les éléments du tableau #(1 1 1 2 34)

Exercise: 21. Convertir le tableau #(1 1 1 2 34) en Bag

Exercise: 22. Ecrire un bloc avec une variable qui prend un bag et qui renvoie un bag. Les valeurs du second bag correspondent au double des valeurs du premier bag

Exercise: 23. Ecrire un bloc avec une variable qui prend un bag et qui renvoie un dictionnaire. Le dictionnaire contient les associations élément du bag ->nombre d'occurrences de l'élément.

Exercise: 24. Modifier le bloc précédent pour qu'il renvoie la somme de tous les éléments du bag paramètre

1.3.4 La classe Dictionary

Exercise: 25. Créer un nouveau dictionnaire. Quelle est sa taille? Pourquoi?

Exercise: 26. Y ajouter l'objet 15 à la clé 1

Exercise: 27. Y ajouter l'association 2 ->5

Exercise: 28. Que se passe-t-il si on ajoute l'association 2 ->6?

Exercise: 29. Inspecter les clés du dictionnaire

Exercise: 30. On décrit un parcours sous la forme d'une liste de trajets. Créer un dictionnaire dont les clés sont les points de départ et les valeurs les points d'arrivée à partir du tableau suivant:

Brest	Quimper
Quimper	Lorient
Nantes	Vannes
Rennes	Nantes

Exercise: 31. Créer un block qui prend en paramètre le dictionnaire et qui écrit sur le Transcript l'ensemble des trajets. exemple: 'De Brest je peux aller à Quimper'