1

Streams: Les accs

Main Author(s): Bernard Pottier, Université de Brest **Stef** ►*add Unit tests*◄

1.1 Définition

Stream et les sous-classes de Stream fournissent un mécanisme d'accès séquentiel sur des objets, internes ou externes.

Lorsqu'une instance de Stream est utilisée, elle a seule le contrôle de l'objet accédé, qui ne devrait pas être utilisé directement. La resynchronisation de l'objet lu ou écrit peut impliquer une opération de mise à jour (flush, ou commit pour assurer la mise à jour d'une entrée-sortie...). La raison est que le mécanisme séquentiel tamponne les données en lecture ou en écriture afin d'économiser les transferts.

1.2 Créations et accès

1.2.1 Créations

On crée un stream[1] en s'appuyant sur des classes telles que ReadStream, WriteStream, ReadWriteStream, lors que l'objet est interne. D'autres classes sont utilisées pour les accès séquentiels externes, pour les entrée-sorties. Certain objets savent rendre des Stream en réponse au message readStream ou writeStream; c'est le cas des fichiers Filename.

On obtient un stream en expédiant le message on: à la classe à instancier. On passe un objet vide lorsque l'on veut écrire, une collection séquençable pleine lorsque l'on veut lire. Le stream s'occupe de faire grossir votre objet lorsque le besoin s'en fait sentir, en respectant sa classe.

- ReadStream on: 'A Plouescat, la joie eclate!'
- (WriteStream on: (Array new: 3))

1.2.2 Extraction du contenu, tests

Le contenu du stream ne devrait pas être accédé directement. Il faut récupérer ce contenu en expédiant le message contents au stream. L'exemple qui suit montre l'insertion et l'extraction de l'objet:

```
| monStream |
monStream := ReadStream on: 'A Plouescat, la joie eclate!'.
monStream contents.   '' 'A Plouescat, la joie eclate!''
```

On peut noter qu'il est possible de savoir si on est au bout d'une lecture (fin de fichier, dernier élément d'un objet), à l'aide du message atEnd. On peut se positionner en fin de stream à l'aide de setToEnd. La position dans le stream est contrôlable à l'aide du message position.

[1] en français, un flôt

```
| monStream |
monStream := ReadStream on: 'A Plouescat, la joie eclate!'.
monStream atEnd. "(printIt) false"
monStream setToEnd ; position. "(printIt) 28"
monStream  position: 12; position "(printIt) 12"
```

1.2.3 Accès en lecture

Lecture séquentielle d'un élément

La lecture est opérée en expédiant le message next. Le stream fait progresser son index interne et effectue d'autres opérations si nécessaire.

```
| array stream |
array := #( 2  3 6 7 9 0 ).
stream := ReadStream on: array.
stream next. stream next. " 3"
```

lecture d'une séquence

On peut prélever un nombre arbitraire d'éléments en utilisant le message nextAvailable: unEntier. Bien entendu, on ne peut dépasser la capacité de l'objet accédé, si celle-ci est bornée.

```
| array stream |
array := #( 2  3 6 7 9 0 ).
stream := ReadStream on: array.
stream nextAvailable: 3. " (printIt) #(2 3 6)"
stream nextAvailable:12." (printIt) #(7 9 0)"
```

lecture d'une séquence sur condition

Il est souvent intéressant de prélever des éléments jusqu'à ce qu'une condition soit obtenue.

- upToEnd: épuise le contenu de l'objet.

```
| array stream |
array := #( 2  3 6 7 9 0 ).
stream := ReadStream on: array.
stream nextAvailable: 3. " (printIt) #(2 3 6)"
stream upToEnd." (printIt) #(7 9 0)"
```

- through: limit: lit jusque la prochaine occurence de l'objet limit.

```
| stream |
stream := ReadStream on: 'C''est a Morlaix que l''on se plait'.
stream through:  $'.
stream through:  $'. " (printIt)'est a Morlaix que l''"
```

1.2.4 Ecriture

En écriture, on envoie nextPut: unElement, qui insère l'éleément au bout de l'objet. On peut provoquer l'insertion d'une collection d'éléments avec nextPutAll: uneCollection

```
| stream |
stream := WriteStream on: (Array new: 4).
stream nextPut: 3/2.
stream nextPutAll: #( 2  3 6 7 9 0 ).
stream contents " #((3/2) 2 3 6 7 9 0)"
```

1.3 Application à l'affichage des messages

La fenêtre Transcript sert à l'affichage des messages. On peut obtenir le texte contenu dans cette fenêtre via le message value:
Transcript value asString

On écrit dans le Transcript comme dans un stream de caractères:
Transcript nextPut: Character cr.
Transcript nextPutAll: 'A Plouarzel, on fait du zele'.
Transcript flush.

Noter que le message flush est indispensable pour provoquer l'affichage immédiat des caractères insérés. Les Stream savent gérer l'insertion de caractères non imprimables plus simplement:
Transcript cr; tab; nextPutAll: 'A Plouarzel, on fait du zele';
space;cr; flush

1.4 Cas des fichiers

Un fichier est repéré par un *nom*, instance de la classe Filename. On peut choisir un nom de fichier (String) via un dialogue spécialisé:

```
| nom |
nom :=  Dialog requestFileName: 'nom du fichier?'
```

Une fois ce nom choisi, on peut lui associer un fichier en convertissant la chaine en une instance de Filename:

```
| nom  fileName |
nom :=  Dialog requestFileName: 'nom du fichier?'.
fileName := nom asFilename.
```

On obtient ensuite aisément un stream en lecture ou écriture en expédiant les messages writeStream:

```
| stream |
stream := 'Transcript.file' asFilename writeStream.
stream nextPutAll: Transcript value.
stream commit ; close.
```

ou readStream, en lecture:

```
| stream nomDuFichier |
nomDuFichier := Dialog requestFileName: 'nom du fichier?'.
stream  := nomDuFichier asFilename readStream.
Transcript nextPutAll:    stream contents.
Transcript flush. stream close.
```

1.5 Exercices

1.5.1 Analyse lexicale

En phrases

On peut obtenir le texte de la fenêtre Transcript en expédiant le message value*. Construire une collection des phrases contenues dans ce texte:*

```
| rStream phrases |
rStream := ReadStream on:  Transcript value asString. "lecture sur la chaine"
phrases := WriteStream on: (Array new: 100). "ecriture sur  des tableaux"
[ rStream atEnd ] whileFalse:
    [ phrases nextPut: ( rStream upTo: $. )].
phrases contents
```

Renverser les mots

On utilise le texte de la fenêtre Transcript. Produire un texte où les mots sont écrits à l'envers...

```
 | rStream texte ligneStream mot |
rStream := ReadStream on:  Transcript value asString. "lecture sur la chaine"
texte := WriteStream on: (String new: 1000). "ecriture du nouveau texte."
[ rStream atEnd ] whileFalse:
    [ ligneStream := ReadStream on: ( rStream upTo: Character cr).
          [ ligneStream atEnd ] whileFalse:
            [mot := ligneStream upTo: Character space .
             texte nextPutAll: mot reverse ; space  ].
             texte cr.
        ].
texte contents
```

1.5.2 Analyse de code

D'eutres classes ont des comportements de Stream. C'est par exemple le cas de l'analyseur lexical Smalltalk, qui peut être réutilisé à d'autres tâches:

```
| texteScanner item items |
item := ''.
items := OrderedCollection new.
texteScanner :=Scanner new on:  'begin Carthago delenda est.  7 +9 = 15 . end' readStream.
[ item = 'end' ]
  whileFalse: [ item :=texteScanner scanToken. items add: item].
items
```