

Smalltalk

MCours.com

Environnement de programmation

- Modèle orienté objet “pur” : tout est objet
- Langage
- Modèle objet supportant héritage, classes, instances, liaison dynamique, envoi de message, ramasse-miettes
- Un ensemble de classes réutilisable, implémentant
 - ◆ fonctions de base
 - ◆ support à la portabilité inter-plateformes : interfaces graphiques utilisateur, aide à la définition et gestion de classes
- Un ensemble d’outils de développement :
 - ◆ examiner, modifier, renommer, ajouter, supprimer des classes
 - ◆ outils de débogage
- Langage interprété

La machine Smalltalk

- Composée de :
 - ◆ l'exécutable *squeak.exe*
 - ◆ l'image : *squeak.image*
 - ◆ le journal : *squeak.changes*
 - ◆ les sources : *squeakVI.sources*
 - ◆ le source Smalltalk de l'environnement Smalltalk : *squeak.sources*
- Etat de la machine virtuelle défini par l'image *.image
- Smalltalk disponible sur PC, Mac, stations Unix
- portage sur une autre plateforme nécessite uniquement de changer de *.exe
- *.image portable directement sauf si références système directes

The screenshot displays the Smalltalk System Browser interface. At the top, a table lists various system classes and their methods. Below this, a class definition for `ifNil: nilBlock` is shown. In the foreground, a `Transcript` window displays the text "bonjour". To the right, a `Workspace` window shows a menu of actions such as "find...", "undo", "copy", "paste", "print it", and "show bytecodes".

System Browser			
Interface-Projects	Boolean	accessing	basicType
Kernel-Objects	False	testing	haltIfNil
Kernel-Classes	Object	comparing	ifNil:
Kernel-Methods	ObjectTracer	copying	ifNil:ifNotNil:
Kernel-Processes	ObjectViewer	dependents access	ifNotNil:
System-Compiler	True	up-dating	ifNotNil:ifNil:
System-Support	UndefinedObject	printing	isColor
System-Sound		class membership	isInteger
System-Network			

```

ifNil: nilBlock
  "Return self, or evaluate the block if I'm == nil (q.v.)"
  + self
  
```

Transcript

```

bonjour
  
```

Workspace

```

Transcript show: 'bonjour'

find... (f)
find again (g)
set search string (h)
do again (i)
undo (z)
copy (c)
cut (x)
paste (v)
do it (d)
print it (p)
inspect it (i)
accept (s)
cancel (l)
show bytecodes
more...
  
```

Envoi de messages

- Les objets sont des instances de classe
- Les messages auxquels un objet peut répondre est défini par le protocole de sa classe
- La manière dont l'objet réagit aux messages est définie par les méthodes de la classe
- Syntaxe
 - ◆ Les noms de classe commencent par une majuscule : Integer
 - ◆ noms des messages commencent par une minuscule, puis tous caractères sauf l'espace
 - ◆ ex : unMessage, monAdresse
 - ◆ Messages paramétrés : des mots-clés suivis de ":" alternent avec les paramètres effectifs
 - ◆ ex : nom: 'Dupont' adresse: '12, rue Marie Curie - 10000 Troyes'

Catégories de messages

- Messages unaires
 - ◆ un nom de message
 - ◆ un opérande qui est l'objet récepteur
 - ◆ exemples :
 - ◆ x sin
 - ◆ Date tomorrow
 - ◆ 5 factorial
- Messages binaires
 - ◆ utilisés pour spécifier des opérations arithmétiques, logiques et de comparaison, à l'aide des caractères + / \ * ~ < > = @ % | & ? ! ,
 - ◆ exemples :
 - ◆ a + b
 - ◆ a >= b

Catégories de messages

■ Messages à mots-clés

- ◆ correspondent à l'appel de procédure avec au moins 2 paramètres
- ◆ Les mots-clés doivent être suivis de “:”
- ◆ exemples :
 - ◆ unObjet nom1: p1 nom2: p2
 - unObjet est l'objet destinaire du message
 - nom1: est la première partie du message
 - p1 est passé à nom1:
 - nom2: est la seconde partie du message
 - p2 est passé à nom2:
 - ◆ Array new: 20

Identification/création d'objets

- ◆ Chaque objet se voit associé un identificateur *unique* lors de sa création (OID)
- ◆ aucune sémantique associée à l'identificateur
- ◆ une variable désignant un objet contient l'identificateur de l'objet
- ◆ un message envoyé à une variable est envoyé à l'objet dont l'identificateur est contenu dans la variable
- ◆ exemple :
 - ◆ variable globale monCours

monCours nom: 'LO02'.

^monCours nom

Self

■ Self :

- ◆ utile pour écrire une méthode d'un objet ayant besoin d'appeler une autre de ses méthodes

■ Exemple

```
nom: aNom
    nom := aNom
adresse: anAdresse
    adresse := anAdresse
nom: aNom adresse: anAdresse
self nom: aNom.
    self adresse: anAdresse.
```

ou

```
nom: aNom adresse: anAdresse
nom := aNom.
adresse := anAdresse.
```

Ordre d'exécution des messages

■ Exécution de gauche à droite : l'évaluation d'un message remplace le message dans l'expression à calculer

■ du plus prioritaire au moins prioritaire

- ◆ expressions parenthésées
- ◆ expression la plus à gauche
- ◆ messages unaires, puis binaires, puis à mots-clés de gauche à droite
- ◆ dans une expression, les variables sont remplacées par les objets vers lesquels elles pointent
- ◆ expression de retour (^)

■ Exemple :

$$6 + 3 * 5 + \frac{3 * 4}{1} 2$$

$$- 3 \quad 4 \quad \underline{\quad} \quad 5$$

Structure de base des méthodes

- 1ère ligne : interface de la méthode
- puis dans un ordre quelconque :
 - ◆ variables temporaires
 - ◆ elles sont toujours locales à la méthode/bloc dans laquelle elles sont déclarées
 - ◆ commentaires
 - ◆ commence par “ et finit par “
 - ◆ expressions
 - ◆ valeur de retour
 - ◆ entraîne la sortie de la méthode
 - ◆ facultative : la valeur par défaut retournée est l’objet receveur du message

```
aMethode: anArg
“exemple de méthode”
| a b |
a := anArg - 7.
b := a * 3.
^b+1
```

Expressions et affectations

- Une expression est composée de l’un des éléments suivants
 - ◆ un nom de variable
 - ◆ un littéral
 - ◆ un message
- Affectation
 - ◆ variable := expression
 - ◆ exemples
 - ◆ x := 7.
 - ◆ k := ‘un caractère’.
 - ◆ uneVariable := j * k.

```
Dans la classe Bidon
-----
nom: aNom adresse: anAdresse
nom := aNom.
adresse := anAdresse.
```

Enchaînement de messages

```
Object subclass: #Bidon
  instanceVariableNames: 'nom
  adresse tel '
  classVariableNames: ''
  poolDictionaries: ''
  category: 'Kernel-Objects'
```

```
c nom: 'coucou' adresse: 'couc2' tel: '45'.
```

--> message d'erreur car la méthode nom: adresse: tel: est inconnue

```
| c|
```

```
c := Bidon new.
```

```
c nom: 'coucou' adresse: 'couc2' tel: '45'.
```

Solution :

```
c nom: 'coucou' ; adresse: 'couc2'; tel: '45'.
```

Opérations

■ Littéraux :

- ◆ instances de String, Number, Character, Symbol, Array ou de leurs sous-classes

- ◆ Number : integer, fraction, floating point

 - ◆ 123, 23, 2.4e7, 3/4

Character : tout caractère Ascii précédé de \$

String : 'toute séquence de caractères'

Symbol : identificateur, sélecteur binaire, sélecteur mot-clé précédé de #

#nom

#+

#at:put:

Array

- ◆ Structure de données dont les éléments peuvent être tout objet valide

- ◆ notation : #('un', '\$2', 'trois')

Opérations

- *nombre opération nombre*
 - ◆ Opération : + - * //(division entière) \\
(reste)
 - ◆ Ex :
 - ◆ 9 \4.
- Nil
 - ◆ Objet signifiant “rien”
 - ◆ Valeur initiale de toutes les variables
 - ◆ |a| ^a

Bloc

[*contenu_bloc**]

- *contenu_bloc* :
 - ◆ déclaration de variable(s) puis 1 ou plusieurs :
 - ◆ expression conditionnelle,
 - ◆ expression de boucle
 - ◆ message
 - ◆ commentaire
- Un bloc a accès aux mêmes variables que la méthode dans laquelle il est défini
- Blocs sans argument :
 - ◆ ifTrue: [x := 2]
- Blocs avec argument(s) :
 - ◆ [:var1 | :var2 | *expression(s)*

var1 et var2 sont locales au bloc

Opérations de comparaison

■ *valeur compareur valeur*

- ◆ *valeur* : toute expression retournant un objet pouvant être comparé
- ◆ *compareur* : > < = ~= >= <= ==
- ◆ Résultat : *true* ou *false*, instances de True et False

■ And et or :

- ◆ & et |
 - ◆ (a > 0) & (b < 0)
 - ◆ (a > 0) & (b < 0) | (x > y)
- ◆ and: et or:
 - ◆ boolean *and*: [code]
 - ◆ |ab|
(1 > 2) and: [ab := Bidon new. ab nom:'abc'. ^ab nom >'aaa']
- ◆ xor:
 - ◆ true xor: false

Opérations de comparaison

■ not

- ◆ (5 > 1) not
- ◆ (5 > 1) not --> false
- not 5 > 1 --> erreur

■ Expression de conditions

boolean ifTrue: [code] ifFalse: [code]

← bloc sans argument

```
[x y val]
x := 1.
y := 2.
(x > y) ifTrue: [val := x]
           ifFalse: [val := y]. --> 2
^val
```

Boucles

- ◆ faire n fois quelque chose
 - ◆ timesRepeat:
- ◆ faire quelque chose tant que une condition est vérifiée
 - ◆ whileTrue:
- ◆ faire quelque chose tant que une condition n' est pas vérifiée
 - ◆ whileFalse:
- ◆ faire quelque chose en utilisant un index, en commençant avec une valeur initiale et en finissant avec une valeur d'arrêt
 - ◆ to:do:

Boucles

■ timesRepeat:

```
|x|  
x := 2.  
3 timesRepeat: [x := x + 1].  
^x
```

■ whileTrue: et whileFalse:

```
“Exécuter la boucle jusque x < y”  
|x y|  
x := 5.  
y := 0.  
[x < y] whileFalse: [y := y + 1].  
^y
```

R = 6

```
“Exécuter la boucle jusque y > x”  
|x y|  
x := 5.  
y := 0.  
[y <= x] whileTrue: [y := y + 1].  
^y
```

R = 6

Boucles

■ to:do:

- ◆ *number1* to: *number2* do: [:var| code].

```
"exécute ce bloc 3 fois, i prenant chaque valeur  
de 1 à 3. A la fin, x vaut 6"  
|x|  
x:=0.  
1 to: 3 do:[i|x := x + i].  
^x
```

Classes

■ Rappels

- ◆ une classe regroupe les objets ayant même structure et même comportement
- ◆ Programmer en Smalltalk = créer des classes, des instances et faire en sorte que ces différents objets s'envoient des messages

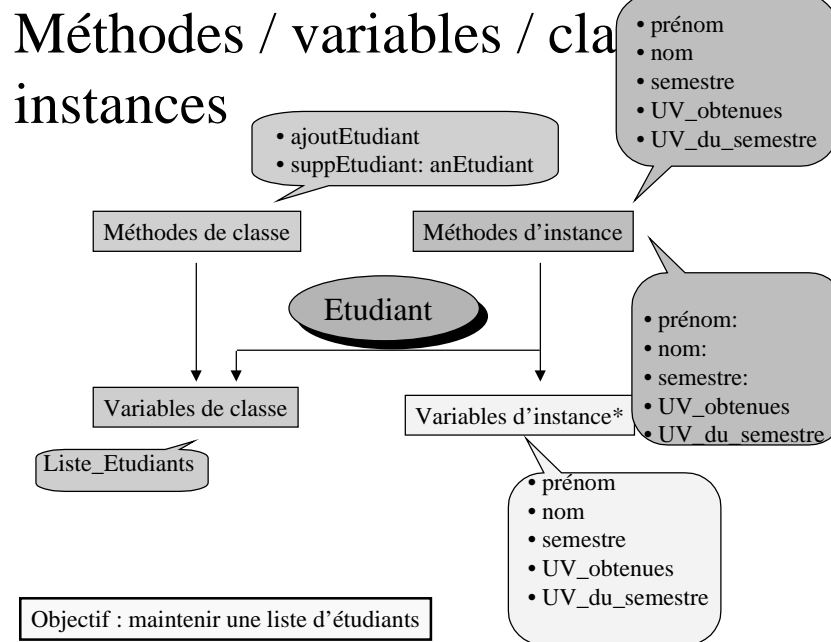
■ Structure de la classe

- ◆ elle définit celle de ses (futures) instances
- ◆ composée de variables de classe (<>variables d'instance) dont le nom commence par une Majuscule

■ Comportement de la classe

- ◆ décrit par des méthodes de classe
- ◆ même notation que pour les méthodes d'instance

Méthodes / variables / classe instances



Création de classes

Etudiant

■ Utilisation de l'environnement de développement

- ◆ Cliquer sur une catégorie de classe -> le texte de création d'une classe s'affiche
- ◆ Définir la classe par des substitutions

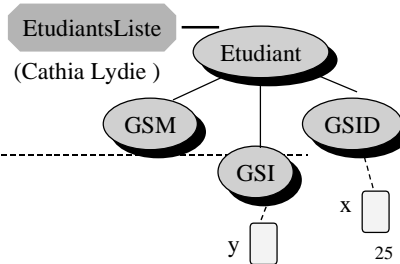
```
Object subclass: #NameOfClass
  instanceVariableNames: 'instVarName1 instVarName2'
  classVariableNames: 'ClassVarName1 ClassVarName2'
  poolDictionaries: ''
  category: 'Collections-Support'
```

```
Object subclass: #Etudiant
  instanceVariableNames: 'prénom nom semestre UV_obtenues UV_du_semestre'
  classVariableNames: 'Liste_Etudiants'
  poolDictionaries: ''
  category: 'Collections-Support'
```

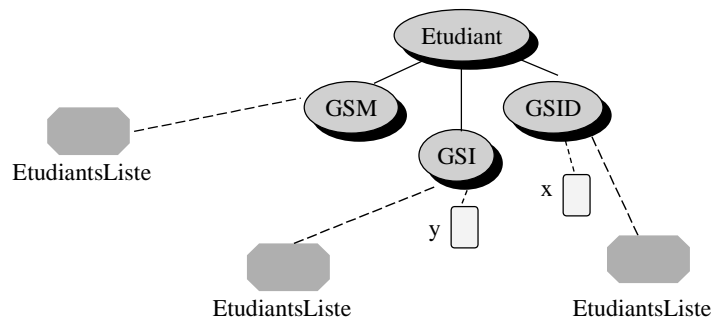
Variables de classe/d'instances

- On souhaite créer des sous-classes GSM, GSID, GSI de Etudiant, en maintenant dans chaque classe une liste d'Etudiants
 - On ne peut utiliser EtudiantsListe, elle est partagée par ses instances et ses sous-classes
- => Définir EtudiantsListe en tant que variable de classe des sous-classes

```
|x y|
x _ GSID new.
x class EtudiantsListe: #(toto titi).
^x
-----
y _ GSI new.
y class EtudiantsListe: #(Cathia Lydie).
^y
```



U.T.T. S. Lorientte.



```
|x y|
x _ GSID new.
x class EtudiantsListe: #(Alain Jerome).
y _ GSI new.
y class EtudiantsListe: #(Cathia Lydie).
```

```
GSID EtudiantsListe.
GSI EtudiantsListe.
```

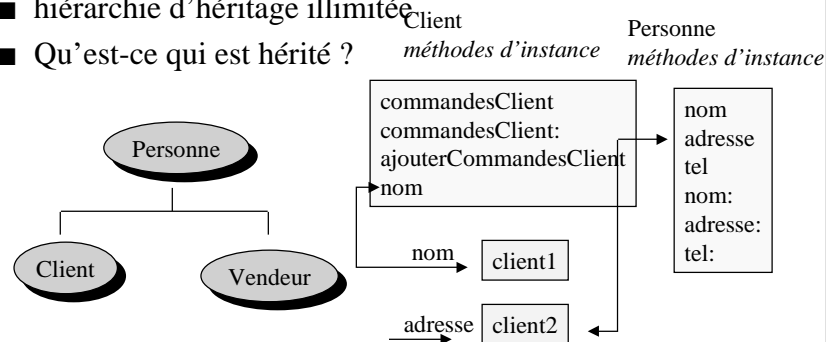
```
(Alain Jerome )
(Cathia Lydie )
```

U.T.T. S. Lorientte.

26

Héritage

- Réutilisabilité
- Extensibilité
- hiérarchie d'héritage illimitée
- Qu'est-ce qui est hérité ?



Spécialisation de classes

- Redéfinition de méthodes
 - ◆ lorsque la méthode doit être distincte dans la sous-classe
 - ◆ pour bloquer l'héritage
 - ◆ dans Vendeur
 - tel
- Transcript show: 'le téléphone des vendeurs est privé'
- Ajouter la gestion des commandes à la classe Client
 - ◆ méthodes
 - ◆ commandesClient, commandesClient:, ajouteCommandesClient:
 - ◆ maintenir une liste des individus
 - ◆ variable de classe
 - ListePersonne dans Personne
 - ◆ méthodes de classe
 - listePersonne et listePersonne:

hérité/pas hérité

■ duplication

- ◆ des variables d'instance de C dans les instances de C et de ses sous-classes

■ Partage

- ◆ des variables de classe de C dans les sous-classes
- ◆ des méthodes d'instance “ “ “ “
- ◆ des méthodes de classe de C “ “ “

■ Redéfinition possible

- ◆ des méthodes d'instance et de classe

■ Redéfinition impossible

- ◆ des variables d'instance et de classe

Classes abstraites

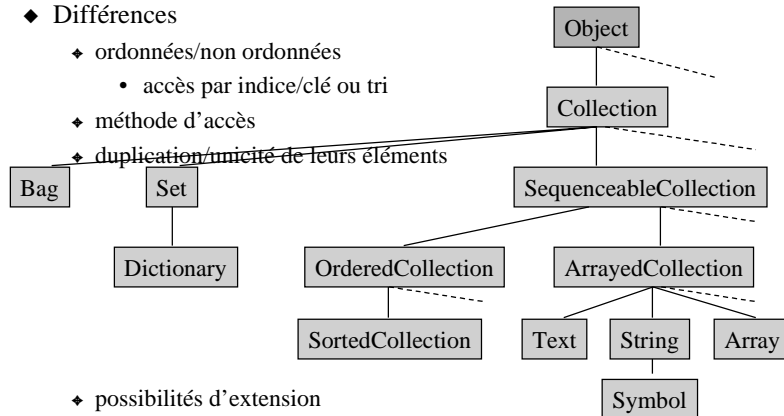
Classes ne pouvant avoir d'instance

- ◆ Bag
new
^super new setDictionary
- ◆ Boolean
new
self error: 'You may not create any more Booleans - this is two-valued logic'
- ◆ etc.

Collections

■ désigne un ensemble de classes

- ◆ Point commun : regroupe des objets *divers*
- ◆ Différences
 - ◆ ordonnées/non ordonnées
 - accès par indice/clé ou tri
 - ◆ méthode d'accès
 - ◆ duplication/unicité de leurs éléments



- ◆ possibilités d'extension

Propriétés

Collections	indexée	taille variable	double	tri	tout objet
Bag	N	O	O	N	sauf nil
Set	N	O	N	N	sauf nil
Array	O	N	O	N	O
OrderedCollection	O	O	O	N	O
SortedCollection	O	O	O	O	O
String	O	N	O	N	Character
Symbol	O	N	N	N	Character
Dictionary	N	O	N	N	Clé + tout objet

Utilisations

■ Bag

```
aBag := #(1 2 3 3 3 3) asBag.  
aBag size
```

6

```
aBag := #(1 2 3 3 3 3) asBag.  
aBag occurrencesOf: 3
```

4

■ Set

```
aSet := #(1 2 3 3 3 3) asSet.  
aSet size.
```

3

```
aSet := #(1 2 3 3 3 3) asSet.  
aSet occurrencesOf: 3.
```

1

■ Array

```
anArray := Array new: 3.  
anArray at: 1 put: 'hi'.  
anArray at: 1
```

'hi'

Utilisations

■ OrderedCollection

```
cars := OrderedCollection new.  
cars add: 'Cadillac'.  
cars addFirst: 'Lexus'.  
cars addLast: 'Corvette'.  
^cars
```

OrderedCollection ('Lexus' 'Cadillac' 'Corvette')

■ SortedCollection

```
 #(4 3 5 2 1) asSortedCollection: [:a :b| a>= b]
```

SortedCollection (5 4 3 2 1)

```
 #(un' deux' trois) asSortedCollection: [:a :b| a size >= b size]
```

SortedCollection ('trois' 'deux' 'un')

Messages

■ création

- ◆ new
- ◆ new:
- ◆ with:, with:with:, with:with:with:,with:with:with:with:

```
c := Array new: 10.
c at: 1 put: 'string 1'.
c at: 10 put: 'string 10'.
c
```

```
c at: 0 put: 'string 01'.
c at: 11 put: 'string 11'.
```

```
Array with: 'String 1' with: 'String 2'.
OrderedCollection with: 1 with: 2 with: 3 with: 4.
```

■ manipulation

- ◆ de collection
 - ◆ itération
 - do:
 - detect:
 - select:

```
"Compter le nombre de voyelles dans aString"
[voyelles]
voyelles := 0.
'ceci est un cours sur Smalltalk' do: [:v| v isVowel
ifTrue: [voyelles := voyelles + 1]].
^voyelles 9
```

```
'ceci est un cours sur Smalltalk' select: [:v| v isVowel]. 'eieuouuaa'
```

Messages

- ◆ énumération
 - collect:
 - reject:
- ◆ ajout d'éléments
 - add:
- ◆ suppression d'éléments
 - remove:
- ◆ concaténation
 - ,
- ◆ d'éléments
 - ◆ suppression
 - removeAt:
 - ◆ modification
 - at:put:
 - ◆ test
 - size

```
'ceci est un cours sur Smalltalk' reject: [:v | v isVowel]
```

```
'cc st n crs sr Smltlk'
```

```
c _ #(1 b c) asOrderedCollection.
c remove: 1.
^c
```

```
OrderedCollection (b c)
```

```
|c|
c := OrderedCollection new.
c add: 'string 1'.
c add: 'string 2'.
c removeAt: 1.
```

```
c OrderedCollection ('string 2')
```

Applicabilité des méthodes

Collections	do:	add:	remove:	at:	at:put:
Bag	O	O	O	N	N
Set	O	O	O	N	N
Array	O	N	N	O	O
OrderedCollection	O	O	O	O	O
SortedCollection	O	O	O	O	N
String	O	O	N	O	O
Symbol	O	N	N	N	O
Dictionary	O	N	N	N	O

Applicabilité des méthodes

Collections	,	detect:	select:	reject:	collect:	remove Index:
Bag	N	O	O	O	O	N
Set	N	O	O	O	O	N
Array	O	O	O	O	O	N
OrderedCollection	O	O	O	O	O	O
SortedCollection	O	O	O	O	O	O
String	N	O	O	O	O	N
Symbol	O	O	O	O	O	N
Dictionary	O	O	O	O	O	N

String

- des caractères seulement
- taille fixe
- indexée
 - ◆ supporte
 - ✦ at:put:
 - ✦ at:
 - ◆ ne supporte pas (taille fixe)
 - ✦ add:
 - ✦ remove:
- aString est constitué de n octets ° n OID
 - ◆ accès par un indice de 1 à n
 - ◆ at: retourne un OID de Character !

```
(String with: $x) at:1 put:$y.  
'x' at: 1 put: $y.
```

MCours.com

Symboles

- Symbol =
 - ◆ classe des sélecteurs
 - ✦ Client nom -> symbole #nom
 - ◆ classe des clés de dictionnaire
- se comporte comme la classe String sauf :
 - ◆ un symbole ne peut être que lu
 - ==> pas de at:put:
 - ◆ un symbole est unique
 - ◆ un symbole ne peut être créé qu'avec des littéraux, sauf "vide"
 - ==> pas de new, new:, with:, with:with:, etc.
 - ◆ #Symbol1 , #Symbol2 = 'Symbol1Symbol2' !
 - ◆ taille fixe
 - ==> pas de add:

Dictionary

- ~ table
- collection non ordonnée
- couple clé/valeur = association (anAssociation)
- anAssociation comprend
 - ◆ un OID de l'objet "clé"
 - ◆ un OID de l'objet "valeur"
- aDictionary = collection d'OID de "anAssociation"
- aucune restriction sur le contenu du dictionnaire
 - clés : souvent strings ou symbols
 - ◆ sauf : clés uniques dans un même dictionnaire
- accès aux valeurs
 - ◆ par l'intermédiaire des clés, par =
 - ◆ aDictionary printInt -> seulement les valeurs

Dictionary

- add: anAssociation

```
[aDictionary]
aDictionary := Dictionary new.
aDictionary add: (Association key: '80000' value: 'Amiens').
```

- removeKey: aKey

```
[aDictionary]
aDictionary := Dictionary new.
aDictionary add: (Association key: '80000' value: 'Amiens').
aDictionary removeKey: '80000'.
```

- at: aKey

```
[aDictionary]
aDictionary := Dictionary new.
aDictionary add: (Association key: '80000' value: 'Amiens').
aDictionary at: '80000'.
```

■ keyAtValue:

```
[aDictionary]
aDictionary := Dictionary new.
aDictionary add: (Association key: '80000' value: 'Amiens').
aDictionary keyAtValue: 'Amiens'.      '80000'
```

■ at: aKey put: aValue

```
[aDictionary]
aDictionary := Dictionary new.
aDictionary add: (Association key: '80000' value: 'Amiens').
aDictionary at: '80000' put: 'Abbeville'.
aDictionary at: '10000' put: 'Troyes'.
aDictionary
```

■ keys

```
OrderedCollection ('code postal = 10000' 'code postal = 80000' )
[aDictionary]
aDictionary := Dictionary new.
aDictionary add: (Association key: '80000' value: 'Amiens').
aDictionary at: '80000' put: 'Abbeville'.
aDictionary at: '10000' put: 'Troyes'.
aDictionary keys.      Set ('10000' '80000' )
```

■ do:

```
[aDictionary]
aDictionary := Dictionary new.
aCollection := OrderedCollection new.
aDictionary add: (Association key: '80000' value: 'Amiens').
aDictionary at: '80000' put: 'Amiens'.
aDictionary at: '10000' put: 'Troyes'.
aDictionary do[:ville]aCollection add: 'ville = ', ville].
aCollection.
```

```
OrderedCollection ('ville = Troyes' 'ville = Amiens' )
```

■ keysDo:

```
[aDictionary]
aDictionary := Dictionary new.
aCollection := OrderedCollection new.
aDictionary add: (Association key: '80000' value: 'Amiens').
aDictionary at: '80000' put: 'Amiens'.
aDictionary at: '10000' put: 'Troyes'.
aDictionary keysDo[:cp]aCollection add: 'code postal = ', cp].
aCollection.
```

```
OrderedCollection ('code postal = 10000' 'code postal = 80000' )
```

Flux de données (streams)

- Structure supportant entrées, sorties, parcours séquentiel, accès direct à des fins de transfert ou d'exploitation
- fenêtre partielle / totale sur une collection
- peut être créé
 - ◆ de rien : fonctionne
 - ◆ comme un buffer
- Toute structure répondant à at:
 - ◆ sur une structure existante, ordonnée

Flux de données (streams)

- Stream : classe abstraite
- ReadStream : accès uniquement en lecture à une collection existante
 - ◆ Erreur en cas de
 - ◆ tentative d'écriture
 - ◆ dépassement de la fin du stream
- WriteStream, ReadWriteStream
 - ◆ si tentative d'accès au-delà de la fin du stream -> message à la collection support "agrandis-toi !"

Flux de données (streams)

```
Smalltalk at: #s put: (ReadStream on: (String new: 0)).  
'ceci' printOn: s.  
' est un' printOn: s.  
s position: 1.  
(s contents)
```

```
"ceci" est un"
```

```
Smalltalk at: #s put: (ReadStream on: (Array new: 0)).  
s next: 4 put: 'Hi'.  
s position: 1.
```

```
('Hi' 'Hi' 'Hi' 'Hi' nil nil nil nil nil nil nil nil nil nil nil nil nil nil )
```

Flux de données (streams)

```
Stream ()  
.....  
PositionableStream ('collection' 'position' 'readLimit')  
  ReadStream ()  
.....  
  WriteStream ('writeLimit')  
.....  
    LimitedWriteStream ('limit' 'limitBlock')  
    ReadWriteStream ()  
.....  
      FileStream ('rmode ' )  
    TextStream ()
```


Stream

- objets supportant l'accès séquentiel aux *collections* et *fichiers* de données séquentielles
- Réagissent à
 - ◆ atEnd -> test de fin de stream
 - ◆ do: -> itération
 - ◆ next -> prochain item, next:, nextMatchFor:
- Positionable Streams :
 - ◆ ReadWriteStream ou WriteStream
 - ◆ Array
 - ◆ String
 - ◆ ReadStream
 - ◆ idem
 - + OrderedCollection

PositionableStream

- Propriétés
 - ◆ basés sur des collections (tjs indexées) organisées en séquences
 - ◆ "tête" de lecture/écriture positionnée sur le dernier objet manipulé, positionnable
 - ◆ Lecture et/ou écriture d'objets
- Réactivité à
 - ◆ contents, isEmpty, position, position:n, reset, setToEnd, skip: n, skipTo:, upToEnd, peek, peekFor: (lecture sans déplacement), upTo:, upToEnd

ReadStream

- Collection support non modifiable
- Opérations
 - ◆ lecture
 - ◆ déplacement dans le flot
 - ◆ création: on:from:to:

```
ReadStream on: 'Exemple de creation d'un flot de donnees en lecture'
```

```
rs := ReadStream on: 'Exemple de creation d'un flot de données en lecture'.  
^rs upTo: $  
    'Exemple'
```

```
rs := ReadStream on: 'Exemple de creation d'un flot de donnees en lecture'.  
^rs next $E
```

ReadStream

```
[galerie visite]  
galerie := #(Picasso Vinci Dali Picasso David David Vinci Monnet).  
visite := ReadStream on: galerie from: 2 to: 4.  
visite setToEnd.  
visite atEnd.
```

WriteStream

- écriture, mais pas de lecture
- souvent utilisés pour construire des string à afficher ou imprimer
- Création :
 - ◆ on: aSequenceableCollection (taille fixe)
 - ◆ extension automatique de la collection si besoin

```
WriteStream on:String new
```

- Ecriture

- ◆ nextPut:, nextPutAll:

```
s _WriteStream on:String new.  
s nextChunkPut:'toto'
```

```
s _WriteStream on:String new.  
s nextPutAll:'bonjour'
```

```
'bonjour'
```

```
aWriteStream
```

ReadWriteStream

- lire
- écrire
- fermeture
 - ◆ close
 - ◆ closed
- compatibilité de fichier
 - ◆ fileIn
 - ◆ fileOutChanges

FileStream

- similaire aux ReadWriteStream mais opèrent sur des fichiers du système d'exploitation
- lisent/écrivent caractères ou octets
- close/open:

```
|file|  
file := FileStream newFileNamed: 'toto2.txt'.  
file nextPutAll: 'bonjour'.  
file contentsOfEntireFile.  
file close.
```

```
|file|  
file := StandardFileStream newFileNamed: 'toto3.txt'.  
file close.  
file open.
```

FileStream

```
|entree sortie|  
entree := FileStream oldFileNamed: 'toto4.txt'.  
Smalltalk at:#f put: entree.  
sortie := StandardFileStream newFileNamed: 'toto4.bak5'.  
entree reset.  
[entree atEnd]  
    whileFalse: [sortie nextPut: (entree next)].  
entree close.  
sortie close.  
(StandardFileStream oldFileNamed: 'toto4.bak5') dataContents.
```