LO02

TUTORIEL SQUEAK

Version 2.7

Sophie Loriette - Université de technologie de Troyes

Introduction

L'original de ce tutoriel concerne SMALLTALK-80 et est disponible au format LATEX à l'adresse : ftp://ftp.enst-bretagne.fr/pub/smalltalk/st_tutorial*.

L' adaptation présentée ici à l' environnement squeak, réalisée par S. Loriette., concerne la version 2.7. Tout commentaire est le bienvenu. Squeak 2.7 est disponible à l'adresse : http://squeak.cs.uiuc.edu/#community

I. Démarrage

A. Les principes de base

a) L'interface multifenêtre

L'environnement est multifenêtre et indépendant de la plateforme d'accueil. Smalltalk tourne aussi bien sous X, Mac ou Windows. Smalltalk offre pour ce faire des classes qui masquent les différences de ces systèmes en uniformisant leur utilisation.

Pour interagir avec le système on utilise essentiellement la souris et ses 2 boutons. Les deux boutons s'utilisent de la manière suivante :

Gauche

Pour sélectionner un objet, une position dans une fenêtre

Pour ouvrir le Launcher (menu principal) en dehors de toute fenêtre

Droite

Dans une fenêtre, pour inspecter un objet, exécuter du code. Attention : dans le browser, le menu varie selon la sous-fenêtre duquel on l'active !

puis avec more...

ou

Shift + droite : formater, imprimer dans un fichier, etc.

Exercice 1 Tapez, Sélectionnez, puis évaluez :

1. 1 + 5

2. $A

3. 'Small','talk'

4. Transcript show: 'Hello World !'

Utilisez le clavier, après avoir positionné le curseur dans une fenêtre Smalltalk d'édition. Les fenêtres d'édition sont nombreuses. Au démarrage, vous pouvez utiliser le Transcript, l' InstallationWorkspace, le SystemWorkspace. Mais utilisez plutôt un espace de travail (Workspace) en sélectionnant l'option Open...Open Workspace du Launcher. Une fois le texte frappé ou copié à partir de ce tutoriel, sélectionnez-le, puis cliquez sur le bouton droit de la souris. Pour évaluer, vous utiliserez l'action du menu local do it ou print it ou encore inspect.

Les différents types d'évaluation analysent le texte sélectionné (Parsing), puis évaluent. L'objet résultat de l'évaluation est traité différemment selon le choix fait :

- do it envoie le message simplement et le résultat n'est visible que par l'intermédiaire d'effets de bord
- print it, le message printOn: est envoyé au résulat ce qui a pour effet de le faire s'imprimer sous forme de chaîne dans la fenêtre où a eu lieu l'évaluation.
- inspect, le message inspect est envoyé au résultat ce qui a pour effet d'ouvrir un inspecteur sur cet objet

b) Le tout objet : principes

1.Le premier principe de Smalltalk est que tout y est objet.
2.Le deuxième principe est que tout objet appartient à une classe.
3.Le dernier principe est que la seule structure de contrôle est l'envoi de messages aux objets

On trouvera donc dans Smalltalk des classes d'objets pour les nombres, les dates, les listes, les fenêtres, les points, les classes, les méthodes compilées, les booléens, les processus, ...

Exercice 2 Cherchez la classe Integer. D'après sa définition, déterminez quelle est sa super-classe. Cherchez ensuite cette super-classe.

Pour cherchez une classe plusieurs techniques se présentent :

1.On connaît ou on imagine son nom. find class.. du menu local au panneau catégories du browser permet de trouver une classe .On donne alors le nom supposé de la classe. Dans le cas contraire, on peut demander un browser sur les classes. Elles sont alors listées par ordre alphabétique (choisir "Browse all" sur le panneau des catégories du Browser). Exemple : cherchez Object

2.On imagine ou on connaît des méthodes que devraient savoir exécuter les objets de cette classe. A partir de toute fenêtre dans laquelle on a tapé le nom d'une méthode, activer l'option implementors of.. sur le nom de la méthode sélectionné (pour y accéder, il faut aller dans "more" du menu accessible par pression sur le bouton droit de la souris), puis on peut demander "Browse full" dans le menu accessible alors par pression sur le bouton droit de la souris sur la ou les classes trouvées

c) Les erreurs à l'évaluation

Les erreurs qui peuvent survenir lors d'une évaluation sont les suivantes :

Référence à un objet inconnu

Smalltalk vous demande alors de considérer cet objet comme variable de classe, variable globale, etc.

Un message est inconnu

Smalltalk vous propose de le corriger, abandonner, etc.

S'il sagit d'une erreur de frappe essayez de corriger, sinon abandonnez.

B. Le Launcher

Le Launcher est le guide principal de navigation dans Smalltalk. Il est accessible, en dehors de toute fenêtre, par le bouton droit de la souris.

Il est composé des commandes suivantes :

keep this menu up
previous project
jump to project...
restore display.
open...

windows...
changes…
help…
appearance…
do..
save
save as…
save and quit
quit...

C. Le Transcript

Cette fenêtre collecte des textes qui lui sont envoyés. C'est une instance de la classe TextCollector. Ces textes sont transmis par le système, ou par les applications que vous développerez.

Exercice 3 Cherchez les messages que vous pouvez envoyer à l'objet Transcript, instance de la classe TextCollector.
Puis faites *afficher* dans le Transcript le texte suivant, après avoir tout effacé :

Premiere ligne
Deuxieme ligne
Fin

La succession des envois de message est indiquée par le caractère <point> (.).

Indications :
Évaluez le texte suivant dans un Workspace :

```
Transcript clear.
Transcript show: 'Premiere ligne'
Transcript show: 'Deuxieme ligne'
Fin
```

Une autre solution consiste à cascader les messages. En effet, ici tous les messages sont envoyés au même objet (Transcript). Pour ce faire, on utilise le séparateur point virgule (;). Ce qui donne :

```
Transcript clear ; show: 'Première ligne' ; cr ; show: 'Deuxième ligne' ; cr ; show: 'Fin'.
```

D. Le SystemWorkspace

Cette fenêtre contient des exemples, et des paramètres de configuration.

E. Le Workspace

Le workspace est un espace de travail vous permettant d'évaluer n'importe quel code.

Pour vérifier l'heure vous pouvez évaluer, après avoir ouvert un Workspace, le texte : Time now. Time est une classe de Smalltalk qui permet d'accéder à l'heure en lui envoyant le message now.

Exercice 4 Cherchez ce que peut faire d'autre la classe Time, et ses instances. Essayez :

1. Time now hours
2. Time now minutes
3. Time new hours: 12 minutes: 0 seconds: 0

Indications ...
Pour chercher de l'information concernant une classe, le moyen le plus direct est de la chercher par le browser

Il est conseillé de faire des sauvegardes régulières de votre image, qui conservera l'état de vos classes à l'instant où vous avez effectué cette opération.
Utilisez pour cela le menu save as...

II. Calculs en Smalltalk

Exemples
Les classes numériques

A. Exemples

Exercice 5 Tapez, sélectionnez, puis évaluez :

1. (8 + (4/2))
2. (2 / 3) * (4 / 3)
3. 56 gcd: 32
4. 17 factorial
5. 5 max: 3
6. 1.4 * 10
7. 2r100 + 2r11
8. 13 sqrt

Utilisez le clavier, après avoir positionné le curseur dans une fenêtre Smalltalk d'édition. Pour évaluer, vous utiliserez l'action du menu local do it ou print it ou encore inspect. Comparez la nature des objets résultat...

Indications ...
Pour observer la classe de l'objet résultant du calcul on peut :

Inspecter le résultat (évaluer avec inspect). La classe de l'objet apparaît dans le titre de la fenêtre (un Inspector). Le résultat est observable en sélectionnant dans l'inspecteur le champ self, qui représente l'objet lui-même.

Évaluer, et imprimer le texte en ajoutant le message class. Par exemple, 3 class imprime SmallInteger.

B. Les classes numériques

La hiérarchie des classes numériques (entre autres) est donnée ci-dessous. Les informations entre parenthèses correspondent aux variables d'instances, c'est-à-dire aux noms des objets constituant l'état des objets de la classe considérée :

```
ProtoObject ()
        Object ()

                Magnitude ()
                        Character ('value' )
                        Date ('day' 'year' )
                        LookupKey ('key' )
                                Association ('value' )
                                        WeakKeyAssociation ()
                                        WeakValueAssociation ()
                        MessageTally ('class' 'method' 'tally' 'receivers' 'senders' 'time' )
                        Number ()
                                Float ()
                                Fraction ('numerator' 'denominator' )
                                Integer ()
                                        LargePositiveInteger ()
                                                LargeNegativeInteger ()
                                        SmallInteger ()
                        Time ('hours' 'minutes' 'seconds' )
```

Certaines classes ne peuvent pas avoir d'instances, exemple : Boolean. Ce sont des classes abstraites qui servent à factoriser des comportements. Ces classes sont essentielles pour la structuration de la hiérarchie d'héritage. Elles se reconnaissent par certaines de leurs méthodes (manières de répondre à un message) qui déclenchent une erreur en signalant qu'une sous classe, dite concrète, aurait dû pouvoir répondre à ce message avant.

Exercice 6 Le message indiquant qu'une méthode est de la responsabilité d'une des sous-classes est subclassResponsibility. Avec cette information, pouvez-vous détecter les classes abstraites de Smalltalk ?

Indications ... Une méthode qui doit être réalisée par une sous classe possède la forme suivante :

```
<sélecteur>

self subclassResponsibility
```

Pour trouver les classes abstraites, il suffit de chercher les implementors (donc des classes) des méthodes qui utilisent (envoient) subclassResponsibility. Dans le workspace, on fera donc Sendors of it... après avoir sélectionné "subclassResponsibility". On trouve : Boolean, Collection, Filestream, Magnitude, Number, Stream, ...

III. Les différents objets

A. Conventions syntaxiques

Les noms des objets sont soumis à une convention simple qui délimite leur portée.

Les objets globaux (au système, à une classe) possèdent un nom qui commence par une majuscule. Le nom des autres commence par une minuscule.

Dans la première catégorie on trouve les classes (Number, Point, Compiler, ...), certains objets globaux comme Transcript, Processor, Undefined, ... L'ensemble des objets globaux est contenu dans un dictionnaire qui s'appelle Smalltalk, lui-même objet global.

Nom	*Maj.*	*Exemple*
Categorie	oui	Numeric-Magnitudes
Classe	oui	Date
Variable de classe	oui	MonthNames
variable globale	oui	Smalltalk
variable pool	non	cr
variable d'instance	non	year
Variable temporaire	non	aDate
Protocole	non	accessing
Méthode	non	monthName

La taille des identificateurs n'est pas limitée, pour améliorer la lisibilité des noms, à chaque changement de mot dans un identificateur, on place une majuscule.

Ainsi, voiciUnIdentificateurDeNomTresLongMaisEnFaitAssezLisibleToutDeMeme.

B. Inspecter les objets

Tous les objets savent répondre au message inspect. En fait, en plus d'inspecter un objet, l'inspecteur permet d'éditer l'objet, d'en changer le contenu, l'état.

Un inspecteur se décompose en deux parties :

1. Une liste des variables d'instances de l'objet inspecté, ou une liste des objets contenus dans une liste ou un dictionnaire si l'objet inspecté est une liste, un tableau ou un dictionnaire.
2. Un éditeur qui affiche et édite l'objet sélectionné dans la liste.

Exercice 7

- ✓ Créez un Point, inspectez le puis modifiez son abscisse. Les modifications sont validées par le choix accept dans le menu local. Pour créer un Point, on s'appuiera sur la méthode @ .

- ✓ Inspectez le tableau :

```
#($A 1 #truc 'machin' 4)
```

Indications ... Pour valider une modification, il faut toujours, à l'aide du menu local, réaliser un accept. Si après avoir édité, on désire annuler, il faut alors choisir cancel. La création d'un point peut être effectuée par l'évaluation de : 9 @ 6, par exemple.

Le contenu d'un objet inspecté sont des objets qui eux-mêmes peuvent être inspectés...

C. Les objets globaux

Les objets globaux du système, qui ne sont pas des classes sont :

Smalltalk
est le dictionnaire de tous les objets globaux. Smalltalk contient, en particulier, toutes les classes.

Exercice 8 Évaluez le texte

```
MonTableau := Array with: 1 with: $A with: 'Un' with: 1.
```

Faites de MonTableau un objet global puis inspectez Smalltalk et MonTableau.

Undeclared

est le dictionnaire des objets non déclarés. Lors de l'évaluation, si une variable n'est pas connue, le compilateur vous propose plusieurs possibilités. Si vous choisissez undeclared, l'objet est placé dans ce dictionnaire. Le compilateur Smalltalk décide de mettre les objets non définis dans Undeclared lorsqu'il charge des fichiers sources par un File In .

D. Les classes

Le système possède 779 classes. Ces classes sont regoupées en catégories qui sont des ensembles de classes. Par exemple, la catégorie Graphics-Primitives contient les classes Point, Rectangle, ...

Exercice 9 Ouvrez un Browser du système, puis cherchez la catégorie qui contient la classe Object. Sélectionnez cette classe, puis observez sa hiérarchie.

Indications ... Le moyen le plus efficace est de chercher la classe Object à partir du menu local des catégories. Smalltalk trouve tout seul la position de la classe recherchée. Pour voir la hiérarchie de Object, il faut sélectionner la classe, si ce n'est déjà fait, puis, à l'aide du menu local du panneau des classes choisir l'action hiérarchie. La hiérarchie apparaît dans la fenêtre de texte du Browser...

Les classes sont des moules d'objets. Les objets sont, en général, caractérisés par des variables d'instance. Toutefois, certaines classes représentent des objets "auto-décrits" comme Integer ou Float, et n'ont pas, à ce titre, de variables d'instance.

Exercice 10 Comparez le résulat de l'inspection d'un Point et d'un Integer.

Indications ... Pour inspecter un Point évaluez (4@6), et pour un Integer, 121. Un inspecteur affiche dans la fenêtre de gauche la liste des variables d'instance de l'objet inspecté, en plus de self, qui est l'objet lui-même.

Les classes sont, conformément au premier principe de Smalltalk, des objets. Ils appartiennent à une classe et peuvent recevoir des messages. Les classes des objets repésentant les classes sont appelées métaclasses et les messages que comprennent les classes, méthodes de classe.

Exercice 11 La méthode de classe la plus utilisée est new, cherchez ses différentes formes. La méthode la plus primitive, celle qui crée des instances dont les variables d'instances sont indéfinies (nil), est basicNew

Pour observer la différence, on ouvre un Browser puis on sélectionne alternativement soit le bouton instance, soit le bouton class qui se trouvent sous la liste des classes. Apparaissent alors les protocoles des méthodes d'instances ou de classe, la hiérarchie des classes ou des métaclasses (qui sont symétriques).

E. Les méthodes

Les méthodes sont les façons de répondre aux messages. Elles ont la forme suivante :

selecteur et arguments

"commentaire entre double quotes"

| variables temporaires entre barres verticales |

expressions Smalltalk séparées par des points.

Ce schéma de définition des méthodes apparaît dès que vous sélectionnez un protocole. Les protocoles sont aux méthodes ce que les catégories sont aux classes : des ensembles pour les organiser.

Exercice 12 Dans la classe Object, sélectionnez le protocole printing. Puis regardez le code des méthodes qui apparaissent.

Indications ... Les méthodes définies dans Object sont définies pour tous les objets du système Smalltalk. Ainsi, la méthode printString permet à chaque objet de construire une chaîne de caractères le représentant. Par exemple :

4 printString donne '4'
(Point x: 0 y: 7) printString donne '0@7'
Transcript printString donne 'a TranscriptStream', ce qui est la méthode par défaut, définie dans Object.

La méthode printString fait en effet appel à la méthode printOn:. Pour voir ce que fait printOn:, il suffit de chercher les messages utilisés par printString... ou alors de chercher les implementors of printOn:.

Trois techniques de navigation sont fondamentales pour trouver de l'information, comprendre le sens d'un message. Ces techniques permettent :

1.De savoir quelles classes savent répondre à un message. Ou encore, quelles sont les différentes façons de répondre à un message : les implementors
2.De savoir quels sont les messages qui utilisent une méthode donnée : les senders
3.De savoir quelles sont les façons de répondre aux messages qui sont utilisés par une méthode : les messages

Exercice 13 Pour mettre en pratique ces outils de navigation,

1.Quelles sont les classes des objets qui savent répondre à ifTrue: ?
2.Quelles sont les classes des objets qui savent répondre à whileTrue: ? Comment whileTrue fonctionne-t-il ?
3.Quelles sont les méthodes qui utilisent la méthode x:y: ?

a) Choix des sélecteurs

Les noms des méthodes ne doivent pas être choisis au hasard. En effet pour exploiter au mieux l'environnement existant, il faut tirer profit de tous les mécanismes déjà construits.

Ainsi, pour imprimer une nouvelle classe d'objet, on ne choisira pas le nom "imprime", qui n'a aucune chance d'être connu par le système (on le vérifiera par la recherche de ses implementors), mais on réutilisera le nom printOn:. Ce faisant, lorsqu'on évaluera avec *print it* une expression retournant un objet de cette nouvelle classe, le système utilisera cette nouvelle façon d'imprimer un objet qui surcharge celle par défaut, définie dans Object.

Certaines conventions dans la forme des noms sont à respecter pour faciliter la lecture du code :

1.Les méthodes qui testent une condition et donc retournent un booléen (true ou false) commencent par is.
2.Les méthodes qui convertissent un objet en un autre commencent par as.

IV. Éléments de syntaxe

A. Constantes

nil unique instance de la classe UndefinedObject
true unique instance de la classe True
false unique instance de la classe False
les Integer en base 10 : 1, 0, -45
les Integer en base b : brxxx, comme 2r101, 3r122, 8r70
les Float 1.2, -2.8e-2, 4r1231e-1, 1.23456d103
les Fraction 3/4
les Character $1, $A, $$
les String 'c''est l''été'
les Symbol #+, #show:
les Array #(1 $E #('non' 'typé') Smalltalk)

Remarquez qu'on n'hésite pas à créer une classe pour une instance unique lorsque celle-ci possède des caractéristiques extrêmement particulières (nil, false, true).

B. Expressions

Les expressions Smalltalk sont toutes interprétées comme envois de messages. La règle de priorité est simple : les messages sont évalués de gauche vers la droite.

Par exemple, 8 - 4 * 2 est compris comme 8 recevant le message - avec le paramètre 4. L'objet résultant de cette première évaluation est donc 4 qui reçoit le message * 2. Le résultat est donc 8 (et non pas 0 avec des règles de priorités classiques, liées à la sémantique des opérateurs).

Unaire
Binaires
Keywords
Priorité

a) Unaire

Les messages unaires sont ceux envoyés à un objet, sans paramètre. Quelques messages utiles :

- new vers une classe, retourne une nouvelle instance.
- initialize vers une classe ou une instance, place des valeurs par défaut dans l'objet.
- printString retourne la forme chaîne d'un objet.
- inspect pour ouvrir un inspecteur d'objet.
- isNil, retourne true si le receveur est nil.
- size, retourne la taille d'un objet ; soit son nombre de variables d'instance, soit le nombre d'objets de la collection (liste, tableau, ...)
- halt, interrompt une évaluation, en ouvrant une fenêtre qui permet le débogage.

b) Binaires

Les méthodes binaires sont associées à des symboles comme :

+ (addition) - (soustraction) * (multiplication) / (division) // (division entière) \\ (reste)
& (et booléen) | (ou booléen)
@ (constructeur de point)
, (concatenation de listes ou de chaînes)

c) Keywords

Les keywords sont très employés, ils permettent de construire des messages avec autant d'arguments que l'on souhaite. Ce sont en fait des symboles qui contiennent des « : » pour introduire chaque paramètre. Ainsi :

new: permet de créer un tableau de taille donné.

Évaluez (Array new: 10) inspect.

with:with:with:with: permet de construire une liste ou un tableau contenant 4 objets, les 4 paramètres suivant les 4 deux-points.

Évaluez (Array with: 10 with: $Q with: 'trois' with: (Array new: 10)) inspect.

d) Priorité

L'évaluation d'une expression commence par l'envoi des messages unaires, puis binaires de gauche à droite puis se termine par l'envoi des keywords. Le parenthésage permet de changer cet ordre d'évaluation.

Exercice 14 Devinez quels sont les résultats des évaluations de :

1. 6 + 4 / 2
2. 1 + 3 negated
3. 1 + (3 negated)
4. (1 + 3) negated
5. 2 raisedTo: 3 + 1

Indications ... Les messages sont interpétés comme suit :

1.(6 + 4) / 2 donc 5
2.1 + (3 negated) donc -2
3.1 + (3 negated) donc -2
4.(1 + 3) negated donc -4
5.2 raisedTo: (3 + 1) donc 16

C. Séquences et Cascades

Une évaluation Smalltalk consiste en une séquence de messages envoyés à divers objets. Chaque instruction (envoi de message) est séparée des autres par un point.

Lorsque plusieurs messages sont envoyés à la suite au même objet, il est possible d'utiliser un raccourci : la cascade. On ne répète pas l'objet receveur à chaque instruction, mais on remplace le point (.) par un point-virgule (;).

L'objet retourné à la suite d'une succession d'évaluations est le résultat de la dernière évaluation.

Exercice 15 En déclarant l'objet liste comme temporaire, comparez les résultats de :

1.liste := OrderedCollection new. liste

2.(OrderedCollection new) add: 1; add: 2; add: 3

3.(OrderedCollection new) add: 1; add: 2; add: 3; yourself

4.(liste := OrderedCollection new) add: 1.
liste add: 2.
liste add: 3

5.liste := (OrderedCollection new) add: 1.
liste add: 2.
liste add: 3.
liste

Attention, le message add: qui rajoute un élément dans la liste retourne l'élément ajouté non pas la liste. Ce comportement est à la source de nombreuses erreurs. Pour retourner la liste, on utilise en cascade le message yourself qui retourne le receveur, ici, la liste.

D. Blocs et structures de contrôle

Les blocs sont des objets Smalltalk qui représentent du code analysé non encore évalué. Un bloc est compris entre crochets ([]).

Exercice 16 Évaluez les différentes expressions :

1.[]

2.[Transcript show: 'différé';cr]

3.[Transcript show: 'différé';cr] value

4. [:paramètre | 2 * paramètre] value: 4

5.[:paramètre | 2 * paramètre. 3 * paramètre] value: 4

6.[:paramètre | t := 2 * paramètre. t * paramètre] value: 4

7.[:paramètre | temp | temp := 2 * paramètre. temp * paramètre] value: 4

8.[:p1 :p2 | p1 / p2] value: 4 value: 2

Un bloc possède la même forme qu'une méthode. Il dispose d'une liste de paramètres, indiqués avant la barre verticale.

Le résultat de l'évaluation du bloc s'obtient en lui envoyant le message value. L'objet retourné est le résultat de la dernière expression évaluée.

Les messages que peut recevoir un bloc sont, respectivement pour 0, 1, 2, 3, 4 et n paramètres, value, value:, value:value:, value:value:value:, value:value:value:value:, et value anArrayOfNArguments.

Exercice 17
Regardez à nouveau la manière de répondre à ifTrue:ifFalse:, pour avoir un exemple d'utilisation des blocs.

Indications ... La méthode ifTrue:ifFalse: est connue par trois classes : Boolean, True, False.

La classe Boolean est abstraite, elle ne peut pas possèder d'instance, car une de ses méthodes répond par self subclassResponsibility. Les deux autres classes, par contre, concrétisent Boolean, elles ne possèdent chacune qu'une seule instance, respectivement, true et false.

Pour répondre au message ifTrue:ifFalse:, selon la classe du récepteur, on évalue le bloc argument associé à la branche ifTrue: ou à la branche ifFalse:.

Instructions conditionnelles
Boucles
Itérations

a) Instructions conditionnelles

Les branchements conditionnels sont réalisés à l'aide des méthodes ifTrue:, ifFalse:, ifTrue:ifFalse:, ifFalse:ifTrue: dont les arguments sont des blocs qui sont évalués ou non selon la classe du receveur.

Exercice 18 Évaluez l'expression suivante :

```
| r |
r := Rectangle fromUser.
r width > r height
  ifTrue:[Transcript show: 'large']
  ifFalse:[Transcript show: 'haut'].
```

b) Boucles

Les boucles whileTrue:, whileFalse, whileFalse: sont réalisées par envoi d'un message à un bloc qui représente la condition d'itération.

Les boucles sur des intervalles sont réalisées par l'envoi du message to:by:do: à un Number. Le bloc argument doit posséder 1 argument : l'indice qui prend les valeurs successives.

Exercice 19 Évaluez l'expression suivante :

```
1 to: 5 do:[:i | Transcript show: i ;cr]
```

c) Itérations

Une autre forme de contrôle est offerte par des itérateurs. Les itérateurs sont des messages envoyés à des objets appelés collections tels que les listes, les tableaux, les dictionnaires, les ensembles que nous détaillerons au paragraphe VII .

Ces itérateurs évaluent un bloc avec un paramètre (parfois deux) pour chacun des objets de la collection.

Exercice 20 Évaluez en inspectant les expressions suivantes :

1.#(1 2 3 4) do:[:i | Transcript show: i printString; cr]

2.#(1 2 3 4) collect: [:n | n@n]

3.(Set with: 1 with: 2 with: 3 with: 4) collect: [:n | n@n]

4.#(1 2 3 4) select:[:n | n even]

5.#(1 2 3 4) reject:[:n | n even]

6.'b??l?an???c' reject:[:carac | carac = $?]

7.'fin.' detect:[:c | c = $.]

8.'fin.' detect:[:c | c = $!] ifNone:[]

9.#(1 2 3 4) inject: 0 into:[:t :i | t + i]

Indications ... L'itérateur collecté retourne un objet de la même nature que le receveur. Regardez sa mise en oeuvre et appréciez l'usage de la méthode species.

E. Terminaison

Pour terminer l'évaluation d'une méthode, on utilise l'accent circonflexe suivi de l'objet retourné, qui peut, bien entendu être le résultat d'une évaluation.

V. Naviguer dans le système

Chercher un objet
Chercher une méthode
L'organisation du système : Le Browser
    Categories
    Classes

Bouton class-instance
Protocols
Méthodes
L'éditeur de texte

A. Chercher un objet

Pour chercher la signification d'un objet dans un code (une méthode) il suffit bien souvent de sélectionner l'objet puis dans le menu local l'option explain. Smalltalk indique alors la nature de l'objet et une façon d'obtenir plus d'informations sur cet objet.

B. Chercher une méthode

Comme vu plus haut, pour les méthodes, le moyen le plus simple est d'utiliser le Browser pour chercher les implementors, les senders, ou les implementors des messages.

C. L'organisation du système : Le Browser

Comme tous les outils Smalltalk, les Browsers utilisent les principes d'interaction classiques : choix d'un objet puis envoi d'un message sélectionné par l'intermédiaire d'un menu local à cet objet.

Dans les Browsers voici quelques options générales des menus locaux :

file out
pour écrire le source de l'objet sélectionné dans un fichier externe au format texte (.st) ou html (.html).
spawn
pour ouvrir un nouveau browser sur l'objet sélectionné.

Categories
Classes
Bouton class-instance
Protocols

Méthodes
L'éditeur de texte

a) Categories

Les catégories sont des ensembles de classes. On peut toutes les éditer en sélectionnant dans le menu local des catégories du Browser l'option browse all. Il est possible, mais non recommandé, de changer cette organisation puis de l'accepter.

Attention, supprimer une catégorie supprime toutes les classes qu'elle contient.

b) Classes

Après avoir sélectionné une classe, le menu offre, entre autres, les options suivantes :

definition
pour voir la définition de la classe (superclasse, variables d'instances, ...)

comment
pour voir le commentaire de la classe.

hierarchy
pour voir la hiérarchie de la classe.

inst var refs
pour voir les variables d'instances et les méthodes qui utilisent directement ces variables d'instances.

class var refs
pour voir les variables de classe et les méthodes qui utilisent directement ces variables de classe.

class refs
pour voir les méthodes qui envoient des messages à la classe.

Le commentaire devrait contenir une explication du rôle de la classe, de la composition de ses instances, des messages publics qu'elle offre, et des messages qui doivent être surchargés dans le cas d'une classe abstraite.

Les méthodes qui utilisent directement les variables d'instances ne devraient être que les méthodes du protocole *accessing*. Toutes les autres références devant se faire par envoi de message (le message d'accès à la variable d'instance) à self (l'objet lui-même.)

c) Bouton class-instance

Les boutons instance et class qui se trouvent sous la liste des classes permettent de voir :

1. les méthodes d'instance
2. les méthodes de classe

Les premières indiquent comment les instances d'une classe répondent aux messages. Les secondes sont des messages envoyés aux classes elle-mêmes, elles servent à la création d'instance, à l'initialisation des classes, ...

Par exemple, dans Point allInstances, allInstances est une méthode de classe, puisque c'est une classe qui reçoit le message, tandis que dans (1@3) x, x est une méthode d'instance.

d) Protocoles

Les protocoles sont des ensembles disjoints de méthodes.

Des protocoles de message de classe sont, par exemple :

instance creation,
class initialization,
constant access,
private,
examples,
documentation

Des protocoles de messages d'instance sont du style :

accessing,
private,
initialize-release,
testing,
printing,
copying,
converting,

comparing,
displaying,
enumerating,
updating,
adding,
menu messages,
controller access,
error handling,
model access

Il est possible de dresser la liste des méthodes de la classe pour rechercher une méthode particulière (find method ...)

Attention, supprimer un protocole supprime toutes les méthodes qu'il contient.

e) Méthodes

Pour chercher les implementors, senders, messages utiliser l'option correspondante du menu local.

f) L'éditeur de texte

Cette fenêtre contient, selon l'action précédente (non exhaustif):

la définition de la classe
le commentaire d'une classe
la hiérarchie - non modifiable
la liste des catégories ou protocoles
le code d'une méthode

Chacune de ces informations (sauf la hiérarchie) est éditable et doit être validée (accept) pour prendre en compte la modification. On peut ainsi, rajouter, modifier la définition, le commentaire d'une classe, modifier le code d'une méthode.

VI. Les éditeurs de texte

La plupart des fenêtres sont des éditeurs de texte dans lesquels il est possible d'évaluer du Smalltalk. Seules les fenêtres d'édition de texte associées à un Browser permettent la programmation, c'est-à-dire l'enregistrement du code dans le système de classes Smalltalk. Ce sont :

1. le Browser (system, class, protocol, implementors, senders)
2. le Debugger

Menu
Polices

A. Menu

Les options du menu local sont les suivantes :

do again
--- recommence l'action précédente d'édition. Ainsi, copy suivi de again cherche l'occurrence suivante. De même, paste suivi de again cherche l'occurrence suivante du mot remplacé pour y placer ^ nouveau le nouveau mot.

undo
--- annule l'action précédente.

copy
--- positionne dans un "buffer" la chaîne sélectionnée.

cut
--- positionne dans un "buffer" la chaîne sélectionnée et l'efface du texte.

paste
--- colle la chaîne la plus récemment placée dans le buffer. <shift> paste permet de choisir parmi les 5 plus récentes chaînes.

do it

--- évalue.

print it
--- évalue et affiche le printString du résultat.

inspect
--- évalue et inspecte le résultat.

accept
--- valide l'édition.

cancel
--- annule l'édition.

show bytecodes
--- génère un fichier texte dans le répertoire courant, ^ partir du contenu de la fenêtre.

Auxquelles sont ajoutées (more...) pour la programmation :

explain
--- donne un texte expliquant la nature de l'objet sélectionné, dans le cas d'un texte compilé.

Et d'autres méthodes….

B. Polices

Les éditeurs de texte manipulent non pas des chaînes (String) mais des Text. Les textes sont des listes de paragraphes auxquels sont attachés des chaînes et des indicateurs de format.

Exercice 21 :
Recherchez la classe texte et son commentaire.

VII. Collections

Collection est une classe abstraite qui factorise le comportement de nombreuses sous-classes. Les collections sont des objets regroupant des objets. Les différentes sous-classes sont caractérisées par les manières d'accéder aux objets et d'en ajouter.

Les plus utilisées des sous-classes sont : Bag, Array, OrderedCollection, SortedCollection, String, Text, Set, Dictionary.

Certaines classes sont indexées, c'est-à-dire que les objets que contiennent leurs instances sont accessibles par un index entier ; c'est le cas des sous classes de SequenceableCollection, OrderedCollection, String, Text. Ces classes permettent l'utilisation des méthodes at: et at:put pour, respectivement, accéder à un objet et mettre à jour un objet à l'indice passé en paramètre du at:.

Les classes non indexées utilisent les méthodes add:, remove:, includes: pour, respectivement, ajouter, supprimer un élément et tester l'appartenance d'un objet.

Sur toutes les collections il est possible d'utiliser les itérateurs do:, select:, collect:, reject:, detect:.

Toute collection sait donner sa taille, son nombre d'éléments, avec le message size.

Les collections dont la taille peut varier (OrderedCollection, Set, Bag, String, ...) peuvent se concaténer grâce au message binaire virgule (,).

Exercice 22 Cherchez les implementors de (virgule)

---

* Les classes
    - o Array
    - o Set
    - o OrderedCollection
    - o Dictionary
    - o String
    - o Bag
    - o SortedCollection
* Les méthodes

A. Les classes

Voir en annexe la liste de toutes les classes Smalltalk

Array
Set
OrderedCollection

Dictionary
String
Bag
SortedCollection

a) Array

Les tableaux sont des objets contenant des objets accessibles par un index.
Les méthodes de base d'utilisation des tableaux Smalltalk sont :

* new: unEntier, pour créer un tableau de taille unEntier.
* at: unEntier, pour accéder au unEntier élément.
* at: unEntier put: unObjet, pour changer le unEntier élément par l'objet unObjet.
* et les itérateurs do:, select:, collect:, reject:, detect:

Un tableau n'est évidemment pas typé et peut contenir toutes sortes d'objets.

Exercice 23 Evaluez les expressions suivantes :

1. MonTableau := Array new: 10.
2. MonTableau at: 2.
3. MonTableau at: 2 put: 3.
4. MonTableau at: 2 put: 3; at: 3 put: 8; at: 8 put: 'huit'.
5. MonTableau inspect.
6. MonTableau printString.
7. MonTableau at: (MonTableau at: (MonTableau at: 2))
8. MonTableau at: '1'. "Erreur"
9. MonTableau at: 11. "Erreur"
10. MonTableau select:[:elem | elem isNil not]
11. MonTableau size

Indications ... Notez qu'à la création avec le message new: toutes les composantes sont nil. Notez aussi l'importance des parenthèses dans l'expression 7. Si vous les omettez, Smalltalk cherche le message at:at:at: qui n'existe pas. Cette erreur est assez fréquente lors de tests avec des méthodes dont le sélecteur est un keywords. Par exemple, unEnsemble includes: unÉlement ifTrue:["faire quelque chose"] est incorrecte, car le message includes:ifTrue: n'est pas défini.

b) Set

Un ensemble est une collection d'objets non dupliqués, non accessibles par un indice. Les méthodes de base d'utilisation des ensembles Smalltalk sont :

* new, pour créer un ensemble vide.
* isEmpty, pour tester la vacuité de l'ensemble.
* add: unObjet, pour ajouter unObjet à l'ensemble, en vérifiant de ne pas le dupliquer.
* addAll: uneCollection, pour ajouter dans l'ensemble tous les éléments de la collection, en supprimant les doublons.
* remove: unObjet, pour supprimer l'objet unObjet.
* remove: unObjet ifAbsent: unBloc, pour supprimer unObjet, mais prévoir un traitement exceptionnel si cet objet n'est pas dans l'ensemble.
* includes:, teste l'appartenance d'un objet.
* et les itérateurs do:, select:, collect:, reject:
* detect: unBloc, pour chercher un élément vérifiant un critère défini par unBloc.
* detect: unBloc ifNone: bloc, pour chercher un élément vérifiant un critère défini par unBloc.
* size, donne le cardinal de l'ensemble.

Un ensemble n'est évidemment pas typé et peut contenir toutes sortes d'objets.

Exercice 24 Evaluez les expressions suivantes :

1. MonEnsemble := Set new.
2. MonEnsemble isEmpty.
3. MonEnsemble add: 1.
4. MonEnsemble add: 2; add: 3; add: 3; add: 'foo'; add: 3.
5. MonEnsemble inspect.

6. MonEnsemble size.

7. MonEnsemble printString.

8. MonEnsemble remove: 1

9. MonEnsemble remove: 8 ifAbsent:[Transcript show: 'opps'; cr]

10. MonEnsemble includes: 'foo'.

11. MonEnsemble collect:[:elem | elem printString]

12. MonEnsemble , (Set with: $3 with: 3 with: 'bar')

c) OrderedCollection

Les OrderedCollection sont des listes d'objets indexés. Elles combinent les méthodes d'accès des tableaux (at:, at:put: et des ensembles add:, addAll: remove:, remove:ifAbsent:, includes:. Une orderedCollection connaît aussi les messages size, , (virgule), et les itérateurs do:, select:, collect:, reject:, detect:).

Les objets contenus dans une OrderedCollection peuvent être dupliqués (apparaître à deux index différents).

Les méthodes spécifiques sont :

addFirst: unObjet
addLast: unObjet
first, last

d) Dictionary

Un Dictionary est un ensemble d'associations. Une Association est un couple (key,valeur) qui est construit par le message <- .

Exercice 25 Cherchez les implementors de <- et inspectez 1 -> 'UN'.

Un dictionnaire étant un Set, il connaît les messages add:, remove:, ...avec la contrainte que les éléments doivent être des Associations.

Attention, les itérateurs do:, select:, collect:, reject:, detect: itèrent sur les valeurs.

Les messages spécifiques des dictionnaires sont :

* values qui retourne le Bag des values.
* keys qui retourne le Set des keys.
* keysDo:, qui permet d'itérer sur les clefs.
* keysAndValuesDo: unBlocADeuxArguments, qui permet d'itérer sur le couple clef-valeur.
* at: aKey, retourne la valeur associée ^ la clef.
* at: aKey ifAbsent: aBlock, retourne la valeur associée ^ la clef et évalue le bloc si la clef n'existe pas.
* at: aKey put: aValue

De nombreux dictionnaires existent dans Smalltalk. Inspectez Dictionary allInstances pour le constater. On trouve Smalltalk, Undeclared, TextConstants, ...

e) String

Les String sont des tableaux de caractères qui peuvent s'agrandir. Un string peut répondre aux méthodes , size, ... et aux itérateurs

La comparaison de chaînes en ignorant les majuscules se réalise par =. La comparaison de chaînes en tenant compte des majuscules se réalise par trueCompare:. La méthode match: permet de réaliser du pattern matching avec les deux symboles spéciaux # pour un caractère quelconque et * pour 0 ou N caractères. Ainsi 'Mme #.*' match: 'Mme E.Peel' est true.

f) Bag

Les Bag sont des ensembles d'objets où la duplication est permise. On trouve les mêmes méthodes que dans Set.

g) SortedCollection

Les SortedCollection sont des OrderedCollection dont les éléments sont triés. Le critère de tri est un bloc à deux arguments que l'on définit à l'aide du message sortBlock: unBloc.

Des exemples de sortBlock sont : [:a :b | a >= b], ou [:x :y | x < y].

B. Les méthodes

Ne sont citées que les méthodes principales des classes principales

Object ()
  Collection ()
   do:, select:, reject:, collect:, detect:, detect:ifNone:, isEmpty, size, add:, remove:, addAll:, removeAll, includes:
     Bag ('contents')
    SequenceableCollection ()
     at:, at:put:, indexOf:, , (virgule)
      ArrayedCollection ()
        Array ()
        CharacterArray ()
          String ()
            Symbol ()
          Text ('string' 'runs')
      Interval ('start' 'stop' 'step')
      OrderedCollection ('firstIndex' 'lastIndex')
       addFirst:, addLast:, removeFirst, first, last
        SortedCollection ('sortBlock')
    Set ('tally')
      Dictionary ()
       values:, keys:, at:, at:ifAbsent:, at:put:, valuesDo:

VIII. Programmer en Smalltalk

Exercice 26 Tout au long de cette section consacrée à la programmation, vous vous efforcerez de mettre en place une classe permettant de réaliser du calcul sur les nombres complexes. La spécification en est la suivante :

un complexe peut rendre sa partie réelle et imaginaire real, imaginary.
un complexe peut s'ajouter à un nombre (respectivement multiplier, soustraire, diviser)
un complexe s'affiche sous la forme a, bi, a+bi ou a-bi

un complexe doit pouvoir tester son égalité avec un autre complexe
un complexe peut calculer son module
une opération complexe dont la partie imaginaire du résultat est nulle retourne un Number

Quelques suggestions... On testera les fonctionnalisés de la classe Complex créée à l'aide des expressions suivantes :

1. Création d'un complexe à partir de 6 i: 2
2. Impression du complexe 4 i: -2 sous la forme 4-2i
3. Accès à la partie réelle (-1 i: 0) realPart
4. Accès à la partie réelle 4 realPart
5. Accès à la partie imaginaire (-1 i: 1) imPart
6. Accès à la partie imaginaire 8 imPart
7. égalité de deux complexes, d'un complexe et d'un réel, ...
8. Addition de deux complexes (1 i: -1) + (-1 i: 1)
9. Multiplication, Division, Soustraction, Module
10. Addition d'un entier (0 i: 1) + 3
11. Addition à un entier 3 + (0 i: 1)
12. (0 i: -1) + (0 i: 1) retourne 0 (SmallInteger)

Un complexe est une valeur arithmétique, presque un point.

Point subclass: #Complex ...

Complex hérite des variables d'instances 'x y' de point, que l'on interprètera en tant que parties réelle et imaginaire.

Ajoutez les protocoles arithmetics, accessing, printing, ... Gardez sous les yeux un browser sur Point, Number pour vous en inspirer. Programmer objet, c'est faire de la programmation par imitation.

La programmation se fait au travers de toutes les fenêtres d'édition de code Smalltalk : Les Browsers du système, des classes de la hiérarchie, des implementors, et le debogueur.

Créer une classe
Documenter une classe
Ajouter des méthodes à une classe
    new
    initialize

protocole accessing
printOn:, printString
= , == et ~=, ~~
Réutilisation
Commentaires sur la programmation orientée objet
Conseils pour la programmation en Smalltalk

A. Créer une classe

Pour créer une classe, il faut :

* Choisir ou créer une catégorie
* Choisir la superclasse
* Choisir la forme des instances (les variables d'instance)
* Choisir les variables de classes (objets connus par toutes les instances)
* Choisir un poolDictionary (ensemble de dictionnaires partagés par plusieurs classes et accessible simplement par les instances)

Ajoutez une catégorie Number-Extension, un schéma de fabrication de classe doit apparaître. Modifier le pour l'adapter à vos besoins, puis accepter. La nouvelle classe doit être créée.

B. Documenter une classe

Pour documenter une classe, on sélectionne dans le menu local des classes, quand la classe à documenter est sélectionnée, l'option comment. Un commentaire par défaut apparaît qui indique ce que l'on attend du commentaire. Modifiez ce texte puis acceptez, la classe est commentée.

Le commentaire devrait contenir, en anglais, si possible :

1. Une description de la classe : son intention, son utilité
2. Une description des variables d'instance
3. Une description des méthodes qui doivent être surchargées
4. Toute information digne d'être citée (trucs, ...)

C. Ajouter des méthodes à une classe

Pour ajouter une méthode, il faut d'abord ajouter ou sélectionner un protocole. Un schéma de création de méthode apparaît, qu'il faut adapter, puis accepter.

L'acceptation déclenche la compilation. Le compilateur vérifie alors la déclaration des objets et des messages.

S'il ne reconnaît pas une variable, un menu propose de définir la variable comme temporaire, undeclared, class var ou global, ou bien propose d'essayer de la corriger.

S'il ne reconnaît pas le message, un menu propose de faire comme si le message existait, et ce sera à vous de le définir ultérieurement, sinon une erreur risque de se produire.

Une fois acceptée, la méthode apparaît dans la liste du protocole sélectionné. Il est possible de la modifier en l'éditant et en la re-acceptant.

---

* new
* initialize
* protocole accessing
* printOn:, printString
* = , == et ~=, ~~

a) new

Ce message sert à créer de nouvelles instances d'une classe. Il est envoyé à une classe, c'est donc une méthode de classe. Certaines classes ne peuvent pas créer des instances (Integer, Character, ...). L'implantation classique de cette méthode est :

```
new
  super new initialize
```

Ceci signifie que pour créer une nouvelle instance, il faut s'y prendre comme sa super classe (super new) puis initialiser (initialize).

Remarquez que si la super classe exécute elle-même super new initialize, le message initialize de la classe du receveur sera exécuté deux fois.

b) initialize

Cette méthode permet de mettre en place certaines variables d'instances dont la forme doit être un objet particulier comme un dictionnaire, une collection ordonnée, un objet par défaut. On trouve, par exemple, l'implantation suivante :

```
initialize
   varInst := OrderedCollection new. "varInst est une liste vide"
   varInst2 := Dictionary new. "varInst2 est un dictionnaire vide"
```

Ce sélecteur est aussi utilisé pour les classes. Il permet d'initialiser des variables de classe. Les variables de classe sont des objets appartenant à une classe qui sont visibles (accessibles) par toutes ses instances et par toutes les instances de ses sous-classes. Lors d'un File In, le message initialize est automatiquement envoyé aux classes qui savent y répondre.

c) protocole accessing

Le protocole accessing regroupe toutes les méthodes qui permettent d'accéder aux variables d'instance. Normalement, ces méthodes doivent être les seules à permettre d'accéder directement à ces variables d'instances. Toutes les autres méthodes, si elle veulent accéder aux objets contenus dans les variables d'instances, doivent passer par les méthodes du protocole accessing.

L'implantation de ces méthodes a la forme **générale** :

```
varInst
   "Retourne la variable d'instance varInst"
   ^varInst
```

Et pour la mise à jour des variables d'instance :

```
varInst: anObject
   "Met ^ jour la variable d'instance varInst"
   varInst := anObject
```

Les accès à varInst seront effectués ultérieurement par l'intermédiaire du message varInst. L'objet lui-même fera référence à ces variables d'instance par l'évaluation de self varInst.

L'accès aux variables d'instance peut être direct, comme dans la méthode d'accès, plutôt que par un envoi de message. Cependant une telle technique, si elle est plus efficace de quelques microsecondes et constitue donc, dans certains cas, une optimisation utile, diminue la réutilisabilité de la classe dans le cas où cette dernière est raffinée par une sous-classe. En effet, en utilisant un accès par envoi de message, il sera possible de modifier simplement la méthode, en surchargeant la méthode accès, pour altérer le comportement alors que par accès direct, le comportement consistera toujours à accéder à la variable d'instance, ce qui, pour certaines applications, peut vouloir être modifié. Dans un souci de réutilisabilité, on limitera donc au maximum les accès directs aux variables d'instance.

d) printOn:, printString

La méthode printOn: est la méthode utilisée par défaut par le système pour imprimer un objet sous la forme d'une chaîne. Pour apprendre à un objet à s'imprimer, il faut donc surcharger printOn:.

Exercice 27 Cherchez les différentes façons de s'imprimer dans le système.
Modifier l'impression des caractères pour que s'affiche : Char = < le caractère >.

e) = , == et ~=, ~~

Il y a deux tests d'égalité en Smalltalk, différenciés dans cette version de squeak. Le test d'égalité d'objet et le test d'égalité de structure.

==
teste si le receveur et le paramètre correspondent au même objet. Ce message ne doit jamais être surchargé.
=
teste si la structure des deux objets est identique ou non. Ce message peut être surchargé (redéfini) pour expliciter la notion d'égalité de deux instances.

Par exemple, (2@2) = (2@2) rend true tandis que (2@2) == (2@2) retourne false. Dans le premier cas, l'égalité entre Point est définie par l'égalité des coordonnées. Mais les deux points correspondent ^ deux instances distinctes de la classe Point. On peut, en effet, changer une coordonnée de l'un sans altérer l'autre.

Les messages ~= et ~~ correspondent respectivement ^ = not et == not.

D. Commentaires sur la programmation orientée objet

Les difficultés principales rencontrées pour programmer sont de :

1. trouver les classes,
2. trouver la position des classes,
3. trouver les noms des méthodes (conna”tre le système),
4. faire le lien avec de potentielles classes utilisatrices (ex : Number et Complex)

Ajouter des classes réutilisables, des classes de base, des structures de données est plus complexe que de définir une application ou une interface. Dans le dernier cas, les effets de bord, les connexions avec d'autres classes utilisatrices sont plus facilement cernables.

Le bénéfice de la réutilisation ne se fait pas sans coût. Bien programmer objet, réutilisable, est difficile. Il faut toujours une bonne connaissance de l'existant.

Les outils de navigation que nous avons déj^ découverts simplifient la recherche dans le système et la compréhension.

E. Conseils pour la programmation en Smalltalk

1. Commentez avec soin toutes les classes, et les méthodes qui le méritent !
   - o Un commentaire de classe doit préciser pourquoi la classe existe, ce qu'elle offre comme services, et pourquoi il est utile de créer ses instances. C'est le minimum vital.
   - o Un commentaire de méthode doit préciser le rôle de la méthode et des paramètres. Point n'est besoin de détailler l'algorithme, pour cela on lit le code ; il faut décrire le quoi pas le comment !
2. Utilisez un protocole private pour grouper les méthodes à accès limité.
3. Accédez et modifiez les variables d'instance par l'intermédiaire des méthodes. N'utilisez pas de nom commençant par "set" ou "get" pour ces méthodes, mais le nom de l'attribut directement (sauf peut-être pour des méthodes privées).
4. Précisez les effets de bords, modification d'objet passés en paramètre,... Le mieux est d'adopter une politique globale, et de mettre en évidence les exceptions...
5. Choisissez bien le nom des méthodes. En général, on distingue deux types de méthode :
   - o Celles qui retournent une valeur,

- Celles qui modifient un objet

On choisira pour les premières des noms comme sélecteur et pour les secondes des verbes.

6. Evitez les abréviations dans les noms des messages, des classes ; seule exception, les variables temporaires.
7. Evitez les tests explicites à une classe, préférez le double-dispatching. Cette technique consiste ^ échanger récepteur et paramètre pour tenir compte de la classe des deux objets lors de l'application d'une méthode. Le système fournit des exemples avec l'addition des "Number". #+ devient selon le récepteur, sumFromInteger:, sumFromFloat: ou sumFromFraction:.
8. Faites relire votre code.
9. Quelques "erreurs" fréquentes
   - Quand vous redéfinissez #=, revoyez #hash. Deux objets égaux (#=) ont le même #hash.
   - #add: retourne l'objet ajouté, pas la collection.
   - nil n'est pas un récepteur acceptable pour #ifTrue: ou #ifFalse:
   - Le récepteur de #whileTrue ou #whileFalse est un BLOC.

IX. Déverminage

Pour la mise au point et la correction d'erreur, Smalltalk dispose d'un outil très performant : le debugger. Pour le faire apparaître, plusieurs solutions s'offrent :

1. Une erreur est détectée par Smalltalk.
   message non compris - un message a été envoyé à un objet qui ne savait pas y répondre.
   accès à un index incorrect dans une collection indexée
   ...
2. vous avez programmé un point d'arrêt à l'aide des messages halt ou halt: aString. Ces messages sont compris par tous les objets, il suffit donc d'évaluer self halt pour programmer un point d'arrêt.

3. vous appuyez sur les touches Ctrl et c.

Dans ces trois cas, une fenêtre s'ouvre. Pour ouvrir le debugger, il suffit de sélectionner dans le menu local de cette fenêtre l'option debug. Pour continuer l'évaluation, il aurait fallu choisir proceed.

Attention : le positionnement de point d'arrêt ne peut se faire n'importe où. En effet, si un point d'arrêt est placé dans une méthode très générale comme l'addition de points, puisque cette méthode est employée systématiquement lors de l'ouverture de fenêtre, le point d'arrêt ne pourra pas permettre l'ouverture de la fenêtre ... Pour s'en sortir dans ce cas, il ne reste éventuellement que le kill -9 d'unix. C'est le prix de l'ouverture du système, on peut faire des erreurs ! On pourra toutefois récupérer le travail fait jusqu'à ce point, grâce aux Change list.

Le debugger est composé de 4 parties que nous allons présenter.

La pile d'exécution
L'environnement d'exécution
La méthode interrompue
step et send
La correction interactive

A. La pile d'exécution

La pile d'exécution est la liste qui apparaît en haut de la fenêtre de debugage. Cette liste contient l'ensemble des messages en cours d'exécution. Le plus haut est celui qui a permis d'ouvrir la fenêtre, le précédent, celui qui, en général, a déclenché le débugage. Chaque ligne a la forme :

Classe du receveur (Classe qui répond)>>nom de message

La méthode correspondant au nom du message donné pour le receveur a été trouvée dans la classe qui répond.

En sélectionnant une de ces lignes, on fait apparaître :

1. Le code source de la méthode, --- la sous-fenêtre centrale
2. Un inspecteur sur le receveur, --- en bas à gauche
3. Un inspecteur sur le contexte d'exécution. --- en bas à droite

B. L'environnement d'exécution

L'environnement d'exécution apparaît en bas à droite. Une première sous-fenêtre contient la liste des objets définissant le contexte :

1. Les variables temporaires de la méthode
2. Les paramètres du message

En sélectionnant l'un de ces objets, il est possible de voir et/ou d'éditer leur valeur.

C. La méthode interrompue

Le code source de la méthode en cours d'exécution est représenté, ainsi que la position courante de l'exécution : le message qui va être envoyé est indiqué en inverse vidéo.

D. step et send

Pour contrôler l'exécution, cinq options sont disponibles par l'intermédiaire du menu local de la pile des messages ou par les deux boutons step et send :

1.proceed --- continue l'exécution
2.restart --- reprend l'exécution de la méthode courante, utile si le contexte ou le receveur ont été modifié manuellement
3.step --- avance d'un envoi de message dans la méthode courante
5.send --- envoie le message en empilant la méthode courante, et en présentant la nouvelle méthode exécutée

E. La correction interactive

Il est possible de modifier le code même des méthodes lorsque l'exécution est arrêtée, ainsi que leur contexte d'exécution.

X. Sauvegarder et restaurer des modifications

La plupart de vos actions sont enregistrées dans le fichier .changes associé à votre image. Ce journal permet, par l'intermédiaire des outils que nous allons maintenant présenter, de conserver et de retrouver les versions précédentes des classes et méthodes que vous développez.

Une autre technique permet de sauvegarder au coup par coup des méthodes, des protocoles, des classes ou des catégories ; ce sont les File out des menus des différents Browsers. Associée au File in, elle permet de transférer du code source Smalltalk d'une image dans une autre. C'est cette technique qui a déjà été utilisée pour intégrer l'application FinancialHistory.

La dernière technique offerte par Smalltalk est la gestion des Projects. Un projet permet de gérer l'ensemble des modifications apportées au système dans un objet spécial : un Change Set.

Changes
File list
Project

A. Changes

L'option Changes du Launcher permet selon la sélection faite :

recent change log
ouvre une liste des derniers .changes.

browse changes methods
ouvre un inspecteur sur le change set.

file out changes
sort par file out toutes les modifications mémorisées dans le change set : classes ajoutées ou redéfinies, méthodes ajoutées ou modifiées, réorganisation des catégories et des protocoles, suppression des classes ou des méthodes.

browse recent submissions
permet de voir les différentes méthodes proposées

B. File list

Cet outil permet d'éditer et de charger par file in un fichier. Il se décompose en trois sous-vues :

1. Vue haut gauche :

Chemin depuis la racine du disque

2.Vue haut droite

Fichiers correspondant au chemin parcouru décrit dans la vue de gauche

3.la dernière vue présente soit des informations concernant le fichier (taille, date de mise-à-jour,...), soit un éditeur sur le fichier. Si ce fichier est un fichier contenant du source Smalltalk, le menu local de cette fenêtre permet de charger une partie du fichier par l'option file it in.

C. Project

Un projet permet de gérer des change set indépendants des autres projets. Ceci permet de faciliter la création de fichier .st concernant un unique projet. En effet, comme vous avez pu le constater, la programmation objet (d'autant plus à la Smalltalk) incite à la création de nombreux petits morceaux de codes épars. Pour garder la cohérence d'une modification, il est utile de disposer de la notion de projet, pour garder la trace des altérations que le système a subit pour la réalisation d'un projet.
Pour créer un projet, choisissez l'option Open project du Launcher.

Pour sortir d'un projet, choisissez l'option Exit project du Launcher.