1

Les objets de Smalltalk-80

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

1.1 Environnement Smalltalk-80

1.1.1 Notion de machine virtuelle

Smalltalk est non seulement un langage de programmation mais aussi un système d'exploitation, un environnement de développement et une bibliothèque de code très importante.
Comme Smalltalk est un interpreteur, il a besoin de conserver le texte initial (fichier *source*) et sa forme interne (fichier *image* - format byte code). Les modifications effectuées sont enregistrées dans un fichier *change*. Le représentation interne générée peut être vue comme un langage d'assemblage pour une machine virtuelle.

1.1.2 Accès à l'image et au binaire

L'image et le binaire se trouve :

```
C:
   vwnc30
       vwnc30
           bin
           image
```

Pour lancer une image, cliquer 2 fois sur l'icône de visualnc.im dans le repertoire `image`

1.2 Organisation du travail

Utilisation des boutons de la souris :

bouton de gauche : sélection de zone, activation de fenêtre

bouton de droite : pop-up menu (menu contextuel associé à la fenêtre

bouton de droite dans la barre de titre : menu d'édition de la fenêtre

Vous utiliserez le *FileList* se trouvant dans l'icône bleue représentant un tiroir pour créer des nouveaux fichiers (créer a:TP1.txt) dans le répertoire Smalltalk.

Ecrire le texte du fichier (les exercices commentés) dans la fenêtre du bas et sauvegarder en utilisant le menu contextuel ("Pop Up Menu"- bouton milieu de la souris- et *Save as*) de la fenêtre.
Toute sauvegarde doit être faite sur votre disquette, soit sous a :

1.3 TP: Observation des objets et règles de priorité

Evaluer : selectionner une zone et faire “print it”
Inspecter : selectionner une zone et faire “inspect”

- Inspecter les expressions suivantes :

```
1
2.0
$a
'une chaine'
1@2
1.0@2.0
7/2
```

Parmi les messages, on distingue

les messages unaires comme new, sin, sqrt, size, first, last, negated)

les messages binaires `+ - * / ** // \\ < <= > >= = ~= == ~~ & | @ ,`

les message à mot clé comme at: put:, x: y:, bitOr:, bitAnd:

Dans une expression, on évalue en priorité en respectant le parenthésage, les messages unaires puis binaires puis à mots clés. Si l’expression ne comporte que des messages de même priorité, l’évaluation se fait classiquement de la gauche vers la droite.

- Evaluer et inspecter les expressions suivantes :

```
7.0/2.0
1 + 1
(1 + 1) printString
(1/2) class
```

Expliquer pourquoi le parenthésage est obligatoire.

- Pas de priorité des opérateurs, l’évaluation suit l’ordre des messages. Evaluer :

```
2 + 3*4
2 + (3*4)
2  + 1/2
2 + (1/2)
```

- Uniformité des messages. Un même message peut être adressé à des objets de types différents. Evaluer et inspecter :

```
2 sqrt
2.0 sqrt
(3/2) sqrt
(3/5) + (6/7)
```

- Arithmétique exacte et conversion de type. Evaluer :

```
(11111111111111111111/11111111111111111112) + (1/11111111111111111112)
2r1000
16rFF
256 printStringRadix:2

(8 bitOr: 1) printStringRadix:2
(8 bitAnd: 9)
2e10 asInteger
```

Faire le ou bit à bit des nombres binaires 1010 et 0011 et donner le résultat en binaire.

- Les variables. Evaluer :

```
| x |
x := 2.

| x |
x := 2 + 1/2.

| x |
x := 2 + 1/2.
x := 2.
```

Quelle est la valeur de x après évaluation de la dernière portion de code?

- Le tableau est aussi un objet. Evaluer :

```
| table  |
table := #(1 2.0 'trois' 44444444444444444444444444444444444444).
table := #(1 3 6 9).
table first.
table last.
table reverse.

| string |
string := '#(1 3 6 9)'.
string first.
string last.
string reverse.

#(10 20 30 40) at: 2

| table |
table := #(1 3 6 9).
table at:1 put: (3/4)

| string |
string := '#(1 3 6 9)'.
string at: 4 put: $r
```

- Donner la liste des objets et des messages définis dans les expressions ci-dessous. Expliquer le résultat de l'évaluation.

```
| aPoint |
```

```
aPoint:= Point x:2 y:1.
aPoint x: aPoint x * 2

| x |
x:=1.5.
x negated rounded.
Fraction  numerator: x*2 denominator: 3 + x negated rounded.
```

1.4 Exercices (A faire en TD)

1.4.1 Tableaux

1. Multiplier par 2 le 2 ème élément d'un tableau,
2. Remplacer la valeur du 2 ème élément d'un tableau par son opposé.
3. Remplacer la valeur du 3 ème élément par la valeur du 2 ème élément.
4. Remplacer la valeur du 3 ème élément par la somme des 2 ème et 3ème (ancienne valeur) éléments.
5. Le 2 ème du tableau étant une fraction, remplacer cette fraction par la fraction inverse dans le tableau.

1.4.2 Nombres

Maximum

La méthode `max: unAutreNombre` appliqué à un nombre renvoie le plus grand des deux nombres.
Exemple : 2 max: 6 renvoie 6.

1. Calculer le maximum de 3 variables a b c contenant des valeurs quelconques.
2. Calculer le maximum de 3 variables a b c contenant des valeurs quelconques **sans utiliser de variables intermédiaires**

Fonctions trigonométriques

Un nombre (en radians) comprend les messages correspondant aux fonctions trigonométriques sin cos tan arcSin arcCos arcTan.

On convertit un nombre de Degré à Radian en lui envoyant le message degreesToRadians.

Calculer (à l'aide d'une fonction trigonométrique) le côté d'un carré dont la diagonale mesure 1.41421 mètres.

Conversion Celsius-Fahrenheit

La formule de conversion Celsius-Fahrenheit est : C = (5/9) (F - 32).
Convertir une variable contenant un nombre (en degrés Fahrenheit), en degrés Celsius.

Conversion binaire-hexadécimale

`//` est l'opérateur de division entière.
`\\` est l'opérateur de modulo.
Soit un nombre hexadécimal (de 0 à 15 en base 10, de 0000 à 1111 en base 2), on désire le convertir en binaire (sans utiliser printStringRadix:).

En effectuant une série de divisions entières, les chiffres binaires sont obtenus **de la droite vers la gauche** grâce au reste de la division entière (le modulo).

1. Appliquer cet algorithme pour convertir le nombre hexa 15 en base 2.
2. Vérifier en utilisant printStringRadix:

1.4.3 Dates

On obtient la date du jour grâce au message `today` envoyé à la classe `Date`.

```
| d |
d := Date today.
```

On peut créer une date grâce au message newDay:unNumeroJour monthNumber:unNumeroMois year:unNumeroAnnee envoyé à la classe Date.

On peut aussi créer une date grâce au message fromDays:nombreJours envoyé à la classe Date.

La méthode asDays envoyé à une date renvoie le nombre de jours depuis le début du siècle (1/01/1901).

```
| d |
d := Date newDay:12 monthNumber:10 year:1998.
d asDays. "35713"
```

1. Calculer le nombre de jours que vous avez déja vécu.
2. Calculer quelle serait la date, si vous aviez l'age que vous avez aujourd'hui et si vous étiez né le 1 Janvier 1901.

1.4.4 Caractères

On peut créer un caractère à partir de son code ASCII en envoyant le message `value: unCodeAscii` à la classe `Character`

On obtient le code ASCII d'un caractère en lui envoyant le message asInteger

```
| c |
c := Character value: 65.
c asInteger. "65"
```

1. Calculer le code ASCII de $a et de $A
2. Convertir un caractère de minuscule à majuscule

1.4.5 Chaînes

On obtient une chaîne à partir d'un caractère en envoyant le message with: unCaractere à la classe String.

Le caractère "blanc" s'obtient en envoyant le message space à la classe Character.

On concatène des chaînes avec l'opérateur ,

```
| s1 s2 |
s1 := String with: $1.
s2 := String with: $2.
s1, s2        "12"
```

1. Fabriquer une chaîne contenant le caractère "blanc"
2. Fabriquer une chaîne contenant votre prénom, un caractère "blanc", votre nom

1.4.6 La classe Point

Dans la classe Point, les messages unaires r et theta permettent de recupérer les coordonnées polaires d'un point.

1. Définir une translation de point par rapport à un vecteur donné.
2. Définir une homothétie.
3. Calculer l'angle formé par deux vecteurs.