Retour sur quelques notions clés dans Smalltalk

Noury Bouraqadi
http://car.mines-douai.fr/noury

Option ISIC
Ecole des Mines de Douai

Retour sur la création et l'initialisation d'objets

Noury Bouraqadi - option ISIC - Dépt. I. A.

Création d'objets - 1

- Objets littéraux : écriture = création
  - Nombres : 2 , 3.14 , …
  - Lettres $a , $z , …
  - Chaînes de caractères :
    - 'Hello'
    - 'l''oiseau'
      - double apostrophe pour inclure une apostrophe dans la chaîne de caractères
  - Symboles : #toto #abcd , …
  - tableaux de littéraux (instances de Array):
    - #(2 $a 'Hello' #bidule)

Noury Bouraqadi - option ISIC - Dépt. I. A.

Création d'objets - 2

- Autres objets = Par envoi de message aux classes
  - Les classes sont des objets
- Message à la classe
  - Point new
  - OrderedCollection with: 2 with: $a with: 'Hello'
    - Crée une collection ordonnée avec les éléments :
      - 2
      - $a
      - 'Hello'
- Message à d'autres objets
  - 640 @ 480
    - Crée une instance de Point d'abscisse 640 et d'ordonnée 480
  - 'Hello' , ' World!'
    - Concaténation = Crée la chaîne de caractères : 'Hello World!'

Noury Bouraqadi - option ISIC - Dépt. I. A.

Création d'instances initialisées

- Tous les champs d'un nouvel objet référencent nil
  - Initialisation nécessaire au bon fonctionnement de l'objet
- Initialisation
  - avec des des valeurs par défaut :
    - de manière "**gloutonne**" (initialize)
    - de manière "**paresseuse**" (accesseurs)
  - Avec des valeurs fournies à la création
- Les différentes solutions peuvent être combinées

Initialisation de manière "gloutonne"

- Définir une méthode d'instance pour chaque valeur par défaut
  - Retourne la valeur par défaut d'un champ donné
- Définir la méthode d'instance **initialize**
  - Obtenir les valeurs par défaut
    - envois de messages à self
  - Affecter les valeurs par défaut aux champs

Exemple d'initialisation "gloutonne"

- Par défaut, les instances de Lampe doivent être
  - éteintes, et de couleur jaune
- Lampe>>estInitialementAllumee
  ^false
- Lampe>>couleurParDefaut
  ^Color yellow
- Lampe>>**initialize**
  super initialize. "obligatoire ! Explication dans l'héritage"
  estAllumee := self estInitialementAllumee.
  couleur := self couleurParDefaut
- Les nouvelles instances de Lampe sont alors :
  - éteintes,
  - de couleur jaune

Initialisation paresseuse

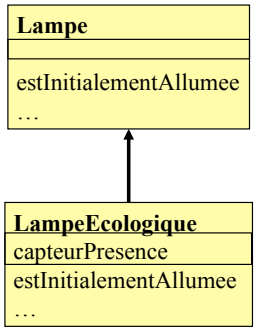
- Intérêt : un champ n'est initialisé que s'il est utilisé
- Initialisation réalisée dans les méthodes d'accès en lecture aux champs (les accesseurs en lecture)
  - Utilisation des accesseurs en écriture
- Exemple :
  - Lampe>>couleur
    couleur ifNil: [self couleur: self couleurParDefaut].
    ^couleur
  - Lampe>>estAllumee
    estAllumee ifNil: [self estAllumee: self estInitialementAllumee].
    ^estAllumee

Initialisation avec valeurs par défaut

- La valeur par défaut peut changer
  - Valeurs par défaut obtenues par un traitement
  - Redéfinition des méthodes avec les valeurs par défaut dans les sous-classes

```
Lampe>>estInitialementAllumee
    ^false

LampeEcologique>>estInitialiementAllumee
    ^Time now hours > 18 and: [
        capteurPresence detectePresence]
```



Initialisation avec valeurs par défaut

- initialize doit référencer la méthode de la superclasse (super)

```
Lampe >> initialize
    super initialize. "Emplacement selon besoins"
    self estAllumee: self estInitialementAllumee.
    self couleur: self couleurParDefaut

LampeEcologique >> initialize
    super initialize.
    capteurPresence := self classeCapteurPresence new

LampeEcologique>>classeCapteurPresence
    ^CapteurPresenceInfraRouge
```


Initialisation avec des valeurs fournies

- Le programmeur fourni les valeurs à l'instanciation
  - Les valeurs sont des arguments du message de création
  - Le message de création est envoyé à la classe
    - Exemple :

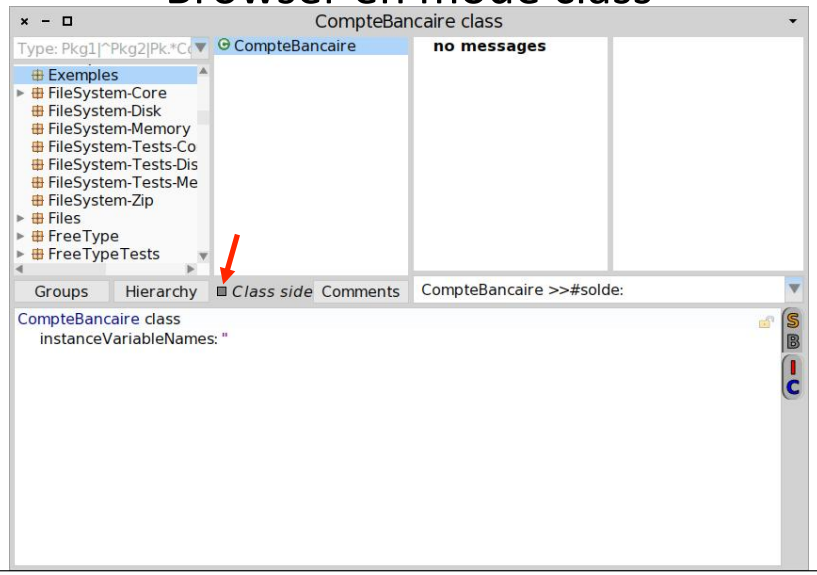
      Lampe lampeDeCouleur: Color red

- Besoin de méthodes de classes

Méthodes de classe

- Rappel : les classes Smalltalk sont aussi des objets
  - Peuvent recevoir des messages
- Méthodes de classe :
  - Sont définies en mode "class" du browser
  - Obéissent aux mêmes règles que les méthodes d'instance
    - Masquage, redéfinition, …
  - Exemple : **subclass:instanceVariableNames:…**
    - méthode de création des sous-classes

Browser en mode class



Exemple d'initialisation avec des valeurs fournies

```
Lampe class>>lampeDeCouleur: uneCouleur
    |uneLampe|
    uneLampe := self new.
    uneLampe couleur: uneCouleur.
    uneLampe allumer.
    ^uneLampe
```


Retour sur les Variables

Variables globales

- Utiliser avec beaucoup de modération
  - Visibles partout
    - Risque "d'effets de bord" indésirables
  - Les objets qu'elles référencent sont persistants
    - Ne sont pas détruits après usage
    - Risque de fuite mémoire (memory leak)
- Lecture/écriture comme n'importe quelle variable
- Déclaration en passant par le dictionnaire des variables globales
  - Clés == noms des variables == instances de Symbol
  - Accessible via la variable globale **Smalltalk**

Exemple de variables globales

- Déclaration
  - Smalltalk at: #Promotion3427 put: Set new
- Utilisation
  - Promotion3427 add: Eleve new
  - Promotion3427 size

Variables d'instance, de classe, de pool
Présentation

- Variables d'instance = champs
  - Privée à chaque instance
  - Accessibles dans les méthodes d'instances seulement
- Variables de classe
  - Partagées entre les instances de la classe et…
    avec les instances des sous-classes (héritage !)
  - Accessibles dans les méthodes d'instance et de classe

Variables d'instance, de classe, de pool
Exemple : Déclaration

- Exemple :

```
Object subclass: #CompteBancaire
    instanceVariableNames: 'solde numero'
    classVariableNames: 'NombreComptesCrees'
    category: 'Exemple'
```


Variables d'instance, de classe, de pool
Exemple : Utilisation

- CompteBancaire class>>new

```
| ancienNombreComptes |
ancienNombreComptes := NombreComptesCrees ifNil: [0].
NombreComptesCrees := ancienNombreComptes + 1.
^super new
```

- CompteBancaire>>initialize

```
super initialize.
numero := NombreComptesCrees
```

Retour sur les blocs de code

Très utiles

Boucles, Conditionnelles, Exceptions, Concurrence ...

Blocs de code

- C'est une fonction anonyme et peut donc avoir des paramètres
  - [:p1 :p2| "plusieurs lignes de code"]
  - p1 et p2 sont des paramètres de bloc
- Peut être évalué à différents moments (de manière retardée)
  - value
  - value:
  - value:ifError:
  - valueWithArguments:

Exemples de blocs - 1

- |blocHello|

```
blocHello := [Transcript show: 'Hello'].
blocHello value. "Affichage de 'Hello' sur le Transcript"
```

- |blocPlusUn |

```
blocPlusUn := [:x| x + 1].
blocPlusUn value: 10. "=> 11"
```

- |blocPlusDelta delta nombre |

```
delta := 4.
blocPlusDelta := [:y| y + delta].
nombre := 8.
blocPlusDelta value: nombre. "=> 12"
```


Exemples de blocs - 2

- |blocSomme|

```
blocSomme := [:n :m| n + m].
blocSomme value: 1 value: 2. "=> 3"
```

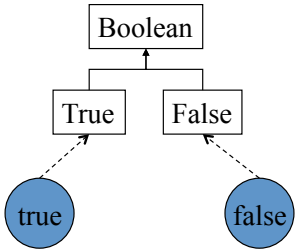
- |nombre factorielle |

```
nombre := 5.
factorielle := 1.
[nombre = 1] whileFalse: [
   factorielle := factorielle * nombre.
   nombre := nombre - 1]. "=> factoriel = 120"
```

Exemples de blocs - 3

- True>>ifTrue: aBlock
  ^aBlock value
- True>>ifFalse: aBlock
  ^nil
- Demo>>factoriel: nombre
  nombre = 1 ifTrue: [^1].
  ^ nombre * (self factoriel: nombre - 1)

- ***False>>ifTrue: aBlock***
  ^nil
- ***False>>ifFalse: aBlock***
  ^aBlock value



Retour sur l'identité des objets

Rappels

- Comparaison

  ==

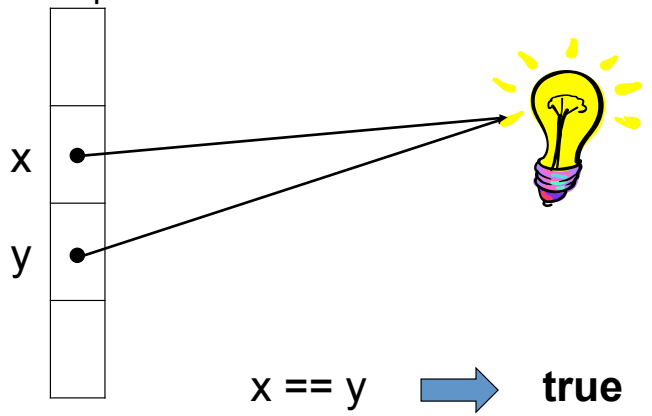
  - Teste l'identité : Compare les **variables**

  =

  - Teste l'égalité : Compare les **objets**

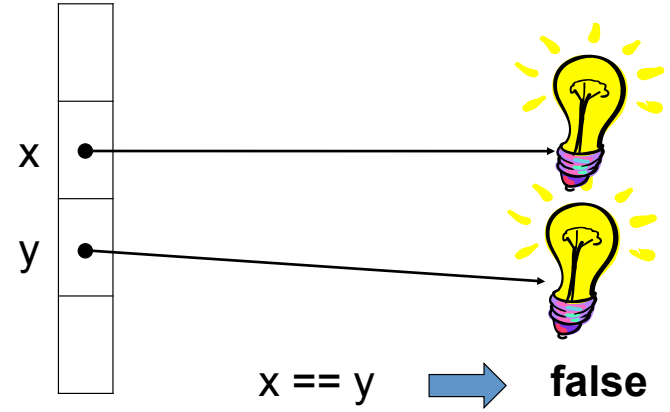
Test d'identité ==

- Compare les variables



Test d'identité ==

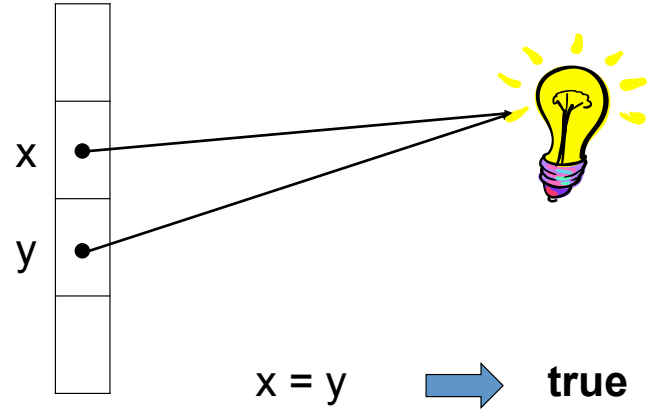
- Compare les variables



Noury Bouraqadi - option ISIC - Dépt. I. A.

Test d'égalité =

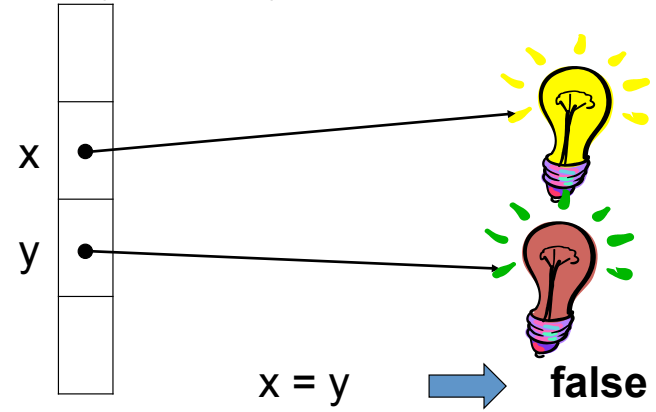
- Compare les objets



Noury Bouraqadi - option ISIC - Dépt. I. A.

Test d'égalité =

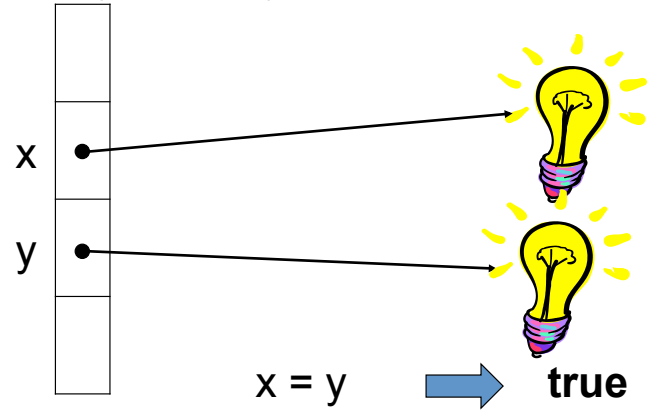
- Compare les objets



Noury Bouraqadi - option ISIC - Dépt. I. A.

Test d'égalité =

- Compare les objets



Noury Bouraqadi - option ISIC - Dépt. I. A.

Test d'égalité =

- Les classes des objets comparés doivent définir :
  - La méthode **=**
  - La méthod **hash**

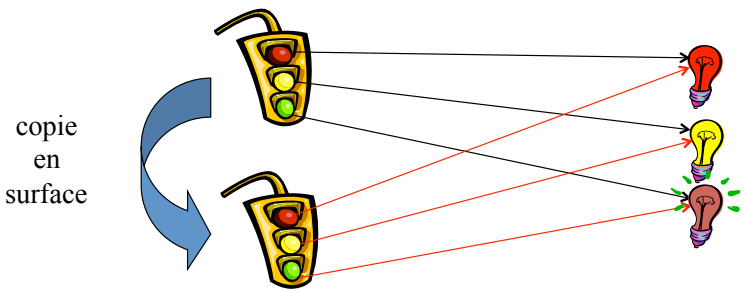
- Lampe>> = autreLampe
  ^(self couleur = autreLampe couleur) and: [
  self estAllumee = autreLampe estAllumee]
- Lampe>>hash
  ^(self couleur hash) bitXor: (self estAllumee hash)

Copie d'objets

- Deux manière de réaliser des copies
  - shallowCopy
    - Copie en "surface"
    - Utilisée par défaut (cf. méthode **copy**)
  - deepCopy
    - Copie en "profondeur"

Copie d'objets en surface

- Copie des références des champs
  - Partage d'objets via la copie de référence



Copie d'objets en profondeur

- Copie d'objets référencés aussi
  - création de nouveaux objets