

Pharo et la POO

Rene Paul Mages

renepaulmages@pharo.fr

European Smalltalk User Group

Journée Méditerranéenne du Logiciel Libre 2017



jMLZ) 2017

Linux
Azur
2017

<https://tinyurl.com/PharoCheatSheet>

Rectificatif du flyer

recto 5 6 1

verso 2 3 4

coquille page 3 cascade

Les 3 clics

8

Une visite de Pharo

- clic :** il s'agit du bouton de la souris le plus fréquemment utilisé et correspond au fait de cliquer avec une souris à un seul bouton sans aucune touche de modifications ; cliquer sur l'arrière-plan de l'image fait apparaître le menu "World" (voir la figure 1.3 (a)) ; nous utiliserons le terme *cliquer* pour définir cette action ;
- clic d'action :** c'est le second bouton le plus utilisé ; il est utilisé pour afficher un menu contextuel *c-à-d.* un menu qui fournit différentes actions dépendant de la position de la souris comme le montre la figure 1.3 (b). Si vous n'avez pas de souris à multiples boutons, vous configurerez normalement la touche de modifications *Ctrl* pour effectuer cette même action avec votre unique bouton de souris ; nous utiliserons l'expression "*cliquer avec le bouton d'action*"⁴.
- meta-clic :** vous pouvez finalement *meta-cliquer* sur un objet affiché dans l'image pour activer le "halo Morphic" qui est une constellation d'icônes autour de l'objet actif à l'écran ; chaque icône représentant une poignée de contrôle permettant des actions telles que *changer la taille* ou *faire pivoter l'objet*, comme vous pouvez le voir sur la figure 1.3 (c)⁵. En survolant lentement une icône avec le pointeur de votre souris, une bulle d'aide en affichera un descriptif de sa fonction. Dans Pharo, meta-cliquer dépend de votre système d'exploitation : Soit vous devez maintenir *SHIFT Ctrl* ou *SHIFT Option* tout en cliquant.

Introduction

- Pharo est une implémentation moderne, libre (sous licence MIT) et complète du langage de programmation Smalltalk et de son environnement (IDE = Environnement de Développement Intégré). Pharo est un fork de Squeak, une réécriture de Smalltalk-80 original.
- Ce podcast (154 secondes) vous donnera une idée précise de Pharo :
<https://www.youtube.com/watch?v=KDvNuOjY4>
- Historique de Smalltalk (voir aussi wikipedia) :
<https://tinyurl.com/EarlyHistorySmalltalk>
- Pharo est utilisé dans de nombreux domaines (dont la robotique) et enseigné dans une trentaine d'universités

PHARO sur le web

Pharo site <http://pharo.org>

Success Stories <http://pharo.org/success>

Community & Help <https://pharo.org/community>

Association <http://association.pharo.org>

Twitter <http://twitter.com/pharoproject>

Slides <http://blog.pharo.fr/pages/PHARO-Slides>

Blog <http://blog.pharo.fr>

PHARO : la documentation

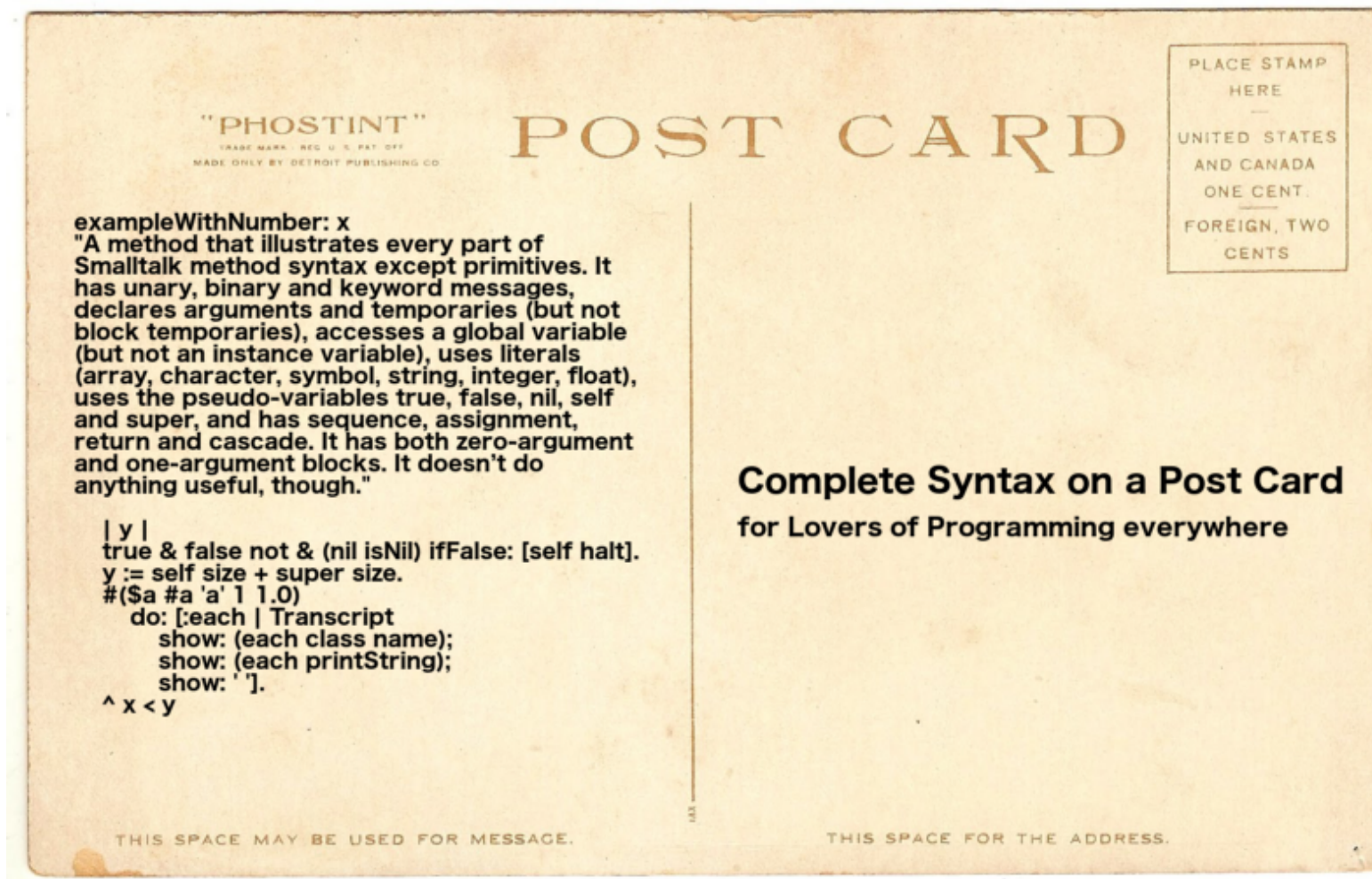
- Le livre « Pharo par l'exemple » (PBE ch3) :
<http://files.pharo.org/books/pharo-par-lexemple/>
- Des livres disponibles en ligne :
<http://files.pharo.org/books/>
- Un MOOC dédié à Pharo :
<http://files.pharo.org/mooc/>
- Une feuille synthétique sur la syntaxe de Pharo :
<https://tinyurl.com/FlyerSyntaxPharo>

Le modèle objet de Pharo

Pharo est un modèle objet à la fois simple, uniforme, puissant, dynamiquement typé :

- Tout est objet (y compris les classes)
- Tout objet est instance de classe
- Toute classe a une super-classe
- Entre les classes l'héritage est simple
- Les variables d'instance sont privées
- Les méthodes sont publiques
- Les blocs sont des clôtures lexicales
- Tout se passe par envoi de messages

La syntaxe tient sur une carte postale



PHARO : la syntaxe (1/2)

- Pharo, comme la plupart des *dialectes modernes de Smalltalk*, adopte une syntaxe proche de celle de Smalltalk-80.
- Une *syntaxe minimaliste* conçue pour envoyer des messages
- | count | est une déclaration de variable locale
- count:= 10 est une affectation
- Le point est un séparateur d'expressions :

count:=10 . count:=count+1

PHARO : la syntaxe (2/2)

Les éléments syntaxiques

- Les expressions sont composées des blocs constructeurs suivants :
- (i) six mots-clés réservés ou *pseudo-variables* :
self , super , nil , true , false , thisContext
- (ii) des expressions constantes pour des *objets littéraux* comprenant les nombres, les caractères, les chaînes de caractères, les symboles et les tableaux
- (iii) des déclarations de variables
- (iv) des affectations
- (v) des blocs ou *fermetures lexicales* – block closures en anglais –
- (vi) des *messages*.

La notion de message (1/3)

- 'Hello' reversed (message UNAIRE)
- 'Hello' asUppercase
- 'Hello' first
- 'Hello' size

- 'Hello' , ' Pharoers' (message BINAIRE)
- >>> 'Hello Pharoers'

- 'Hello' at: 1 (message à MOTS-CLES)
- 'Hello' at: 2
- 'Hello' copyFrom: 1 to: 3

La notion de message (2/3)

- Trois types de messages :

Message unaire : 4 factorial

Message binaire : 6 + 24

Message à mots-clés : 30 between:10 and:100

- La priorité :

3 factorial + 4 factorial between: 10 and: 100

((3 factorial) + (4 factorial)) between:10 and: 100

La notion de message (3/3)

"La notion de cascade"

| anOrderedCollection |

anOrderedCollection := OrderedCollection new.

anOrderedCollection add: 1.

anOrderedCollection add: 2.

anOrderedCollection add: 3.

anOrderedCollection := OrderedCollection new.

anOrderedCollection add: 1; add: 2; add: 3.

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

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

Les blocs

- Des crochets [] définissent un bloc, aussi connu sous le nom de *block closure* ou *fermeture lexicale*, laquelle est un objet à part entière représentant une fonction. Comme nous le verrons, les blocs peuvent avoir des arguments et des variables locales.
- [2017+ 1] value
- [count := count + 1]
- [:c | c == cr ifTrue: [count := count + 1]].

ni if, ni then, ni else

- Les conditions et les itérations :
- `(2 < 7) ifTrue : ['ok'] ifFalse['rien ne va plus']`
- `4 timesRepeat: [' SophiaTech ']`
- `n:=1. [n < 1000] whileTrue: [n := n*2].`
- `n:=1. to: 10 do: [:count | n := n + count].`

Un exemple de méthode (1/2)

The screenshot shows the Smalltalk IDE interface. The title bar reads "Integer>>#timesRepeat:". The left pane shows a "Scoped" view of the "kernel" environment, with "Numbers" selected. The middle pane shows the class hierarchy for "Integer", including subclasses like "BoxedFloat64", "SmallFloat64", "Fraction", "ScaledDecimal", "LargeInteger", and "LargeNegativeInteger". The right pane shows the "History Navigator" with "timesRepeat:" selected. Below the panes, the source code for the #timesRepeat: method is displayed.

timesRepeat: aBlock

"Evaluate the argument, aBlock, the number of times represented by the receiver."

```
| count |  
count := 1.  
[count <= self]  
  whileTrue:  
    [aBlock value.  
     count := count + 1]
```


Un exemple de méthode (2/2)

factorial

"Answer the factorial of the receiver."

self = 0 if True: [[^] 1].

self > 0 if True: [[^] self * (self - 1) factorial].

self error: 'Not valid for negative integers'

Les collections (PBE ch9)

De nombreuses collections dont les chaînes de caractères (instances de la classe String) :

```
| chaine1 chaine2 chaine3 chaine4 |
```

```
chaine1 := 'Le'.
```

```
chaine2 := 'campus'.
```

```
chaine3 := 'SophiaTech'.
```

```
chaine4 := chaine1, chaine2, chaine3
```

Des classes et des méthodes

202

Les collections

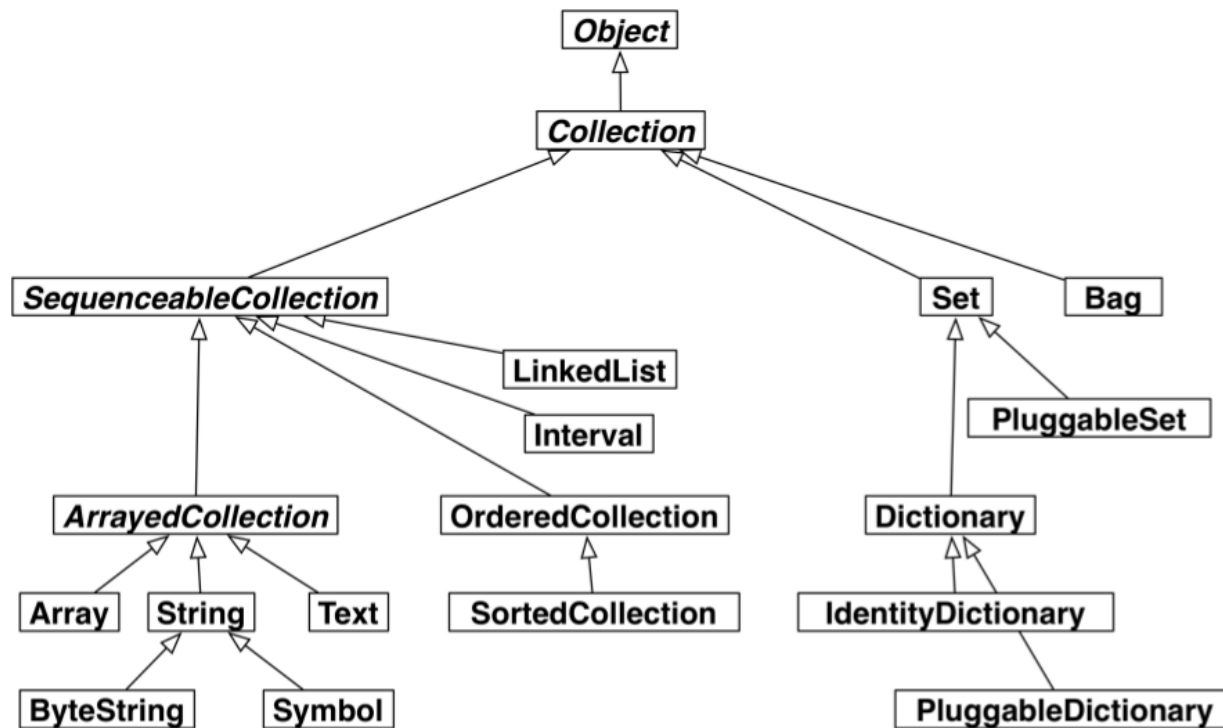


FIGURE 9.1 – Certaines des classes majeures de collections de Pharo.

Les classes de base (PBE ch8)

Chapitre 8

Les classes de base

Une grande partie de la magie de Smalltalk ne réside pas dans son langage mais dans ses bibliothèques de classes. Pour programmer efficacement en Smalltalk, vous devez apprendre comment les bibliothèques de classes servent le langage et l'environnement. Les bibliothèques de classes sont entièrement écrites en Smalltalk et peuvent facilement être étendues, puisqu'un paquetage peut ajouter une nouvelle fonctionnalité à une classe même s'il ne définit pas cette classe.

Notre but ici n'est pas de présenter en détail l'intégralité des bibliothèques de classes de Pharo, mais plutôt d'indiquer quelles classes et méthodes clés vous devrez utiliser ou surcharger pour programmer efficacement. Ce chapitre couvre les classes de base qui vous seront utiles dans la plupart de vos applications : Object, Number et ses sous-classes, Character, String, Symbol et Boolean.

Pharo help

Le canal #pharo de Freenode est moins actif que *discord* :

- <https://pharo.org/community>

Les deux listes de diffusion :

- http://lists.pharo.org/mailman/listinfo/pharo-users_lists.pharo.org
- http://lists.pharo.org/mailman/listinfo/pharo-dev_lists.pharo.org

Des questions ?

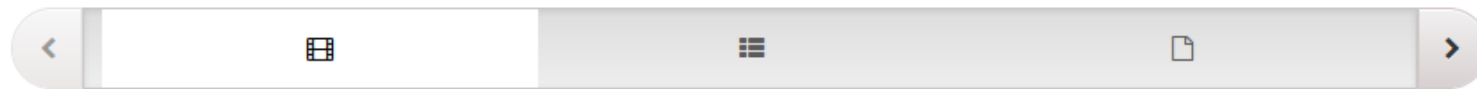
- Tous mes remerciements :

à Stéphane Ducasse, Serge Stinckwich, Marcus Denker, Noury Bouraqadi, Alexandre Bergel, Luc Fabresse, Damien Cassou, Hilaire Fernandes et à toute la communauté Pharo.

à ESUG : <http://www.esug.org>

- Mon email : renepaulimages@pharo.fr
- Mon blog : <http://blog.pharo.fr>

La machine virtuelle de Pharo

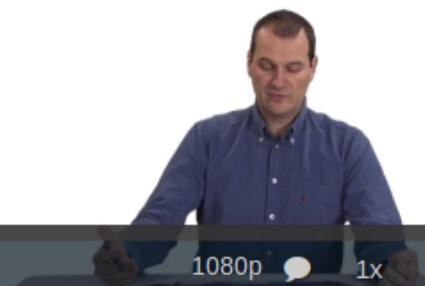


6.6. [LECTURE] RUNTIME ARCHITECTURE

Execution Model

Pharo virtual machine (VM) executes compiled code

- The virtual machine and its plugins are platform specific (different versions for different OSES)
- VMs exist for MacOS, Windows, Linux (different versions), iOS, ARM, Android



0:33 8:41 1080p 1x

Télécharger la vidéo en qualité : [Haute \(1080p\)](#) / [Normale \(720p\)](#) / [Mobile \(480p\)](#)

PHARO et son installation

- Sur une machine GNU/Linux :
- <http://pharo.org/gnu-linux-installation>
- Sur une machine OSX ou Windows :
- <https://pharo.org/download>
- 4 fichiers principaux dont :
- 1 Machine Virtuelle (VM) , PharoV60.sources , Pharo6.1.image, Pharo6.1.changes

Les 4 précédentes conférences

- Par Damien Cassou aux JM2L 2007
- Par Hilaire Fernandes aux JM2L 2008
- Par Marcus Denker aux JM2L 2009
- Par Noury Bouraqadi aux JM2L 2010

<http://www.linux-azur.org/~rmages/smalltalk/>

conditions et itérations

Il n'y a pas de syntaxe particulière pour les structures de contrôle ; ce ne sont que des messages qui, sous certaines conditions, évaluent des blocs :

```
n := 1.  
10 timesRepeat: [ n := n*2 ].  
n    →    1024
```

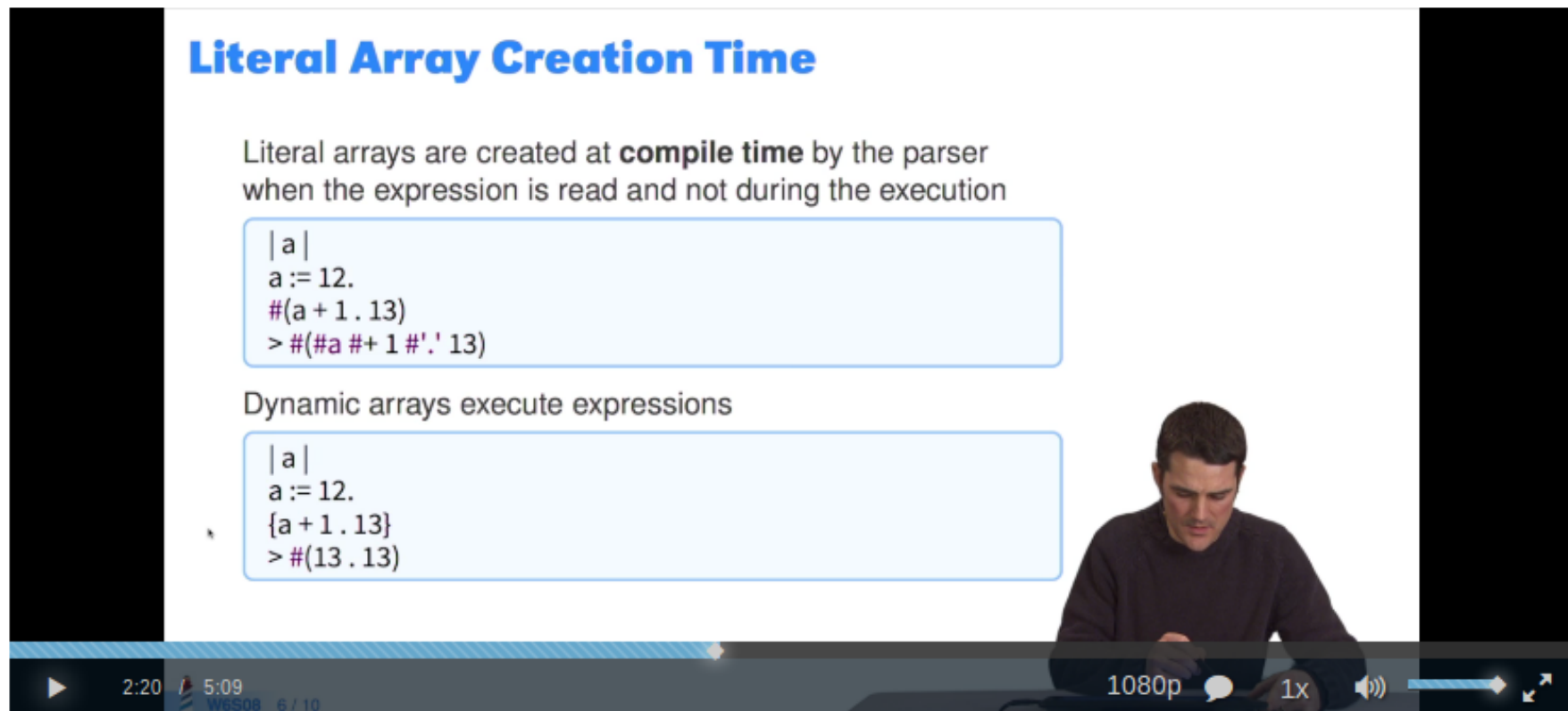
```
n := 0.  
(1 to: 10) do: [ :element | n := n + element ].  
n    →    55
```

self , super, yourself

- *self est une pseudo-variable* qui pointe vers l'objet (le receveur) sur lequel la méthode courante s'exécute. (PBE ch5)
- *super est aussi une pseudo-variable* qui pointe vers le receveur mais l'algorithme de lookup commence sa recherche dans la super-classe du receveur et non dans sa classe .(PBE ch5)
- En revanche yourself est une méthode (PBE ch9) :
- `OrderedCollection new add: 1; add: 2; yourself`

Les tableaux (dynamiques ou littéraux)

6.8. [LECTURE] DYNAMIC VS. LITERAL ARRAYS



Literal Array Creation Time

Literal arrays are created at **compile time** by the parser when the expression is read and not during the execution

```
| a |  
a := 12.  
#(a + 1 . 13)  
> #(#a #+ 1 #' 13)
```

Dynamic arrays execute expressions

```
| a |  
a := 12.  
{a + 1 . 13}  
> #(13 . 13)
```

The screenshot shows a video player interface with a man in the bottom right corner. The video player controls at the bottom indicate a duration of 2:20 / 5:09, a video quality of 1080p, and a volume of 1x.

Télécharger la vidéo en qualité : Haute (1080p) / Normale (720p) / Mobile (480p)

Classes et Meta-classes

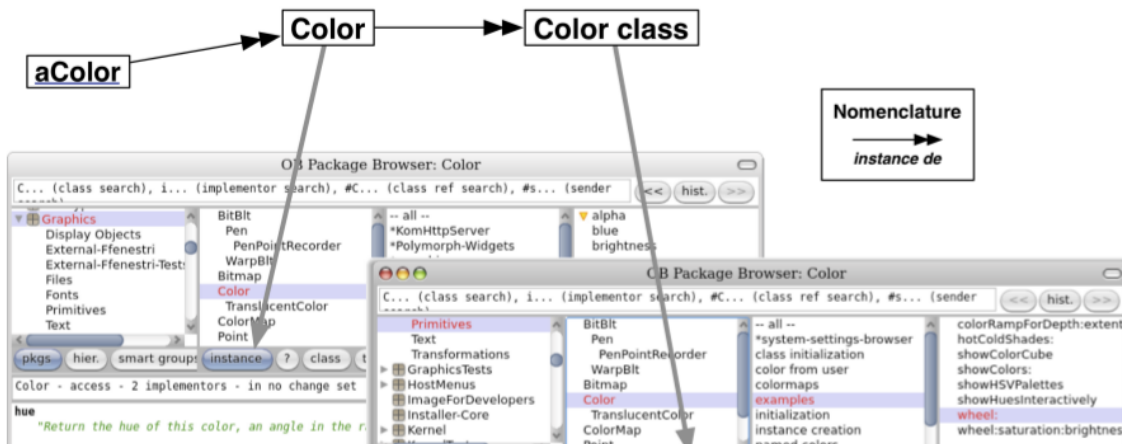
Tout objet est instance de classe

89

Le côté instance et le côté classe

Puisque les classes sont des objets, elles peuvent avoir leurs propres variables d'instance ainsi que leurs propres méthodes. Nous les appelons *variables d'instance de classe* (en anglais *class instance variables*) et *méthodes de classe*, mais elles ne sont véritablement pas différentes des variables et méthodes d'instances ordinaires : les variables d'instance de classe ne sont seulement que des variables d'instance définies par une méta-classe. Quant aux méthodes de classe, elles correspondent juste aux méthodes définies par une méta-classe.

Une classe et sa méta-classe sont deux classes distinctes, et ce, même si cette première est une instance de l'autre. Pour vous, tout ceci sera somme toute largement trivial : vous n'aurez qu'à vous concentrer sur la définition du comportement de vos objets et des classes qui les créent.



Pharo : les frameworks

- Pour une vue d'ensemble :
- <http://files.pharo.org/books/enterprise-pharo/>
- Deux des plus connus :
- <http://www.seaside.st/>
- <http://agilevisualization.com>
- Et des centaines moins connus :
- <http://catalog.pharo.org>

Le framework SEASIDE (PBE ch12)

- Seaside a été créé en 2002 par Avi Bryant et Julian Fitzell pour construire des applications web.
- Le site web de Seaside :
<http://www.seaside.st>
- Entièrement dédié à Seaside : le chapitre 14 du livre Updated Pharo By Example :
<https://tinyurl.com/UpdatedPharoByExample>

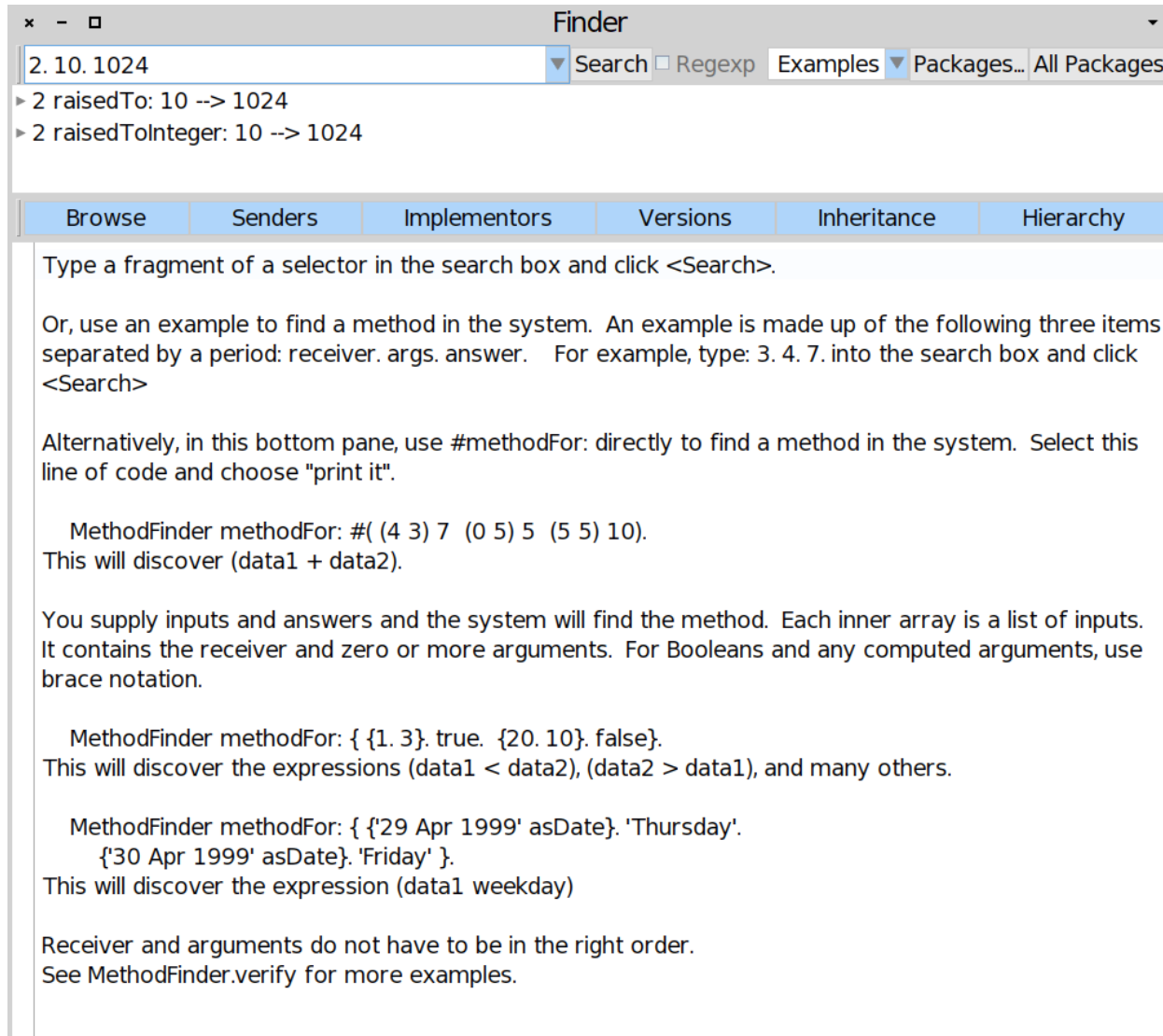
Pharo 7.0

- La prochaine version de Pharo est en préparation. Vous pouvez visionner (en 39 minutes) ses principales caractéristiques :
- <https://www.youtube.com/watch?v=OGkGgx4iymM>
- Conseil d'abonnement à une liste de diffusion :
- http://lists.pharo.org/mailman/listinfo/pharo-users_lists.pharo.org

Senders , implementors

- Un exemple de sender (ou méthode émettrice) :
- Avec la méthode factorial
- Un exemple d'implementor (ou méthode contenante) :
- Avec la méthode factorial
- Explications détaillées à l'aide du browser Nautilus ...

Le finder de Pharo



Finder

2. 10. 1024 Search Regexp Examples Packages... All Packages

- ▶ 2 raisedTo: 10 --> 1024
- ▶ 2 raisedToInteger: 10 --> 1024

Browse Senders Implementors Versions Inheritance Hierarchy

Type a fragment of a selector in the search box and click <Search>.

Or, use an example to find a method in the system. An example is made up of the following three items separated by a period: receiver. args. answer. For example, type: 3. 4. 7. into the search box and click <Search>

Alternatively, in this bottom pane, use #methodFor: directly to find a method in the system. Select this line of code and choose "print it".

```
MethodFinder methodFor: #( (4 3) 7 (0 5) 5 (5 5) 10).
```

This will discover (data1 + data2).

You supply inputs and answers and the system will find the method. Each inner array is a list of inputs. It contains the receiver and zero or more arguments. For Booleans and any computed arguments, use brace notation.

```
MethodFinder methodFor: { {1. 3}. true. {20. 10}. false}.
```

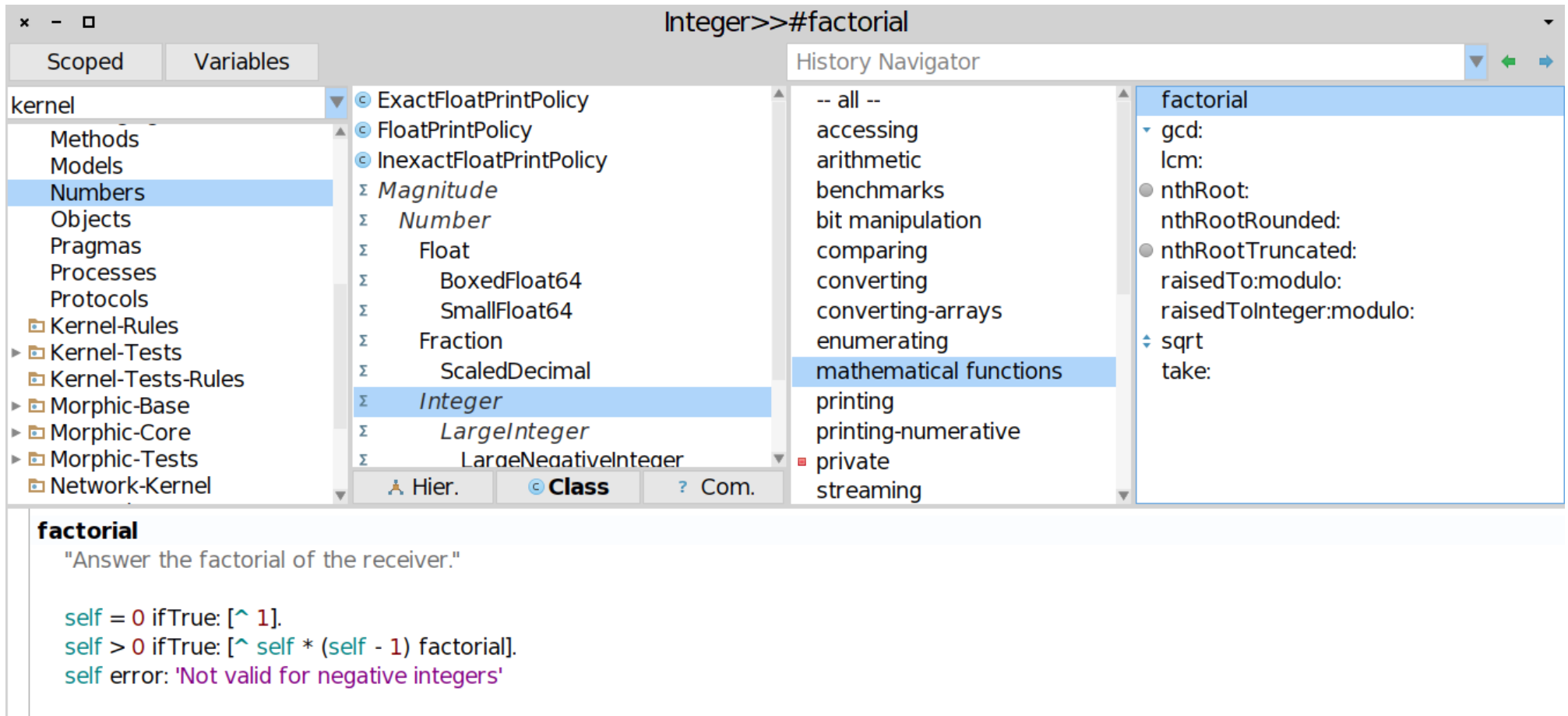
This will discover the expressions (data1 < data2), (data2 > data1), and many others.

```
MethodFinder methodFor: { {'29 Apr 1999' asDate}. 'Thursday'.  
{'30 Apr 1999' asDate}. 'Friday' }.
```

This will discover the expression (data1 weekday)

Receiver and arguments do not have to be in the right order.
See MethodFinder.verify for more examples.

Nautilus le browser du système (1/2)



The screenshot shows the Nautilus file manager interface. The main window title is "Integer>>#factorial". The left sidebar shows a tree view of the file system, with "Numbers" selected. The main pane displays a list of classes and methods, with "Integer" selected. The "History Navigator" pane shows a list of categories, with "mathematical functions" selected. The bottom pane shows the source code for the "factorial" method.

```
kernel
├── Methods
├── Models
├── Numbers
├── Objects
├── Pragmas
├── Processes
├── Protocols
├── Kernel-Rules
├── Kernel-Tests
├── Kernel-Tests-Rules
├── Morphic-Base
├── Morphic-Core
├── Morphic-Tests
└── Network-Kernel
```

ExactFloatPrintPolicy
FloatPrintPolicy
InexactFloatPrintPolicy
Magnitude
Number
Float
BoxedFloat64
SmallFloat64
Fraction
ScaledDecimal
Integer
LargeInteger
LargeNegativeInteger

-- all --
accessing
arithmetic
benchmarks
bit manipulation
comparing
converting
converting-arrays
enumerating
mathematical functions
printing
printing-numerative
private
streaming

factorial
gcd:
lcm:
nthRoot:
nthRootRounded:
nthRootTruncated:
raisedTo:modulo:
raisedToInteger:modulo:
sqrt
take:

factorial
"Answer the factorial of the receiver."

self = 0 ifTrue: [^ 1].
self > 0 ifTrue: [^ self * (self - 1) factorial].
self error: 'Not valid for negative integers'

Nautilus le browser du système (2/2)

The screenshot displays the Nautilus IDE interface for the `LinkedList` class, specifically focusing on the `#collect` method. The interface is divided into several panes:

- Classes:** A list of classes including `Array`, `Heap`, `Interval`, `LinkedList` (selected), `ManifestCollectionsSequenceable`, `OrderedCollection`, `SortedCollection`, `OrderedDictionary`, `OrderedIdentityDictionary`, `SharedQueue`, and `SparseLargeTable`.
- Protocols:** A list of protocols including `-- all --`, `accessing`, `adding`, `copying`, `enumerating` (selected), `private`, `removing`, and `testing`.
- Methods:** A list of methods including `collect` (selected), `collect:thenReject:`, `collect:thenSelect:`, `do:`, `linksDo:`, `select:`, `select:thenCollect:`, and `species`.
- Editing:** The source code for the `collect` method is displayed, showing a block of code that evaluates a block with each element of the receiver and collects the results into a new collection.
- Comments:** A description of the `collect` method is provided, explaining its behavior as a sequential collection and its relationship to `OrderedCollection`.
- Quality Assistant's Feedback:** A section at the bottom of the interface provides feedback on the code quality, including a warning icon and the text "Quality Assistant's Feedback".

Smalltalk May Be the Nikola Tesla of the IT Industry

- <https://hackernoon.com/smalltalk-is-the-nikola-tesla-of-the-it-industry-dbef0a8ddd57>
- « Smalltalk was the brainchild of Alan Kay, a true visionary who led a brilliant team at Xerox PARC. Today, Smalltalk is greatly underestimated. Although Kay never thought of Smalltalk as the central focus of his vision, it remains a powerful force for the advancement of programming technology. More than four decades later, no programming language has yet to catch up to Smalltalk in terms of simplicity and elegance, minimal cognitive friction, object-oriented purity, elegant live coding and debugging, enormous programmer productivity, and professional respect. » Richard Kenneth