Pharo Syntax in a Nutshell

Stéphane Ducasse

http://stephane.ducasse.free.fr/
stephane.ducasse@inria.fr

```
exampleWithNumber: x
   "This method illustrates every part of Smalltalk method syntax except primitives."
   <aMethodAnnotation>

   | y |
   true & false not & (nil isNil)
      ifFalse: [self halt].
   y := self size + super size.
   #($a #a 'a' 1 1.0)
      do: [ :each | Transcript
                  show: (each class name);
                  show: (each printString);
                  show: ' ' ].
   ^ x < y
```

Originally Made for Kids

- Read it as a non-computer-literate person:

```
| bunny |
bunny := Actor fromFile: 'bunny.vrml'.
bunny head doEachFrame:
   [ bunny head
      pointAt: (camera
              transformScreenPointToScenePoint: Sensor mousePoint
              using: bunny)
      duration: camera rightNow ]
```

Getting the Pharo Logo

```
(ZnEasy getPng: 'http://pharo.org/web/files/pharo.png')
   asMorph openInWindow.
```

A real example

```
Integer>>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’
```

Sending a request

```
ZnClient new
    url: 'http://localhost:8080/books/1';
    formAt: 'author' put: 'SquareBracketAssociates';
    formAt: 'title'  put: 'Pharo For The Enterprise';
    put
```

Language Constructs

- ^ expression return
- "..." comments
- # symbol or array
- '...'string
- [ ] block or byte array
- . separator and not terminator
- ; cascade (sending several messages to the same object)
- | local or block variable definition
- := assignment
- $ character
- : end of selector name
- e, r number exponent or radix
- <annotation> method annotation

Syntax in a Nutshell

- comment: "a comment"
- character: $c $h $a $r $a $c $t $e $r $s $# $@
- string: 'a nice string' 'lulu' 'l'idiot'
- symbol: #mac #+
- array: #(1 2 3 (1 3) $a 4)
- byte array: #[1 2 3]
- integer: 1, 2r101
- real: 1.5, 6.03e-34,4, 2.4e7
- float: $1/33$
- boolean: true, false
- point: 10@120
- Note that @ is not an element of the syntax, but just a message sent to a
- number. This is the same for $/$, bitShift, ifTrue:, do: ...

Syntax in a Nutshell (II)

- assigment: var := aValue
- block (lexical closure): [:var ||tmp| expr...]
- temporary variable: |tmp|
- block variable: :var
- unary message: receiver selector
- binary message: receiver selector argument
- keyword based: receiver keyword1: arg1 keyword2: arg2...
- cascade: message ; selector ...
- separator: message . message
- result: ^
- parenthesis: (...)

Conditionals are also message sends

```
Weather isRaining
   ifTrue: [self takeMyUmbrella]
   ifFalse: [self takeMySunglasses]
```

- `ifTrue:ifFalse` is sent to an object: a boolean!

Loops are also message sends

```
1 to: 100 do: [: i| Transcript << i ]]
> 1
> 2
> 3
> 4
```

- `to:do:` is a message sent to a small integer

Loops are also message sends

```
#(1 2 −4 −86)
   do: [ :each | Transcript show: each abs printString ; cr ]
> 1
> 2
> 4
> 86
```

- Yes we ask the collection object to perform the iteration

Messages and their composition

- Three kinds of messages
  - Unary: `Node new`
  - Binary: `1+2`, `3@4`
  - Keywords: `aTamagoshi eat: #cooky furiously: true`
- Message Priority
  - (Msg) $>$ unary $>$ binary $>$ keywords
  - Same Level from left to right
- Example:

```
(10@0 extent: 10@100) bottomRight
```

- Creates a rectangle and asks its bottom right corner

```
s isNil ifTrue: [ self halt ]
```

Blocks

- A kind of anonymous method
- Can be passed as method argument or stored in variables
- Functions

```
fct(x)= x*x+3.
fct(2).
```

```
fct :=[:x| x * x + 3].
fct value: 2
```

Block Usage

```
    #(1 2 3) do: [:each | Transcript show: each printString ; cr]
```

```
Integer>>factorial
    | tmp |
    tmp:= 1.
    2 to: self do: [ :i | tmp := tmp * i ]
```

Class definition

```
Object subclass: #Point
    instanceVariableNames: 'x y'
    classVariableNames: ''
    category: 'Graphics'
```

Method definition

- Normally defined in a browser or (by directly invoking the compiler)
- Methods are public
- Always return `self`

```
Node>>accept: thePacket
    "If the packet is addressed to me, print it.
    Else just behave like a normal node"

    thePacket isAddressedTo: self)
        ifTrue: [ self print: thePacket ]
        ifFalse: [ super accept: thePacket ]
```

Instance Creation are Messages Too!

- Messages sent to instance

```
'1', 'abc'
1@2
```

- Basic class creation messages are `new` and `new:` sent to a class

```
Monster new
```

- Class specific message creation (messages sent to classes)

```
Tomagoshi withHunger: 10
```

Conclusion

- Compact syntax
- Few constructs
- Messages
- Closures