Pharo: Syntax in a Nutshell

S. Ducasse and M. Denker

http://www.pharo-project.org

Less is better

- No constructors
- No types declaration
- No interfaces
- No packages/private/protected
- No parametrized types
- No boxing/unboxing
- **And really powerful**

Objects are instances of Classes

Objects are instances of Classes

```
(10@200)
```

Objects are instances of Classes

```
(10@200) class
```

Objects are instances of Classes

```
(10@200) class

Point
```

Classes are objects too

Classes are objects too

Point selectors

Classes are objects too

Point selectors

```
> an IdentitySet(#eightNeighbors #+ #isZero
#sortsBefore: #degrees #printOn: #sideOf:
#fourNeighbors #hash #roundUpTo: #min: #min:max:
#max #adaptToCollection:andSend: #quadrantOf:
```

Classes are objects too

Point instVarNames

Classes are objects too

```
Point instVarNames

 >#('x' 'y')
```

Methods are public

Instance variables are protected

Single Inheritance

Single Inheritance

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

Complete Syntax on a PostCard

```
exampleWithNumber: x

"A method that has unary, binary, and key word messages, declares arguments and
temporaries (but not block temporaries),  accesses a global variable (but not and
instance variable), uses literals (array, character, symbol, string, integer, float), uses the
pseudo variable true false, nil, self, and super, and has sequence, assignment, return
and cascade. It has both zero argument and one argument blocks."

        |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
```

Language Constructs

^	return
“	comments
#	symbol or array
‘	string
[ ]	block or byte array
.	separator and not terminator (or namespace access in VW)
;	cascade (sending several messages to the same instance)
|	local or block variable

Syntax

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

3 kinds of messages

Unary messages

```
5 factorial
Transcript cr
```

Binary messages

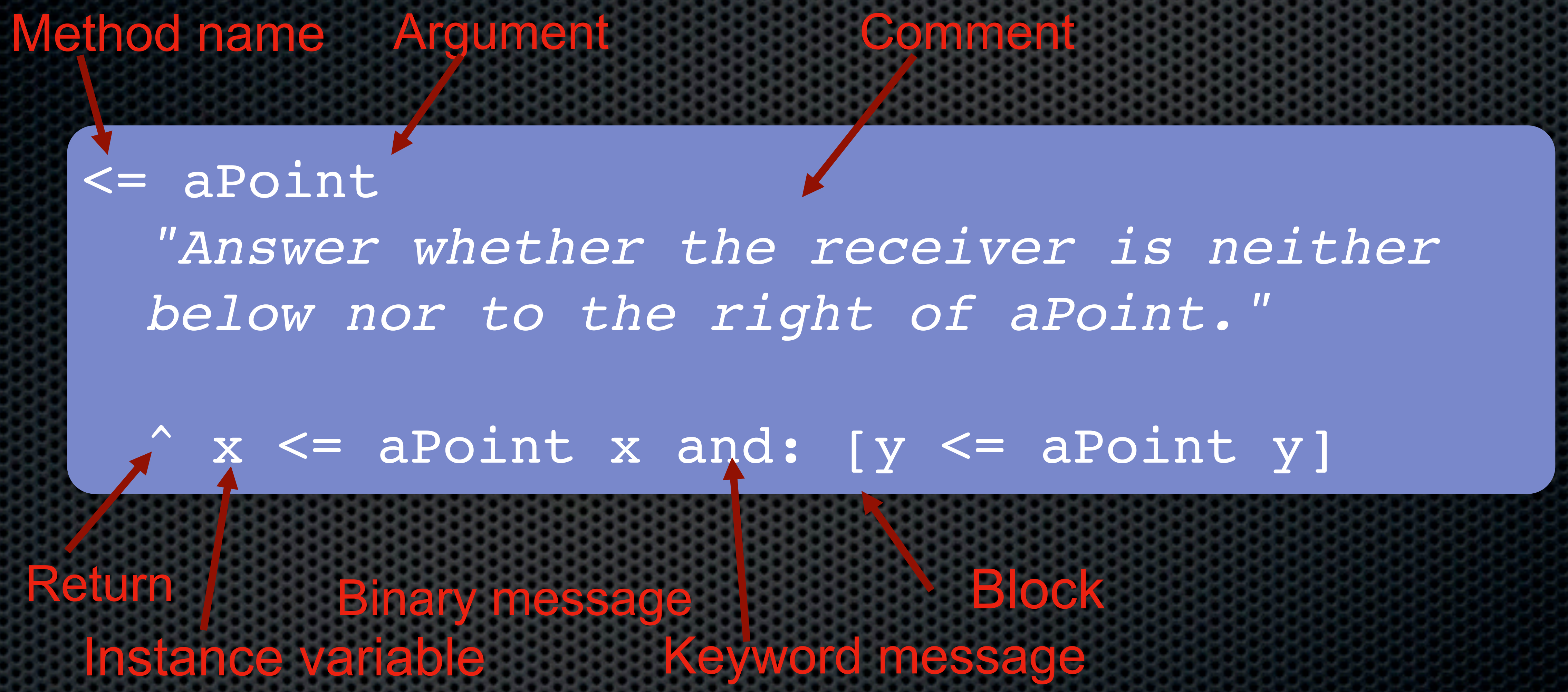
```
3 + 4
```

Keywords messages

```
3 raisedTo: 10 modulo: 5

Transcript show: 'hello world'
```

A typical method in Point



```
<= aPoint
   "Answer whether the receiver is neither
   below nor to the right of aPoint."

   ^ x <= aPoint x and: [y <= aPoint y]
```

```
(2@3) <= (5@6)
```

```
true
```

Blocks

- Anonymous method
- Passed as method argument or stored
- Functions

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

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

Block usage

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


#(1 2 3) do: [:each | each crLog]
```

Statements and cascades

Temporary variables

Statement

```
| p pen |
p := 100@100.
pen := Pen new.
pen up.
pen goto: p; down; goto: p+p
```

Cascade

Control structures

Every control structure is realized by message sends

```
4 timesRepeat: [Beeper beep]
```

```
max: aNumber
  ^ self < aNumber
      ifTrue: [aNumber]
      ifFalse: [self]
```

Simple and elegant