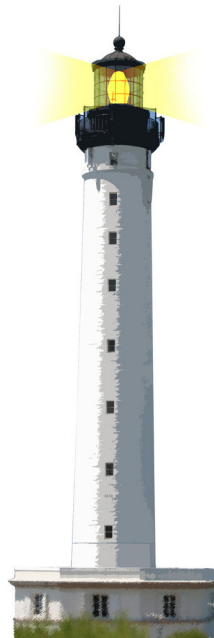


Class and Method Definitions



Class and Method Definitions in Pharo

- classes and methods are defined within tools
- there is no dedicated syntax



Class Definition in Pharo

The screenshot shows the Pharo IDE interface for the `Point` class. The window title is "Point". The interface is divided into several panes:

- Scoped Variables:** A list of categories including BasicObjects, Chronology, Classes, Copying, Exceptions, Messaging, Methods, Models, and Numbers. "BasicObjects" is selected.
- History Navigator:** A list of categories including Character, CombinedChar, Margin, Point, and Rectangle. "Point" is selected.
- Method List:** A list of methods including -- all --, accessing, arithmetic, comparing, converting, copying, extent functions, geometry, interpolating, point functions, polar coordinates, and printing.
- Code Editor:** Contains the following class definition for `Point`:

```
Object subclass: #Point
  instanceVariableNames: 'x y'
  classVariableNames: ''
  category: 'Kernel-BasicObjects'
```

At the bottom of the IDE, there are buttons for "Hier.", "Class", and "? Com.".

Class Definition is a Message

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

We send the message `subclass:inst...` to the superclass to create the class



Method Definition in Pharo

The screenshot shows the Pharo IDE interface. At the top, the window title is "Integer>>#factorial". Below the title bar, there are two tabs: "Scoped" and "Variables". To the right of these tabs is a "History Navigator" with a search field and navigation arrows. The main area is divided into three panes. The left pane shows a tree view of the object model, with "Numbers" selected. The middle pane shows the class hierarchy for "Integer", with "Integer" selected. The right pane shows the "factorial" method definition. Below the panes, the source code for the "factorial" method is displayed.

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'
```

Method Definition in Pharo

```
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'
```

In which class is factorial defined?

Presentation Convention

In this lecture, a method will be displayed as

```
Integer >> 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'
```

- **Integer** » is not part of the syntax
 - it tells you the method's class

Presentation Convention

Integer>>#factorial

Scoped Variables History Navigator

Type: Pkg1|^Pkg2|P|

- Numbers
- Objects
- Pragmas
- Processes
- Protocols
- Kernel-Tests
- Keymapping-Cor
- Keymapping-Key

- ExactFloatPrintPolicy
- FloatPrintPolicy
- InexactFloatPrintPolicy
- Magnitude
- Number
- Float
- Fraction
- ScaledDecimal
- Integer

Hier. Class Com.

accessing
arithmetic
benchmarks
bit manipulation
comparing
converting
converting-arrays
enumerating
filter streaming
mathematical func

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'
```

1/6 [1] Format as you read W +L

Remember Messages

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'

- factorial is the method name
- =, >, * and - are binary messages
- factorial is an unary message
- ifTrue: and error: are keyword messages
- the caret ^ is for returning a value

A Method Returns self by Default

```
Game >> initializePlayers
self players
  at: 'tileAction'
  put: ( MITileAction director: self )
```

is equivalent to

```
Game >> initializePlayers
self players
  at: 'tileAction'
  put: ( MITileAction director: self ).
^ self    "-- optional"
```



Class Methods

Point class>>#x:y:

Scoped Variable: History Navigator

Type: Pkg1|^Pkg2|

- Character
- CombinedChar
- Margin
- Point
- Rectangle

-- all --

instance creation

*System-Setting

r:degrees:

x:y:

StephaneDucasse 11/10/2015 18:35 - instance creation - 14 senders - 2 implementors

```
x: xInteger y: yInteger
"Answer an instance of me with coordinates xInteger and yInteger."

^ self basicNew setX: xInteger setY: yInteger
```

1/4 [1] Format as you read W +L

- press the button class to define a class method
- in lectures, we add class

Point class >> x: xInteger y: yInteger

"Answer an instance of me with coordinates xInteger and yInteger."

^ self basicNew setX: xInteger setY: yInteger

What You Should Know

- A class is defined by sending a message to its superclass
- Classes are defined inside packages
- Methods are public
- By default a method returns the receiver, `self`
- Class methods are just methods of the class side



A course by

S. Ducasse, G. Polito, and Pablo Tesone



Except where otherwise noted, this work is licensed under CC BY-NC-ND 3.0 France
<https://creativecommons.org/licenses/by-nc-nd/3.0/fr/>