

Class and Method Definitions

Damien Cassou, Stéphane Ducasse and Luc Fabresse

W1S06



<http://www.pharo.org>



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 editing the `Point` class. The window title is "Point". On the left, the "Variables" pane shows a tree view with "BasicObjects" selected. The "History Navigator" pane on the right lists various categories like "accessing", "arithmetic", etc. The main editor displays the following class definition:

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

At the bottom of the editor, there is a status bar with "1/4 [1]", a "Format as you read" checkbox, and a warning icon with the text "Excessive number of methods".

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 with the following components:

- Window Title:** Integer>>#factorial
- Left Panel (Scopes):** Shows a tree view with 'Numbers' selected. Other categories include Objects, Pragmas, Processes, Protocols, Kernel-Tests, Keymapping-Cor, and Keymapping-Key.
- Class Browser:** Lists classes including ExactFloatPrintPolicy, FloatPrintPolicy, InexactFloatPrintPolicy, Magnitude, Number, Float, Fraction, ScaledDecimal, and Integer (highlighted).
- History Navigator:** Lists various mathematical and utility methods such as accessing, arithmetic, benchmarks, bit manipulation, comparing, converting, enumerating, filter streaming, and mathematical func.
- Method List:** Shows the 'factorial' method selected, with sub-methods like gcd, lcm, nthRoot, nthRootRounded, nthRootTruncated, raisedTo:modulo, raisedToInteger:modulo, sqrt, and take.
- Method Definition:**

```
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'
```
- Bottom Bar:** Shows '1/6 [1]' and a checkbox for 'Format as you read' with 'W' and '+L' buttons.

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 ifTrue: [ ^ 1 ].
self > 0 ifTrue: [ ^ 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

The screenshot shows the Pharo IDE interface. The top window is titled "Integer>>#factorial". The left sidebar shows a class hierarchy with "Integer" selected. The right sidebar shows a list of methods, with "factorial" selected. The main editor area displays the source code for the factorial method:

```
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'
```

At the bottom of the editor, there is a status bar showing "1/6 [1]" and a checkbox for "Format as you read" which is currently unchecked.

In Pharo, the method belongs to the selected class

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

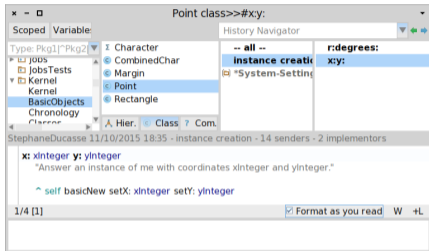
```
Game >> initializePlayers
self players
  at: 'tileAction'
  put: ( MITileAction director: self )
```

is equivalent to

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



Class Methods



- 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



and



in collaboration with



Inria 2020

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/>