Pharo MOOC

Stphane Ducasse, Damien Cassou, Luc Fabresse

2015

Contents

Part I
Week 1

1 Prise en main premier lments syntaxiques

Pharo: an immersive object-oriented programming language

http://www.pharo.org

D.Cassou - S. Ducasse - L. Fabresse
Univ. Lille / Inria / MinesTelecom

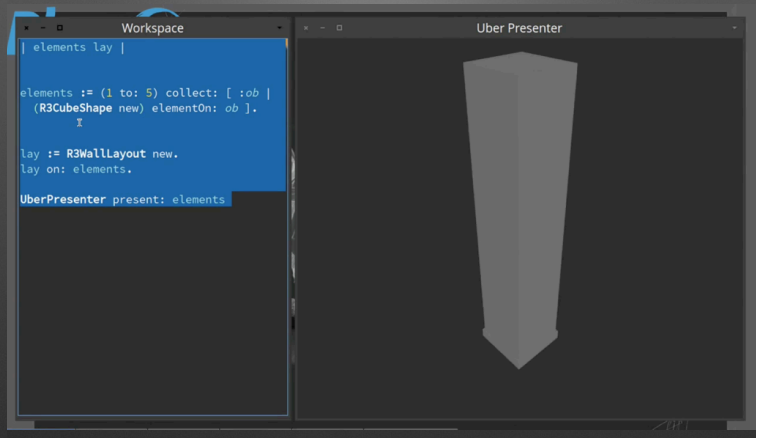
Objectives

- Be immersed into a live object world
- Deeply understand Object-Oriented Design and fundamental mechanisms
- Learn a set of design heuristics
- Work with real examples
- Develop and deploy a web app from start to end

Immersive

- Only think about objects
- Only manipulate objects
- Interact with living objects constantly

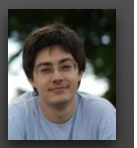
Immersive!

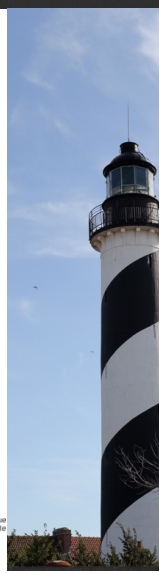


http://youtu.be/CuimMwuZiGA

About us

- Experts in Object-Oriented Programming
  - Traits (influenced Perl, Php, Scala)
- Experts in Object-Oriented Design
- Pharo core developers
- Authors of several books
  - Object-oriented reengineering patterns
  - Pharo by example, Deep into Pharo
  - Dynamic Web Development in Seaside
  - Entreprise Pharo: a web perspective





http://pharo.org

Pure & elegant
Fun, simple
Highly productive
Excellent for teaching
Empowering Tools
Full access

Pharo: an immersive object-oriented system

http://www.pharo.org

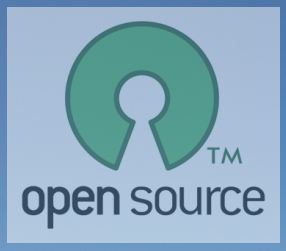
D.Cassou - S. Ducasse - L. Fabresse
Univ. Lille / Inria / MinesTelecom

Pharo?

Pharo!

- http://www.pharo.org
- System: Pure object language + full IDE
- Inspired by Smalltalk
- Powerful, elegant and fun to program
- Great community
- Living system under your fingers

License



Elegant!

- Full syntax on a postcard
- Simple but powerful object model

Complete Syntax on a Postcard

```
exampleWithNumber: x
    "A method that illustrates every part of Smalltalk method syntax"
    <menu>
    | 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
```

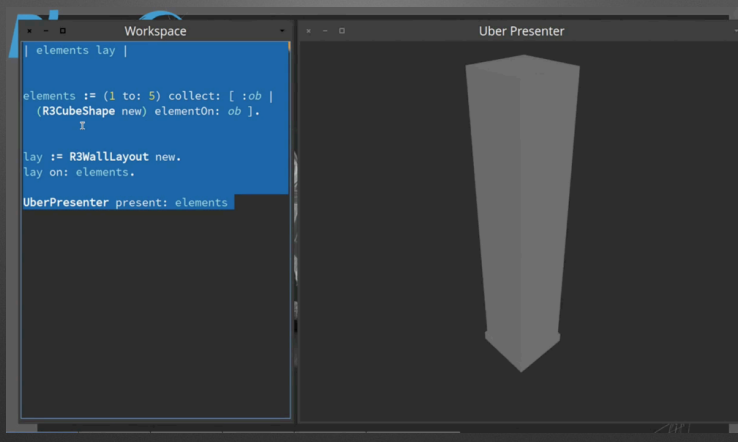
Object Model

- Dynamically typed
- **Everything** is an instance of a class
- All methods are public and virtual
- Attributes are protected
- Single Inheritance

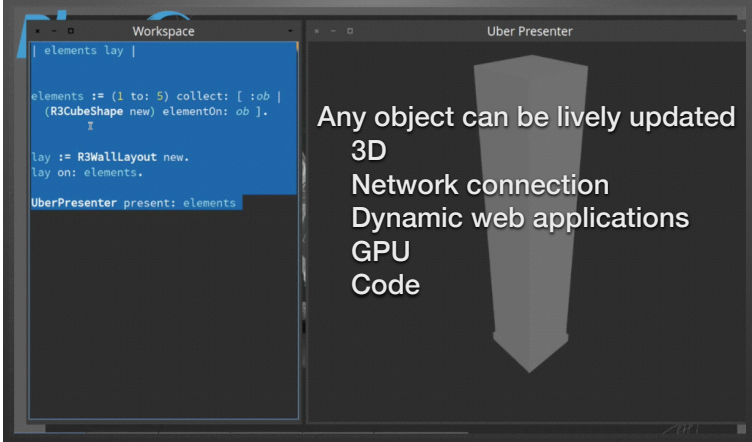
Immersive?

Immersive!

- Deep contact with objects
- Highly interactive programming sessions
- Reflective, inspectable



http://youtu.be/CuimMwuZiGA

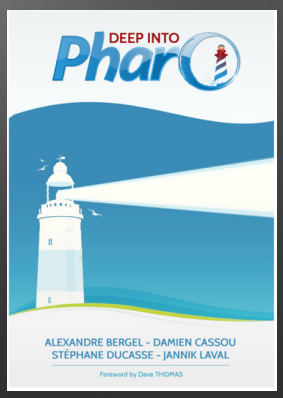


Pharo by Example

- Pharo by example http://www.pharobyexample.org
  - translated to french, merci!
  - translated to spanish, gracias!
  - translated to japanese, ありがとう!
- Currently updated to Pharo 40

Deep into Pharo

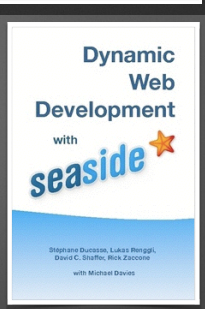
http://books.pharo.org



Entreprise Pharo: a Web perspective

http://books.pharo.org

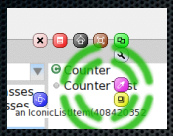
Other books



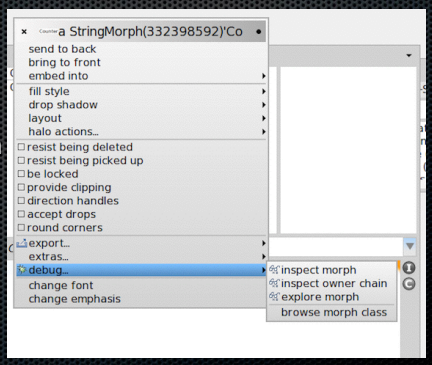
- Numerical Methods in Pharo
- Dynamic Web Development with Seaside
  - http://book.seaside.st

Learning from the system…

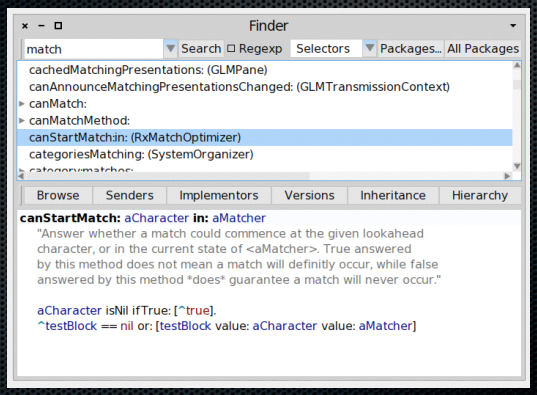
Click on it :)
Cmd+shift+option



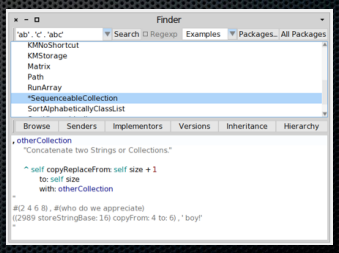
- Cmd-Shift+option



Finder :)



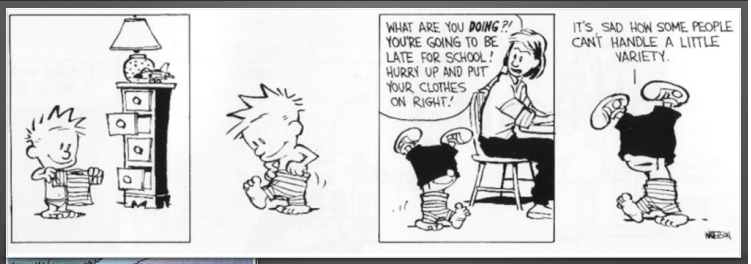
give examples and get the methods that works!

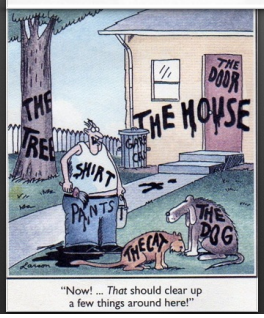


A system to learn advanced oo design

How to implement not?

- false not -> true
- true not -> false

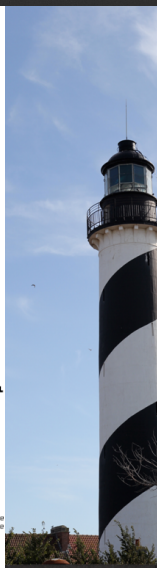




Pharo Humour

http://pharo.org

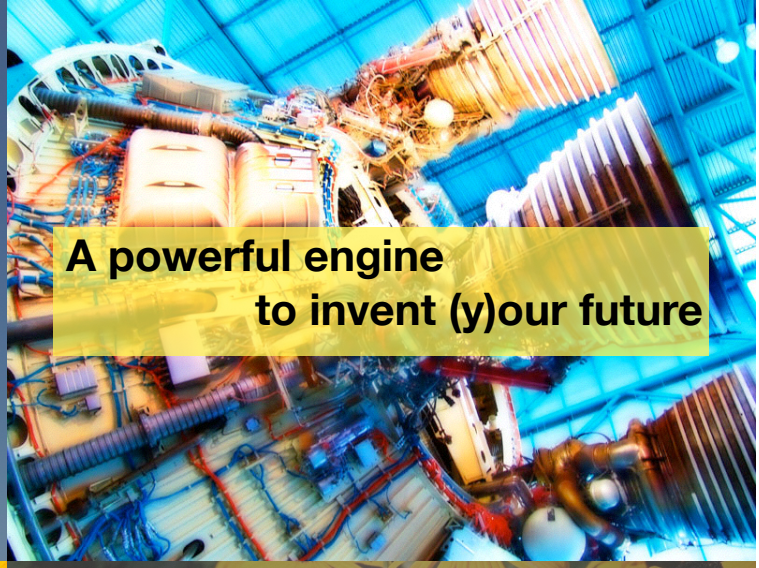
Fun, simple
Excellent for teaching
Pure & elegant
Highly productive
Empowering Tools
Full access



Pharo's vision

http://www.pharo.org

D.Cassou - S. Ducasse - L. Fabresse
Univ. Lille / Inria / MinesTelecom



An ecosystem where innovation/business bloom

Pharo's Teachers

• Uni. of Buenos Aires • Uni. of Bern • Uni. of Maroua • Uni. of Brussels • Ecole des Mines de Douai • Uni. de Savoie • Ivan Franko Nat. Uni. of Lviv • Czech Technical Uni. • CULS Prague • Uni. of Quilmes • Uni. of La Plata • Northern Michigan Uni. • Uni. Technologica Nacional (UTN) • Uni. Catholic of Argentina • Uni. of Santiago • Uni. Policnica de Catalunya • Uni. de Bretagne Occidentale • Uni. of Tomsk • Uni. of Fernhagen • IT University of Copenhagen • Uni. Cat del Sacro Cuore of Brescia • Uni. Lyon • Uni. Yaounde

Research Groups

Lafhis (AR)
Software Composition Group (CH)
CAR (FR)
RMOD (FR)
Ummisco (IRD)
Reveal (CH)
Lysic (FR)
CEA-List (FR)
Uqbar (AR)
OC (FR)
CCMI-FIT (CZ)
ASERG (BR)
Pleiad (CL)

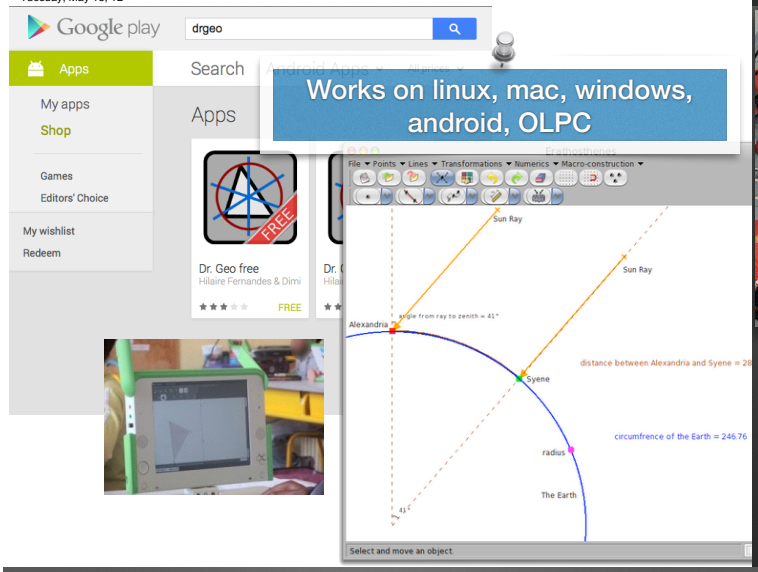
Some Companies

www.2denker.de
www.airflowing.com
www.beta9.be
www.bombardier.com
www.cmsbox.com
www.finworks.biz
seaside.gemstone.com
www.inceptive.be
www.majcon.de
www.mindclue.ch
www.miriamtech.com
www.netstyle.ch
www.panasoft.com
www.pinesoft.co.uk
www.promedmedical.net
www.sharedlogic.ca
www.smallworks.com.ar
www.trantaria.com
www.yesplan.be
www.synectique.eu
www.sorabito.com
www.objectprofile.com
www.pharocloud.com
debrispublishing.com
spesenfuchs.de
norizzk.com

Pharo Web Stack is Gorgeous

- Seaside (component, Javascript, REST)
- Zinc (HTTP, HTTPS, REST)
- Magritte Metamodelling (no form)
- Protocols/Encoding: Oauth, JSON, STON,...
- Database: noSql, mongoDB, riak, relational databases

Tuesday, May 15, 12



Pharo is our vehicule

Pharo is just starting, **every** single day we improve it

Pharo is open you can learn and help

Pharo is an Enabler

"One of the things that drew me to do the Delay refactoring, is simply that I could. That is, I was amazed that I could dig so deep so easily, see a path to improvement and effect change at a fundamental level. Excepting complexities with the Continuous Integration due to "changing the wheels on the car at 100km/h" (and one slip), it seems to have gone reasonably smoothly. That sense of mastery is seductive."

Enabler: Turtles all the way

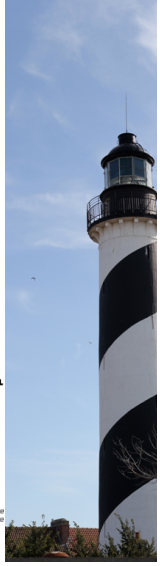
A. Bryant developed Seaside in Pharo's ancestor (he knew ruby, python, scheme, C, objective-C, …).

Because he could manipulate the stack behind the back of developers. Seaside (http://www.seaside.st) is based on on-demand stack reification.

Industrial Consortium

http://consortium.pharo.org

- Promote Pharo
- Sustain Pharo development
- Provide trustable visibility
- Provide support



http://pharo.org

Fun, simple
Excellent for teaching
Pure & elegant
Highly productive
Empowering Tools
Full access

Pharo Object Model in a Nutshell

Elegance and Simplicity

Stéphane Ducasse, Damien Cassou, and Luc Fabresse

Less is really more :)

- No constructors
- No static methods
- No type declarations
- No interfaces
- No packages/private/protected modifiers
- No parametrized types
- No boxing/unboxing
- still *really* powerful :)

Only Objects, Messages and Closures

- Only *objects*: mouse pointer, booleans, arrays, numbers, strings, windows, scrollbars, canvas, files, trees, compilers, sound, url, socket, fonts, text, collections, stack, shortcut, streams...
- and *messages* sent to these objects.
- *Closures* are like anonymous methods.

Simple and Uniform

- *everything* is an object, instance of a class.
- Classes are objects too!
- All computation between objects is done via *message passing*.
- There is only ONE method lookup for all objects:
  - Only late binding, only virtual calls
  - We use the term: *sending a message*

Pharo Object Model

- Instance variables are private to the object (instance-based) but accessible from subclasses.
- Methods are public.
- Single inheritance between classes.

Class Definition

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

- The class `Object` is asked to create the class `Point`,
- with the instance variables `x` and `y`,
- in the `Graphics` package.

Messages

Computation between objects is done via message sends:

```
Margin>>expandRectangle: aRectangle
  "Answer a rectangle whose size has been expanded by the receiver
      which represents each rectangle corner."

  | l r t b |
  l := aRectangle left − self left.
  r :=aRectangle right + self right.
  t := aRectangle top − self top.
  b := aRectangle bottom + self bottom.
  ^ Rectangle origin: l @t extent: ((r − l ) @ (b − t ))
```

Method Definition

- Defined in a browser (or by invoking the compiler)
- Methods are public
- Methods are virtual (*i.e.,* looked up at runtime)
- By default return `self`

```
Node >> dist: aPoint
  "Answer the distance between aPoint and the receiver."
  | dx dy |
  dx := aPoint x − x.
  dy := aPoint y − y.
  ^ (dx * dx + (dy * dy)) sqrt
```

Instance-Creation with Messages

- Messages sent to an instance

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

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

```
Monster new
Array new: 6
```

- Specific instance-creation message (sent to a class)

```
Tomagoshi withHunger: 10
```


Summary

- Everything is an object.
- Messages are sent to objects.
- Methods are late bound.
- Instances are created by sending messages to other objects, or classes.

Pharo Syntax in a Nutshell

Stéphane Ducasse

Getting a Feel About the Syntax

In this lecture we want to give you the general feel to get started

- Overview of simple syntactical elements
- Message composition
- Overview of closure syntax

The Complete Syntax

```
exampleWithNumber: x
  "This method illustrates the complete syntax except primitive
      invocation."
  <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.
```

- Message `getPng:` sent to the `ZnEasy` class
- Messages `asMorph` and `openInWorld`
- Class names start with an uppercase character
- Keyword message `getPng:`
- Unary messages `asMorph` and `openInWorld`

A Simple Method

```
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'
```


A Simple Method

```
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
- `factorial` is the method name
- `=`, `>`, `*` and `-` are binary messages
- `^` is for returning a value
- `factorial` is an unary message
- `ifTrue:` is a message sent to a boolean expression

Sending an HTTP Request

```
ZnClient new
  url: 'https://en.wikipedia.org/w/index.php';
  queryAt: 'title' put: 'Pharo';
  queryAt: 'action'  put: 'edit';
  get
```

- `new` is a message sent to a class
- `queryAt:put:` is a keyword message
- `get` is a unary message
- `;` sends all messages to the same receiver

Common Elements

- comment: `"a comment"`
- character: `$c` , `$@`
- string: `'a nice string'` `'l''idiot'`
- symbol: `#linux` `#+`
- array: `#(1 2 3 4)` `#('b' 'c')`
- integer: `1` , `2r101`
- real: `1.5` , `6.03e-34` , `2.4e7`
- fraction: `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:` ...

Language Construct Overview

- `"..."` comments
- `$` character
- `'...'` string
- `#` symbol or array
- `[ ]` block or byte array
- e, r number exponent or radix
- `<annotation>` method annotation
- `(...)` parenthesis
- `.` separator (not terminator)
- `;` cascade (several messages sent to the 1 object)

Essential Constructs

- temporary variable declaration: `| tmp |`
- variable assigment: `var := aValue`
- block (lexical closure): `[ expr... ]`
- block argument: `[ :var | ... ]`
- block temp. variable: `[ | var | ... ]`

Messages

3 kinds of messages

- unary message: `receiver selector`
- binary message: `receiver selector argument`
- keyword message: `receiver key1: arg1 key2: arg2`

Others

- cascade: `receiver selector ; selector ...`
- separator: `message . message`
- return: `^ expresion`

Conditionals are also Message Sends

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

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

Read their implementation, this is not magic!

Loops are also Message Sends

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

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

With Iterators

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

- The collection object is asked to do the iteration
- Many other messages implements loops: `timesRepeat:` , `do:` , `to:by:do:` , `whileTrue:` , `whileFalse:` , ...

Messages and Their Composition

3 kinds of messages

- Unary: `Node new`
- Binary: `1+2` , `3@4`
- Keywords: `aTamagoshi eat: #cooky furiously: true`

Message Priority

- (Msg) > unary > binary > keyword
- Same-Level messages: from left to right

Message Examples

- Creates a rectangle and asks a corner:

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

- Fetches and opens a picture:

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


Blocks

- A kind of anonymous method
- Are plain objects:
  - can be passed as method arguments
  - can be stored in variables
  - can be returned
- Look like maths functions:
  - `fct(x) = x*x+3` can be written `fct := [ :x | x * x + 3 ]`
  - `fct(2)` can be written `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

- Defined in a browser (or by invoking the compiler)
- Methods are public
- Methods are virtual (*i.e.,* looked up at runtime)
- By default return `self`

```
Node >> dist: aPoint
  "Answer the distance between aPoint and the receiver."
  | dx dy |
  dx := aPoint x − x.
  dy := aPoint y − y.
  ^ (dx * dx + (dy * dy)) sqrt
```

- Messages sent to an instance

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

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

Conclusion

- Compact syntax
- Few constructs but really expressive
- Support for Domain Specific Languages
- Mainly messages and closures

Runtime Architecture

Stéphane Ducasse

Execution Model

The Pharo virtual machine (VM) executes compiled code

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

Pharo code is compiled to bytecodes

- Bytecodes are platform neutral instructions

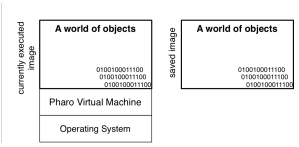
The virtual machine performs dynamically bytecodes to assembly generation

Virtual Machine

- Pharo.exe, Pharo.app... is the virtual machine.
- There are two modes:
  - from command-line or in interactive (UI) mode.
- It executes compiled code.
- Compiled code is packaged/stored in an *image*.
- The virtual machine only needs the *image* to execute programs.

About Image Files

- *.image* files act as cache of objects:
  - simple objects (points, strings, arrays)
  - but also **compiled** classes and **compiled** methods
- Each time we save the image, all objects are saved to disc.
- At startup we get back all the objects we saved.
- In particular the PC (program counter) is also saved and restored so frozen execution is



About Change Files

.changes file is a tape of all the changes performed in the system.

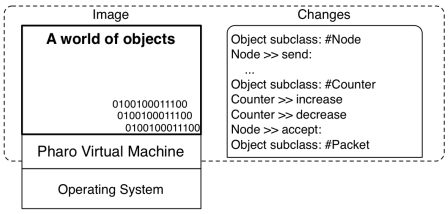
- It logs class creation/deletion, method addition/removal, actions...
- It is used to browse versions.
- It can replay/undo actions.

A change is associated to an image.

- when the browsers want to display class/method definition they look in the changes file associated to the current image.

About Change Files

- A change is associated to an image.
- The image contains all the objects in binary form. It can be executed without the changes file.
- The changes file simply contains the textual representation of the changes made to the image.



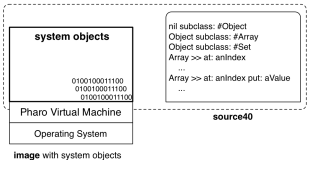
Save your code using a package and version control system

- Change and image are handy to develop
- But **they are not a software engineering artefact**
- Always have a loading script that takes an image, load your code, run the tests, build your application
- Usually we:
  - save code using a Version Control System (monticello, git)
  - use an integration server to build automatically applications.

About the Source/Changes File

PharoXX.sources

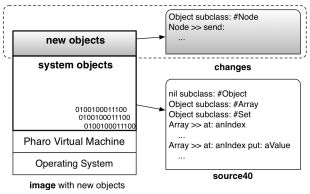
- Contains the *textual* definition of **system** classes and predefined objects
- Is read-only
- Created during release of new Pharo versions.
- Shared by all the users (images)



When you define new classes

During development or code loading,

- custom objects are compiled in the image
- custom definitions are added to the changes file
- Still you can browse the definition of the system class (stored in the *PharoXX.sources*)



New change management is coming

Pharo change system

- Getting improved
  - new recording mechanism
  - better replay
  - new tooling
- Will integrate better with Git and other modern distributed version control systems
- Offering new ways to produce images

Conclusion

- Powerful deployment
- Capture of living system
- Fast boot-time
- Support micro commits
- Will use modern version control

2 Exercices