

Reflection: Basic Introspection

Damien Cassou, Stéphane Ducasse and Luc Fabresse

W5S08



<http://www.pharo.org>



What Definitions Say: Reflection

Reflection is the ability of a program to manipulate as data something representing the state of the program during its own execution.

- **Introspection** is the ability for a program to **observe** and therefore **reason** about its own state.
- **Intercession** is the ability for a program to **modify** its own execution state or alter its own interpretation or meaning.



What Definitions Say: Reification

Reification is the process to transform implicit to explicit (objects)

- i.e., getting the stack as an object
- i.e., getting a class as an object



Pharo is a Reflective System

A system having itself as application domain and that is causally connected with this domain can be qualified as a reflective system Maes'87

- A reflective system has an internal representation of itself
- A reflective system is able to act on itself with the assurance that its representation will be causally connected (i.e. up to date)



Inspector

How does it access internal object state?

The image shows a code playground window titled "Playground" with the following code:

```
#(#January #February #March #April #May #June  
#July #August #September #October #November  
#December) asOrderedCollection
```

Below the code is the "Inspector on an OrderedCollection [12 items] (#January #February #March #April #...)" window. It has tabs for "Items", "Raw", and "Meta". The "Items" tab is selected, showing a table of internal state:

Variable	Value
{ } self	an OrderedCollection [12 items] (#January #February #March #April #...
▶ { } array	an Array [12 items] (#January #February #March #April #May #June #...
Σ firstIndex	1
Σ lastIndex	12

At the bottom of the inspector, the "self" variable is highlighted, showing its value as "self".

State Introspection

Accessing and setting object state

```
Object >> instVarAt: aNumber
```

```
Object >> instVarAt: aNumber put: anObject
```

```
Object >> instVarNamed: aString
```

```
Object >> instVarNamed: aString put: anObject
```

Accessing/Setting State Example

```
pt := 10@3.  
pt instVarNamed: 'x'.  
> 10  
pt instVarNamed: 'x' put: 33.  
pt  
> 33@3
```

- Violates encapsulation
- But this is for tools and during development

Accessing Class

```
Object >> class
```

```
'hello' class  
(10@3) class  
Smalltalk class  
Class class  
Class class class  
Class class class class
```


Querying the System

`OrderedCollection` allSuperclasses size.

`OrderedCollection` allSelectors size.

`OrderedCollection` allInstVarNames size.

`OrderedCollection` selectors size.

`OrderedCollection` instVarNames size.

`OrderedCollection` subclasses size.

`OrderedCollection` allSubclasses size.

`OrderedCollection` linesOfCode.

Querying the System

SystemNavigation default browseAllImplementorsOf: #,

The screenshot shows a window titled "Implementors of, [18]". The list of implementors is as follows:

Implementor	Package
AbstractFileReference (copying)	[FileSystem-Core]
FileReference (navigating)	[FileSystem-Core]
Announcement class (public)	[Announcements-Core]
Collection (copying)	[Collections-Abstract]
AnnouncementSet (adding)	[Announcements-Core]
Matrix (copying)	[Collections-Unordered]
SequenceableCollection (copying)	[Collections-Abstract]
RunArray (copying)	[Text-Core]
Exception class (exceptionselector)	[Kernel]
ExceptionSet (exceptionselector)	[Kernel]
IRSequence (copying)	[OpalCompiler-Core]
KMKeyCombination (combining)	[Keymapping-KeyCombinations]
KMKeyCombinationSequence (combining)	[Keymapping-KeyCombinations]
KMNoShortcut (combining)	[Keymapping-KeyCombinations]
KMStorage (accessing)	[Keymapping-Core]

The "Source" tab is selected, showing the following code:

```
, extension
^ self resolve, extension
```

Sending a Message by its Name

- How to implement a menu or a button?
- Need to send a message to a receiver given a message selector



Sending a Message by its Name

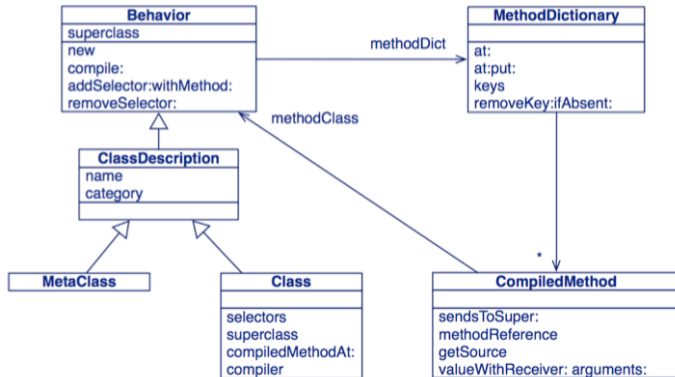
```
Object >> perform: aSymbol  
Object >> perform: aSymbol with: arg
```

- Asks an object to execute a message
- Normal lookup is performed

```
5 factorial  
5 perform: #factorial
```



Classes Hold Compiled Methods



```
OrderedCollection
└─ root: OrderedCollection
  └─ superclass: SequenceableCollection
    └─ methodDict: a MethodDictionary(#add:->a Co
      └─ #add:: a CompiledMethod (223)
        └─ header: #(17040896 #(#primitive: 0) #()
          └─ literal1: #addLast:
            └─ literal2: a MethodProperties
              └─ literal3: #OrderedCollection->OrderedCol
                └─ key: #OrderedCollection
                  └─ value: OrderedCollection
                    └─ 17: '<70> self'
                      └─ 18: '<10> pushTemp: 0'
                        └─ 19: '<E0> send: addLast:'
                          └─ 20: '<7C> returnTop'
                            └─ 21: 163
                              └─ 22: 135
                                └─ 23: 141
                                  └─ 24: 252
                                    └─ #add:after:: a CompiledMethod (4042)
                                      └─ #add:afterIndex:: a CompiledMethod (739)
                                        └─ self getSource a Text for 'add: newObject'
                                          └─ ^self addLast: newObject!
```

Executing a Compiled Method

`CompiledMethod >> valueWithReceiver:arguments:`

- no lookup performed

`(Integer>>#factorial)`
`valueWithReceiver: 5 arguments: #()`

`(SmallInteger>>#factorial)`
`valueWithReceiver: 5 arguments: #()`



Summary

- Just a part of the reflective power!
- Everything is an object and can be introspected
- Grab objects and talk to them
- Have a look at inspector code



A course by



and



in collaboration with



Inria 2016

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/>