



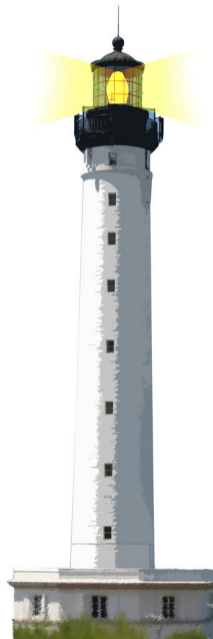
# Learning Object-Oriented Programming and Design with TDD

## Iterators

a Key Concept

Stéphane Ducasse

<http://stephane.ducasse.free.fr>



# What You Will Learn

- What is an iterator?
- Understand the power of iterators
- Offer an overview of iterators in Pharo



# Applying a message and collecting the results

For example, imagine we want to get whether a number is odd or not for a complete collection:

```
#(2 -3 4 -35 4) areOddNumbers  
>>> #(false true false true)
```

A possible limited solution

```
| result aCol|  
aCol := #(2 -3 4 -35 4).  
result := Array new: aCol size.  
1 to: aCol size do:  
  [ :each | result at: each put: (aCol at: each) odd ].  
result
```

## A better solution: Using collect:

collect: applies the block to each element and returns a collection (of the same kind than the receiver) with the results

- Works on any collection (Set, Array, OrderedCollection,...)
- collect: evaluates the block for each element (using value:)
- In the block, each element is sent odd
- collect: returns a new collection (of the same kind of the receiver) with all results
- [Think object] We ask the collection to do something for us

```
#(2 -3 4 -35 4) collect: [ :each | each odd]  
>>> #(false true false true)
```

## Another collect: Example

We want to know the absolute value of each element

```
 #(16 -11 68 19) collect: [:i | i abs ]
```

```
>>> #(16 11 68 19)
```

# Basic Iterators Overview

- do: (iterate)
- collect: (iterate and collect results)
- select: (select matching elements)
- reject: (reject matching elements)
- detect: (get first element matching)
- detect:ifNone: (get first element matching or a default value)
- includes: (test inclusion)
- and a lot more...

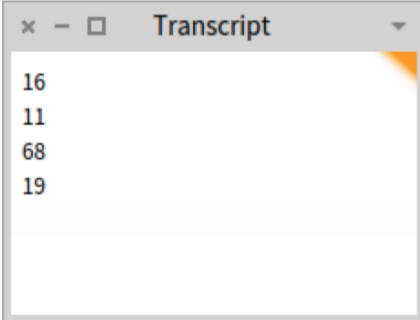


## do: an Action on Each Element

- Iterates on each elements
- Applies the block on each elements

```
#(16 11 68 19) do: [ :each | Transcript show: each ; cr ]
```

Here we print each element and insert a carriage return

A screenshot of a window titled "Transcript" with standard window controls (close, minimize, maximize). The window contains the output of the code block above, which are the numbers 16, 11, 68, and 19, each on a new line.

```
16  
11  
68  
19
```

# select: Elements Matching a Criteria

To select some elements, use `select`:

```
$(16 11 68 19) select: [ :i | i odd ]  
> $(11 19)
```



# With Unary Messages, No Block Needed

When a block is just one message, we can pass an unary message selector

```
#(16 11 68 19) select: [:i | i odd ]
```

is equivalent to

```
#(16 11 68 19) select: #odd
```

# reject: Some Elements Matching a Criteria

To filter some elements, use `reject`:

```
 #(16 11 68 19) reject: [ :i | i odd ]  
 > #(16 68)
```



# detect: The First Elements That...

To find the first element that matches, use `detect`:

```
$(16 11 68 19) detect: [ :i | i odd ]  
> 11
```

## detect:ifNone:

To find the first element that matches else return a value, use detect:ifNone:

```
#(16 12 68 20) detect: [ :i | i odd ] ifNone: [ 0 ]  
> 0
```



# Pharo code is Compact!

```
ArrayList<String> strings = new ArrayList<String>();  
for(Person person: persons)  
    strings.add(person.name());
```

is expressed as

```
strings := persons collect: [ :person | person name ]
```

- Yes in Java 8.0 it is finally simpler

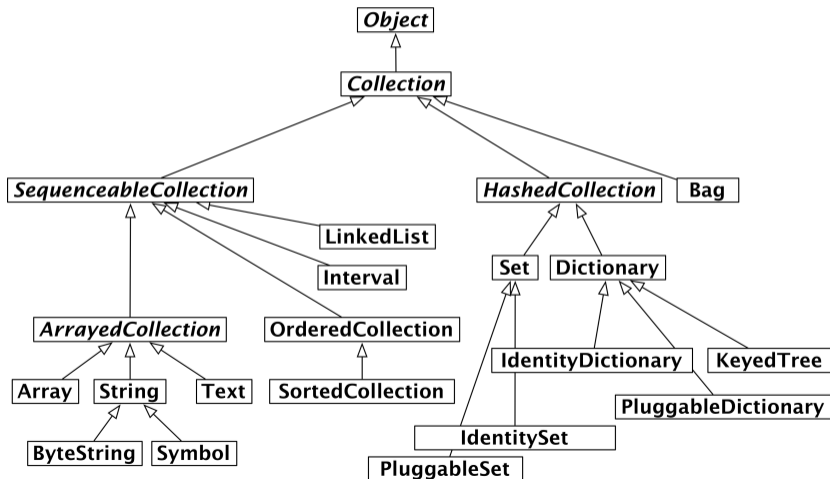
```
strings = persons.stream().map(person -> person.getName())
```

- But it is like that in Pharo since day one!
- Iterators are deep into the core of the language and libraries



# Part of the Collection Hierarchy

Iterators work polymorphically on the entire collection hierarchy. Below a part of the Collection hierarchy.



# Think objects!

- With iterators we **tell** the collection to **iterate on itself**
- As a client we do not have to know the internal logic of the collection
- Each collection can implement differently the iterator



# Some Powerful Iterators

- anySatisfy: (tests if one object is satisfying the criteria)
- allSatisfy: (tests if all objects are satisfying the criteria)
- reverseDo: (do an action on the collection starting from the end)
- doWithIndex: (do an action with the element and its index)
- pairsDo: (evaluate aBlock with my elements taken two at a time.)
- permutationsDo: ...

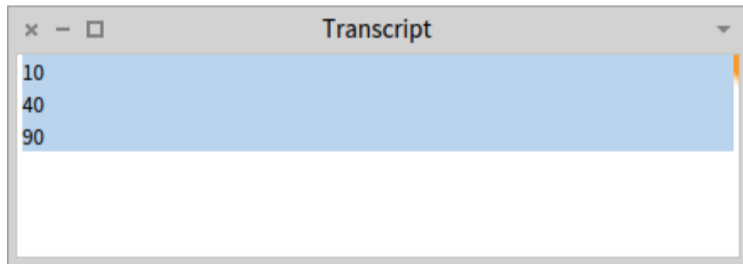




# Iterating Two Structures

To iterate with:do:

```
#(1 2 3)  
with: #(10 20 30)  
do: [:x :y | Transcript show: (y * x) ; cr ]
```



The screenshot shows a window titled "Transcript" with a light blue background. The text "10", "40", and "90" is displayed on three separate lines, representing the output of the code execution.

with:do: requires two structures of the same length

# A little challenge

```
#() xxx
```

```
>>> ""
```

```
#('a') xxx
```

```
>>> 'a'
```

```
#('a' 'b') xxx
```

```
>>> 'a, b'
```

```
#('a' 'b' 'c') xxx
```

```
>>> 'a, b, c'
```



# Use do:separatedBy:

```
String streamContents: [:s |  
  #('a' 'b' 'c')  
  do: [:each | s << each ]  
  separatedBy: [ s << ', ' ]  
]  
>>> 'a, b, c'
```

# Grouping Elements

To group elements according to a grouping function: `groupBy`:

```
#(1 2 3 4 5 6 7) groupBy: #even  
> a PluggableDictionary(false->#(1 3 5 7) true->#(2 4 6) )
```

# Flattening Results

How to remove one level of nesting in a collection? Use flatCollect:

```
#( #(1 2) #(3) #(4) #(5 6)) collect: [ :each | each ]  
> #( #(1 2) #(3) #(4) #(5 6))
```

```
#( #(1 2) #(3) #(4) #(5 6)) flatCollect: [ :each | each ]  
> #(1 2 3 4 5 6)
```

# Opening The Box

- You can learn and discover the system
- You can define your own iterator
- For example how `do:` is implemented?

```
SequenceableCollection >> do: aBlock
```

```
"Evaluate aBlock with each of the receiver's elements as the argument."
```

```
1 to: self size do: [:i | aBlock value: (self at: i)]
```



# Analysis

- Iterators are really powerful because they support polymorphic code
- All the collections support them
- New ones are defined
- Missing controlled navigation as in the Iterator design pattern



# Summary

- Iterators are your best friends
- Simple and powerful
- Enforce encapsulation of collections and containers





# Resources

- Pharo Mooc - W3S09 Videos <http://mooc.pharo.org>
- Pharo by Example <http://books.pharo.org>



A course by Stéphane Ducasse  
<http://stephane.ducasse.free.fr>

Reusing some parts of the Pharo Mocc by

Damien Cassou, Stéphane Ducasse, Luc Fabresse  
<http://mocc.pharo.org>



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/>