

Learning Object-Oriented Programming, Design and TDD with Pharos

Stéphane Ducasse

March 11, 2019

Copyright 2017 by Stéphane Ducasse.

The contents of this book are protected under the Creative Commons Attribution-ShareAlike 3.0 Unported license.

You are **free**:

- to **Share**: to copy, distribute and transmit the work,
- to **Remix**: to adapt the work,

Under the following conditions:

Attribution. You must attribute the work in the manner specified by the author or licensor (but not in any way that suggests that they endorse you or your use of the work).

Share Alike. If you alter, transform, or build upon this work, you may distribute the resulting work only under the same, similar or a compatible license.

For any reuse or distribution, you must make clear to others the license terms of this work. The best way to do this is with a link to this web page:
<http://creativecommons.org/licenses/by-sa/3.0/>

Any of the above conditions can be waived if you get permission from the copyright holder. Nothing in this license impairs or restricts the author's moral rights.



Your fair dealing and other rights are in no way affected by the above. This is a human-readable summary of the Legal Code (the full license):
<http://creativecommons.org/licenses/by-sa/3.0/legalcode>

Contents

| | |
|---|-----------|
| Illustrations | ii |
| 1 Solution of challenge yourself | 1 |
| 1.1 Challenge: Message identification | 1 |
| 1.2 Challenge: Literal objects | 2 |
| 1.3 Challenge: Kind of messages | 3 |
| 1.4 Challenge: Results | 4 |
| 1.5 Challenge: unneeded parentheses | 5 |
| Bibliography | 7 |

Illustrations



Solution of challenge yourself

1.1 Challenge: Message identification

```
[ 3 + 4

  receiver: 3
  selector: +
  arguments: 4
  result: 7

Date today

  receiver: Date
  selector: today
  arguments: _
  result: The date of today

#('' 'World') at: 1 put: 'Hello'

  receiver: #('' 'World')
  selector: at:put:
  arguments: 1 and 'Hello'
  result: #('Hello' 'World')

#(1 22 333) at: 2

  receiver: #(1 22 333)
  selector: at:
  arguments: 2
  result: 22
```

```
[
#(2 33 -4 67) collect: [ :each | each abs ]

receiver: #(2 33 -4 67)
selector: collect:
arguments: [ :each | each abs ]
result: #(2 33 4 67)

25 @ 50

receiver: 25
selector: @
arguments: 50
result: 25@50 (a point)

SmallInteger maxVal

receiver: the class SmalltalkInteger
selector: maxVal
arguments: _
result: returns the largest small integer

#(a b c d e f) includesAll: #(f d b)

receiver: #(a b c d e f)
selector: includesAll:
arguments: #(f d b)
result: true

true | false

receiver: true
selector: |
arguments: false
result: true

Point selectors

receiver: Point
selector: selectors
arguments: _
result: a long arrays of selectors understood by the class Point
]
```

1.2 Challenge: Literal objects

What kind of object does the following literal expressions refer to? It is the same as asking what is the result of sending the class message to such expressions.

1.3 Challenge: Kind of messages

```
[ 1.3
  > Float
  #node1
  > Symbol
  #(2 33 4)
  > Array
  'Hello, Dave'
  > String
  [ :each | each scale: 1.5 ]
  > Block
  $A
  > Character
  true
  > Boolean
  1
  > SmallInteger
```

1.3 Challenge: Kind of messages

Examine the following messages and report if the message is unary, binary or keyword-based.

```
[ 1 log
  > Unary
  Browser open
  > Unary
  2 raisedTo: 5
  > Keyword-based
  'hello', 'world'
  > Binary
```

```
[ 10@20
> Binary
point1 x
> Unary
point1 distanceFrom: point2
> Keyword-based
```

1.4 Challenge: Results

Examine the following expressions. What is the value returned by the execution of the following expressions?

```
[ 1 + 3 negated
> -2
1 + (3 negated)
> -2
2 raisedTo: 3 + 2
> 32
| anArray |
anArray := #('first' 'second' 'third' 'fourth').
anArray at: 2
> 'second'
#(2 3 -10 3) collect: [ :each | each * each]
> #(4 9 100 9)
6 + 4 / 2
> 5
2 negated raisedTo: 3 + 2
> -32
#(a b c d e f) includesAll: #(f d b)
> true
```


1.5 Challenge: unneeded parentheses

Putting more parentheses than necessary is a good way to get started. Such practice however leads to less readable expressions. Rewrite the following expressions using the least number of parentheses.

```
x between: (pt1 x) and: (pt2 y)
```

is equivalent to

```
x between: pt1 x and: pt2 y
```

```
((#(a b c d e f) asSet) intersection: #(f d b) asSet))
```

is equivalent to

```
#(a b c d e f) asSet intersection: #(f d b) asSet
```

```
(x isZero)
  ifTrue: [...]
```

```
(x includes: y)
  ifTrue: [...]
```

is equivalent to

```
x isZero
  ifTrue: [...]
```

```
(x includes: y)
  ifTrue: [...]
```

```
(OrderedCollection new)
  add: 56;
  add: 33;
  yourself
```

is equivalent to

```
OrderedCollection new
  add: 56;
  add: 33;
  yourself
```

```
((3 + 4) + (2 * 2) + (2 * 3))
```

is equivalent to

```
3 + 4 + (2 * 2) + (2 * 3)
```

```
(Integer primesUpTo: 64) sum
```

No changes

```
( 'http://www.pharo.org' asUrl) retrieveContents  
is equivalent to  
'http://www.pharo.org' asUrl retrieveContents
```

Bibliography