Dynamic Vs. Literal Arrays
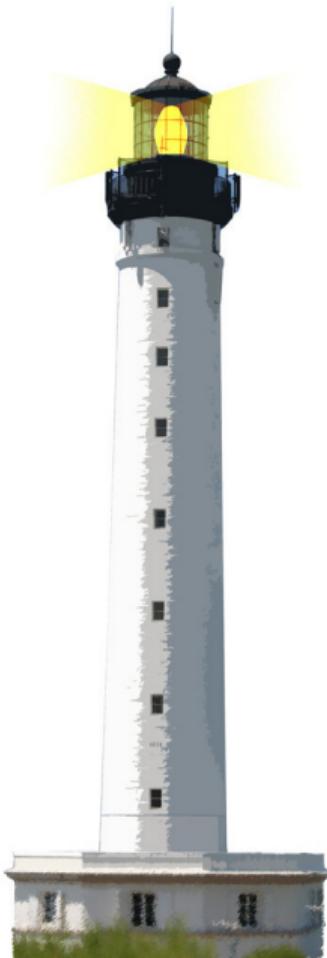
Damien Cassou, Stéphane Ducasse and Luc Fabresse

What You Will Learn

- Literal arrays are not created using messages
- Dynamic arrays are created at runtime using messages
- Still they are all instances of Array

Remember: Literal Arrays

Literal array definition can only contain objects that have a textual (literal) representation: numbers, strings, nil, symbols, boolean

```
#(45 'milou' 1300 true #tintin)
> #(45 'milou' 1300 true #tintin)
```

Literal arrays are instances of the class Array

```
#(45 38 1300 8) class
> Array
```

Literal/Dynamic Arrays

A literal array

```
#(45 38 'milou' 8)
> #(45 38 'milou' 8)
```

A dynamic array

```
Array with: 45 with: 38 with: 'milou' with: 8
> #(45 38 'milou' 8)
```

Both are Array instances

Dynamic Array Compact Syntax

Defining dynamic arrays is tedious

```
| array |
array := (Array new: 2).
array
   at: 1 put: 10 @ 10 ;
   at: 2 put: (Point x: 100 y: 200).
array
```

{ expression1 . expression2 } is syntactic sugar to create
dynamic arrays

```
{(10@20) . (100@200)}
{Point x: 10 y: 20 . Point x: 100 y: 200}
```

Literal Array Creation Time

Literal arrays are created at **compile time** by the parser
when the expression is read and not during the execution

```
| a |
a := 12.
#(a + 1 . 13)
> #(#a #+ 1 #'.' 13)
```

Dynamic arrays execute expressions

```
| a |
a := 12.
{a + 1 . 13}
> #(13 13)
```

Literal vs. Dynamic

{} executes expressions while #() not

```
{(10@20) . (10@20)}
> {(10@20) . (10@20)}

#((10@20) . (10@20))
> #(#(10 #@ 20) #'.' #(10 #@ 20))
```

```
{(10@20) . (10@20)} size
> 2

#((10@20) . (10@20)) size
> 3
```

Nested Arrays

() inside a literal array produces a nested literal array

```
#((10@20) (100@200))
> #(#(10 #@ 20) #(100 #@ 200))
```

```
#((10@20) . (100@200)) first
> #(10 #@ 20)
```

Summary

- Only one kind of Arrays
- Three ways
  - Literal syntax: #( ) (no message)
  - Using messages Array new: 3
  - Syntactic sugar: Dynamic { . . }

A course by

 and

in collaboration with