# A glimpse at essential collections

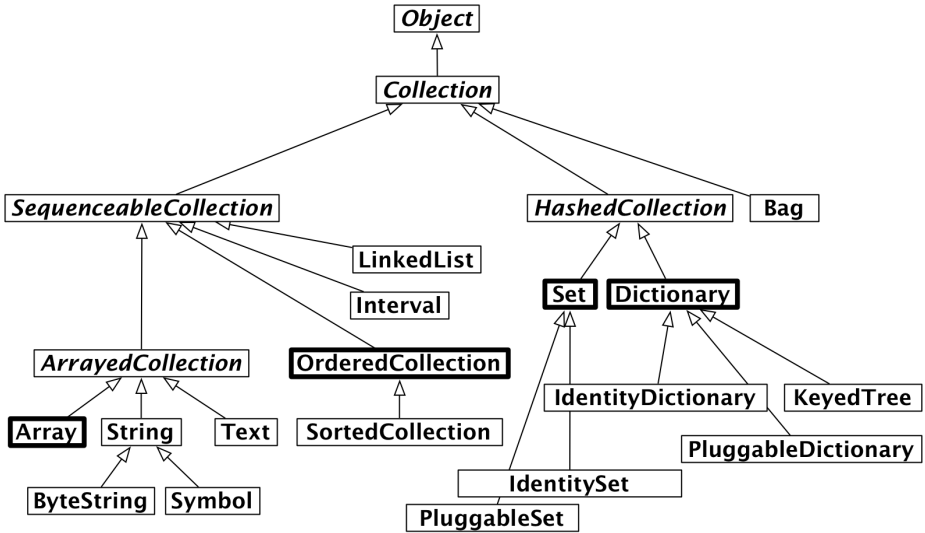
Stéphane Ducasse

http://stephane.ducasse.free.fr/ stephane.ducasse@inria.fr

# Collections In a Nutshell

- Pharo has a rich hierarchy of collection
- Common API: `size` , `do:` , `select:` , `includes:` , `collect:` ...
- Element 1 is at index 1
- Can contain any objects
- Most common ones:
  - `Set` (no duplicates)
  - `OrderedCollection` (dynamically growing)
  - `Array` (fixed size, direct access)
  - `Dictionary` (key-based)

# Essential Collection Inheritance Hierarchy In a Nutshell



# Common API

- Most iterators works on all collections

1. creation `with: anElement`, `with:with:`, `withAll: aCollection`
2. accessing: `size`, `capacity`, `at: anIndex`, `at: anIndex put: anElement`
3. testing `isEmpty`, `includes: anElement`, `contains: aBlock`, `occurrencesOf: anElement`
4. adding `add: anElement`, `addAll: aCollection`
5. removing `remove: anElement`, `remove: anElement ifAbsent: aBlock`, `removeAll: aCollection`
6. enumerating `do: aBlock`, `collect: aBlock`, `select: aBlock`, `reject: aBlock`, `detect: aBlock`, ...
7. converting `asBag`, `asSet`, `asOrderedCollection`, `asSortedCollection`, `asArray`

# Creating variable size objects

- Message `new` instantiates one object
- Message `new: size` creates an object specifying its size

```
Array new: 4
> #(nil nil nil nil)

Array new: 2
> #(nil nil)

(OrderedCollection new: 1000) capacity
> 1000

OrderedCollection withAll: #(7 3 13)
> an OrderedCollection(7 3 13)

Set withAll: #(7 3 13)
a Set( 7 3 13)
```

# Creation variants with default value

```
OrderedCollection new: 5 withAll: 'a'
> an OrderedCollection('a' 'a' 'a' 'a' 'a')
```

# Array

- Fixed size collection
- Direct access: `at:` / `at:put:`
- Has literal syntax: `#( ... )`

Returning the size of the collection

```
#('Calvin' 'hates' 'Suzie') size
> 3
```

Accessing the second element of the receiver

```
#('Calvin' 'hates' 'Suzie') at: 2
> 'hates'
```

```
#('Calvin' 'hates' 'Suzie') at: 55
> Error
```

# Literal Arrays

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

- Literal arrays are created by the parser: when the expression is read, i.e. when the method is compiled, not the method is executed.

# Literals Arrays

Literal arrays are equivalent to a dynamic version.

- A literal array

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

- A dynamic array

```
Array with: 45 with: 38 with: 1300 with: 8
> #(45 38 1300 8)
```

- `{ . . }` is syntactic sugar to create dynamic arrays

```
> Array with: 45 with: 38 with: 1300 with: 8
> #(45 38 1300 8)
> { 45 . 38 . 1300 . 8 }
```

# Literals vs. Dynamic Arrays

There is no message executed when a literal array is created.

```
#(45 + 38 1300 8)
> #( 45 #+ 38 1300 8)

#(45 + 38 1300 8) size
> 5
```

Dynamic arrays are created during the program execution.

```
Array with: 45 + 38 with: 1300 with: 8
> #( 83 . 1300 . 8)

{ 45 + 38 . 1300 . 8 }
> #( 83 . 1300 . 8)
```

# First element starts at 1

```
#('Calvin' 'hates' 'Suzie') at: 2
> 'hates'

#('Calvin' 'hates' 'Suzie') asOrderedCollection  at: 2
> 'hates'
```

# at: to access, at:put: to set

```
#('Calvin' 'hates' 'Suzie') at: 2 put: 'loves'
> #('Calvin' 'loves' 'Suzie')
```

# Variants

- An array of symbols:

```
#(#calvin #hobbes #suzie)
> #(#calvin #hobbes #suzie)
#(calvin hobbes suzie)
> #(#calvin #hobbes #suzie)
```

- An array of strings:

```
#('calvin' 'hobbes' 'suzie')
> #('calvin' 'hobbes' 'suzie')
```

# Collection can contain heterogenous objects

- Heterogenous

```
#('calvin' (1 2 3))
> #('calvin' #(1 2 3))

#('lulu' 1.22 1)
> #('lulu' 1.22 1)
```

# Byte Arrays

- Fixed size arrays.
- Elements are integers between 0 and 255.
- Has a special syntax: `#[ 1 2 255 ]`

# OrderedCollection

- Growing size
- `add:` , `remove:`

```
| ordCol |
ordCol := OrderedCollection new.
ordCol add: 'Reef'; add: 'Pharo'; addFirst: 'Pharo'.
ordCol
> an OrderedCollection('Pharo' 'Reef' 'Pharo')
ordCol add: 'Seaside'.
ordCol
> an OrderedCollection('Pharo' 'Reef' 'Pharo' 'Seaside')
```

```
#('Pharo' 'Reef' 'Pharo' 'Pharo') asOrderedCollection
> an OrderedCollection('Pharo' 'Reef' 'Pharo' 'Pharo')
```

# Set

- No duplicates
- Growing size
- `add:` , `remove:`

```
#('Pharo' 'Reef' 'Pharo' 'Pharo') asSet
> a Set('Pharo' 'Reef')
```

# Summary

- Easy to use collections.
- Common vocabulary.
- Simple conversion between them.