



/Présentation

Méthode
d'apprentissage et
analyses de librairies



/Sommaire



/01 /Méthode d'apprentissage

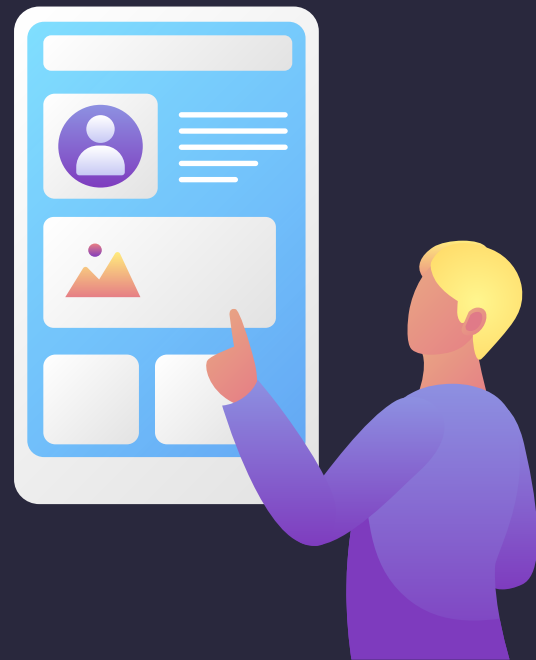
/02 /Analyse de données - Containers-OrderedSet

/03 /Analyse de données - Containers-LinkedList

/04 /Conclusion

/01

/Méthode d'apprentissage

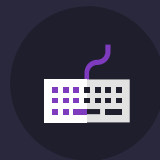


Deux manières selon la situation



/En cours

- En étant guidée par un cours



/Seule

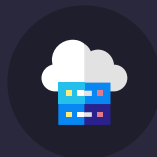
- En étant en autonomie

**Clé d'apprentissage :
appliquer concrètement en continuant d'apprendre**

Apprendre en cours



/Ecouter



/Essayer



/Faire un exercice



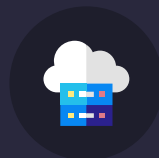
/Un autre exercice



Apprendre seule



**/Survoler la
documentation**



**/Commencer à
programmer**



**/Regarder les
ressources
disponibles**



/Continuer



/02

/Analyse de données - Containers-OrderedSet



Containers-OrderedSet

- Liste ordonnée d'éléments
- Ne peut contenir deux fois le même élément

```
| set1 set2 |  
set1 := CTOorderedSet withAll: #(3 5 2 3).  
set1.
```

| | Value |
|---|-------|
| 1 | 3 |
| 2 | 5 |
| 3 | 2 |

Du point de vue de l'utilisateur

/Initialisation

- `.new` : pour créer une liste vide
- `withAll: #()` : pour créer une liste en précisant les éléments

```
| set1 set2 |  
set1 := CTOrderedSet withAll: #(a b).  
set2 := CTOrderedSet withAll: #(b c).  
set1 union: set2.
```

Value

1 a

2 b

3 c

/Utilisation

Quelques méthodes :

- `addFirst`
- `addLast`
- `removeAt`
- `remove First`
- `sixth`
- `ninth`
- `difference`
- `intersection`
- `Union`
- ...

Point de vue de l'implémentation

Une classe de tests



Une classe de développements



Point de vue de l'implémentation

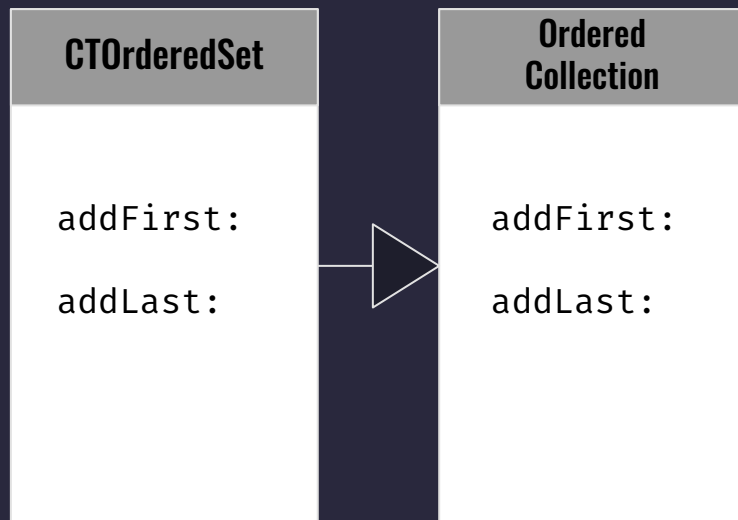
Quelques méthodes

- `addFirst:`
- `addLast:`
- `allSubsets:`



Point de vue de l'implémentation

Héritage



Point de vue de l'implémentation

Exemple de réutilisation

```
addLast: newObject  
    "Add newObject to the end of the receiver u  
  
    (self includes: newObject)  
        ifFalse: [ ^ super addLast: newObject ].  
    ^ newObject
```

Point de vue de l'implémentation

Méthodes non-réutilisés :

- isSubsetOf:
- isSupersetOf:
- allLargestSubsets:
- allSubsets:



Tests

- 85 Tests (**green**), majoritairement des tests positifs.
- Peu de tests négatifs.

Quels comportements quand des input invalides sont fournis ?

```
|test test2|  
test:= #( ) asOrderedSet.  
  
test intersection:nil.  
  
a CTOorderedSet()
```



SCENARIO 1

```
|test|  
test:= #(1) asOrderedSet.  
test at: #x.
```



RESULTAT 1

Instance of SmallInteger did not understand #isByteString

Stack

| Class | Method | Package |
|---------------------|------------------------|---------------------|
| ByteSymbol (String) | compare:with:collated: | Collections-Strings |
| ByteSymbol (String) | < | Collections-Strings |

Proceed Into Over Through Run to Restart Return Where is? Create Advanced Step

```
5 (ByteArray with: 97 with: 0 with: 0 with: 0) asString >>> true
6 "('abc' sameAs: 'aBc' asWideString) >>> true"
7 "('aBc' asWideString sameAs: 'abc') >>> true"
8 "('a000' asWideString ~= (ByteArray with: 97 with: 0 with: 0 with: 0) asString) >>> true"
9 "((ByteArray with: 97 with: 0 with: 0 with: 0) asString sameAs: 'Abcd' asWideString) >>> false"
10 "('a000' asWideString sameAs: (ByteArray with: 97 with: 0 with: 0 with: 0) asString) >>> false"
11
12 (string1 isByteString and: [string2 isByteString]) ifTrue: [
13     ^ ByteString compare: string1 with: string2 collated: order].
14 "Primitive does not fail properly right now"
15 ^ String compare: string1 with: string2 collated: order
```

ByteSymbol (x)

```
|test|  
test:= #(1) asOrderedSet.  
test at: nil.
```



RESULTAT 2

#< was sent to nil

Stack

| Class | Method | Package |
|----------------------------------|--------|--------------------------|
| CTOrderedSet (OrderedCollection) | at: | Collections-Sequenceable |
| UndefinedObject | Dolt | - |

```
4  "((OrderedCollection new add: 34; yourself) at: 1) >>> 34"
5  "(#(40 41 42) asOrderedCollection at: 1) >>> 40"
6  "(#(40 41 42) asOrderedCollection at: 2) >>> 41"
7  "(#(40 41 42) asOrderedCollection at: 3) >>> 42"
8
9  | index |
10 anInteger < 1
11     ifTrue: [ self errorSubscriptBounds: anInteger ].
12 (index := anInteger + firstIndex - 1) > lastIndex
13     ifTrue: [ self errorSubscriptBounds: anInteger ].
14 A array at: index
```

a CTOrderedSet [1 item] (1)

Boundary testing & Coverage

- Accès à l'index 0 ?
- Accès à un index supérieur à la taille du **set**, ...
- Coverage : 91,67%
- Une méthode non testée : **asOrderedSet**



Tests proposés

testAsOrderedSetWithDuplicate

```
"Creating an OrderedSet from a Collection with duplicated "  
| col expected|  
col := #(1 2 2 3 4) asOrderedCollection asOrderedSet .  
expected := CTOorderedSet newFrom: #(1 2 3 4).  
self assert: col equals: expected.
```

testAsOrderedSetEmpty

```
"Creating an OrderedSet from an empty Collection"  
| col expected|  
col := #() asOrderedCollection asOrderedSet .  
expected := CTOorderedSet newFrom: #().  
self assert: col equals: expected.
```

testAsOrderedSet

```
"Creating an OrderedSet from a Collection"  
| col expected|  
col := #(1 2 3 4) asOrderedCollection asOrderedSet .  
expected := CTOorderedSet newFrom: #(1 2 3 4).  
self assert: col equals: expected.
```



/03

/Analyse de données - Containers-LinkedList



Introduction LinkedList

Définition et types

Collection ordonnée d'éléments dont la structure comporte une valeur et un lien vers l'élément suivant.

On distingue :

Les listes simplement chaînées et les listes doublement chaînées.

Introduction LinkedList

Utilisation (Pharo) - LinkedList

```
|list|
```

```
list := LinkedList new.  
list add:10.
```

=>

```
|list|
```

```
list := LinkedList newFrom: #(10).  
list linkAt: 1.
```

a ValueLink (ValueLink(10))

Raw

Breakpoints

Meta

Variable

Value

self

ValueLink(10)

nextLink

nil

Σ value

10

add: after: - add: afterLink: - add: before: - add: beforeLink:
- addLast: - addFirst:

Actions sur la liste


```
|test|  
test:= LinkedList new.  
test add:10.  
test add:20.  
test add:21.  
test add:30.  
test addFirst: #start.  
test removeLast .  
test removeFirst .  
  
test select[:elem| elem even]
```

```
|test|  
test:= LinkedList new.  
test add:10.  
test add:20.  
test add:21.  
test select[:elem| elem even]
```

a LinkedList(10 20)



- ▲ add:
- add:after:
 - add:afterLink:
 - add:before:
 - add:beforeLink:
 - addFirst:
 - addLast:
- at:
- at:put:
 - at:putLink:
- ▲ collect:
- ▲ collect:thenReject:
- ▲ collect:thenSelect:
- containsCycle
- ▲ copyWith:

instance side  ☐

- ☒ CTLinkedList
- ☐ SequenceableCollection
- ☐ Collection
- ☐ Object
- ☐ ProtoObject
- accessing
- adding
- copying
- enumerating
- ☒ private
- removing
- testing
- overrides

- do:
- ▲ first
 - firstLink
- indexOf:startingAt:ifAbsent:
- isEmpty
- last
 - lastLink
- linkAt:
- linkAt:ifAbsent:
- linkOf:
- linkOf:ifAbsent:
- linkOn:
- linksDo:
- ▲ postCopy



Introduction LinkedList

Utilisation (Pharo)

- DoubleLinkedList

```
|list|  
list := CTDoublLinkedList new.  
  
list add:10.  
list add:1 afterLink:(list linkAt:1).  
list add:20 beforeLink:(list linkAt:2).
```

=>

a CTDoublLinkedList(10 20 1)

| Raw Breakpoints Meta | |
|----------------------|------------------------------|
| Variable | Value |
| self | a CTDoublLinkedList(10 20 1) |
| head | a CTDoublLink (10) |
| tail | a CTDoublLink (1) |

Actions DoubleLinkedList

```
|test|
test:= DoubleLinkedList new.
test addAll: #(2 54).

test removeFirst.
```

=>

```
|test|
test:= DoubleLinkedList new.
test add: #(2 54).

test
```

| Variable | Value |
|--------------|--------------------|
| self | a DoubleLinkedList |
| head | a DoubleLink |
| self | a DoubleLink |
| Σ value | 54 |
| nextLink | nil |
| previousLink | nil |
| tail | a DoubleLink |
| self | a DoubleLink |
| Σ value | 54 |
| nextLink | nil |
| previousLink | nil |



```
T = (TCTLinkedList)
  add:
  add:afterLink:
  add:beforeLink:
  addAll:
  addFirst:
  addLast:
  asArray
  T at: (TCTLinkedList)
  collect:
  T collect:thenSelect: (TCTLinkedList)
  do:
  T do:separatedBy: (TCTLinkedList)
  emptyCheck
  first
```

```
instance side
  [x] CTDoubeLinkedList
  [ ] Object
  [ ] ProtoObject
  accessing
  adding
  converting
  enumerating
  printing
  [x] private
  removing
  testing
```

```
firstLink
  T hasEqualElements: (TCTLinkedList)
  T hash (TCTLinkedList)
  T ifNotEmpty: (TCTLinkedList)
  includes:
  isEmpty
  T isEmpty (TCTLinkedList)
  last
  lastLink
  T linkAt: (TCTLinkedList)
  T linkAt:ifAbsent: (TCTLinkedList)
  linkOn:
  linksDo:
  T printElementsOn: (TCTLinkedList)
  T printOn: (TCTLinkedList)
```



Point de vue de l'implémentation

Deux classes de
tests



Cinq classes de
développements



Point de vue de l'implémenteur

Quelques méthodes

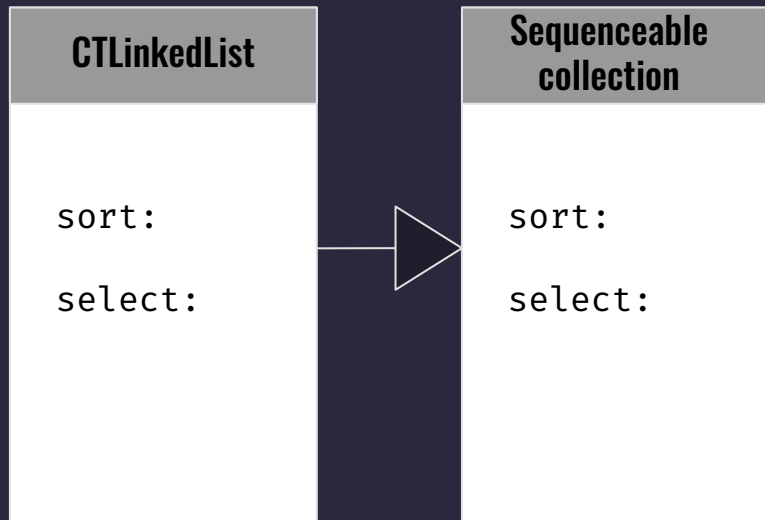
→ `addFirst:`

→ `add:`

→ `removeFirst:`



Point de vue de l'implémentation



Point de vue de l'implémentation

Questionnements :

→ Pourrait-il y avoir plus de réutilisations de méthodes ?



Point de vue de l'implémentation

Questionnements :

→ Add: qui appelle addLast:

```
add: aLinkOrObject  
    "Add aLink to the end of the receiver's list. Answer aLink."  
  
    ^self addLast: aLinkOrObject
```

Point de vue de l'implémentation

Questionnements :

→ Utilisation de trait au lieu de l'héritage ?

```
T hasEqualElements: (TCTLinkedList)  
T hash (TCTLinkedList)  
T ifNotEmpty: (TCTLinkedList)
```

Tests

- Deux classes de tests : `CTDoubleLinkedListTests` et `CTLinkedListTest`
- 273 tests au total, dont 2 qui ne passent pas (couverture de 91%)
 - ◆ `testTAddWithOccurrences`
 - ◆ `testCopyReplaceAllWithManyOccurrence`
- Report de bug pour `testTAddWithOccurrences`
- Librairie déjà existante dans Pharo

/04

/Conclusion





Merci pour votre attention

› Théo, Ronick, Noémie ‹

