

# Présentation



---

Gaci Noufel

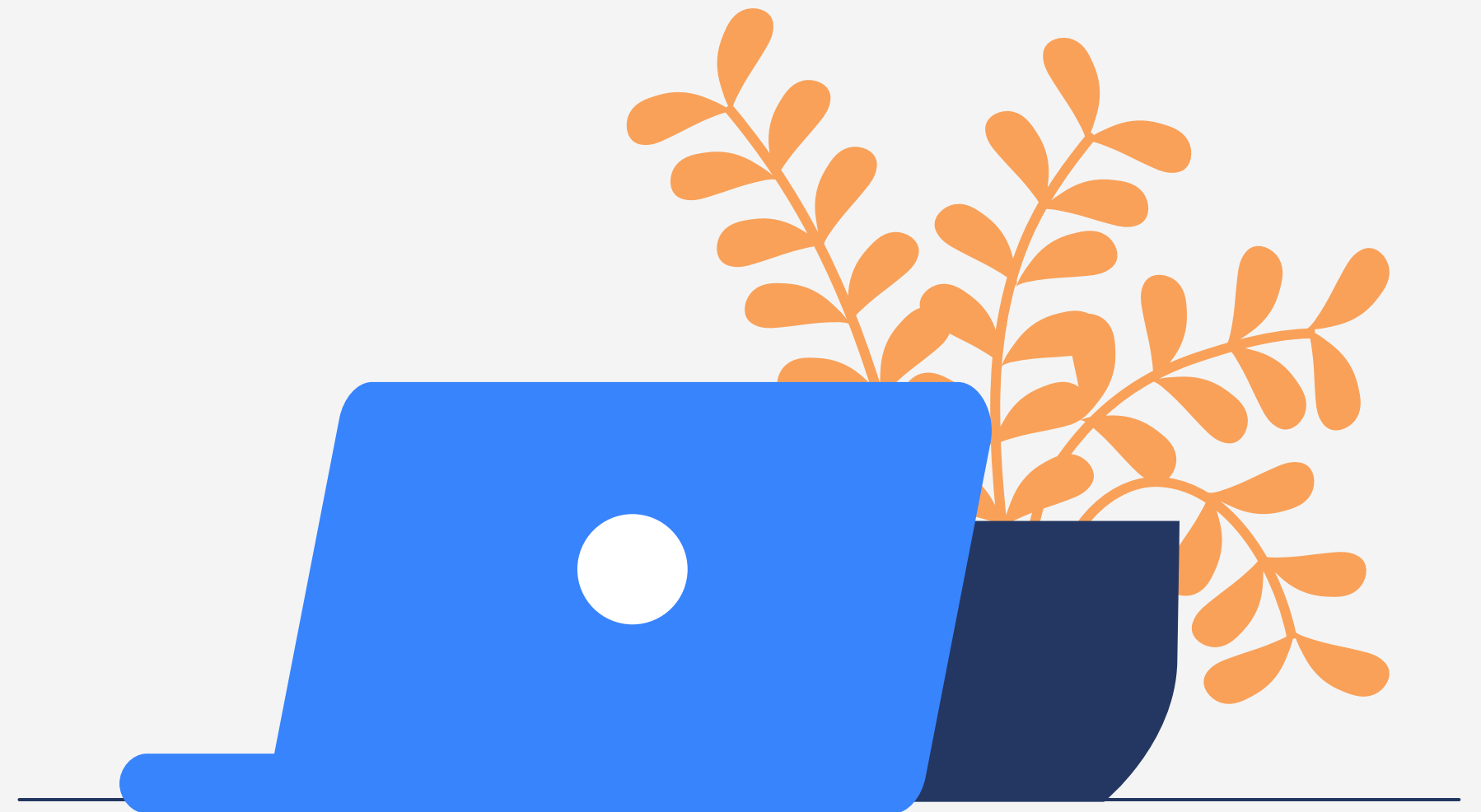
# Sommaire

## 1. Méthode d'apprentissage

## 2. Containers OrderedSet

- Définition
- Point de vue de l'utilisateur
- Point de vue de l'implémentation
- Les Tests

## 3. Conclusion



# Méthode d'apprentissage

En cours



40%

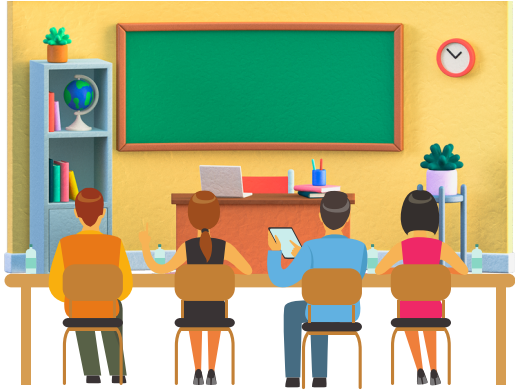
Seul



60%

# Plusieurs méthodes pour 1 seul objectif

## En cours



- Ecouter attentivement
- Relire les cours
- Demander d'aide des collègues , prof
- Comprendre juste le fondamentale

## Seul



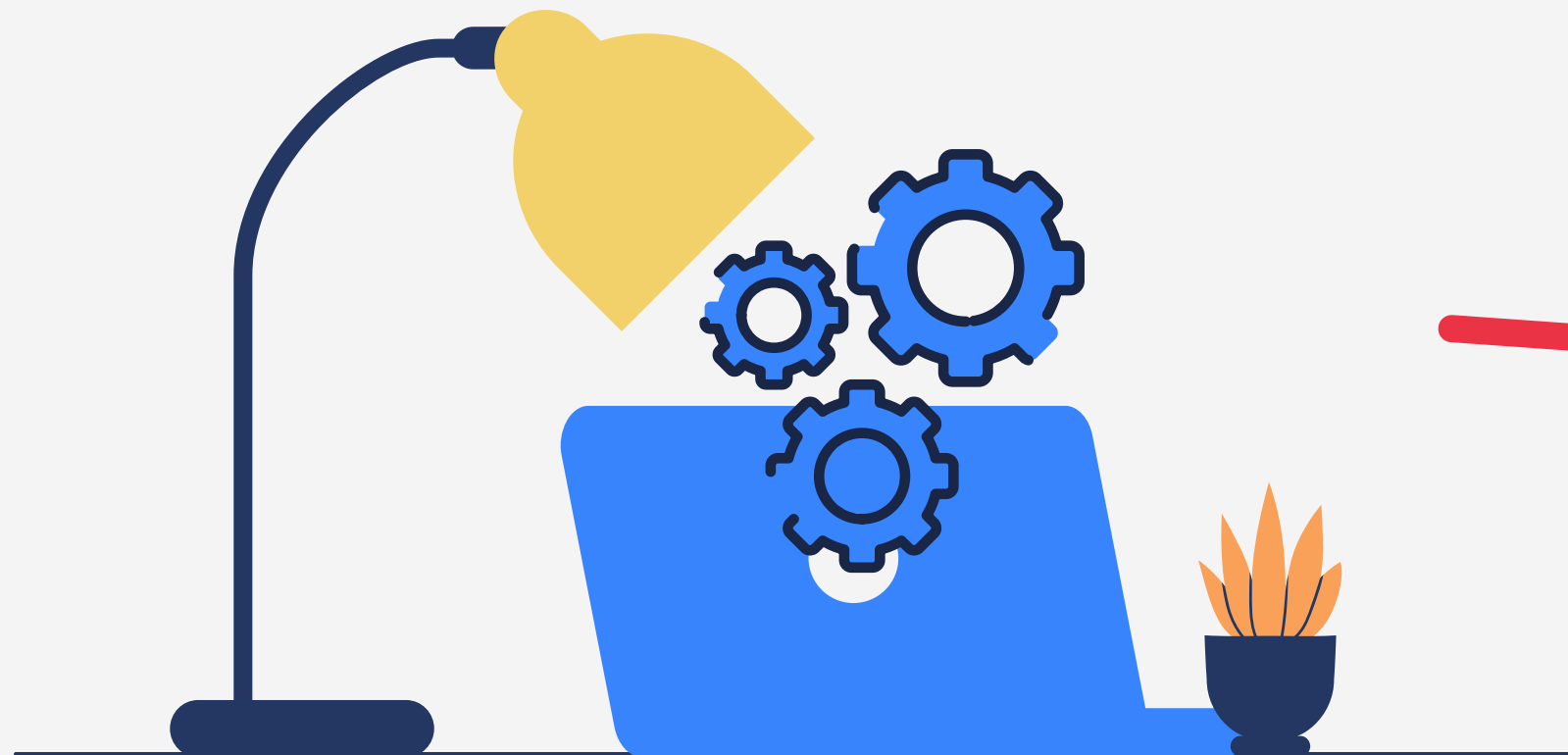
- Consulter la documentation
- Chercher sur internet
- Vidéos sur Youtube en différentes langues
- Comparer avec d'autres langages métrisés
- Programmer et apprendre des erreurs



# 2.Containers OrderedSet

## Définition

- ✗ est une structure de données mutables qui est un hybride d'une OrderedCollection et un Set traditionnel.
- ✗ Pour créer une telle nouvelle collection avec la librairie existante, on a deux choix soit on hérite de Set et dupliquez le code de OrderedCollection , ou nous héritons de OrderedCollection et dupliquons le code de Set . Dans les deux cas, nous devons dupliquer le code car l'héritage multiple n'existe pas dans Pharo



→ `OrderedCollection subclass: #CTOrderedSet  
instanceVariableNames: ''  
classVariableNames: ''  
package: 'Containers-OrderedSet'`

❌ Il se souvient de l'ordre de ses entrées.

❌ Chaque entrée a un numéro d'index qui peut être recherché.

```
myAccount := CTOorderedSet new.  
myAccount add: 20; add: 55; add: -22.  
myAccount at:2.
```

Integer	Raw	Breakpoints
key	value	
decimal	55	

Transc005

❌ Orderedset, First index est 1 , non 0.

```
1 myAccount:= #(100 200 300)  
  asOrderedSet.  
2 myAccount at: 0.
```

```
1 errorSubscriptBounds: index  
2 "Create an error notification that an  
3  
4 SubscriptOutOfBounds signalFor: index
```

a CTOorderedSet [3 items] (1...

Type	Variable	Value
implicit	{ } self	a CTOorderedSet [3 it
arg	Σ index	0
inst. var	{ } array	an Array [3 items] (10
inst. var	Σ firstIndex	1

Transc001 1

Transc002 2

Transc003 3

Transc004 4

# Point de vue de l'implémentation



Comment se passent les choses, d'abord?



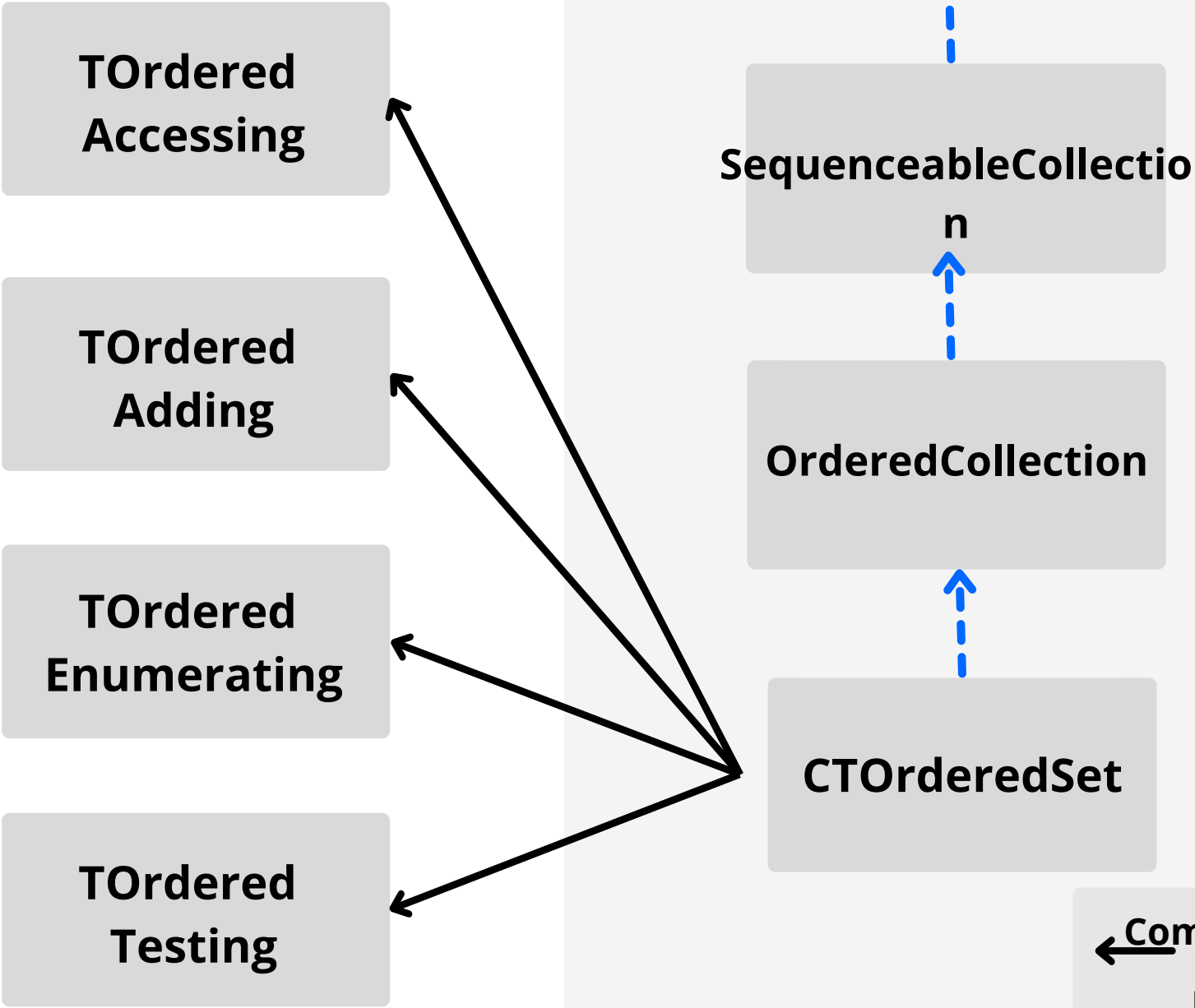
# Comment se passent les choses, d'abord?

## Traits :

les traits sont des ensembles de méthodes conçues pour être réutilisées en groupe dans les classes. pour définir des comportements supplémentaires dans une classe, la classe peut composer un ensemble de traits.

```
employeeSet:= CTOorderedSet newFrom:{
'id1'->200. 'id2'->400. 'id2'-> 400}.
employeeSet at:2 put:'id3'->500.
^employeeSet.
```

Items	Raw
Value	
1	'id1'->200
2	'id3'->500



← Composé de :  
←-- Hérite de :

CTDuplicateException !

{ } CTOrderedSet !

{ } Collection

instance side

as yet unclassified

accessing

adding

enumerating

testing

overrides

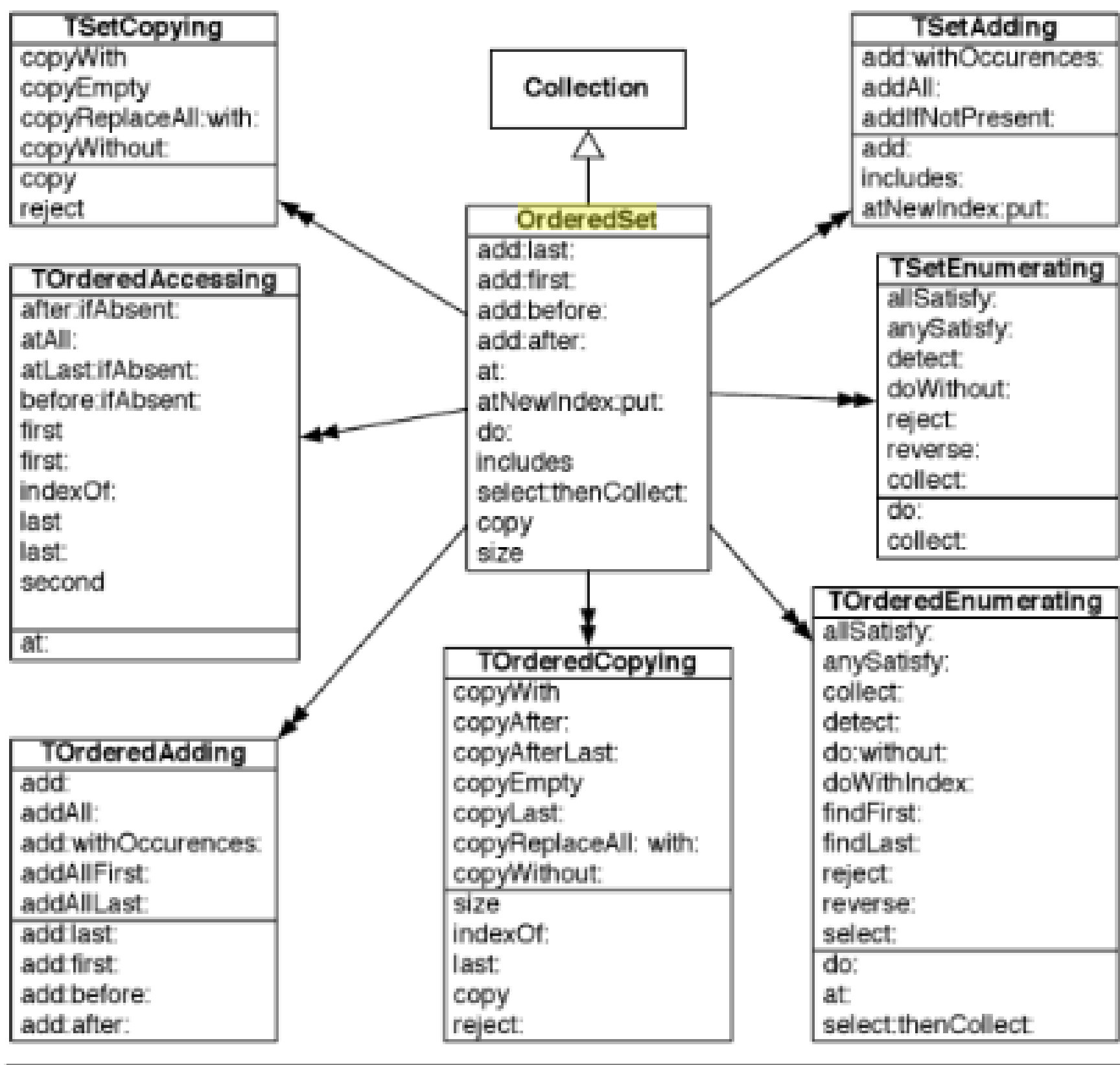
- addFirst:
- addLast:
- allLargestSubsets
- allSubsets
- at:put:
- difference:
- intersection:
- isSubsetOf:
- isSupersetOf:
- union:

```
employeeSet:= CTOorderedSet
withAll:{'id1'->200. 'id2'->400}.
employeeSet addFirst: 'id0'->980.
^employeeSet.
```

Items	Raw
Value	
1	'id0'->980
2	'id1'->200
3	'id2'->400



# Suite ...



accessing

adapting

adding

comparing

converting

copying

enumerating

● *addAll: (OrderedCollection)*

● *addAllFirst: (OrderedCollection)*

● *addAllFirstUnlessAlreadyPresen*

● *addAllLast: (OrderedCollection)*

▼ *addAssignToFloatArray: (Collect*

● *addFirst:*

● *addFirst: (OrderedCollection)*

Avec Orderedset, nous disposons  
de plusieurs méthodes :

Grace au mécanisme de traits, nous pourrons partager le  
comportement entre des classes distinctes, les traits sont  
des collections des méthodes qui sont utilisés entre les  
classes sans héritage.

Map de traits utilisés par OrderedSet

# Point de vue de l'implémentation



## Exemple d'une méthode réutilisée

printing

private

removing

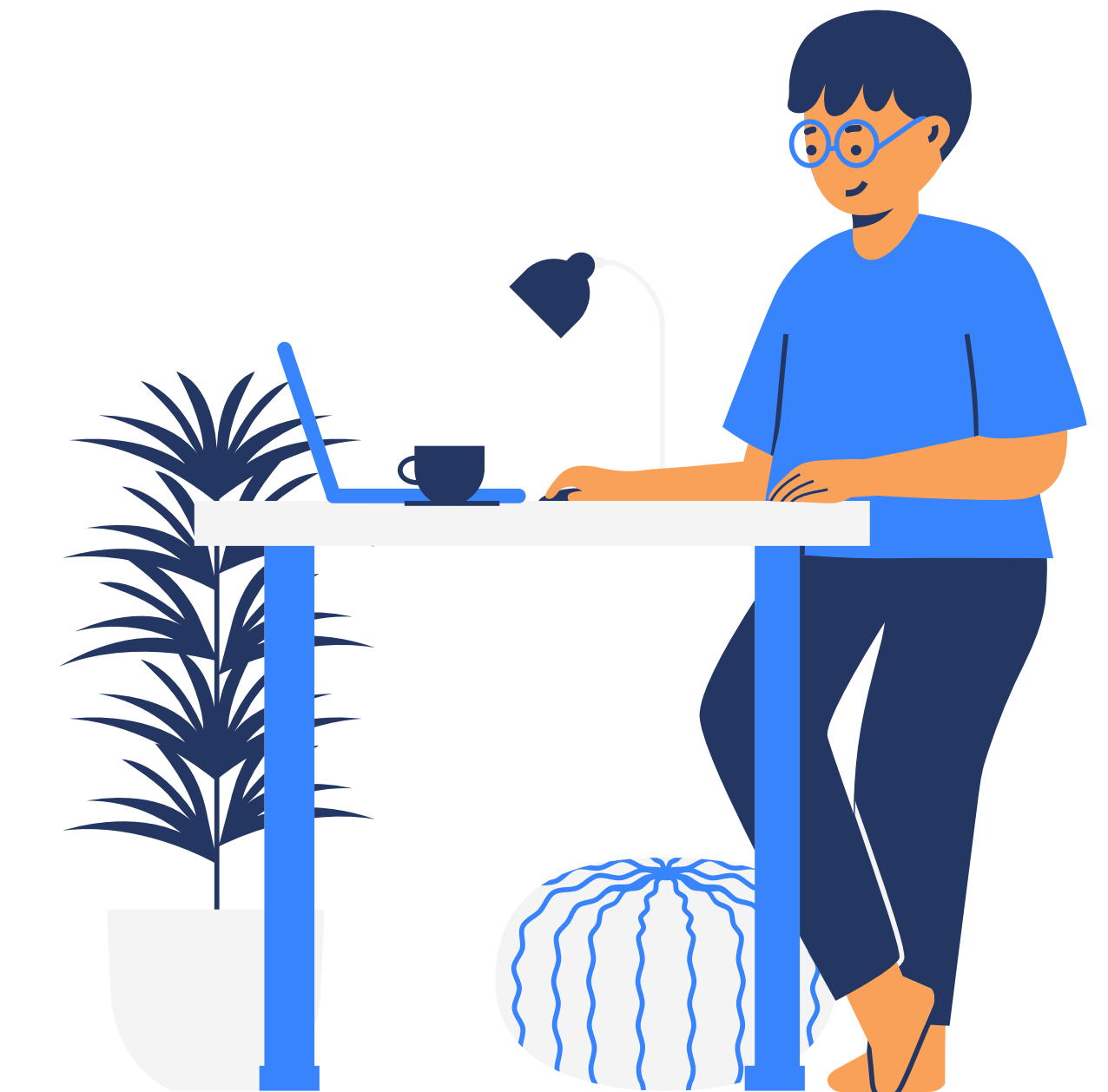
removeAll(Collection)

removeAll(OrderedCollection)

removeAll(Collection)

```
1 myAccount := CTOorderedSet new.  
2 myAccount add: 120.  
3 myAccount add: 500.  
4 myAccount add: 300.  
5 myAccount removeAll.  
6 ^myAccount .
```

Items	Raw
Index	



## 01 Accessing

at: put:

```
myAccount := CTOorderedSet withAll:
{'paul'->80. 'michel'->250.
'Ali'->120}.
myAccount at: 2 put: 'michel'->177 .
^myAccount .
```

Items	Raw
⚡ ⚡ Value	
1 'paul'->80	
2 'michel'->177	
3 'Ali'->120	

## 02 Adding

addFirst:

```
myAccount := CTOorderedSet withAll:
{'paul'->80. 'michel'->250 }.
myAccount addFirst: 'Adel'->177 .
myAccount addLast: 'Lio' -> 40.
^myAccount .
```

Items	Raw
⚡ ⚡ Value	
1 'Adel'->177	
2 'paul'->80	
3 'michel'->250	
4 'Lio'->40	

## 03 Enumerating

difference :

```
s1:= CTOorderedSet withAll: #(aa cc).
#(dd aa cc ) difference: s1.
```

Items
⚡ ⚡ Value
1 dd

difference:

## 04 Testing

isSubsetOf:

```
s1:= #(a c) asOrderedSet.
^s1 isSubsetOf: #(d a c ).
```

Raw	Breakpoints	Meta	
⚡ Variable			⚡ Value
Ⓢ self			true

Autres exemples d'utilisation :

# Test

85 test en **vert**

Des test réalisés pour assurer le bon fonctionnement de Set et Collection  
Concernant la duplication et ...etc

test de 'asOrderedset' **non réalisé**

test de 'copyFrom: \_ to : \_ ' **non réalisé**

```
myAccount:= #(100 200 300 400 500)
asOrderedSet.
myAccount2 := CTOorderedSet new.
myAccount2:=myAccount copyFrom: 2 to:
4.
^myAccount2. |
```

Items	
	Value
1	200
2	300
3	400





# Point de vue de l'implémentation


## Test

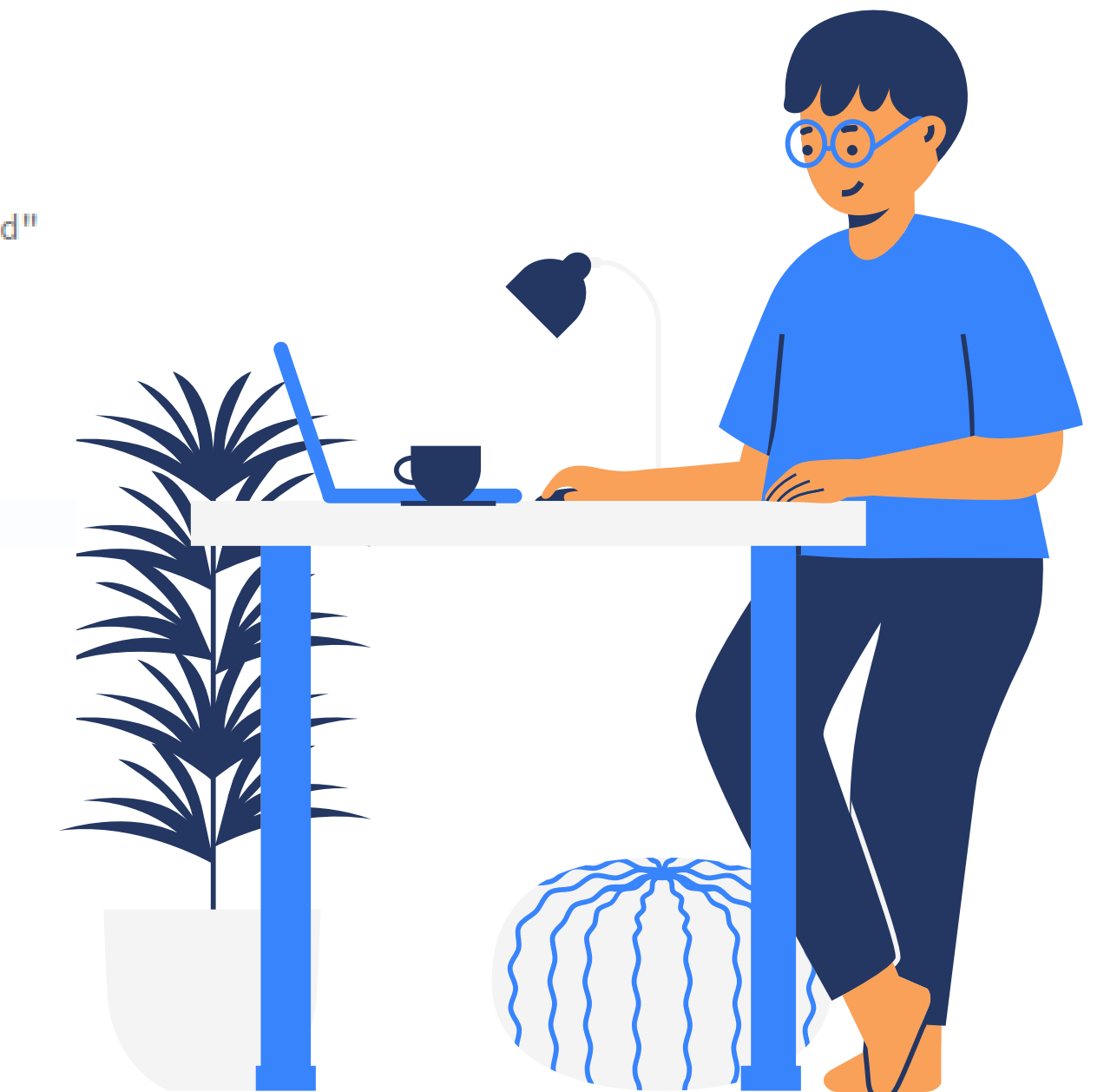
### Création de testCopyFromToOrderedset

#### testCopyFromToOfOrderedSet

"Allows one to create a copy of the receiver that contains elements from position start to end"

```
| myAccount1 myAccount2 myAccount3 |  
myAccount1 := #(100 200 300 400) asOrderedSet .  
myAccount2 := myAccount1 copyFrom: 1 to: 2.  
self assert: myAccount2 equals: #(100 200) asOrderedSet. "-> is Good ! "  
  
myAccount3 := myAccount1 copyFrom: 4 to: 2.  
self assertEmpty: myAccount3. "Index 4 >2 -> return Empty "  
  
self should: [ myAccount1 copyFrom: 5 to: 7 ] raise: Error. "-> Index out Limit"
```

 testCopyFromToOfOrderedSet





Merci pour  
votre attention !