

Développement Orienté Objets

Kotlin : collections

Arnaud Lanoix Brauer

Arnaud.Lanoix@univ-nantes.fr



Nantes Université

Département informatique

Les collections en Kotlin

Les tableaux `Array<E>` en Kotlin ont des **limites**

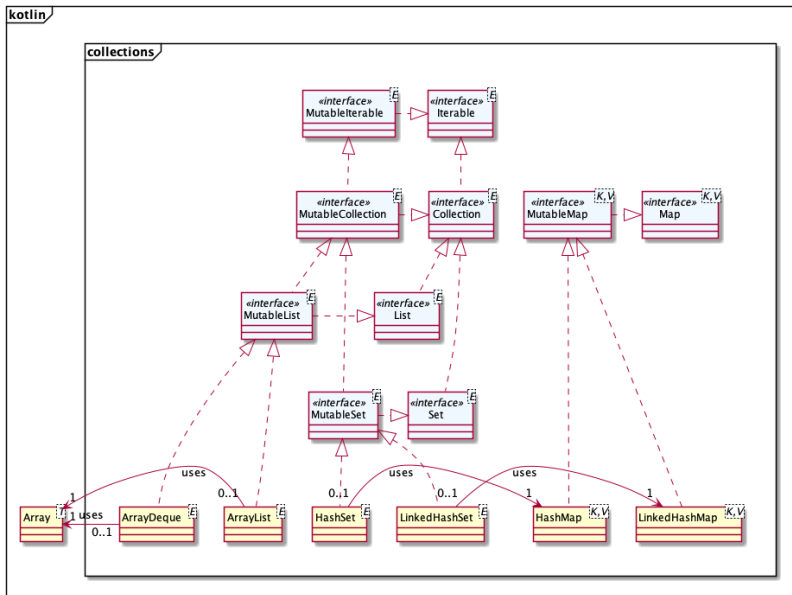
- taille fixée à l'initialisation et difficile à modifier ensuite
- initialisés à `null`, obligeant à des vérifications

Le package `kotlin.collections` (importé par défaut) propose plusieurs **collections génériques**

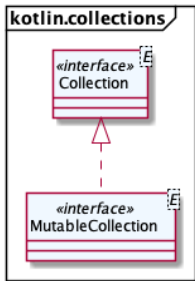
- `List<E>` et `MutableList<E>` : collections **ordonnées**, **doublons possible**
- `Set<E>` et `MutableSet<E>` : collections **sans doublon**, ordre non garanti
- `Map<K,V>` et `MutableMap<K,V>` : **dictionnaires** = ensembles de (clef,valeur)

<https://kotlinlang.org/docs/collections-overview.html#collection-types>

Package `kotlin.collections`



Collection et MutableCollection



- `Collection<E>` = n'importe quelle **collection non modifiable**

- ▶ `size : Int`
- ▶ Opérateur Kotlin `in` pour un élément `E`
- ▶ `containsAll(elements:Collection<E>):Boolean`
- ▶ `isEmpty():Boolean`
- ▶ Opérateurs Kotlin `+` et `-`, pour un `E` ou une `Collection<E>`
- ▶ ...

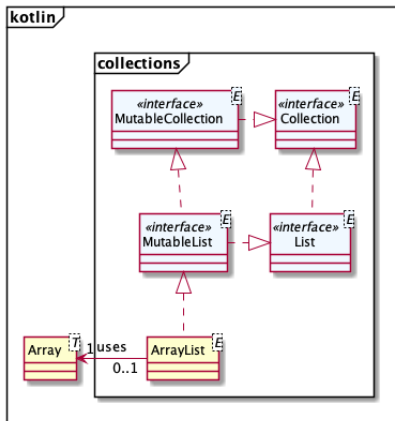
- `MutableCollection<E>` = n'importe quelle **collection modifiable**

- ▶ `add(element:E):Boolean` ou opérateur `+=`
- ▶ `addAll(elements:Collection<E>):Boolean`
- ▶ `remove(element:E):Boolean` ou opérateur `-=`
- ▶ `removeAll(elements:Collection<E>): Boolean`
- ▶ ...

List et MutableList

Liste

= structure de données **dynamique** permettant de stocker des éléments de manière ordonnée, à un **indice** donné



List et MutableList

- `List<E>` = une liste non modifiable

- ▶ Opérateur d'accès indicé `[i]` en lecture

- ▶ `indexOf(element : E) : Int`

- ▶ `lastIndexOf(element : E) : Int`

- ▶ `subList(fromIndex: Int, toIndex: Int): List<E>`

- ▶ `binarySearch(element : E) : Int`

- ▶ ...

- `MutableList<E>` = une liste modifiable

- ▶ Opérateur d'accès indicé `[i]` en écriture

- ▶ `add(index : Int, element : E)`

- ▶ `sort()` et `sortDescending()`

- ▶ `shuffle()`

- ▶ ...

ArrayList

- Implémentation **par défaut** de l'interface `MutableList<E>`
- utilise un `Array<T>`
 - ▶ procède à des **redimensionnements** automatiques
- Les fonctions
 - ▶ `listOf(..) : List<E>` et
 - ▶ `mutableListOf(..) : MutableList<E>`

instancient de manière **sous-jacente** un objet de type `ArrayList<E>`

- Constructeurs possible :
 - ▶ `ArrayList<E>()`,
 - ▶ `ArrayList<E>(initialCapacity : Int)` ou
 - ▶ `ArrayList<E>(elements : Collection<E>)`

Déclarer des listes

- Pour construire une liste **non modifiable** (en lecture seule), utilisez

`listOf(..) : List<E>` :

```
val entiers0 = listOf<Int>(42,4,-2,-42,42,0)
val prenom0 = listOf<String>("Jean-Francois", "Christine",
                             "Ali","Jean-Francois", "Arnaud")
val animaux = listOf<Animal>(Chien("Rogue"), Chien("Potter"),
                             Chat("Gaga"))
```

- Pour construire une liste **modifiable**, instanciez un `ArrayList<E>`

```
val noms = ArrayList<String>()
```

- ou utilisez `mutableListOf(..) : MutableList<E>` :

```
val noms = mutableListOf<String>()
val prenom = mutableListOf<String>("Arnaud", "Jean-Francois",
                                   "Christine", "Ali", "Jean-Francois", "Arnaud")
```


Utiliser des listes

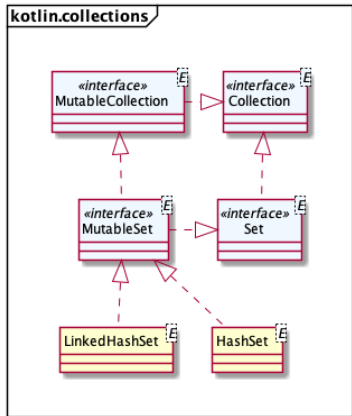
```
var prenoms = mutableListOf<String>(  
    "Jean-Francois",  
    "Christine")  
println(prenoms)  
prenoms += "Arnaud"  
println(prenoms)  
prenoms.add(1, "Ali")  
println(prenoms)  
var prenomsX = mutableListOf<String>(  
    "Jean-Marie",  
    "Jean-Francois")  
prenoms.addAll(prenomsX)  
println(prenoms)  
prenoms -= "Jean-Marie"  
println(prenoms)  
var result : Int  
result = prenoms.size  
println(result)  
println(prenoms[3])  
prenoms[3] = "Arnaud"  
for (i in 0 until prenoms.size) {  
    println(prenoms[i])  
}  
for (nom in prenoms) {  
    println(nom)  
}
```

```
var condition : Boolean  
condition = "Ali" in prenoms  
println(condition)  
condition = "Jean-Marie" in prenoms  
println(condition)  
condition = "Jean-Marie" !in prenoms  
println(condition)  
result =  
    prenoms.indexOf("Jean-Francois")  
println(result)  
result =  
    prenoms.lastIndexOf("Jean-Francois")  
println(result)  
result =  
    prenoms.indexOf("Jean-Marie")  
println(result)  
prenoms.shuffle()  
println(prenoms)  
prenoms.sort()  
println(prenoms)  
prenoms.sortDescending()  
println(prenoms)
```

Set et MutableSet

Ensemble

= structure de données **dynamique** permettant de stocker des éléments de manière **unique**, c-à-d qu'elle ne contient **pas de doublons**.



HashSet et LinkedHashMap

= Deux implémentations possible de l'interface `MutableSet<E>`

- `HashSet<E>` utilise `HashMap<E,*>` comme structure sous-jacente
 - ▶ ne préserve pas l'ordre d'insertion
- `LinkedHashSet<E>` utilise `LinkedHashMap<E,*>` comme structure sous-jacente
 - ▶ préserve l'ordre d'insertion
 - ▶ plus coûteux
- `HashMap<K,V>` et `LinkedHashMap<K,V>` sont basées sur des tables de hachage pour détecter efficacement les doublons
 - ▶ utilisent la fonction `hashCode()` de `K`
- Les fonctions
 - ▶ `setOf(..) : Set<E>` et
 - ▶ `mutableSetOf(..) : MutableSet<E>`

instancient de manière sous-jacente un objet `LinkedHashSet<E>`

Déclarer des ensembles

- Pour construire un ensemble **non modifiable (en lecture seule)**, utilisez

`setOf(..) : Set<E>` :

```
val entiers = setOf<Int>(42,4,-2,-42,0)
val animaux = setOf<Animal>(Chien("Rogue"), Chien("Potter"),
                             Chat("Gaga"))
```

- Pour construire un ensemble **modifiable**, instanciez `HashSet<E>`

```
val noms = HashSet<String>()
```

- ou instanciez `LinkedHashSet<E>`

```
val noms = LinkedHashSet<String>()
```

- ou utilisez `mutableSetOf(..) : MutableSet<E>` :

```
val noms = mutableSetOf<String>()
val prenom = mutableSetOf<String>("Arnaud", "Christine",
                                   "Ali", "Jean-Francois")
```

Utiliser des ensembles

```
var prenom = mutableSetOf<String>(  
    "Jean-Francois",  
    "Christine")  
println(prenom)  
prenom += "arnaud"  
println(prenom)  
var condition : Boolean  
condition = prenom.add("Ali")  
println(condition)  
println(prenom)  
condition = prenom.add("Christine")  
println(condition)  
println(prenom)  
var prenomX = mutableSetOf<String>(  
    "Jean-Marie",  
    "Jean-Francois")  
prenom.addAll(prenomX)  
println(prenom)
```

```
prenom -= "Jean-Marie"  
println(prenom)  
var result = prenom.size  
println(result)  
for (nom in prenom) {  
    println(nom)  
}  
condition = "Ali" in prenom  
println(condition)  
condition = "Jean-Marie" in prenom  
println(condition)  
condition = "Jean-Marie" !in prenom  
println(condition)
```