R3.04 : Qualité de développement

Rappels Kotlin (2)

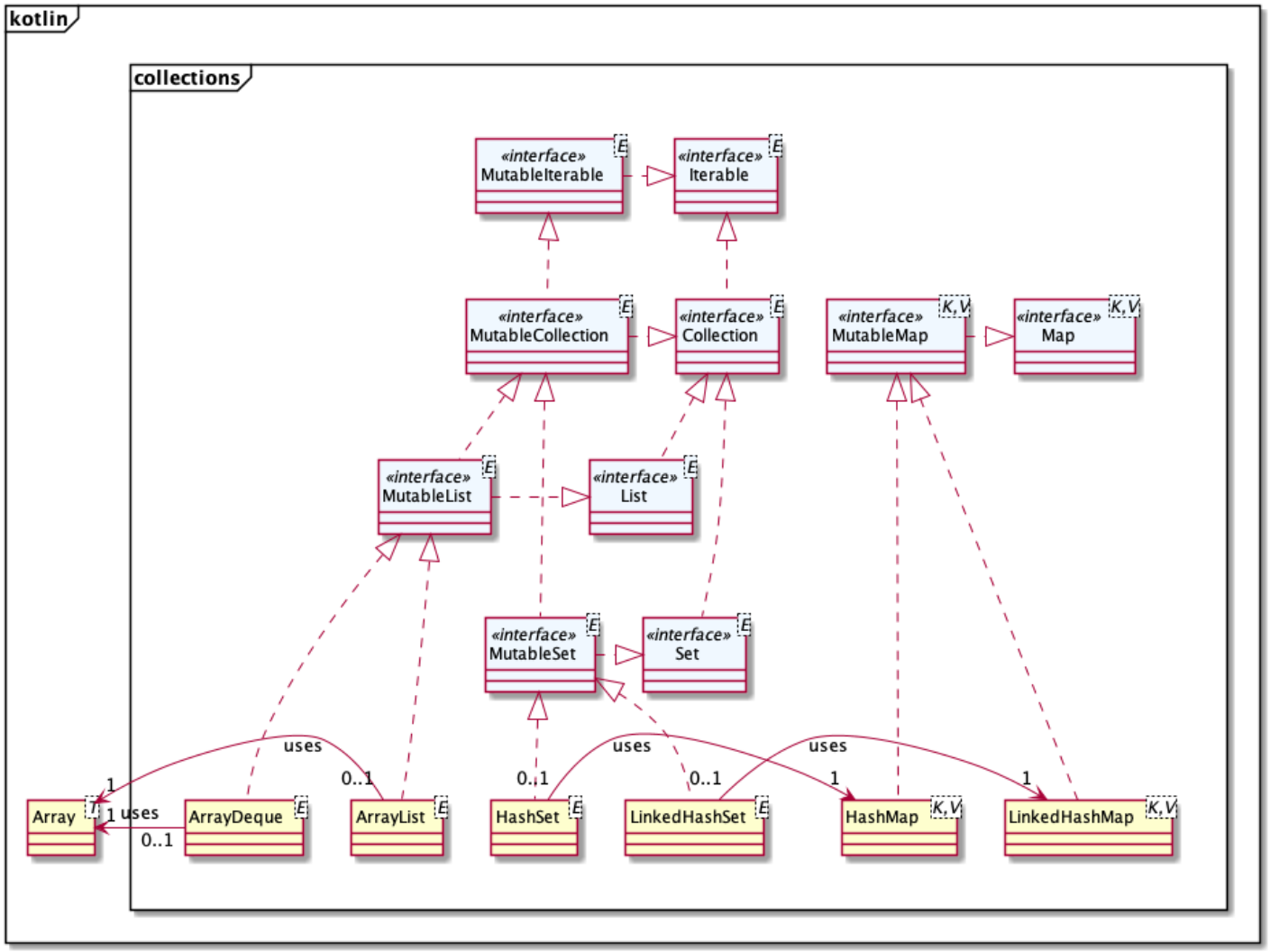
Arnaud Lanoix Brauer
Arnaud.Lanoix@univ-nantes.fr

Département informatique

Sommaire

IUT Nantes

Package `kotlin.collections`



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

IUT Nantes

`List<E>` et `MutableList<E>`

collections ordonnées, doublons possible

- `ArrayList` = implémentation de l'interface `MutableList<E>`
- utilise un `Array<E?>` sous-jacent
  - 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>)`

IUT Nantes

`Set<E>` et `MutableSet<E>`

collections sans doublon, ordre non garanti

`HashSet` et `LinkedHashSet` = implémentations possible de l'interface `MutableSet<E>`

- `HashSet<E>` ne préserve pas l'ordre d'insertion
- `LinkedHashSet<E>` préserve l'ordre d'insertion, mais c'est plus coûteux

Les deux implémentations 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>`

IUT Nantes

Sommaire

IUT Nantes

Capturer des exceptions

```
val prenoms = arrayOf<String>(
"Jean-Francois", "Ali",
"Christine", "Jean-Francois", "Arnaud")

fun acces(pos : Int, div : Int)
= prenoms[pos/div]
}

fun main() {
val indice = 8
val facteur = 0
var prenom = ""
try {
  prenom += acces(indice, facteur)
  println("### ok")
}
catch(e:ArithmeticException) {
  println("*** Erreur : $e ***")
  prenom += acces(indice, 2)
}
catch(e:ArrayIndexOutOfBoundsException) {
  println("*** Erreur : $e ***")
  prenom += acces(0, 1)
}
finally {
  println("Prenom : $prenom")
}
```

- le bloc `try` englobe le code à risque ; son exécution est interrompue dès la survenue d'une exeption
- le bloc `catch` est exécuté s'il correspond à l'exception levée
- si aucun bloc `catch` ne correspond à l'exception alors elle est remontée
- le bloc optionnel `finally` est exécuté à la suite dans tous les cas

IUT Nantes

Lever une exception

Pour lever une exception il suffit d'utiliser l'instruction `throw` suivie d'une exception.
Exécuter une instruction `throw` provoque l'interruption instantanée du code, et

- remonte jusqu'à un bloc `try... catch` correspondant à l'exception
- ou provoque la terminaison du programme

Exemple

```
fun acces(indice : Int) : String {
if (indice < 0 || indice >= prenoms.size)
    throw IllegalArgumentException("$indice")
return prenoms[indice]
}
```

IUT Nantes

Lever une exception

Pour lever une exception il suffit d'utiliser l'instruction `throw` suivie d'une exception.
Exécuter une instruction `throw` provoque l'interruption instantanée du code, et

- remonte jusqu'à un bloc `try... catch` correspondant à l'exception
- ou provoque la terminaison du programme

Exemple

```
fun acces(indice : Int) : String {
if (indice < 0 || indice >= prenoms.size)
    throw IllegalArgumentException("$indice")
return prenoms[indice]
}
```

IUT Nantes

`require` and `check`

- La fonction `require` permet de vérifier la valeur d'un paramètre (de classe, de méthode) et éventuellement de lever une exception `IllegalArgumentException`

```
fun acces(indice : Int) : String {
  require(indice >= 0){"indice negatif"}
  require(indice < prenoms.size){"indice trop grand"}
  return prenoms[indice]
}
```

  - Il y a aussi une fonction `requireNotNull`

- La fonction `check` permet de vérifier la valeur des variables/objets à un point quelconque du programme et éventuellement de lever une exception `IllegalStateException`

  - Il y a aussi une fonction `checkNotNull`

IUT Nantes