R2.01 - Développement Orienté Objets

Les exceptions en Kotlin

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

IUT Nantes
Pôle Sciences et technologie

Nantes Université

Département informatique

Exemple de trace de pile d'exécution

L'exécution du programme suivant :

```
fun main() {
  val prenoms = arrayOf<String>("Jean-Francois", "Ali",
                  "Christine", "Jean-Francois", "Arnaud")
  prenoms[10]
}
```

provoque le message d'erreur suivant :

```
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException:
      Index 10 out of bounds for length 5
  at MainExceptionKt.main(MainException.kt:4)
  at MainExceptionKt.main(MainException.kt)
  at java.base/jdk.internal.reflect.NativeMethodAccessorImpl.invoke0(Native Meth
  at java.base/jdk.internal.reflect.NativeMethodAccessorImpl.invoke(NativeMetho
  at java.base/jdk.internal.reflect.DelegatingMethodAccessorImpl.invoke(Delegat
  at java.base/java.lang.reflect.Method.invoke(Method.java:568)
  at org.jetbrains.kotlin.runner.AbstractRunner.run(runners.kt:64)
  at org.jetbrains.kotlin.runner.Main.run(Main.kt:176)
  at org.jetbrains.kotlin.runner.Main.main(Main.kt:186)
```

IUT Nantes

Exemple de trace de pile d'exécution

L'exécution du programme suivant :

```
fun main() {
  val prenoms = arrayOf<String>("Jean-Francois", "Ali",
                  "Christine", "Jean-Francois", "Arnaud")
  prenoms[10]
}
```

provoque le message d'erreur suivant :

```
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException:
      Index 10 out of bounds for length 5
  at MainExceptionKt.main(MainException.kt:4)
  at MainExceptionKt.main(MainException.kt)
  at java.base/jdk.internal.reflect.NativeMethodAccessorImpl.invoke0(Native Meth
  at java.base/jdk.internal.reflect.NativeMethodAccessorImpl.invoke(NativeMetho
  at java.base/jdk.internal.reflect.DelegatingMethodAccessorImpl.invoke(Delegat
  at java.base/java.lang.reflect.Method.invoke(Method.java:568)
  at org.jetbrains.kotlin.runner.AbstractRunner.run(runners.kt:64)
  at org.jetbrains.kotlin.runner.Main.run(Main.kt:176)
  at org.jetbrains.kotlin.runner.Main.main(Main.kt:186)
```

IUT Nantes

Exceptions

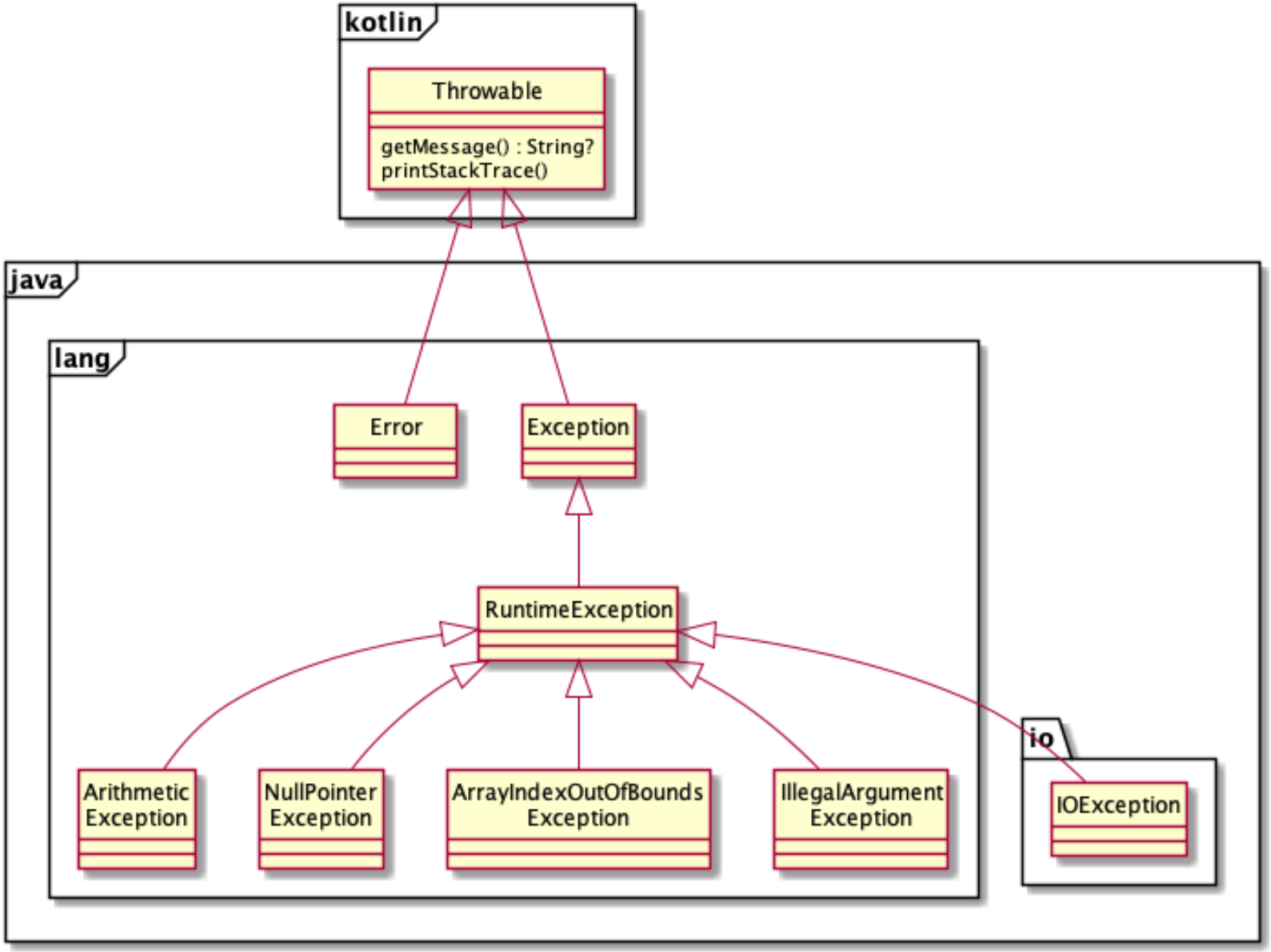
= mécanisme de signalement des erreurs consistant à sortir du flot d'exécution standard pour exécuter un traitement spécifique ou terminer le programme de manière précoce

Quelques erreurs courantes provoquant des exceptions :

- réaliser une division par 0
  $\Rightarrow$ `ArithmeticException`
- manipuler un objet non-initialisé
  $\Rightarrow$ `NullPointerException`
- accéder à la case 42 d'un tableau de 10 éléments seulement
  $\Rightarrow$ `ArrayIndexOutOfBoundsException`
- utiliser un argument inattendu pour une fonction
  $\Rightarrow$ `IllegalArgumentException`
- écrire ou lire un fichier inexistant
  $\Rightarrow$ `IOException`
- ...

IUT Nantes

Exceptions



IUT Nantes

Capturer des exceptions

= réaliser un traitement spécifique pour prendre en compte l'erreur et éviter que l'exception ne provoque la terminaison précoce du programme

```
try {
  // Portion de code
  // suceptible de lever une
  // exception
}
catch (e : SomeException) {
  // Portion de code a executer
  // si l'exception survient
}
catch (e : OtherException) {
  // Portion de code a executer
  // si l'exception survient
}
...
finally {
  /* optionnel */
  // Portion de code execute
  // dans tous les cas
}
```

- 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

Capturer des exceptions : exemple

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

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

Deux exceptions peuvent survenir :

- `post/div` peut provoquer `ArithmeticException`
- `prenoms[...]` peut provoquer `ArrayIndexOutOfBoundsException`

```
fun main() {
  essai(3, 1)
  essai(8, 0)
  essai(10, 2)
}
```

```
fun usage(indice : Int, facteur : Int) {
  var prenom = "Inconnu"

  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")
  }
}
```

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

Mais pourquoi lever des exceptions ?

On peut signaler des erreurs d'autres manières :

- valeur de retour spécifique

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

- `-1`, `"Inconnu"` ou `false` par exemple
- il faut pouvoir "réserver" des valeurs

- valeur de retour `null`

```
fun acces(indice : Int) : String? {
  if (indice < 0 || indice >= prenoms.size)
    return null
  return prenoms[indice]
}
```

- oblige à retourner un résultat nullable

- Et si pas de valeurs de retour ?
- **Il n'y a pas de règles, mais il faut savoir faire avec les exceptions**

IUT Nantes

Mais pourquoi lever des exceptions ?

On peut signaler des erreurs d'autres manières :

- valeur de retour spécifique

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

- `-1`, `"Inconnu"` ou `false` par exemple
- il faut pouvoir "réserver" des valeurs

- valeur de retour `null`

```
fun acces(indice : Int) : String? {
  if (indice < 0 || indice >= prenoms.size)
    return null
  return prenoms[indice]
}
```

- oblige à retourner un résultat nullable

- Et si pas de valeurs de retour ?
- Il n'y a pas de règles, mais il faut savoir faire avec les exceptions

IUT Nantes

Mais pourquoi lever des exceptions ?

On peut signaler des erreurs d'autres manières :

- valeur de retour spécifique

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

- `-1`, `"Inconnu"` ou `false` par exemple
- il faut pouvoir "réserver" des valeurs

- valeur de retour `null`

```
fun acces(indice : Int) : String? {
  if (indice < 0 || indice >= prenoms.size)
    return null
  return prenoms[indice]
}
```

- oblige à retourner un résultat nullable

- Et si pas de valeurs de retour ?
- Il n'y a pas de règles, mais il faut savoir faire avec les exceptions

IUT Nantes

Mais pourquoi lever des exceptions ?

On peut signaler des erreurs d'autres manières :

- valeur de retour spécifique

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

- `-1`, `"Inconnu"` ou `false` par exemple
- il faut pouvoir "réserver" des valeurs

- valeur de retour `null`

```
fun acces(indice : Int) : String? {
  if (indice < 0 || indice >= prenoms.size)
    return null
  return prenoms[indice]
}
```

- oblige à retourner un résultat nullable

- Et si pas de valeurs de retour ?
- **Il n'y a pas de règles, mais il faut savoir faire avec les exceptions**

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`

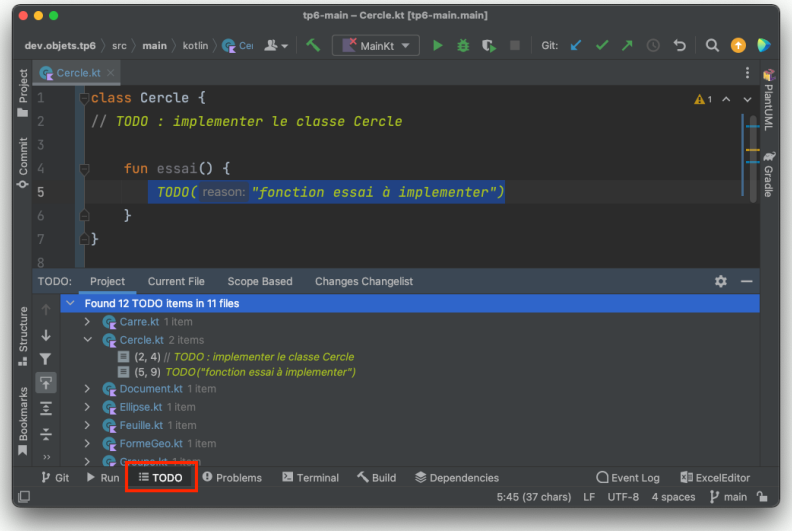
IUT Nantes

`TODO("...")`

La fonction Kotlin `TODO("...")` permet d'indiquer que l'on n'a pas encore implémenter le code nécessaire

- lève `kotlin.NotImplementedError` si le code est appelé

```
fun acces(indice : Int) : String {
        if (indice < 0 || indice >= prenoms.size)
          TODO("reste a implementerce cas precis")
        else
          return prenoms[indice]
}
```



IntelliJ IDEA liste tous les TODO présents dans le code

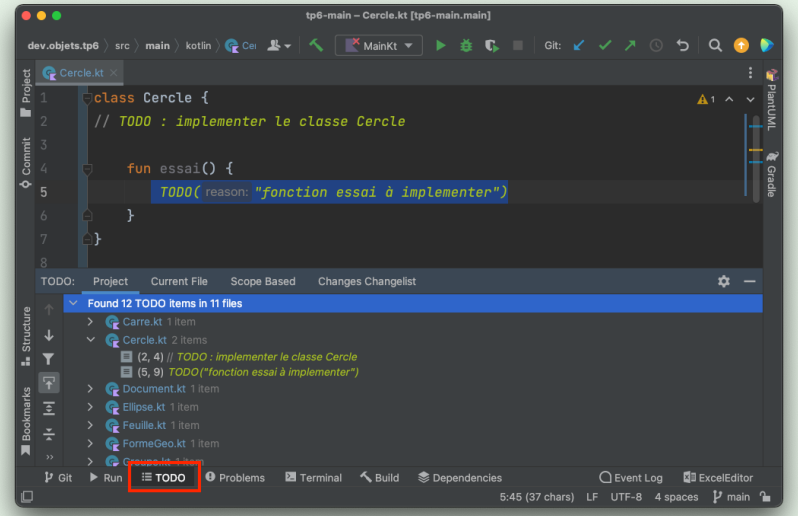
- `TODO("...")`
- `// TODO : ...` ne lève pas d'exeption

`TODO("...")`

La fonction Kotlin `TODO("...")` permet d'indiquer que l'on n'a pas encore implémenter le code nécessaire

- lève `kotlin.NotImplementedError` si le code est appelé

```
fun acces(indice : Int) : String {
        if (indice < 0 || indice >= prenoms.size)
          TODO("reste a implementerce cas precis")
        else
          return prenoms[indice]
}
```



IntelliJ IDEA liste tous les TODO présents dans le code

- `TODO("...")`
- `// TODO : ...` ne lève pas d'exeption

Manipuler des exceptions personnalisées

Il est bien entendu possible de créer, de lever et ensuite de capturer des exceptions personnalisées propres à votre contexte métier.
Il suffit de créer une nouvelle classe qui hérite de `Exception`

```
class IndiceException(message : String)
: Exception(message)
```

On l'utilise ensuite commme une autre exception

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

IUT Nantes