Kotlin

Mme Mounira Zouaghi

Cours V3.0

1. Les fondamentales de Kotlin

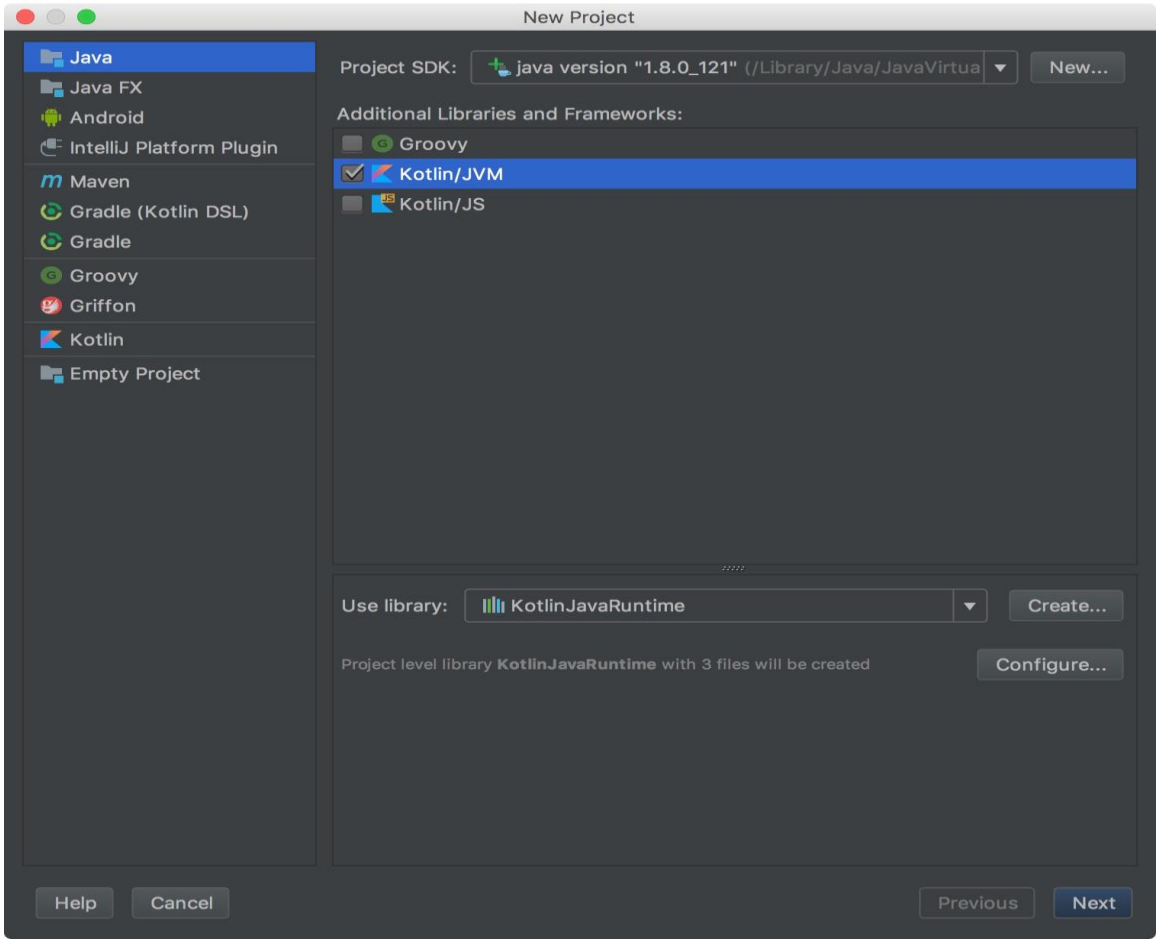
Plan de cours

1. **Une Première application kotlin**
2. **Le syntaxe de base**
3. **Les instructions de contrôle Kotlin**
4. **Les fonctions**
5. **Les boucles**
6. **Les tableaux**

Une première application kotlin

Step 1: Select “Create New Project”.

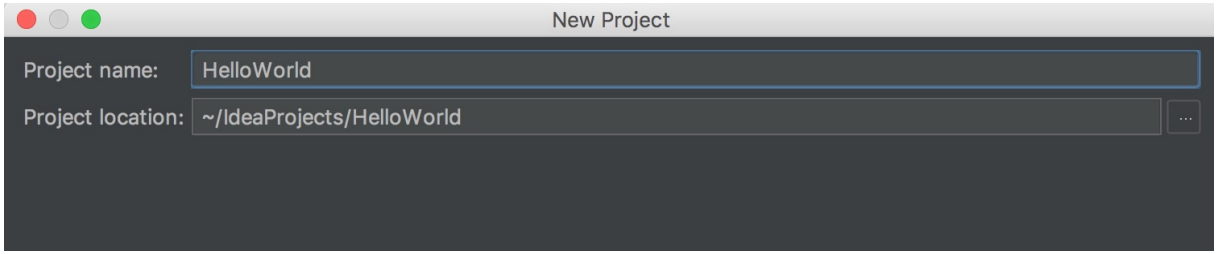
Une première application kotlin

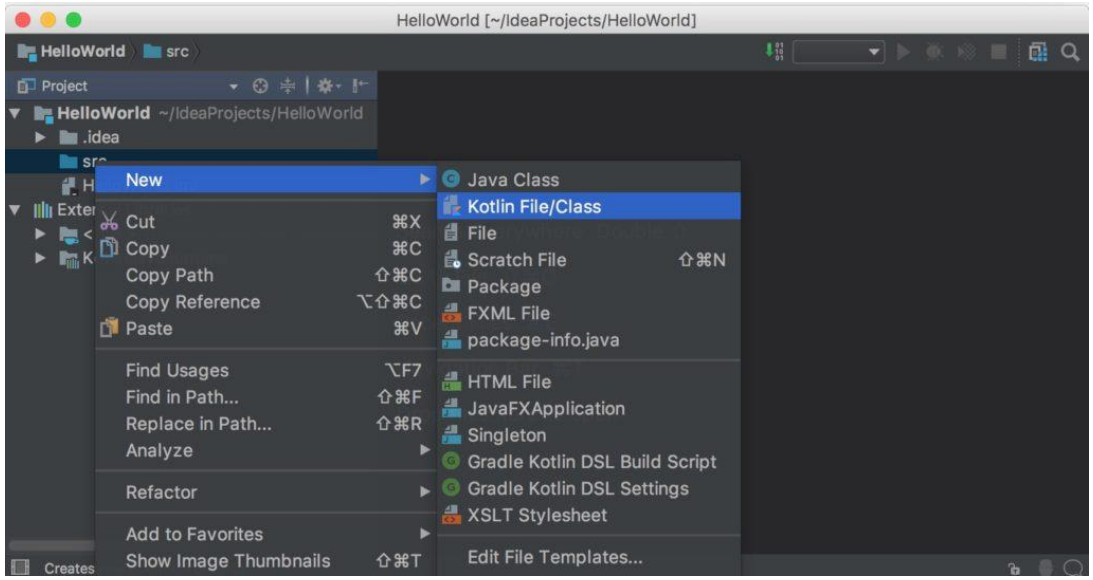


Step 2: In “Additional Libraries and Frameworks” section, select **Kotlin/JVM** and click “Next”.

Une première application kotlin

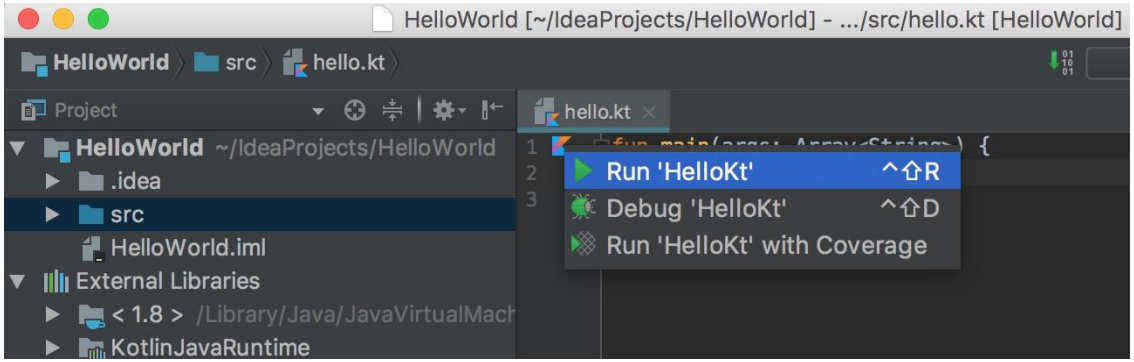
Step 3: Give the Project Name and click “Finish”.

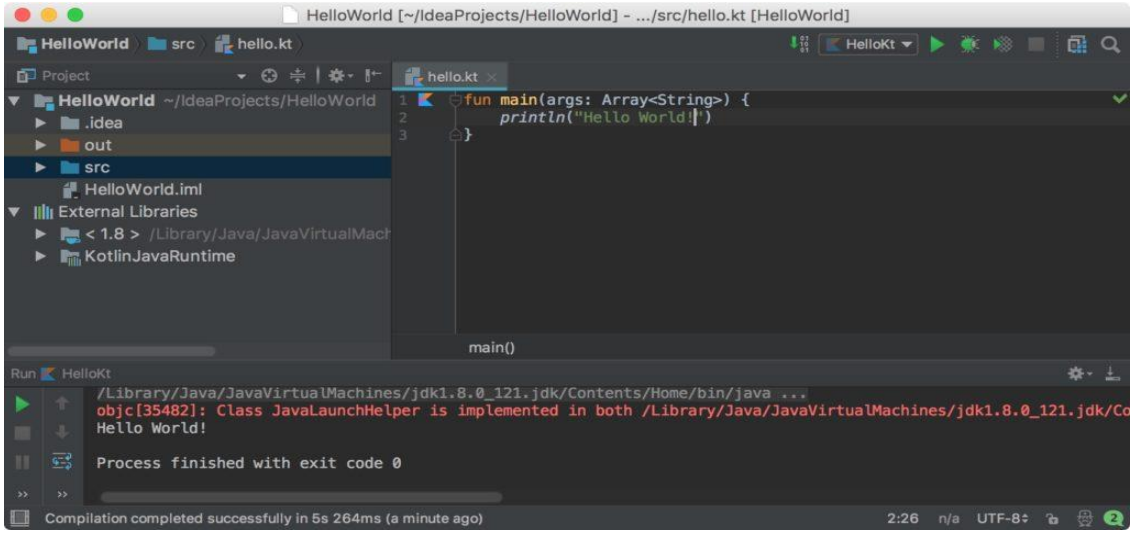




Step 4: Create a new Kotlin File in “src” folder of Kotlin Project.

Une première application kotlin





Step 5: Type the code and Run application

Une fonction main kotlin vs Class Main

- En Java, il faut toujours définir une classe
- Kotlin a besoin de jvm comme java
- Est ce que kotlin peut s'en passer des classes ?

->non

->kotlin compiler crée une classe interne qui sera utilisée par le Java Runtime

Déclaration des variables

- On distingue les variables dont on peut changer la valeur, et les variables immuables qui ne sont pas tout à fait des constantes
  - le mot-clé **val** permet de déclarer une variable immuable,
  - le mots-clé **var** permet de déclarer une variable altérable.

type de variable après le nom, et pas avant comm

```
val nom:String = " XXX"
// nom = « YYY" // Interdit !!! Car nom a été déclaré avec val
var age? = 10
age += 12
// aucun problème car age est altérable.
```

Les constantes

- En java on peut déclarer des constantes

- On peut déclarer des constantes kotlin comme suit:

Le mot clé **const** suivi de **val**

Les types de base

- Les types numérique de base sont le même qu'en java: **long, short, double, int , float, byte**
- En kotlin tout est **objet**

```
2.toString()
10.downTo(0) // génère le range (c'est à dire l'intervalle) décroissant de 10 à 0.
```

- Le type String dispose des mêmes méthodes qu'en Java, mais aussi des méthodes supplémentaires (all, any….)

```
println("martin".all{it.isLowerCase()})  // Teste si le mot entier est en basse casse.
```

Représentation

```
//underscore
val oneMillion = 1_000_000
val creditCardNumber = 1234_5678_9012_3456L
val socialSecurityNumber = 999_99_9999L
val hexBytes = 0xFF_EC_DE_5E
val bytes = 0b11010010_01101001_10010100_10010010
```

```
//comparaison des références
val a: Int = 10000
println(a === a) // Prints 'true'
val boxedA: Int? = a
val anotherBoxedA: Int? = a
println(boxedA === anotherBoxedA) // !!!Prints 'false'!!!
```

Null Safety

- **Java.lang.NullPointerException** se déclenche lorsqu'on veut accéder à une variable null.
- Pour résoudre ce problème:
  - Par défaut les variables ne peuvent pas avoir des valeurs nulles
  - Si on veut déclarer une variable qui peut avoir la valeur null il faut explicitement l'autoriser

Exemple

```
var canBeNull:String?="testNullSafety"
val l=canBeNull.length
```

Une variable qui peut être null, on ne peut pas l´utiliser sana vérifier qu´il n´est pas null

```
var canBeNull:String?="testNullSafety"
val l=canBeNull?.length
```

```
var canBeNull:String?="testNullSafety"
val l=canBeNull!!.length
```

```
var canBeNull:String?="testNullSafety"

val l=if(canBeNull!=null) canBeNull else 0
```

Cas Pratique1

1. Les fonctions

Les fonctions(1)

```
//declaration
fun double(x: Int): Int {
    return 2 * x
}
```

```
//appel
val result = double(2)
```

Les fonctions(2):autre écriture

```
fun sayHello() = println("Hello !")
```

En Kotlin, **toutes** les fonctions **retournent une valeur**, même si aucun type de retour n'est mentionné (le type retourné par défaut sera alors **Unit**).

```
fun sayHello(): Unit = println("Hello !")
```

```
private fun minOf(a: Int, b: Int) = if (a < b) a else b
```

Une fonction n'est **pas obligée** de posséder un **corps** et peut contenir uniquement une **expression**.

Cas Pratique2

Type intervalle: Ranges

```
val i:Int =2
for (j in 1 ≤ .. ≤ 4)
print(j) // prints "1234"

if (i in 1 ≤ .. ≤ 10)
{ // equivalent of 1 <= i && i <= 10 ZWSP
    println("we found your number --"+i)
}
```

2. Les Instructions de contrôle

L'instruction if…else

```
if (testExpression)
{ // codes to run if testExpression is true }
else
{
// codes to run if testExpression is false
}
```

Exemple:

```
val number = -10
if (number > 0)
 { print("Positive number") }
else { print("Negative number") }
```

Exemple If expression

```
val result = if (number > 0)
{ "Positive number" }
Else
 { "Negative number" }
println(result)
```

If est une expression

Cas Pratique

When…

- When remplace switch en java, traite plusieurs alternatives. When est une expression

```
fun main(args: Array<String>)
 {
 val a = 12 val b = 5 println("Enter operator either +, -, * or /")
val operator = readLine()
val result = when (operator)
{ "+" -> a + b
"-" -> a - b
"*" -> a * b
"/" -> a / b
else -> "$operator operator is invalid operator." }
println("result = $result")
}
```

Boucle for

```
for (item in collection)
{ // body of loop }
```

En Kotlin, un intervalle est toujours **fermé** et **inclusif**, c'est-à-dire que sa seconde valeur sera incluse dans l'itération

```
fun main(args: Array<String>)
{ for (i in 1..5)
{ println(i) }
}
```

```
for (x in 1..10 step 2) {
    print(x)
}
println()
for (x in 9 downTo 0 step 3) {
    print(x)
}
```

Cas Pratique

While et do…while

Fonctionne comme en java

```
while (x > 0) {
    x--
}


do {
    val y = rechercheData()
} while (y != null) ///y est visible ici
```

Cas Pratique

Les tableaux (initialisation, declaration)

```
//Array of Integer
var arr0: Array<Int>
//Initialisation
arr0=arrayOf<Int>(1,10,4,6,15,1,10,4,6,15)
```

```
// Array of integers of a size of 10
var arr = IntArray( size: 10)
//Initialisation
arr=intArrayOf(1,10,4,6,15,1,10,4,6,15)
```

```
var arr4= arrayOf(1,10,4, "Ajay","Prakesh")
```

Les tableaux (initialisation, declaration)

```
// Array of integers of a size of N initialized with a default value of 2
    val arr1 = Array( size: 10) { i -> 2 *i}

// Array of integers of a size of N initialized with a default value of 2
    val arr2= Array( size: 10) { i -> 2 }
```

Les tableaux (parcours)

```
arr1.forEach {  it: Int
println("l´element $it")}

arr1.forEachIndexed{ index, item -> println("$item $index") }
```

Les tableaux

```
tab.sum()
tab.sort()
 tab.sorted() // idem que sort()
//mais se contente de retourner le résultat au lieu de modifier le tableau contenu dans tab
 tab.reverse()
tab.reversed()
tab.min()
 tab.max()
tab.first()
tab.last()
tab.take(3) // les 3 premiers éléments du tableau sans le modifier
 tab.drop(3) // le tableau sans ses 3 premiers éléments sans le modifier
```

Cas Pratique