# Formations
# Tristan SALAÜN

Pour Kotlin, Android (Java et Kotlin)
premier niveau et avancé

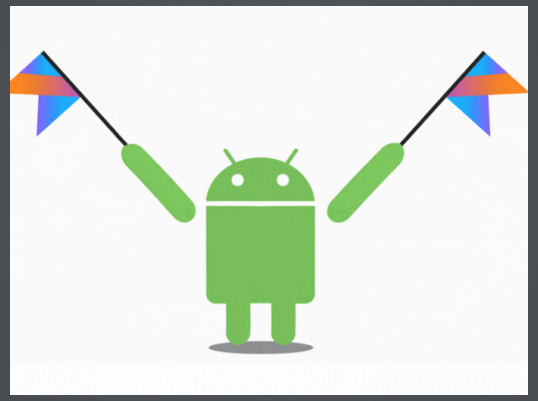
# Supports disponibles

- Android premier niveau.
- Android, perfectionnement (Réf. IOD).
- Kotlin.
- Kotlin, développer des applications pour Android (Réf. OTA).
- Kotlin mise en oeuvre (Réf. OTB).

# Utilisation de cette présentation

Appuyons sur la touche ?

# Kotlin, développer des applications pour Android



Le support en PDF.

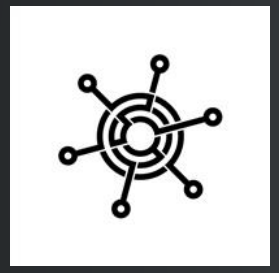
# Présentations

- Qui suis-je.
- Qui êtes-vous ? Quelles sont vos attentes.
- Présentation du plan.

# Présentations

## Qui suis-je

- Développeur d'applications mobiles Android (depuis 2009).
- Formateur Android/Java et Kotlin, et Javacard (scolaires et pros).
- Co-fondateur (1/7) de Startup Marseille.



- Fondateur de Light4Events.

# Présentations

## Qui êtes-vous ?

Tour de table :

- Prénom.
- Expérience (Java, Kotlin, Mobile Android/iOS).
- Attentes.

# Kotlin pour Android

# Introduction

- Pourquoi le Kotlin ?
- Introduction à la JVM (Java Virtual Machine).
- Interpréteur en ligne.
- La structure d'une application Kotlin.
- Kotlin et IntelliJ IDEA.
- Les conventions de nommage en Kotlin.

# Pourquoi le Kotlin ?

- Kotlin et Java sont 100 % interopérables.
- 2010 : idée (JetBrains).
- 2015 : naissance, disponible sur GitHub.
- Origine du nom : l'île de Kotlin.
- Concis, sûr et pragmatique.
- Statiquement typé, mais inférence de types.
- C'est aussi un langage fonctionnel.

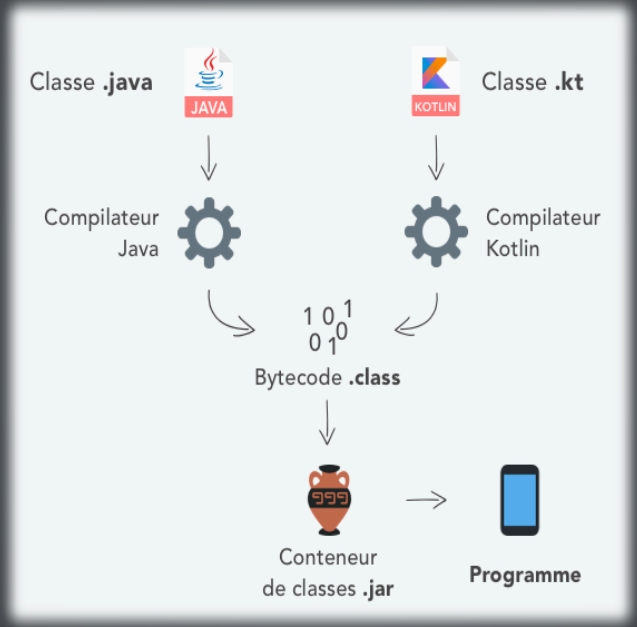
# Pourquoi le Kotlin ?

## Inférence de type

```kotlin
val number = 123
val message = "Hello world !"
fun sayHello() = "Hello world !"
```

Copier

# Introduction à la JVM (Java Virtual Machine)



# Lancer notre premier script

## En ligne

Nous pouvons utiliser le compilateur en ligne à l'adresse : https://play.kotlinlang.org.

```
1 fun main() {
2     println("Hello, World!")
3 }
```

Copier

# Lancer notre premier script

Nombreux exercices en ligne : https://play.kotlinlang.org/koans.

# La structure d'une application Kotlin

## Les répertoires

La structure des répertoires suit la structure des packages.
Le package racine sera ignoré, par exemple :
Si le projet est dans le package `org.example.kotlin`, alors les fichiers seront placés directement dans le répertoire racine contenant les sources.
Les fichiers dans le package `org.example.kotlin.network.socket` seront placés dans le sous répertoire : `network/socket`.

# La structure d'une application Kotlin

## Les fichiers

Un fichier ne contenant qu'une seule classe, sera nommé du nom de celle-ci (en utilisant le pascal case), suivit de l'extension `.kt`.
Si le fichier contient plusieurs classes, ou seulement des déclarations top niveau (top level declarations), alors, choisir un nom qui correspond le mieux au contenu du fichier.

# La structure d'une application Kotlin

## Dans le fichier source

Généralement le contenu d'une classe est organisé de la manière suivante :

- Déclaration des propriétés et des blocs d'initialiseurs.
- Les constructeurs secondaires.
- Les déclarations des méthodes.
- L'objet compagnon.

Regrouper les méthodes (classiques et d'extension) ensembles.
Gardez une organisation cohérente sur tout le projet.
L'implémentation d'une interface gardera l'ordre de déclaration dans celle-ci.

# Kotlin et IntelliJ IDEA

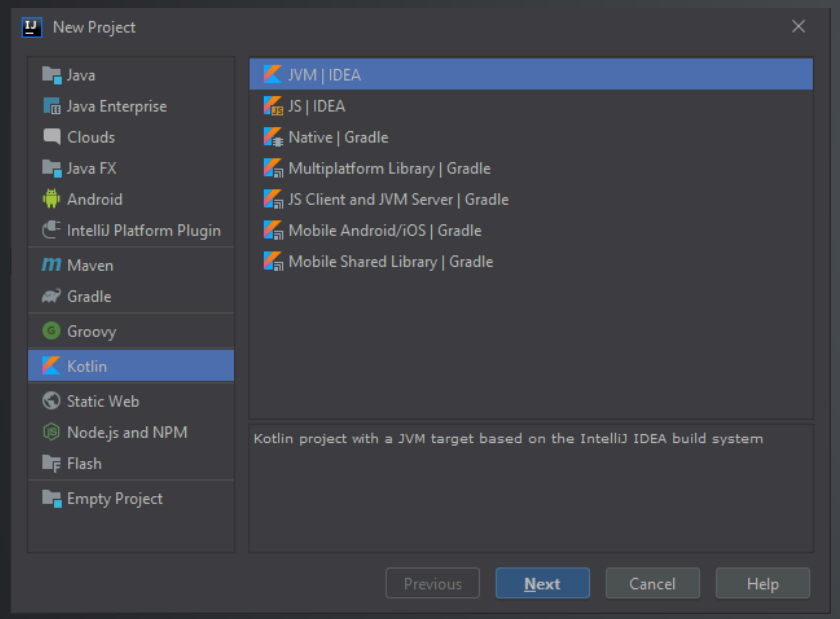
## Installation

Commençons par installer InteliJ : Télécharger.
Optons pour la version **Community**.

# Kotlin et IntelliJ IDEA

## Création du projet

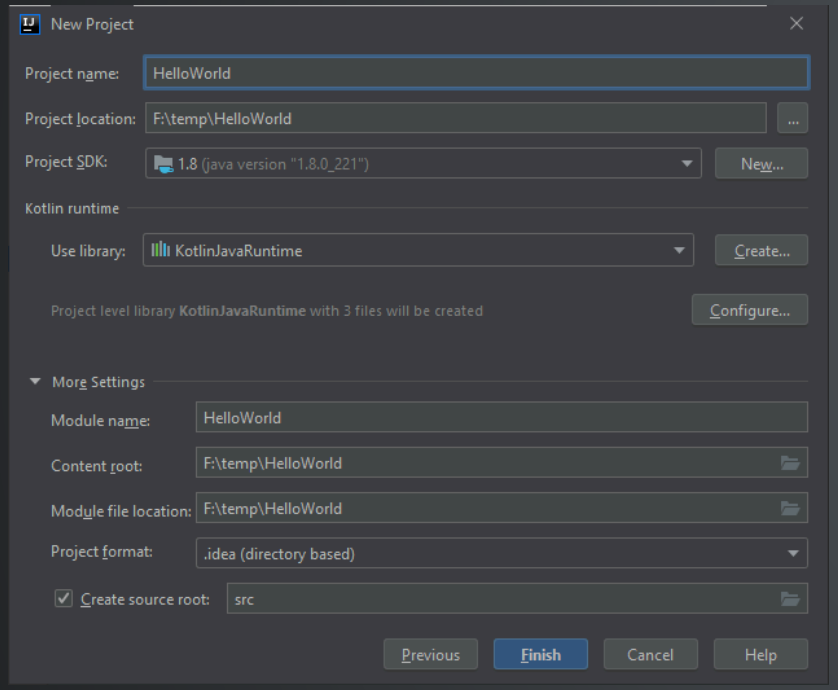
Il est temps de créer notre première application : **File / New / Project**. Sélectionner **Kotlin / JVM | IDEA**.
Nouvelle version ici.



# Kotlin et IntelliJ IDEA

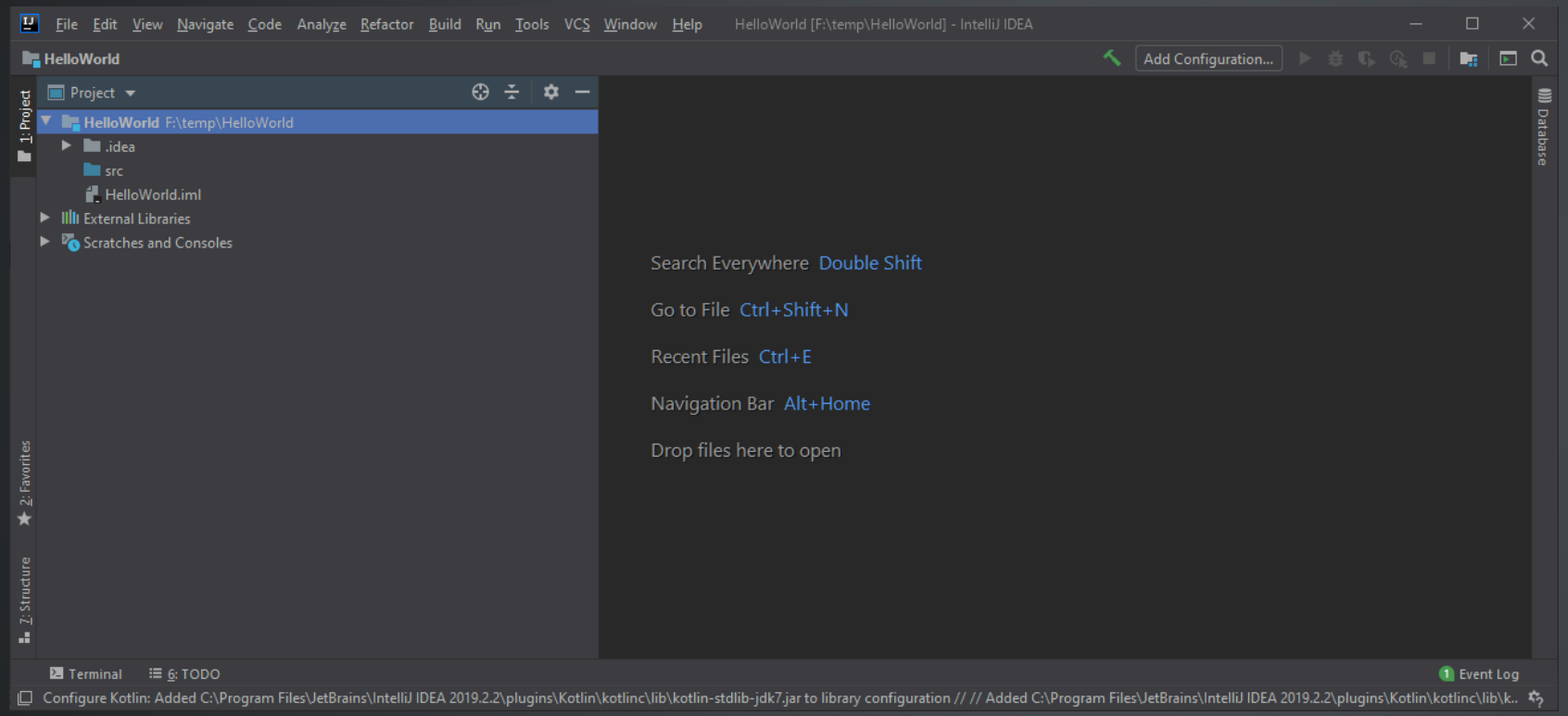
## Nommage

Nommons notre projet, par exemple **HelloWorld** :



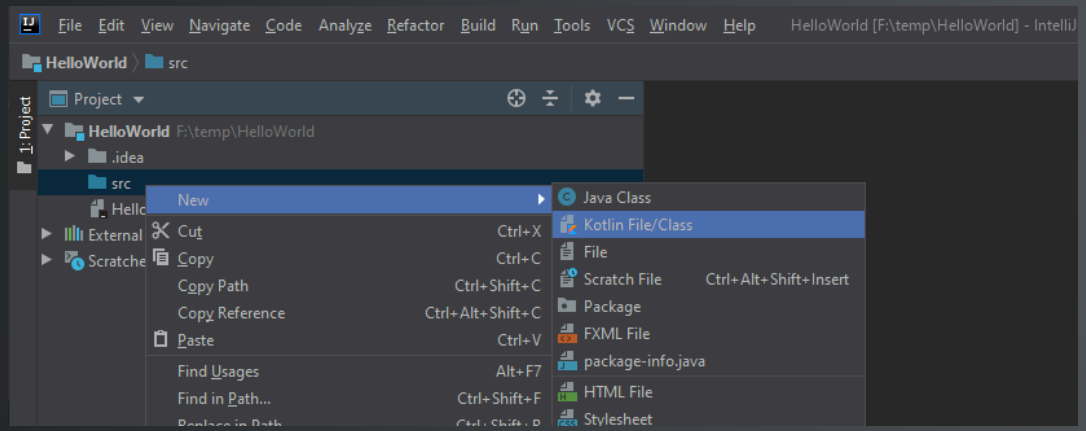
# Kotlin et IntelliJ IDEA

Nous devrions obtenir le résultat suivant :



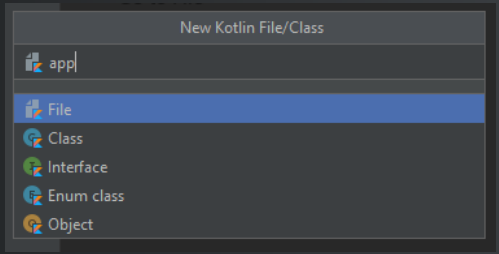
# Kotlin et IntelliJ IDEA

Créons un nouveau fichier dans le répertoire source. Click droit, **New / Kotlin File/Class** :



# Kotlin et IntelliJ IDEA

Nommons le **app** :



# Kotlin et IntelliJ IDEA

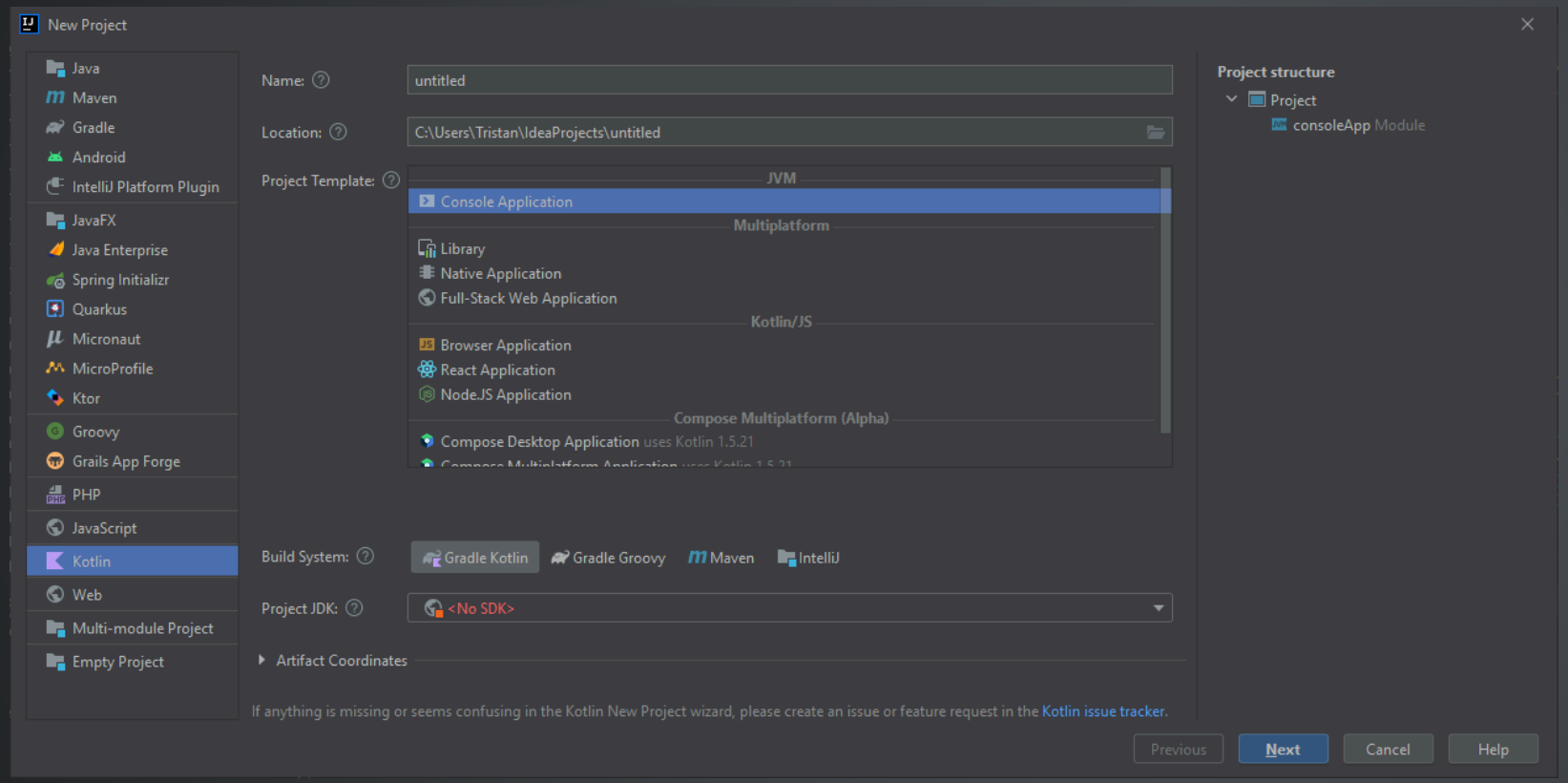
Ajoutons la fonction principale main :
Taper `main`, puis la touche **TAB**, pour lancer l'autocomplétion.
Il reste juste à compléter le corps de la méthode pour obtenir le résultat suivant :

```
1 fun main() {
2     println("Bonjour le monde")
3 }
```

Copier

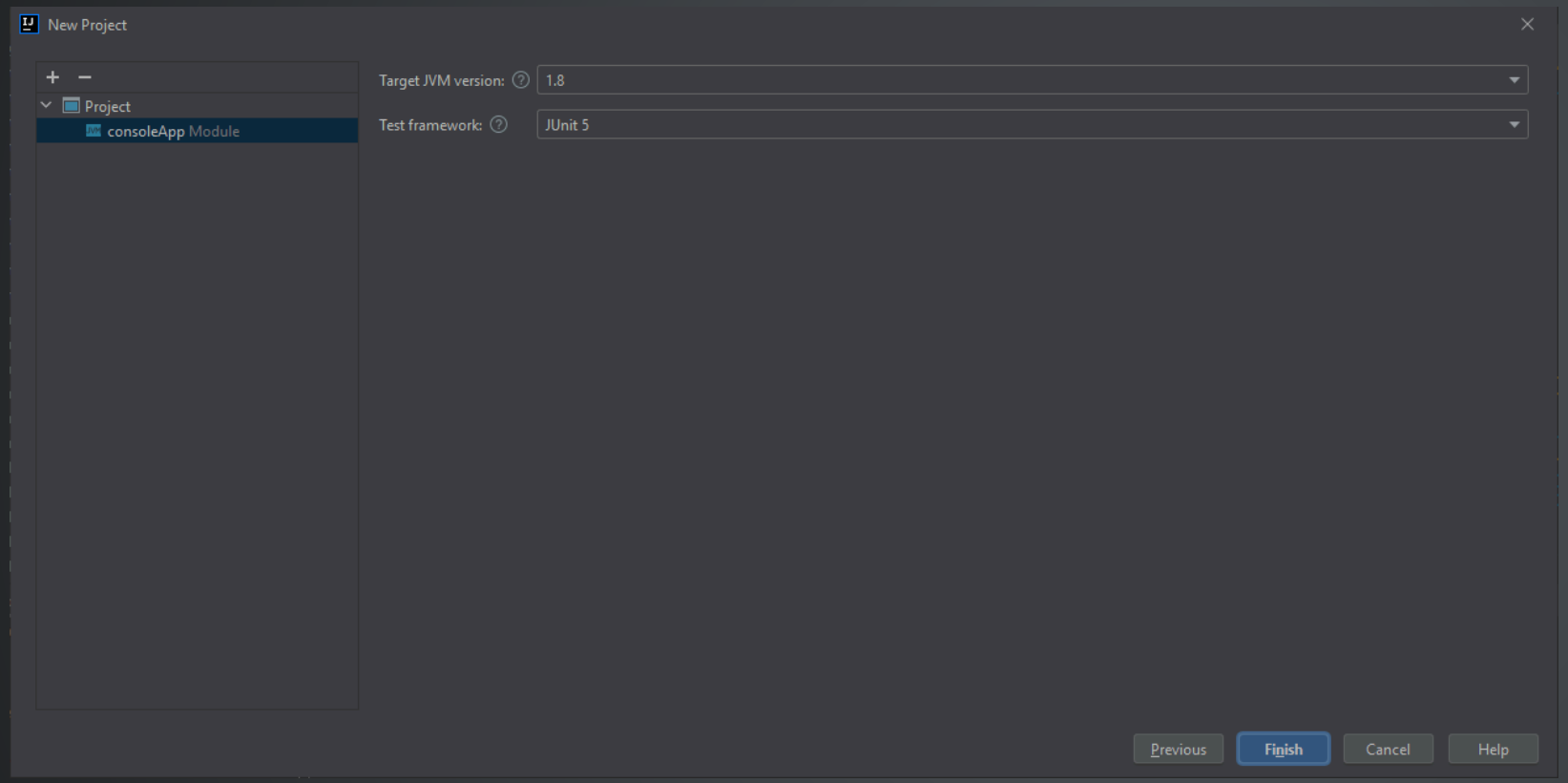
# Kotlin et IntelliJ IDEA

Nommons notre projet, par exemple **HelloWorld** :



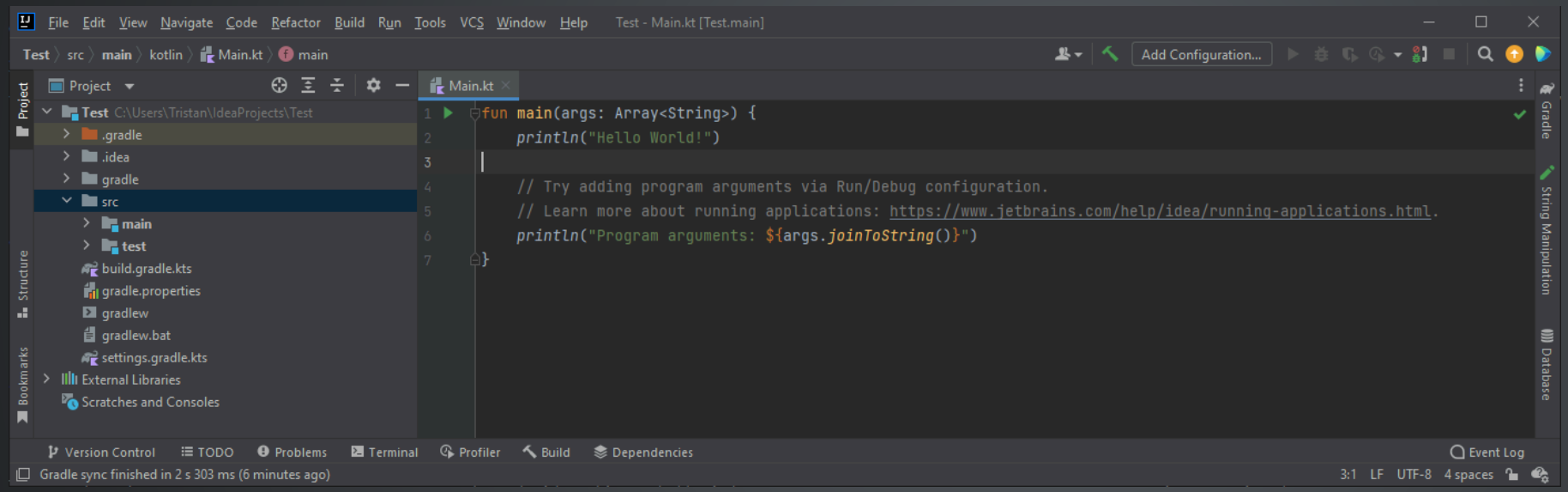
# Kotlin et IntelliJ IDEA

Gardons les paramètres par défaut :



# Kotlin et IntelliJ IDEA

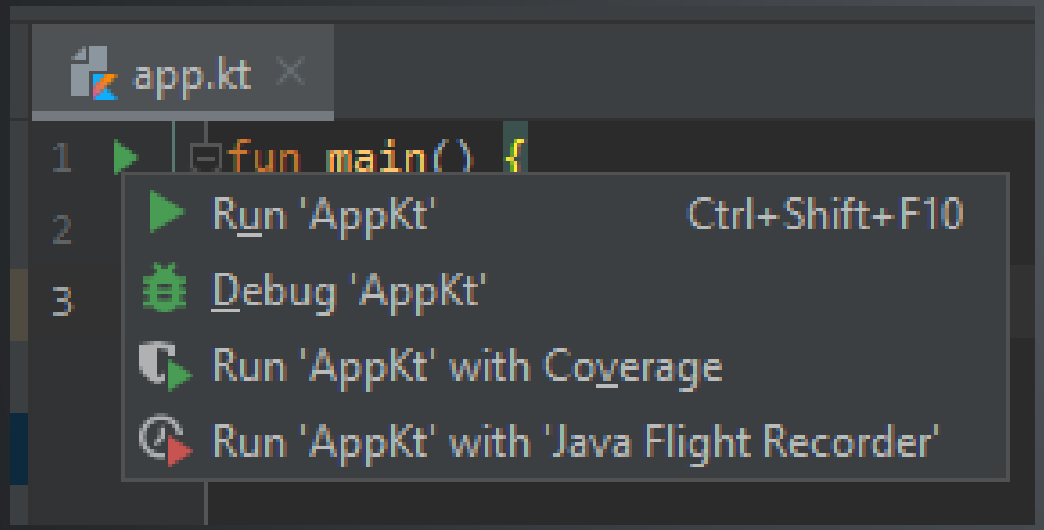
Nous obtenons :



# Kotlin et IntelliJ IDEA

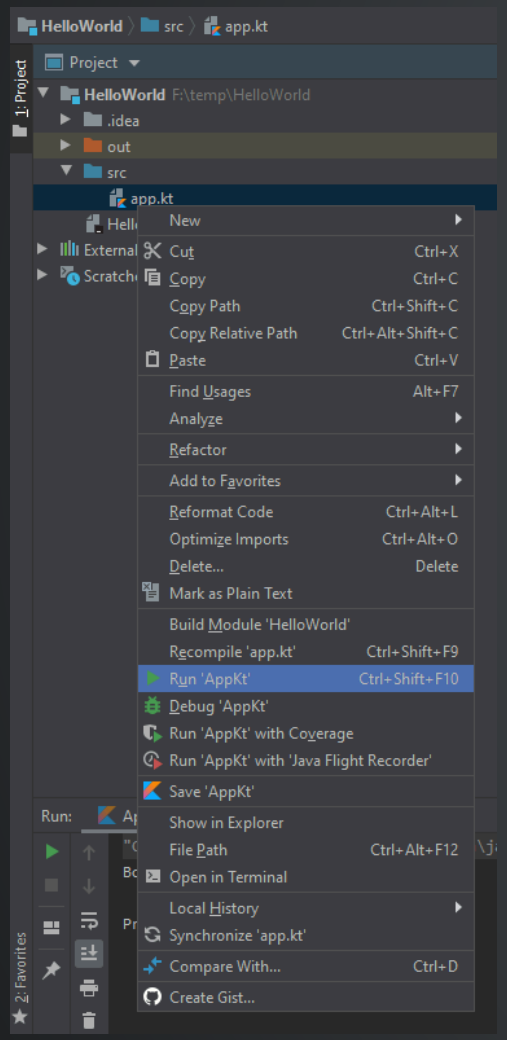
## Exécuter le code

Il y a plusieurs façons de lancer le code, la plus rapide est de cliquer sur le bouton vert **Run** :



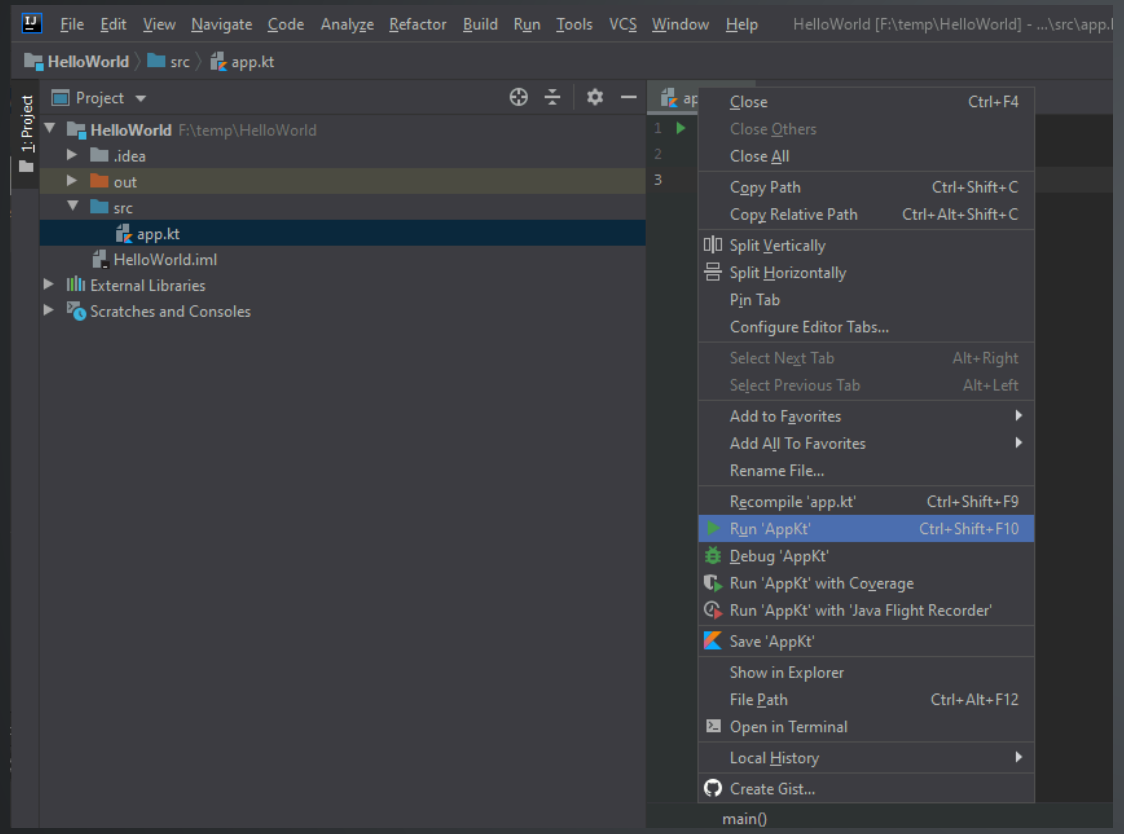
# Exécuter le code

Ou encore :



# Exécuter le code

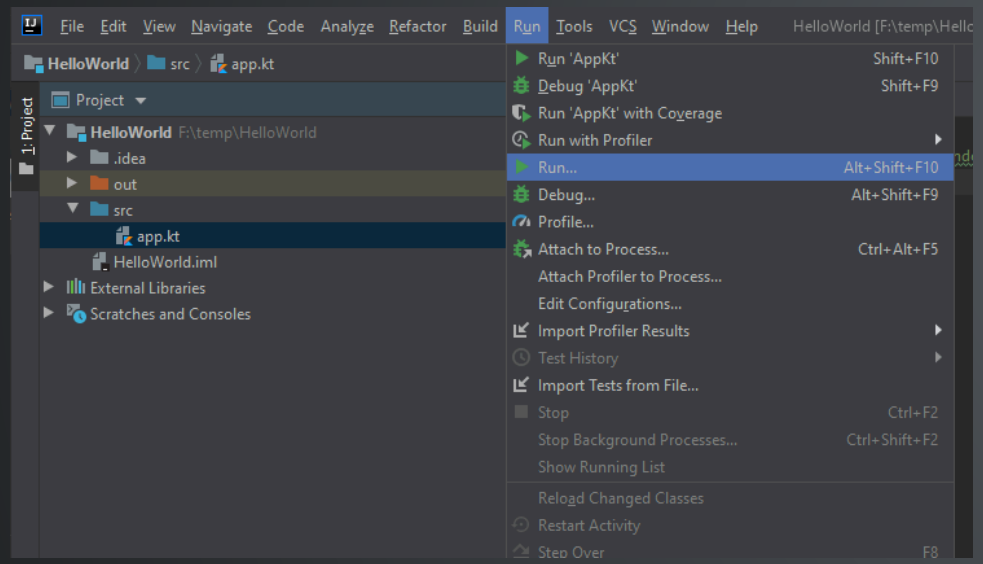
Ou bien :



# Kotlin et IntelliJ IDEA

## Exécuter le code

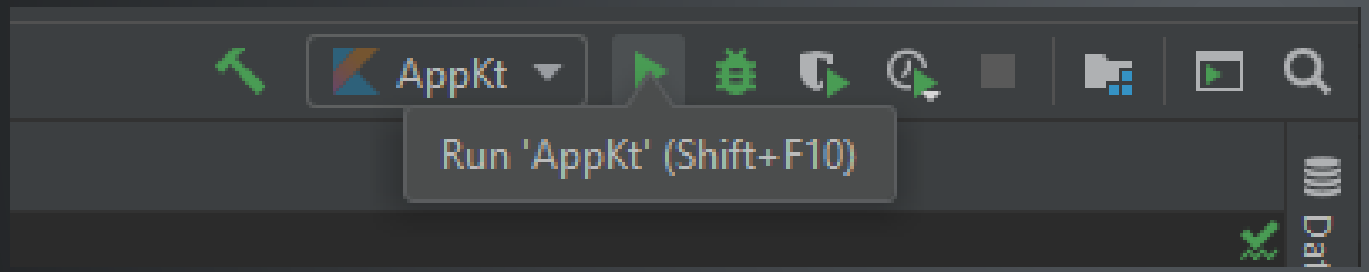
Ou bien encore :



# Kotlin et IntelliJ IDEA

## Exécuter le code

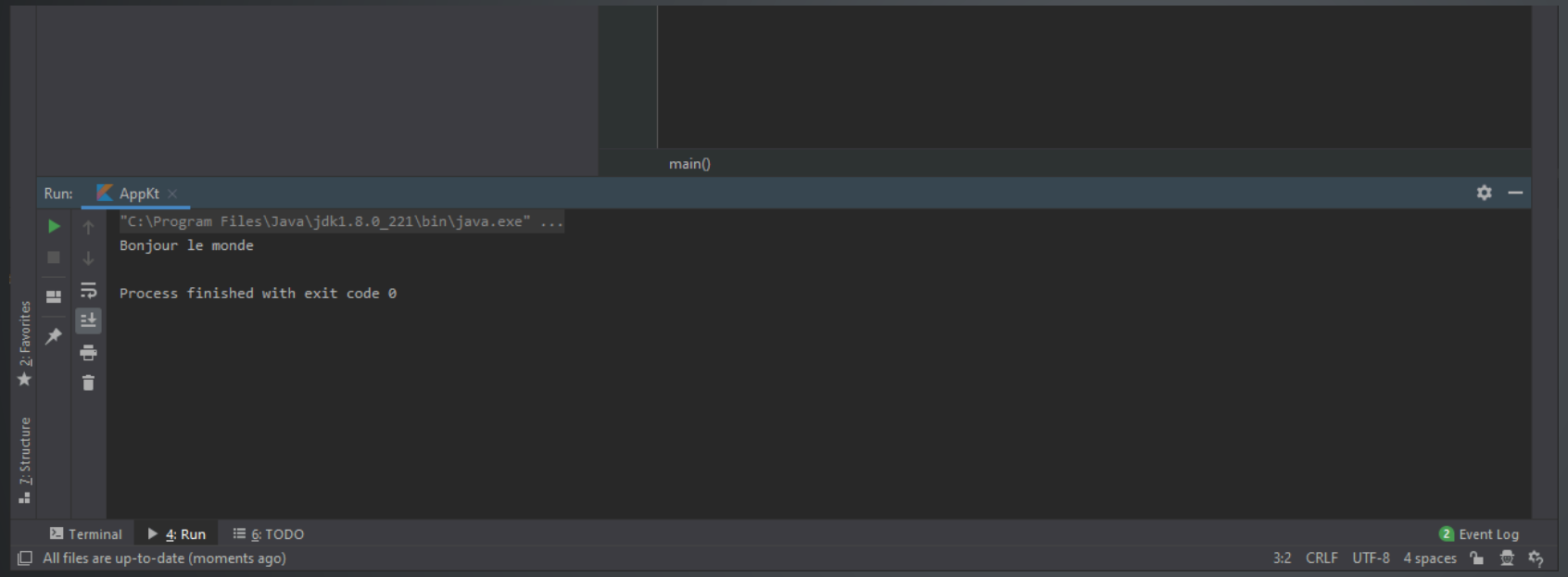
Ce qui permet d'avoir un nouveau raccourci :



# Kotlin et IntelliJ IDEA

## Exécuter le code

Ce qui nous donnera le résultat :



# Kotlin et IntelliJ IDEA

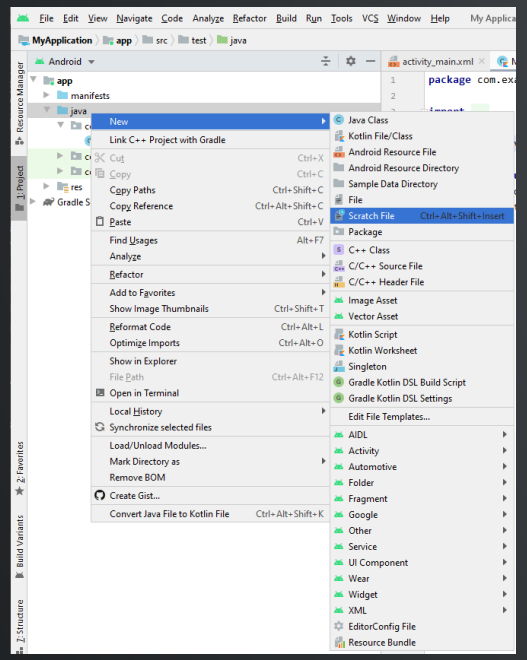
Nous pouvons aussi directement lancer notre code dans :

- Des Scratches : File | New | Scratch file et sélectionner le type Kotlin.
- Une fenêtre REPL : Tools | Kotlin | Kotlin REPL (Control + Return pour valider).

# Le scratch

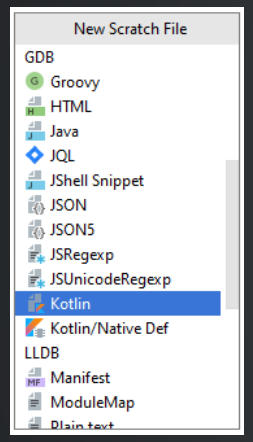
Commençons par ouvrir un espace pour faire nos tests : un scratch.
Au choix :

- Menu : File | New | Scratch file.
- Ctrl + Alt + Shift + Inser.
- Sur Android Studio : Click droit sur app dans notre projet (par exemple), puis : New/Scratch File.



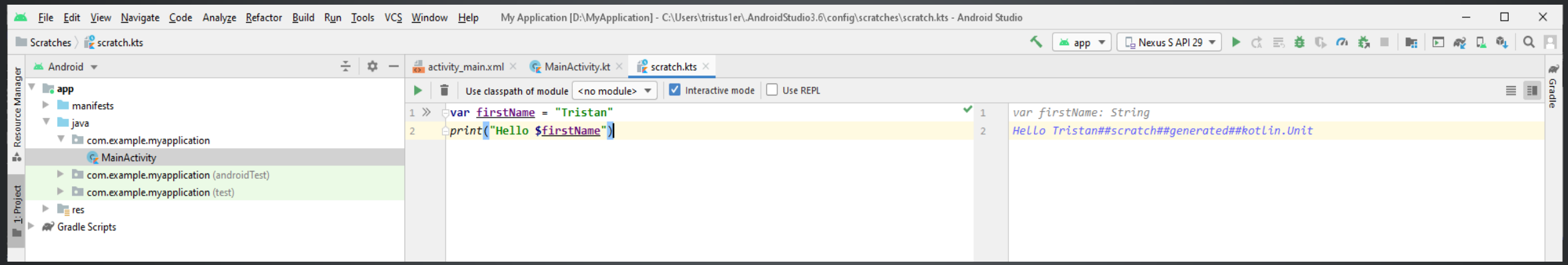
# Kotlin et IntelliJ IDEA

Choisissons le type de fichier : Kotlin :



# Kotlin et IntelliJ IDEA

Pour obtenir le résultat suivant :




Référence vers les chaînes de caractère.

# Les conventions utilisées avec Kotlin

## Règles de nommage

Elles sont identiques à celles en Java (nom des packages et des méthodes). À une différence prête : le nom des méthodes de fabrique est identique à celui de la classe :

```
1 abstract class Foo { ... }
2
3 class FooImpl: Foo { ... }
4
5 fun Foo(): Foo { return FooImpl(...) }
```

Copier

# Les conventions utilisées avec Kotlin

## Nommage des propriétés

Les constantes (propriétés marquées avec un `const`, les propriétés top level ou propriétés `val` d'un objet sans fonction `get` custom, qui contient une valeur profondément immuable), doit utiliser des majuscules, et _ comme séparateur :

```
1 const val MAX_COUNT = 8
2 val USER_NAME_FIELD = "UserName"
```

Copier

Les fonctions top level ou les propriétés d'un objet dont les valeurs peuvent évoluer, utiliserons la notation camel-case :

```
1 val mutableCollection: MutableSet<String> = HashSet()
```

Copier

Le nom des propriétés qui contiennent une référence à un objet Singleton, peuvent utiliser la même règle de nommage :

```
1 val PersonComparator: Comparator<Person> = /*...*/
```

Copier

Pour les valeurs des enums, il est possible d'utiliser les majuscules séparées par des _, ou l'écriture camel-case, en commençant par une majuscule, selon l'usage.

# Les conventions utilisées avec Kotlin

## Choisir le bon nom

Le nom d'une classe est souvent un nom, qui définit la nature de la classe : `List`, `PersonReader`.
Le nom des méthodes est plus souvent un verbe ou une phrase, décrivant son action : `close`, `readPersons`. Le nom doit aussi faire comprendre s'il modifie l'objet ou s'il en retourne un nouveau. Par exemple : `sort` modifie la collection, alors que `sorted` retournera une copie de la collection triée.

Les noms doivent être clairs, sur leur fonctionnement, ils doivent donc éviter de contenir des noms génériques tels que `Manager`, `Wrapper`, etc.

Quand vous utilisez des acronymes dans un nom, mettre en majuscules si sa taille est de 2 lettres (`IOStream`), mais ne mettez que la première lettre en majuscule s'il est plus long (`XmlFormatter`, `HttpInputStream`).

# Bases de Kotlin

- Déclaration de variables en Kotlin.
- Utilisation de variables "Basic Types" en Kotlin.
- Les commentaires.
- Structures conditionnelles If et When.
- Boucles et ranges en Kotlin.
- Collections en Kotlin.
- Packages et imports en Kotlin.

# Déclaration de variables en Kotlin

- val : valeur
- var : variable

```kotlin
val message = "Hello world !"
val message: String = "Hello world !"
var message: String = "Hello world !"
```

Copier

# Déclaration de variables en Kotlin

Le code :

```kotlin
val name: String = "Tristan"
val age: Int = 41
val isDeveloper: Boolean = true
```

Copier

Est équivalent à :

```kotlin
val name = "Tristan"
val age = 41
val isDeveloper = true
```

Copier

# Déclaration de variables en Kotlin

Une valeur peut être assignée dans le même bloc que sa déclaration :

Copier

```
1  import kotlin.random.Random
2
3  fun isUserHappy() = Random.nextInt(0, 100) % 2 == 0
4  fun main() {
5      val message: String
6      if (isUserHappy())
7          message = "That's great"
8      else
9          message = "What's going on?"
10
11     println(message)
12 }
```

# Déclaration de variables en Kotlin

## Null safety

En Kotlin, c'est à nous de préciser qu'une variable peut prendre une valeur nulle.
Le code suivant, ne compilera pas.

```
1 var name: String = "Tristan"
2 name = null
```

Copier

Alors que le code suivant est correct.
Nous précisons que la variable peut prendre une valeur nulle avec le ?.

```
1 var name: String? = "Tristan"
2 name = null
```

Copier

# Déclaration de variables en Kotlin

## Null safety (suite)

Pour utiliser une variable possiblement nulle, nous ne pouvons pas l'utiliser simplement.
L'exemple suivant ne compilera pas :

```
1 var name: String? = "Tristan"
2 name.toUpperCase()
```

Copier

Il faut donc prendre une précaution en utilisant cette variable, en utilisant ? :

```
1 var name: String? = "Tristan"
2 name?.toUpperCase()
```

Copier

Si la valeur de la variable contient null, alors la méthode ne sera pas appelée.

# Déclaration de variables en Kotlin

## Utilisation d'une variable dans une chaîne de caractère

Nous pouvons faire référence à une variable avec le symbole $, par exemple :

```kotlin
val name = "Tristan"
println("Hello $name")
```

Copier

Référence vers les chaînes de caractère.

# Déclaration de variables en Kotlin

## Les constantes

Le mot clé static n'existe pas en Kotlin, donc pour déclarer une constante, on utilise le mot clé `const`.
Le code Java suivant :

```java
public static final String SERVER_URL = "http://my.api.com/";
```

Copier

Deviendra en Kotlin :

```kotlin
const val SERVER_URL = "http://my.api.com/"
```

Copier

# Utilisation de variables "Basic Types" en Kotlin

Plusieurs types basiques sont disponibles en Kotlin :

- Les nombres.
- Les caractères.
- Les booléens.
- Les tableaux.
- Les chaînes de caractères (String).
- Les types spéciaux.

# Nombres

| Type | Size (bits) | Min value | Max value |
|---|---|---|---|
| Byte | 8 | -128 | 127 |
| Short | 16 | -32768 | 32767 |
| Int | 32 | -2,147,483,648 ($-2^{31}$) | 2,147,483,647 ($2^{31} - 1$) |
| Long | 64 | -9,223,372,036,854,775,808 ($-2^{63}$) | 9,223,372,036,854,775,807 ($2^{63} - 1$) |

```
1 val one = 1 // Int
2 val threeBillion = 3000000000 // Long
3 val oneLong = 1L // Long
4 val oneByte: Byte = 1
```

Copier

# Nombres

## Nombres à virgule

| Type | Size (bits) | Significant bits | Exponent bits | Decimal digits |
|---|---|---|---|---|
| Float | 32 | 24 | 8 | 6-7 |
| Double | 64 | 53 | 11 | 15-16 |

```
1 val pi = 3.14 // Double
2 val e = 2.7182818284 // Double
3 val eFloat = 2.7182818284f // Float, actual value is 2.7182817
```

Copier

# Caractères

Les caractères sont représentés par le type Char

```
1 fun check(c: Char) {
2     if (c == 1) { // ERROR: incompatible types
3         // ...
4     }
5 }
```

Copier

Pour écrire un caractère, on utilise la simple "quote" '.
Exemple : 't'.
Les caractères spéciaux sont préfixés par \.
Exemple : \t, \b, \n, \r, \', \", \\ and \$.
Pour les autres caractères, on peut utiliser la syntaxe Unicode : '\uFF00'.

# Caractères

Le charactère peut être convertis explicitement :

Copier

```
fun decimalDigitValue(c: Char): Int {
    if (c !in '0'..'9')
        throw IllegalArgumentException("Out of range")
    return c.toInt() - '0'.toInt() // Explicit conversions to numbers
}
```

# Les booléens

Les booléens sont représentés par le type Boolean qui peut prendre 2 valeurs : true et false.
Les opérations sur les booléens sont les suivantes :

- || – opération logique OU
- && – opération logique ET
- ! - négation

# Les tableaux

Les tableaux sont représentés par la classe Array qui dispose des méthodes `get` et `set`
(qui se transforment en [ ] grâce à la convention de surcharge des opérateurs),
et de la propriété size, entre autres.

```
class Array<T> private constructor() {
    val size: Int
    operator fun get(index: Int): T
    operator fun set(index: Int, value: T): Unit

    operator fun iterator(): Iterator<T>
        // ...
}
```

Copier

# Les tableaux

La méthode `arrayOf(1, 2, 3)` permet de créer des tableaux, `arrayOfNulls()` va créer un tableau de valeurs nulles.
Le constructeur `Array` prend en paramètre le nombre d'éléments et la fonction pour remplir le tableau.

```
1 var array = arrayOf(1, 2, 3)
2 array.forEach { println(it) }
3 var arrayOfNull: Array<Int?> = arrayOfNulls(5)
4 arrayOfNull.forEach { println(it) }
5 val asc = Array(5) { i -> (i * i).toString() }
6 asc.forEach { println(it) }
```

Copier

# Les tableaux

L'opérateur [ ] est équivalent à l'appel des méthodes `get()` et `set()`.

```
1 var array = arrayOf(1, 2, 3)
2 println(array[2])
3 array[0] = 4
4 array.forEach { println(it) }
```

Copier

# Les chaînes de caractères (String)

Les chaînes de caractères sont représentés par la classe String qui sont immutable. Les éléments d'une chaîne de caractère peuvent être accessibles avec l'opérateur [ ].
Il est possible de concaténer des chaînes avec l'opérateur +, même si l'on préférera les templates ($ dans les chaînes de caractères).

```
1 val s = "abc" + 1
2 println(s + "def")
```

Copier

# Les chaînes de caractères (String)

## Modèles (String templates)

Les chaînes de caractère peuvent contenir des expressions (des morceaux de code), qui sont préfixés par $.

```
1 val i = 10
2 println("i = $i") // affiche "i = 10"
```

Copier

Il est possible d'inclure une expression entre ${}, et d'afficher le symbole $ lui-même. Cela fonctionne aussi dans une chaîne brute (raw string).

```
1 println("$name.length is ${name.length}")
2 println("price: ${'$'}9.99")
3
4 val price = """
5 ${'$'}9.99
6 """
```

Copier

# Les types "spéciaux"

Kotlin comporte deux types "spéciaux" :

- Any qui correspond à Object en Java (n'importe quel type d'objet).
- Unit qui correspond à `void` en Java. Autrement dit : rien.

# Les commentaires

Il existe deux types de commentaires, comme en Java :

```
/*
Les commentaires sur plusieurs
lignes, pour pouvoir bien s'exprimer.
Vous pouvez écrire autant que vous voulez.
*/
// ceci est un commentaire uniligne.
```

Copier

# Structures conditionnelles If et When

## L'expression if

En Kotlin `if` est une expression : il retourne une valeur. De ce fait l'opérateur ternaire `?` n'existe plus.

Copier

```kotlin
// Usage traditionnel
var max = a
if (a < b) max = b

// Avec else
var max: Int
if (a > b) {
    max = a
} else {
    max = b
}

// Comme une expression
val max = if (a > b) a else b
```

# Structures conditionnelles If et When

## L'expression if (suite)

Les branches du `if`, peuvent être des blocs, dont la dernière expression est la valeur du bloc :

```
val max = if (a > b) {
    print("Choose a")
    a
} else {
    print("Choose b")
    b
}
```

Copier

Quand le `if` est utilisé comme expression (pour retourner une valeur ou pour assigner une valeur), l'expression doit obligatoirement comporter la branche `else`.

# Structures conditionnelles If et When

## Selon le cas (when)

Le `when` est l'équivalent du `switch` en Java.

```
var apiReponse = 404
when (apiReponse) {
    200 -> "OK"
    404 -> "NOT FOUND"
    401 -> "UNAUTHORIZED"
    403 -> "FORBIDDEN"
    else -> "UNKNOWN"
}
```

Copier

Quand le when est utilisé comme expression (pour retourner une valeur ou pour assigner une valeur), l'expression doit obligatoirement comporter la branche else, sauf si tous les cas sont couverts (d'un enum par exemple).

# Boucles et ranges en Kotlin

## Tant que faire se peut (while)

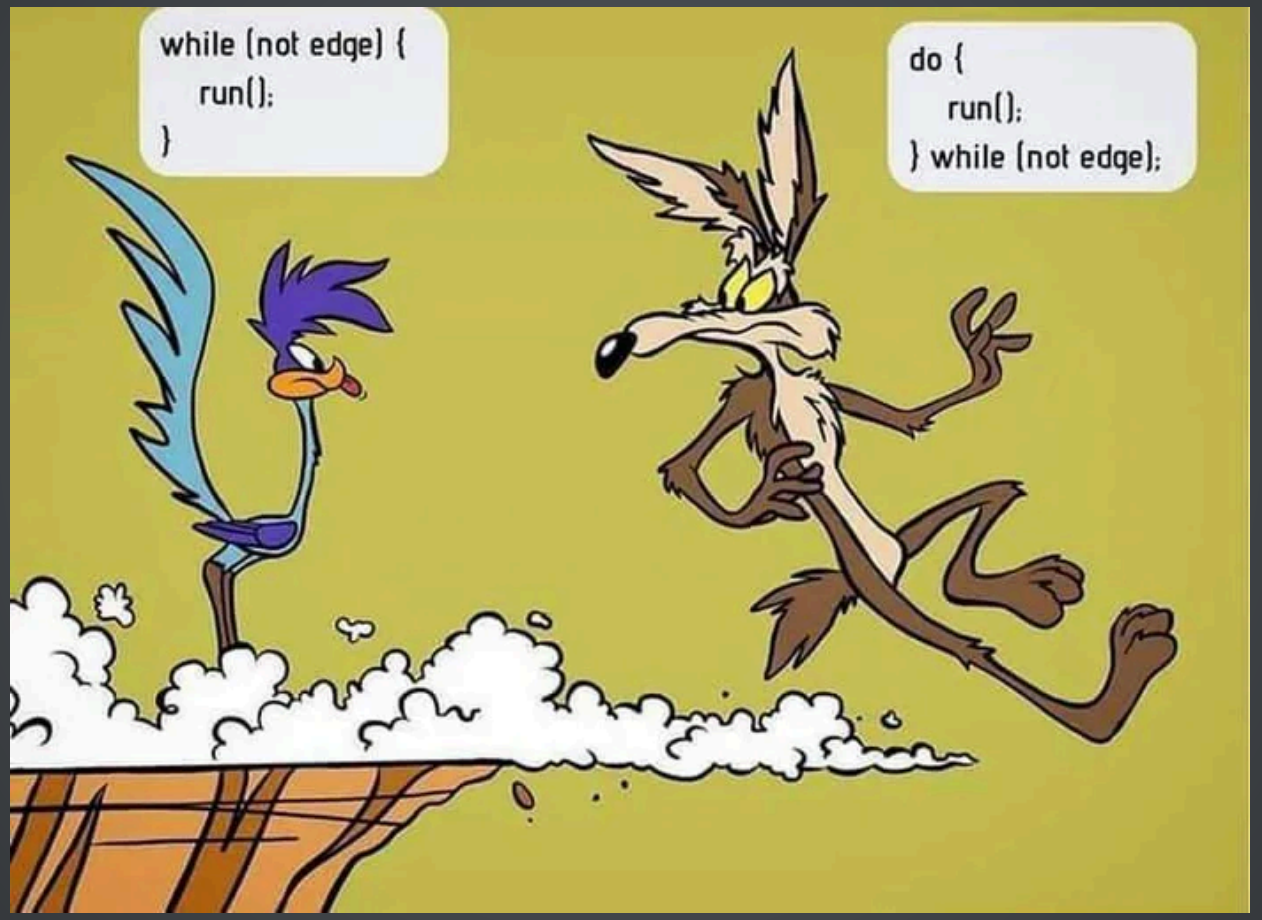
En Kotlin, la syntaxe du `while` est exactement la même qu'en Java :

```
1 var isRaining = true
2 while (isRaining){
3     println("I don't like rain.")
4 }
5
6 do {
7     println("I don't like rain.")
8 } while (isRaining)
```

Copier

# While illustration





# Boucles et ranges en Kotlin

## Boucle pour (for loop)

La boucle `for`, peut itérer sur tout ce qui fournit un itérateur, sa syntaxe est la suivante :

```
for (item in collection) print(item)
```

Copier

Par exemple sur une liste de chaînes de caractères, cela donne :

```
val names = listOf("Jake WHARTON", "Joe BIRCH", "Robert MARTIN")

for(name in names) {
    println("This developer rocks: $name")
}
```

Copier

# Boucles et ranges en Kotlin

## Les intervalles

Il est possible d'itérer sur des intervalles de valeur, qui sont définis avec la méthode `rangeTo()`, correspondant à l'opérateur `...`
Cet opérateur est souvent complété par les fonctions `in` ou `!in`. Exemple :

```
1 if (i in 1..4) {  // equivalent à 1 <= i && i <= 4
2     println(i)
3 }
```

Copier

Pour définir un intervalle, souvent utilisés dans les boucles for, la syntaxe est la suivante :

```
1 for (i in 1..4) println(i)
```

Copier

Pour utiliser l'ordre décroissant, la syntaxe sera :

```
1 for (i in 4 downTo 1) println(i)
```

Copier

# Boucles et ranges en Kotlin

## Break et continue

Ils fonctionnent de la même manière qu'en Java.

# Collections en Kotlin

Kotlin propose plusieurs structures pour gérer des groupes d'objets en nombre variables (possiblement 0).
Si vous êtes familiers de ces concepts, passons à la suite. Sinon continuons ...

Une collection est une structure qui regroupe des objets de même type. Ces objets sont appelés des éléments ou des items.

Il existe plusieurs types de collection :

- Une liste (List), est une collection ordonnée d'objets auxquels nous pouvons accéder via leur position/index (un nombre entier). Un élément peut être présent une ou plusieurs fois dans la liste. Exemple d'une phrase qui comporte des mots, dont l'ordre est important, et qui peuvent se répéter.
- L'ensemble (Set), est une collection d'éléments uniques. L'ordre dans un ensemble n'a pas d'importance. Par exemple les lettres de l'alphabet.

# Collections en Kotlin

- Le dictionnaire (Map), est un ensemble d'éléments composés d'une paire (clé-valeur). Les clés ont des valeurs uniques qui désignent un seul objet de la collection. Les valeurs peuvent apparaître en plusieurs fois. Cette structure est employée pour stocker une connexion logique entre 2 objets, par exemple, un numéro d'employé et sa fiche descriptive.

Le comportement de chaque type de collection sera toujours le même, peu importe le type des objets stockés dans ces structures.

# Collections en Kotlin

En Kotlin, il y a deux types principaux de collections :

- En lecture seule (read-only/immutable).
- En lecture/écriture (mutable) (ajout, retrait, modification des éléments).

Note : Une collection mutable peut être stockée dans une valeur (val) :

```
1 val numbers = mutableListOf("one", "two", "three", "four")
2 numbers.add("five")   // this is OK
3 //numbers = mutableListOf("six", "seven")      // compilation error
```

Copier

# Collections en Kotlin

## Plusieurs exemples de collections `List` et `Set` :

```
val listOfNames = listOf("Jake WHARTON", "Joe BIRCH", "Robert MARTIN")
listOfNames[0]
//listOfNames[0] = "Mathieu NEBRA" // Error: List is immutable

val mutableListOfNames = mutableListOf("Jake WHARTON", "Joe BIRCH", "Robert MARTIN")
mutableListOfNames[0]
mutableListOfNames[0] = "Mathieu NEBRA" // OK

val setOfNames = setOf("Jake WHARTON", "Joe BIRCH", "Robert MARTIN")
setOfNames.first()
//setOfNames.add("Mathieu NEBRA") // Error: Set is immutable

val mutableSetOfNames = mutableSetOf("Jake WHARTON", "Joe BIRCH", "Robert MARTIN")
mutableSetOfNames.first()
mutableSetOfNames.add("Mathieu NEBRA") // OK
```

Copier

## Exemple de tableau et de `Map` :

```
var arrayOfNames = arrayOf("Jake WHARTON", "Joe BIRCH", "Robert MARTIN")
var mapOfNames = mapOf(0 to "Jake WHARTON", 1 to "Joe BIRCH", 2 to "Robert MARTIN")
```

Copier

# Packages et imports en Kotlin

Un fichier de code source peut commencer par la déclaration d'un package :

```kotlin
package org.example

fun printMessage() { /*...*/ }
class Message { /*...*/ }

// ...
```

Copier

Tout le contenu (tels que les classes et les fonctions) du fichier de code source appartiendront au package déclaré. Dans l'exemple ci-dessus, le nom complet de `printMessage()` est `org.example.printMessage()`, de la même manière, le nom complet de `Message` est `org.example.Message`.
Si le nom du package n'est pas précisé, le contenu du fichier appartient au package par défaut qui n'a pas de nom.

# Packages et imports en Kotlin

## Imports

Chaque fichier peut faire appel à des classes externes, pour cela, on va faire appel à des `import` :

```
1 import org.example.Message // Message is now accessible without qualification
```

Copier

Il est aussi possible d'importer tout un groupe de classes avec l'utilisation de * :

```
1 import org.example.* // everything in 'org.example' becomes accessible
```

Copier

# Packages et imports en Kotlin

## Imports divers

Le mot clé `import` est aussi utilisé pour importer d'autres types que des classes, telles que :

- Fonctions et propriétés top level
- Fonctions et propriétés déclarées dans des déclarations object
- Fonctions d'extensions
- Constantes d'un `ENUM`

Exercices : faisons les deux premiers exercices :
Practice: Kotlin Fundamentals

# Les fonctions - Partie 1

- Fonctions en Kotlin.
- Fonctions "expression seule".
- Paramètres des fonctions en Kotlin.
- Returns et Local Returns en Kotlin.

# Fonctions en Kotlin

## Déclaration

On utilise le mot clé fun

```
1 fun double(x: Int): Int {
2     return 2 * x
3 }
```

Copier

# Fonctions en Kotlin

## Utilisation

Appel traditionnel :

```
1 val result = double(2)
```

Copier

Appel d'une fonction membre d'un objet :

```
1 Stream().read() // créé une instance de la class Stream et appelle la fonction read()
```

Copier

# Fonctions "expression seule"

## Les fonctions "expression seule" (single-expression function)

Il n'est pas nécessaire d'utiliser les accolades, un simple = peut les remplacer :

```
1 fun double(x: Int): Int = x * 2
```

Copier

Il n'est même pas nécessaire de préciser le type de retour, qui est inféré par le compilateur :

```
1 fun double(x: Int) = x * 2
```

Copier

# Paramètres des fonctions en Kotlin

## Les paramètres simples

Chaque paramètre (explicitement typé), est séparé par une virgule :

```
1 fun powerOf(number: Int, exponent: Int) { /*...*/ }
```

Copier

# Paramètres des fonctions en Kotlin

## Paramètres par défauts

Les paramètres peuvent avoir une valeur par défaut (exprimée avec =) quand un argument n'est pas précisé :

```
1 fun read(b: Array<Byte>, off: Int = 0, len: Int = b.size) { /*...*/ }
```

Copier

# Paramètres des fonctions en Kotlin

## Paramètres nommés

Chaque paramètre peut être nommé explicitement. Cela est particulièrement pratique pour les fonctions qui ont beaucoup de paramètres ou des paramètres par défauts :

```
fun reformat(str: String,
             normalizeCase: Boolean = true,
             upperCaseFirstLetter: Boolean = true,
             divideByCamelHumps: Boolean = false,
             wordSeparator: Char = ' ') {
/*...*/
}
```

Copier

# Paramètres des fonctions en Kotlin

## Paramètres nommés utilisation

Nous pouvons utiliser les paramètres par défaut :

```
reformat(str)
```

Copier

Nous pouvons préciser tous les paramètres :

```
reformat(str, true, true, false, '_')
```

Copier

L'appel avec tous les paramètres nommés est bien plus clair :

```
reformat(str,
    normalizeCase = true,
    upperCaseFirstLetter = true,
    divideByCamelHumps = false,
    wordSeparator = '_'
)
```

Copier

# Returns en Kotlin

## Retour d'une fonction

Une fonction qui ne retourne rien, retourne implicitement un type `Unit`.
Il n'est pas besoin de retourner explicitement la valeur :

```
fun printHello(name: String?): Unit {
    if (name != null)
        println("Hello ${name}")
    else
        println("Hi there!")
    // "return Unit" or "return" is optional
}
```

Copier

Il n'est même pas obligatoire de préciser ce type retour :

```
fun printHello(name: String?) {
    ...
}
```

Copier

# Classes en Kotlin

- La POO.
- Une classe.
- Les attributs.
- Méthodes (Functions Members).
- Visibilité des membres en Kotlin.
- Héritage en Kotlin.
- La surcharge.
- Abstract Classes en Kotlin.
- Interface en Kotlin.
- Data Classes en Kotlin.
- Enum Classes en Kotlin.

# La POO : Programmation Orientée Objet

L'objectif de la Programmation Orientée Objet est d'organiser le code afin de mieux pouvoir le réutiliser.
Pour cela, nous utilisons l'encapsulation qui permet de :

- Rassembler dans une même structure :
  - Attributs (données) ou "variables membres".
  - Méthodes (fonctions).
- Garantir l’intégrité des données en :
  - Ne laissant visible que ce qui doit être réellement utilisé (visibilité).
  - N’accédant aux données que par des méthodes.

# La POO : Programmation Orientée Objet

Classe = description abstraite d'un objet.
Instancier une classe = créer un objet sur son modèle (grâce au constructeur).
Propriété = attribut accessible par un getter et/ou setter.

# Une classe

## Déclaration

On utilise le mot clé class

```
1 class Invoice { /*...*/ }
```

Copier

Une déclaration de classe consiste en : un nom, un entête (qui spécifie le type des paramètres, le constructeur primaire, etc.) et le corps de la classe, le tout entouré d'accolades. L'entête et le corps de la classe sont optionnels. Si la classe n'a pas de corps, les accolades sont optionnelles :

```
1 class Empty
```

Copier

# Une classe

## Le constructeur

Une classe peut avoir un **constructeur primaire** et un ou plusieurs **constructeurs secondaires**. Le constructeur primaire fait partie intégrante de l'entête : il est placé juste après le nom de la classe.

```
1 class Person constructor(firstName: String) { /*...*/ }
```

Copier

Si le constructeur n'a pas d'annotations, ou de modificateurs de visibilité, le mot clé `constructor` n'est pas obligatoire :

```
1 class Person(firstName: String) { /*...*/ }
```

Copier

# Une classe

## La forme concise du constructeur

C'est cette forme que l'on utilisera de préférence :

```
1 class Person(val firstName: String, val lastName: String, var age: Int) { /*...*/ }
```

Copier

Les propriétés peuvent être :

- En lecture seule : `val`
- En lecture ET écriture (mutable) : `var`

# Une classe

## Exemple de classe User

En Java une classe User serait écrite comme suit :

```java
public class User {

    // PROPERTIES
    private String email;
    private String password;
    private int age;

    // CONSTRUCTOR
    public User(String email, String password, int age) {
        this.email = email;
        this.password = password;
        this.age = age;
    }

    // GETTERS
    public String getEmail() { return email; }
    public String getPassword() { return password; }
```

Copier

En Kotlin son équivalent est :

```kotlin
class User(var email: String, var password: String, var age: Int)
```

Copier

# Une classe

## Les valeurs par défaut

En Kotlin, pas besoin de définir plusieurs constructeurs pour gérer les valeurs des paramètres par défaut, il suffit de préciser leur valeur avec le signe = dans le constructeur :

```
1 class Customer(val customerName: String = "")
```

Copier

# Une classe

## Utilisation (instantiation) d'une classe

En Kotlin, il n'y a pas de new, on utilise directement le nom de la classe :

```
1 val user = User("hello@gmail.com", "azerty", 41)
```

Copier

Pour accéder aux champs, la notation à `.` est utilisée :

```
1 val user = User("hello@gmail.com", "azerty", 41)
2 println(user.email) // Getter
3 user.email = "my_new_email@gmail.com" // Setter
4 println(user.email) // Getter
```

Copier

# Les attributs

Les propriétés en Kotlin peuvent être déclarées en lecture/écriture (mutable) en utilisant le mot clé `var`, ou en lecture seule (immutable) en utilisant le mot clé `val` :

```
class Address {
    var name: String = "Holmes, Sherlock"
    var street: String = "221b Baker Street"
    var city: String = "London"
    var state: String? = null
    var zip: String = "NW1 6XE"
}
```

Copier

Pour accéder aux propriétés, il suffit d'utiliser son nom :

```
fun copyAddress(address: Address): Address {
    val result = Address() // there's no 'new' keyword in Kotlin
    result.name = address.name // accessors are called
    result.street = address.street
    // ...
    return result
}
```

Copier

# Méthodes (Functions Members)

Une méthode membre d'une classe, est définie comme suit :

```
class Sample() {
    fun foo() { print("Foo") }
}
```

Copier

Comme déjà vu, pour appeler une telle méthode, il suffit d'utiliser la notation à point :

```
Sample().foo() // creates instance of class Sample and calls foo
```

Copier

# Visibilité des membres en Kotlin

En Kotlin la visibilité par défaut est `public`, les modifieurs de visibilité disponibles sont :

- `private` : Un membre déclaré comme `private` sera visible uniquement dans la classe où il est déclaré.
- `protected` : Un membre déclaré comme `protected` sera visible uniquement dans la classe où il est déclaré ET dans ses sous-classes (via l'héritage).
- `internal` : Un membre déclaré comme `internal` sera visible par tous ceux du même module. Un module est un ensemble de fichiers compilés ensemble (comme une librairie Gradle ou Maven, par exemple).
- `public` : Un membre déclaré comme `public` sera visible partout et par tout le monde.

# Héritage en Kotlin

Toutes les classes en Kotlin héritent de la super class `Any` (équivalent à `Object` en Java), qui est la superclasse par défaut de toutes les classes qui n'ont pas déclaré de super type :

```
1 class Example // Implicitly inherits from Any
```

Copier

La classe Any à 3 méthodes : `equals()`, `hashCode()` et `toString()`.
Pour déclarer un super type, la syntaxe est la suivante :

```
1 open class Base(p: Int)
2
3 class Derived(p: Int): Base(p)
```

Copier

# La surcharge

## La surcharge des méthodes

En Kotlin, tout doit être explicite, une méthode qui peut être surchargée sera marquée avec le modifieur `open` :

```
1 open class Shape {
2     open fun draw() { /*...*/ }
3     fun fill() { /*...*/ }
4 }
5
6 class Circle(): Shape() {
7     override fun draw() { /*...*/ }
8 }
```

Copier

Une méthode qui redéfinit une autre de la classe parente est elle-même redéfinissable, à moins de la marquer comme `final` :

```
1 open class Rectangle(): Shape() {
2     final override fun draw() { /*...*/ }
3 }
```

Copier

# La surcharge

## L'appel à la classe parente

Comme en Java, le mot clé pour appeler le parent est `super` :

Copier

```
open class Rectangle {
    open fun draw() { println("Drawing a rectangle") }
    val borderColor: String get() = "black"
}

class FilledRectangle: Rectangle() {
    override fun draw() {
        super.draw()
        println("Filling the rectangle")
    }

    val fillColor: String get() = super.borderColor
}
```

# Classes abstraites en Kotlin

Une classe et quelques membres peuvent être déclarés `abstract`. Un membre abstrait n'a pas d'implémentation dans la classe. Le mot clé open n'est pas nécessaire, dans le cas d'une classe ou méthode abstraite, car cela est évident.
Note : il est possible de surcharger un membre non abstrait par un abstrait :

Copier

```
open class Polygon {
    open fun draw() {}
}

abstract class Rectangle: Polygon() {
    override abstract fun draw()
}
```

# L'objet compagnon

Nous utiliseront principalement l'objet compagnon comme déclaration de méthodes ou d'attributs statiques à la classe.

Copier

```
class MyClass1 {
    companion object {
        private const val TAG = "MyClass1"
        fun myMethod() { ... }
    }
}
```

# Interface en Kotlin

Les interfaces en Kotlin peuvent contenir des méthodes abstraites, mais aussi des méthodes implémentées.
Ce qui les différencies d'une classe abstraite, est le fait qu'elles ne contiennent pas d'état.
Elles peuvent comporter des propriétés, mais elles doivent être abstraites ou fournir une implémentation pour les accesseurs.
Une interface est définie en utilisant le mot clé `interface` :

```
interface MyInterface {
    fun bar()
    fun foo() {
      // optional body
    }
}
```

Copier

Implémenter une interface :

```
class Child: MyInterface {
    override fun bar() {
        // body
    }
}
```

Copier

# Data Classes en Kotlin

Nous écrivons régulièrement des classes, dont le rôle est de contenir de l'information. Dans ce genre de classes, des fonctionnalités et des fonctions utilitaires sont souvent déduites mécaniquement des données. En Kotlin, ces classes sont appelées data class, et sont donc marquées `data` :

```kotlin
data class User(val name: String, val age: Int)
```

Copier

Le compilateur va générer les membres suivant, en utilisant toutes les propriétés déclarées dans le constructeur principal :

- `equals()/hashCode()`.
- `toString()` sous la forme `"User(name=Tristan, age=40)"`.
- Les fonctions `componentN()` correspondant aux propriétés dans leur ordre de déclaration.
- La fonction `copy()`.

# Data Classes en Kotlin

## La copie

Il arrive régulièrement de devoir copier un objet, et de modifier quelques-unes de ses propriétés, tout en gardant le reste intact. C'est le but de la fonction générée `copy()`. Pour la classe `User` ci-dessous, l'implémentation serait la suivante :

```
1 fun copy(name: String = this.name, age: Int = this.age) = User(name, age)
```

Copier

Ce qui nous permet d'écrire :

```
1 val jack = User(name = "Jack", age = 1)
2 val olderJack = jack.copy(age = 2)
```

Copier

# Enum Classes en Kotlin

L'usage le plus simple d'une classe enum est d'implémenter une énumération sûre :

```kotlin
enum class Direction {
    NORTH, SOUTH, WEST, EAST
}
```

Copier

Chaque constante de l'énumération est un objet. Les constantes sont séparées par une virgule.

# Les fonctions - Partie 2

- Expressions Lambda en Kotlin.
- Extensions de fonctions en Kotlin.

# Lambda expression en Kotlin

Les fonctions en Kotlin sont des fonctions "première classe" (first-class), c'est-à-dire qu'elles peuvent être stockées dans des variables, des structures de donnée, passées en argument et retournées depuis des fonctions d'ordre supérieur (higher-order functions). Vous pouvez utiliser les fonctions, comme n'importe quel autre type classique.

# Lambda expression en Kotlin

## Fonctions d'ordre supérieur

Une fonction d'ordre supérieur est une fonction qui prend en paramètre ou retourne une fonction.
Un très bon exemple est la fonctionnalité `fold` pour les collections, qui prend une valeur initiale pour l'accumulateur et une fonction pour combiner les items. Le résultat est obtenu en appliquant la fonction sur l'item courant et l'accumulateur, pour en faire changer la valeur. Exemple :

```kotlin
fun <T, R> Collection<T>.fold(
    initial: R,
    combine: (acc: R, nextElement: T) -> R
    ): R {
    var accumulator: R = initial
    for (element: T in this) {
        accumulator = combine(accumulator, element)
    }
    return accumulator
}
```

Copier

Dans le code ci-dessus, le paramètre `combine` à un type de fonction `(R, T) -> R`, donc il accepte une fonction qui prend 2 arguments, de types `R` et `T` et retourne une valeur de type `R`. Cette fonction est appelée dans une boucle `for`, et la valeur de retour est ensuite assignée à l'accumulateur.

# Lambda expression en Kotlin

Pour appeler la méthode `fold`, nous devons passer une instance de type fonction en argument et les expressions lambdas sont couramment utilisées à cet usage :

Copier

```kotlin
val items = listOf(1, 2, 3, 4, 5)

// Lambdas are code blocks enclosed in curly braces.
items.fold(0, {
    // When a lambda has parameters, they go first, followed by '->'
    acc: Int, i: Int ->
    println("acc = $acc, i = $i, ")
    val result = acc + i
    println("result = $result")
    // The last expression in a lambda is considered the return value:
    result
})
```

Copier

```kotlin
// Parameter types in a lambda are optional if they can be inferred:
val joinedToString = items.fold("Elements:", { acc, i -> "$acc $i" })

// Function references can also be used for higher-order function calls:
val product = items.fold(1, Int::times)
```

# Lambda expression en Kotlin

## Expression lambda et fonctions anonymes

Une expression lambda ou une fonction anonyme, sont des fonctions littérales, c'est-à-dire des fonctions qui ne sont pas déclarées, mais qui sont passées directement comme expression. Dans l'exemple suivant :

```
1 max(strings, { a, b -> a.length < b.length })
```

Copier

La fonction `max` est une fonction d'ordre supérieur, qui prend une fonction en second paramètre. Cet argument est une expression, qui est elle-même une fonction : une fonction littérale qui correspond à la fonction nommée suivante :

```
1 fun compare(a: String, b: String): Boolean = a.length < b.length
```

Copier

# Lambda expression en Kotlin

## Syntaxe de l'expression lambda

La syntaxe complète d'une expression lambda est la suivante :

```
1 val sum: (Int, Int) -> Int = { x: Int, y: Int -> x + y }
```

Copier

Une expression lambda est toujours entourée d'accolades. Les paramètres dans la syntaxe complète, sont déclarés dans ces accolades et leur typage est optionnel. Le corps est situé après la `->`. Si le type de retour inféré de la lambda n'est pas `Unit`, la dernière (et possiblement seule) expression dans le corps de la lambda est considéré comme la valeur de retour.
En retirant toutes les annotations optionnelles, le code devient :

```
1 val sum = { x: Int, y: Int -> x + y }
```

Copier

# Lambda expression en Kotlin

## Passer une lambda en paramètre

En Kotlin, il y a une convention : si le dernier paramètre d'une fonction est une fonction, alors l'expression lambda, passée en paramètre peut être placée à l'extérieur des parenthèses :

```
1 val product = items.fold(1) { acc, e -> acc * e }
```

Copier

Cette notation est appelée lambda de fin (trailing lambda).
Si la lambda, est le seul argument, alors les parenthèses peuvent être complètement omises :

```
1 run { println("...") }
```

Copier

# Lambda expression en Kotlin

## it : le nom implicite du paramètre unique

Il est courant qu'une expression lambda ait un unique paramètre. Si le compilateur peut déterminer la signature de lui-même, alors il n'est pas obligatoire de déclarer le paramètre unique et omettre par la même occasion `->`. Le paramètre sera implicitement déclaré avec le mon `it` :

```
1 var ints = listOf(0, 1,-2,3,-1)
2 ints.filter { it > 0 } // this literal is of type '(it: Int) -> Boolean'
```

Copier

# Simplification d'une méthode Java en Kotlin

Exemple de code en Java :

```java
button.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        Log.d(TAG, "User clicked button");
    }
});
```

Copier

# Simplification d'une méthode Java en Kotlin

Qui va afficher un message de log dans la console, de type DEBUG :

Copier

```
button.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        Log.d(TAG, "User clicked button");
    }
});
```

# Simplification d'une méthode Java en Kotlin

La version utilisant les lambdas Java :

```
1 button.setOnClickListener ( v -> {
2     Log.d(TAG, "User clicked button");
3 });
```

Copier

# Simplification d'une méthode Java en Kotlin

L'équivalent en Kotlin :

```
1 button.setOnClickListener( { v: View -> Log.d(TAG, "User clicked button"); })
```

Copier

# Simplification d'une méthode Java en Kotlin

Le ; en fin de ligne n'est pas nécessaire :

```
1 button.setOnClickListener( { v: View -> Log.d(TAG, "User clicked button"); } )
```

Copier

# Simplification d'une méthode Java en Kotlin

La lambda peut être sortie des parenthèses de la méthode :

```
1 button.setOnClickListener( { v: View -> Log.d(TAG, "User clicked button")  } )
```

Copier

# Simplification d'une méthode Java en Kotlin

Les parenthèses de la méthode ne sont pas nécessaires :

```
1 button.setOnClickListener() { v: View -> Log.d(TAG, "User clicked button")  }
```

Copier

# Simplification d'une méthode Java en Kotlin

Le type du paramètre est inféré par le compilateur :

```
1 button.setOnClickListener   { v: View -> Log.d(TAG, "User clicked button")  }
```

Copier

# Simplification d'une méthode Java en Kotlin

Le paramètre est unique, et n'est pas utilisé dans notre expression, donc inutile :

```
1 button.setOnClickListener   { v        -> Log.d(TAG, "User clicked button")  }
```

Copier

# Simplification d'une méthode Java en Kotlin

Ce qui nous donne finalement le code suivant ...

```
1 button.setOnClickListener   {           Log.d(TAG, "User clicked button")  }
```

Copier

# Simplification d'une méthode Java en Kotlin

Avec les espaces superflus en moins, cela donne :

```
1 button.setOnClickListener{ Log.d( TAG, "User clicked button" ) }
```

Copier

Qui correspond au code Java :

```
1 button.setOnClickListener(new View.OnClickListener() {
2     @Override
3     public void onClick(View v) {
4         Log.d(TAG, "User clicked button");
5     }
6 });
```

Copier

Ou encore :

```
1 button.setOnClickListener ( v -> {
2     Log.d(TAG, "User clicked button");
3 });
```

Copier

Exercices : reprenons nos execices :

Practice: Kotlin Fundamentals

# Extensions

Kotlin permet d'étendre les fonctionnalités d'une classe, sans avoir besoin d'en hériter ni d'utiliser le design pattern du décorateur. Il faut utiliser la déclaration spéciale appelée : extensions. Par exemple, il est possible d'ajouter des nouvelles fonctions à une classe d'une librairie externe, dont le code source n'est pas disponible. Il est possible d'utiliser ces fonctions comme si elles faisaient partie intégrante du code source original. Ce mécanisme est appelé *extension de fonctions*. Il est aussi possible d'ajouter des propriétés à une classe existante en utilisant le mécanisme d'*extension de propriétés*.

# Extensions

## Extensions de fonctions en Kotlin

Pour déclarer une fonction d'extension, nous devons la préfixer par le nom du type receveur, c'est-à-dire le type qui va être étendu. L'exemple ci-dessous ajoute la fonction `swap` à `MutableList<Int>`

```
fun MutableList<Int>.swap(index1: Int, index2: Int) {
    val tmp = this[index1] // 'this' corresponds to the list
    this[index1] = this[index2]
    this[index2] = tmp
}
```

Copier

Le mot clé `this`, dans la fonction d'extension, fait référence à l'objet receveur (qui est juste avant le point). Maintenant, nous pouvons appeler cette fonction sur tous les objets de type `MutableList<Int>`.

```
val list = mutableListOf(1, 2, 3)
list.swap(0, 2) // 'this' inside 'swap()' will hold the value of 'list'
```

Copier

Testez ce code pour en comprendre la puissance.

# Délégation

- Délégation de propriétés en Kotlin

# Délégation de propriétés en Kotlin

Comme vous le savez maintenant, pour utiliser une propriété en Kotlin, on utilise simplement son nom, préfixé d'un `.` :

```
class Foo {
    var prop: String? = null
}

val foo = Foo()
foo.prop = "something"
val another = foo.prop
```

Copier

Vous pouvez aussi écrire des getters et des setters sur mesure :

```
var str: String
    get() = this.toString()
    set(value) {
        println(value)
        field = value
    }
```

Copier

# Délégation de propriétés en Kotlin

Parfois nos méthodes getters et setters contiennent le même code. Pour éviter la duplication du code, ou simplement encapsuler la logique de ces fonctions, vous pouvez utiliser la délégation de propriétés :

Copier

```
import kotlin.reflect.KProperty

class Example {
    val someName by NameDelegate()
    val otherName by NameDelegate()
}

class NameDelegate {
    operator fun getValue(thisRef: Any?, property: KProperty<*>): String {
        return property.name
    }
}

fun main() {
    val example = Example()
    println(example.someName)
    println(example.otherName)
```

Que va afficher le code ci-dessus ?
Est-il possible d'affecter une valeur à l'attribut `someName`, et pourquoi ?

# Délégation de propriétés en Kotlin

Autre exemple de délégation :

Copier

```kotlin
import kotlin.reflect.KProperty

class Delegate {
    operator fun getValue(thisRef: Any?, property: KProperty<*>): String {
        return "$thisRef, thank you for delegating '${property.name}' to me!"
    }

    operator fun setValue(thisRef: Any?, property: KProperty<*>, value: String) {
        println("$value has been assigned to '${property.name}' in $thisRef.")
    }
}
```

Reprenez l'exemple précédent, sans changer le code de la fonction, `main` et remplacer la délégation `NameDelegate` par la délégation `Delegate` ci-dessus. Que constatez vous ?

# Délégation de propriétés en Kotlin

## Delegate standards

Kotlin fournit plusieurs classes de délégations standards :

- "lazy properties" : les valeurs sont calculées lors du premier accès.
- "observable properties" : les listeners sont notifiés lors des changements de la propriété.
- "vetoable properties" : il est possible de mettre un veto sur le changement d'une valeur.
- "notNull properties" : permet de vérifier que la valeur est définie avant un premier accès.

# Délégation de propriétés en Kotlin

## Delegate `lazy`

Exemple de mise en œuvre de la délégation `lazy` :

```kotlin
val myVar: String by lazy {
    println("Lazy init")
    "Hello"
}
println("myVar is not initialized yet")
println("$myVar My dear friend")
```

Copier

De manière plus classique/concise, nous utiliserons :

```kotlin
val myString by lazy { "Some Value" }
```

Copier

Que va afficher le code ci-dessus ?

# Délégation de propriétés en Kotlin

## Delegate `observable`

Comme toutes les délégations standards, la classe de délégation `observable` se trouve dans la classe `Delegates`. Elle prend en paramètre une valeur initiale et une lambda qui sera exécutée à chaque fois que la valeur du champ sera modifiée, sa signature est la suivante :

```
inline fun <T> observable(
    initialValue: T,
    crossinline onChange: (property: KProperty<*>, oldValue: T, newValue: T) -> Unit
): ReadWriteProperty<Any?, T>
```

Copier

# Délégation de propriétés en Kotlin

## Exemple d'utilisation

```kotlin
var updated = false
var max: Int by Delegates.observable(0) { property, oldValue, newValue ->
    updated = true
}

println(max) // 0
println("updated is ${updated}")

max = 10
println(max) // 10
println("updated is ${updated}")
```

Copier

Que va afficher le code ci-dessus ?

# Délégation de propriétés en Kotlin

## Delegate `vetoable`

La syntaxe est pratiquement la même que `observable`, sauf que la lambda, doit retourner un `Boolean`, qui indique si la valeur doit être modifiée, ou non.
Cette délégation est parfaite pour garantir qu'une valeur est comprise dans un interval cohérent, ou pour implémenter un framework de validation simplement.

```
var age: Int by vetoable(initialValue = 0) { property, oldValue, newValue ->
  newValue > 0
}
```

Copier

# Délégation de propriétés en Kotlin

## Delegate `notNull`

C'est le plus simple des délégations standards. Il fonctionne comme `lateinit`, dans le sens où il lève une `IllegalStateException` si la variable est accédé avant d'être initialisée.

```
1 var age by notNull<Int>()
2 fun main() = println(age)
```

Copier

# Generics

- Generics en Kotlin.

# Generics en Kotlin

Tout comme en Java, en Kotlin les classes peuvent avoir des paramètres typés :

```
class Box<T>(t: T) {
    var value = t
}
```

Copier

En général pour créer une instance de ce genre de classe, nous devons fournir le type de l'argument :

```
val box: Box<Int> = Box<Int>(1)
```

Copier

Mais si le type du paramètre peut être inféré, par exemple depuis le type des paramètres du constructeur, ou par un autre moyen, alors il est possible d'omettre le type des arguments :

```
val box = Box(1) // 1 est de type Int, donc le compilateur peut en déduire que nous utilisons un type: Box<Int>
```

Copier

# Generics en Kotlin

Les types peuvent être génériques, mais les fonctions aussi, dans ce cas-là, le type générique est noté avant le nom de la fonction :

```
fun <T> singletonList(item: T): List<T> {
    // ...
}

fun <T> T.basicToString(): String { // extension function
    // ...
}
```

Copier

Pour appeler les méthodes génériques, il faut préciser le type après le nom de la méthode :

```
val l = singletonList<Int>(1)
```

Copier

Mais si le type du paramètre peut être inféré, alors il est possible d'omettre le type :

```
val l = singletonList(1)
```

Copier

# Autres fonctionnalités

- Casting de types en Kotlin.
- Gestion des exceptions.
- Déclaration de constantes.
- Initialisation tardive.

# Casting de types en Kotlin

Il est possible de tester le type d'une variable :

```
fun foo(x: Any) {
    if (x is Person) {
        println("${x.name}") // This wouldn't compile outside the if
    }
}
```

Copier

Notez que nous n'avons pas besoin de caster l'objet x, pour quelle raison, à votre avis ?

Pour changer le type d'une variable, il est possible de le faire explicitement :

```
val p = x as Person
```

Copier

Si l'objet n'est pas vraiment de type `Person` (ou d'une sous classe), alors l'exception `ClassCastException` sera levée.
Si vous n'êtes pas sûr du type, mais que vous pouvez vous satisfaire d'un null, si l'instance n'est pas de type `Person`, vous pouvez utiliser `as?`. Notez que le type de retour sera `Person?` :

```
val p = x as? Person
```

Copier

# Gestion des exceptions

## Les classes d'exception

Toutes les classes d'exceptions sont des descendantes de la classe `Throwable`. Toutes les exceptions ont un message, une pile d'exécution (stack trace), et une cause optionnelle.
Pour lever une exception, il faut utiliser l'expression `throw` :

```
1 throw Exception("Hi There!")
```

Copier

Pour attraper une exception, il faut utiliser l'expression `try` :

```
1 try {
2     // some code
3 }
4 catch (e: SomeException) {
5     // handler
6 }
7 finally {
8     // optional finally block
9 }
```

Copier

Il peut y avoir 0 ou plus blocs de `catch`, le block `finally` est optionnel. Toutefois, il doit y avoir au moins un block catch ou un block finally.

# Gestion des exceptions

## Try est une expression

Try est une expression, et en tant que telle, elle doit avoir une valeur de retour.

```
1 val numValue: Int? = try { parseInt(input) } catch (e: NumberFormatException) { null }
```

Copier

La valeur retournée par l'expression est, soit la dernière expression du block try, ou la dernière expression du bloc catch. Le contenu du block finally ne modifie pas le résultat de l'expression.

# Déclaration de constantes

Une propriété qui est connue au moment de la compilation est annotée avec le mot clé `const`.
Elle peut être utilisée dans les annotations :

```
1 const val SUBSYSTEM_DEPRECATED: String = "This subsystem is deprecated"
2
3 @Deprecated(SUBSYSTEM_DEPRECATED) fun foo() { ... }
```

Copier

# Initialisation tardive

Les valeurs déclarées comme n'acceptant pas de valeur nulle, doivent être initialisées dans le constructeur, toutefois, cela est parfois peu pratique. Par exemple des propriétés qui peuvent être initialisés via une injection de dépendance, ou dans une méthode `setup` dans un test unitaire. Nous ne voulons toutefois pas que la variable puisse avoir une valeur nulle, pour ce faire, nous pouvons utiliser le modifieur `lateinit` :

Copier

```
public class MyTest {
    lateinit var subject: TestSubject

    @SetUp fun setup() {
        subject = TestSubject()
    }

    @Test fun test() {
        subject.method()  // dereference directly
    }
}
```

# Programmation asynchrone

- Le problème de la programmation asynchrone.
- Coroutines en Kotlin et l'implémentation des coroutines.

# Le problème de la programmation asynchrone

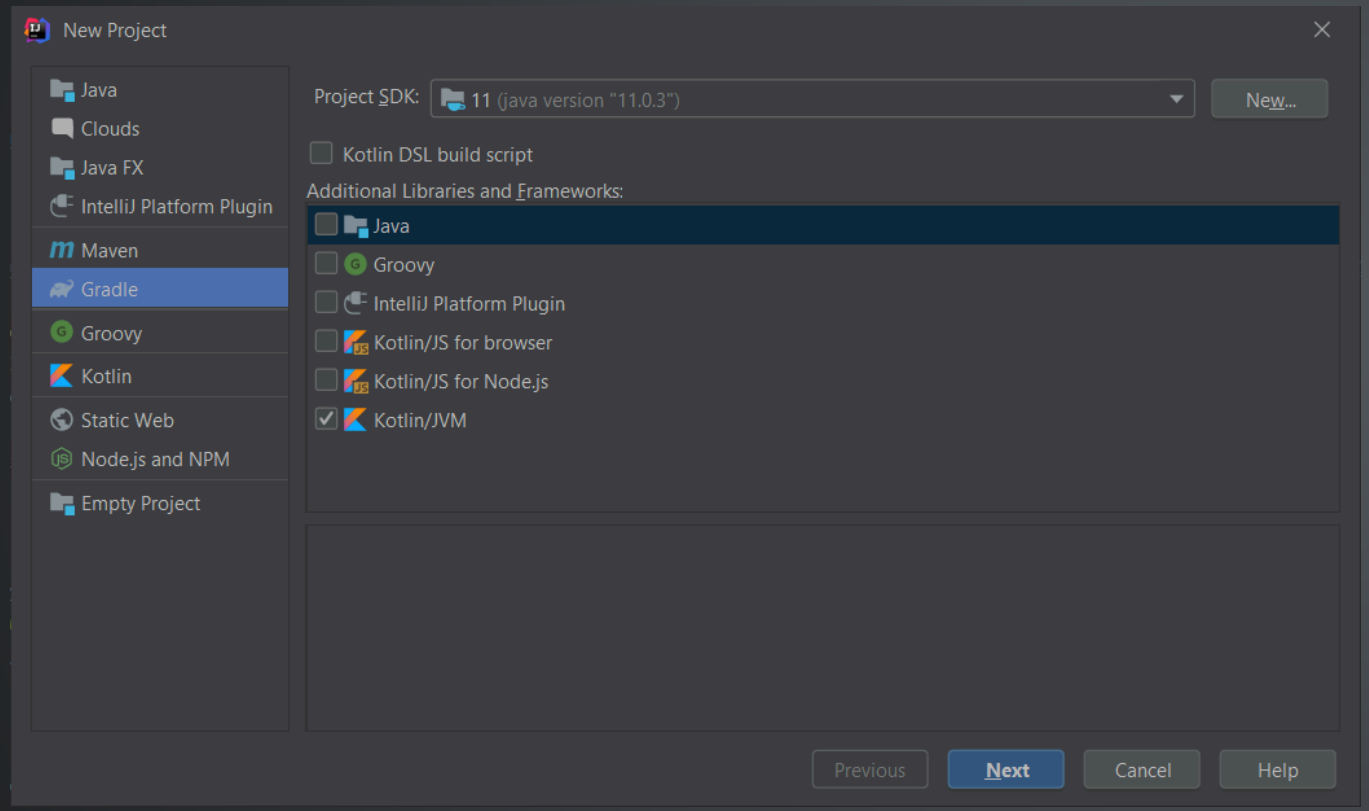
Depuis des décennies, les développeurs sont confrontés à un problème à résoudre : comment faire pour que les applications ne se bloquent pas. Que l'on développe pour un ordinateur, un mobile ou un serveur, nous voulons éviter que l'utilisateur attende, ou encore pire, des goulots d'étranglements qui empêcheraient à l'application de passer à l'échelle.

Plusieurs approches sont possibles pour résoudre ce problème :

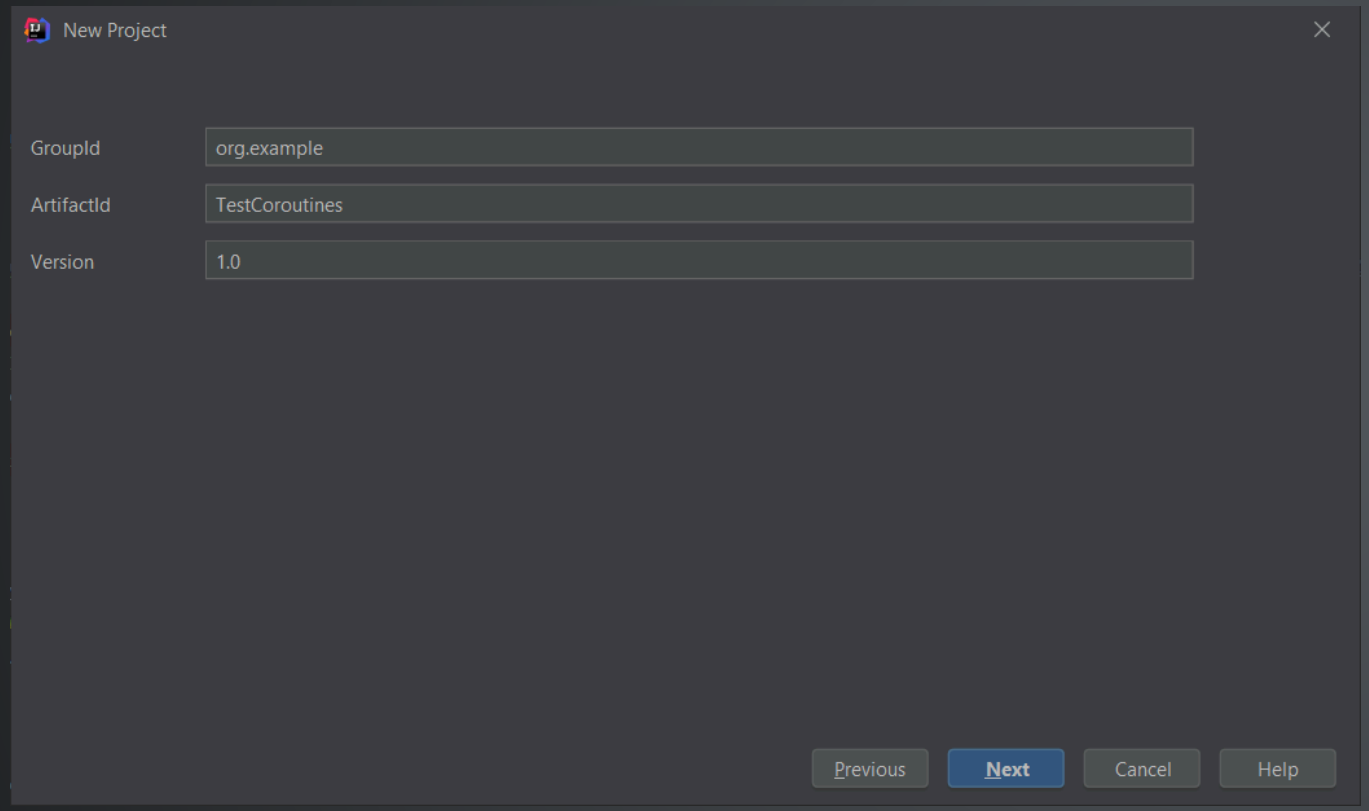
- Traitements en parallèle (Threading).
- Les "callbacks".
- Futures, Promises et al.
- Les extensions réactives.
- Coroutines.

# Mise en place du projet

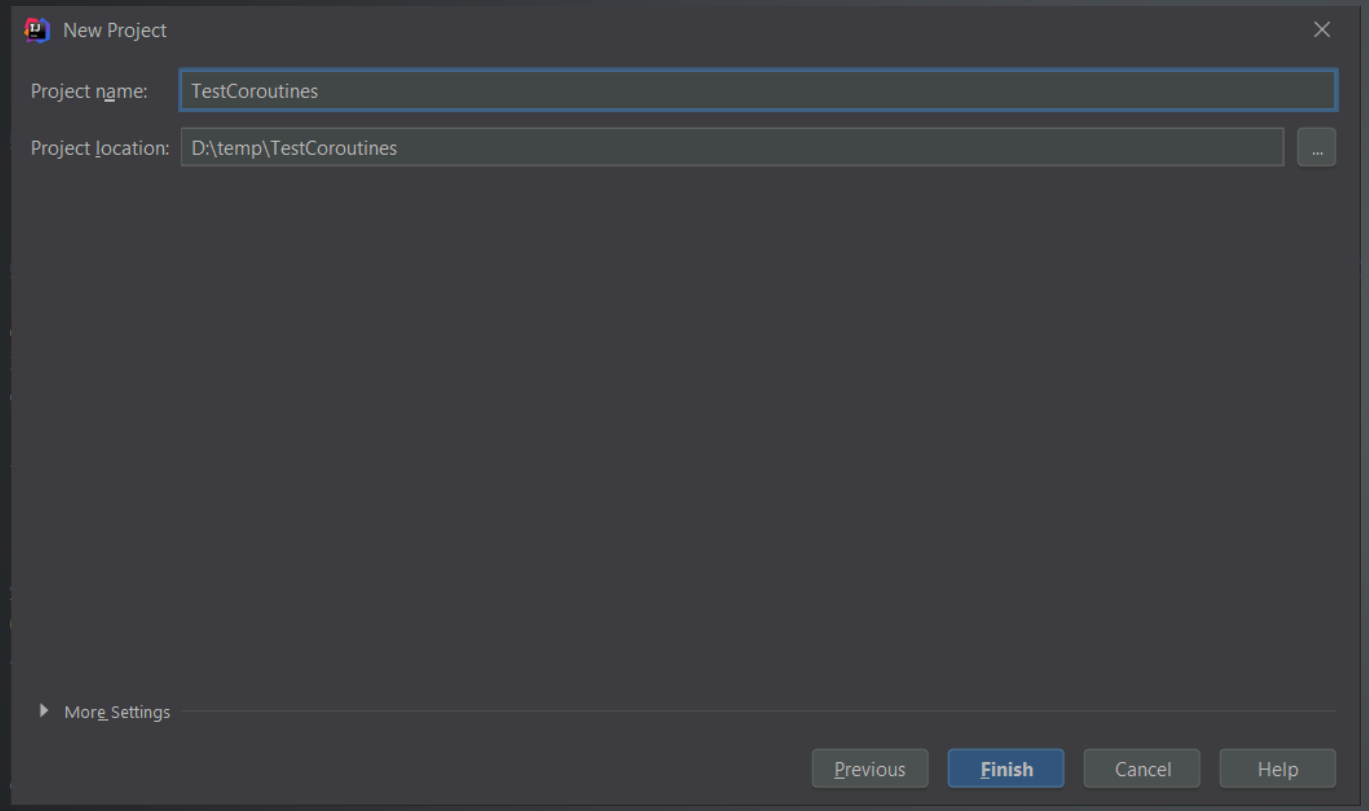
Créez un nouveau projet : **File / New / Project**. Sélectionner **Gradle / Kotlin / JVM**.



# Nommage du module

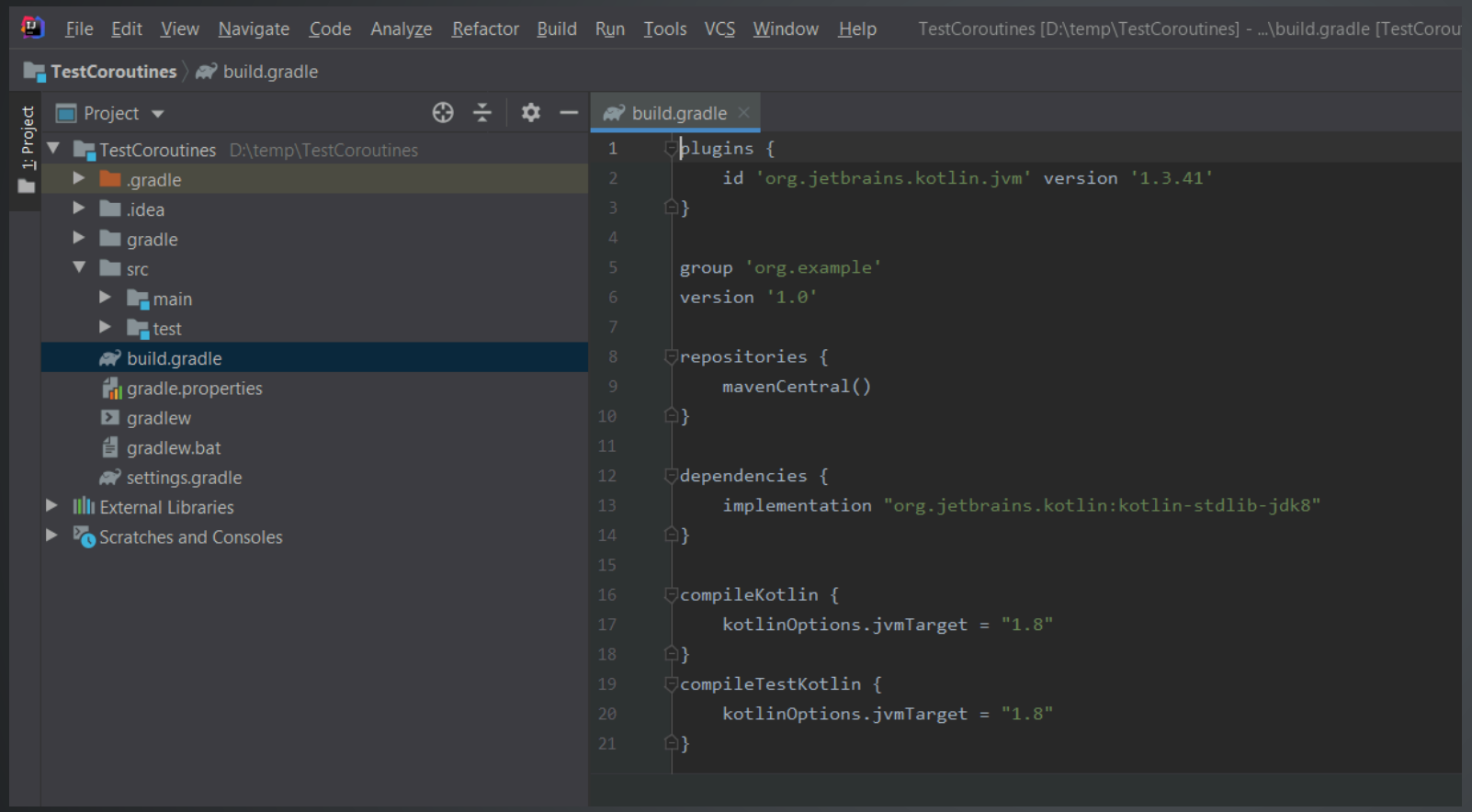


# Nommage du projet



# Coroutines en Kotlin

Ouvrez le fichier `build.gradle`



# Coroutines en Kotlin

## Ajoutez la dépendance

```
dependencies {
    ...
    implementation("org.jetbrains.kotlinx:kotlinx-coroutines-core:1.8.1")
}
```

Copier

# Coroutines en Kotlin

Une coroutine pourrait être vue comme un thread léger, car une coroutine peut être lancée en parallèle, s'attendre les unes les autres et communiquer ensembles. Mais la grosse différence est le coût de celles-ci : pratiquement rien. Il est possible d'en créer des centaines, sans impacter les performances, contrairement aux threads classiques.

Pour lancer une coroutine, le point de départ est la fonction `launch {}` :

```
launch {
//    ...
}
```

Copier

Les coroutines, utilisent un pool de threads pour fonctionner, mais un thread peut faire tourner plusieurs coroutines, donc il n'est pas nécessaire d'avoir beaucoup de threads lancés.

# Coroutines en Kotlin

Lançons notre première coroutine :

```
println("Start")

// Start a coroutine
GlobalScope.launch {
    delay(1000)
    println("Hello")
}

Thread.sleep(2000) // wait for 2 seconds
println("Stop")
```

Copier

La fonction `delay()` fonctionne comme la fonctions `Thread.sleep()`, mais elle ne bloque pas le thread, elle suspend simplement la coroutine. Le thread retourne dans le pool de thread. Et la coroutine, reprendra son fonctionnement sur un thread disponible.

Q1 - Que se passe-t-il si l'on supprime la ligne `Thread.sleep(2000)` ?

Q2 - Que se passe-t-il si l'on remplace `Thread.sleep(2000)` par `delay(2000)` ?

# Coroutines en Kotlin

## Lint

Nous avons une note qui nous informe que nous n'avons pas le droit d'utiliser les coroutines de la sorte. Pour retirer le "warning" nous pouvons ajouter `@OptIn(DelicateCoroutinesApi::class)` :

```
1 @OptIn(DelicateCoroutinesApi::class)
2 fun main() {
3 ...
```

Copier

## Blocage du thread principal

Pour pouvoir utiliser la fonction `delay(...)`, nous allons l'appeler dans une fonction `runBlocking {}` :

```
runBlocking {
    delay(2000)
}
```

Copier

Ce qui nous donnera au final :

```
import kotlinx.coroutines.GlobalScope
import kotlinx.coroutines.delay
import kotlinx.coroutines.launch
import kotlinx.coroutines.runBlocking

fun main() {
    println("Start")

// Start a coroutine
    GlobalScope.launch {
        delay(1000)
        println("Hello")
    }

    // Step 1
    //Thread.sleep(2000) // wait for 2 seconds

```

Copier

# Coroutines en Kotlin

## Lançons beaucoup d'opérations

Nous allons lancer 1 000 000 d'opérations
Commençons avec une simple boucle :

```
1 val c = AtomicLong()
2
3     for (i in 1..1_000_000L) {
4         c.addAndGet(i)
5     }
6
7     println(c.get())
```

Copier

Que se passerait-il si l'opération prenait 1 seconde ?
Par exemple en mettant en place une pause avec `Thread.sleep(1000)`
Cela prendrait environ 11 jours, 13 heures, 46 minutes et 40 secondes.

# Coroutines en Kotlin

## Lançons beaucoup de thread

Nous allons comparer les threads et les coroutines, en lançant, disons 1 million de processus en parallèle
Commençons avec les `threads` :

```
val c = AtomicLong()

for (i in 1..1_000_000L)
    thread(start = true) {
        c.addAndGet(i)
    }

println(c.get())
```

Copier

Que remarquez-vous, concernant la charge de la machine ?

# Coroutines en Kotlin

## Lançons beaucoup de thread

Écrivez le même code avec des coroutines :

```
1 val c = AtomicLong()
2
3 for (i in 1..1_000_000L)
4     GlobalScope.launch {
5         c.addAndGet(i)
6     }
7
8 println(c.get())
```

Copier

Que constate-t-on ?
Si l'on ajoute une pause (en Coroutine on utilise `delay(1000)`), que ce passe-t-il ? Et pourquoi ?
Les coroutines n'ont pas terminé, quand la fonction `main` se termine.

# Mise en place

## Étapes de mise en place :

- Android Studio.
- ADB.
- Téléphone en mode développeur.
- Coloration de logcat.
- Mise en place des templates.
- Mise en place de scrcpy.

# Mise en place

## Android Studio

Pour commencer, nous allons vérifier que nous avons bien Android Studio fonctionnel avec la dernière version :

# Mise en place

## ADB

Vérifions que ADB est bien accessible en ligne de commande, cela nous sera utile par la suite.

- Ouvrons une invite de commande (Windows + R, cmd).
- Saisissons : `adb version`.

Si ce n'est pas le cas, ajoutons le au "PATH".

# Mise en place

## Téléphone en mode développeur

Nous vérifions que notre téléphone est en mode développeur :

- En ligne de commande tapons : `adb devices`
- Vérifions dans Android Studio que le device apparaît bien.

Dans le cas contraire, nous vérifions que le téléphone a bien le mode développeur activé.

## Coloration de logcat

Nous allons colorer le logcat afin qu'il soit plus lisible :

- File / Settings
- Cherchons : "logcat"
- Editor / Color Scheme / Android Logcat
- Les couleurs que j'ai définies (à titre d'exemple) :

| | |
|---|---|
| Assert | C00000 |
| Debug | 41BB2E |
| Error | FF6B68 |
| Info | F5F550 |
| Verbose | BBBBBB |
| Warning | F4AC40 |

# Mise en place

Les LiveTemplates, sont des raccourcis qui permettent d'écrire rapidement des bouts de code. Je vous fournis une série de LiveTemplates que nous utiliserons dans la suite de la formation. Procédons comme suit :

- Récupération des templates :
  - Projet GitHub LiveTemplates
  - Cliquons sur le bouton Code
  - Download ZIP
- Les copier dans :
  - Windows: `C:\Users<your_user>\AppData\Roaming\Google\AndroidStudio2022.3\templates` (par ex, selon la version installée)
  - Linux: `~/.AndroidStudio"version"/config/templates`
  - macOS: `~/Library/Preferences/AndroidStudio"version"/templates`
- File/Invalidate caches/Restart...
- Just Restart.
- File/Settings.

# Mise en place

## Utilisation

Pour utiliser les LiveTemplates, il suffit de taper le début de l'abréviation, lancer l'autocomplétion et le LiveTemplate sera exécuté.

- Par exemple tous les lives templates pour compose `and_compose_`.
- Par exemple tous les lives templates pour les tests UIAutomator commencent par `and_uia_`.
- Les autres templates commencent tous par `and_` et sont filtrés par le type de fichier dans lequel on se trouve (XML, Java, Kotlin).

# Mise en place

## Shell scripts

Afin d'automatiser certaines tâches, il est possible de les scripter. Pour cela je vous propose une série de scripts que vous pourrez utiliser après la formation :
Récupération des scripts :

- Projet GitHub AndroidBashScripts
- Cliquons sur le bouton Code
- Download ZIP

Pour les lancer sous Windows il faudra installer Git.

# Mise en place

## scrcpy

Nous allons installer `scrcpy` qui nous permettra de partager l'affichage de notre téléphone.

- Téléchargeons la dernière version sur GitHub.
- Décompressons le fichier ZIP dans un répértoire ( `C:\tools` par exemple) ce qui donnera dans notre cas : `C:\tools\scrcpy-win64-v2.3.1` .
- Lançons l'utilitaire avec `scrcpy` .

Une solution si vous avez installé `winget` est d'utiliser la commande suivante :

```
1 winget install --silent --id=Genymobile.scrcpy  -e
```

Copier

# Réalisation d'une application Android simple en Kotlin

Nous allons commencer par poser les bases d'une application Android en Kotlin.
Nous allons suivre les étapes suivantes :

- Installer et lancer Android Studio.
- Créer un projet : IMC.
- Mettre en place le view binding.
- Définir notre interface graphique.
- Implémenter le code métier.

# Réalisation d'une application Android simple en Kotlin

## Installer et lancer Android Studio

Il suffit d'avoir une bonne connexion internet, et d'aller sur https://developer.android.com/studio.

# Réalisation d'une application Android simple en Kotlin

Créer un projet : IMC

# Réalisation d'une application Android simple en Kotlin

## Mettre en place le view binding

Pour cela, nous allons suivre les indications de la page : View Binding.

# Réalisation d'une application Android simple en Kotlin

## Modifions le build.gradle (Module :app)

```
android {
    ...
    buildFeatures {
        viewBinding true
    }
}
```

Copier

Si l'on est en kotlin (build.gradle.kt) :

```
android {
    ...
    buildFeatures {
        viewBinding = true
    }
}
```

Copier

## Modifions la méthode `onCreate` de notre Activity

```
override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    binding = ResultProfileBinding.inflate(layoutInflater)
    val view = binding.root
    setContentView(view)
}
```

Copier

# Réalisation d'une application Android simple en Kotlin

## Usage

Nous pouvons maintenant utiliser nos éléments graphiques directement depuis la classe binding, par exemple :

```
1 binding.name.text = viewModel.name
2 binding.button.setOnClickListener { viewModel.userClicked() }
```

Copier

# Réalisation d'une application Android simple en Kotlin

## Définir une première interface graphique

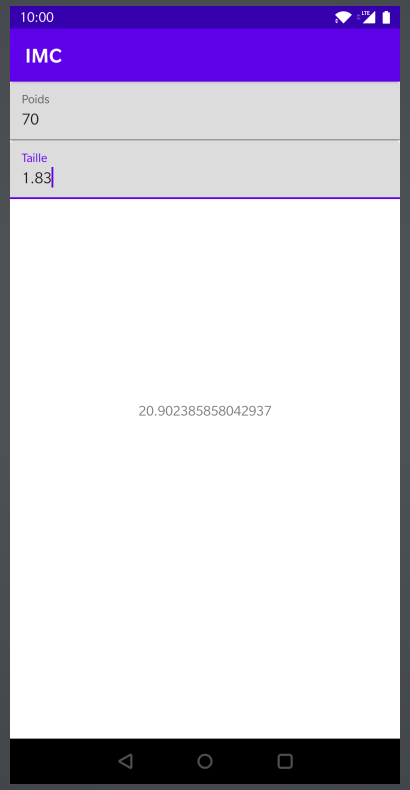
# Réalisation d'une application Android simple en Kotlin

## Implémenter les méthodes de base

- Ajouter un tag aux boutons avec des valeurs différentes.
- Définit une callback qui sera appelée par le `setOnClickListener`.
- Assigner la callback à nos 3 boutons en utilisant `with`.

# L'application IMC

## Définir notre interface graphique



# L'application IMC

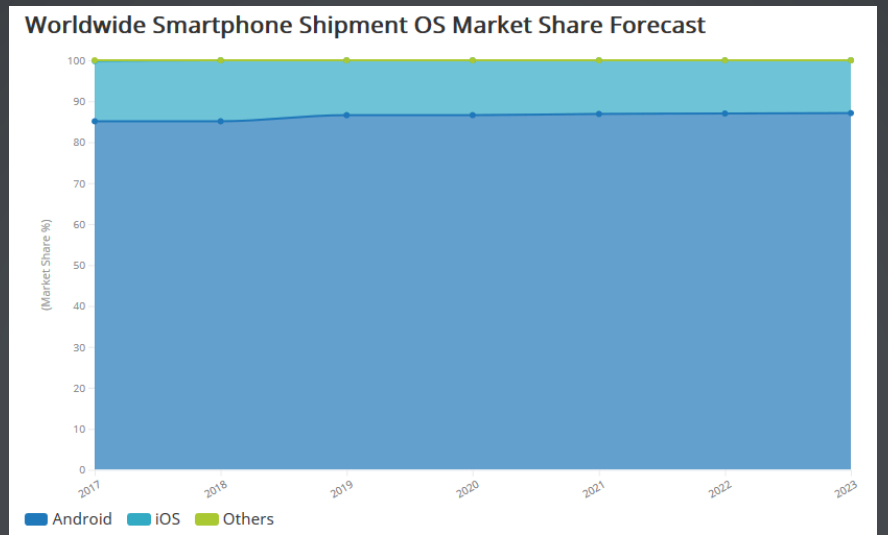
## Implémenter le code métier

- Écouter les changements de valeur des champs textes.
- Calculer et afficher la valeur de l'IMC avec la formule : poids / taille (en m)².
- Afficher des indications supplémentaires.
- Enregistrer la taille.

# La plate-forme Android

- L'architecture Android, Linux. Historiques et fonctionnalités.
- Les terminaux cibles.

# Android dans le monde



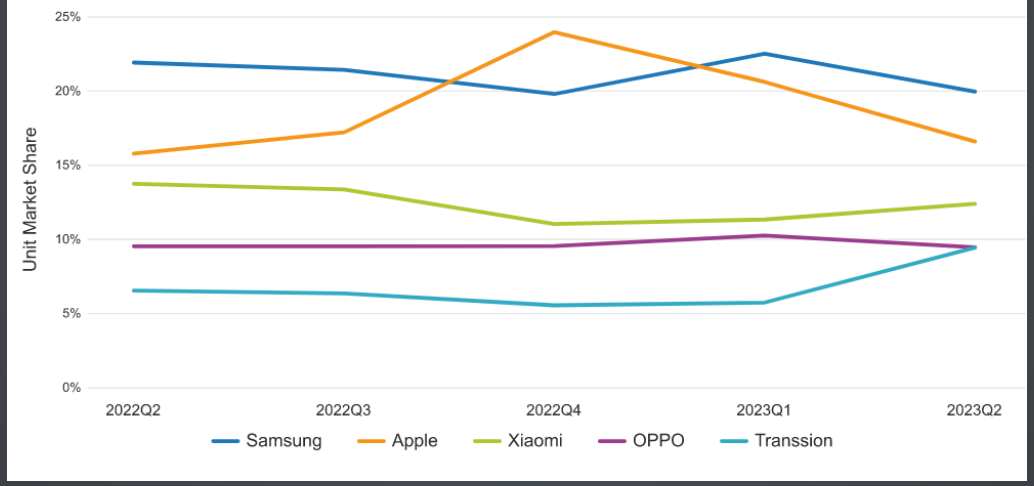

| Year | 2017 | 2018 | 2019 | 2020 | 2021 | 2022 | 2023 |
|---|---|---|---|---|---|---|---|
| Android | 85.1% | 85.1% | 86.6% | 86.6% | 86.9% | 87.0% | 87.1% |
| iOS | 14.7% | 14.9% | 13.4% | 13.4% | 13.1% | 13.0% | 12.9% |
| Others | 0.2% | 0.0% | 0.0% | 0.0% | 0.0% | 0.0% | 0.0% |
| **TOTAL** | **100.0%** | **100.0%** | **100.0%** | **100.0%** | **100.0%** | **100.0%** | **100.0%** |

Source : https://www.idc.com/promo/smartphone-market-share/os

# Android dans le monde

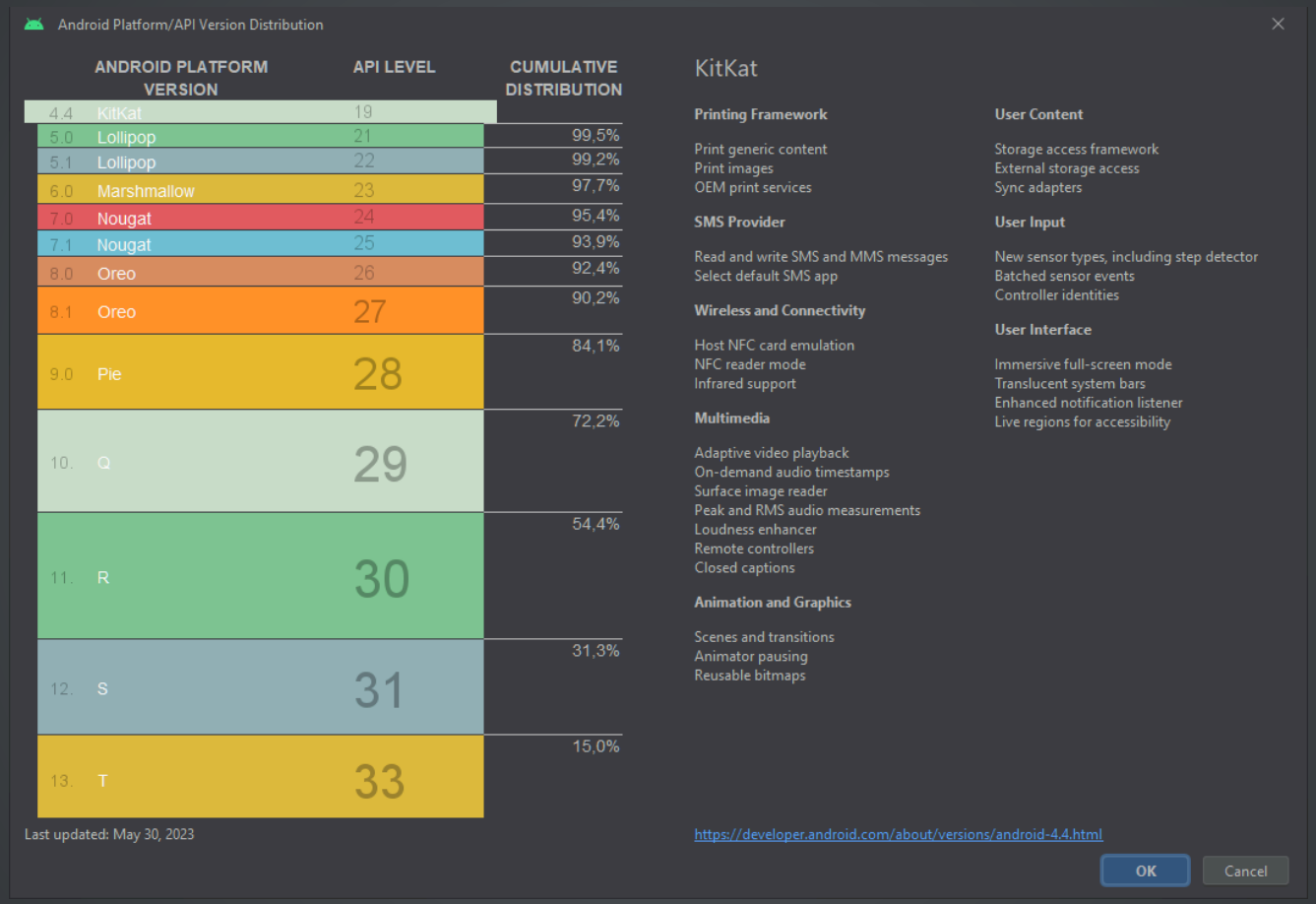
Data table for "Worldwide Top 5 Smartphone Companies, 2023Q2 Unit Market Share" chart

| Company | 2022Q2 | 2022Q3 | 2022Q4 | 2023Q1 | 2023Q2 |
|---|---|---|---|---|---|
| Samsung | 21.9% | 21.4% | 19.8% | 22.5% | 20.0% |
| Apple | 15.8% | 17.2% | 24.0% | 20.6% | 16.6% |
| Xiaomi | 13.8% | 13.4% | 11.0% | 11.4% | 12.4% |
| OPPO | 9.5% | 9.5% | 9.6% | 10.3% | 9.5% |
| Transsion | 6.5% | 6.3% | 5.6% | 5.7% | 9.4% |
| Others | 32.4% | 32.1% | 30.0% | 29.5% | 32.1% |

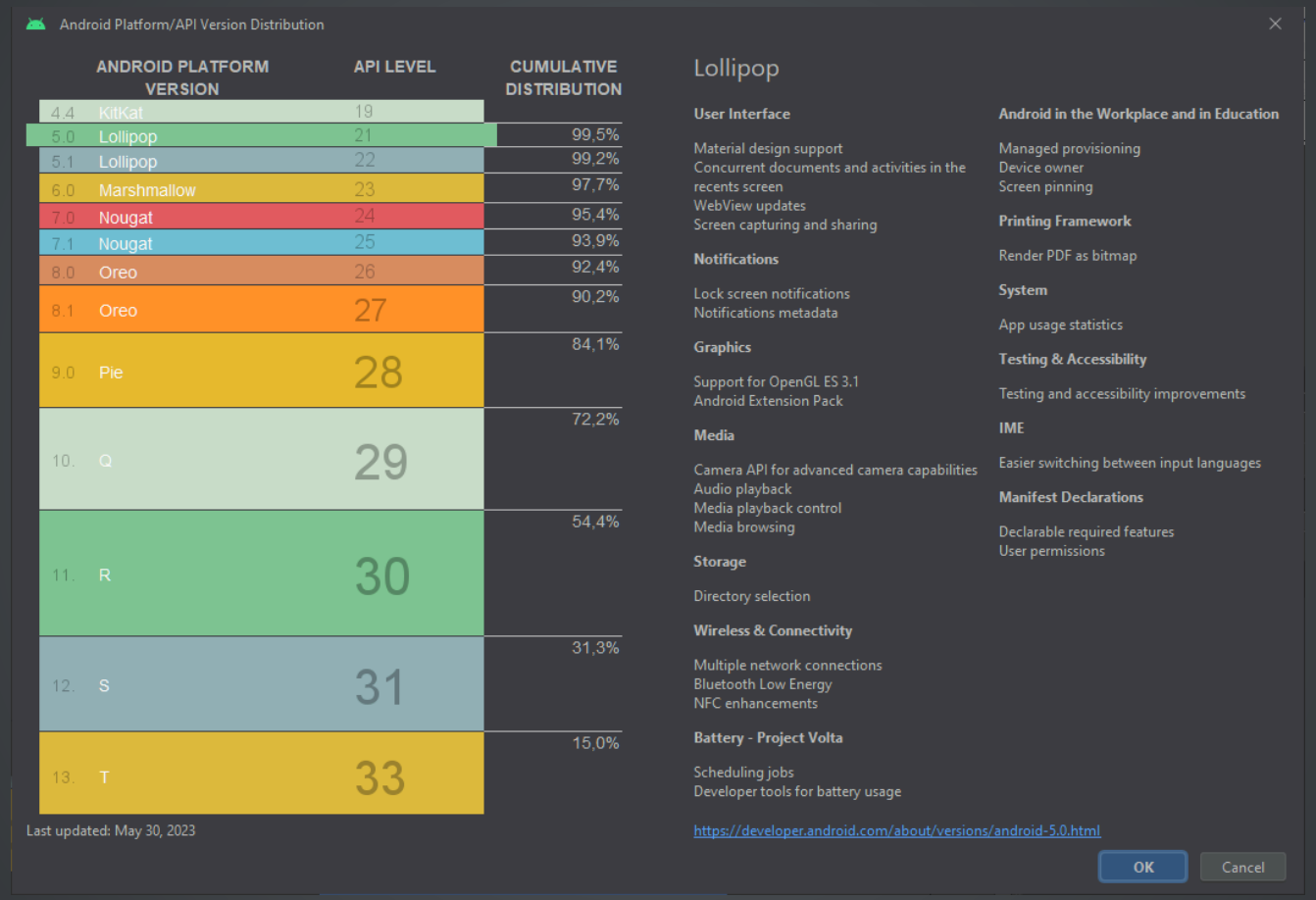



Source : https://www.idc.com/promo/smartphone-market-share/vendor

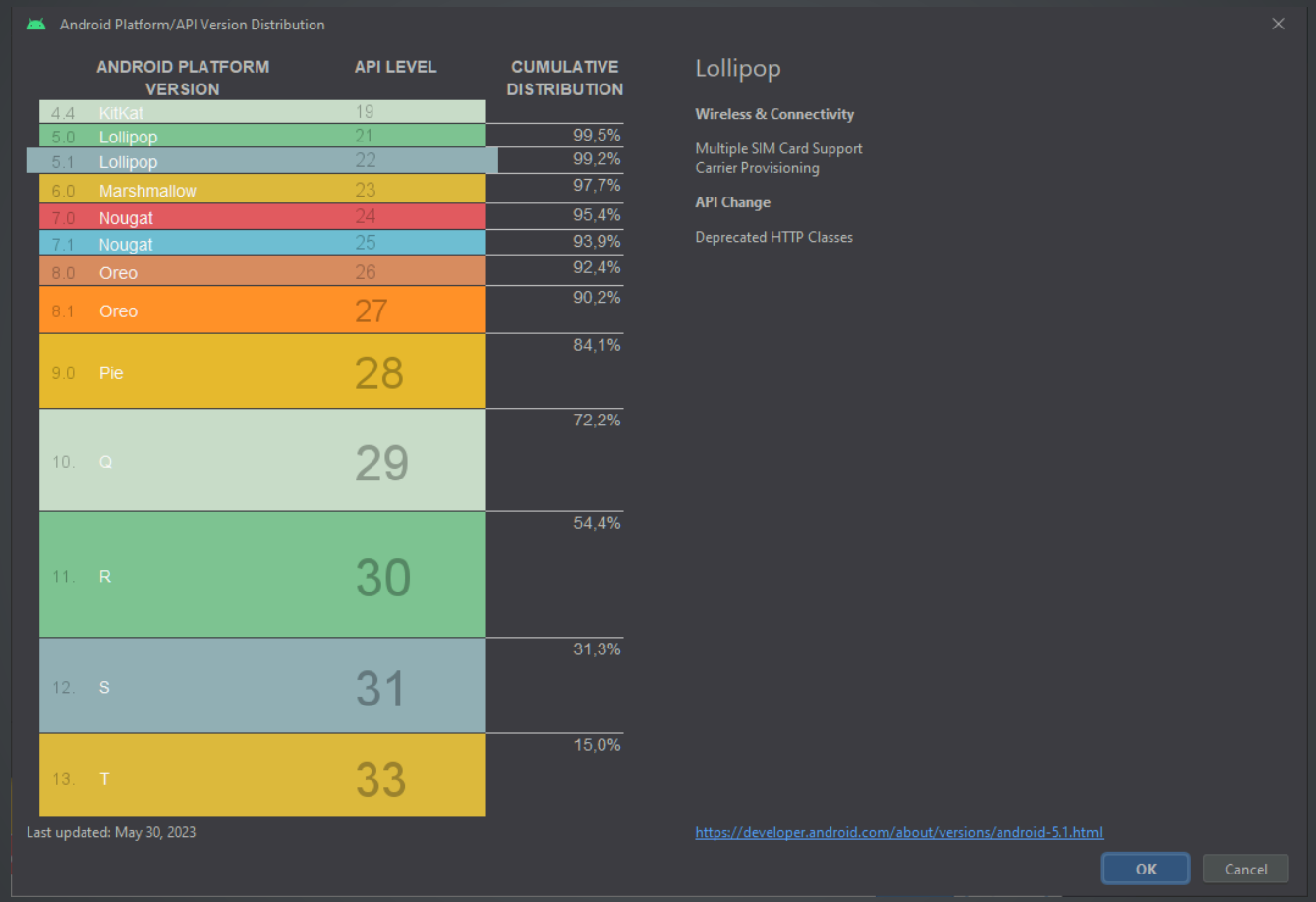
# L'histoire d'Android



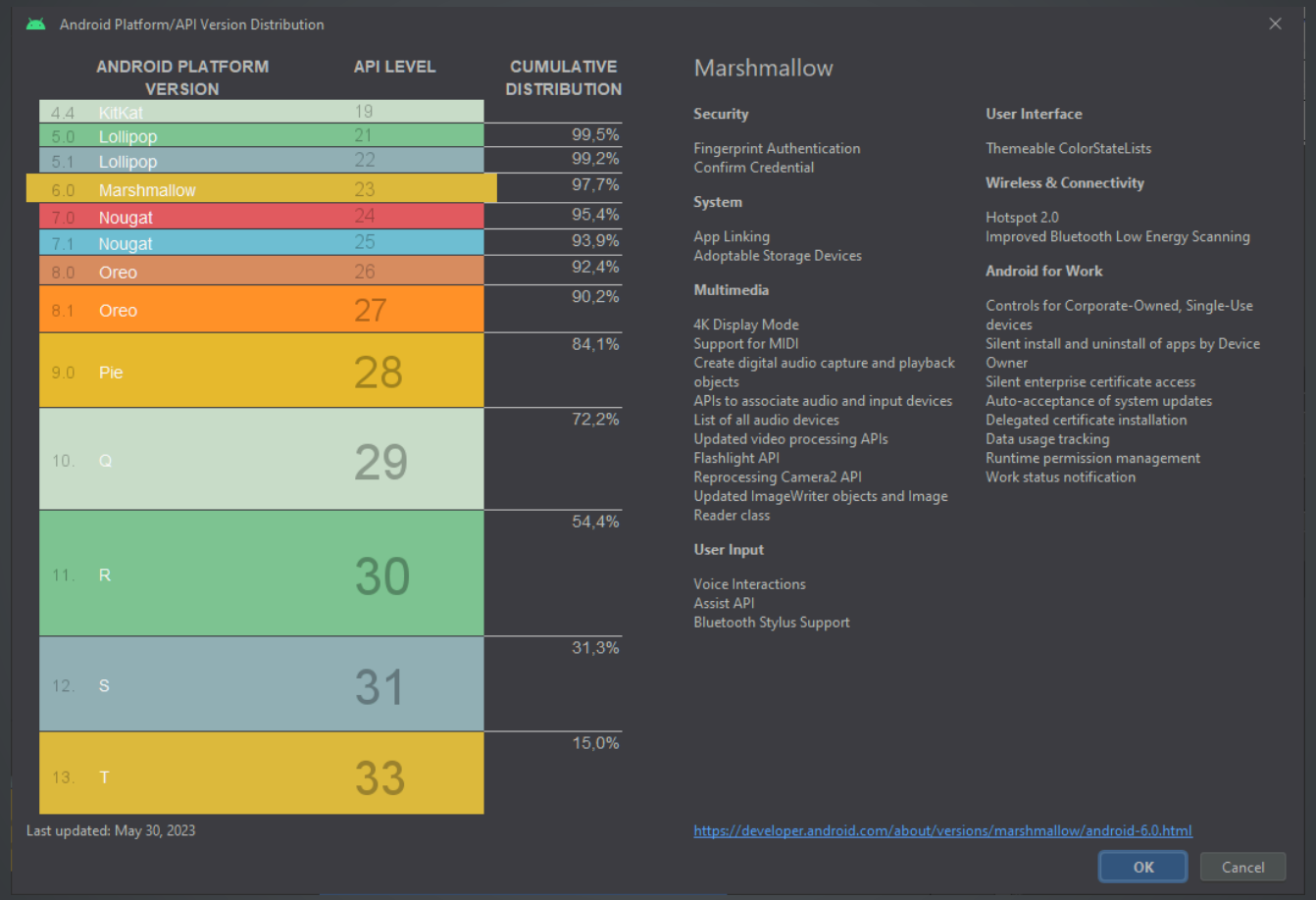
# L'histoire d'Android



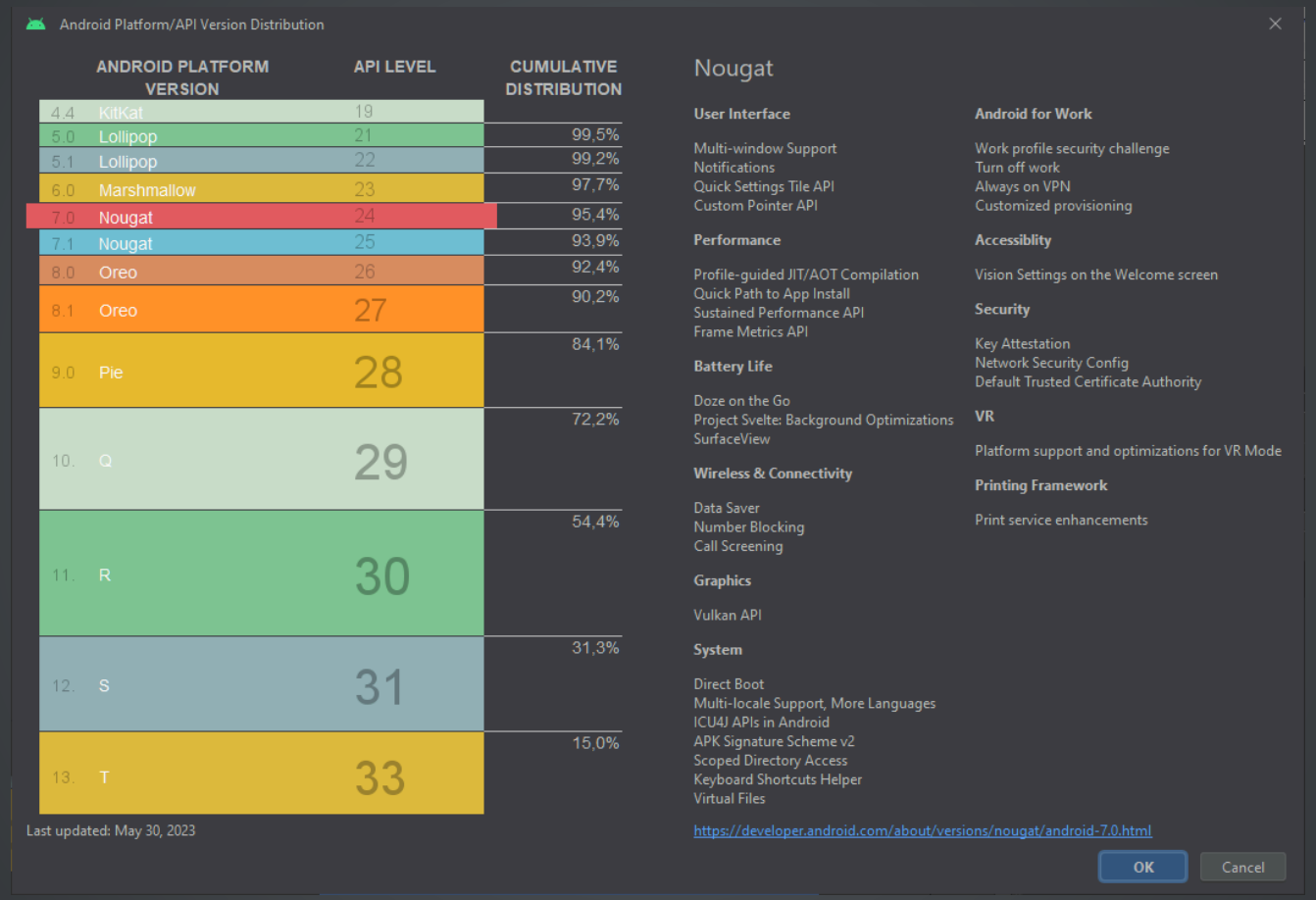
# L'histoire d'Android



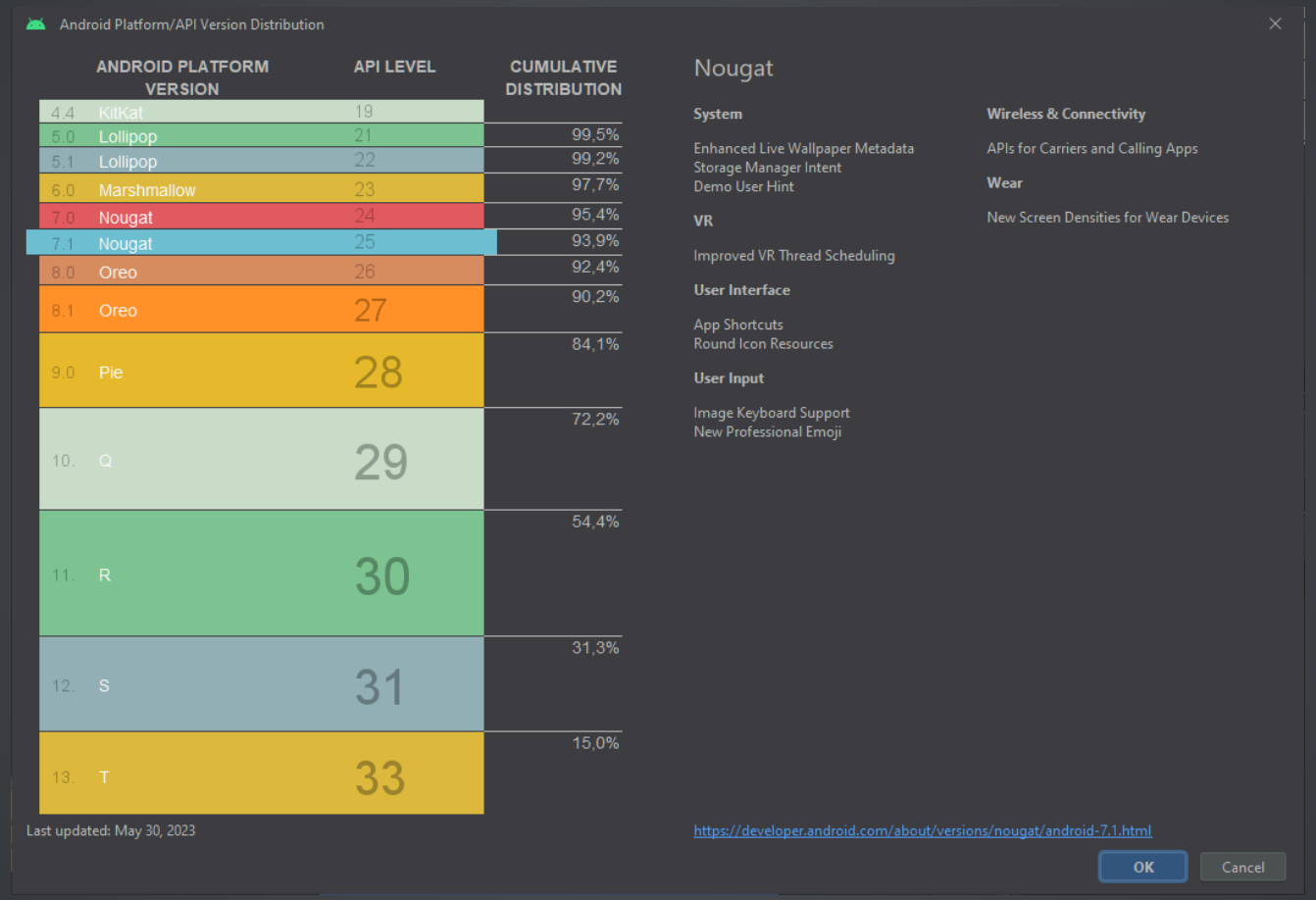
# L'histoire d'Android



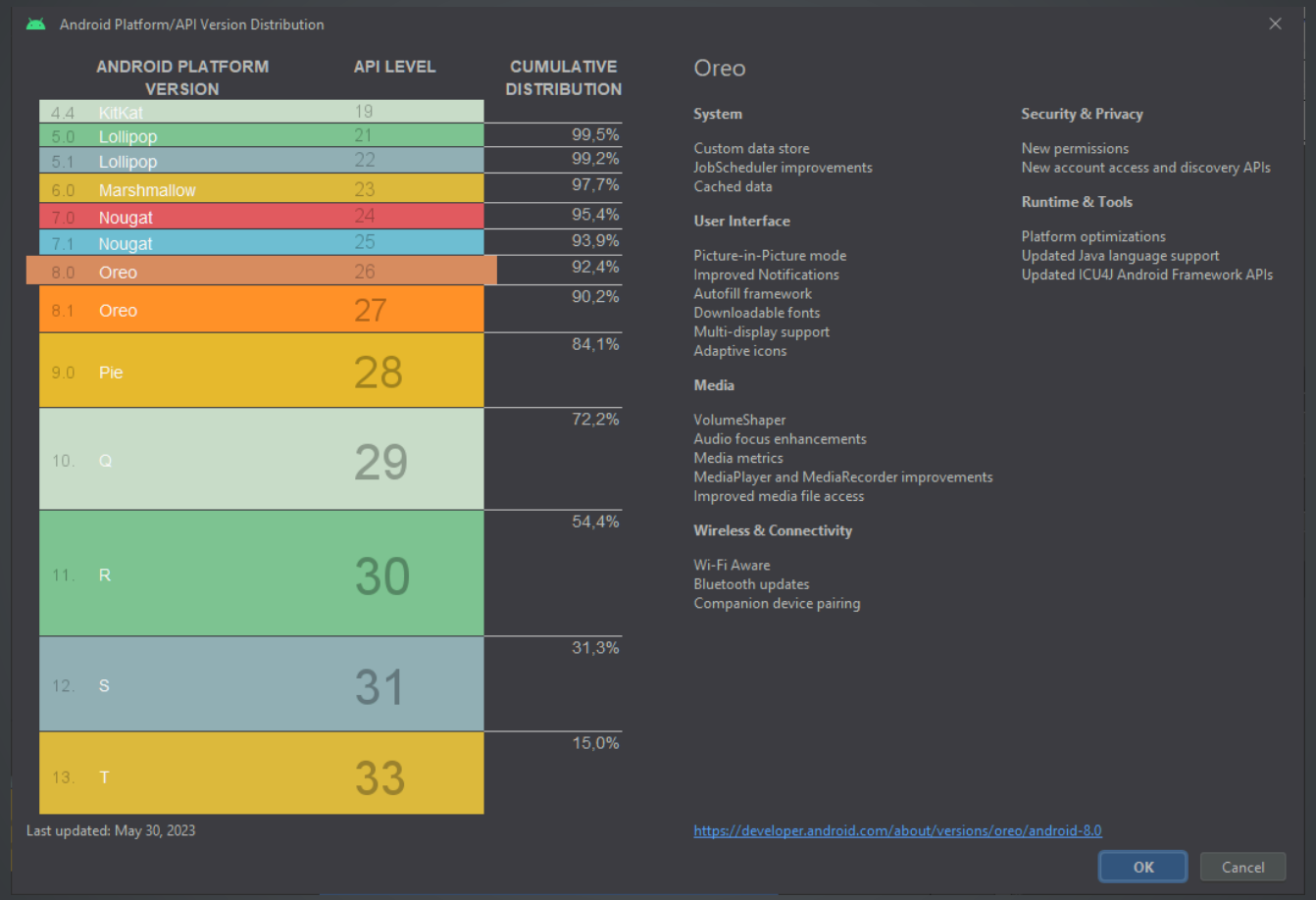
# L'histoire d'Android



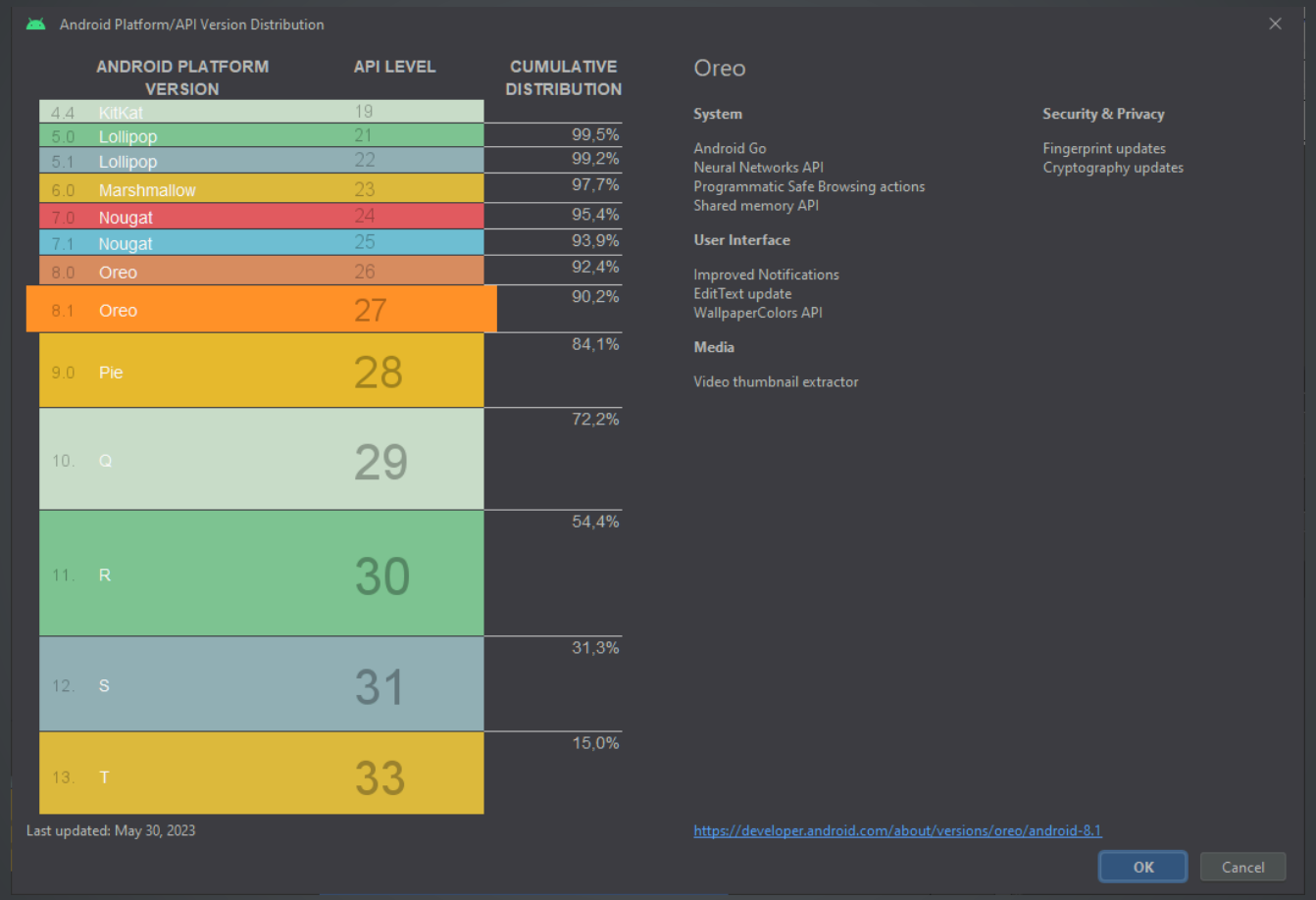
# L'histoire d'Android



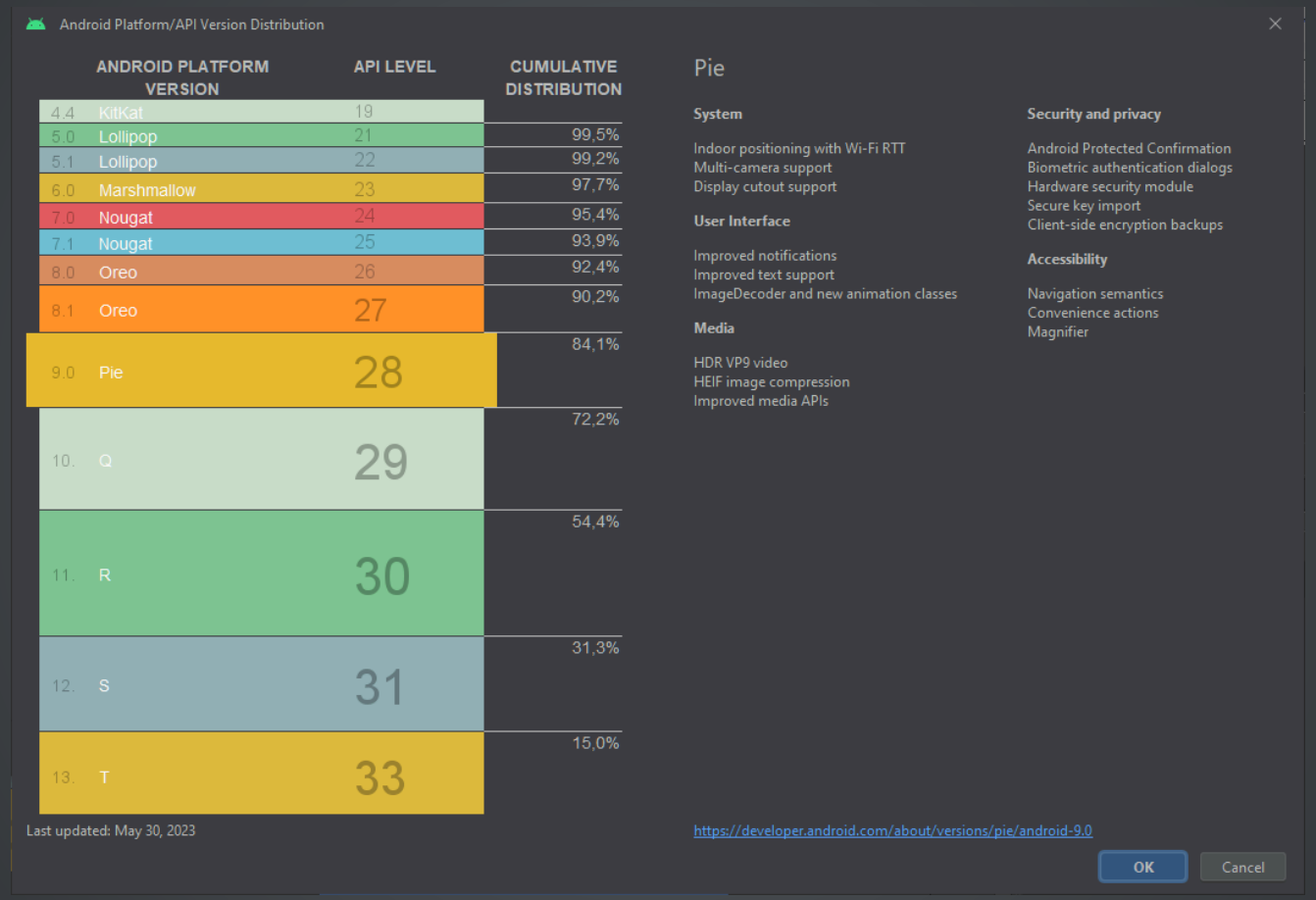
# L'histoire d'Android



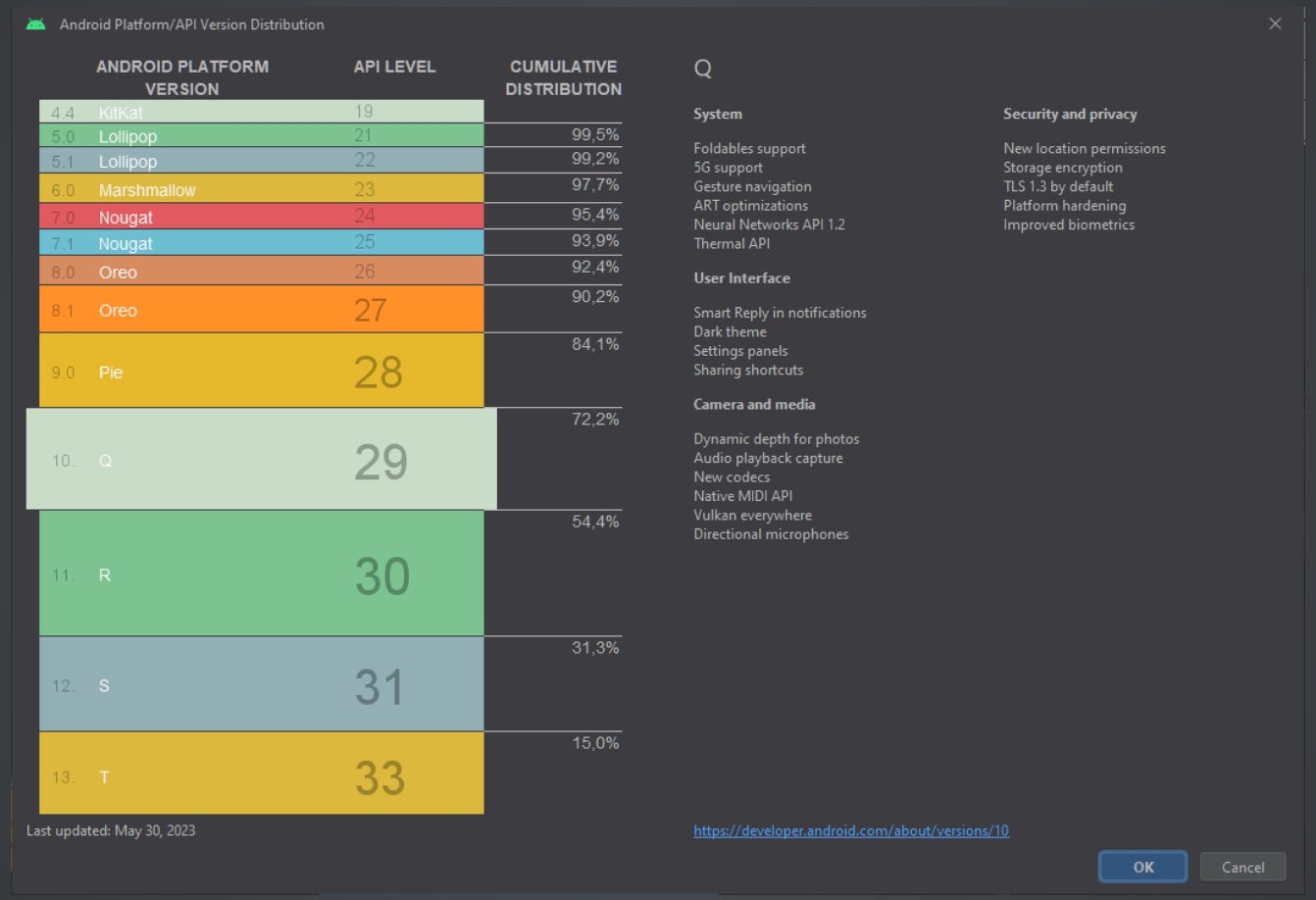
# L'histoire d'Android



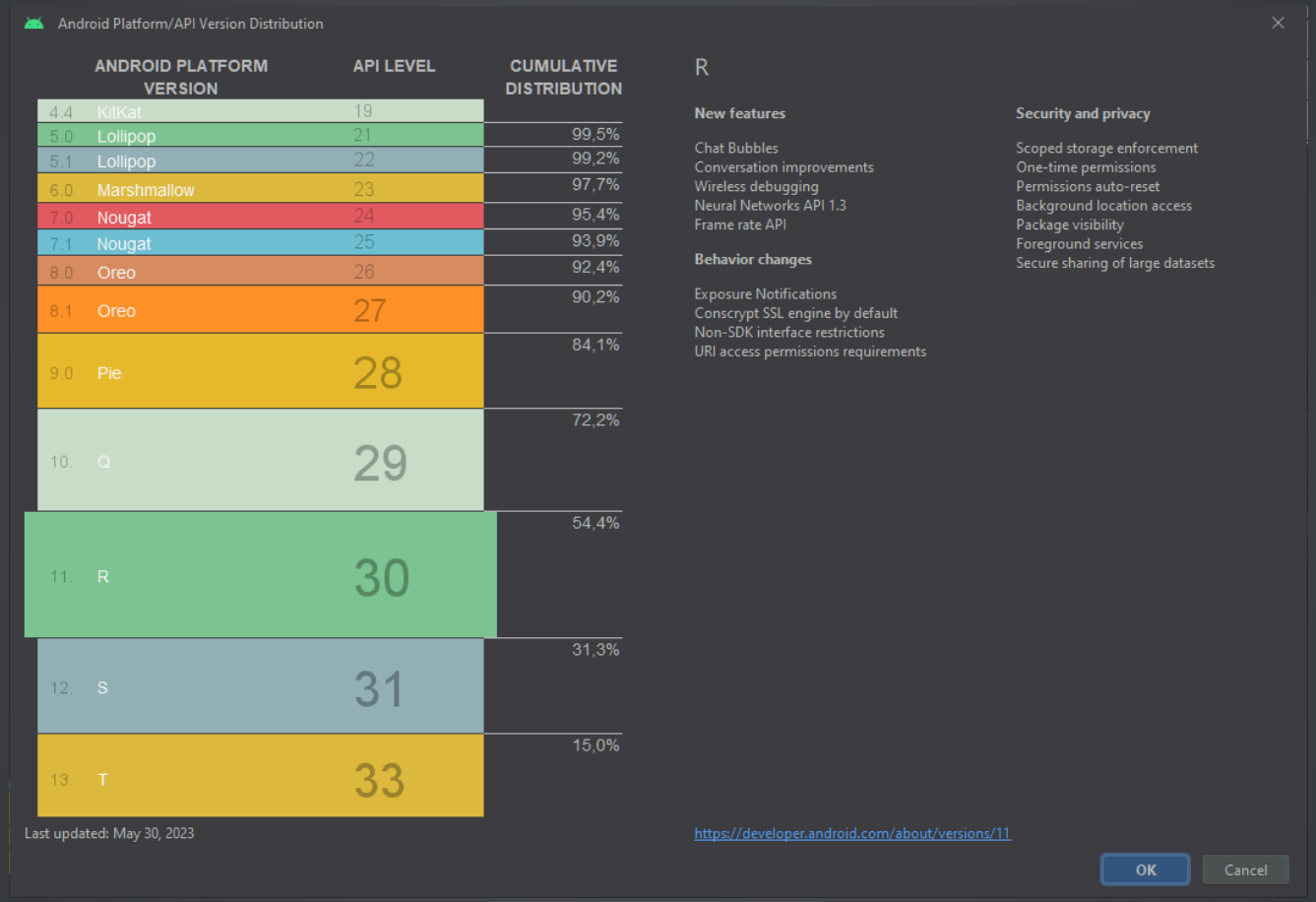
# L'histoire d'Android



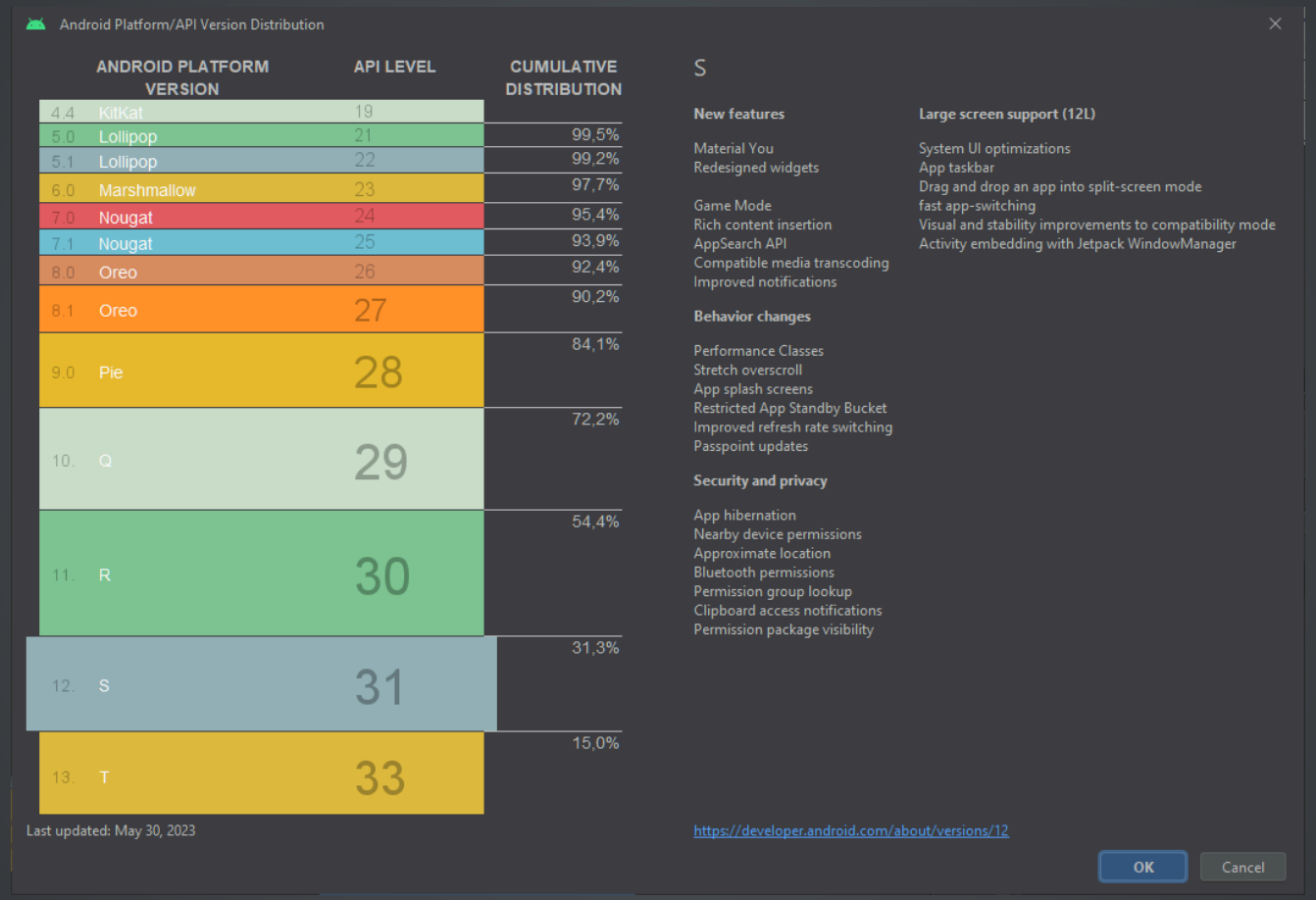
# L'histoire d'Android



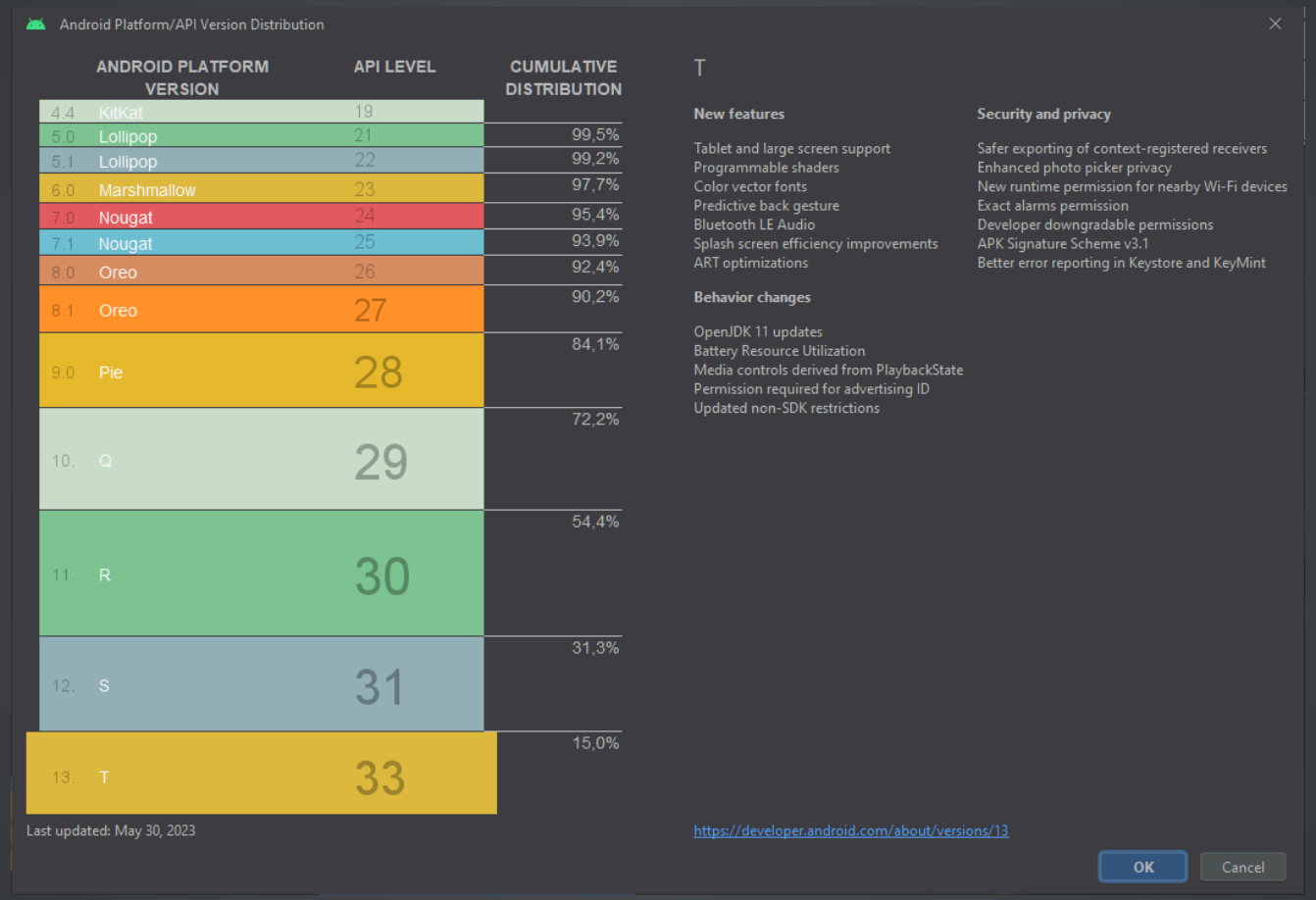
# L'histoire d'Android



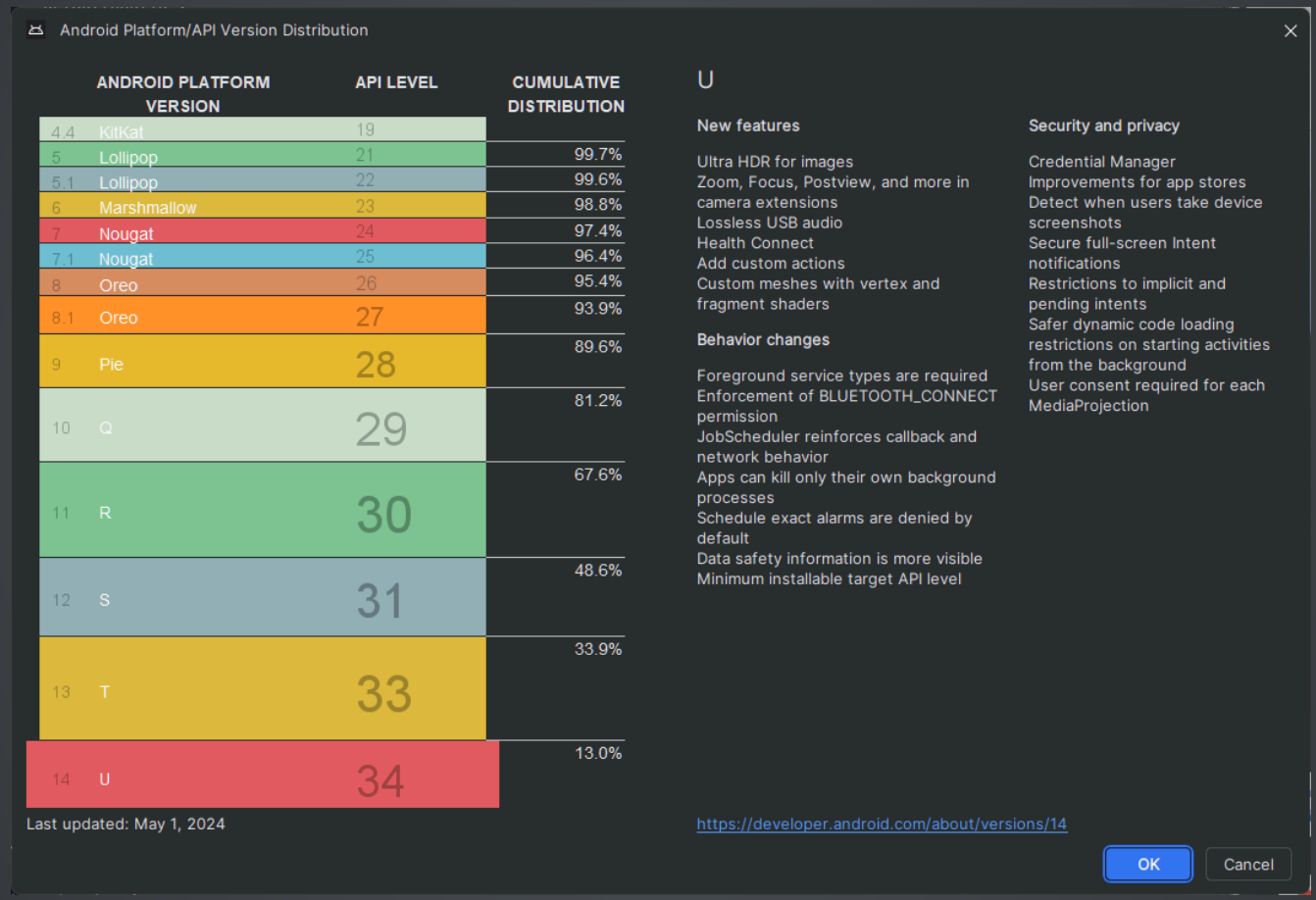
# L'histoire d'Android



# L'histoire d'Android



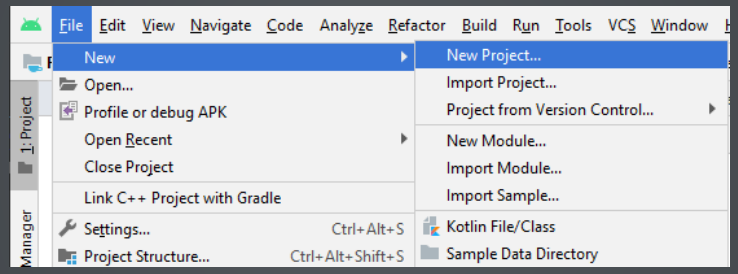
# L'histoire d'Android



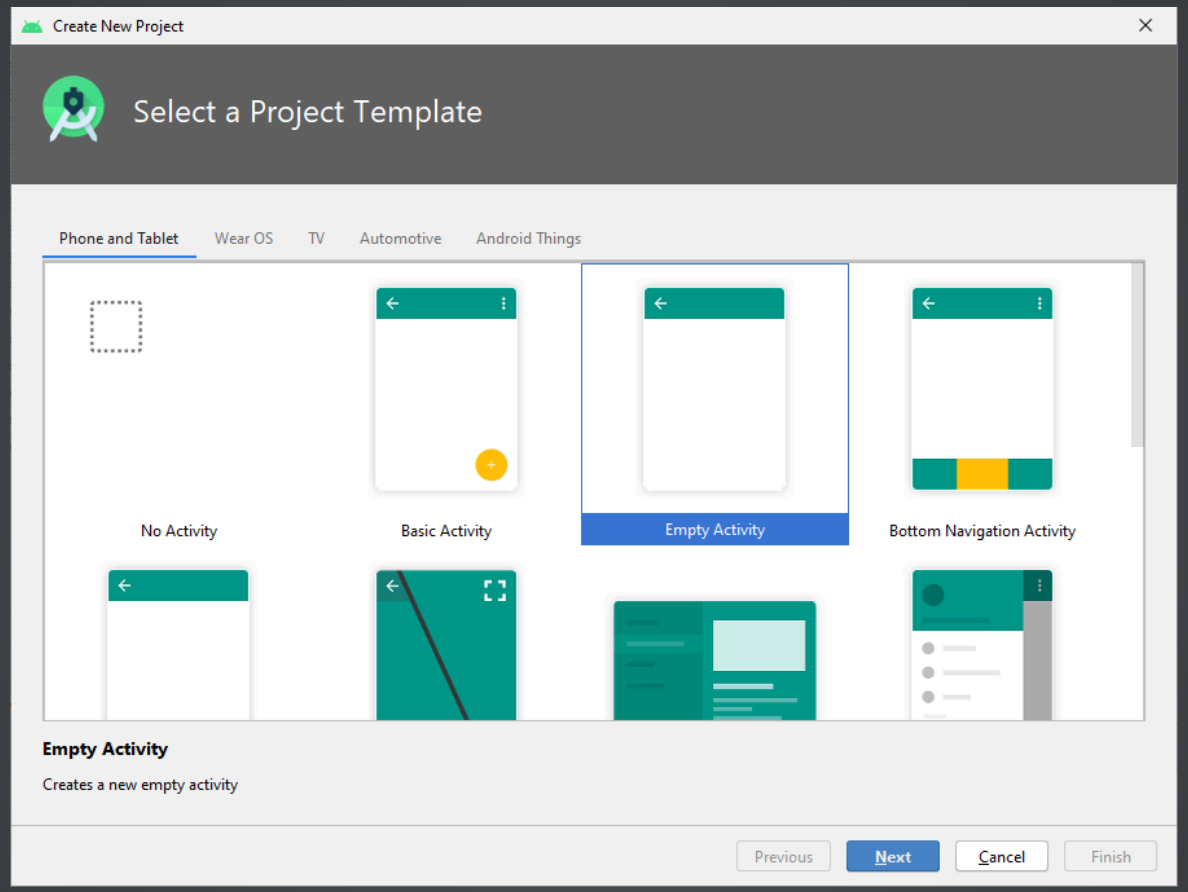
# Les terminaux cibles

Pour obtenir la distribution des versions, vous devez créer un nouveau projet Android, puis sur l'écran "Configure Your Project" cliquons sur "Help me choose".

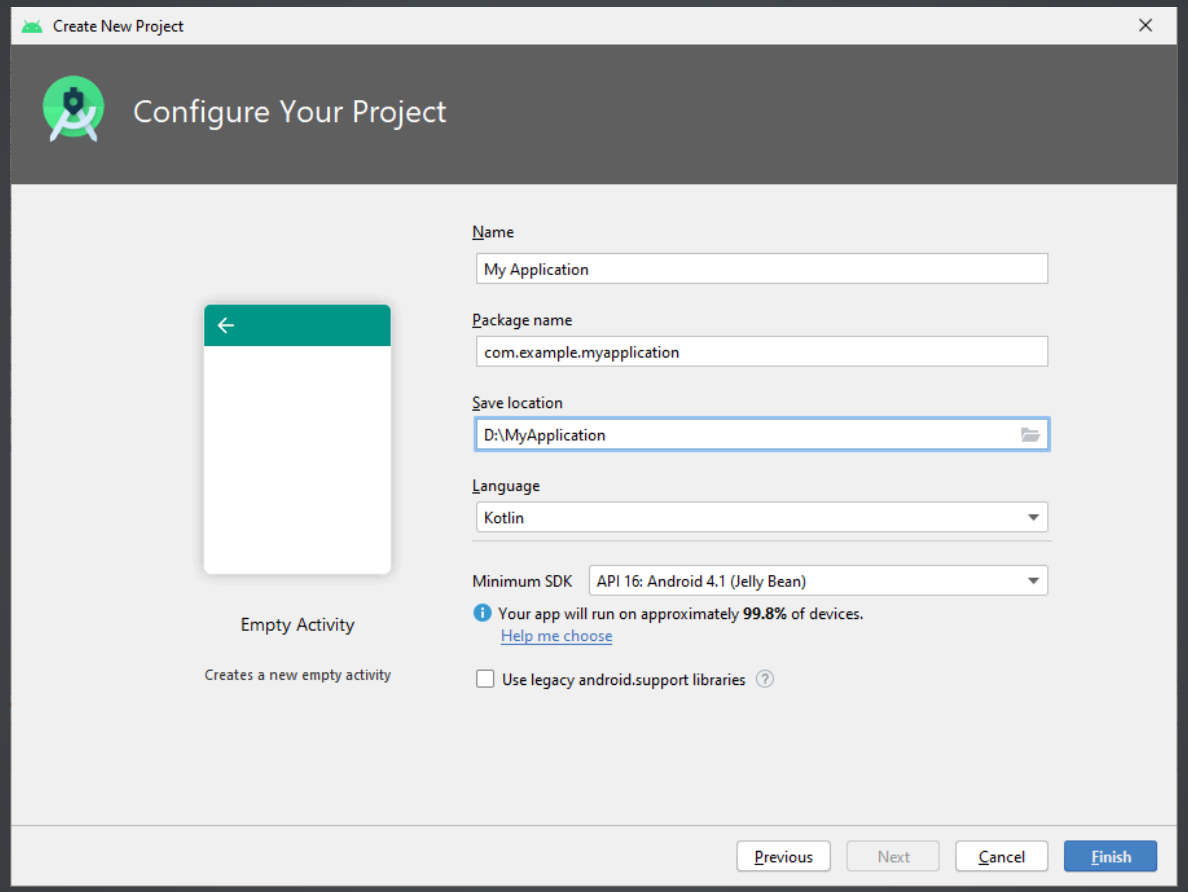
# Les terminaux cibles



# Les terminaux cibles



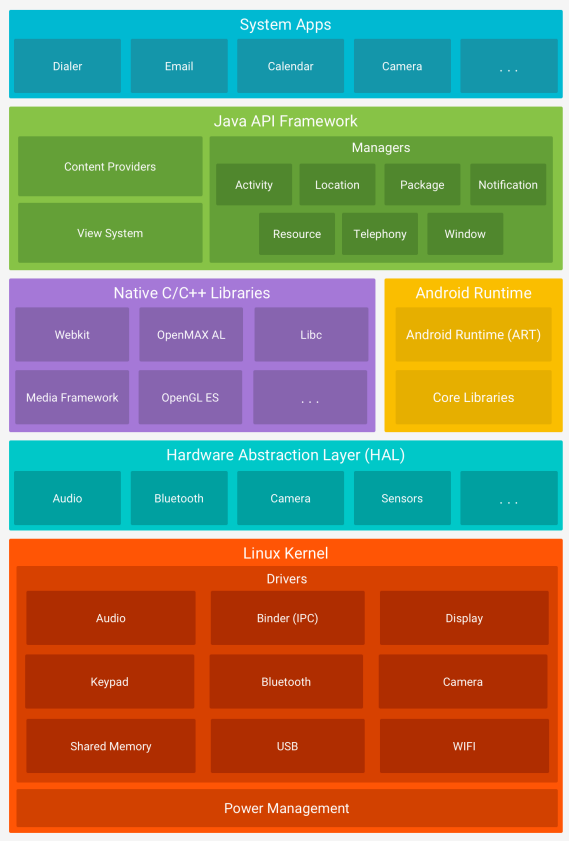
# Les terminaux cibles



# Développement Android

- Les concepts de base. Le cycle développement.
- Les classes de base du framework.
- Le projet sous Android Studio.
- L'émulateur du SDK. Les outils du SDK, SDK manager, AVD manager.
- L'utilisation des outils sous Android Studio : debugger, profiler, etc.
- Les paramètres du manifest.
- La production de l'application, la publication.

# Les concepts de base




Source : https://developer.android.com/guide/platform

# Les concepts de base

## Optimisation

Les applications sur smartphone doivent veiller à optimiser tous les points, tels quel :

- Mémoire.
- Processeur.
- Radio (WiFi, 3G, BT).
- Graphique.

En effet tous ces points vont avoir une répercussion sur la consommation d'énergie, et donc sur la batterie.

# Les concepts de base

## Le cycle développement

Le développement d'une application peut suivre plusieurs schémas, nous allons en détailler une possibilité :

1. Lancement : toutes les applications commencent par une idée. Cette idée est généralement affinée jusqu'à constituer une base solide pour une application.
2. Conception : la phase de conception consiste à définir l'expérience utilisateur de l'application, comme la présentation générale, son fonctionnement, etc., ainsi que la conversion de cette expérience utilisateur en une interface utilisateur appropriée, généralement avec l'aide d'un infographiste.
3. Développement : généralement la phase la plus consommatrice de ressources, c'est la phase de création réelle de l'application.
4. Stabilisation : quand le développement est suffisamment avancé, l'assurance qualité commence généralement à tester l'application et les bogues sont corrigés. Le plus souvent, une application passe par une phase bêta limitée, durant laquelle un public plus large a la possibilité de l'utiliser, de fournir des commentaires et d'obtenir des modifications.
5. Déploiement

# Les concepts de base

## Conception des interfaces utilisateur

Afin de définir notre interface utilisateur, Google fournit des guides pour bien réaliser cette tâche : https://developer.android.com/design/index.html
Il nous appartiendra à bien qualifier notre cible (type de terminaux : smartphone, tablette, TV, montre, ordinateur de voiture, ...).

# Les concepts de base

## Phases de développement

Nous pouvons passer par plusieurs phases pour réaliser l'application finale :

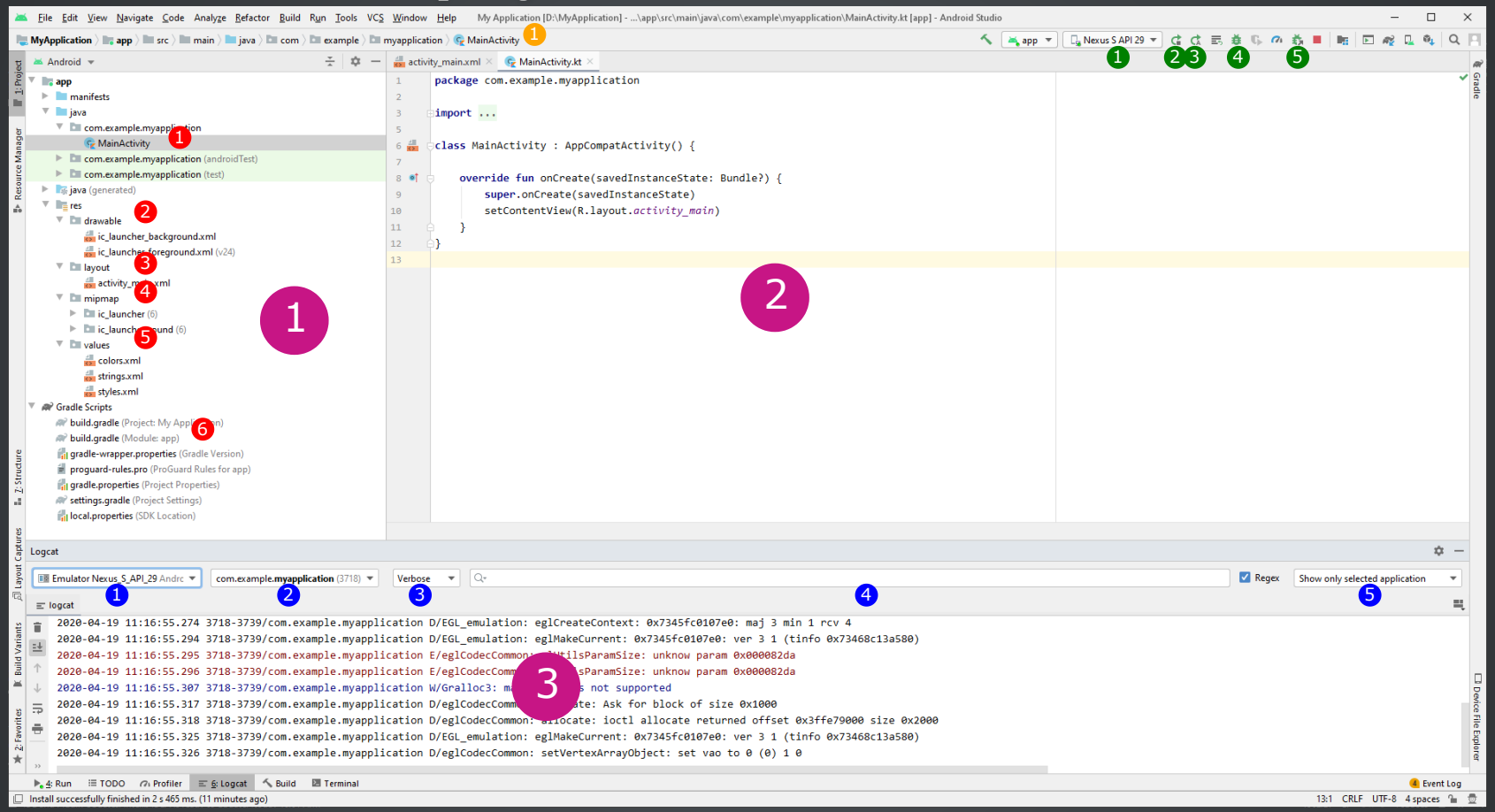
- Prototype/MVP : l’application est toujours en phase de preuve de concept, et seules les fonctionnalités principales ou des parties spécifiques de l’application fonctionnent. Des bogues majeurs y sont présents.
- Alpha : les fonctionnalités principales sont généralement entièrement présentes dans le code (qui est généré, mais pas entièrement testé). Des bogues majeurs sont encore présents, des fonctionnalités périphériques peuvent ne pas encore être présentes.
- Bêta : la plupart des fonctionnalités sont maintenant terminées, une partie des tests et de la correction des bogues a été effectuée. Des problèmes majeurs connus peuvent encore être présents.
- Version Release Candidate : toutes les fonctionnalités sont terminées et testées. Sauf si de nouveaux bogues apparaissent, l’application est candidate à la publication.

# Les classes de base du framework

Plusieurs classes de bases sont disponibles dans le Framework, nous les détaillerons par la suite :

- Activity
- Fragment (3.0+)
- Service
- Content providers
- Broadcast receivers
- Intent

# Le projet sous Android Studio



# Le projet sous Android Studio

Les grandes parties en violet :
1. La liste des fichiers.
2. La zone d'édition.
3. Les logs.

# Le projet sous Android Studio

Détail des points en rouge :

1. Le code de notre application.
2. Le répertoire des ressources.
3. Le répertoire contenant les mises en forme des écrans (les layouts).
4. Le répertoire contenant les versions des icônes de l'application.
5. Le répertoire contenant les autres ressources.

# Le projet sous Android Studio

Détail des points en orange :

1. Le chemin complet du fichier courant sélectionné.

# Le projet sous Android Studio

Détail des points en vert :

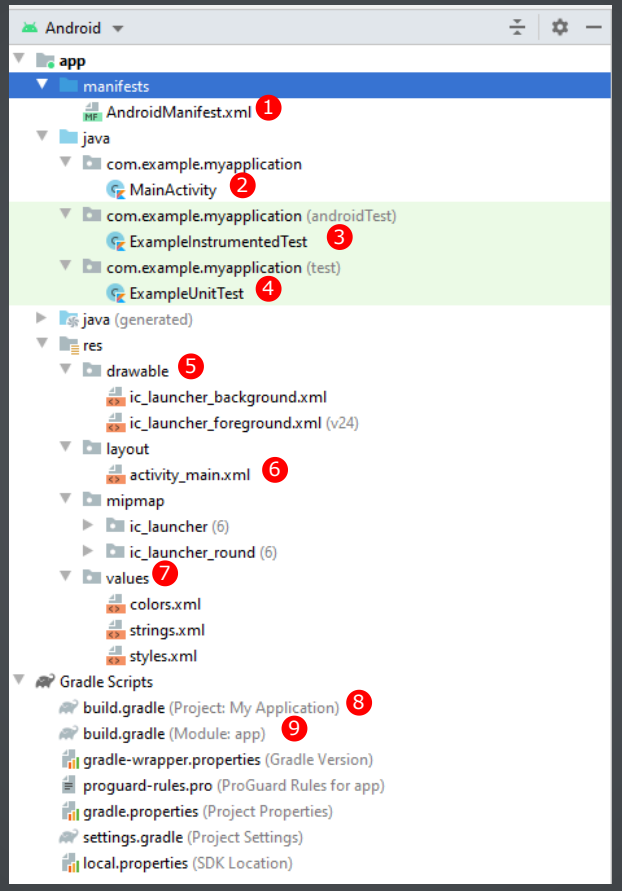
1. Le choix de la cible pour lancer le programme.
2. Relancer l'application.
3. Relancer l'activity.
4. Relancer en mode débug.
5. Attacher à la volée le debugger.

# Le projet sous Android Studio

Détail des points en bleu, il s'agit du Logcat :

1. Nous pouvons choisir le device sur lequel se connecter.
2. Sélectionner le process pour filtrer les messages.
3. Sélectionner le type de log des messages.
4. Filtrer les messages.
5. Activer le filtre.

# Le projet sous Android Studio



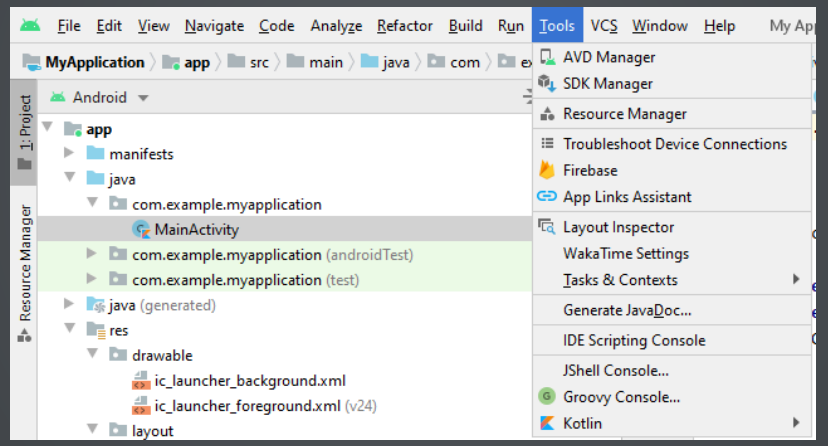
# Le projet sous Android Studio

Les fichiers principaux rouge :

1. Le manifest (AndroidManifest.xml), qui contient la carte d'identité de notre application.
2. Le code de l'application.
3. Le code des tests graphiques.
4. Le code des tests unitaires.
5. Les ressources graphiques.
6. Les écrans (layouts) de notre application.
7. Les ressources (texte, couleurs, dimensions, styles, ...) de notre application.
8. Le fichier de configuration global du système de compilation (gradle).
9. Le fichier de configuration de notre projet (celui qui sera le plus souvent modifié).

# Émulateur

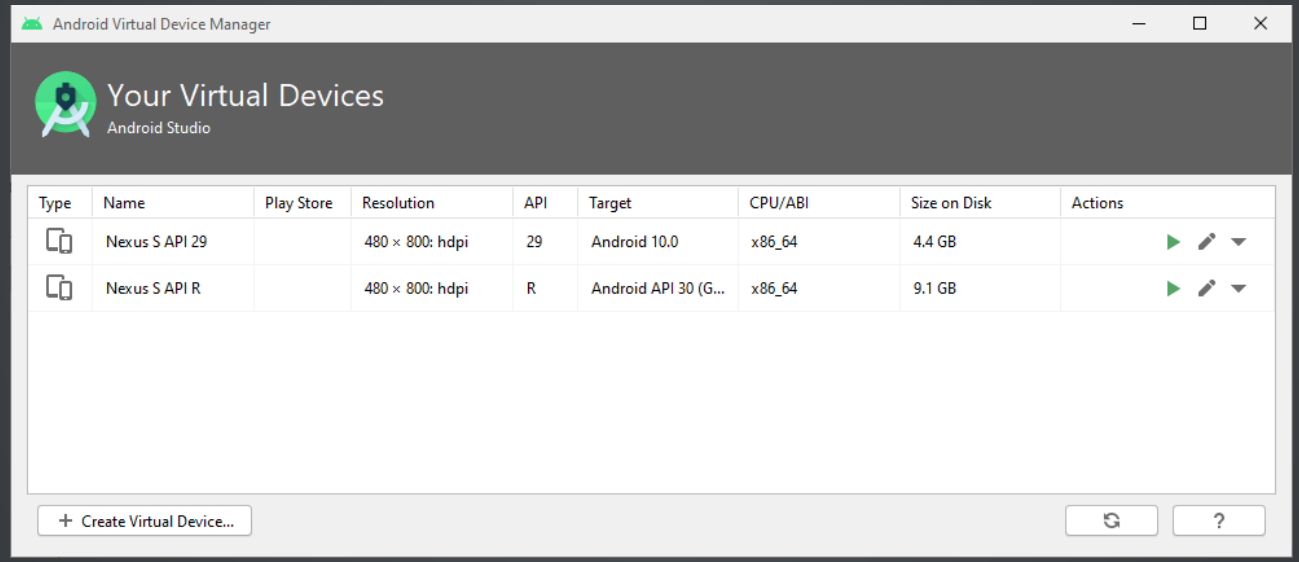
Lançons l'écran de gestion des machines virtuelles, avec le menu : `Tools/AVD Manager`



# Émulateur

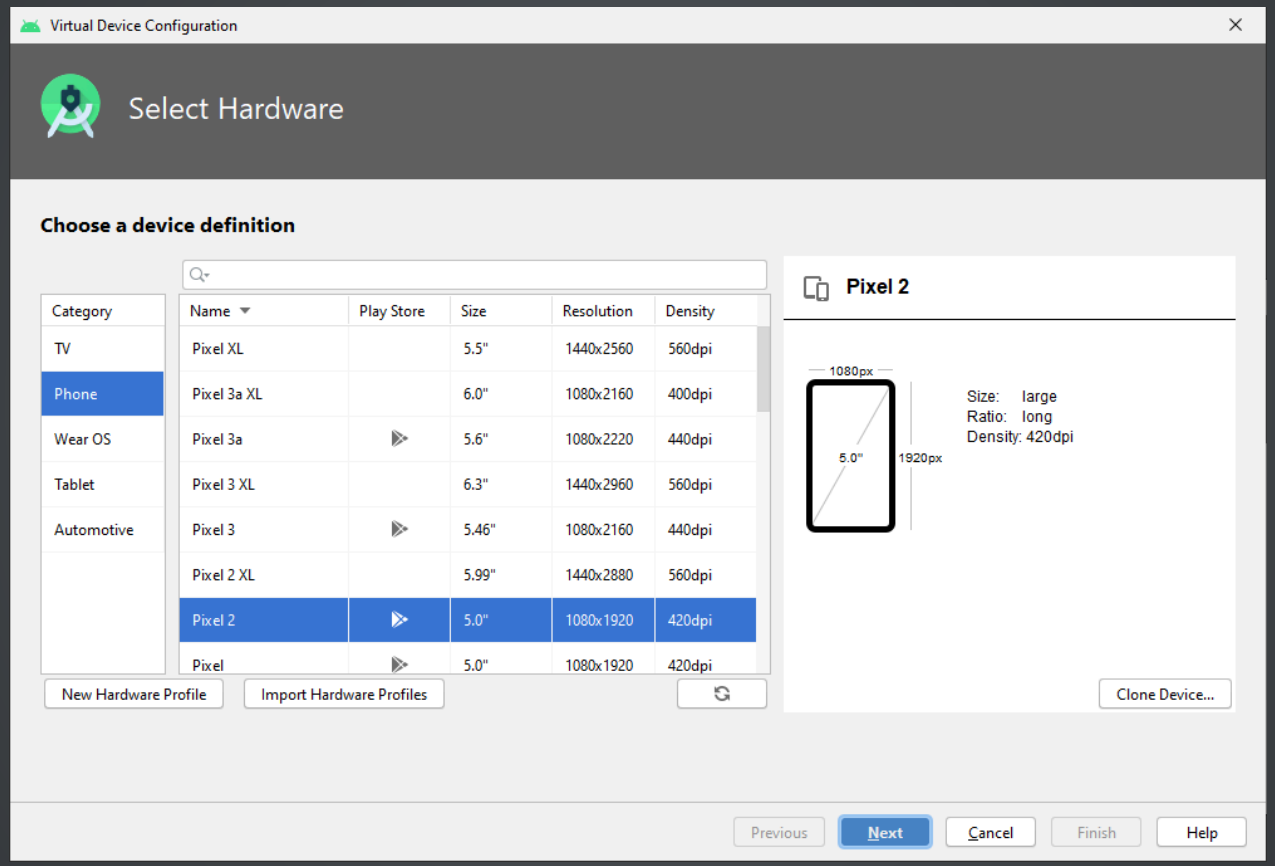
## Choix de la machine virtuelle

Sur cet écran, nous pouvons choisir la machine virtuelle à lancer/configurer/supprimer.
Nous pouvons aussi en créer une nouvelle, en cliquant sur le bouton "Create Virtual Device..."



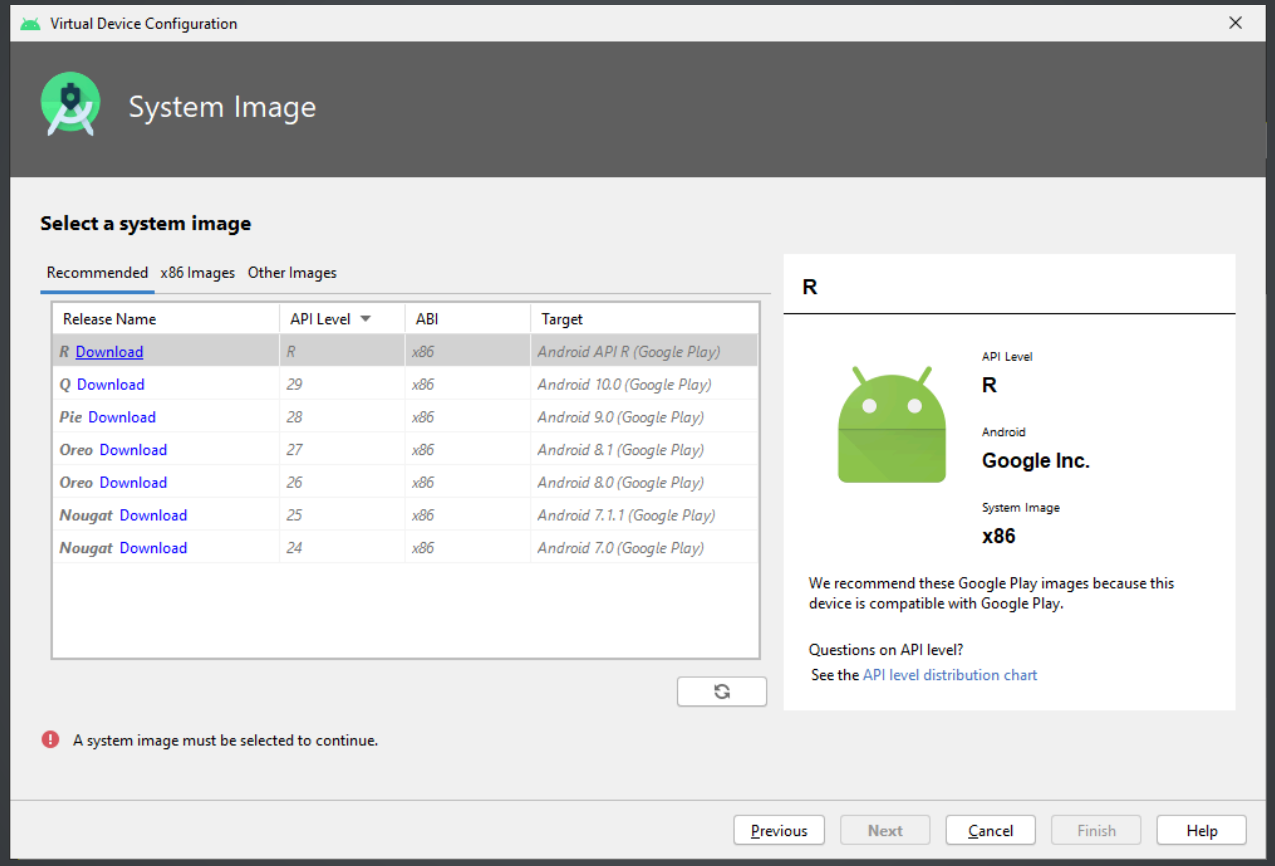
# Émulateur

Choisissons le modèle de device que nous voulons émuler :



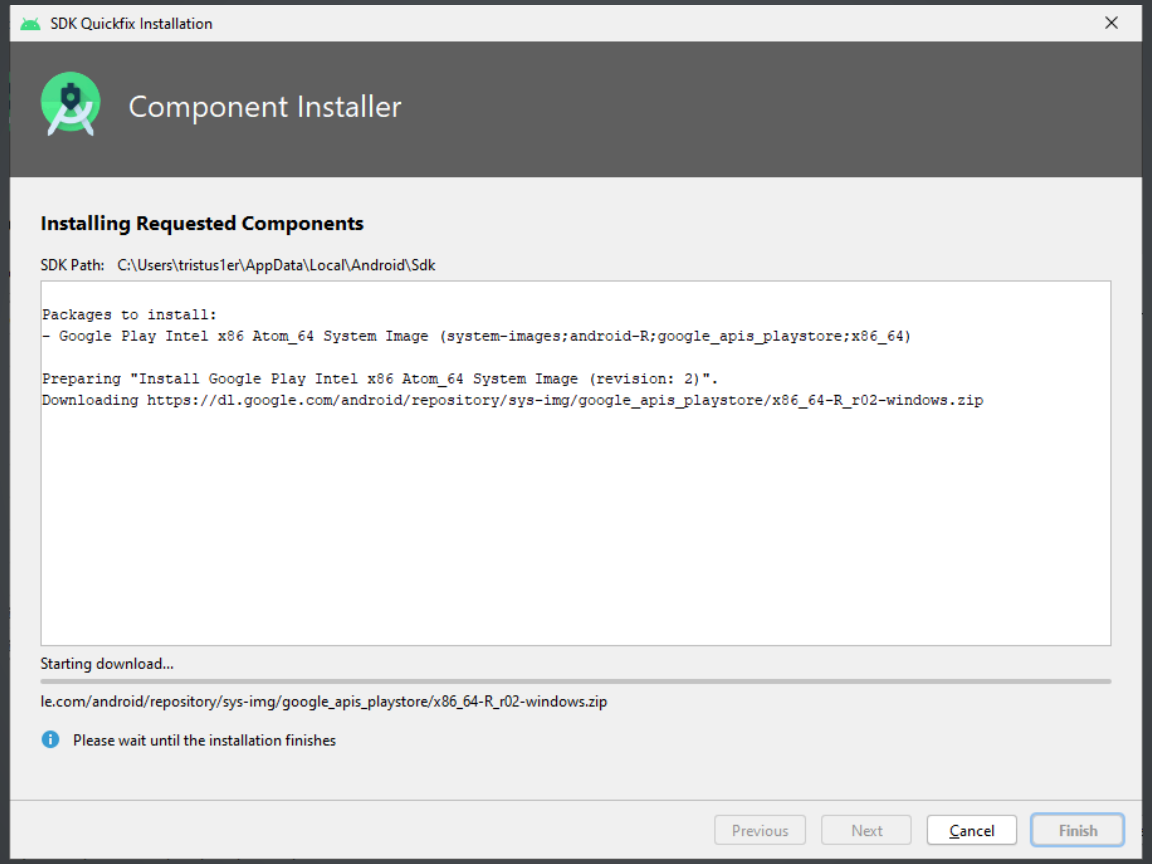
# Émulateur

Choisissons la version d'Android :



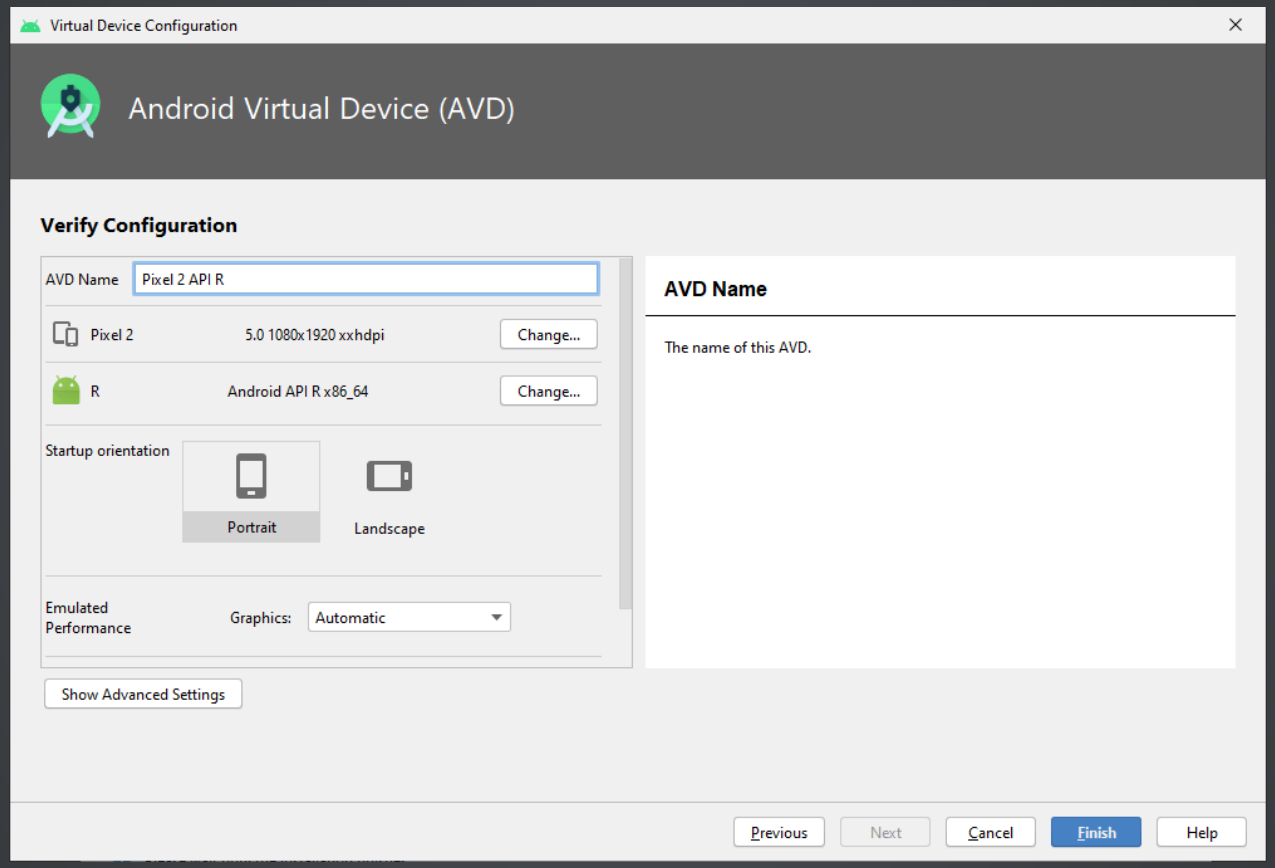
# Émulateur

En cas de besoin, nous pouvons directement télécharger une image :



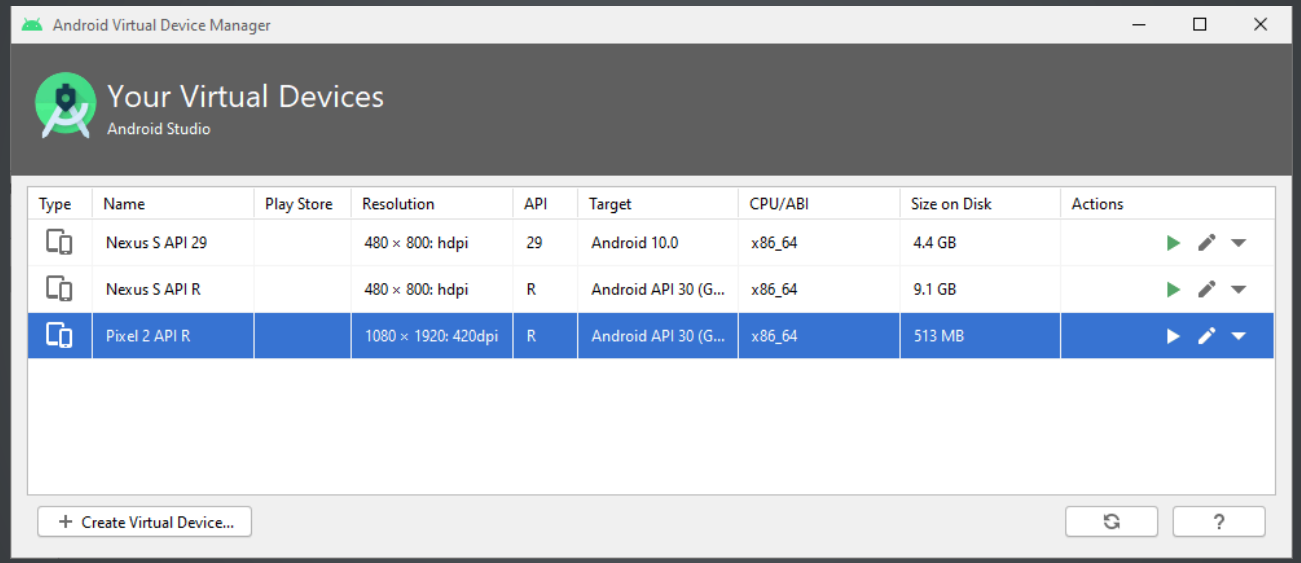
# Émulateur

Donnons un nom à notre machine virtuelle, nous laissons les paramètres par défaut :



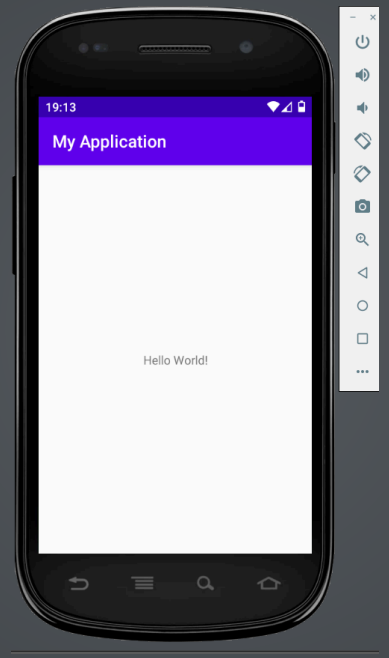
# Émulateur

Notre machine virtuelle est maintenant disponible, lançons la :



# Émulateur

L'émulateur permet de faire fonctionner nos applications sur des versions d'Android que nous n'avons pas forcément sur notre smartphone :

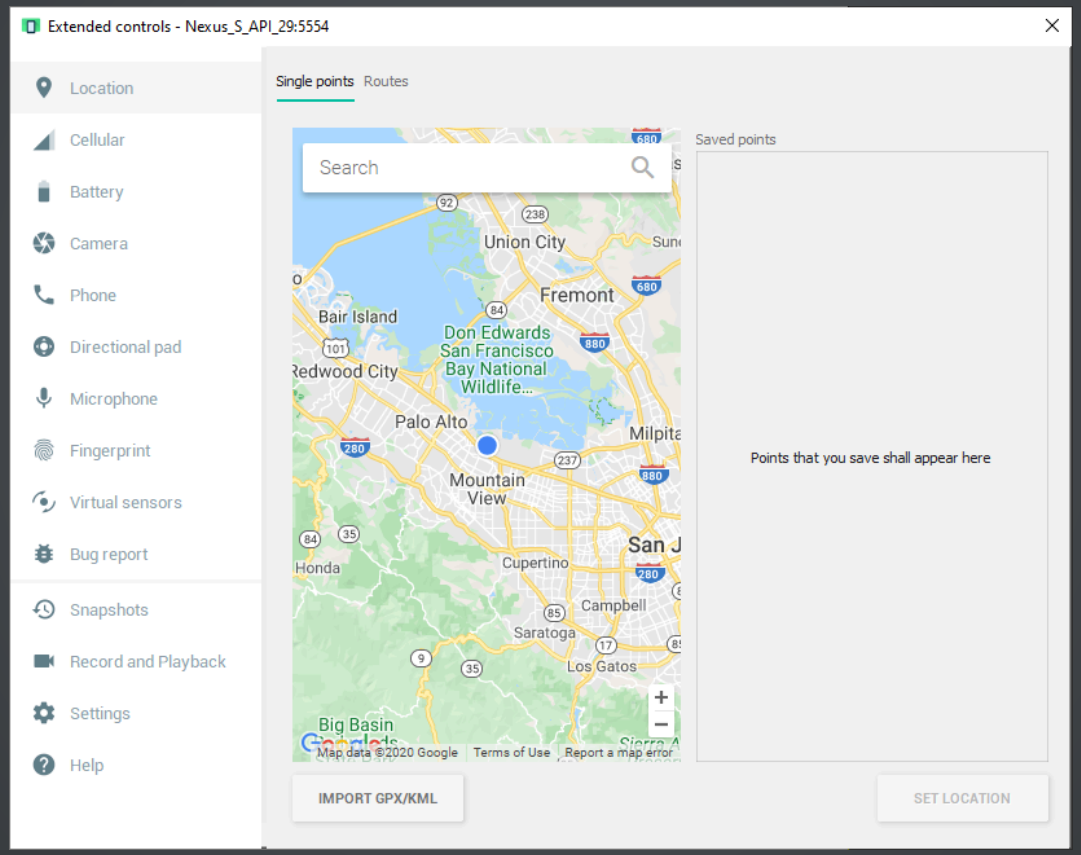



Cliquons sur les "..."

# Émulateur

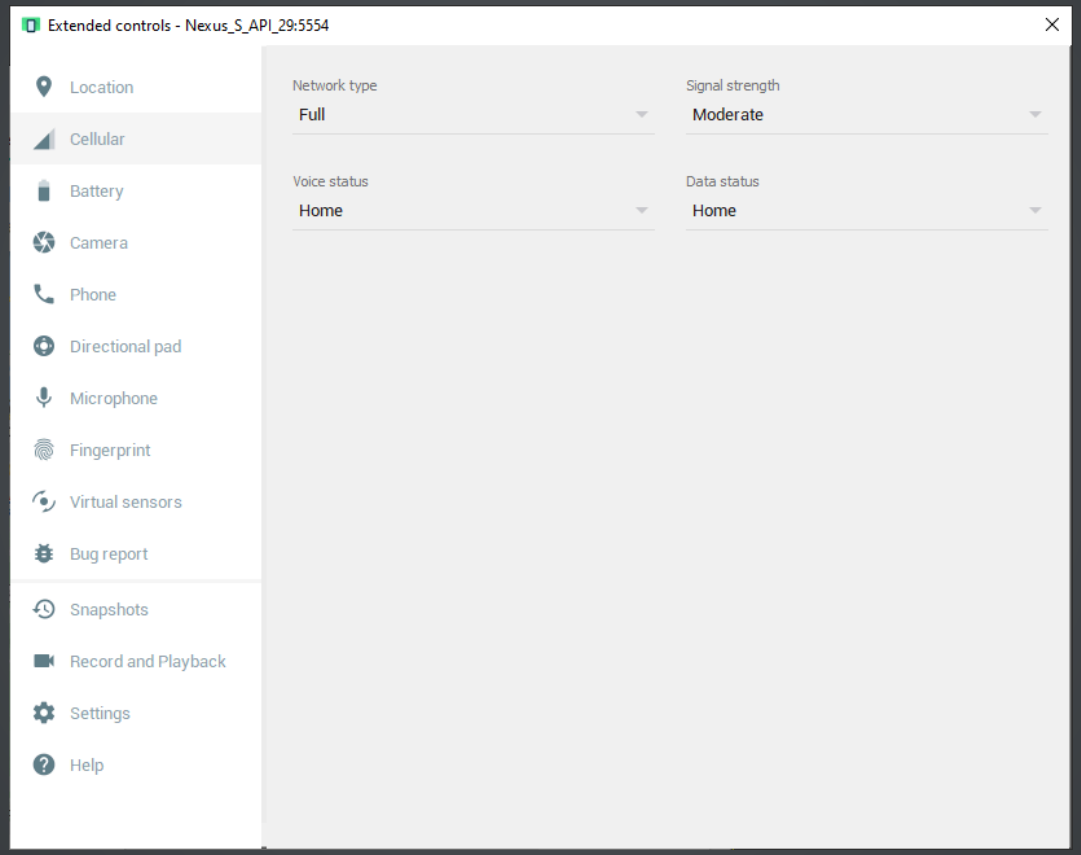
Nous pouvons :

Gérer la position du device directement sur une carte (nouvelle fonctionnalité Android Studio 3.6) :



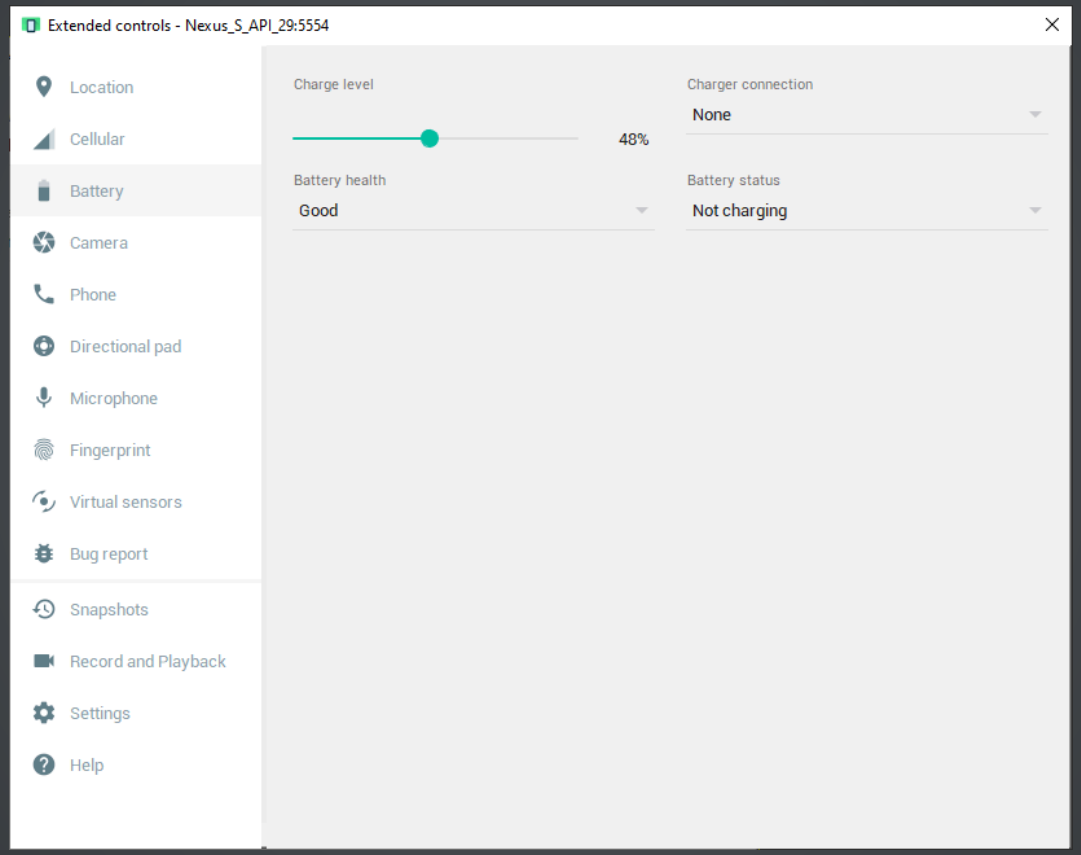
# Émulateur

Gérer la réception réseau :



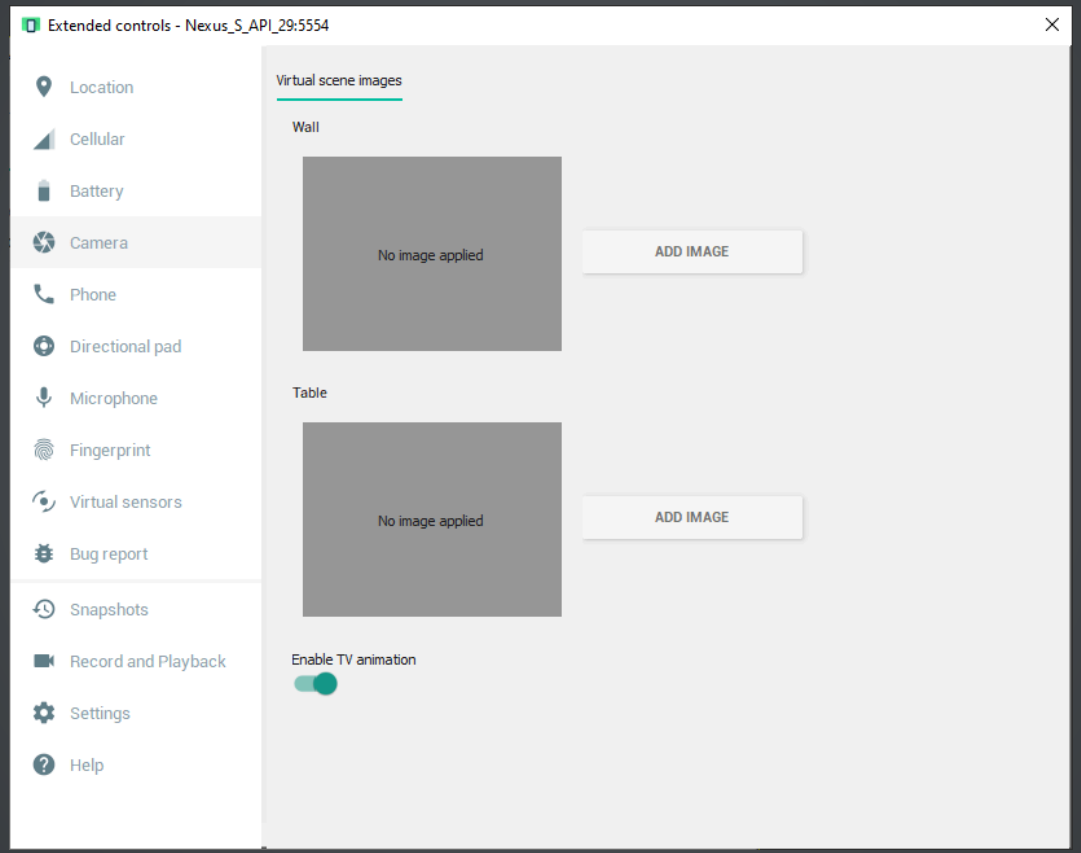
# Émulateur

Définir le niveau de batterie :



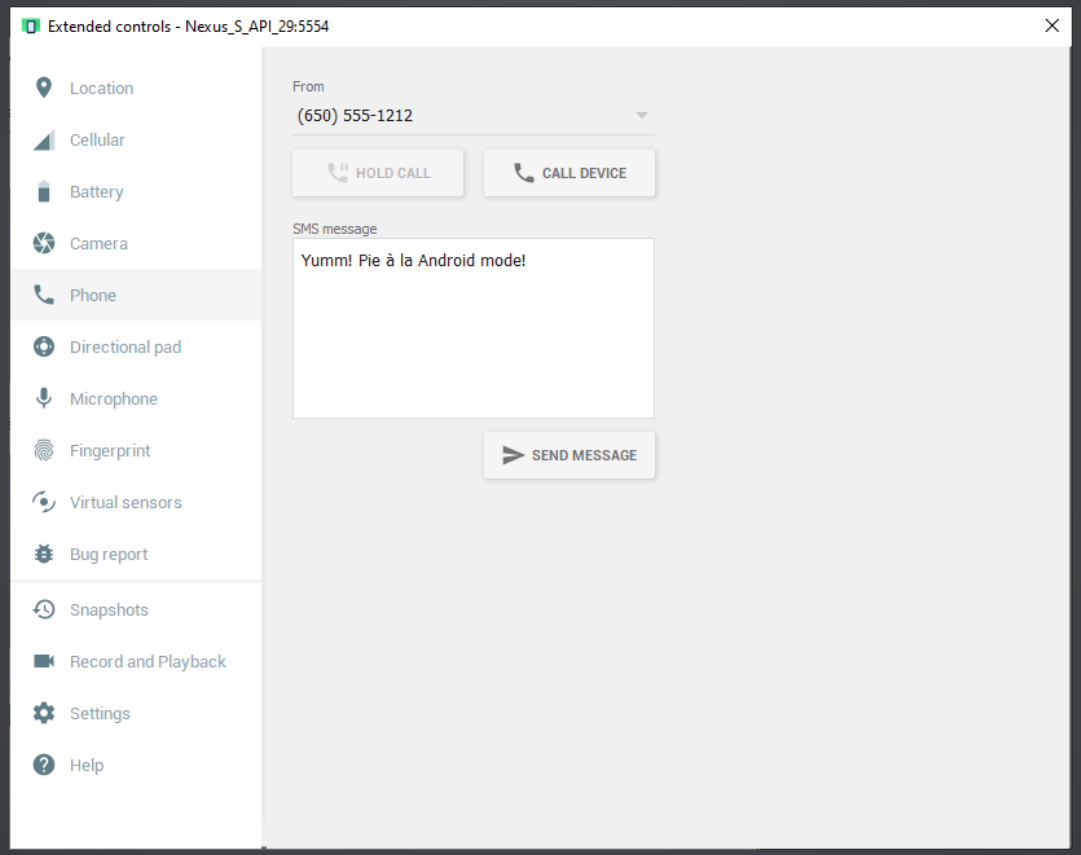
# Émulateur

Définir ce que "voit" la/les caméra(s) :



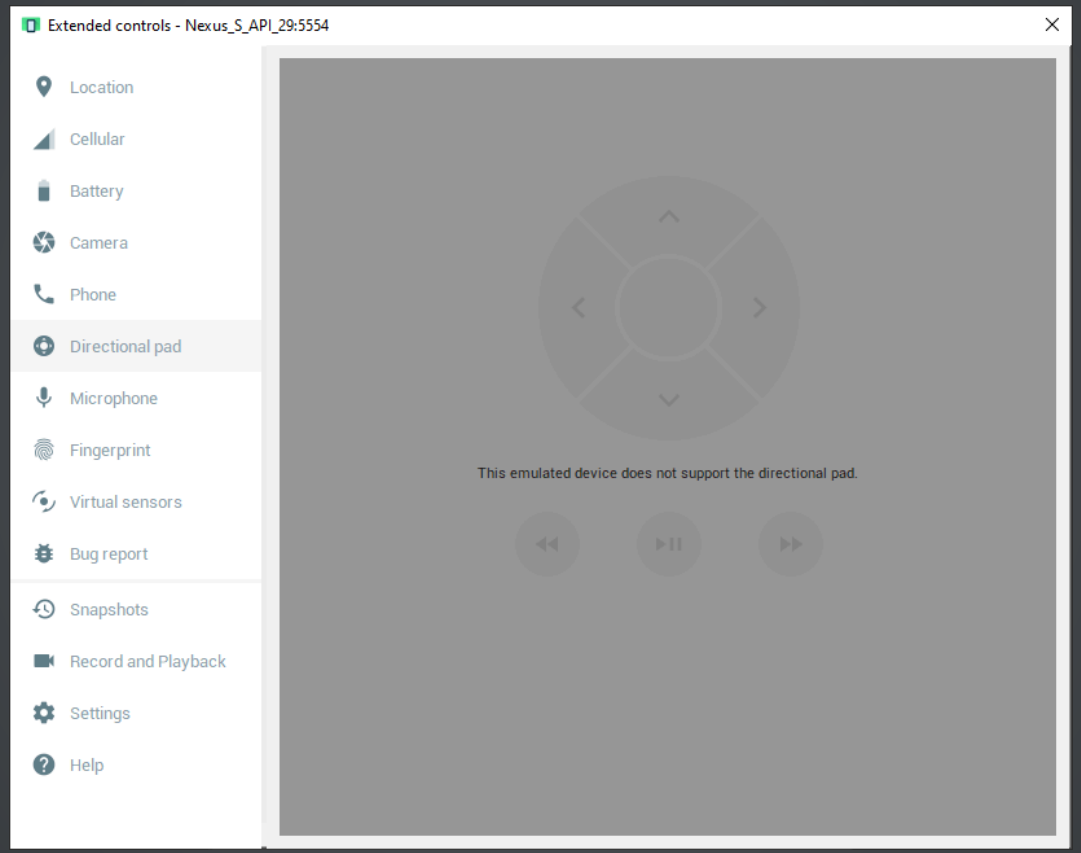
# Émulateur

Lancer des appels téléphoniques
et envoyer des SMS :



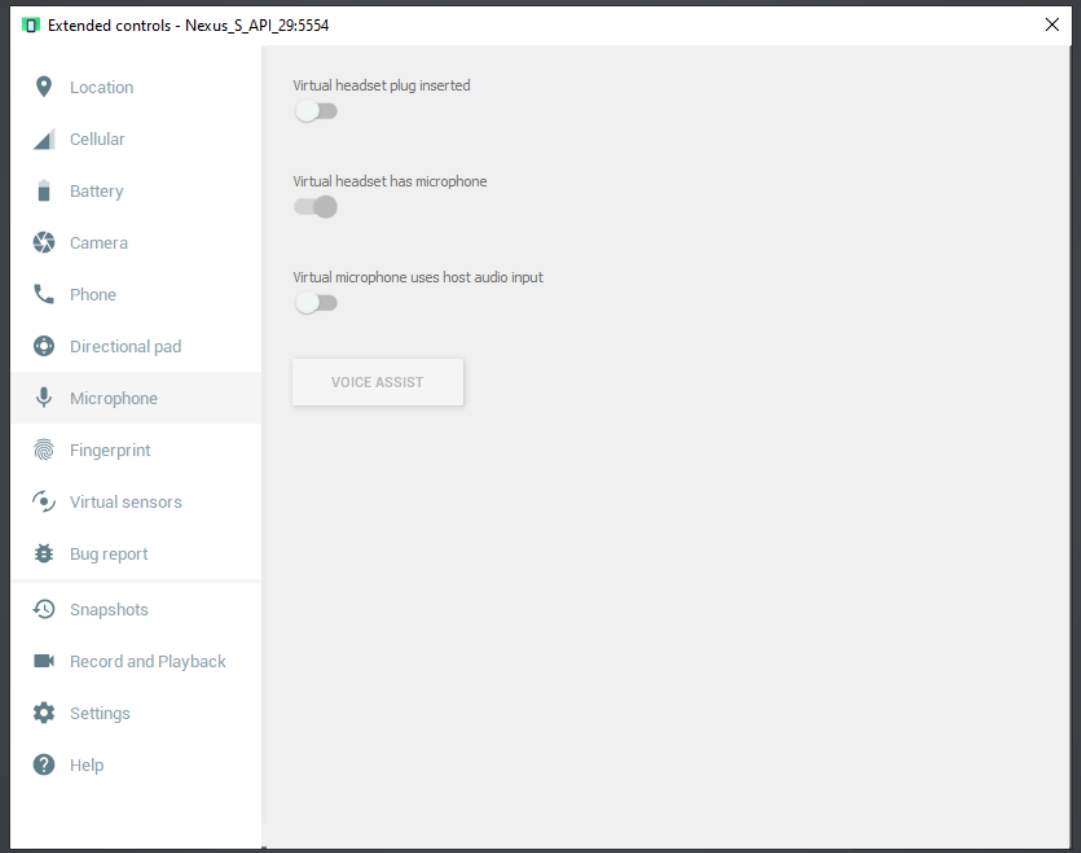
# Émulateur

Émuler le pad :



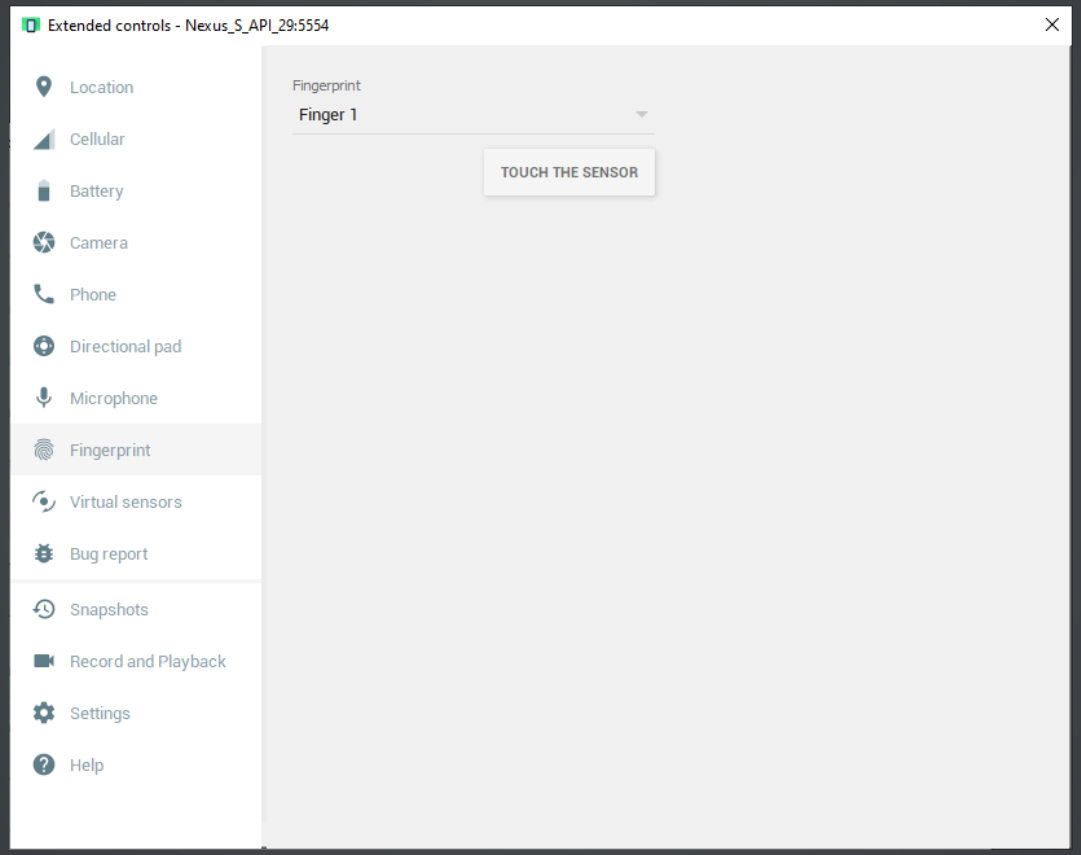
# Émulateur

Émuler la gestion du micro :



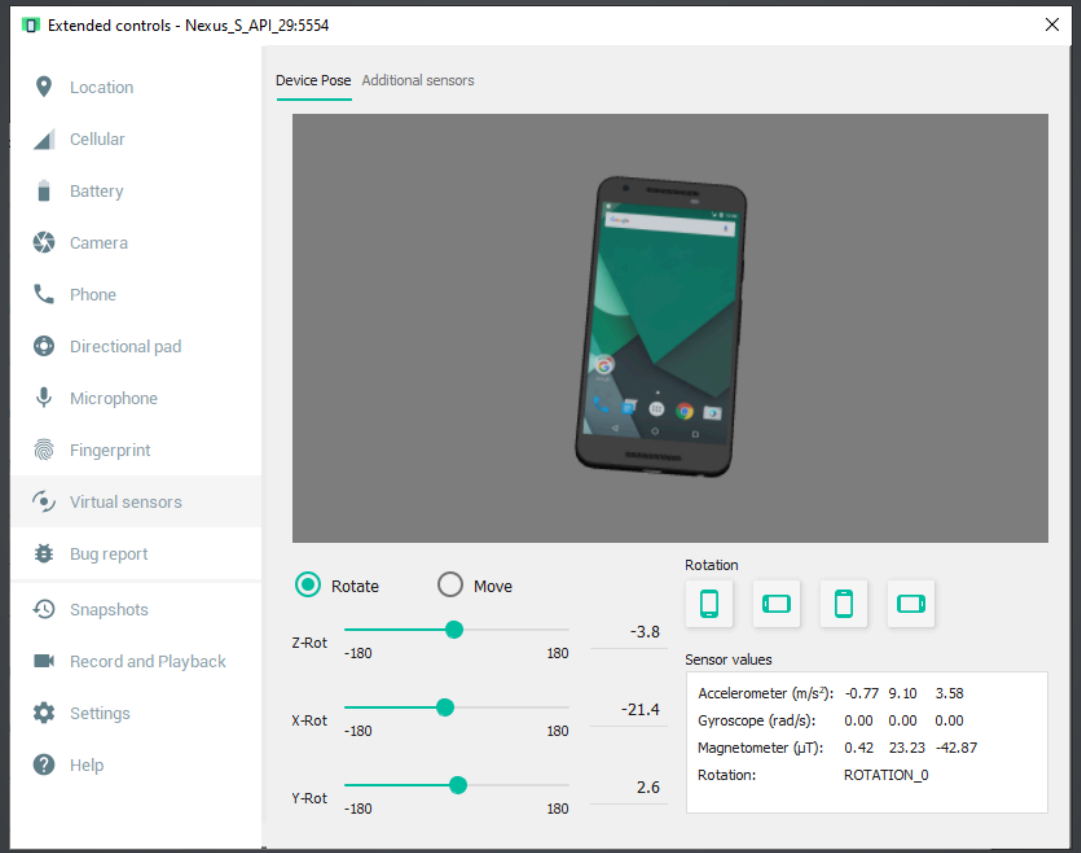
# Émulateur

Gérer les empruntes tactiles :



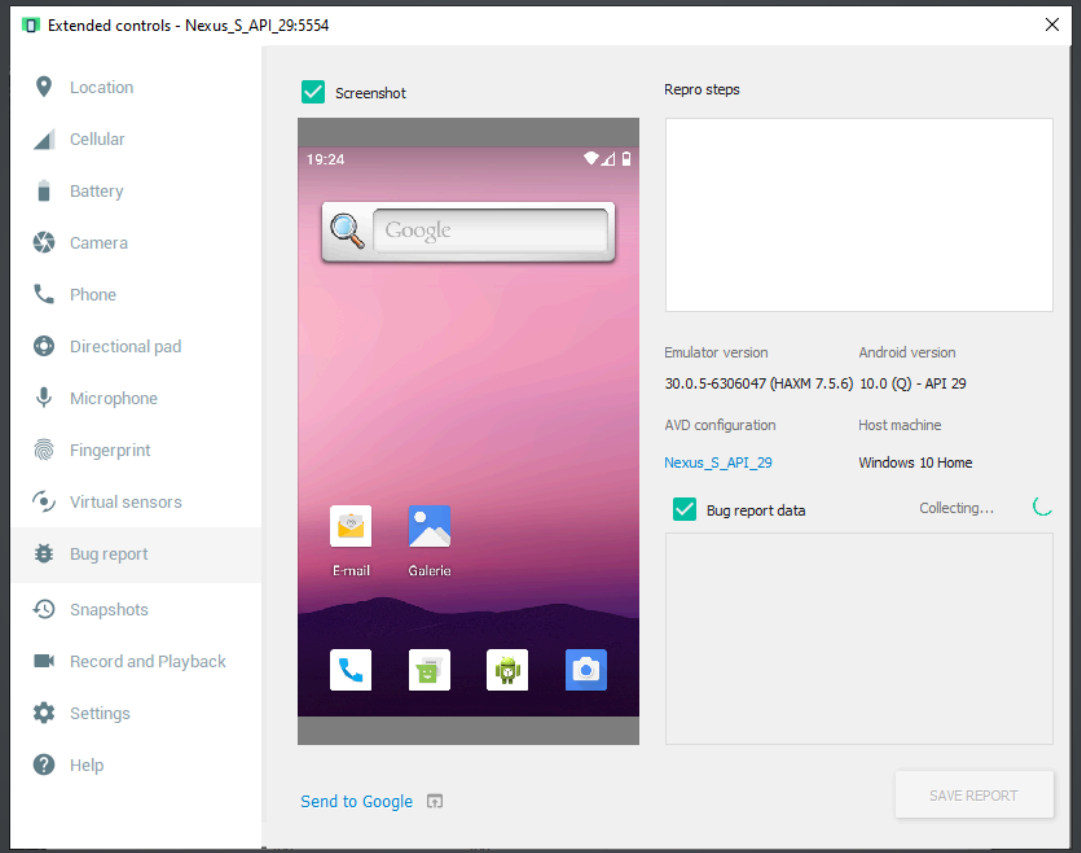
# Émulateur

Émuler les capteurs (rotations, et accélération) :



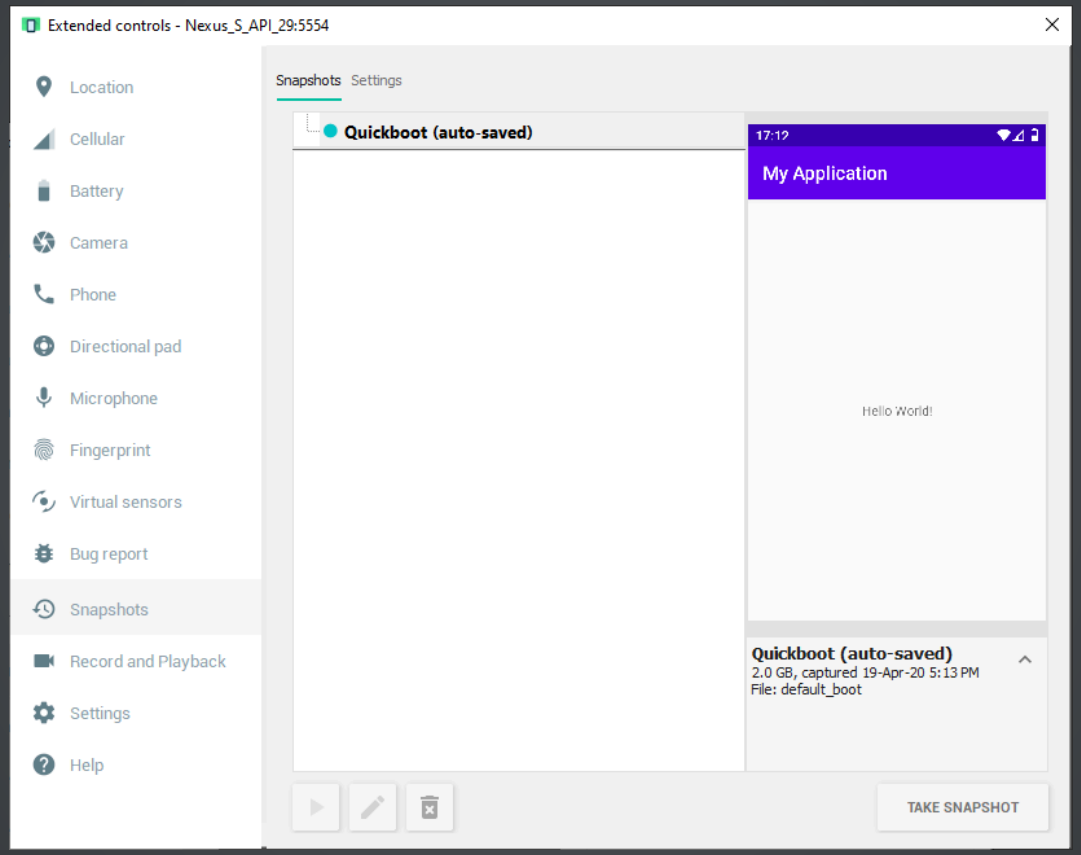
# Émulateur

Envoyer un rapport de bug :



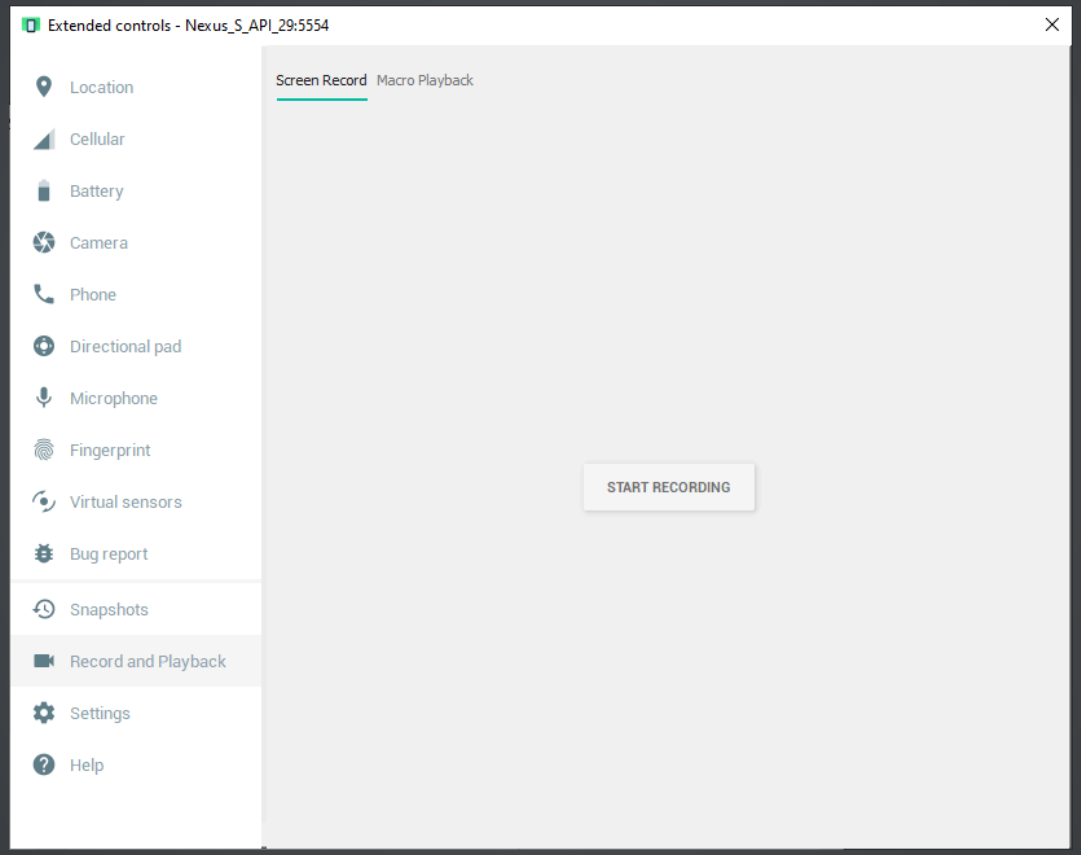
# Émulateur

Définir des points de restauration :



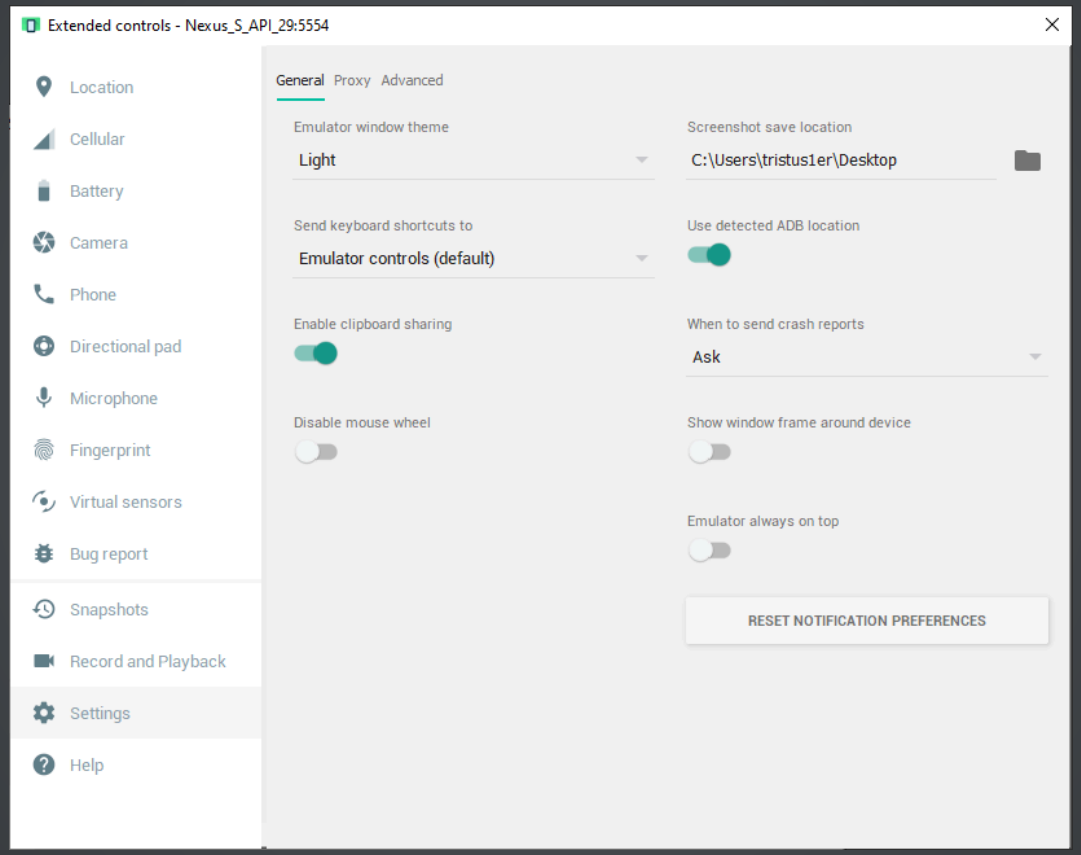
# Émulateur

Enregistrer l'écran :



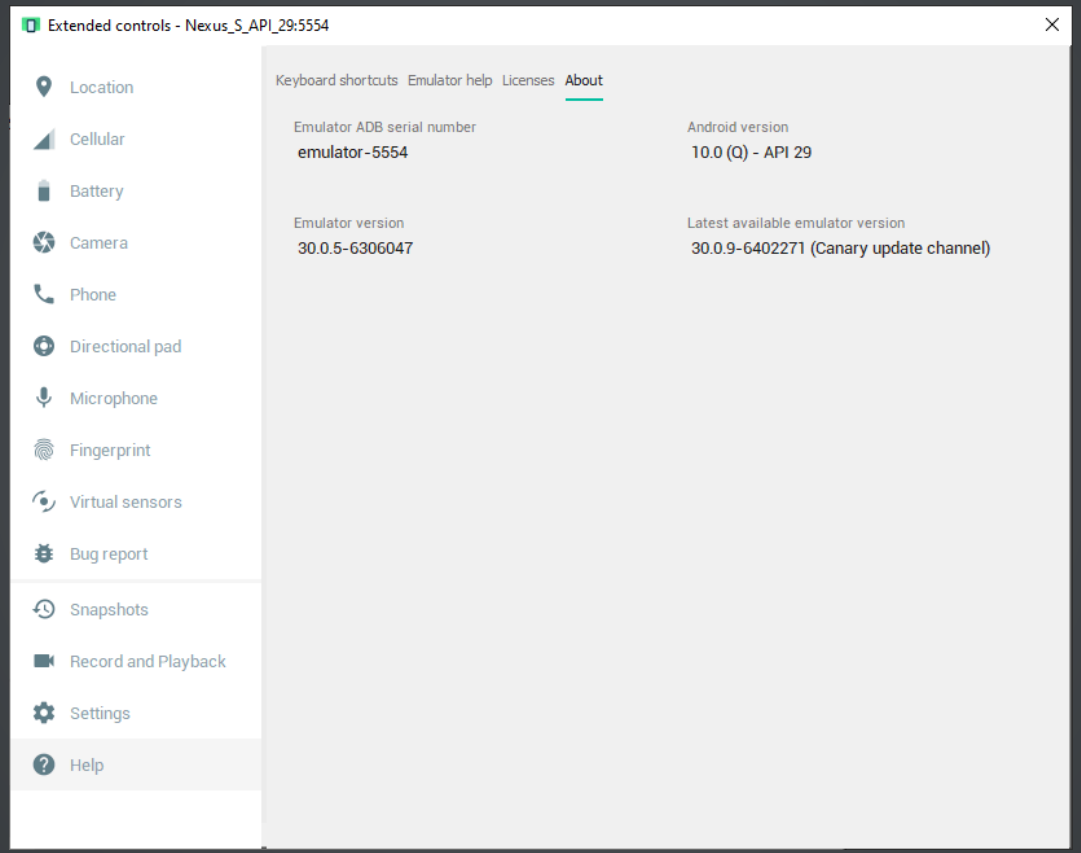
# Émulateur

Définir certains paramètres :



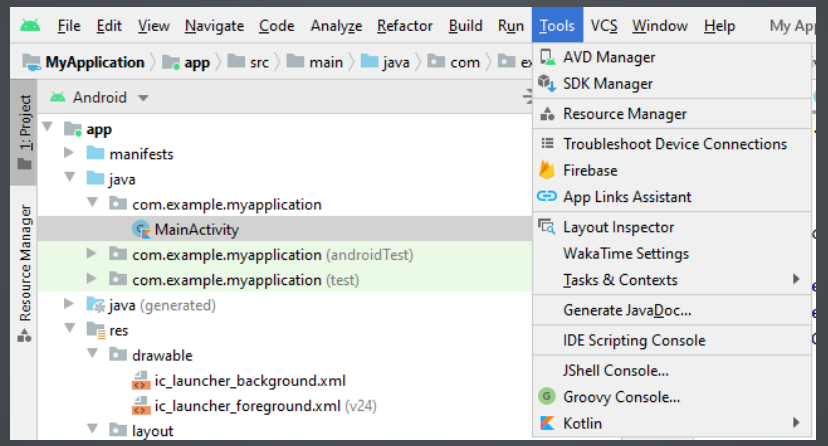
# Émulateur

Obtenir des informations sur l'émulateur :



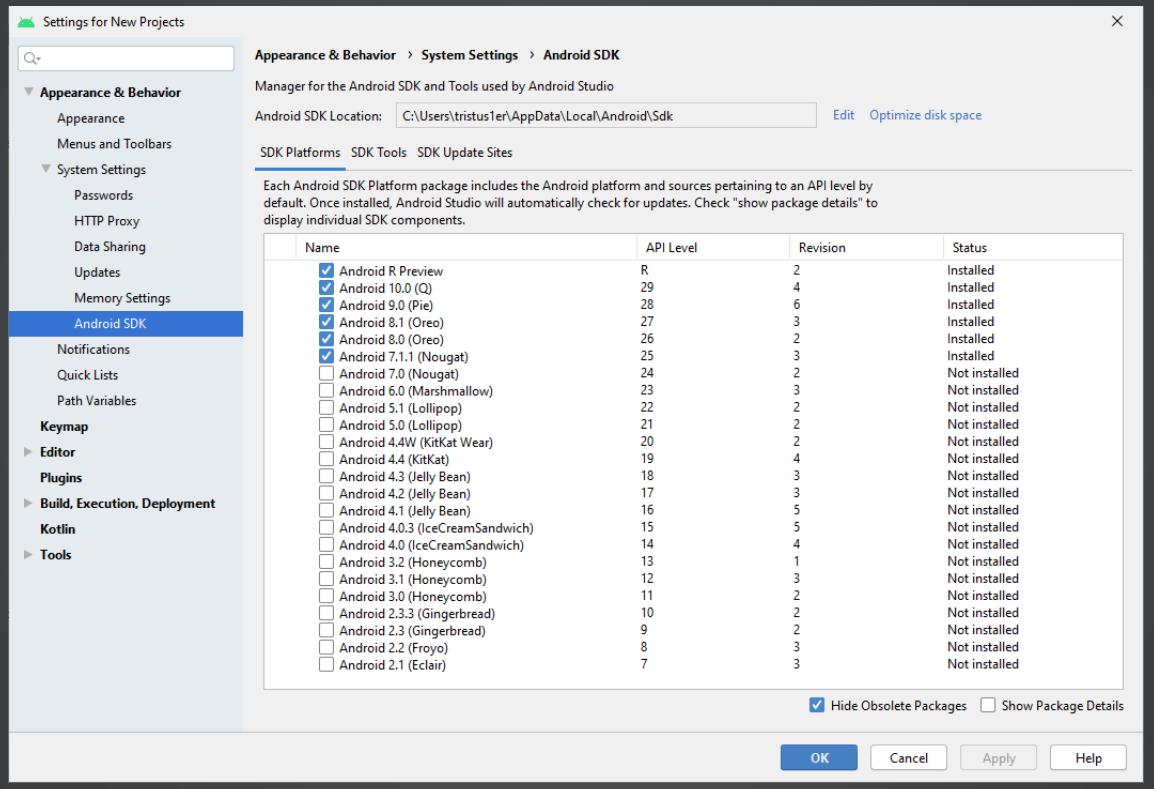
# SDK

Lançons l'écran de gestion du SDK :
`Tools/SDK Manager`



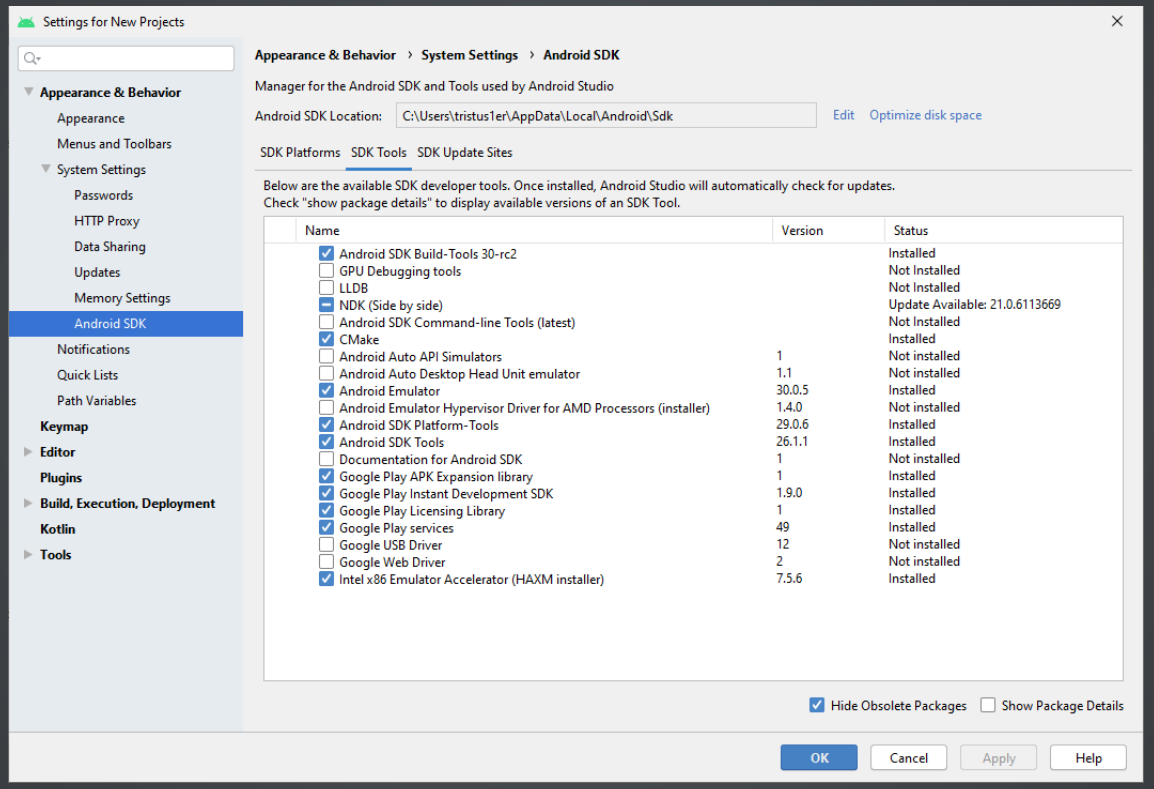
# SDK

Nous pouvons choisir les versions de l'OS à télécharger :



# SDK

Et les outils à installer :



# Le Manifest

Le manifest est le fichier qui décrit l'application, pour les outils de compilation, pour le système d'exploitation et pour Google Play. On y retrouve principalement :

- Les interfaces avec l'extérieur : les activities, les services, ...
- Les permissions que nous allons utiliser.
- Le matériel nécessaire à notre application (NFC, Bluetooth, ...).

Tous les détails ici.

# Le Manifest

## Exemple

Copier

```xml
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android" package="com.example.myapplication">
    <application
            android:allowBackup="true"
            android:icon="@mipmap/ic_launcher"
            android:label="@string/app_name"
            android:roundIcon="@mipmap/ic_launcher_round"
            android:supportsRtl="true"
            android:theme="@style/AppTheme">
        <activity android:name=".MainActivity">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>
```

# La production de l'application, la publication

## La production de l'application

Une fois développée, principalement via Android Studio (il existe d'autres moyens, par exemple via des outils de développement cross plate-forme), l'application est enregistrée dans un APK/App Bundle, qui contient :

- Le code de l'application.
- Les ressources.
- Les assets.
- Les certificats/signatures (pour l'App Bundle, la signature est effectuée par Google).
- Le Manifest.

Le fichier APK est un fichier ZIP qu'il est possible d'ouvrir avec n'importe quel utilitaire sachant traiter ce type de fichier (7Zip, Winzip, ...).

# La production de l'application, la publication

## La publication

Une fois packagée, l'application est le plus souvent publiée sur la boutique de Google : Google Play. Il existe d'autres boutiques, des stores alternatifs, tels que :

- F-Droid (applications open source)
- APK Miror
- App brain
- Amazon
- GETJAR
- Uptodown App Store
- Aptoide
- QooApp

# La production de l'application, la publication

Il peut aussi s'agir de stores de constructeurs, tels que :

- Huawei AppGallery
- Samsung Galaxy App
- Boutique Mi

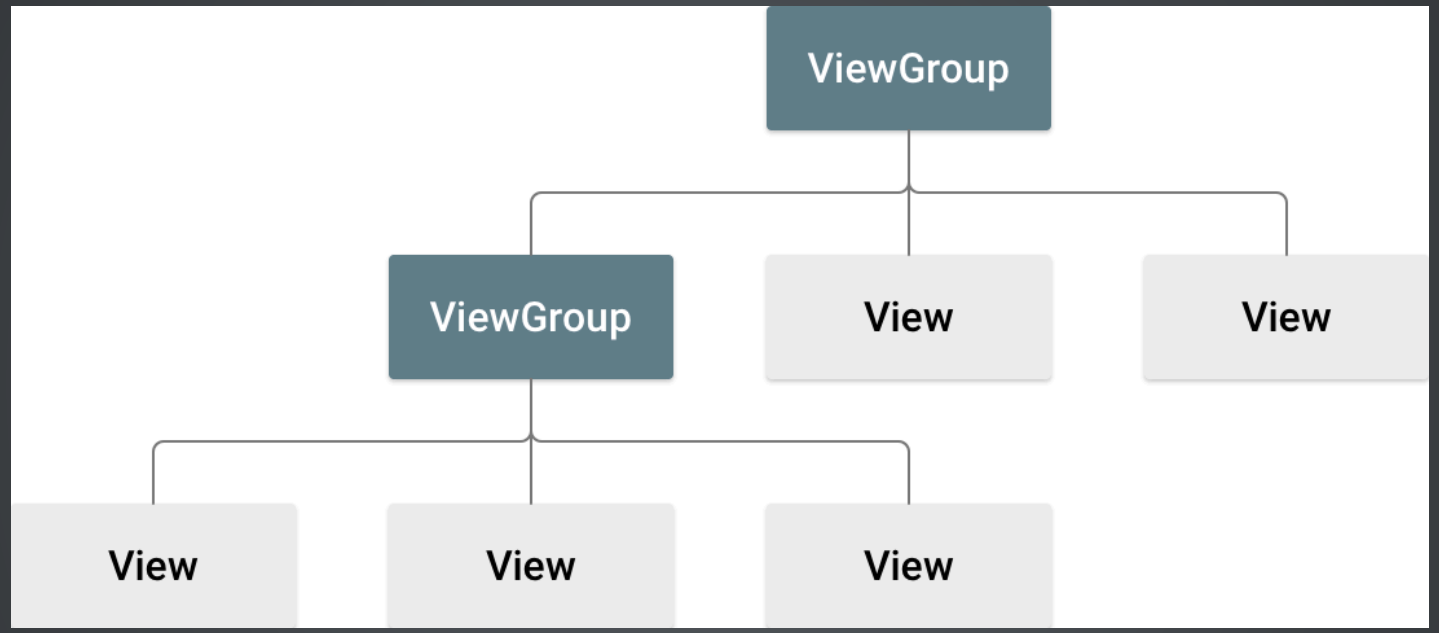
# La production de l'application, la publication

La solution classique consiste quand même à publier sur Google play store, via la Google Play Console. Pour cela, il faudra un compte Google, et débourser 25€ pour avoir la possibilité de publier une application sur le store, à vie.

# Les interfaces utilisateurs

- Organisation générale du layout.
- Exemple de layouts : LinearLayout, RelativeLayout, ConstraintLayout.
- Les ressources : drawables, string, styles, ...
- La gestion événementielle.

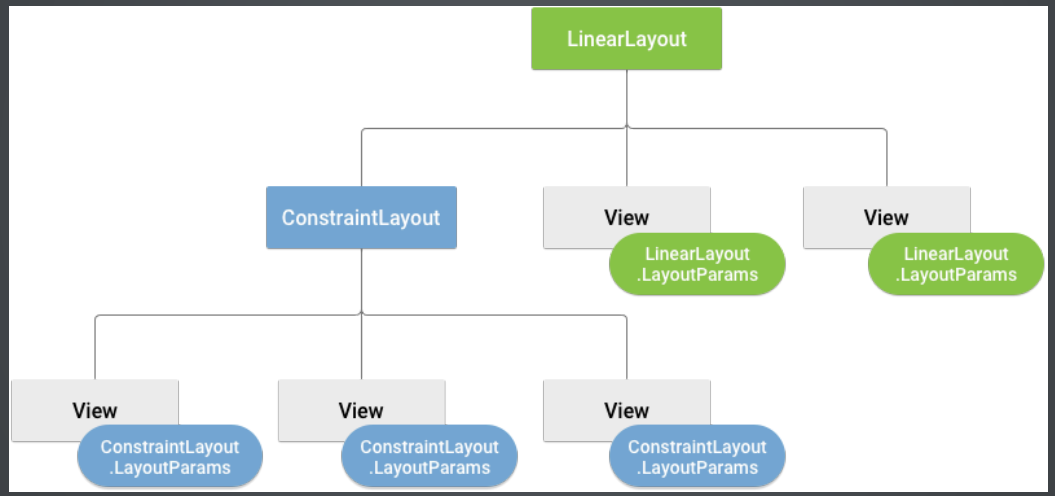
# Organisation générale du layout




Source : https://developer.android.com/guide/topics/ui/declaring-layout

# Organisation générale du layout

A chaque élément déclaré dans le layout, nous pouvons associer un id qui nous permettra de le retrouver/référencer dans le code (Java/Kotlin/XML).
Pour chaque layout, des contraintes particulières peuvent être appliquées (nous allons voir plusieurs layouts dans les slides suivantes).




Source : https://developer.android.com/guide/topics/ui/declaring-layout

# Exemple de layouts

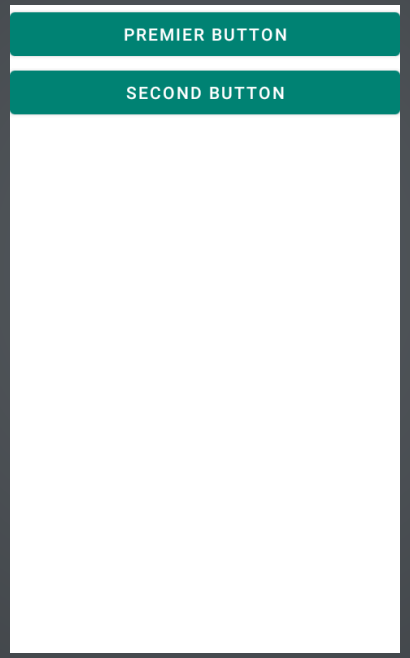
## Exercices sur les Layouts

Dans tous les prochains exercices, vous n'utiliserez qu'un seul layout par écran.

# Exemple de layouts

## LinearLayout Exercice 1

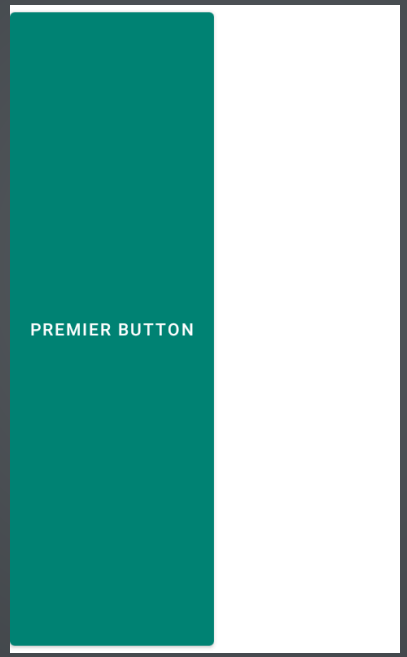
Les 2 boutons sont positionnés l'un au-dessous de l'autre, et prennent toute la largeur :



# Exemple de layouts

## LinearLayout Exercice 2

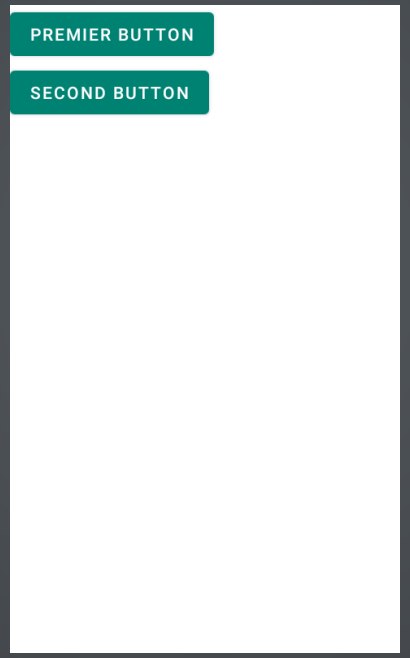
Le bouton prend toute la hauteur de l'écran :



# Exemple de layouts

## LinearLayout Exercice 3

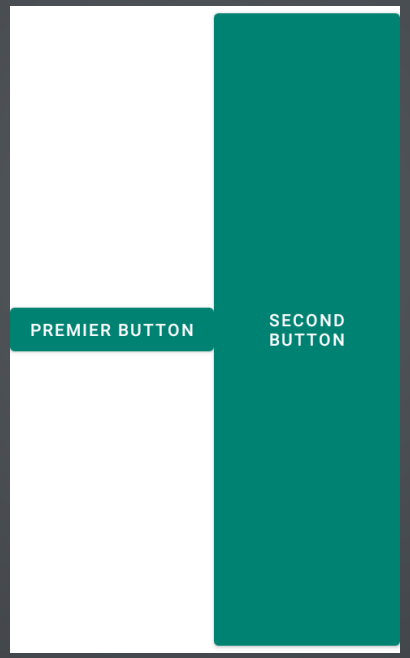
Les 2 boutons sont positionnés l'un au-dessous de l'autre, et prennent la largeur nécessaire à leur affichage :



# Exemple de layouts

## LinearLayout Exercice 4

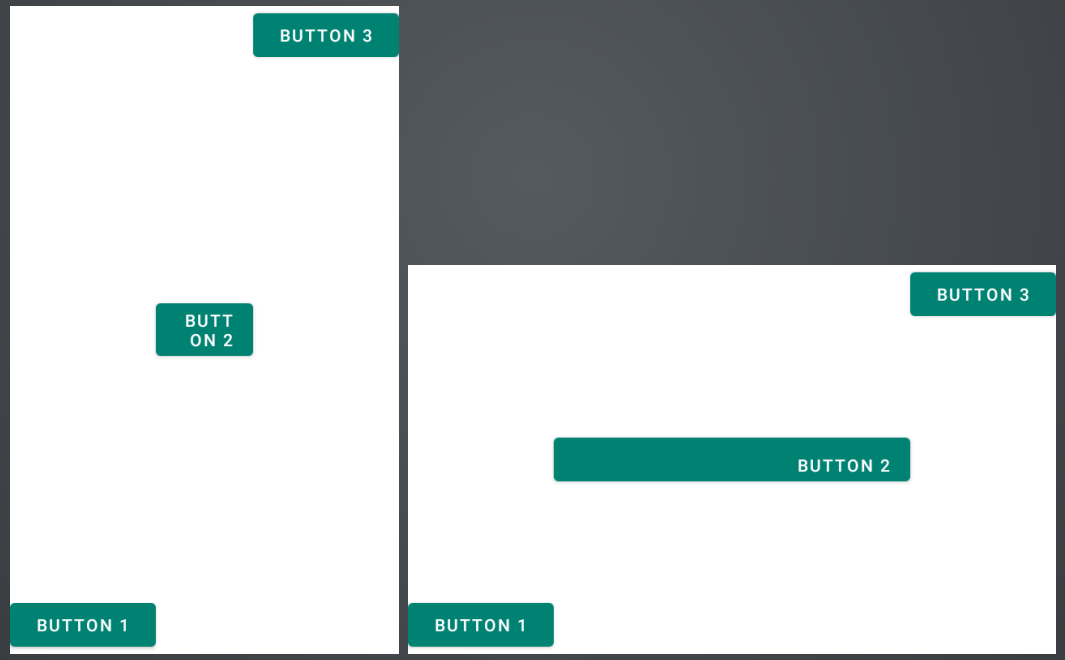
Les 2 boutons sont positionnés l'un à côté de l'autre, centré verticalement, le premier prend la hauteur nécessaire à son affichage, et le second prend toute la hauteur :



# Exemple de layouts

## LinearLayout Exercice 5

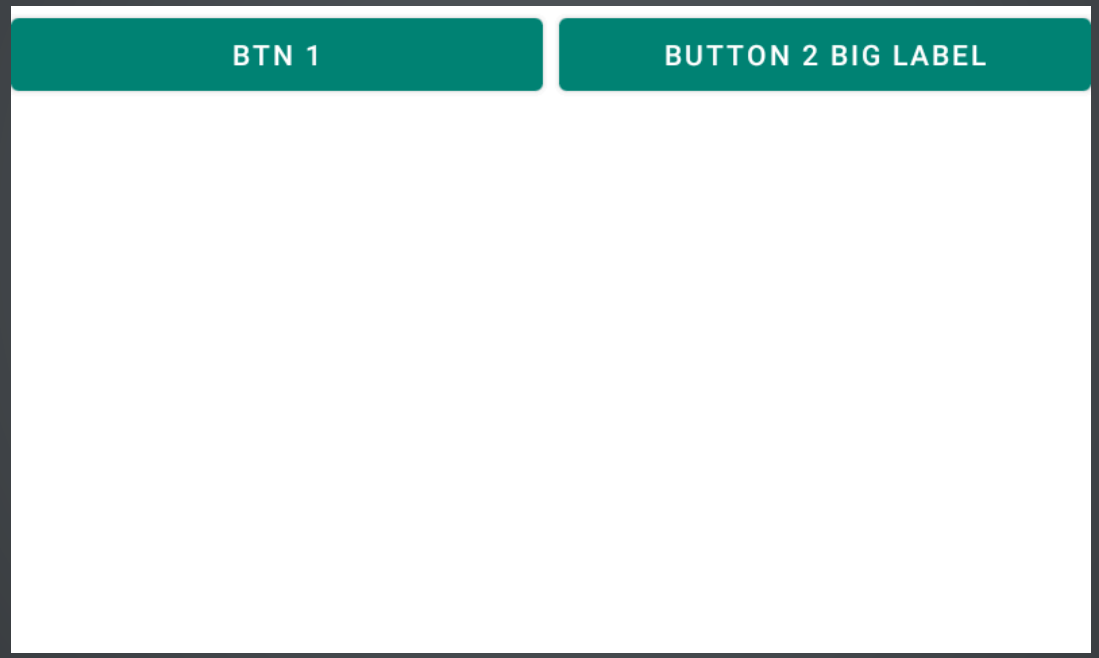
Les 3 boutons sont affichés les uns à côté des autres. Le bouton 1 est en bas, de largeur fixe : 120dp, celui du milieu prend la place disponible, et son texte est aligné en bas à droite. Le 3ième bouton est de largeur fixe : 120dp :



# Exemple de layouts

## LinearLayout Exercice 6

Les 2 boutons sont affichés l'un à côté de l'autre, et prennent exactement la moitié de la largeur de l'écran, avec 2 labels de taille très différente :



# Exemple de layouts

## Afficher un contenu très grand

Dans le cas de l'utilisation d'une listview, verticale, permettant d'ajouter beaucoup d'éléments sur notre écran, il est préférable d'englober notre LinearLayout dans une ScrollView, par exemple :

Copier

```
<?xml version="1.0" encoding="utf-8"?>
<ScrollView xmlns:android="http://schemas.android.com/apk/res/android"
            android:layout_width="match_parent"
            android:layout_height="match_parent">

    <LinearLayout
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:orientation="vertical">

        <Button
                android:layout_width="match_parent"
                android:layout_height="wrap_content"
                android:text="BTN" />

        <Button
                android:layout_width="match_parent"
```

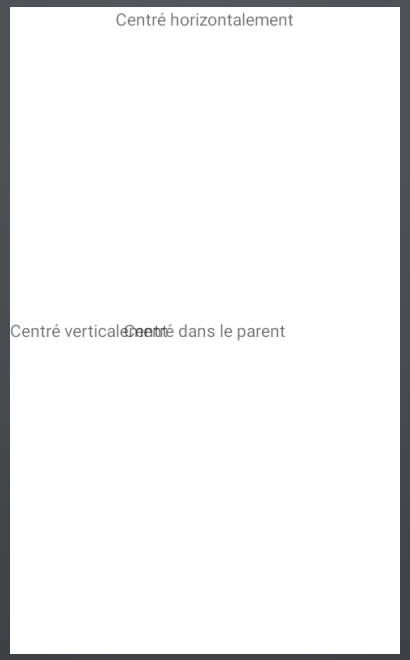
# RelativeLayout Exercice 1

# Exemple de layouts

Les 3 TextView sont alignés de la manière suivante :

- Centré horizontalement
- Centré verticalement
- Centré dans le parent

Que remarquez-vous d'inapproprié ?



# RelativeLayout Exercice 2

# Exemple de layouts

Nous allons positionner les TextView suivant les paroles de la chanson (dans le même ordre dans le XML) :

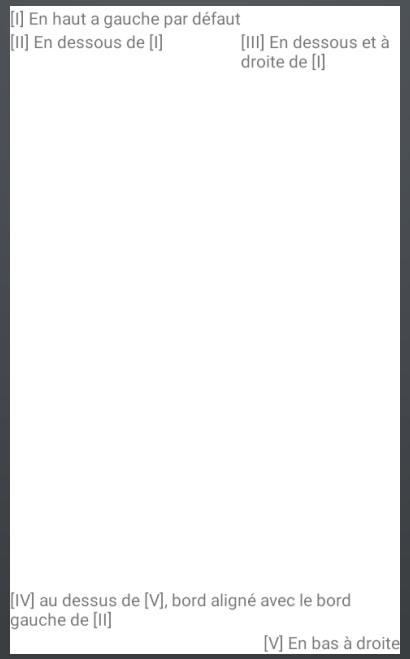
- En haut
- En bas
- A gauche
- A droite
- Ces soirées-là ! (centrée dans le parent)

# RelativeLayout Exercice 3

# Exemple de layouts

Nous allons maintenant positionner les TextView suivant les consignes suivantes :

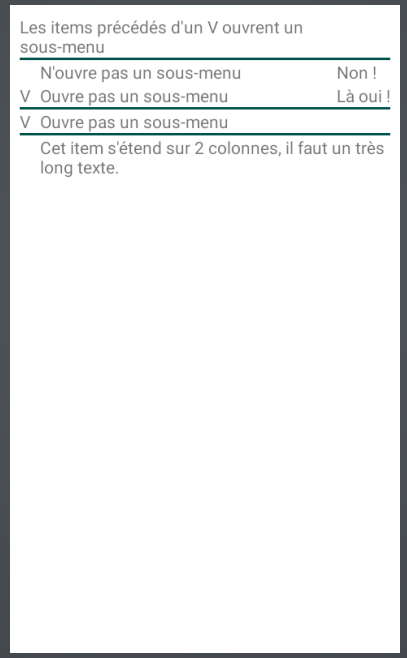
- [I] En haut à gauche par défaut
- [II] En dessous de [I]
- [III] En dessous et à droite de [I]
- [IIII] En dessous et à droite de [I]
- [IV] au-dessus de [V], bord aligné sur le bord gauche de [II]
- [V] En bas à droite



# Exemple de layouts

## TableLayout Exercice 1

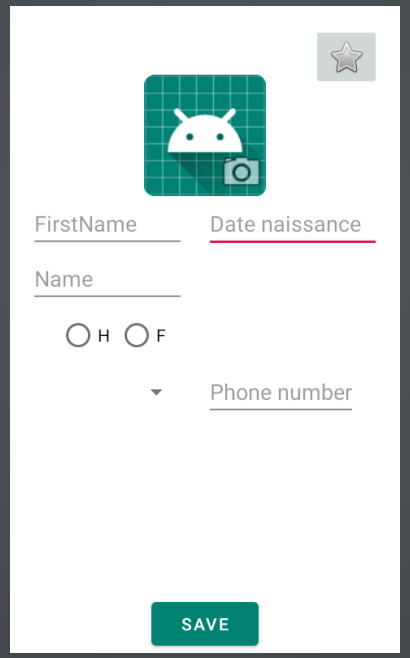
Le table layout nous permet de faire des alignements de type tableaux :



# Exemple de layouts

## ConstraintLayout Exercice 1

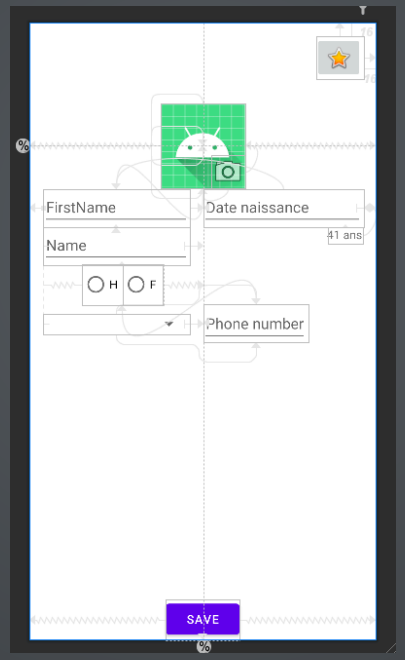
Le constraint layout est le dernier disponible. Il permet de réaliser des mises en forme complexes, tout en gardant une structure de layout plate (pas de layout imbriqué dans un autre layout) :



# Exemple de layouts

## ConstraintLayout Exercice 1 (détails)

Le détail des contraintes mises en places pour réaliser cette mise en forme de l'écran :



# Les ressources

## Les différentes ressources

Les ressources peuvent être de plusieurs types :

- anim : des animations que l'on peut utiliser pour animer des éléments de notre interface graphique.
- drawable : des images, bitmap ou vectorielles.
- layout : l'organisation de nos écrans.
- menu : la description des menus (des activités, des fragments, ...)
- mipmap : les icônes de notre application.
- raw : des données brutes (mp3 par exemple).
- xml : des données en XML.

# Les ressources

## Les différentes ressources

- values
  - arrays : des tableaux (de chaînes de caractères ou d'entiers)
  - colors : la définition de nos couleurs.
  - bools : des booléens.
  - dimens : des dimensions (tailles d'images, de texte, de marges, ...).
  - integers : des valeurs numériques.
  - plurials : des chaînes de caractères prenant en compte le pluriel.
  - strings : nos chaînes de caractères.
  - styles : pour afficher nos éléments.

# Les ressources

## Accéder aux valeurs en Java/Kotlin

Nous allons voir plus en détail l'utilisation d'une ressource. Dans un premier temps, pour les récupérer, en Java/Kotlin, nous utiliserons la syntaxe suivante :

```
[package.]R.type.nom

// Par exemple
R.string.app_name
```

Copier

# Les ressources

## Accéder aux valeurs en XML

En XML la syntaxe est similaire :

```
@[package:]type/nom

// Par exemple :
@string/app_name
```

Copier

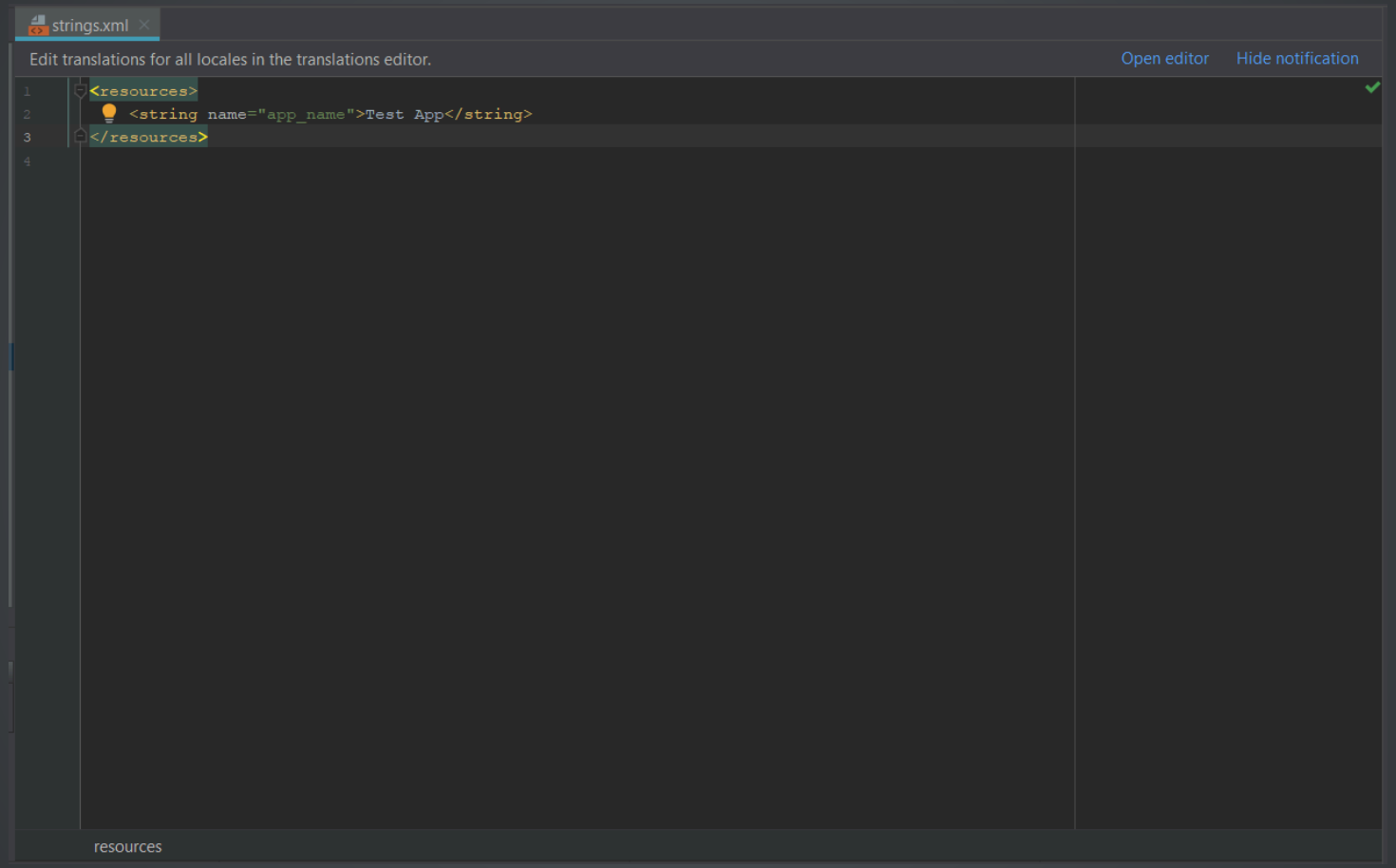
# Les ressources

## Internationalisation

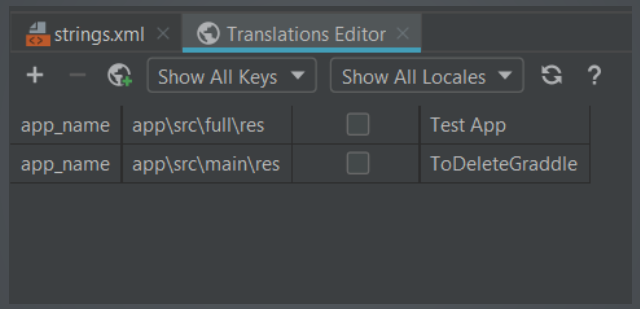
Voyons le fichier de ressource `strings.xml` et son utilisation pour internationaliser notre application.

- Ajouter une langue.
- Ajouter des clés.
- Tester dans l'éditeur.
- Tester sur l'émulateur.
- Tester sur le téléphone.

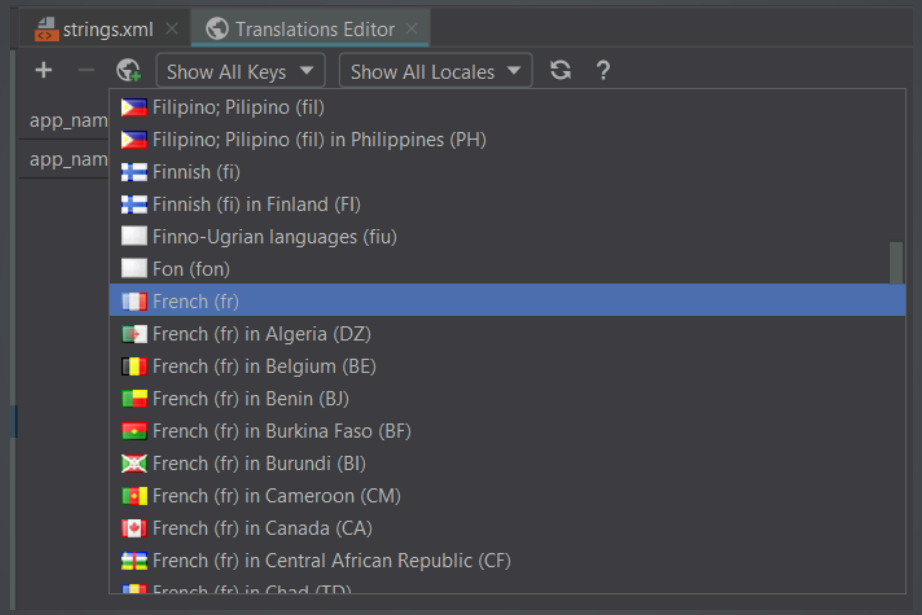
# Les ressources



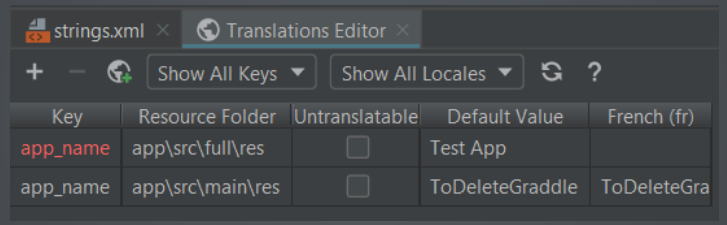
# Les ressources



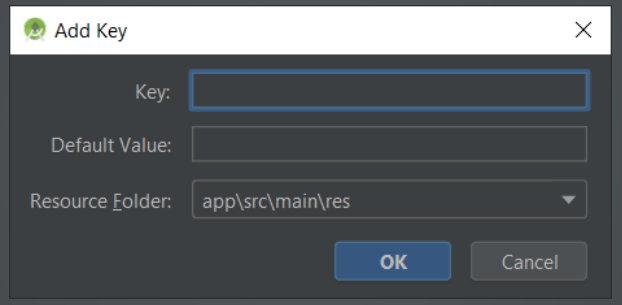
# Les ressources



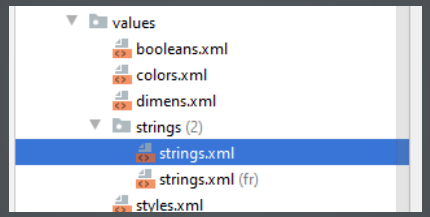
# Les ressources



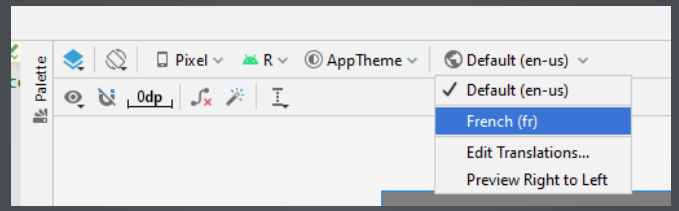
# Les ressources



# Les ressources



# Les ressources



# Les ressources

## Apostrophes et guillemets

Le format XML nous impose des contraintes par rapport aux caractères ' et ", regardons comment nous pouvons gérer ces cas :

```
<string name="good_example">"This'll work"</string>
<string name="good_example_2">This\'ll also work</string>
<string name="bad_example">This doesn't work</string>
<string name="bad_example_2">XML encodings don't work</string>
```

Copier

# Les ressources

## Formatage valeurs

Nous pouvons ajouter des variables dans notre texte, qui seront remplacées par les valeurs passées en paramètre, par exemple :

```
1 <string name="welcome_messages">Hello, %1$s! You have %2$d new messages.</string>
```

Copier

```
1 Resources res = getResources();
2 String text = String.format(res.getString(R.string.welcome_messages), username, mailCount);
```

Copier

# Les ressources

## Le pluriel

Les ressources permettent aussi de gérer le pluriel :

Copier

```xml
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <plurals name="numberOfSongsAvailable">
        <!- zero, one, two, few, many, other
        As a developer, you should always supply "one" and "other"
        strings. Your translators will know which strings are actually
        needed for their language. Always include %d in "one" because
        translators will need to use %d for languages where "one"
        doesn't mean 1 (as explained above).
        <item quantity="one">%d song found.</item>
        <item quantity="other">%d songs found.</item>
    </plurals>
</resources>
```

# Les ressources

## Exemple d'utilisation du pluriel

Il s'agit d'une valeur dynamique, nous utilisons donc du code pour définir la valeur de la chaîne de caractère, par exemple :

```
1 int count = getNumberOfsongsAvailable();
2 Resources res = getResources();
3 String songsFound = res.getQuantityString(R.plurals.numberOfSongsAvailable, count, count);
```

Copier

# Les ressources

## Mise en forme

Nous avons à disposition quelques outils de mise en forme de notre texte :

```
<b> for bold text.
<i> for italic text.
<u> for underline text.
```

Copier

Par exemple :

```
<string name="welcome">Welcome to <b>Android</b>!</string>
```

Copier

# Les ressources

## Tableaux

Nous pouvons aussi définir des tableaux, par exemple :

```xml
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string-array name="planets_array">
        <item>Mercury</item>
        <item>Venus</item>
        <item>Earth</item>
        <item>Mars</item>
    </string-array>
</resources>
```

Copier

Que l'on utilisera :

```
val planets = resources.getStringArray(R.array.planets_array)
```

Copier

# Les ressources

## Dimens

Le fichier dimens.xml contiendra des définitions de taille, par exemple la taille d'une image, d'une marge, d'un texte, ....
Nous avons plusieurs types de dimensions disponibles :

- px : mesure en pixel, à bannir
- dp : mesure qui s'adapte à la densité de pixel de l'écran.
- sp : mesure utilisée pour les textes, car elle prend en compte le paramétrage de l'utilisateur relatif à la taille de la police de caractère.

# Les ressources

## Colors

Nous pouvons définir toutes nos couleurs dans ce fichier. Par exemple :

Copier

```xml
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <color name="colorPrimary">#6200EE</color>
    <color name="colorPrimaryDark">#3700B3</color>
    <color name="colorAccent">#03DAC5</color>
</resources>
```

# Les ressources

## Styles

Les styles nous permettent de regrouper des caractéristiques d'affichage d'éléments (des titres, des entêtes, ...). Par exemple :

Copier

```
<style name="Counter" parent="AppTheme">
    <item name="android:textColor">@color/l4e_counter_text</item>
    <item name="android:background">@drawable/rounded_corner</item>
    <item name="android:padding">@dimen/l4e_counter_padding</item>
</style>

<style name="DialogTitle" parent="AppTheme">
    <item name="android:padding">@dimen/l4e_dialog_title_text_padding</item>
    <item name="android:textSize">@dimen/l4e_dialog_title_text_size</item>
    <item name="android:textStyle">bold</item>
</style>
```

# La gestion événementielle

## Ajout d'un bouton

Nous allons ajouter un bouton à notre interface, via l'éditeur graphique, ou via l'éditeur texte.

# La gestion événementielle

Afin de réagir aux click de n'utilisateur, nous allons brancher un View.OnClickListener, ou un View.OnLongClickListener.
Pour cela, nous avons 4 options que nous allons détailler :

- Par héritage
- Classe anonyme
- Attribut
- XML

Voyons tout cela en détail.

# La gestion événementielle

## Par héritage

Suivons les étapes suivantes :

- Faisons étendre notre MainActivity de View.OnClickListener.
- Implémentons la méthode en utilisant l'ampoule rouge proposée par l'éditeur.
- Remplissons la méthode onClick générée.
- Associons notre listener à un bouton

Contenu de la méthode onClick :

```
when(v?.id){
    R.id.activity_main_button_test -> Log.d(TAG, "onCreate click")
}
```

Copier

Association du listener avec notre classe.

```
findViewById<Button>(R.id.activity_main_button_test).setOnClickListener(this@MainActivity)
```

Copier

# La gestion événementielle

## Par attribut

Déclarons un attribut de classe :

```
private val btnListener: View.OnClickListener = View.OnClickListener {
    when (it?.id) {
        R.id.activity_main_button_test -> Log.d(TAG, "onCreate click")
    }
}
```

Copier

Attribuons notre listener à notre bouton :

```
findViewById<Button>(R.id.activity_main_button_test).setOnClickListener(btnListener)
```

Copier

# La gestion événementielle

## Par lambda

C'est la méthode la plus courante :

```
1 findViewById<Button>(R.id.activity_main_button_test).setOnClickListener(){ Log.d(TAG, "onCreate click") }
```

Copier

# La gestion événementielle

## Par XML

Dans le layout de notre activity, sur le bouton, ajoutons l'attribut onClick :

```
1 android:onClick="handleClick"
```

Copier

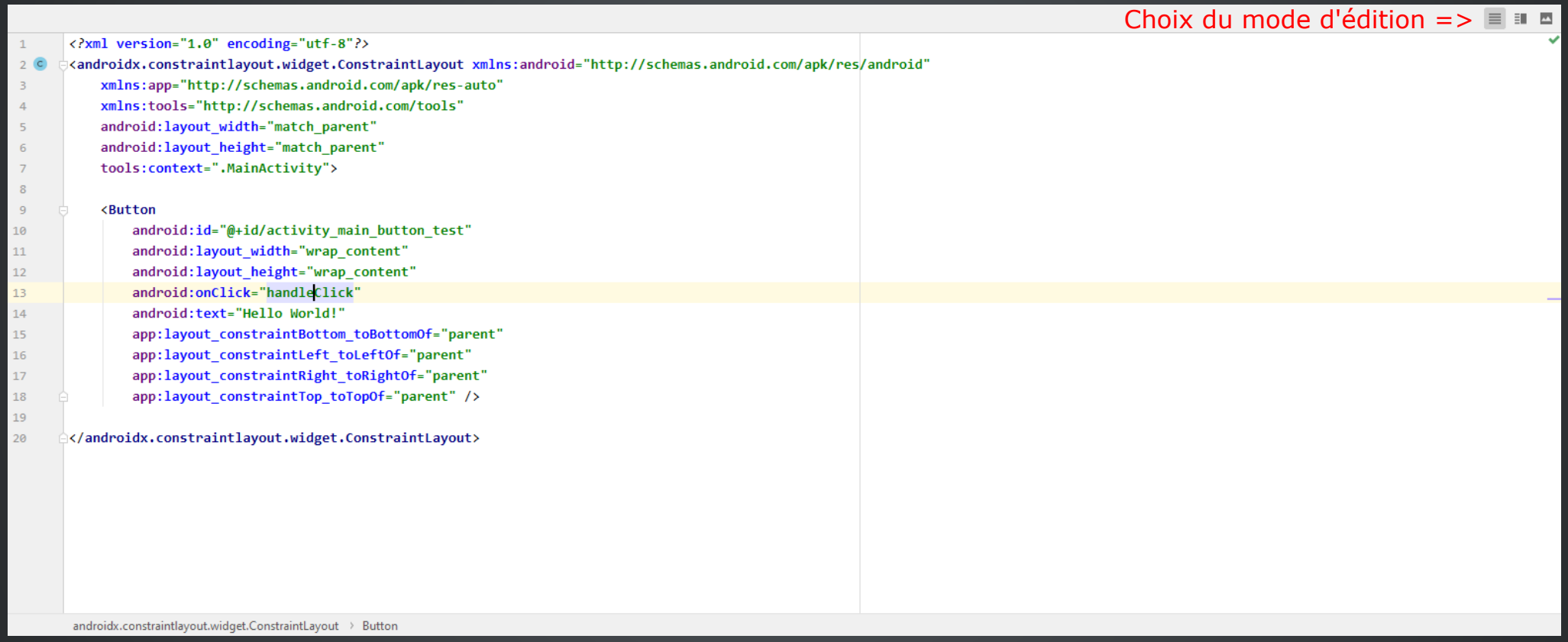
Créons la méthode en utilisant l'aide de l'éditeur, il nous reste plus qu'à implémenter la méthode.

# Les éléments graphiques

Plusieurs éléments graphiques de base sont à notre disposition, nous allons en voir plusieurs, et la correspondance entre la partie graphique et la partie texte.

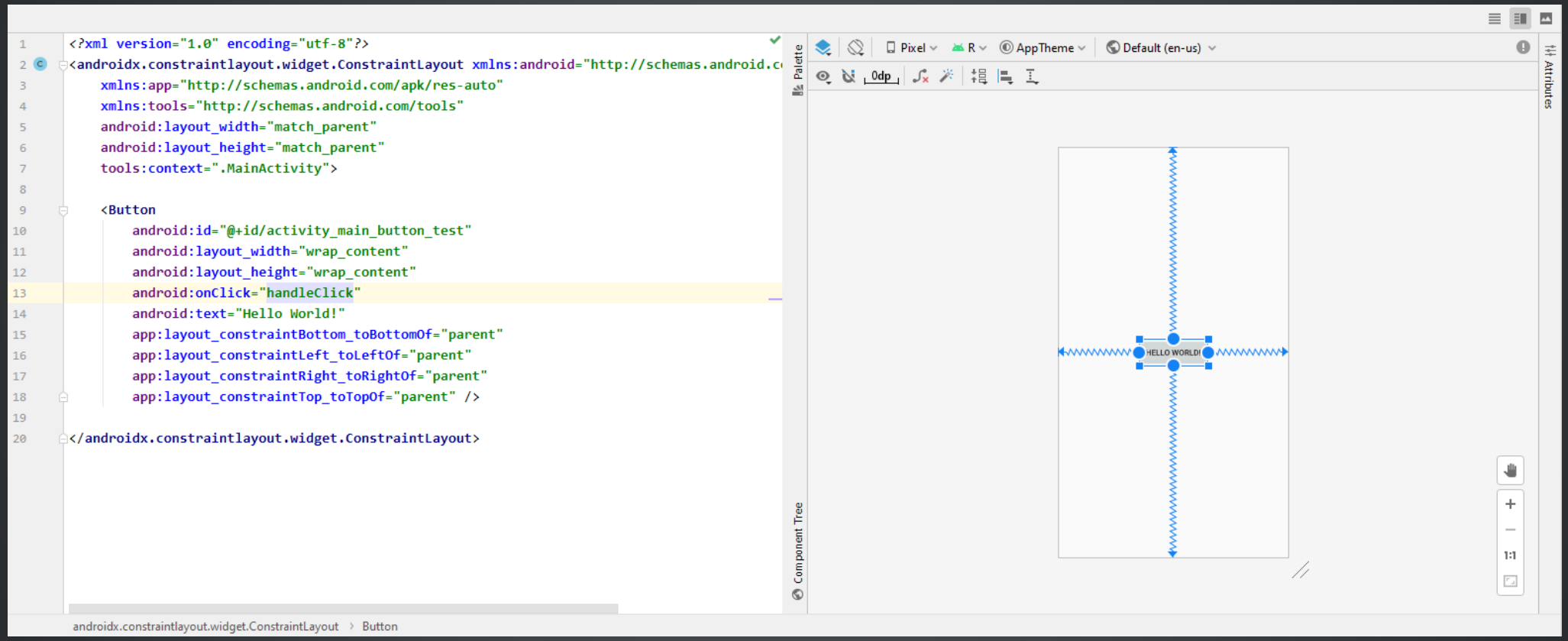
# Les éléments graphiques

## Éditeur : Code



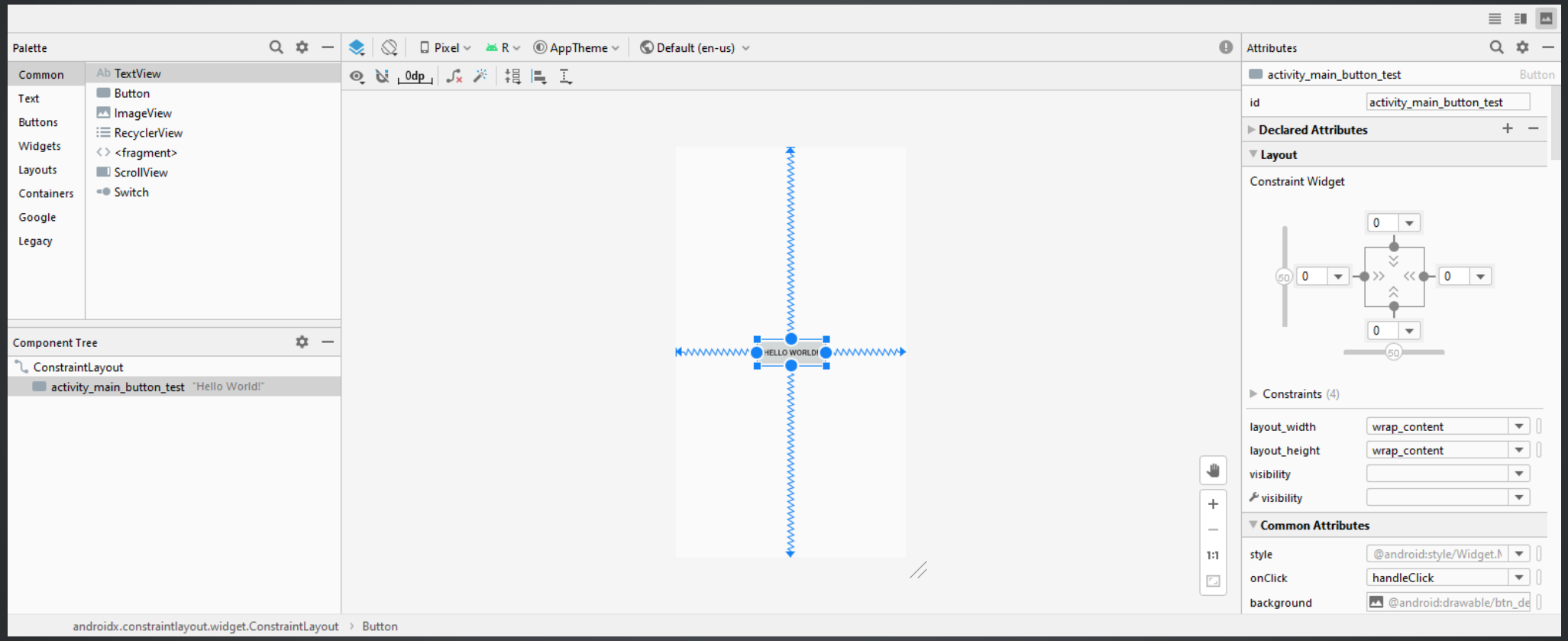
# Les éléments graphiques

## Éditeur : Split



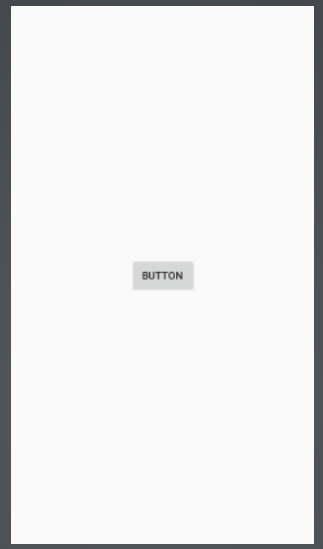
# Les éléments graphiques

## Éditeur : Design



# Les éléments graphiques

## Button

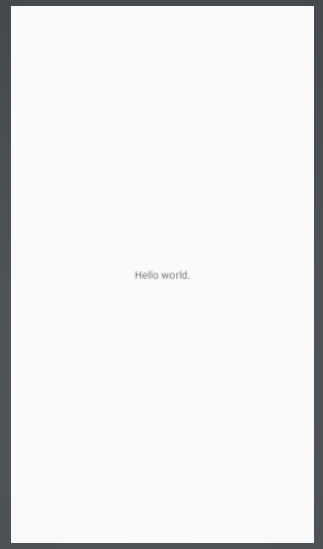



Copier

```
<Button
    android:id="@+id/activity_main_button_test"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:onClick="handleClick"
    android:text="Button"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintLeft_toLeftOf="parent"
    app:layout_constraintRight_toRightOf="parent"
    app:layout_constraintTop_toTopOf="parent" />
```

# Les éléments graphiques

## Texte affiché

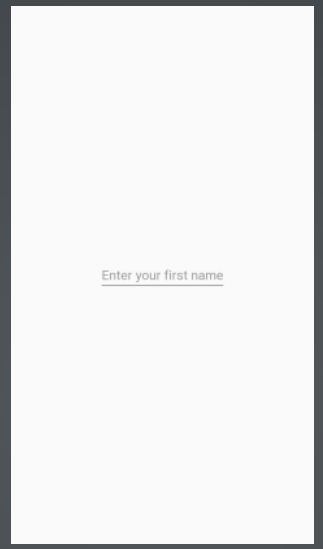



Copier

```
<TextView
    android:id="@+id/activity_main_textView_test"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Hello world."
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintLeft_toLeftOf="parent"
    app:layout_constraintRight_toRightOf="parent"
    app:layout_constraintTop_toTopOf="parent" />
```

# Les éléments graphiques

## Champ texte




```
<EditText
    android:id="@+id/activity_main_editText_test"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:hint="Enter your first name"
    android:inputType="textPersonName"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintLeft_toLeftOf="parent"
    app:layout_constraintRight_toRightOf="parent"
    app:layout_constraintTop_toTopOf="parent" />
```

Copier

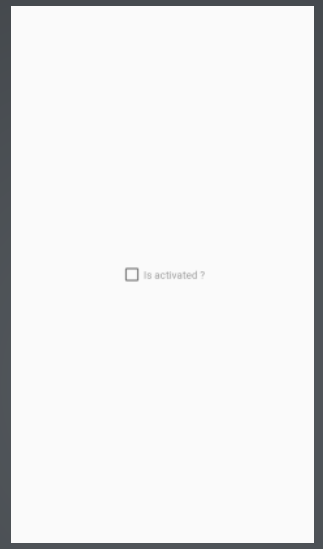
# Les éléments graphiques

## Champ texte (Material design)

Nous pouvons aussi utiliser la version moderne du champ texte, avec le LiveTemplate `and_material_design_edittext`.
Quelles différences notez vous ?

# Les éléments graphiques

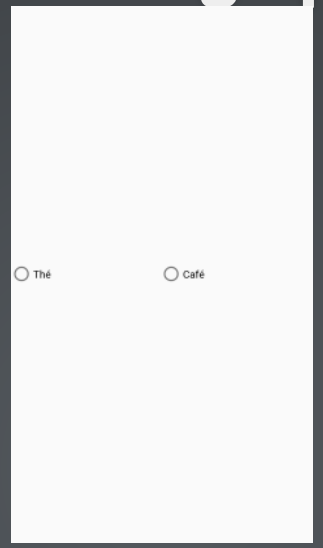
## Case à cocher




```
<CheckBox
    android:id="@+id/activity_main_checkBox_test"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:hint="Is activated ?"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintLeft_toLeftOf="parent"
    app:layout_constraintRight_toRightOf="parent"
    app:layout_constraintTop_toTopOf="parent" />
```

Copier

# Boutons de choix




Copier

```
<RadioGroup
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:orientation="horizontal"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintLeft_toLeftOf="parent"
    app:layout_constraintRight_toRightOf="parent"
    app:layout_constraintTop_toTopOf="parent">

    <RadioButton
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_weight="1"
        android:text="Thé" />

    <RadioButton
        android:layout_width="wrap_content"
```

# Les éléments graphiques

## Liste de choix


Dans la méthode onCreate de notre Activity utilisons le LiveTemplate : android_spinner_string_array.

# Les éléments graphiques

## Liste de choix encore plus rapide.

Gardons le `string-array` contenant la liste des planètes, retirons tout le code pour gérer le `Spinner` et ajoutons l'attribut `android:entries="@array/planets_array"`.

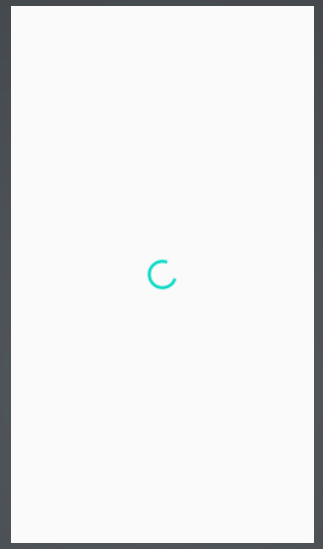
```
<Spinner
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:entries="@array/planets_array" />
```

Copier

Régulièrement des facilités de développement sont disponibles dans le SDK ou l'IDE.

# Les éléments graphiques

## Barre de progression

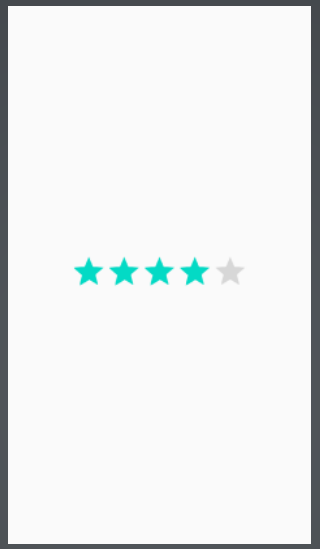


```
<ProgressBar
    android:id="@+id/activity_main_progressBar_test"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:indeterminate="true"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintLeft_toLeftOf="parent"
    app:layout_constraintRight_toRightOf="parent"
    app:layout_constraintTop_toTopOf="parent" />
```

Copier

# Les éléments graphiques

## Notation (étoiles)

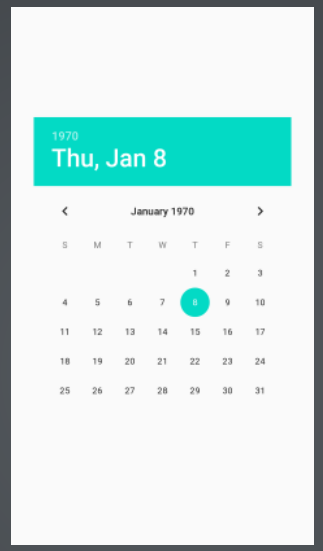


```
<RatingBar
    android:id="@+id/activity_main_ratingBar_test"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:numStars="5"
    android:rating="4"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintLeft_toLeftOf="parent"
    app:layout_constraintRight_toRightOf="parent"
    app:layout_constraintTop_toTopOf="parent" />
```

Copier

# Les éléments graphiques

## Date

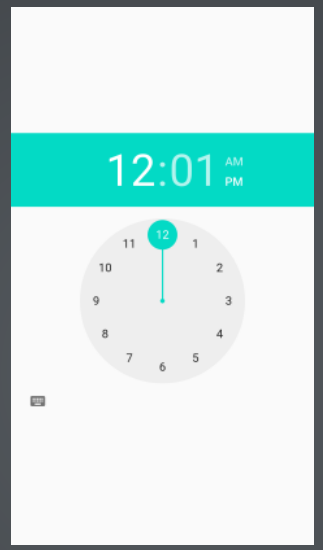



```xml
<DatePicker
    android:id="@+id/activity_main_datePicker_test"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintLeft_toLeftOf="parent"
    app:layout_constraintRight_toRightOf="parent"
    app:layout_constraintTop_toTopOf="parent" />
```

Copier

# Les éléments graphiques

## Time

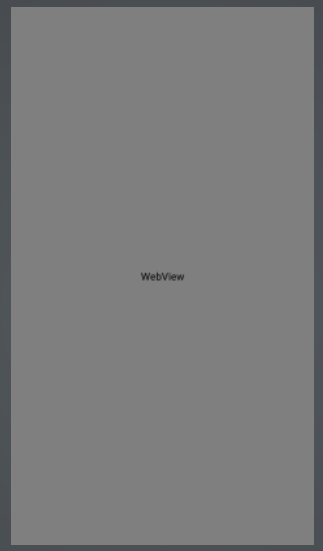



```
<TimePicker
    android:id="@+id/activity_main_timePicker_test"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintLeft_toLeftOf="parent"
    app:layout_constraintRight_toRightOf="parent"
app:layout_constraintTop_toTopOf="parent" />
```

Copier

# Les éléments graphiques

## Vue Web




```
<WebView
        android:id="@+id/activity_main_webView_test"
        android:layout_width="match_parent"
        android:layout_height="match_parent" />
```

Copier

# Les éléments graphiques

## Vue Web (suite)

Pour fonctionner, nous allons préciser l'URL que nous souhaitons afficher dans la méthode onCreate de notre Activity :

```
1 var webView = findViewById<WebView>(R.id.activity_main_webView_test)
2 webView.loadUrl("http://light4events.fr")
```

Copier

Quel est le problème qui empêche la page de s'afficher ? Comment résoudre le problème ?
Que remarque-t-on quand on clique sur un lien ? Comment résoudre le problème ?

# Les éléments graphiques

## Material design

Nous avons utilisé une forme de l'`EditText` en Material Design, nous avons d'autres LiveTemplate à disposition, en voici la liste exaustive :

- and_material_design_divider
- and_material_design_edittext
- and_material_design_progress
- and_material_design_slider
- and_material_design_switch

# Le modèle de composants

- Les Intents et leur gestion par l'activité.
- La relation activité mère-fille.
- Les fragments, les services, les IntentServices.

# Les Intents et leur gestion par l'activité

Les différents modules applicatifs ne sont pas directement instanciés par le développeur, un bus de messages permet au système de choisir le composant à monter en mémoire.
Les messages sont de type Intent.
Les valeurs envoyées dans l'Intent sont contenues dans un Bundle (optionnel).
Les intentions décrivent quelle est l'opération qui devra être effectuée.
Plusieurs composants applicatifs peuvent répondre à un même Intent, dans ce cas, l'utilisateur aura alors le choix.
Nous allons tester plusieurs Intent simple pour mettre en œuvre ces principes.

# Les Intents et leur gestion par l'activité

## LiveTemplates Intent

Nous avons à notre disposition plusieurs intents, entre autres :

- and_intent_mail : pour envoyer un email avec les champs préremplis.
- and_intent_map : pour ouvrir l'application Maps affichant une coordonnée.
- and_intent_phone_call : pour lancer le numéroteur téléphonique.
- and_intent_sms : pour envoyer un SMS (pour lancer l'application de SMS, prête à envoyer le SMS).
- and_intent_web : pour lancer un navigateur web pour afficher une URL.

# Les Intents et leur gestion par l'activité

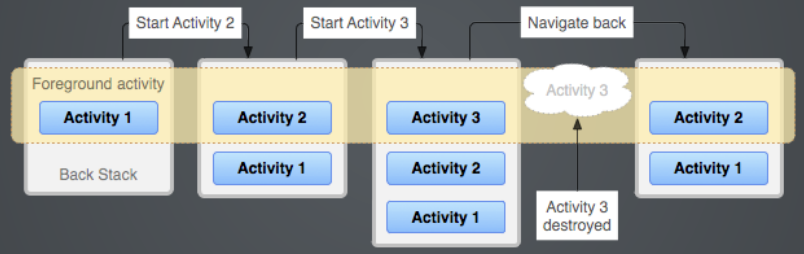
De la même manière, il est possible de déclarer dans le manifest des capacités de nos Activity à gérer des Intents. Par défaut l'Activity principale (MainActivity en général) gère l'appel par le Launcher. Dans le AndroidManifest.xml, nous voyons les lignes suivantes :

Copier

```
<intent-filter>
    <action android:name="android.intent.action.MAIN" />
    <category android:name="android.intent.category.LAUNCHER" />
</intent-filter>
```

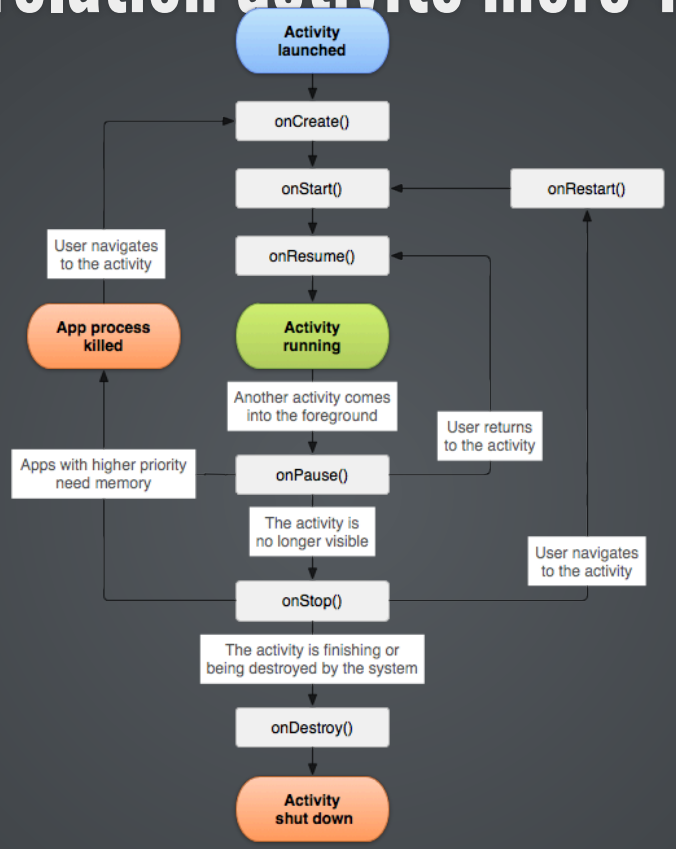
# La relation activité mère-fille

Une application contient souvent de multiples activités.
Lorsqu'on en lance une, celle précédemment ouverte se stoppe (pause). La nouvelle activité se retrouve au-dessus de la pile de type « last in, first out » c'est-à-dire « dernier arrivé, premier sorti ».




Référence.

# La relation activité mère-fille




Référence.

# La relation activité mère-fille

## Mise en oeuvre

Pour visualiser les principales étapes du cycle de vie de notre activity, utilisons le LiveTemplate : and_lifecycle_activity, dans la classe, mais en dehors d'une méthode.
Au besoin, nous utilisons le LiveTemplate : and_tag pour définir la constante TAG au début de notre classe.
Démarrons notre application, regardons les logs et tournons l'écran.
Que remarquons-nous ?

# La relation activité mère-fille

## Communication inter Activity

Nous allons maintenant voir comment lancer une nouvelle Activity, et lui envoyer des informations. Pour cela, nous allons passer par plusieurs étapes :

- Ajoutons un champ texte à notre interface.
- Récupérons la valeur du texte saisie.
- Envoyons-la à notre Activity2.
- Affichons la dans l'Activity2.

# La relation activité mère-fille

Nous déclarons le champ texte et le bouton dans notre layout :

Copier

```
<EditText
        android:id="@+id/activity_main_editText_value"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent" />

<Button
        android:id="@+id/activity_main_button_send"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Envoyer"
        app:layout_constraintTop_toBottomOf="@id/activity_main_editText_value" />
```

# La relation activité mère-fille

Créons notre Activity2 via le click droit/New/Activity/Empty Activity.
Lançons cette activité lors du click bouton, avec en paramètre le texte de l'EditText : Nous déclarons une variable d'instance :

```
lateinit var etValue: EditText
```

Copier

Puis ajoutons le code suivant à notre méthode onCreate :

```
etValue = findViewById<EditText>(R.id.activity_main_editText_value)
findViewById<Button>(R.id.activity_main_button_send).setOnClickListener() {
    var intent = Intent(this@MainActivity, Main2Activity::class.java)
    intent.putExtra(INTENT_PARAMS_EXTRA_VALUE, etValue.text.toString())
    startActivity(intent)
}
```

Copier

Je m'aide du LiveTemplate and_intent_activity pour écrire le code.

# La relation activité mère-fille

Du côté de Main2Activity, je récupère la valeur envoyée, dans la méthode onCreate, avec le code :

```
1 var value = savedInstanceState?.getString(INTENT_PARAMS_EXTRA_VALUE) ?: intent.getStringExtra(INTENT_PARAMS_EXTRA_VALUE)
```

Copier

Je m'aide du LiveTemplate and_intent_onCreate pour écrire le code.
J'ajoute le même champ et bouton que la première activité, et j'affecte la valeur reçue à l'EditText :

```
1 findViewById<EditText>(R.id.activity_main2_editText_value).setText(value)
```

Copier

# La relation activité mère-fille

Une activity peut aussi retourner des valeurs, par exemple demandons au système de sélectionner un contact de notre répertoire téléphonique.
Pour cela, sur un click de bouton, par exemple, je vais lancer le code généré par le LiveTemplate and_intent_select_contact et suivre les instructions.

# La relation activité mère-fille

## Retour de valeur

Voyons ensemble comment ce mécanisme fonctionne, je souhaite demander une information à une Activity, qui se lancera et me renverra une valeur (dans notre cas, une chaîne de caractère, mais cela peut être n'importe quel type d'information : un contact de notre répertoire, une image, ...).
Pour cela 3 étapes sont nécessaires :

- Activity1 : lancer l'Activity2 avec la méthode : startActivityForResult
- Activity2 : effectuer notre traitement (dans notre cas, laisser l'utilisateur saisir un texte et valider avec un bouton).
- Activity1 : récupérer la valeur envoyée par l'Activity2.

# La relation activité mère-fille

Dans l'Activity, déclarons la constante :

```
1 const val REQUEST_CODE_GET_TEXT = 4567 // Arbitrary value
```

Copier

Dans l'Activity, toujours, lançons l'appel à l'Activity2, en attendant une réponse :

```
1 var intent = Intent(this@MainActivity, Main2Activity::class.java)
2 startActivityForResult(intent, REQUEST_CODE_GET_TEXT)
```

Copier

# La relation activité mère-fille

Dans l'Activity2, je récupère la valeur saisie par l'utilisateur et je la renvoie. Pour cela, je m'aide du LiveTemplate : and_intent_returnValue. Nous pouvons maintenant récupérer la valeur renvoyée par l'Activity2, dans l'Activity1.
Pour cela, nous utilisons le LiveTemplate and_onActivityResult en dehors d'une méthode dans la classe MainActivity.

# La relation activité mère-fille

Pour améliorer l'expérience utilisateur, nous pouvons préciser quel type d'informations l'utilisateur doit saisir afin de présenter un clavier adapté.
Les détails : Specify the input method type.
Par exemple pour un numéro de téléphone :

Copier

```
<EditText
    android:id="@+id/phone"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:hint="@string/phone_hint"
    android:inputType="phone" />
```

# La relation activité mère-fille

## Gestion du bouton OK

Nous pouvons aussi modifier le bouton OK/Valider du clavier pour refléter l'action à réaliser et déclencher le code directement depuis l'action sur le clavier, au lieu de passer par un bouton. Par exemple pour notre champ téléphone :

```
android:hint="@string/phone_hint"
android:inputType="phone"
android:imeOptions="actionSend"
```

Copier

Dans notre Activity, nous gérons le click sur le bouton "Envoyer" du clavier avec :

```
etValue.setOnEditorActionListener(){v, actionId, event ->
    if(actionId == EditorInfo.IME_ACTION_SEND){
            Log.d(TAG, "Action send to be handled.")
        true
    } else {
        false
    }
}
```

Copier

# Les fragments, les services, les IntentServices

## Fragments

Étudions les templates :

- click droit / New / Activity / Primary/Detail Flow
- click droit / New / Activity / Tabbed Activity

# Les fragments, les services, les IntentServices

## Intent service

Créons un IntentService en utilisant le template :
File / New / Service / Service (IntentService)

# Les fragments, les services, les IntentServices

## Service

Créons un Service en utilisant le template :
File / New / Service / Service

# Les fragments, les services, les IntentServices

Création d'une implémentation d'exemple (en Kotlin) : un service de génération de random.
Pour cela dans notre service, utilisons dans la classe de service un LiveTemplate pour écrire l'implémentation du service.

```
1 and_service_implementation
```

Copier

Pour appeler notre service (en Kotlin), nous utiliserons dans notre activity, un LiveTemplate, et nous suivrons à chaque fois les indications en TODO.

```
1 and_service_calling
```

Copier

# La gestion des données

- Les préférences.
- Les fichiers, le stockage interne et externe.
- SQLite / Room.
- Les Content Provider.

# Les préférences

## Stocker une préférence

Le framework nous met à disposition une possibilité de stocker des informations dans l'application : les SharedPreferences. Sous forme de clé/valeur, nous pouvons stocker de l'information, dans un fichier, lié à l'application. Pour cela, nous utilisons le code suivant :

```
1 val sharedPref = PreferenceManager.getDefaultSharedPreferences(applicationContext)
2 val editor = sharedPref.edit()
3 editor.putString(SHARED_PREFS_VALUE, "Value")
4 editor.commit()
```

Copier

Ou plus rapidement le LiveTemplate : and_SharedPreferences_save.
Nous n'oublions pas d'ajouter au gradle, la dépendance :
`implementation 'androidx.preference:preference-ktx:1.2.1'`.

# Les préférences

## Charger une préférence

Pour récupérer la valeur stockée, nous utilisons :

```
1 val sharedPref = PreferenceManager.getDefaultSharedPreferences(applicationContext)
2 val defaultValue = resources.getInteger(R.string.value_default)
3 sharedPref.getInt(SHARED_PREFS_VALUE, defaultValue)
```

Copier

Ou plus rapidement le LiveTemplate : and_SharedPreferences_load.
Nous pouvons bien entendu mutualiser l'objet `sharedPref`.

La version plus moderne consiste à utiliser un datastore.

# Les fichiers

Copier

```
@file:JvmName("FileTools")

package com.example.myapplication

import android.content.Context
import java.io.File

fun readFile(context: Context, isExternal: Boolean, path: String, fileName: String): String {
    val currentFile = getFile(context, isExternal, path, fileName)

    // READ
    // val inputAsString = FileInputStream(currentFile).bufferedReader().use { it.readText() }

    var sb = StringBuilder()
    currentFile?.forEachLine { sb.append( "$it\n") }

    return sb.toString()
```

# Les fichiers

## Utilisation

Utilisons les méthodes que nous venons d'écrire :

```
var isExternal = false
var path = "trs"
var fileName = "test.txt"
var append = true

writeFile(this@MainActivity, isExternal, path, fileName, "AAAAA\n", append)

val stringContent = readFile(this@MainActivity, isExternal, path, fileName)
if (BuildConfig.DEBUG) {
    Log.d(TAG, "onCreate $stringContent")
}
```

Copier

# SQLite / Room

Android propose un système de base de donnée : SQLite, celui-ci étant un peu complexe à utiliser, plusieurs outils ont été mis en place pour gérer une base de donnée plus facilement (des ORM : Mapping Objet-Relationnel). L'ORM officiel sur Android est maintenant ROOM.
Voyons un peu plus en détail la mise en œuvre de cette base de donnée. Nous allons étudier 2 méthodologies différentes (utilisant 2 outils différents) :
1. En utilisant la documentation officielle : ROOM.
2. En utilisant des LivesTemplates (and_db_room_00_documentation) (En JAVA uniquement).

# SQLite / Room

## Aller plus loin

Les bases de données ne pouvant pas être appelées dans le thread graphique et pouvant évoluer dans le temps, il est recommandé d'utiliser les LiveData pour gérer l'affichage des données. Nous pouvons suivre le tutoriel suivant : Android Room with a View - Kotlin.

# Les Content Provider

## Définition

Le content provider permet de partager avec d'autres applications des données, de manière standardisée.
Plus de détails, ici.

# Les Content Provider

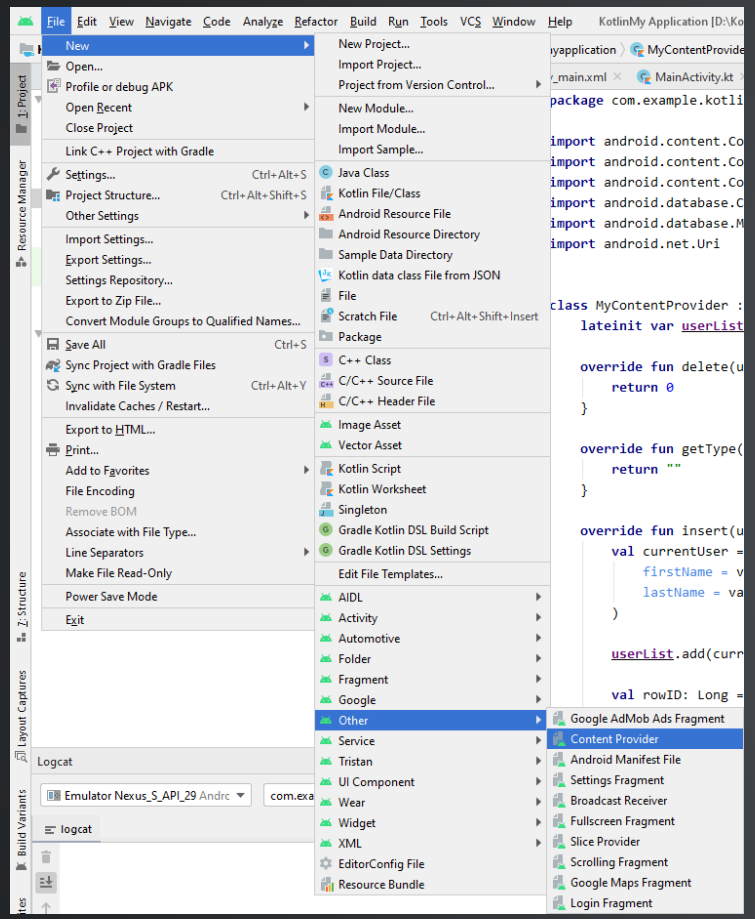
## Utilisation d'un content provider

Nous allons utiliser un content provider, par exemple le MediaStore. Créons un nouveau projet ContentProviderClient, et listons les sons disponibles sur notre téléphone :

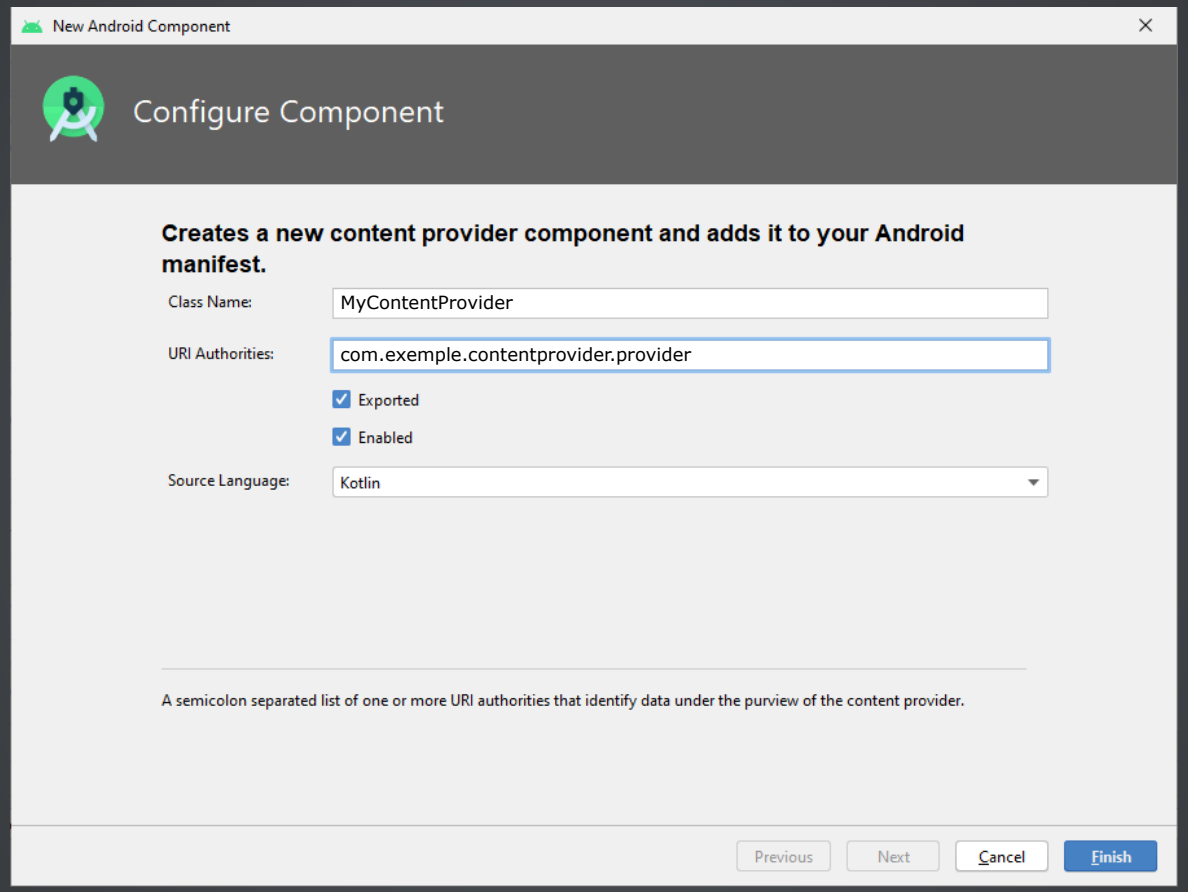
Copier

```
Uri media = MediaStore.Audio.Media.EXTERNAL_CONTENT_URI;
String[] projection = { MediaStore.Audio.Media._ID,           // 0
        MediaStore.Audio.Media.ARTIST,        // 1
        MediaStore.Audio.Media.TITLE,         // 2
        MediaStore.Audio.Media.ALBUM_ID,      // 3
        MediaStore.Audio.Media.ALBUM,         // 4
        MediaStore.Audio.Media.DATA,          // 5
        MediaStore.Audio.Media.DISPLAY_NAME,  // 6
        MediaStore.Audio.Media.DURATION };    // 7

String selection = MediaStore.Audio.Media.IS_MUSIC + " != 0";

Cursor cursor = getContentResolver().query(media,projection,selection,null,null);
while(cursor.moveToNext()){
    if(BuildConfig.DEBUG){
        Log.d(TAG, "onCreate " +
                        cursor.getString(0) + " " +
```

# Les Content Provider



# Les Content Provider



# Les Content Provider

## POJO : La classe User

Déclarons une nouvelle classe qui va contenir nos données, la classe `User` :

```
1 data class User(var firstName: String, var lastName: String)
```

Copier

# Les Content Provider

## Query

Pour commencer, nous allons simplement retourner une liste en dur via le code :

Copier

```
override fun query(
    uri: Uri, projection: Array<String>?, selection: String?,
    selectionArgs: Array<String>?, sortOrder: String?
): Cursor? {

    val cursor = MatrixCursor(arrayOf("name", "firstname"))

        cursor.newRow()
            .add("name", "SALAUN")
            .add("firstname", "Tristan")

        cursor.newRow()
            .add("name", "SNOW")
            .add("firstname", "John")

    return cursor
}
```

# Les Content Provider

## ArrayList

Afin d'avoir une certaine persistance et aller plus rapidement, dans l'implémentation du ContentProvider, nous allons utiliser une ArrayList.
Déclarons une variable de classe :

```
lateinit var userList: ArrayList<User>
```

Copier

Nous allons initialiser la liste dans la méthode onCreate :

```
override fun onCreate(): Boolean {
    // Init some values at the create
    userList = arrayListOf(User("Tristan", "SALAUN"))
    return true
}
```

Copier

Notre méthode query va donc prendre la forme :

```
override fun query(
    uri: Uri, projection: Array<String>?, selection: String?,
    selectionArgs: Array<String>?, sortOrder: String?
    ): Cursor? {

    val cursor = MatrixCursor(arrayOf("name", "firstname"))

    for (curentUser in this.userList) {
        cursor.newRow()
        .add("name", curentUser.lastName)
        .add("firstname", curentUser.firstName)
    }

    // cursor.newRow()
    // .add("name", "SALAUN")
    // .add("firstname", "Tristan")
```

Copier

# Les Content Provider

La méthode insert prendra quant à elle la forme :

Copier

```
override fun insert(uri: Uri, values: ContentValues?): Uri? {
    val currentUser = User(
        firstName = values?.getAsString(UserContract.MyDatas.KEY_COL_FIRSTNAME),
        lastName = values?.getAsString(UserContract.MyDatas.KEY_COL_NAME)
    )

    userList.add(currentUser)

    val rowID: Long = userList.size.toLong()

    val _uri =
    ContentUris.withAppendedId(UserContract.MyDatas.CONTENT_URI, rowID)
    context!!.contentResolver.notifyChange(_uri, null)
    return _uri
}
```

# Les Content Provider

## Partager notre Content Provider

L'objectif étant que notre Content provider soit utilisable par d'autres applications, il est d'usage de définir une classe Contrat, dans notre cas la classe UserContract :

Copier

```
import android.net.Uri
import android.provider.BaseColumns

class UserContract {

    companion object {
        // The Authority
        private const val AUTHORITY = "com.exemple.contentprovider.provider"

        // The path to the data… and explain
        private const val PATH_TO_DATA = "users" //Vous pouvez déclarer plusieurs paths (les paths utilisent les /)
    }

    interface MyDatas : BaseColumns {
        companion object {

            // The URI and explain, with example if you want
```

# Les Content Provider

## Création de ContentProviderClient

C'est la classe UserContract que nous utilisons aussi dans le projet client de notre ContentProvider : ContentProviderClient.
Créons le projet `ContentProviderClient`.

# Les Content Provider

## Lecture dans ContentProviderClient

Ouvrons le projet `ContentProviderClient`, et ajoutons le code pour interroger notre nouveau `ContentProvider` :

Copier

```
Cursor cursor = getContentResolver().query(Uri.parse("content://com.exemple.contentprovider.provider"), null, null, null, null);
if(cursor.moveToFirst()) {
    StringBuilder strBuild=new StringBuilder();
    while (!cursor.isAfterLast()) {
        strBuild.append("\n"+cursor.getString(0)+ "-"+ cursor.getString(1));
        cursor.moveToNext();
    }
    if(BuildConfig.DEBUG){
        Log.d(TAG, "onCreate " + strBuild);
    }
}
```

# Les Content Provider

## Insertion dans ContentProviderClient

Ajoutons maintenant un appel pour insérer une donnée dans le `ContentProvider` :

Copier

```
// Defines an object to contain the new values to insert
val newValues = ContentValues().apply {
    /*
     * Sets the values of each column and inserts the data. The arguments to the "put"
     * method are "column name" and "value".
     */
    put(UserContract.MyDatas.KEY_COL_FIRSTNAME, "John")
    put(UserContract.MyDatas.KEY_COL_NAME, "Doe")
}

val newUri = contentResolver.insert(
    UserContract.MyDatas.CONTENT_URI,   // The UserDictionary content URI
    newValues                           // The values to insert
)
```

# Les Content Provider

Pour aller un peu plus loin : Content provider basics.

# La gestion réseau

- Les infos de connectivité. Utiliser HTTP.
- Parser du JSON.
- Les accès aux Web Services : Volley, Retrofit.

# Connectivité et HTTP

Commençons par mettre en place une classe utilitaire : `NetworkTool` :

Copier

```
import android.content.Context
import android.net.ConnectivityManager
import android.net.NetworkCapabilities

import android.net.NetworkInfo
import android.os.Build

class NetworkTool {

    companion object {

        // NEED : <uses-permission android:name="android.permission.ACCESS_NETWORK_STATE"/>
        private fun isNetworkAvailable(context: Context): Boolean {
            val connectivityManager = context.getSystemService(Context.CONNECTIVITY_SERVICE) as ConnectivityManager
            if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.M) {
                val nw = connectivityManager.activeNetwork ?: return false
                val actNw = connectivityManager.getNetworkCapabilities(nw) ?: return false
```

# Connectivité et HTTP

## Puis notre receiver : `NetworkStateReceiver`

Copier

```
import android.content.BroadcastReceiver
import android.content.Context
import android.content.Intent
import android.net.ConnectivityManager
import android.net.NetworkInfo
import android.util.Log
import java.lang.Boolean

class NetworkStateReceiver: BroadcastReceiver() {

    companion object {
        private const val TAG = "NetworkStateReceiver"
    }

    // post event if there is no Internet connection
    override fun onReceive(context: Context?, intent: Intent) {
        Log.d(TAG, "onReceive $intent")
```

# Connectivité et HTTP

Il ne nous reste plus qu'à enregistrer le listenner : nous commençons par déclarer l'attribut de classe :

```
1 private val networkStateReceiver = NetworkStateReceiver()
```

Copier

Puis dans la méthode onCreate on enregistre le listenner :

```
1 registerReceiver(networkStateReceiver, IntentFilter(ConnectivityManager.CONNECTIVITY_ACTION))
```

Copier

Et on le désenregistre dans le onStop :

```
1 unregisterReceiver(networkStateReceiver)
```

Copier

Pour tester, on peut couper la connexion WiFi par exemple, et la réactiver.

# Parser du JSON

Nous allons utiliser la librairie GSON.
Nous allons suivre les étapes suivantes :

- Ajouter une référence à la librairie dans nos dépendances.
- Générer du JSON à partir d'objets.
- Parser du JSON pour instancier des objets.

Cet exemple est largement inspiré de : Android : Utiliser Gson pour faciliter l'utilisation du Json

# Parser du JSON

## Ajout de la dépendance.

Ajoutons une dépendance dans notre fichier build.gradle (Module: app) :

```
dependencies {
    ...
    // JSon parser
    implementation 'com.google.code.gson:gson:2.10.1'
}
```

Copier

# Parser du JSON

## Générer du JSON à partir d'un objet.

Créons l'objet Book :

```
data class Book(var id: String, var name: String, var author: String, var genre: String, var numpages: Int, var releaseDate: String, var cover: String)
```

Copier

Et remplissons une liste de Book :

```
var list = arrayListOf(
    Book(
        id = "01fsEF", name = "1984",
        author = "George Orwell",
        genre = "Fiction dystopique",
        numpages = 376,
        releaseDate = "1949",
        cover = "1984.png"
    ),
    Book(
        id = "3H1J0n",
        name = "Le Meilleur des mondes",
        author = "Aldous Huxley",
        genre = "Science-fiction",
        numpages = 285,
        releaseDate = "1932",
        cover = "meilleur-des-mondes.jpg"
```

Copier

# Parser du JSON

Nous pouvons maintenant transformer notre liste d'objets en JSON.

Copier

```
1 val gson = Gson()
2 val listType = object : TypeToken<ArrayList<Book?>?>() {}.type
3 val jsonResult: String = gson.toJson(list, listType)
4 Log.d(TAG, "onCreate jsonResult = $jsonResult")
```

# Parser du JSON

## Bonus

Une fonction pour faire un formatage "pretty print" :

```
1 private fun jsonToPrettyFormat(jsonString: String?): String? {
2     val json = Gson().fromJson(jsonString, JsonElement::class.java)
3     val gson = GsonBuilder()
4         .serializeNulls()
5         .disableHtmlEscaping()
6         .setPrettyPrinting()
7         .create()
8     return gson.toJson(json)
9 }
```

Copier

Que l'on appellera par exemple :

```
1 Log.d(TAG, "onCreate jsonResult = ${jsonToPrettyFormat(jsonResult)}")
```

Copier

# Parser du JSON

## JSON vers objet

Nous allons maintenant réaliser l'opération inverse, consistant à passer du JSON (format texte) vers des objets :

```
1 // Deserialization
2 val bookList2: ArrayList<Book> = Gson().fromJson(jsonResult, listType)
3 Log.d(TAG, "onCreate bookList2 = $bookList2")
```

Copier

# Parser du JSON

## Autres librairies

Nous avons vu l'usage de GSON, mais il existe d'autres librairies (liste non exhaustive) :

- Jackson
- klaxon
- Moshi
- kotlinx.serialization

Un article comparant les solutions (2019)
Un article comparant les solutions (2020)
Un article comparant les solutions (2022)

# Les accès aux Web Services

Il existe plusieurs façons de faire des appels à des web services, par exemple :

- HttpURLConnection (sans librairie).
- Volley (sur GitHub).
- Android Asynchronous Http Client.
- Retrofit.

La solution la plus populaire, semble être Retrofit, c'est celle-là que nous allons utiliser. Il est à noter que Ktor gagne en popularité.

# Les accès aux Web Services

## Application de news

Nous allons écrire une application nous permettant d'afficher des nouvelles, en provenance d'un site proposant une API gratuitement (pour les développeurs). Les étapes que nous allons suivre sont :

- Création du compte sur le site.
- Ajout de la librairie à notre projet.
- Écriture des fichiers POJO.
- Écriture de l'interface avec l'API.
- Instantiation du client.
- Lancement d'une requête.

La source du sujet est inspiré de : Live data, ViewModel, Retrofit Android Architecture Component.

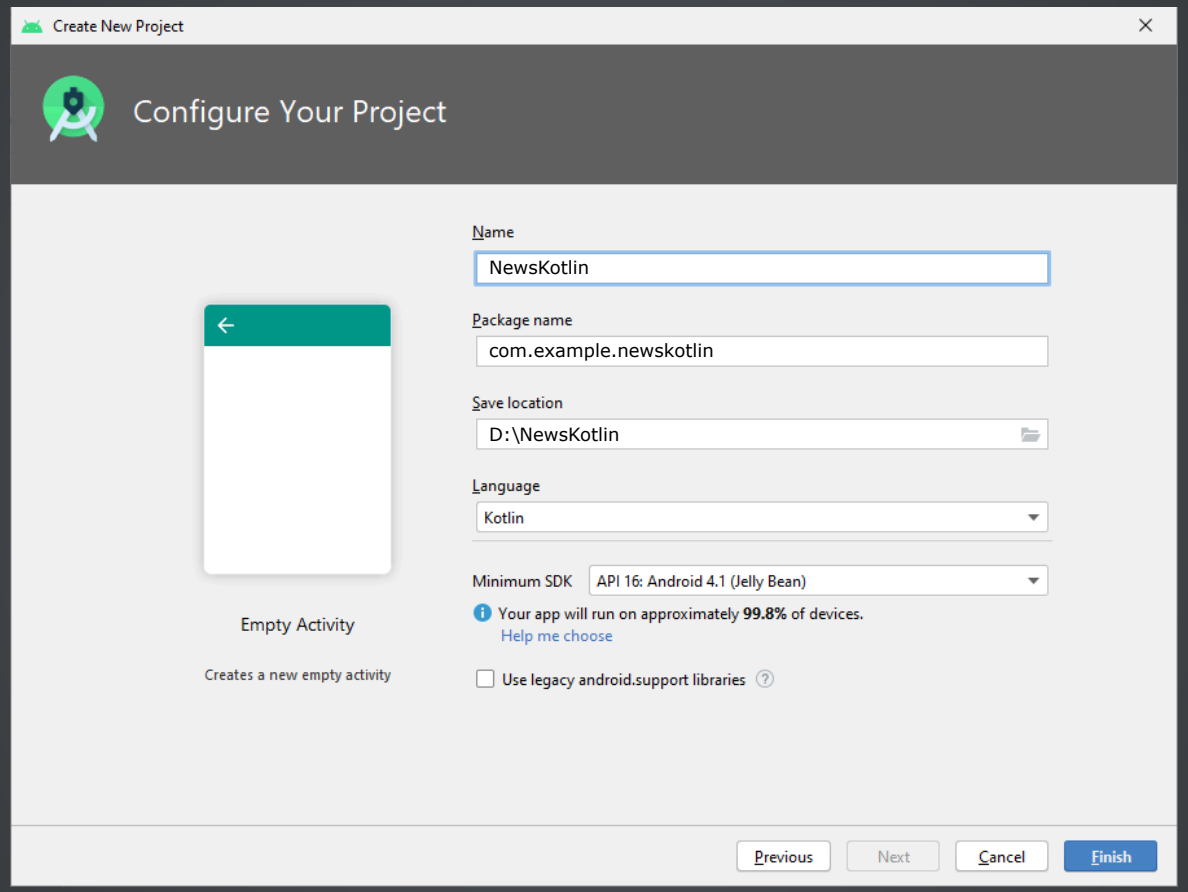
# Les accès aux Web Services

## Points techniques

Dans cette application, nous verrons les points suivants :

- LiveData.
- La librairie GSON.
- La librairie Retrofit.
- Le service en ligne JSON => POJO.
- Le service de news par API : News API.

# Les accès aux Web Services



# Les accès aux Web Services

## Ajout des librairies à notre projet

Copier

```
// JSon parsing
implementation 'com.google.code.gson:gson:2.10.1'

// Network calls
implementation 'com.squareup.retrofit2:retrofit:2.8.1'
implementation 'com.squareup.retrofit2:converter-gson:2.8.1'

// handle recyclerview
implementation "androidx.recyclerview:recyclerview:1.3.2"

// handle livedata life cycle in fragments
implementation "androidx.fragment:fragment-ktx:1.8.0"

// Handle images loading (choose one)
// implementation 'com.squareup.picasso:picasso:2.71828'
implementation 'com.github.bumptech.glide:glide:4.15.1'
```

# Les accès aux Web Services

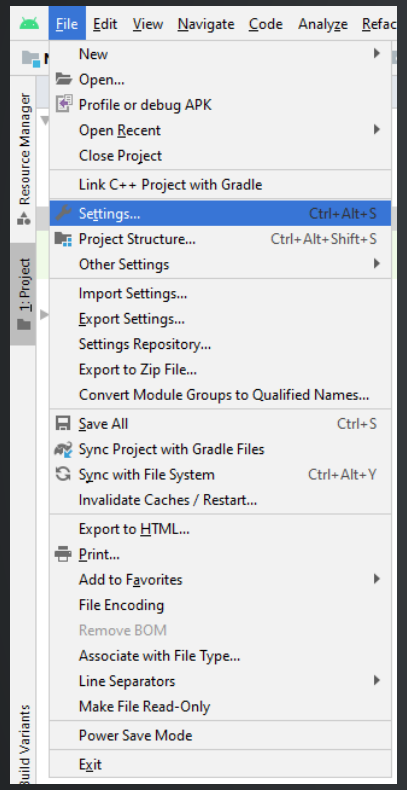
## Écriture des fichiers POJO

Pour écrire les fichiers qui correspondent à notre API de news, nous allons suivre les étapes suivantes :

- Nous allons installer le plugin JSON To Kotlin.
- Nous allons utiliser le JSON renvoyé par le site web.
- Nous allons le copier/coller dans le plugin JSON To Kotlin.
- Renseigner le champ Class Name: avec : NewsResponse.
- Il ne reste qu'à cliquer sur Generate.

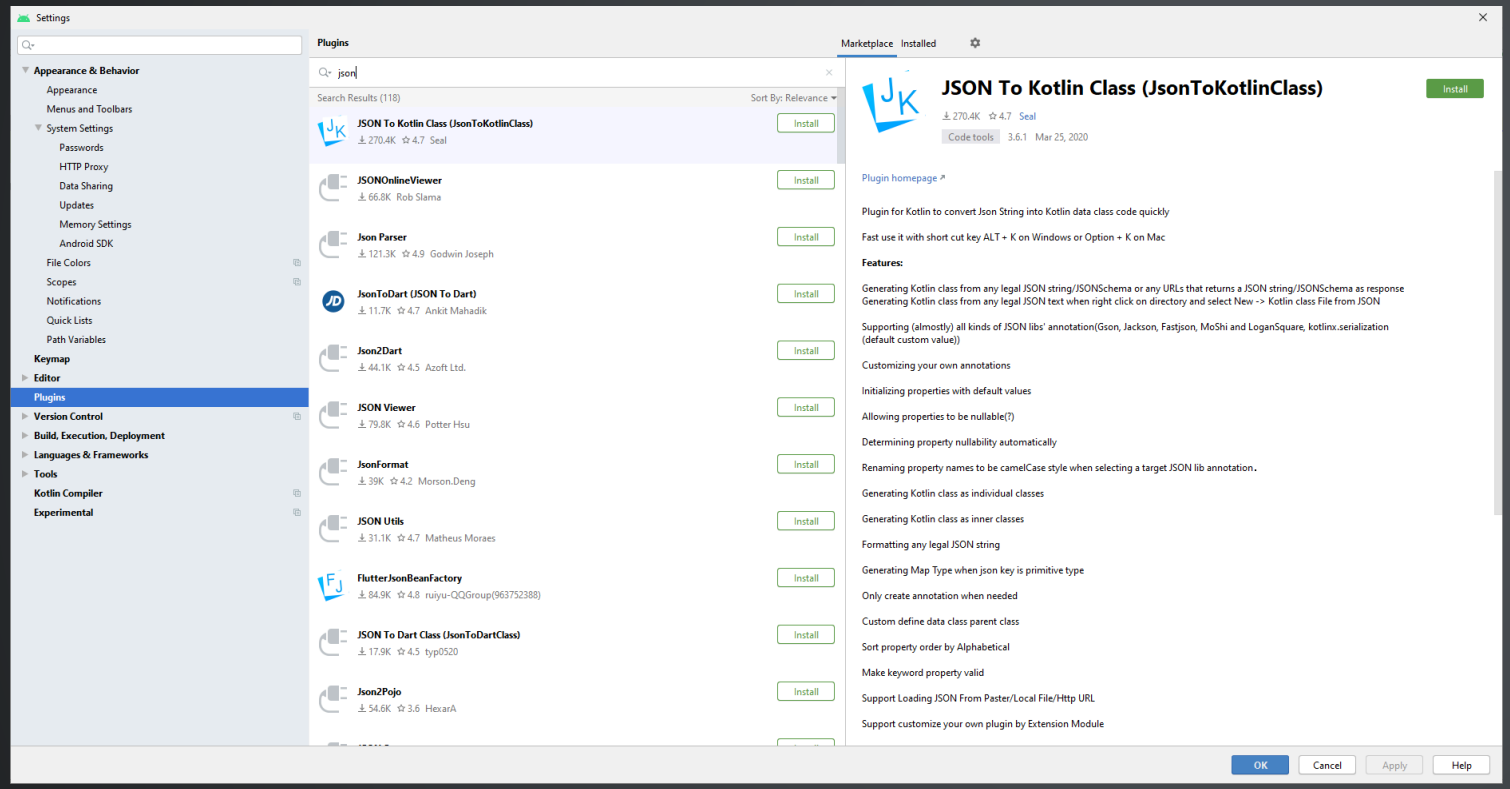
# Les accès aux Web Services

## Installons le plugin JSON to Kotlin



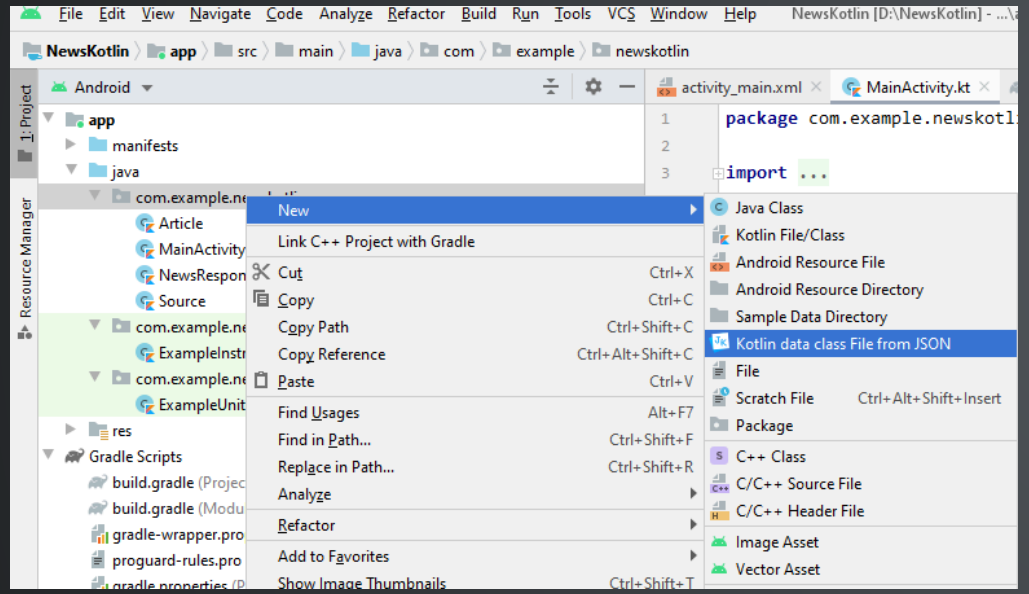
# Les accès aux Web Services

## Installons le plugin JSON to Kotlin



# Les accès aux Web Services

## Utilisons JSON to Kotlin



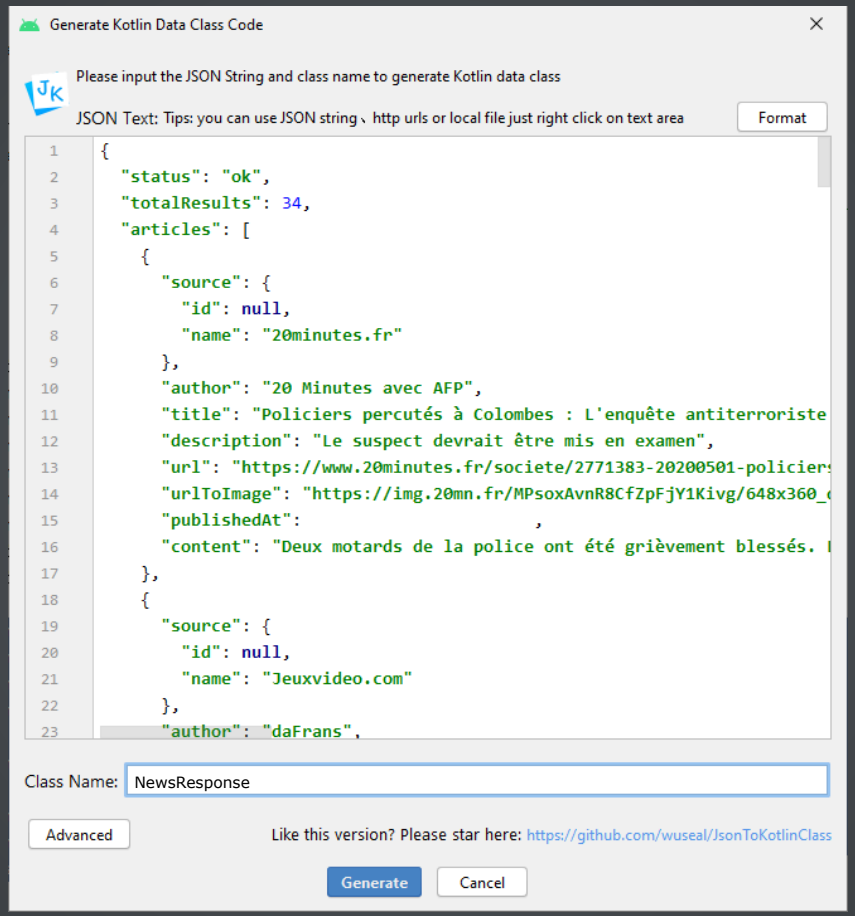
# Les accès aux Web Services

## Création du compte sur le site de news

Afin d'accéder à l'API de News, nous allons suivre les étapes suivantes :

- Créer un compte (gratuit) sur le site News API : https://newsapi.org/.
- Aller dans la section : "News sources" (tout en bas), puis France.
- Copier l'URL de l'API (par exemple : http://newsapi.org/v2/top-headlines?country=fr&apiKey=API_KEY)
- Récupérer le contenu du message (click droit, code source de la page => view-source:http://newsapi.org/v2/top-headlines?country=fr&apiKey=API_KEY (sur chrome).
- Générer les POJO (détail page suivante).

# Les accès aux Web Services



# Les accès aux Web Services

## Paramétrage avancé

Il est possible de configurer le générateur avec le bouton "Advanced". Nous allons par exemple activer l'Annotation Gson.
Nous avons donc maintenant 3 nouvelles classes :

- NewsResponse.
- Article.
- Source.

Vérifions que les types des attributs sont bien des `String`.

# Les accès aux Web Services

## Écriture de l'interface avec l'API

Créons une nouvelle interface : NewsApi contenant le code suivant :

Copier

```
import retrofit2.Call
import retrofit2.http.GET
import retrofit2.http.Query

interface NewsApi {
    @GET("top-headlines")
    fun getNewsList(
        @Query("country") newsSource: String?,
        @Query("apiKey") apiKey: String?
    ): Call<NewsResponse?>?
}
```

# Les accès aux Web Services

## Creation de la classe de service

Copier

```
import retrofit2.Retrofit
import retrofit2.converter.gson.GsonConverterFactory

class RetrofitService {
    private val retrofit = Retrofit.Builder()
        .baseUrl("https://newsapi.org/v2/")
        .addConverterFactory(GsonConverterFactory.create())
        .build()

    fun <S> createService(serviceClass: Class<S>): S {
        return retrofit.create(serviceClass)
    }
}
```

# Les accès aux Web Services

## NewsRepository

Copier

```
import android.util.Log
import androidx.lifecycle.MutableLiveData
import retrofit2.Call
import retrofit2.Callback
import retrofit2.Response

object NewsRepository {
    private const val TAG = "NewsRepository"

    private var newsApi: NewsApi? = null

    init {
        newsApi = RetrofitService().createService(NewsApi::class.java)
    }

    fun getNews(country: String?, key: String?): MutableLiveData<NewsResponse?> {
        val newsData = MutableLiveData<NewsResponse?>()
```

# Les accès aux Web Services

## NewsViewModel

Copier

```
import androidx.lifecycle.LiveData
import androidx.lifecycle.MutableLiveData
import androidx.lifecycle.ViewModel

class NewsViewModel : ViewModel() {
    private var mutableLiveData: MutableLiveData<NewsResponse?>? = null
    fun init() {
        if (mutableLiveData != null) {
            return
        }
        mutableLiveData = NewsRepository.getNews("fr", "96c6792fdad6434fbc3a893daba40e0f")
    }

    fun getNewsRepository(): LiveData<NewsResponse?>? {
        return mutableLiveData
    }
}
```

# Les accès aux Web Services

## Ajout d'un fragment

Nous allons ajouter un fragment `NewsFragment` à notre projet, et dans le layout du fragment, ajouter une `RecyclerView` :

Copier

```
<?xml version="1.0" encoding="utf-8"?>
<FrameLayout xmlns:android="http://schemas.android.com/apk/res/android"
             xmlns:tools="http://schemas.android.com/tools"
             android:layout_width="match_parent"
             android:layout_height="match_parent"
             tools:context=".NewsFragment">
    <androidx.recyclerview.widget.RecyclerView
            android:id="@+id/rvNews"
            android:layout_width="match_parent"
            android:layout_height="match_parent" />
</FrameLayout>
```

# Les accès aux Web Services

## Modification du layout

Nous allons ajouter ce fragment au layout de notre `activity_main.xml` :

Copier

```xml
<?xml version="1.0" encoding="utf-8"?>
<FrameLayout xmlns:android="http://schemas.android.com/apk/res/android"
             xmlns:tools="http://schemas.android.com/tools"
             android:layout_width="match_parent"
             android:layout_height="match_parent"
             tools:context=".MainActivity">
    <fragment android:id="@+id/news_fragment"
              android:name="com.example.todeletenetwork.NewsFragment"
              android:layout_width="match_parent"
              android:layout_height="match_parent" />
</FrameLayout>
```

# Les accès aux Web Services

## Création du layout pour afficher une news

Nous allons par exemple écrire un layout : item_news.xml :

Copier

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout xmlns:android="http://schemas.android.com/apk/res/android"
                                xmlns:app="http://schemas.android.com/apk/res-auto"
                                xmlns:tools="http://schemas.android.com/tools"
                                android:layout_width="match_parent"
                                android:layout_height="wrap_content"
                                android:layout_margin="5dp">

    <TextView
            android:id="@+id/item_news_textView_title"
            android:layout_width="0dp"
            android:layout_height="wrap_content"
            android:transitionName="title"
            app:layout_constraintEnd_toStartOf="@id/item_news_imageView_image"
            app:layout_constraintStart_toStartOf="parent"
            app:layout_constraintTop_toTopOf="parent"
            tools:text="Title title title title title title title title title title title title title title title title title title title title title title title title
```

# Les accès aux Web Services

## Création de l'adapter : NewsAdapter

Copier

```
1  import android.content.Context
2  import android.view.LayoutInflater
3  import android.view.View
4  import android.view.ViewGroup
5  import android.widget.ImageView
6  import android.widget.TextView
7  import androidx.recyclerview.widget.RecyclerView
8  import androidx.recyclerview.widget.RecyclerView.ViewHolder
9  import com.bumptech.glide.Glide
10
11 class NewsAdapter(val context: Context, val myDataset: List<Article>):
12     RecyclerView.Adapter<NewsAdapter.MyViewHolder>() {
13     private var mDataset: List<Article>? = null
14     private var mInflater: LayoutInflater? = null
15
16     init {
17         mInflater = LayoutInflater.from(context)
```

# Les accès aux Web Services

## Appel dans notre fragment

Déclarons les variables de classes suivantes :

```
1 private var newsAdapter: NewsAdapter? = null
2 private val newsViewModel: NewsViewModel by viewModels()
3 private var rvHeadline: RecyclerView? = null
4 private var articleArrayList = arrayListOf<Article>()
```

Copier

# Les accès aux Web Services

## Appel dans notre fragment

Déclarons la fonction suivante :

Copier

```
1  private fun setupRecyclerView(context: Context) {
2      if (newsAdapter == null) {
3          newsAdapter = NewsAdapter(context, articleArrayList)
4          rvHeadline?.layoutManager = LinearLayoutManager(context)
5          rvHeadline?.adapter = newsAdapter
6          //rvHeadline?.setItemAnimator(DefaultItemAnimator())
7          //rvHeadline?.setNestedScrollingEnabled(true)
8      } else {
9          newsAdapter?.notifyDataSetChanged()
10     }
11 }
```

# Les accès aux Web Services

## Appel dans notre fragment

Remplaçons le corps de notre méthode onCreateView :

Copier

```
// Inflate the layout for this fragment
val rootLayout = inflater.inflate(R.layout.fragment_news, container, false)

context?.let { safeContext ->
    rvHeadline = rootLayout.findViewById(R.id.rvNews)
    newsViewModel.init()
    newsViewModel.getNewsRepository()?.observe(viewLifecycleOwner) {
        val newsArticles = it?.articles
        if (newsArticles != null) {
            articleArrayList.addAll(newsArticles)
            newsAdapter?.notifyDataSetChanged()
        }
    }
    setupRecyclerView(safeContext)
}
return rootLayout
```

# Les accès aux Web Services

## Erreurs possibles

Si vous avez l'erreur :

```
java.lang.NoSuchMethodError: No static method metafactory(Ljava/lang/invoke/MethodHandles
```

Copier

Il faut activer la compatibilité JVM 1.8 en ajoutant dans le build.gradle, le code suivant :

```
 android {
...
    compileOptions {
        sourceCompatibility JavaVersion.VERSION_1_8
        targetCompatibility JavaVersion.VERSION_1_8
    }
    kotlinOptions {
        jvmTarget = "1.8"
    }
}
```

Copier

# Les accès aux Web Services

## Erreurs possibles

Si vous avez l'erreur, sur l'émulateur :

```
1 java.net.SocketException: socket failed: EPERM (Operation not permitted)
```

Copier

Il suffit que nous désinstallions l'application, pour la réinstaller.

# Les accès aux Web Services

## Erreurs possibles

Si nous avons une SocketTimeoutException, il suffit d'allonger la durée du Timeout.
Dans la classe : RetrofitService remplaçons

```
private val retrofit = Retrofit.Builder()
    .baseUrl("https://newsapi.org/v2/")
    .addConverterFactory(GsonConverterFactory.create())
    .build()
```

Copier

Par :

```
var client = OkHttpClient.Builder()
    .connectTimeout(100, TimeUnit.SECONDS)
    .readTimeout(100, TimeUnit.SECONDS).build()
private val retrofit = Retrofit.Builder()
    .baseUrl("https://newsapi.org/v2/").client(client)
    .addConverterFactory(GsonConverterFactory.create(Gson()))
    .build()
```

Copier

Cela nous permet d'attendre un peu plus de temps avant d'avoir une erreur. Il faut dans ce cas-là, afficher une indication à l'utilisateur pour qu'il comprenne qu'il doit attendre.

# Les accès aux Web Services

## Erreurs possibles

Si nous avons l'erreur suivante :

```
Caused by: java.lang.IllegalArgumentException: Binary XML file line #11: Must specify unique android:id, android:tag, or have a parent with an id for com.example.todeletenetwork.NewsFragment
```

Copier

Il suffit de préciser un id au tag `fragment` dans le layout de notre activity.

# Compléments

- Les capteurs. Les API Google de localisation.

# Utilisation des capteurs

## Mise en œuvre de capteurs.

Nous allons développer une application qui va éteindre l'écran quand on est proche de l'écran, et le rallumer quand on est à distance. Pour cela plusieurs étapes :

- Nous allons lister les capteurs.
- Récupérer le capteur de proximité.
- S'enregistrer sur les changements.
- Réagir aux changements.

# Utilisation des capteurs

## Liste des capteurs (Java).

Pour lister tous les capteurs disponibles sur notre smartphone, utilisons le code suivant :

Copier

```
SensorManager sensorManager = (SensorManager) getSystemService(Context.SENSOR_SERVICE);
List<Sensor> liste = sensorManager.getSensorList(Sensor.TYPE_ALL);
if (BuildConfig.DEBUG) {
    for (Sensor currentSensor : liste) {
        Log.d(TAG, "onCreate sensor " + currentSensor);
    }
}
```

# Utilisation des capteurs

## Liste des capteurs (Kotlin).

Pour lister tous les capteurs disponibles sur notre smartphone, utilisons le code suivant :

```kotlin
val sensorManager = getSystemService(Context.SENSOR_SERVICE) as SensorManager
var liste = sensorManager.getSensorList(Sensor.TYPE_ALL)
if (BuildConfig.DEBUG) {
    for (currentSensor in liste) {
        Log.d(TAG, "onCreate sensor $currentSensor")
    }
}
```

Copier

# Utilisation des capteurs

## Récupérons le capteur de proximité (Java).

```
1 private Sensor mProximitySensor = null; // En attribut de classe.
2 mProximitySensor = sensorManager.getDefaultSensor(Sensor.TYPE_PROXIMITY);
```

Copier

# Utilisation des capteurs

## Récupérons le capteur de proximité (Kotlin)

```kotlin
var mProximitySensor:Sensor? = null // En attribut de classe.
mProximitySensor = sensorManager.getDefaultSensor(Sensor.TYPE_PROXIMITY)
```

Copier

# Utilisation des capteurs

## Enregistrons-nous, sur les changements (Java).

Dans un premier temps, nous récupérons le manager des capteurs :

```
1 private SensorManager mSensorManager = null; // En attribut de classe.
2 mSensorManager = (SensorManager) getSystemService(Context.SENSOR_SERVICE);
```

Copier

Puis nous nous enregistrons sur les changements uniquement quand l'application est active :

```
1  @Override
2  protected void onResume() {
3      super.onResume();
4      mSensorManager.registerListener(mSensorEventListener, mProximitySensor, SensorManager.SENSOR_DELAY_NORMAL);
5  }
6
7  @Override
8  protected void onPause() {
9      super.onPause();
10     mSensorManager.unregisterListener(mSensorEventListener, mProximitySensor);
11 }
```

Copier

# Utilisation des capteurs

## Le listener (Java).

Le listener qui va réagir aux changements est définis par :

```
final SensorEventListener mSensorEventListener = new SensorEventListener() {
    public void onAccuracyChanged(Sensor sensor, int accuracy) {
        // Que faire en cas de changement de précision ?
    }

    public void onSensorChanged(SensorEvent sensorEvent) {
        if(BuildConfig.DEBUG){
            Log.d(TAG, "onSensorChanged " + sensorEvent.values.length);
            Log.d(TAG, "onSensorChanged " + sensorEvent.values[0]);
        }
    }
};
```

Copier

Il ne nous reste plus qu'à implémenter notre code fonctionnel.
Exemple plus poussé d'utilisation du capteur de proximité pour éteindre l'écran.

# Utilisation des capteurs

## Enregistrons-nous, sur les changements (Kotlin)

Dans un premier temps, nous récupérons le manager des capteurs :

```kotlin
var mSensorManager: SensorManager? = null // En attribut de classe.
mSensorManager = getSystemService(Context.SENSOR_SERVICE) as SensorManager
```

Copier

Puis nous nous enregistrons sur les changements uniquement quand l'application est active :

```kotlin
override fun onResume() {
    super.onResume()
    mSensorManager?.registerListener(
        mSensorEventListener,
        mProximitySensor,
        SensorManager.SENSOR_DELAY_NORMAL
    )
}

override fun onPause() {
    super.onPause()
    mSensorManager?.unregisterListener(mSensorEventListener, mProximitySensor)
}
```

Copier

# Utilisation des capteurs

## Le listener (Kotlin)

Le listener qui va réagir aux changements est définis par :

Copier

```kotlin
val mSensorEventListener: SensorEventListener = object : SensorEventListener {
    override fun onAccuracyChanged(sensor: Sensor, accuracy: Int) {
        // Que faire en cas de changement de précision ?
    }

    override fun onSensorChanged(sensorEvent: SensorEvent) {
        if (BuildConfig.DEBUG) {
            Log.d(TAG, "onSensorChanged ${sensorEvent.values.size}")
            Log.d(TAG, "onSensorChanged ${sensorEvent.values[0]}")
        }
    }
}
```

Il ne nous reste plus qu'à implémenter notre code fonctionnel.
Exemple plus poussé d'utilisation du capteur de proximité pour éteindre l'écran.

# Annexe

## Mode développeur

Pour mettre son téléphone Android en mode développeur :

- Allons dans les paramètres.
- À propos du téléphone.
- Cliquons plusieurs fois sur le numéro de build.
- Une fois débloqué, nous pouvons activer le mode débugger, et la connexion USB.

Si au lieu de cliquer sur la version du build, nous cliquons sur la version d'Android, une surprise nous attend :-)

# Annexe

## Erreur classique

Une erreur : `Unsupported major.minor version 52.0` nous indique, que nous avons compilé le code avec une version de JDK 1.8 et que nous essayons de le lancer avec une version antérieure :

# Références

## Les références

- La référence : https://kotlinlang.org/docs/reference
- La source : https://github.com/JetBrains/kotlin
- Autre ressource très complète en FR : https://riptutorial.com/Download/kotlin-fr.pdf
- Rappels des bases : https://kotlinlang.org/docs/reference/basic-syntax.html

# Références

## Cours en ligne

- Sur Openclassrooms : https://openclassrooms.com/fr/courses/5353106-initiez-vous-a-kotlin
- Sur CodinGame : https://www.codingame.com/playgrounds/28826/formation-kotlin/les-bases-de-kotlin
- Kotlin Koans : https://play.kotlinlang.org/koans/overview

# Références

## Tests/Challenges

- https://github.com/Kotlin/kotlin-koans-edu
- https://tech.io/explore
- https://github.com/SK1dev/KotlinChallenges/blob/master/README.md
- https://github.com/SK1dev/KotlinChallengesPart2
- https://github.com/igorwojda/kotlin-coding-puzzle

# Références

## Articles sur des points précis

- https://typealias.com/
- http://zetcode.com/all/
- Pourquoi adopter Kotlin : https://medium.com/videdressing-engineering/pourquoi-je-suis-pass%C3%A9-%C3%A0-kotlin-7d40d79054a4

# Références

## Livres

- https://kotlinlang.org/docs/books.html

# Références

## Liste des mots clés

- https://kotlinlang.org/docs/reference/keyword-reference.html

# Références

## Kotlin pour les applications Backend

- https://soat.developpez.com/tutoriels/kotlin-application-backend/

# Références

## Kotlin pour Android

- https://www.udacity.com/course/ud9012
- https://blog.ippon.fr/2017/12/11/introduction-a-kotlin-pour-android/
- https://kotlinlang.org/docs/reference/android-overview.html
- https://developer.android.com/kotlin
- https://codelabs.developers.google.com/codelabs/android-room-with-a-view-kotlin/#0
- https://codelabs.developers.google.com/codelabs/kotlin-coroutines/#0