

Hello world :)

Hello World :)

Sommaire

Android != Java

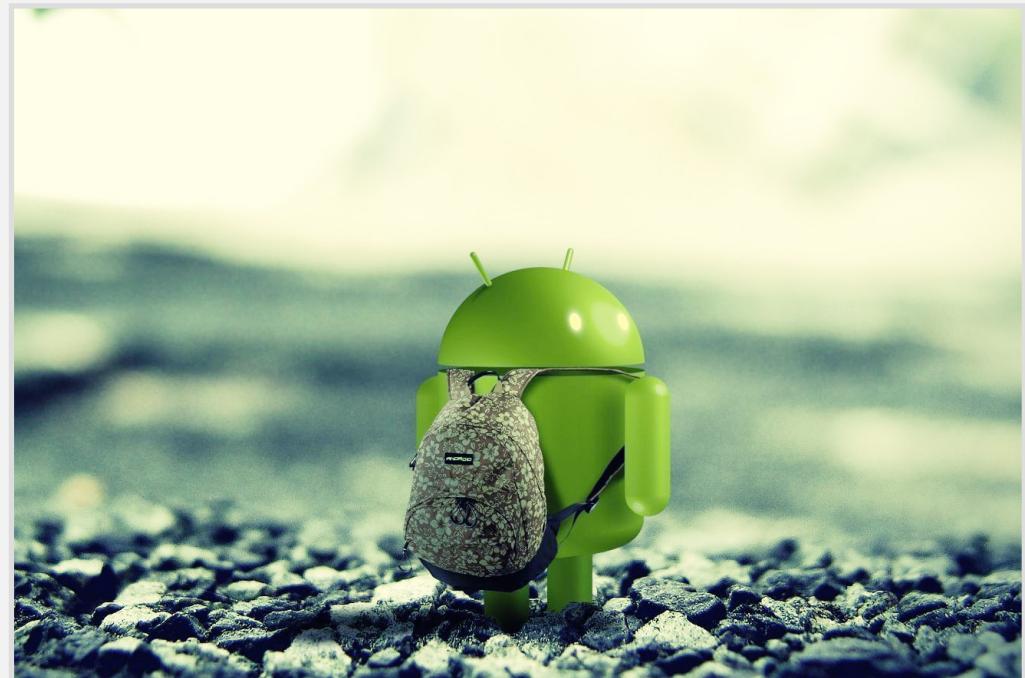
Quelques rappels Java

Architecture d'un projet

Afficher un message

Android Studio

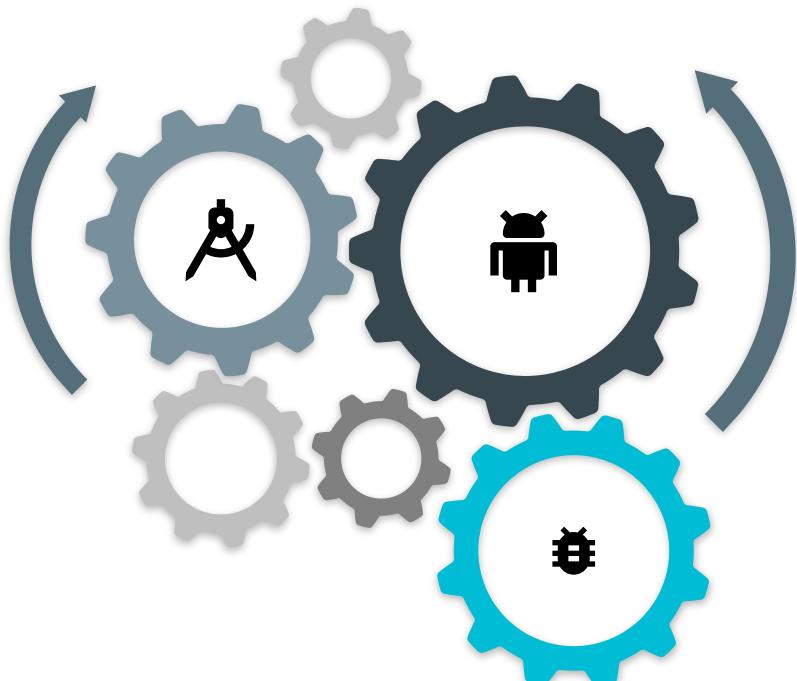
Kotlin 😍



Android utilise la syntaxe Java

[Android != Java](#)[Quelques rappels Java](#)[Architecture d'un projet](#)[Android Studio](#)[Kotlin](#)

3



Presque du Java

Android n'intègre pas une JVM classique, mais une machine virtuelle spécialement conçue :

- ✓ Dalvik (jusqu'à Android 4.4)
- ✓ ART (depuis Android 5.0)

Syntaxe Java

- ✓ Syntaxe Java 7 (toutes versions)
- ✓ Syntaxe Java 8
(en version *bêta* - une partie des fonctions)

Hello World :)

Sommaire

Android != Java

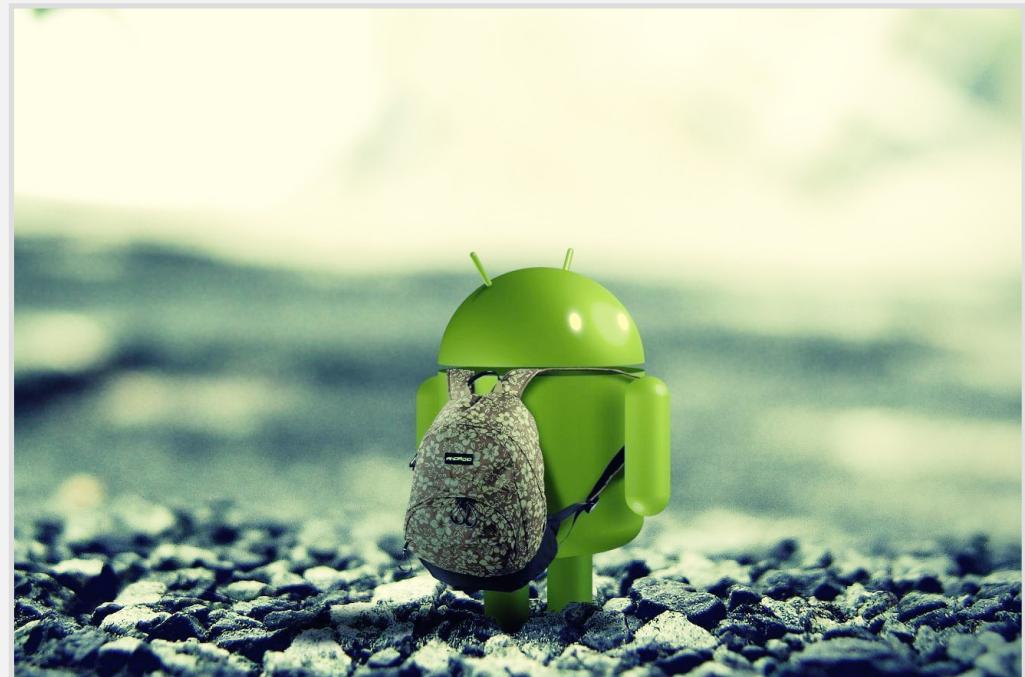
Quelques rappels Java

Architecture d'un projet

Afficher un message

Android Studio

Kotlin 😍



Quelques optimisations



Android != Java

Quelques rappels Java

Architecture d'un projet

Android Studio

Kotlin

5



Constantes

```
static final int val = 1;
```



Tableaux / Listes

Privilégier les tableaux



Chaînes de caractères

`StringBuilder` / `StringBuffer`



Getter/setter

A éviter (surtout pour Dalvik)



Eviter les HashMap

Privilégier les `ArrayMap` / `SparseArray`



Classes internes

A éviter (surtout pour Dalvik)

Java : les bases

[Android != Java](#)[**Quelques rappels Java**](#)[Architecture d'un projet](#)[Android Studio](#)[Kotlin](#)

6

byte, short, int, long
char
float, double
boolean

Types primitifs

Qu'il faut privilégier aux objets

```
int[] array = new int[2];  
array[0] = 1;  
  
int[] array = new int[] {0,1,2};  
  
MyObject[] objects = new MyObject[42];
```

JAVA

Tableaux

Qu'il faut privilégier aux Collections
(si possible)

Java : les Collections

[Android != Java](#)[Quelques rappels Java](#)[Architecture d'un projet](#)[Android Studio](#)[Kotlin](#)

List : ArrayList, LinkedList, Vector, Stack...

Map : HashMap, TreeMap...

Set : TreeSet, HashSet, SortedSet...

Collections

A utiliser avec parcimonie

```
ArrayList<String> list = new ArrayList(2);  
  
list.add("toto");  
  
list.remove(0);  
  
list.size();  
  
Collections.sort(list);
```

JAVA

Exemple d'utilisation des ArrayList

Java : les bases

[Android != Java](#)[Quelques rappels Java](#)[Architecture d'un projet](#)[Android Studio](#)[Kotlin](#)

8

```
while (condition) {}  
  
for (int i = 0; i != 10; i++) {}  
  
for (String str : strings) {}  
  
do {} while (i > 0)
```

JAVA

Les boucles

```
if () {} else if {} else {}  
  
switch() { case 0: break; default: ;  
  
int max = i > 0 ? 10 : 20;
```

JAVA

Les conditions

Le switch fonctionne avec les numériques et chaînes de caractères uniquement

Java : orienté objet

[Android != Java](#)[Quelques rappels Java](#)[Architecture d'un projet](#)[Android Studio](#)[Kotlin](#)

9

Les classes héritent forcément d'*Object*

Pas d'héritage multiple (*extends*)

Plusieurs interfaces (*implements*)

Implémentation avec le mot clé *new*

 JAVA

Les règles

```
public class Toto extends List implements  
ClickListener, LongClickListener {  
  
    public class Toto() {}  
  
    public String toString() { return "a"; }  
  
    public String equals(Object o) {  
        return o == this;  
    }  
}
```

 JAVA

Exemple de classe

Java : interfaces vs classes abstraites

Android != Java

Quelques rappels Java

Architecture d'un projet

Android Studio

Kotlin

10

```
public interface MyListener {  
  
    void onButtonClicked();  
  
}
```

JAVA

Une interface

```
public abstract class MyListener {  
  
    public abstract void init();  
  
    public String toString() { return "toto"; }  
  
}
```

JAVA

Une classe abstraite

Java : passage par valeur ou référence

Android != Java

Quelques rappels Java

Architecture d'un projet

Android Studio

Kotlin

11

```
public void editA(int a) {  
    a++;  
}  
  
int a = 0;  
editA(a);
```

JAVA

Par valeur

La variable a n'est pas modifiée
a = 0 à la fin du programme

```
public class B {  
    public int mInt;  
    public B(int anInt) {  
        mInt = anInt;  
    }  
}  
  
public void editB(B b) {  
    B.mInt++;  
}  
  
B b = new B(0);  
editB(b);
```

JAVA

Par référence

L'Objet conserve les modifications faites
dans la méthode
b = 1 à la fin du programme

Java : cheat sheet

Android != Java

Quelques rappels Java

Architecture d'un projet

Android Studio

Kotlin

12

Visibilité : Classes

- **Public** : visible par tous
- **<défaut>** : uniquement par les classes de son package

Mots clés

- **final** : classe, méthode ou attribut qui ne peut plus être modifié (constante)
- **static** : classe, méthode ou attribut qui est visible de manière globale
- **try {} catch {} finally** : gérer les exceptions
- **throw new Exception()** : lever une nouvelle exception
- **this** : classe courante
- **MyObject.this** : classe parente depuis une classe interne
- **super()** : appel du constructeur parent
- **super.method()** : appel de la méthode parente

Visibilité : Attributs

- **Public** : visible par tous
- **<défaut>** : par les classes de son package et ses fils
- **Protected** : <default> + visible par les fils dans les autres paquets
- **Private** : uniquement par la classe

Annotations

- **@Override** : redéfinit une méthode implémentée dans une classe parente (optionnel)

Méthodes

- **Surcharge des paramètres** :
void method()
void method(int a)
- **Nombre indéterminé de paramètres** :
void method(int... objects)

Java 8 : comment l'activer ?

[Android != Java](#)[Quelques rappels Java](#)[Architecture d'un projet](#)[Android Studio](#)[Kotlin](#)

13



```
android {  
  
    compileOptions {  
  
        sourceCompatibility JavaVersion.VERSION_1_8  
        targetCompatibility JavaVersion.VERSION_1_8  
  
    }  
  
}
```

GROOVY

app/build.gradle

Ajouter les lignes en bleu

Java 8 : les interfaces avec méthode



Android != Java

Quelques rappels Java

Architecture d'un projet

Android Studio

Kotlin

14

```
public interface MyTimeInterface {  
  
    void setTime(int hour, int minute);  
  
    default printTime(int hour, int minute) {  
  
        Log.d("MyTimeInterface", hour + "h:" + minute + "m");  
    }  
}
```

A dark circular badge with the word "JAVA" written in white capital letters.

Une interface Java avec l'implémentation d'une méthode

Java 8 : les lambdas



Android != Java

Quelques rappels Java

Architecture d'un projet

Android Studio

Kotlin

15

```
view.setOnClickListener(new View.OnClickListener() {  
  
    public void onClick(View v) {  
  
        ...  
  
    }  
  
});
```

A dark circular badge with the word "JAVA" written in white capital letters.

Soit le code suivant

Java 8 : les lambdas



Android != Java

Quelques rappels Java

Architecture d'un projet

Android Studio

Kotlin

16

```
view.setOnClickListener(v -> Log.d("TAG", "Click!"));  
  
view.setOnClickListener(v -> {  
    Log.d("TAG", "1ère instruction!");  
    Log.d("TAG", "2ème instruction!");  
});  
  
view.setOnClickListener(View::getMeasuredHeight);
```

Avec une seule
instruction

Avec plusieurs
instructions

Sur le même objet
qui est envoyée

Peut être remplacé par

Java 8 : les lambdas



Android != Java

Quelques rappels Java

Architecture d'un projet

Android Studio

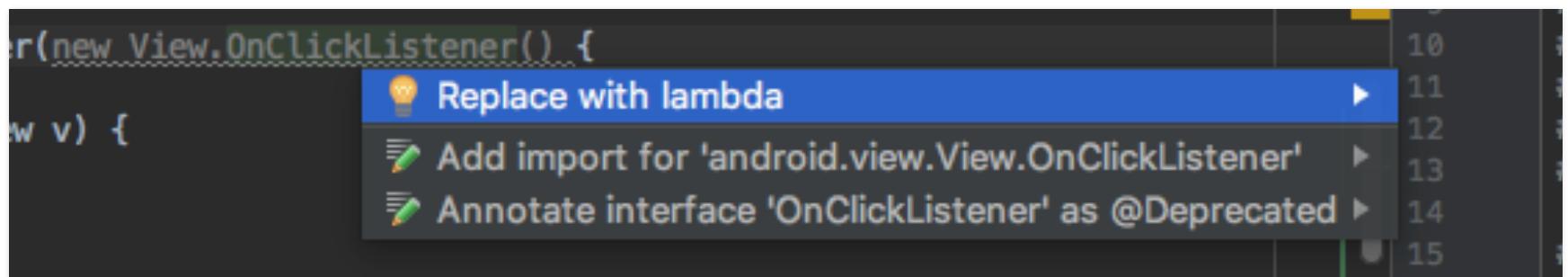
Kotlin

17

```
toolbar = findViewById(R.id.activity_logs_toolbar);

viewPager.setOnClickListener(new View.OnClickListener() {
    @Override
    Anonymous new View.OnClickListener() can be replaced with lambda more... (⌘F1)
    }
});
```

Android Studio indique lorsqu'on peut remplacer par des lambdas



En cliquant sur Alt+Enter, le remplacement par un lambda est alors proposé

Hello World :)

Sommaire

Android != Java

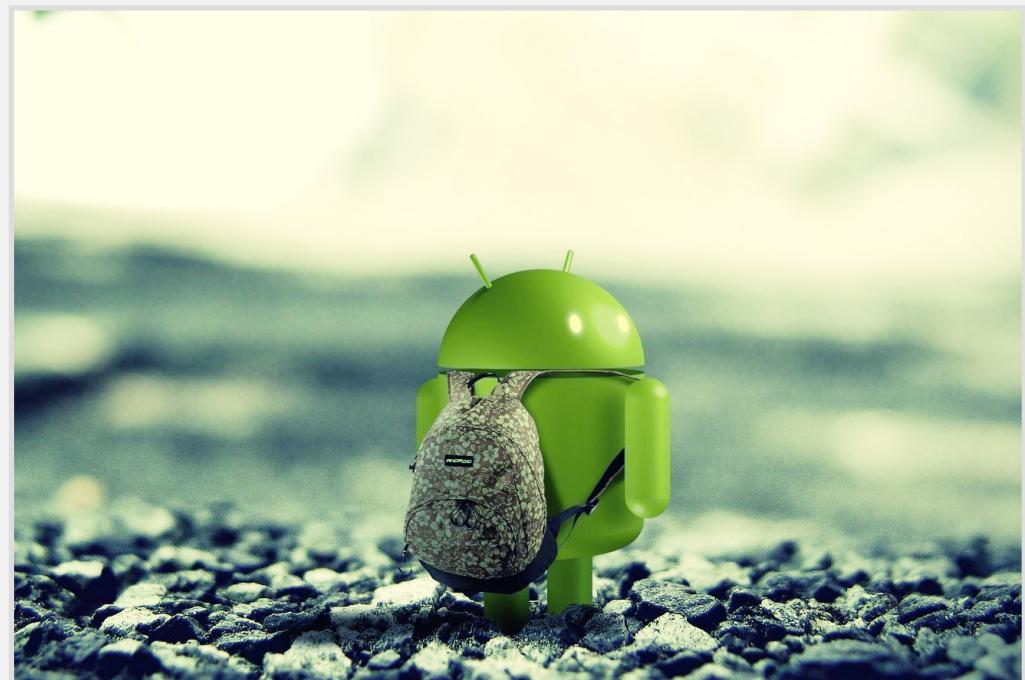
Quelques rappels Java

Architecture d'un projet

Afficher un message

Android Studio

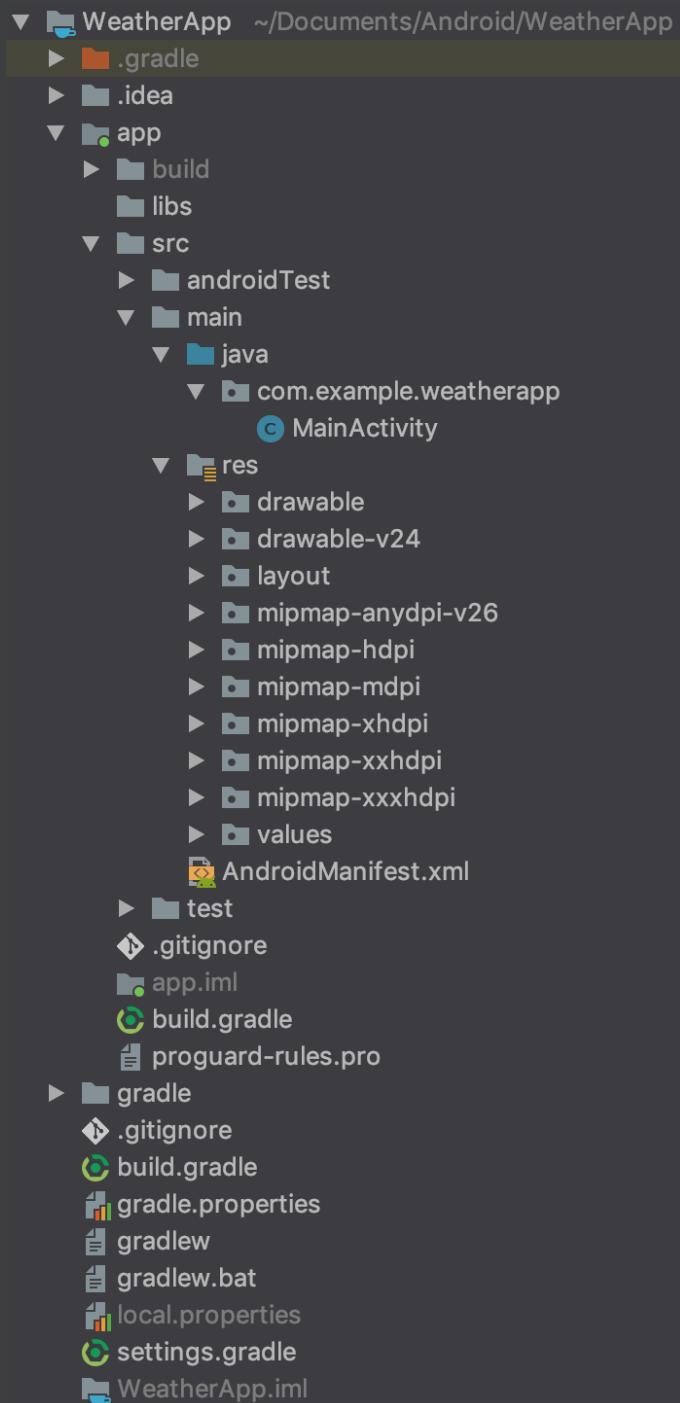
Kotlin 😍



Architecture d'un projet

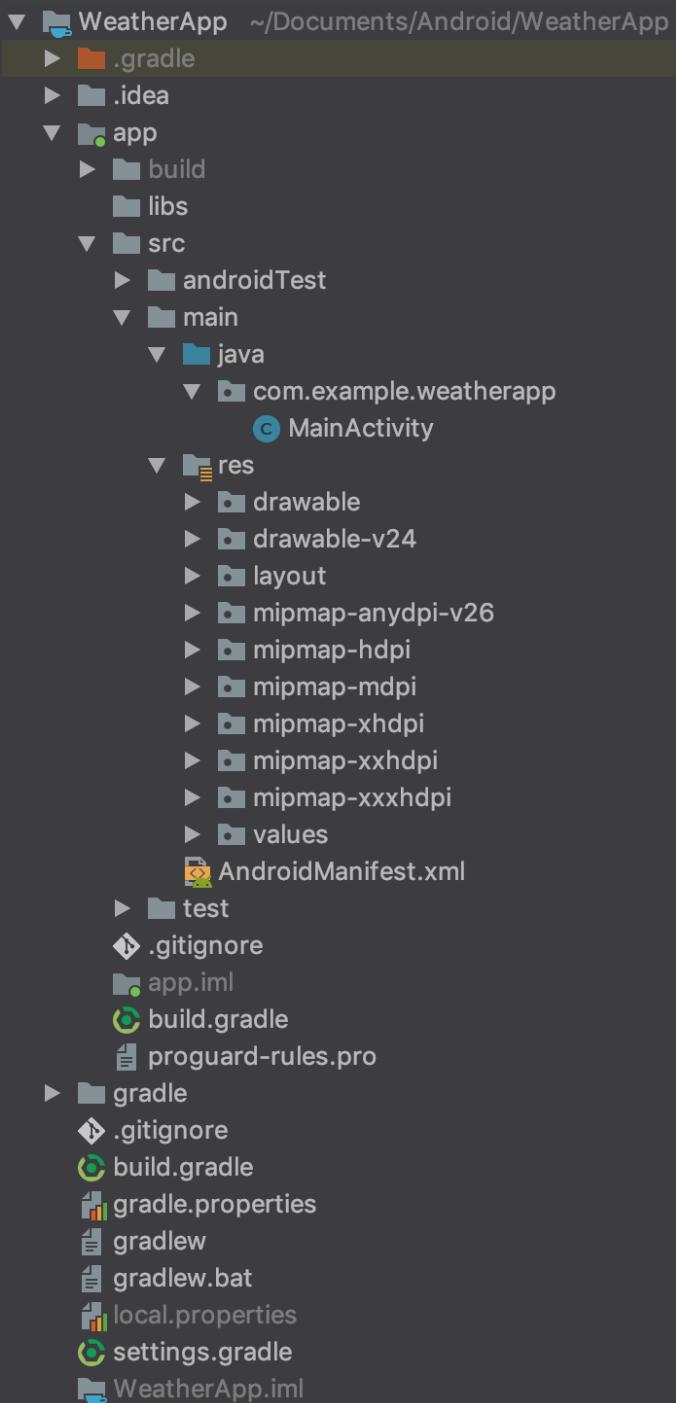
build : tout ce qui est généré

- generated
- intermediates
- **outputs**
 - apk & lint



Architecture d'un projet

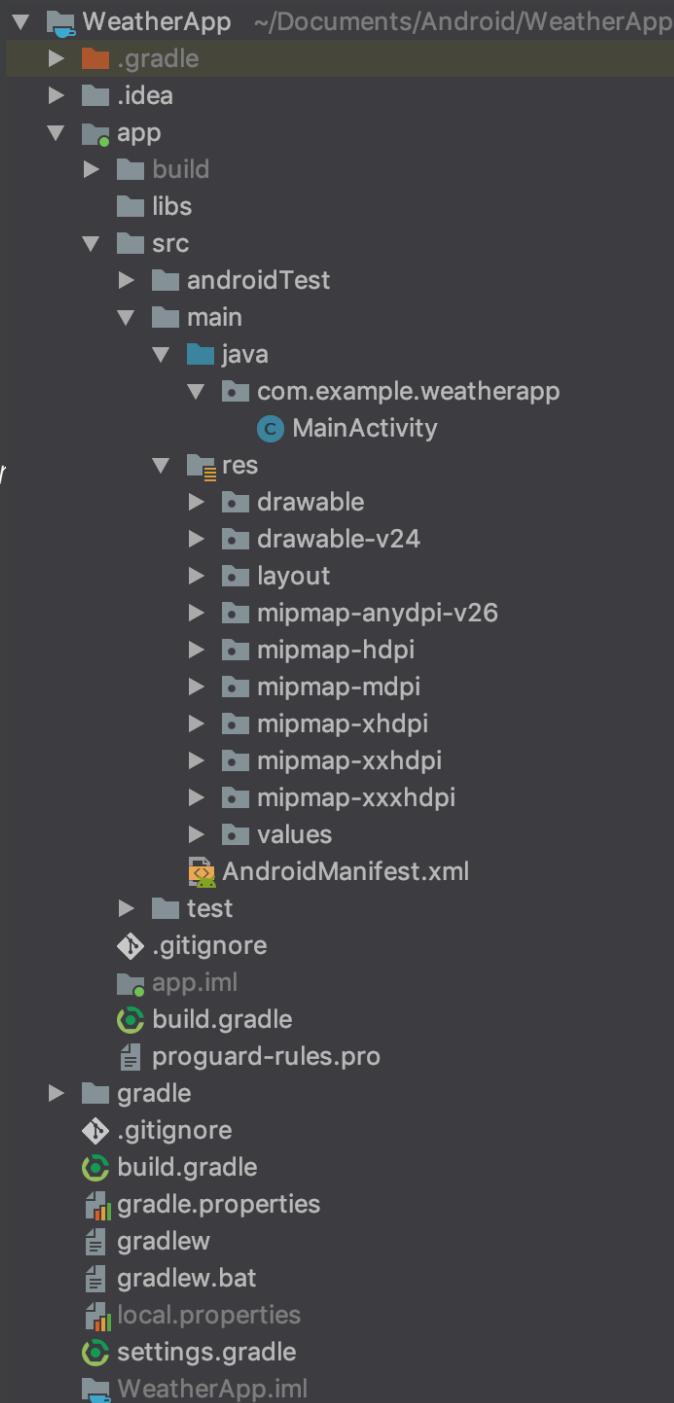
libs : emplacement des jars



Architecture d'un projet

src : code source de l'application

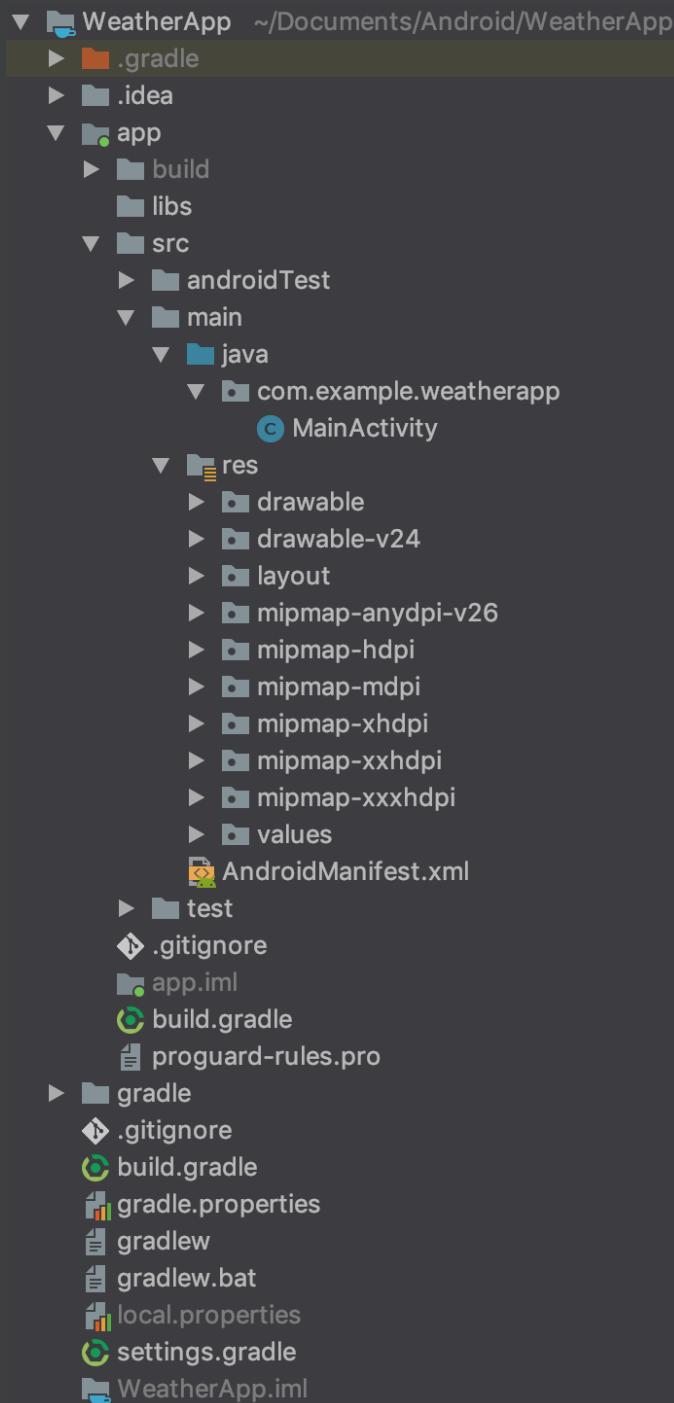
- **main** : emplacement par défaut du code (sans *flavor*, ni *variant*)
 - **java** : code Kotlin ou Java
 - **res** : ressources Android
 - **AndroidManifest.xml**
 - **aidl** : IDL
 - **jniLibs** : code natif compilé (.so)
 - **cpp** : code c/c++
 - **assets** : fichiers statiques



Architecture d'un projet

src/main/res : ressources

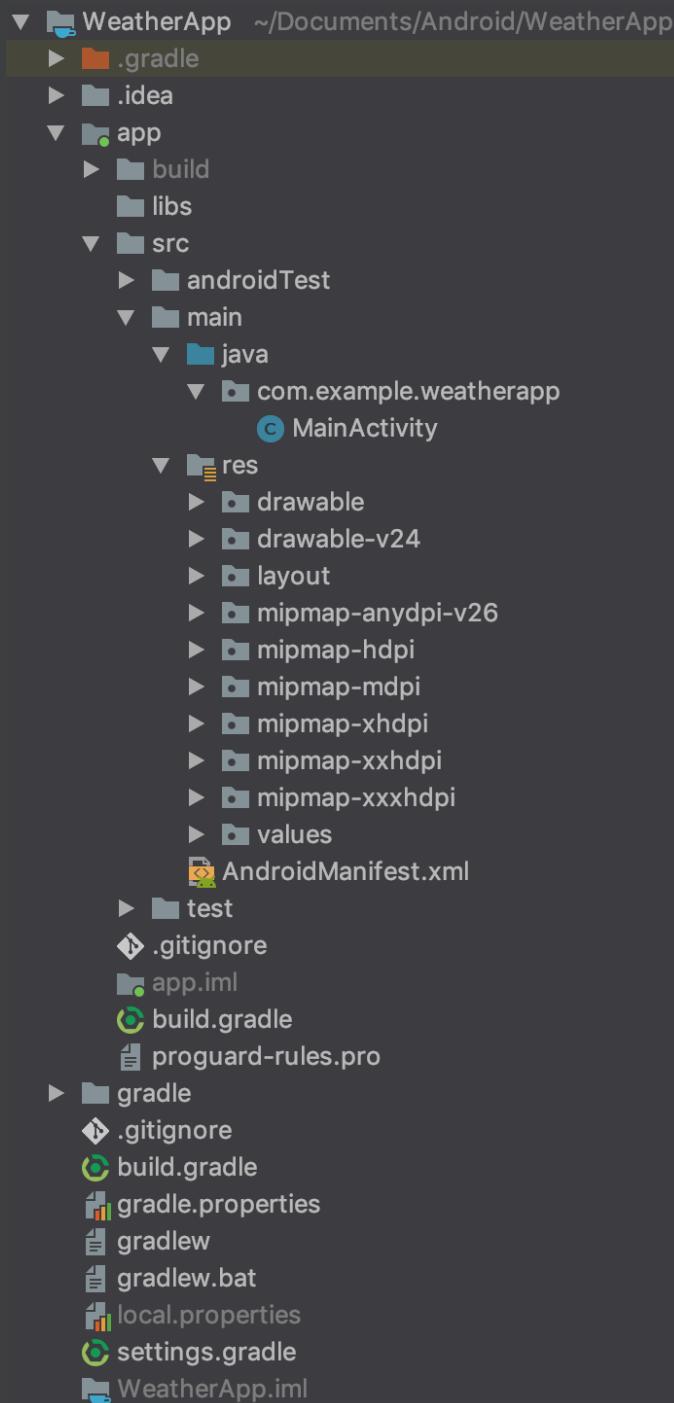
- Les noms de fichiers doivent commencer par
 - une lettre
 - en minuscule
- Ex : `activity_helloworld`
`ActivityHelloWorld`



Architecture d'un projet

src/main/res : ressources

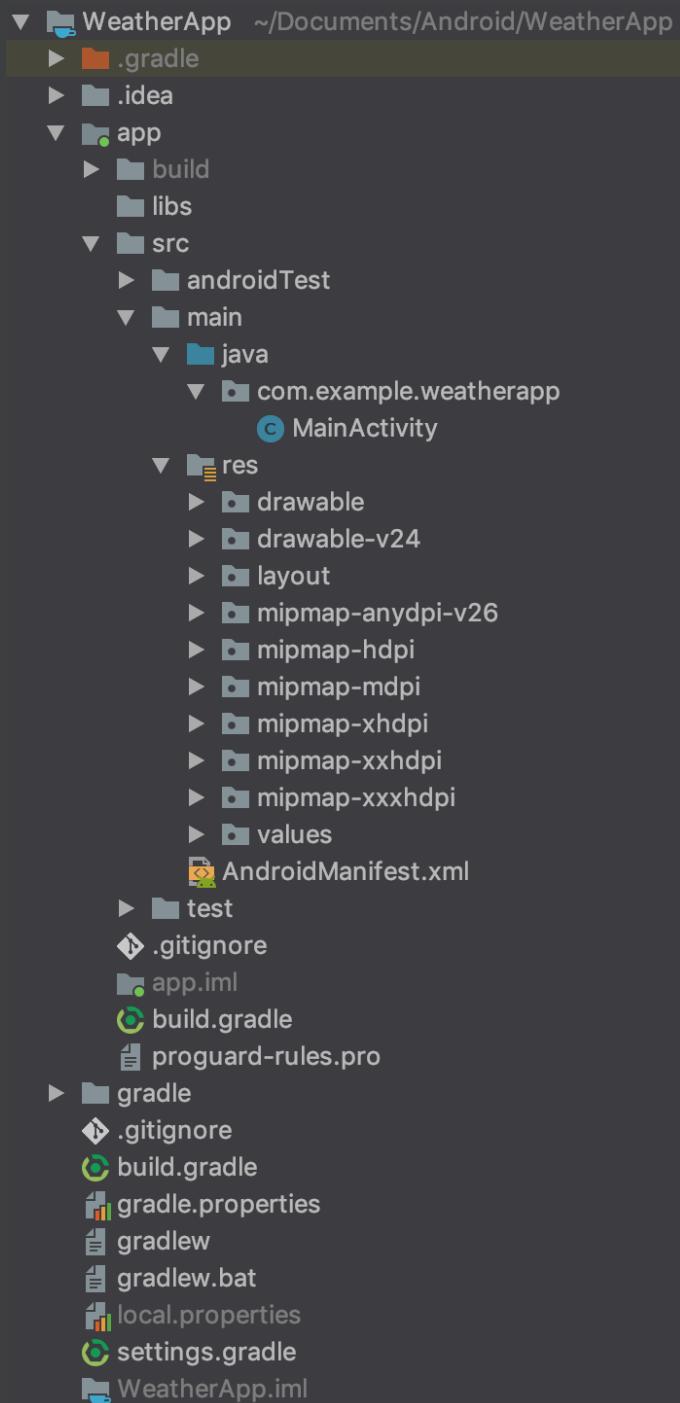
- **drawable** : ressources graphique
- Attention : ne pas mettre des fichiers avec le même nom, mais avec de extensions différentes
(ex : *a.png* & *a.jpg*)



Architecture d'un projet

src/main/res : ressources

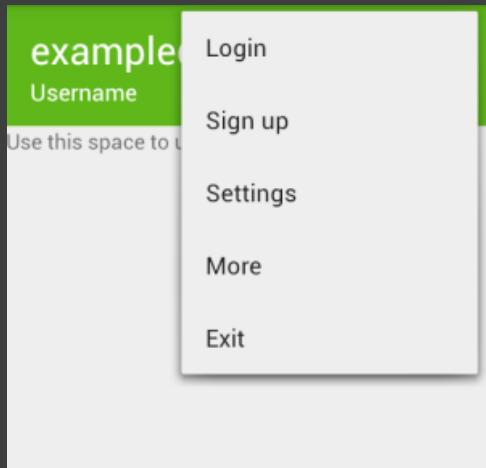
- **layout** : contient tous les layouts de l'application



Architecture d'un projet

src/main/res : ressources

- **menu** : menus d'une ActionBar/Toolbar ou d'autres composants

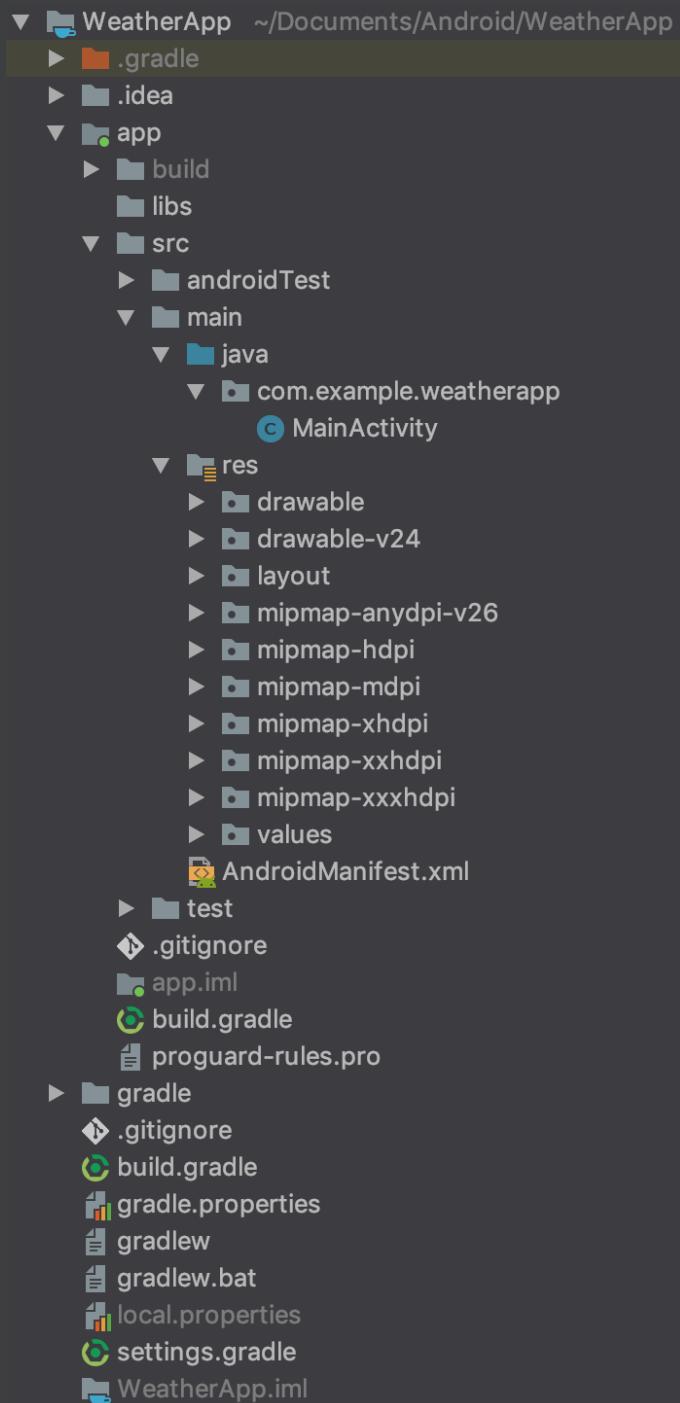


▼	WeatherApp	~/Documents/Android/WeatherApp
►	.gradle	
►	.idea	
▼	app	
►	build	
►	libs	
▼	src	
►	androidTest	
▼	main	
▼	java	
►	com.example.weatherapp	
●	MainActivity	
▼	res	
►	drawable	
►	drawable-v24	
►	layout	
►	mipmap-anydpi-v26	
►	mipmap-hdpi	
►	mipmap-mdpi	
►	mipmap-xhdpi	
►	mipmap-xxhdpi	
►	mipmap-xxxhdpi	
►	values	
●	AndroidManifest.xml	
►	test	
◆	.gitignore	
■	app.iml	
●	build.gradle	
📄	proguard-rules.pro	
►	gradle	
►	.gitignore	
●	build.gradle	
📊	gradle.properties	
📄	gradlew	
📄	gradlew.bat	
📊	local.properties	
●	settings.gradle	
📁	WeatherApp.iml	

Architecture d'un projet

src/main/res : ressources

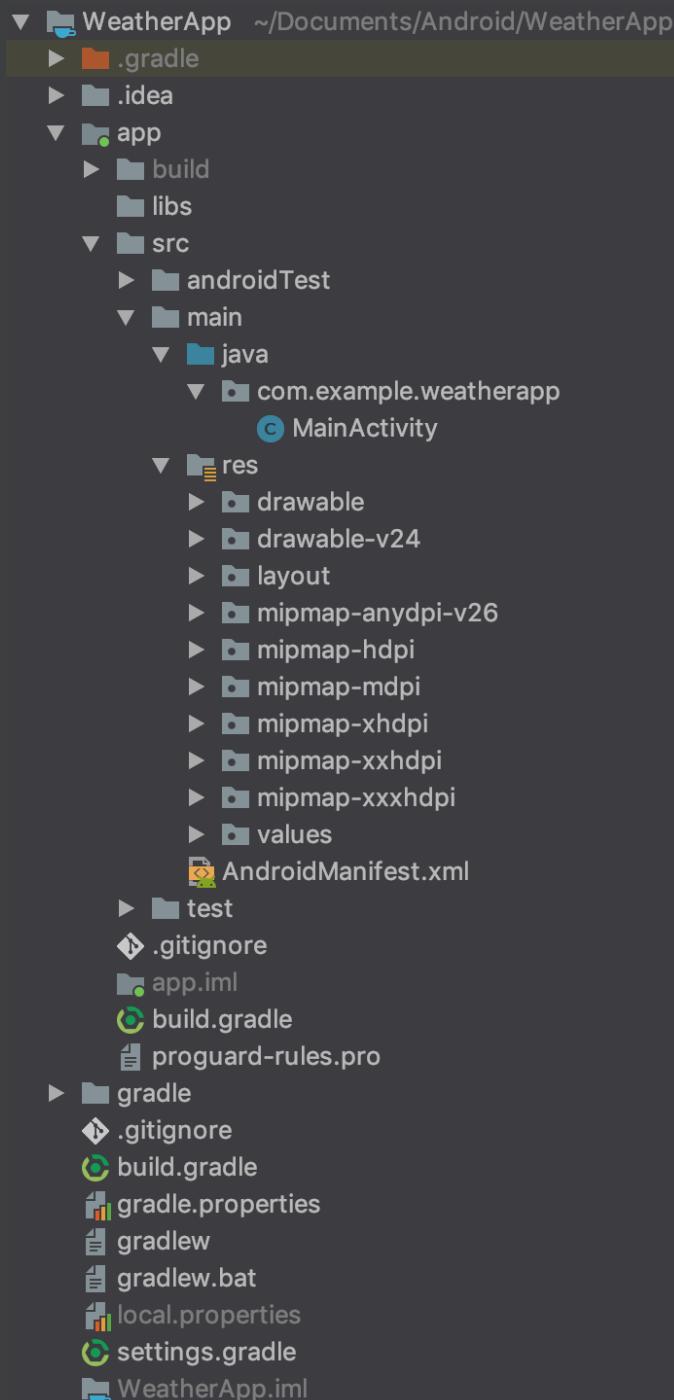
- **mipmap** : icône(s) pour l'application



Architecture d'un projet

src/main/res : ressources

- **values** : données diverses
 - **ids**
 - **arrays** : tableaux
 - **colors** : couleurs
 - **dimens** : dimensions
 - **strings** : traductions
 - **styles**
 - **raw** : fichiers (audio, CSV...)
 - **xml** : données (ex : configuration d'un widget)



Les versions d'Android avec leur code (API Level)

Android != Java

Quelques rappels Java

Architecture d'un projet

Android Studio

Kotlin

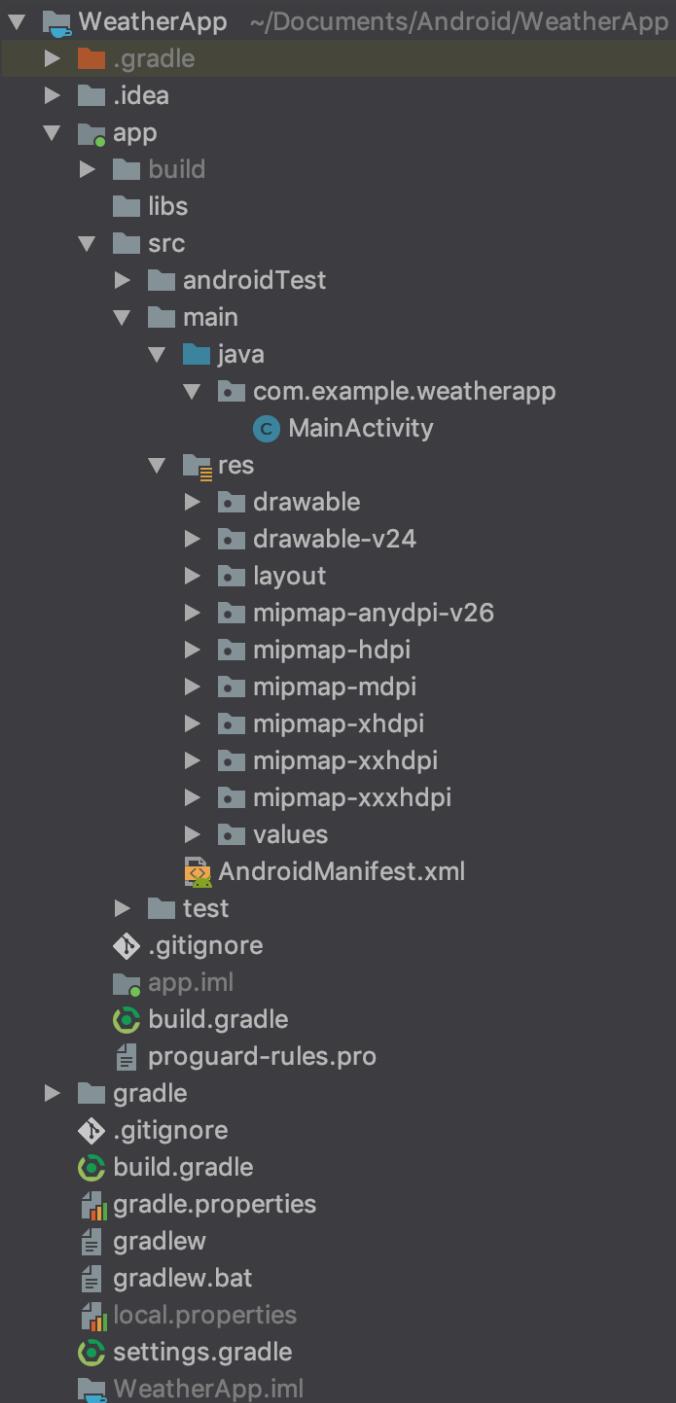
28

Versions	API Level	Version Code
Android 1.0	1	BASE
Android 1.1	2	BASE_1_1
Android 1.5	3	CUPCAKE
Android 1.6	4	DONUT
Android 2.0	5	ECLAIR
Android 2.0.1	6	ECLAIR_0_1
Android 2.1	7	ECLAIR_MR1
Android 2.2	8	FROYO
Android 2.3	9	GINGERBREAD
Android 2.3.3+	10	GINGERBREAD_MR1
Android 3.0	11	HONEYCOMB
Android 3.1	12	HONEYCOMB_MR1
Android 3.2	13	HONEYCOMB_MR2
Android 4.0	14	ICE_CREAM SANDWICH
Android 4.0.3	15	ICE_CREAM_SANDWICH_MR1

Versions	API Level	Version Code
Android 4.1	16	JELLY_BEAN
Android 4.2	17	JELLY_BEAN_MR1
Android 4.3	18	JELLY_BEAN_MR2
Android 4.4	19	KITKAT
Android 4.4W	20	KITKAT_WATCH
Android 5.0	21	LOLLIPOP
Android 5.1	22	LOLLIPOP_WATCH
Android 6.0	23	M
Android 7.0	24	N
Android 7.1	25	N_MR1
Android 8.0	26	O
Android 8.1	27	O_MR1
Android 9.0	28	P
Android 10.0	29	Q
Android 11.0	30	P

Architecture d'un projet

build.gradle : Configuration du build



Hello World :)

Sommaire

Android != Java

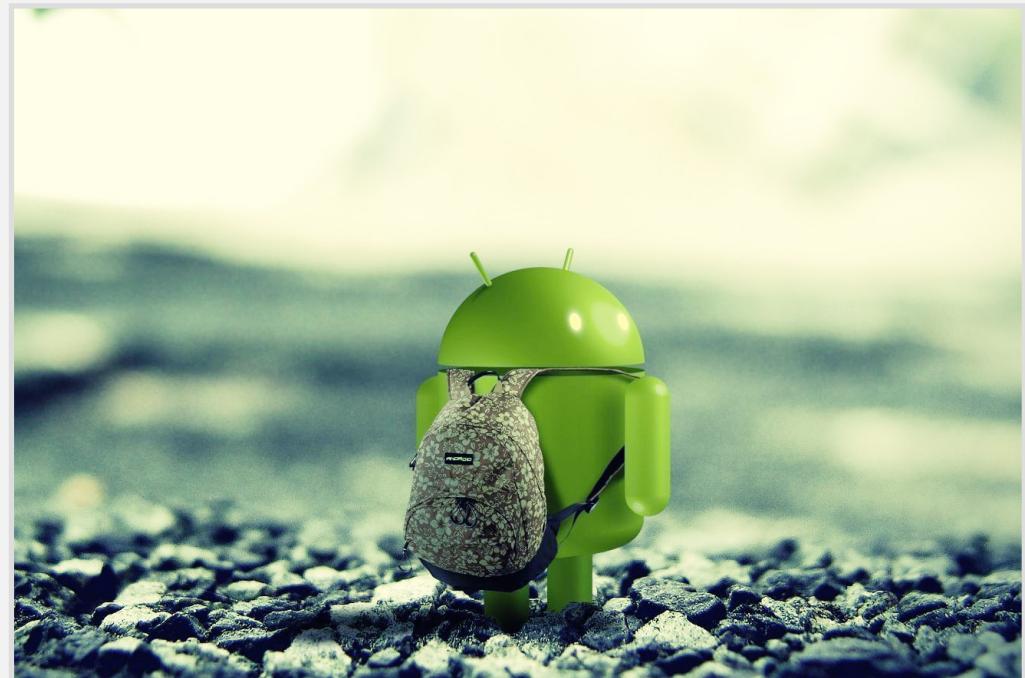
Quelques rappels Java

Architecture d'un projet

Afficher un message

Android Studio

Kotlin 😍



Afficher un message à l'utilisateur

[Android != Java](#)[Quelques rappels Java](#)[Architecture d'un projet](#)**Android Studio**[Kotlin](#)

31

Application visible ?

Application a déjà été ouverte ?

Information importante ?

Bloquer l'interface utilisateur ?

Notification

Notification

Toast

Dialog

Snackbar

Toasts

[Android != Java](#)[Quelques rappels Java](#)[Architecture d'un projet](#)**Android Studio**[Kotlin](#)

32

The screenshot shows a mobile application interface. At the top, there is a dark grey toast notification with white text that reads "No network connection!". Below this, the main content area has a teal background. On the left side of the teal area, there is some Java code:

```
Toast.makeText(context,  
        "No network connection!",  
        Toast.LENGTH_SHORT).show()
```

Below the code, the word "Toasts" is displayed in a large, bold, black font. Underneath "Toasts", the text "Afficher un message à l'utilisateur" is written in a smaller, black font. Further down, the text "La durée* peut-être définie à :" is followed by a bulleted list: "- courte (Toast.LENGTH_SHORT)" and "- ou longue (Toast.LENGTH_LONG)". At the bottom of the teal section, there is a small note in italicized black font: "*mais n'est pas configurable".

Snackbar

[Android != Java](#)[Quelques rappels Java](#)[Architecture d'un projet](#)**Android Studio**[Kotlin](#)

33

Mon message

```
Snackbar.make(  
    findViewById(android.R.id.content),  
    "No network connection!",  
    Snackbar.LENGTH_SHORT)  
.show()
```

Snackbar

Afficher un message à l'utilisateur (configurable)

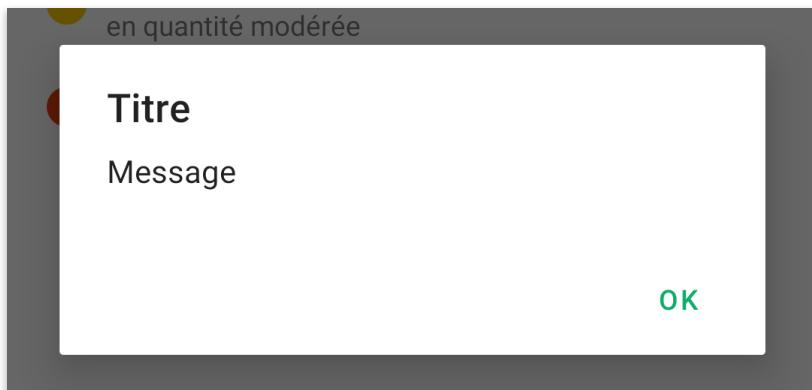
- La durée* peut-être définie à :
- courte (`Snackbar.LENGTH_SHORT`)
- ou longue (`Snackbar.LENGTH_LONG`)

*mais n'est pas configurable

Dialog

[Android != Java](#)[Quelques rappels Java](#)[Architecture d'un projet](#)**Android Studio**[Kotlin](#)

34



AlertDialog

Afficher un message à l'utilisateur avec :

- titre
- message
- un ou plusieurs boutons
(positif / négatif / neutre)

```
AlertDialog.Builder(context)
    .setTitle("Titre")
    .setMessage("Message")
    .setPositiveButton("Ok", {
        dialog, _ ->
        Log.d("BUTTON", "Bouton OK cliqué")
        dialog.dismiss()
    })
    .create()
    .show()
```

Logs



Android != Java

Quelques rappels Java

Architecture d'un projet

Android Studio

Kotlin

35

Type de log

Filtre par application

TAG

The screenshot shows the Android Studio Logcat window. On the left, there's a tree view of connected devices, with one entry for a Genymotion Samsung Galaxy S4. The main pane displays log entries from the application 'fr.g123k.myapplication'. The log entries are timestamped at 02-07 14:28:17.200 and show various OpenGL and EGL initialization messages. At the top of the Logcat window, there are filters for 'Log level: Verbose' and 'app: fr.g123k.myapplication'. A blue callout bubble points to the 'app:' filter with the text 'Filtre par application'. Another blue callout bubble points to the word 'fr.g123k.myapplication' in the app filter with the text 'TAG'.

```
02-07 14:28:17.200 3012-3012/fr.g123k.myapplication D/Angle: -90.0
02-07 14:28:17.200 3012-3012/fr.g123k.myapplication D/Angle: -45.0
02-07 14:28:17.200 3012-3012/fr.g123k.myapplication D/Angle: 0.0
02-07 14:28:17.200 3012-3012/fr.g123k.myapplication D/Angle: 45.0
02-07 14:28:17.200 3012-3012/fr.g123k.myapplication D/Angle: 90.0
02-07 14:28:17.200 3012-3012/fr.g123k.myapplication D/Angle: -90.0
02-07 14:28:17.200 3012-3012/fr.g123k.myapplication D/Angle: -45.0
02-07 14:28:17.200 3012-3012/fr.g123k.myapplication D/Angle: 0.0
02-07 14:28:17.200 3012-3012/fr.g123k.myapplication D/Angle: 45.0
02-07 14:28:17.200 3012-3012/fr.g123k.myapplication D/dalvikvm: Late-enabling CheckJNI
02-07 14:28:34.508 3092-3092/fr.g123k.myapplication D/MainActivity: 0
02-07 14:28:34.508 3092-3092/fr.g123k.myapplication D/MainActivity: 1
02-07 14:28:34.564 3092-3092/fr.g123k.myapplication D/libEGL: loaded /system/lib/egl/libEGL_genymotion.so
02-07 14:28:34.568 3092-3092/fr.g123k.myapplication D/: HostConnection::get() New Host Connection established 0xb93a1588, tid 3092
02-07 14:28:34.576 3092-3092/fr.g123k.myapplication D/libEGL: loaded /system/lib/egl/libGLESv1_CM_genymotion.so
02-07 14:28:34.584 3092-3092/fr.g123k.myapplication D/libEGL: loaded /system/lib/egl/libGLESv2_genymotion.so
02-07 14:28:34.628 3092-3092/fr.g123k.myapplication W/EGL_genymotion: eglSurfaceAttrib not implemented
02-07 14:28:34.632 3092-3092/fr.g123k.myapplication E/OpenGLRenderer: Getting MAX_TEXTURE_SIZE from GradienCache
02-07 14:28:34.644 3092-3092/fr.g123k.myapplication E/OpenGLRenderer: MAX_TEXTURE_SIZE: 16384
02-07 14:28:34.644 3092-3092/fr.g123k.myapplication E/OpenGLRenderer: Getting MAX_TEXTURE_SIZE from Caches::initConstraints()
02-07 14:28:34.644 3092-3092/fr.g123k.myapplication E/OpenGLRenderer: MAX_TEXTURE_SIZE: 16384
02-07 14:28:34.644 3092-3092/fr.g123k.myapplication D/OpenGLRenderer: Enabling debug mode 0
```

Android Studio

Logs

[Android != Java](#)[Quelques rappels Java](#)[Architecture d'un projet](#)**Android Studio**[Kotlin](#)

36

```
Log.i(TAG, "text") : info  
  
Log.d(TAG, "text") : debug  
  
Log.v(TAG, "text") : verbose  
  
Log.e(TAG, "text") : error  
  
Log.wtf(TAG, "text") : What A Terrible Failure
```

Logs

Le 1er argument est le tag, le second le message associé

Eviter les print() / println()

Hello World :)

Sommaire

Android != Java

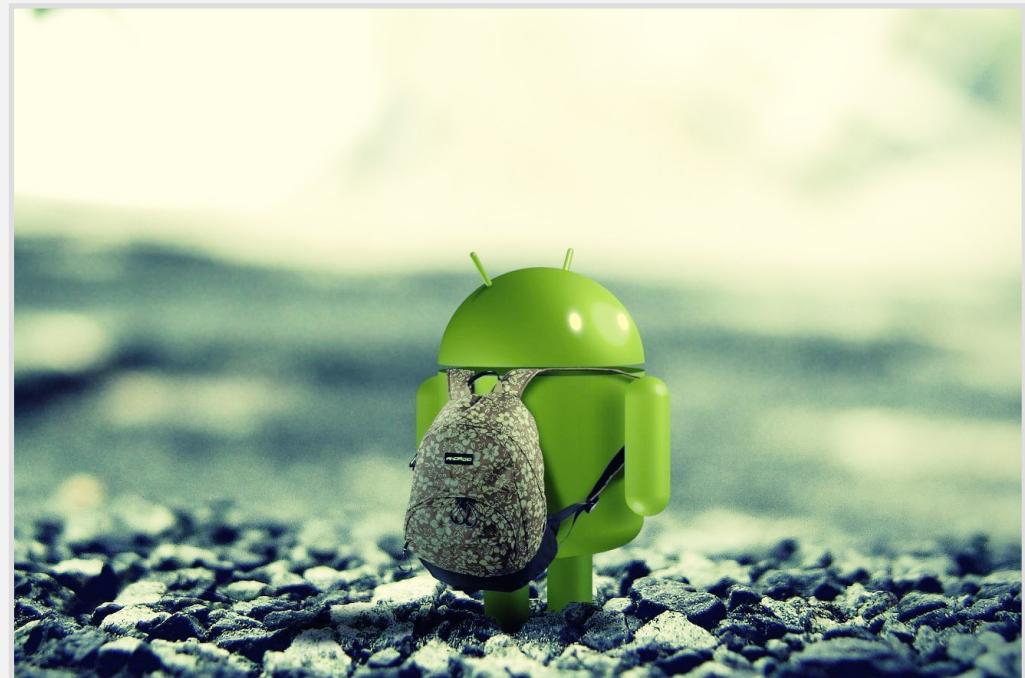
Quelques rappels Java

Architecture d'un projet

Afficher un message

Android Studio

Kotlin 😍



Android Studio : *cheat sheet*



Android != Java

Quelques rappels Java

Architecture d'un projet

Android Studio

Kotlin

38

Raccourcis (Mac)

- **Cmd + O** : recherche d'une classe
- **Cmd + Shift + O** : recherche d'un fichier
- **Shift (x2)** : rechercher dans tout le projet
- **Cmd + D** : dupliquer la ligne / sélection
- **Cmd + L** : aller à la ligne x
- **Ctrl + Tab** : basculer entre les fichiers
- **Cmd + Alt + I** : indenter le code
- **Cmd + Alt + D** : reformatter le xml
- **Ctrl + O** : liste des méthodes qui peuvent être *overridées*

Templates

- **const** : private final int VAR = xx;
- **fori** : for (int i = 0; i != ...; i++)
- **logd** : Log.d(TAG, "xx");
- La liste complète dans Preferences -> Editor -> Live Templates

Annotations

- **@Nullable** : la valeur peut être nulle
- **@NonNull** : la valeur ne peut pas être nulle
- **@Keep** : Ne pas obfuscuer
- **@IntRes** : Id d'une resource
- **@LayoutRes** : Id d'un layout (ressource)
- **@StringRes** : Id d'une string (ressource)
- **@IntDef** : liste de int (équivalent enum)

Créer un émulateur (Google)

Android != Java

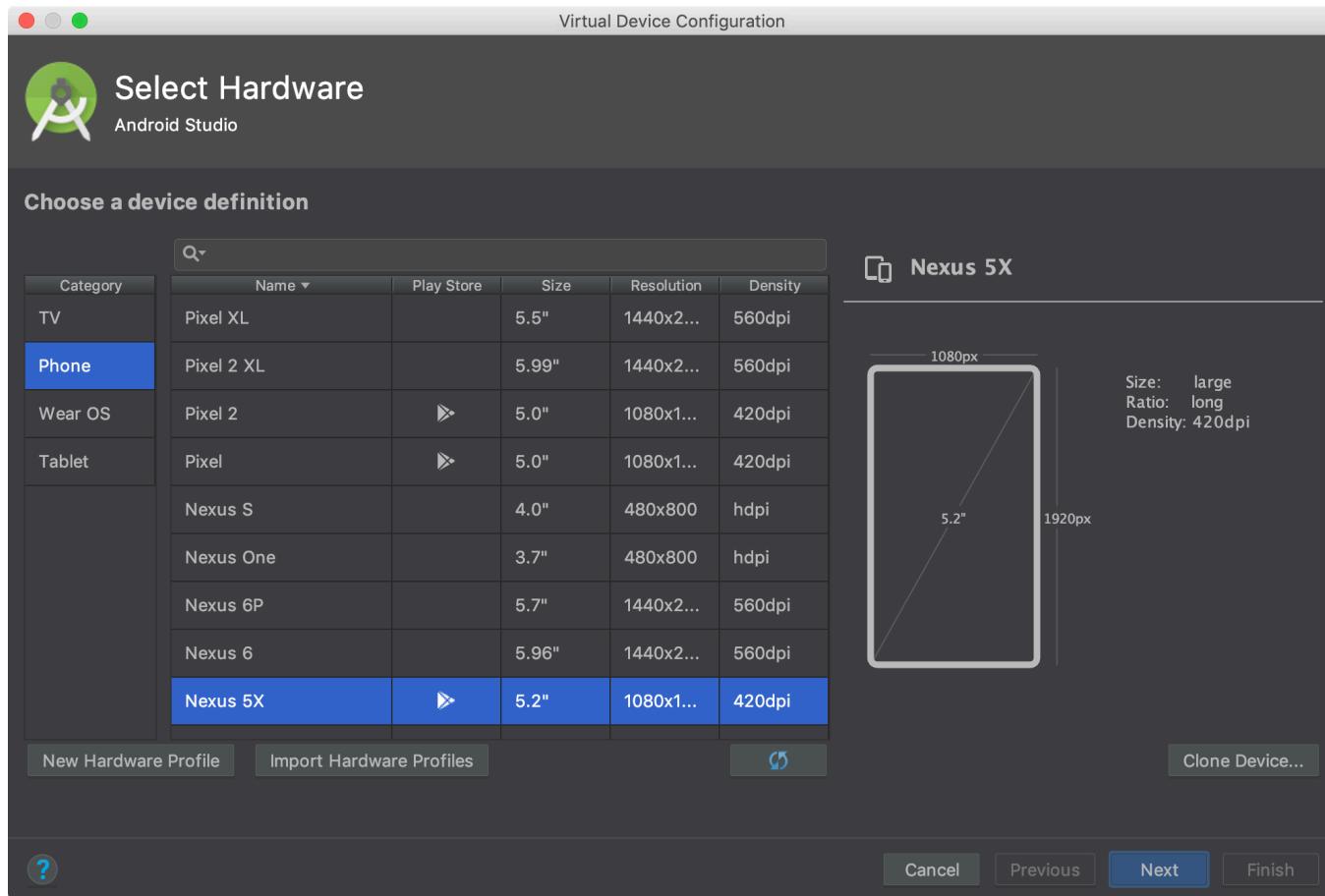
Quelques rappels Java

Architecture d'un projet

Android Studio

Kotlin

39



Etape 1 : Choisir le type et la taille du terminal

Créer un émulateur (Google)

Android != Java

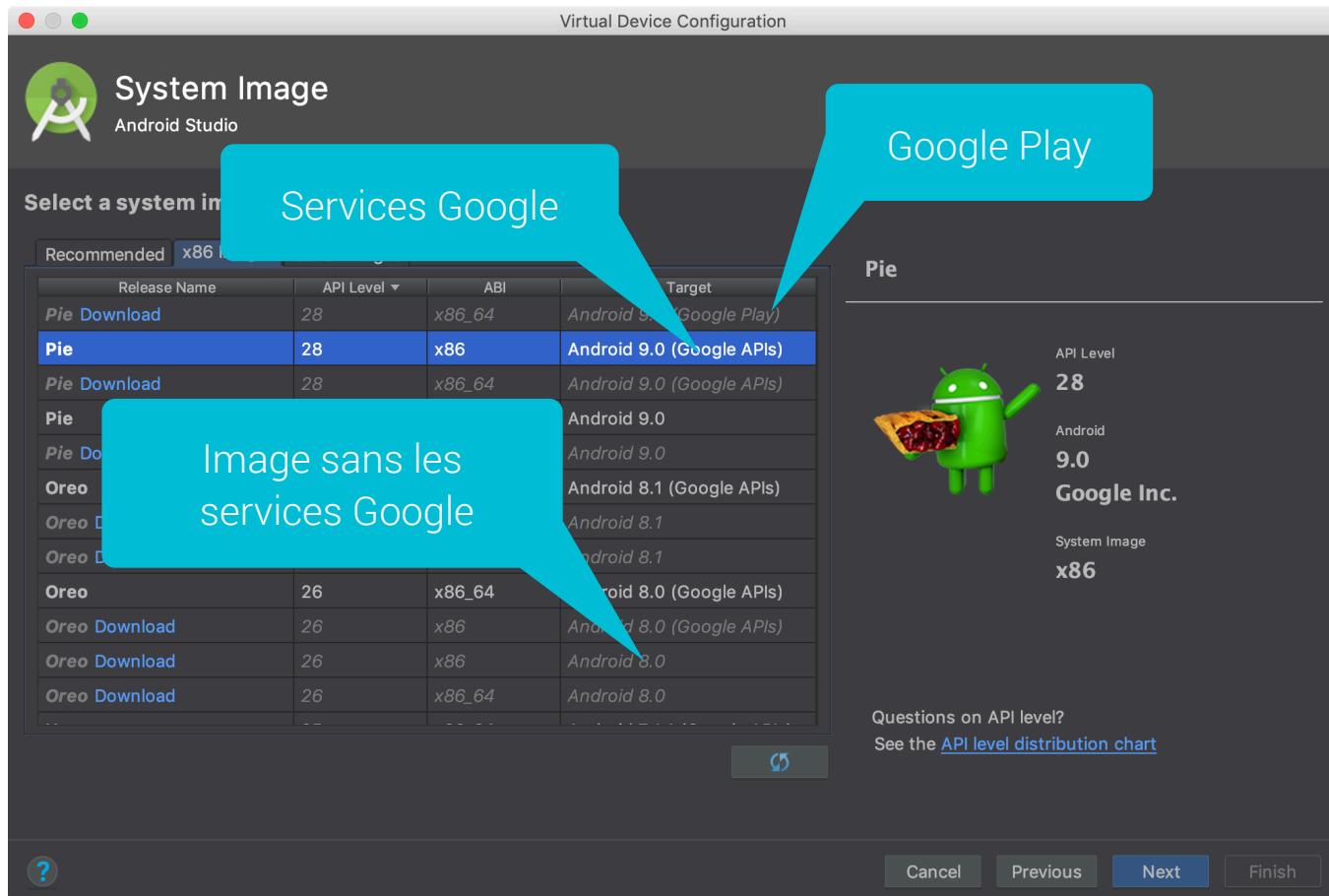
Quelques rappels Java

Architecture d'un projet

Android Studio

Kotlin

40



Etape 2 : Choisir le type d'image

Créer un émulateur (Google)

Android != Java

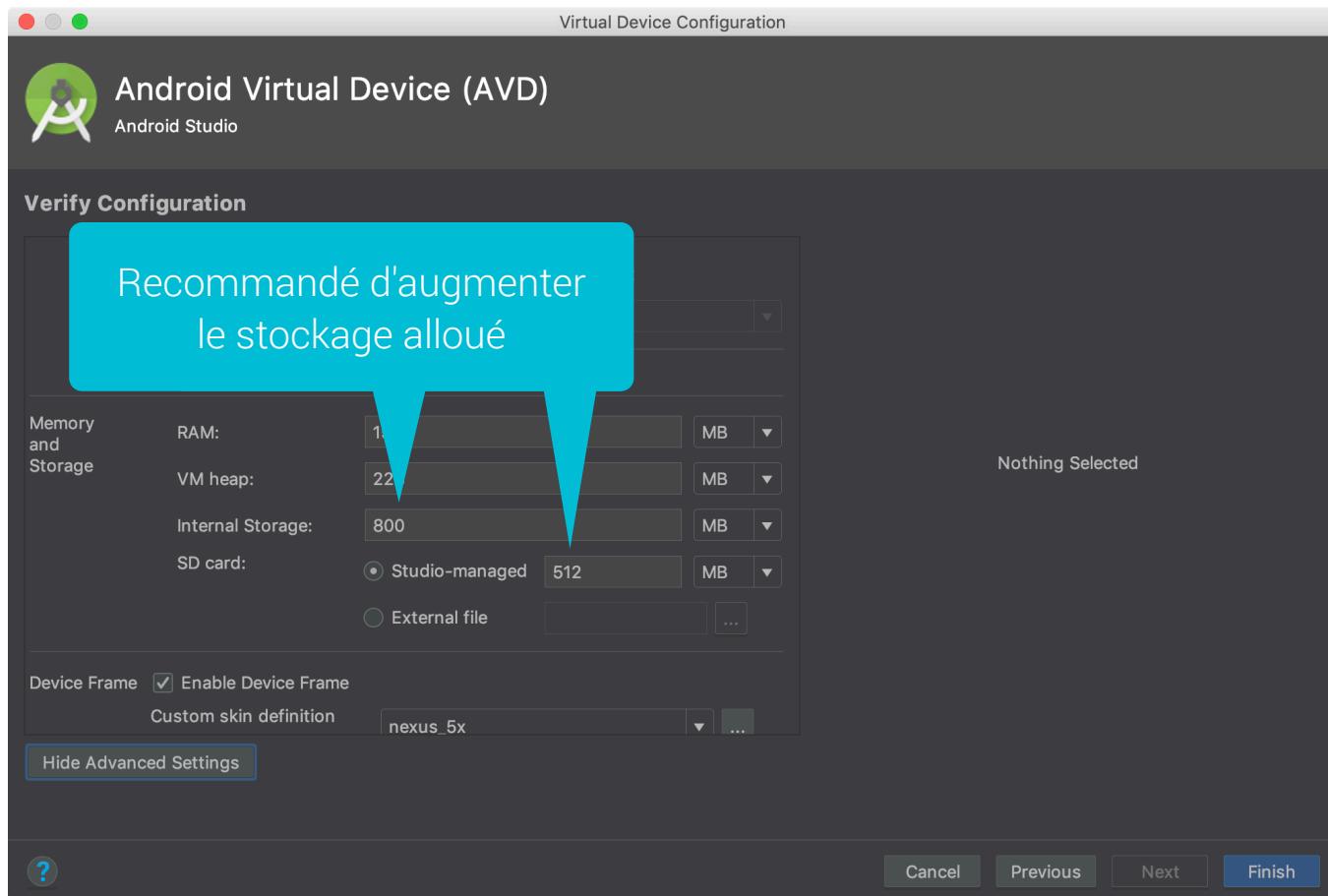
Quelques rappels Java

Architecture d'un projet

Android Studio

Kotlin

41



Etape 3 : Configurer l'environnement

Les options de l'émulateur (Google)

Android != Java

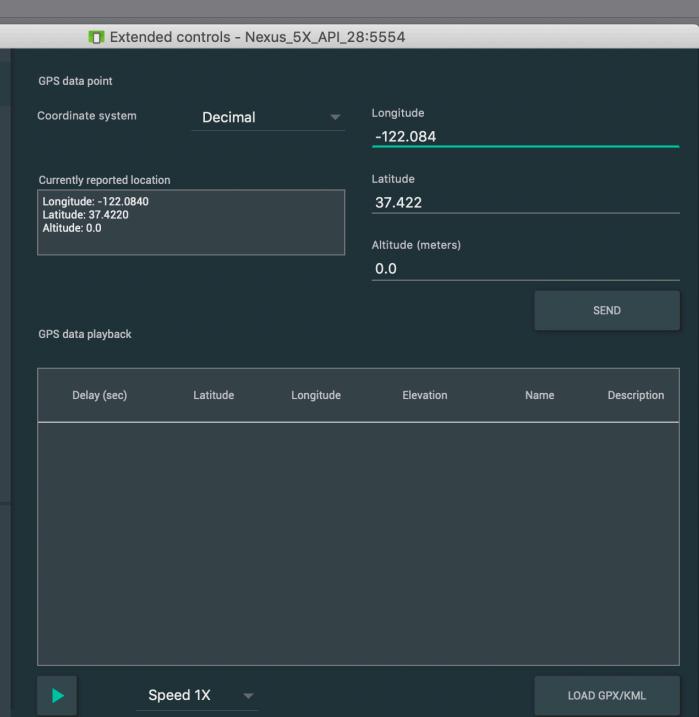
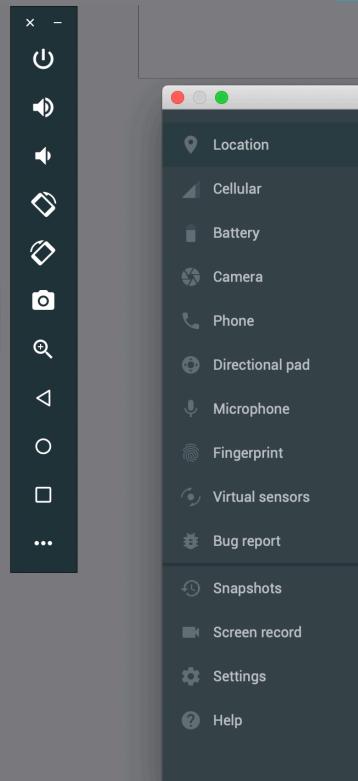
Quelques rappels Java

Architecture d'un projet

Android Studio

Kotlin

42



Hello World :)

Sommaire

Android != Java

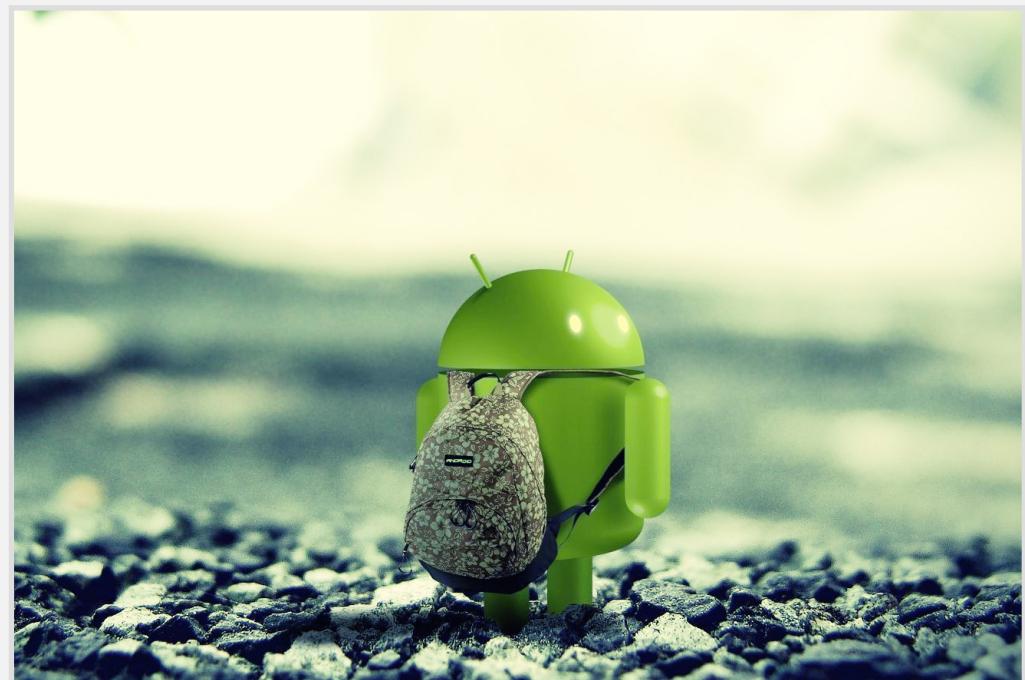
Quelques rappels Java

Architecture d'un projet

Afficher un message

Android Studio

Kotlin 😍

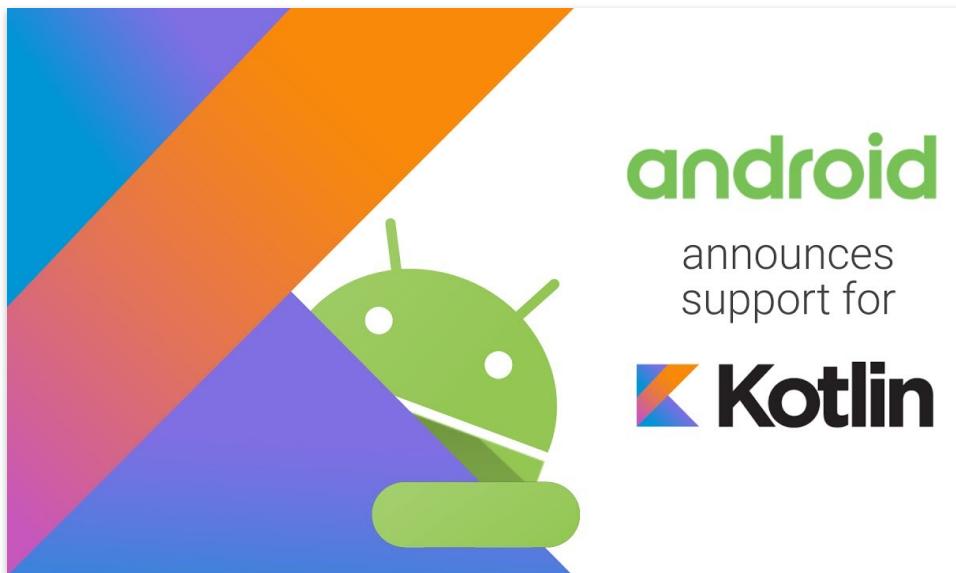


On peut aussi développer en utilisant du Kotlin

[Android != Java](#)[Quelques rappels Java](#)[Architecture d'un projet](#)[Android Studio](#)[Kotlin](#)

44

Kotlin



En mai 2017, Google annonce supporter le langage **Kotlin** (développé par JetBrains) en plus de Java - qui restera maintenu.

- ✓ Plus de *null checks*
- ✓ Inférence de type
- ✓ Extensions
- ✓ Casts intelligents
- ✓ Surcharge des opérateurs
- ✓ Peu cohabiter avec du code Java
- ✓ Support des coroutines
- ✓ Peu générer du code natif (Kotlin native)
- ✓ Support du Web : client & serveur (Kotlin for JS)

Les bases

Android != Java

Quelques rappels Java

Architecture d'un projet

Android Studio

Kotlin

45



Fichiers (*.kt)

Peut contenir plusieurs classes et des fonctions en dehors de toute classe

Packages

Similaire à ce que l'on fait en Java

Syntaxe

Point virgule à la fin des instructions optionnelles

Mutable/immutable

`val` : variable en lecture seule

`var` : variable modifiable à tout moment

Instantiation

Nul besoin du mot clé `new`, l'appel au constructeur suffit

```
package fr.g123k.helloworld

class MyClass1
class MyClass2

fun myMethod() : String { return "toto" }

fun variables() {

    var myVar1 = 0
    myVar1++

    val myVar2 = 0
    myVar2++
}
```

Un variable créée avec
`val` est immutable

KOTLIN

Un exemple de fichier Kotlin

Les types basiques

Android != Java

Quelques rappels Java

Architecture d'un projet

Android Studio

Kotlin

46



Uniquement des types

Il n'y a pas de type primitif comme en Java

Nombres

- Double (64) et Float (32)
- Long (64) et Int (32)
- Short (16) et Byte (8)
- Tous ont des méthodes .toInt() / toShort()

Caractères

- String
- Char

"Autres"

- Boolean
- Non signé :
 - kotlin.UInt / kotlin.ULong
 - kotlin.UByte / kotlin.UShort

```
// Pour une meilleure lisibilité,  
// on peut mettre une séparateur
```

```
val oneMillion = 1000000  
val oneMillion = 1_000_000
```

```
oneMillion.toDouble()
```

```
// Concaténation de String "abcdef"  
val concat : String = "abc" + "def"
```

```
// Variables dans une String  
val hello = "Hello $username!"
```

KOTLIN

Quelques exemples avec les types

La gestion de la "nullabilité" (1/2)

[Android != Java](#)[Quelques rappels Java](#)[Architecture d'un projet](#)[Android Studio](#)[Kotlin](#)

47



Dans le langage

Le langage force la gestion du null

? : peut être null

Un objet se terminant par un point d'interrogation peut être null

Safe call

Le point d'interrogation derrière un objet, une méthode ou un attribut, est similaire à un `if != null`

Ignorer la nullabilité

Si on est sûr qu'un objet n'est pas null, la notation `!!` permet de contourner la vérification du système

```
var a: String = "abc"
a=null
a.length
```

```
var b: String? = "abc"
b = null
b.length
```

```
if (b != null) b.length
b?.length

b!!.length
```

KOTLIN

Quelques exemples avec la gestion du null



La gestion de la "nullabilité" (2/2)

[Android != Java](#)[Quelques rappels Java](#)[Architecture d'un projet](#)[Android Studio](#)**Kotlin**

49



L'opérateur Elvis

Une valeur qui gère le null
Pratique pour assigner une valeur à une variable

```
val a : Int = if (b != null) ? b.length : 0  
  
val a = b?.length ?: 0
```



Safe cast

Pour prévenir les ClassCastException,
l'opérateur ? peut être utilisé pour les cast

```
val isInt : Int? = a as? Int
```

Le *when* est un *switch* amélioré

[Android != Java](#)[Quelques rappels Java](#)[Architecture d'un projet](#)[Android Studio](#)[Kotlin](#)

50



Switch

On utilise le mot clé `when` pour faire l'équivalent des `switch` en Java

Vérification

- Par valeur "unique"
- Par valeur "ou"
- Sur un intervalle (appartient ou non)
- Par type
- Appel à une méthode
- Else...

Return when

Tout comme le `return if`, peut être utilisé pour retourner une valeur

```
fun hasPrefix(x: Any) : String =  
  
    when (x) {  
  
        1 -> "cas $x"  
        2 -> "cas 2"  
        3,4 -> "cas 3 ou 4"  
        in 5..7 -> "cas 5, 6 ou 7"  
        !in 8..9 -> "cas différent de 8 ou 9"  
        is String -> "cas x est une String"  
        x.method() -> "cas où la method renvoie true"  
        else -> { print("else")  
                  "sinon" }  
  
    }  
  
}
```

Le égal "=" est l'équivalent d'un `return`

Que se passe-t-il lorsqu'on appelle `print(hasPrefix(true))` ?

KOTLIN

Un exemple d'une fonction retournant une valeur gérée par un `when`

Les fonctions

Android != Java

Quelques rappels Java

studio

Kotlin

```
method1("test")  
method1(param1 = "test")
```

```
method2()  
method2("test")  
method2(param1 = "test")
```



Fonctions/méthodes

Commencent par le mot clé **fun**

Valeur de retour

Si aucune valeur n'est renvoyée, on ne met pas de type de retour (~ **Unit**)

Le type de retour est renseigné après les éventuels paramètres

Paramètres

Le nom de la variable est avant le type

Valeur par défaut

Chaque variable peut avoir une valeur par défaut.

En cas d'absence, il est obligatoire de donner une valeur

```
// Méthode classique qui ne retourne rien  
fun method1(param1: String) { ... }  
fun method1(param1: String) : Unit { ... }
```

```
// Si le paramètre param1 n'est pas donné, il vaudra "hello"  
fun method2(param1: String = "hello") { ... }
```

```
// Méthode qui retourne une valeur  
fun method3(param1: String = "hello") : String {  
    return param1  
}
```

```
// Return if  
fun method4(param1: Int, p2: String): String {  
    return if (param1 > p2) param1  
    else param2  
}
```

```
val a = method3()  
val a = method3("test")  
val a = method3(param1 = "test")
```

Des exemples de fonctions en Kotlin

Les extensions de fonction + "top-level function"

Android != Java

Quelques rappels Java

Architecture d'un projet

Android Studio

Kotlin

52

```
fun String.isCool() = this.endsWith(".cool")  
  
"edouard@marquez.cool".isCool()
```

KOTLIN

Une extension de fonction pour les String

```
package fr.g123k.helloworld  
  
fun debug(msg: String) {  
    print("DEBUG: $msg")  
}
```

KOTLIN

Top-level function = fonction définie
en dehors de toute classe

```
import fr.g123k.helloworld.warning  
  
fun myMethod() {  
    warning("Object is not initialized yet")  
}
```

KOTLIN

Il suffit d'importer la fonction ensuite

Les classes

[Android != Java](#)[Quelques rappels Java](#)[Architecture d'un projet](#)[Android Studio](#)[Kotlin](#)

53



Créer une classe

Toute classe est identifiée par le mot clé `class` suivi de son nom

Constructeur

Le constructeur (principal) est défini directement dans l'entête (après `class XXX`).

Le constructeur donné dans le header ne contient pas de code, il faut passer par le bloc `init{}` pour cela
(il peut y en avoir de multiples)

Attributs

Les attributs (préciser `val/var`) peuvent être placés soit dans le header, soit directement dans le corps

Les getter/setter sont automatiquement créés, sauf si les attributs sont privés.
Ex: `"private val brand: String"`

```
class Car {}  
class Car  
  
class Car constructor(val brand: String, var model) {}  
class Car(val brand: String, var model) {}  
  
class Car(val brand: String, var model: String) {  
    init {  
        model = "Classe Car"  
    }  
}
```

Si on ne spécifie pas `val/var`, il s'agit simplement d'un paramètre d'un constructeur.

Sinon cela devient un attribut/membre de la classe

KOTLIN

Un exemple de création de classe en Kotlin

Les constructeurs d'une classe

[Android != Java](#)[Quelques rappels Java](#)[Architecture d'un projet](#)[Android Studio](#)[Kotlin](#)

54



Constructeurs secondaires

En plus du constructeur du header, on peut en créer d'autres dans le corps via le mot clé `constructor`

Il faudra toutefois et obligatoirement appeler le constructeur principal

```
class Car constructor(val brand: String, val model: String) {  
  
    constructor(brand: String) : this(brand,  
        when (brand) {  
            "Renault" -> "Clio"  
            else -> "Unknown"  
        }  
    )  
}
```



Constructeur principal optionnel

Si on supprime le constructeur principal (= du header), on peut tout de même créer un / des constructeurs secondaires

```
class Car1 {  
  
    val brand: String  
    val model: String  
    constructor(brand: String, model: String) {  
        this.brand = brand  
        this.model = model  
    }  
}
```

L'héritage

Android != Java

Quelques rappels Java

Architecture d'un projet

Android Studio

Kotlin

55



Méthodes

Pour que le comportement d'une méthode puisse être modifiée par la classe fille, il faut lui accorder le mot clé **open**

Du côté de la classe fille, il faut ajouter le mot clé **override**



Attributs

Sur les attributs, il faut également ajouter le mot clé **open** sur la classe parente.

Sur les attributs modifiés de la classe fille, il faut ajouter le mot clé **override**.

Pour une val -> getter

Pour une var -> getter et / ou setter

```
open class Car() { open fun maxSpeed: Int() = 100 }
```

```
class Clio : Car() {  
    override fun maxSpeed: Int() = 200  
}
```

```
open class Car(open val car: String)
```

```
class Clio : Car() {  
    override val brand = "Renault"  
    // ou  
    override val brand: String get() = "TODO"  
}
```

L'héritage

Android != Java

Quelques rappels Java

Architecture d'un projet

Android Studio

Kotlin

56

Héritage

En Kotlin, on a un héritage

Toutes les classes héritent de la classe Any

Pour qu'une classe puisse être héritée, il faut lui accorder le mot clé **open**



Apporte les méthodes :

- `toString()`
- `hashCode()`
- `equals(other: Any?)`

```
open class Car(val brand: String, val model: String)
```

```
class Clio : Car("Renault", "Clio")
```

Héritage et constructeurs multiples

On ne reprend que les constructeurs que l'on souhaite (+ on peut en créer de nouveaux)

On appelle ensuite le `super()`.



```
class Clio1 : Car1 {  
    constructor(brand: String, model: String) :  
        super("Renault", "Clio")  
}
```

Classe abstraite / interface

Android != Java

Quelques rappels Java

Architecture d'un projet

Android Studio

Kotlin

57



Classe abstraite

Peut avoir des méthodes et attributs

Ajouter le mot clé **abstract** devant la classe

Ajouter le mot clé **abstract** devant chaque
méthode abstraite

```
abstract class Car(val brand: String, val model: String) {  
  
    abstract fun maxSpeed(): Int  
  
}
```



Interface

Commence par le mot clé **interface**

Contient des fonctions avec ou non le
comportement implémenté

Contient des attributs non initialisés

```
interface Car {  
  
    val brand: String  
  
    fun maxSpeed()  
    fun category() = "A"  
  
}
```

En Java

Une classe avec deux attributs

```
public class JavaPerson {  
  
    private String firstName;  
    private String lastName;  
  
    public JavaPerson(String firstName, String lastName) {  
        this.firstName = firstName;  
        this.lastName = lastName;  
    }  
  
    public String getFirstName() {  
        return firstName;  
    }  
  
    public void setFirstName(String firstName) {  
        this.firstName = firstName;  
    }  
  
    public String getLastName() {  
        return lastName;  
    }  
  
    public void setLastName(String lastName) {  
        this.lastName = lastName;  
    }  
  
    @Override  
    public String toString() {  
        return "JavaPerson{" +  
            "firstName='" + firstName + '\'' +  
            ", lastName='" + lastName + '\'' +  
            '}';  
    }  
  
    @Override  
    public boolean equals(Object o) {  
        if (this == o) return true;  
        if (o == null || getClass() != o.getClass()) return false;  
        JavaPerson that = (JavaPerson) o;  
        return Objects.equals(firstName, that.firstName) &&  
            Objects.equals(lastName, that.lastName);  
    }  
  
    @Override  
    public int hashCode() {  
        return Objects.hash(firstName, lastName);  
    }  
}
```

```
data class Person(var firstName: String,  
                 var lastName: String)
```

En Kotlin

Une classe avec deux attributs

Les "data class"

[Android != Java](#)[Quelques rappels Java](#)[Architecture d'un projet](#)[Android Studio](#)[Kotlin](#)

60



Stocker des données

Raccourci pour les classes ne contenant que des données.

Implémente automatiquement les méthodes :

- equals()
- hashCode()
- toString()

Facilite la création de copies

```
class A(val param1: Int, val param2 : Int)

data class B(val param1: Int, val param2 : Int)

print(A(0, 0))                                A@27c170f0
print(B(0, 0))                                m2=0
print(A(0, 0) == A(0, 0))                      false
print(B(0, 0) == B(0, 0))                      true
print(B(0, 0).copy(param2 = 1))
```

B(param1=0, param2=1)

Les enums

Android != Java

Quelques rappels Java

Architecture d'un projet

Android Studio

Kotlin

61

```
enum class Position {  
    RIGHT, LEFT, TOP, BOTTOM  
}  
  
enum class Color(val rgb: String) {  
    RED("#FF0000"),  
    GREEN("#00F00"),  
    BLUE("#0000FF"),  
}  
  
print(Color.RED) -> RED  
print(Color.RED.rgb) -> "#FF0000"
```

KOTLIN

Les **enum** sont un *type spécial* de classe

also / apply / let / run

Android != Java

Quelques rappels Java

Architecture d'un projet

Android Studio

Kotlin

62

```
val length = text?.let {  
    println("Je veux afficher la longueur")  
    it.length  
} ?: 0
```

KOTLIN

Le **let** permet ici de récupérer
la longueur du texte

```
val length = text?.run {  
    println("Je veux afficher la longueur")  
    length  
} ?: 0
```

KOTLIN

Le **run** permet ici aussi de récupérer
la longueur du texte

```
val car = Car().also {  
    it.model = "Clio"  
    it.brand = "Renault"  
    it.color = Color.blue  
}
```

KOTLIN

Le **also** aide notamment à
l'initialisation d'objet

```
val car = Car().apply {  
    model = "Clio"  
    brand = "Renault"  
    color = Color.blue  
}
```

KOTLIN

Le **apply** aide notamment à
l'initialisation d'objet

also / apply / let / run

[Android != Java](#)[Quelques rappels Java](#)[Architecture d'un projet](#)[Android Studio](#)**Kotlin**

63

	Retourne l'objet	Retourne le résultat
it	also	let
this	apply	run

La visibilité

[Android != Java](#)[Quelques rappels Java](#)[Architecture d'un projet](#)[Android Studio](#)**Kotlin**

64

Membres d'une classe

**private**

Uniquement dans la classe

**protected**

private + visible dans les sous-classes

**internal**

A l'intérieur du module

**public**

Visible partout

La visibilité

[Android != Java](#)[Quelques rappels Java](#)[Architecture d'un projet](#)[Android Studio](#)**Kotlin**

65

Dans un package

(fonction, propriétés, classes...)

**private**

Uniquement dans le fichier

**internal**

A l'intérieur du module

**public**

Visible partout

Les collections



Android != Java

Quelques rappels Java

Architecture d'un projet

Android Studio

Kotlin

66



Tableau (Array)

```
Array<T> : tableau (immutable)
val a = arrayOf(1, 2, 3)
a[0] = 0

a.map { it + 2 }
    .filter { value -> value % 2 == 0 }
    .firstOrNull { it > 5 }
```



Map

```
Map<S, T> : Map immutable
mapOf(1 to 'a', 2 to 'b')
```

```
MutableMap<S, T> : Map mutable
mutableMapOf(1 to 'a').put(2, 'b')
```



Liste

```
List<T> : liste immutable
listOf(1, 2, 3)

MutableList<T> : list mutable
mutableListOf(1, 2).add(3)
```



Set

```
Set<S, T> : Set immutable
setOf(1, 2)
```

```
MutableSet<S, T> : Map mutable
mutableSetOf(1).add(2)
```