Année 2022-2023

Développement Mobile Android

Sébastien MOREIRA

PLAN

1. Présentation
2. OS Android
3. Android : Outils et plateforme
4. Kotlin
5. Android : Développement

1. Présentation

Android est un système d'exploitation open source initialement orienté pour les smartphones et tablettes, mais qui s'étend aujourd'hui aux montres connectées (Wear OS), voitures (Android Auto), smart tv (Android TV).

Plusieurs solutions existent pour développer des applications mobiles sous Android :

- **Flutter (cross plateforme)**
  Solution poussée par Google qui exploite le langage dart (développé par Google)
- **Unity (cross plateforme)**
  Solution orientée jeux vidéo
- **Xamarin (C#) / React Native (Javascript) (cross plateforme)**
- **Ionic / Web App (Hybride)**
- **Natif (Android uniquement)**
  C'est la solution qui sera étudiée dans ce cours.

1. Présentation

Les problématiques et avantages des solutions mobiles

- Les contraintes matérielles et environnementales
  - Puissance et batterie limités
  - Faible ressources
  - Accès au réseau variable
- Les avantages
  - Accès à l’information
  - Accès à un panel de capteurs
    - Micro, Caméras (front / back)
    - GPS
    - Position de l'appareil : accéléromètre, champ magnétique, gyroscope etc.
    - Lumière, Proximité, Température …
    - Spécifiques

1. Présentation
Quelle solution choisir ?

Applications hybrides :

+ Un seul code dans une techno “web” (JS, HTML, CSS)
+ Coût plus faible au lancement
- Interfaces limitées
- Utilisations des capteurs

Applications cross plateforme :

+ Même code pour toutes les plateformes
+ Coût plus faible au lancement
- Exploitation des capteurs plus difficile

Applications natives :

+ Expérience utilisateur plus fiable à chaque OS
+ Accès à l’ensemble des capteurs
+ Communauté plus importante sur les techno natives
+ Meilleurs performances
- Coût plus élevé au lancement

Quelques questions à se poser :

- Quelle est l'affinité des développeurs ?
- Quelles sont les compétences disponibles ?
- Quel est le budget ?
- L’utilisation des capteurs est-il nécessaire ?
- L’objectif est de tester un marché ?

2. OS Android

L’architecture du système Android

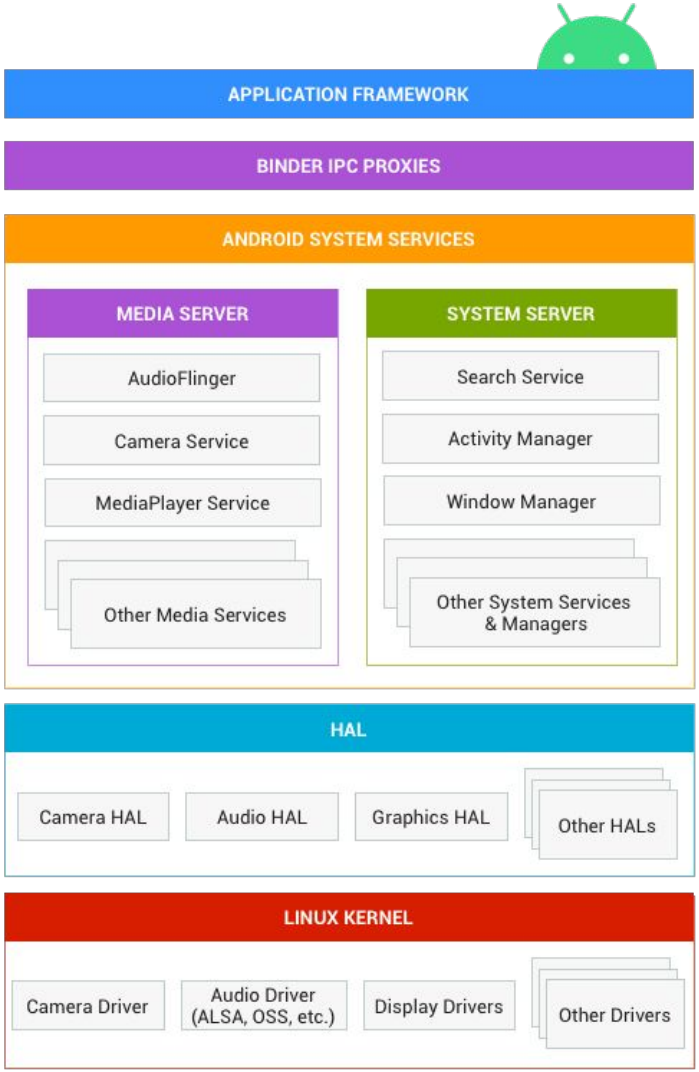
Application Framework: Le framework d'application est le plus souvent utilisé par les développeurs, c’est cette couche qui contient les applications

Binder IPC Proxies: Permet au framework d’application d'interagir avec les services système

Android System services: Les fonctionnalités exposées par les API du framework d'application communiquent avec les services système pour accéder au matériel sous-jacent. Android comprend deux groupes de services : système (tels que le gestionnaire de fenêtres et le gestionnaire de notifications) et multimédia (services impliqués dans la lecture et l'enregistrement des médias).

Hardware abstraction layer (HAL): Un HAL définit une interface standard que les fournisseurs de matériel doivent implémenter, ce qui permet à Android d'être agnostique quant aux implémentations de pilotes de niveau inférieur.

Linux kernel: Android utilise une version du noyau linux, optimisé pour l’embarquée mobile.



2. OS Android

Les versions d'Android

ANDROID PLATFORM VERSION	API LEVEL	CUMULATIVE DISTRIBUTION
4.1 Jelly Bean	16	
4.2 Jelly Bean	17	99,9%
4.3 Jelly Bean	18	99,7%
4.4 KitKat	19	99,7%
5.0 Lollipop	21	98,8%
5.1 Lollipop	22	98,4%
6.0 Marshmallow	23	96,2%
7.0 Nougat	24	92,7%
7.1 Nougat	25	90,4%
8.0 Oreo	26	88,2%
8.1 Oreo	27	85,2%
9.0 Pie	28	77,3%
10. Q	29	62,8%
11. R	30	40,5%
12. S	31	13,5%

Q

System

Foldables support
5G support
Gesture navigation
ART optimizations
Neural Networks API 1.2
Thermal API

User Interface

Smart Reply in notifications
Dark theme
Settings panels
Sharing shortcuts

Camera and media

Dynamic depth for photos
Audio playback capture
New codecs
Native MIDI API
Vulkan everywhere
Directional microphones

Security and privacy

New location permissions
Storage encryption
TLS 1.3 by default
Platform hardening
Improved biometrics

Last updated: August 4th, 2022

https://developer.android.com/about/versions/10

3. Android : Outils et plateforme

Langages

Les langage de développement sur android studio sont :

- **Java :** Initialement langage officiel pour le développement Android
- **Langage C / C++** avec l'utilisation du NDK Java
- **Kotlin :**
  - Annonce en 2017 de la volonté de Google de migrer vers ce langage
  - Adoption en 2019 comme langage officiel pour le développement Android

3. Android : Outils et plateforme

Android Studio

Android Studio est l'environnement de développement (**IDE**: Integrated Development Environment) officiel supporté par Google.

Il contient tous les outils nécessaires au développement d'une application Android.

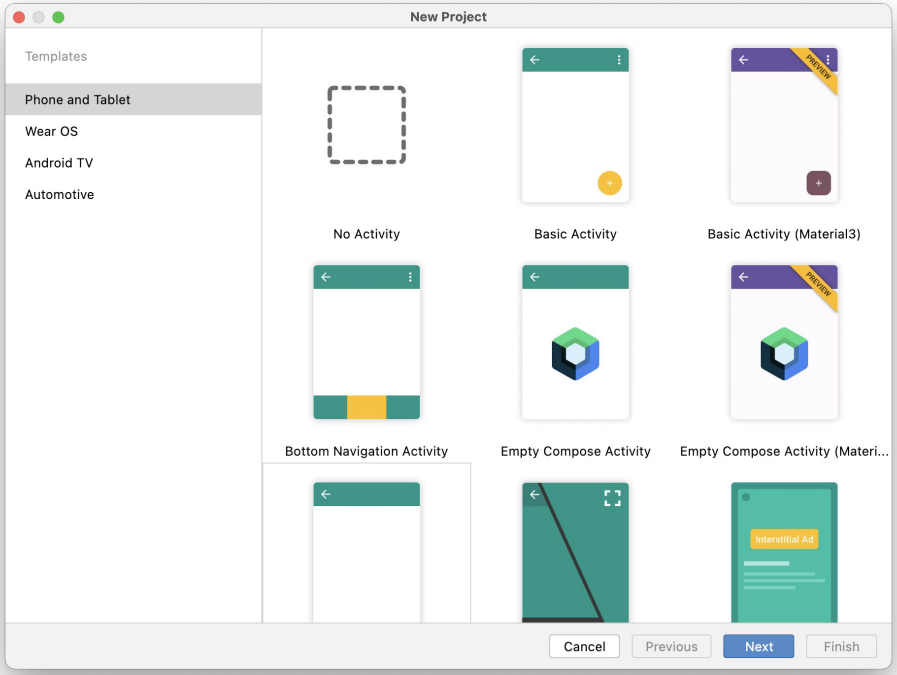
Quelques ressources pour l'installation d'Android Studio:

L'IDE est disponible ici : https://developer.android.com/studio

Le guide d'installation est disponible ici : https://developer.android.com/studio/install

3. Android : Outils et plateforme

Android Studio - Créer un projet

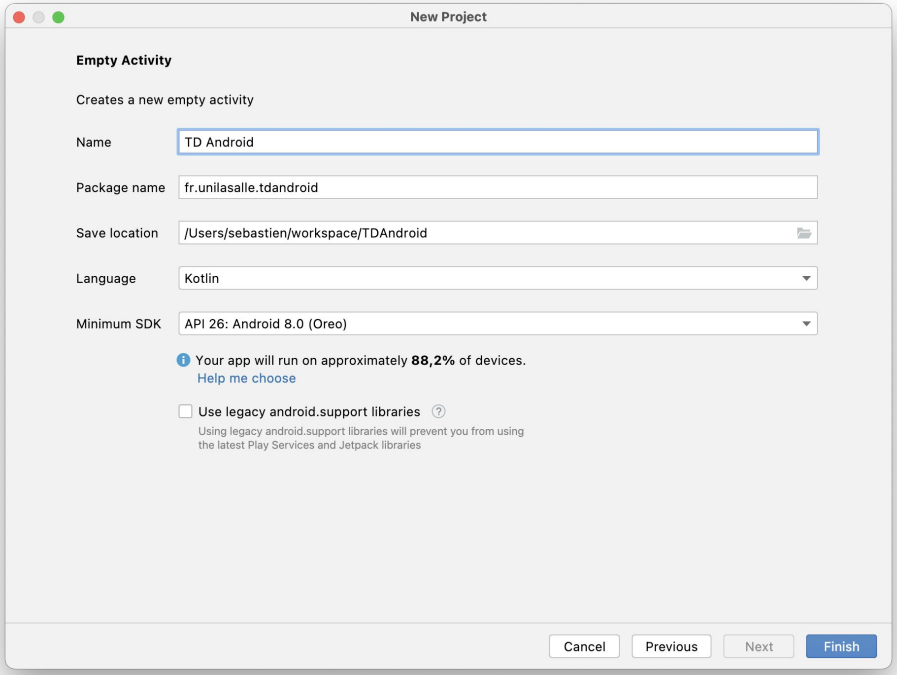


Pour créer un nouveau projet :

- Lancer **Android Studio**
- Cliquer sur **Create New Project**
- Choisir **Empty Activity** > **Next**

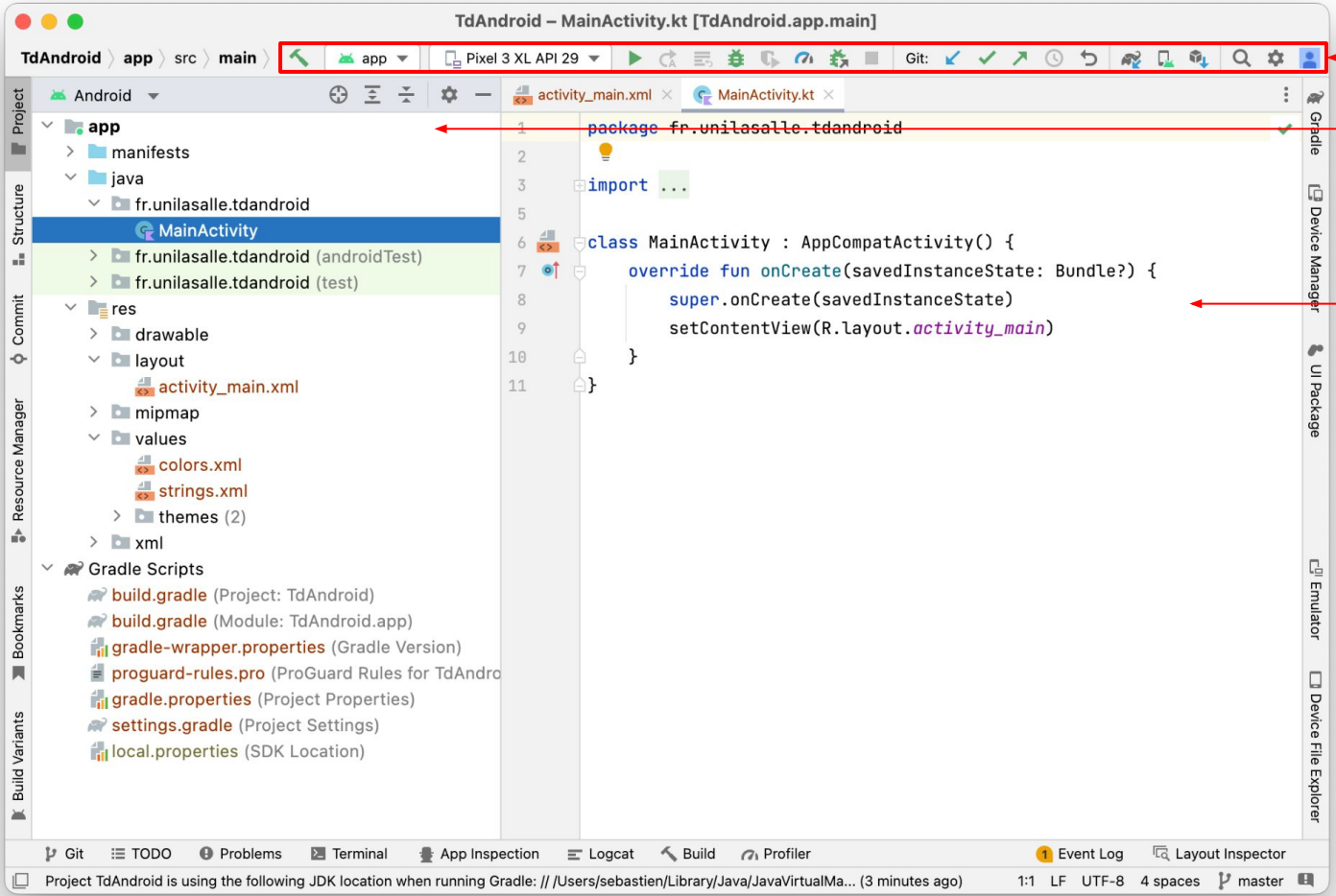
3. Android : Outils et plateforme

Android Studio - Créer un projet



Renseigner les informations du projet

- Son nom (ex: TDUnilasalleAndroid)
- Son nom de package (ex : [nom].unilasalle.td)
- Son emplacement
- Langage > **Kotlin**
- Minimum SDK > API 23 (1)
- Cliquer sur **Finish**



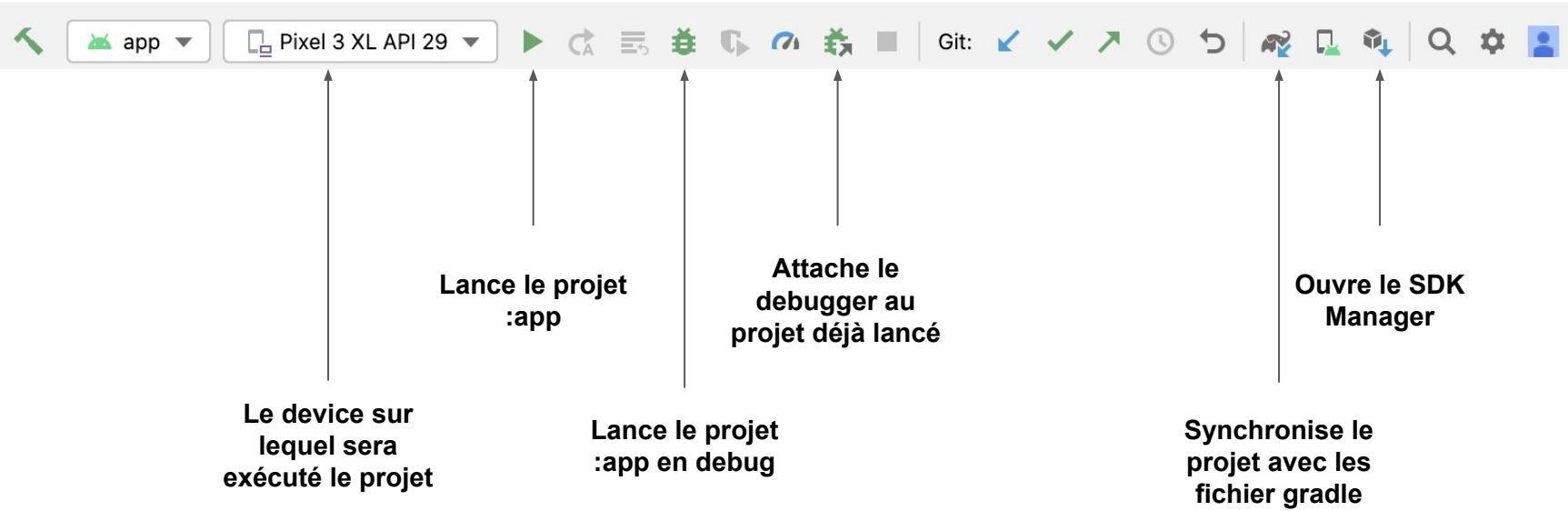
Barre d'outils

Arborescence de fichiers

Fenêtre d'édition

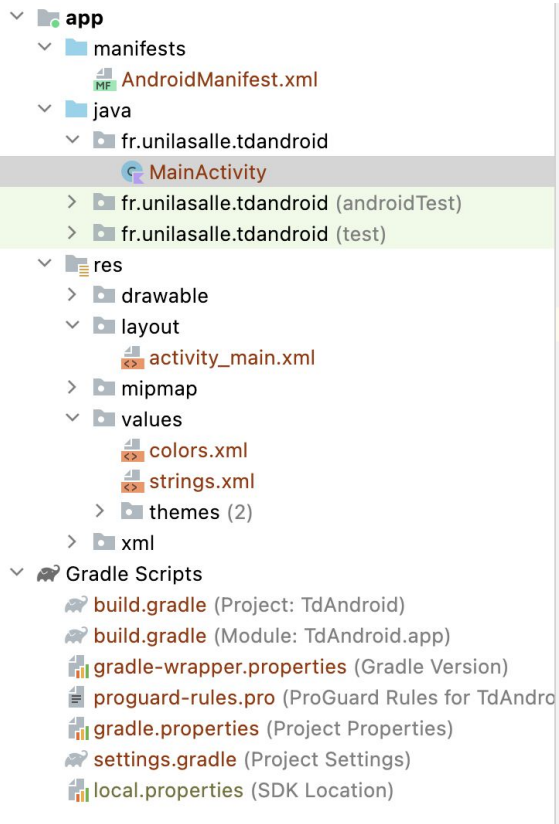
3. Android : Outils et plateforme

Android Studio - La barre d'outils



3. Android : Outils et plateforme

Android Studio - L'arborescence de fichiers



- **app** : racine du projet

- **manifests** : contient le fichier **AndroidManifest.xml**
  Permet de renseigner des informations sur l'application avant son exécution.

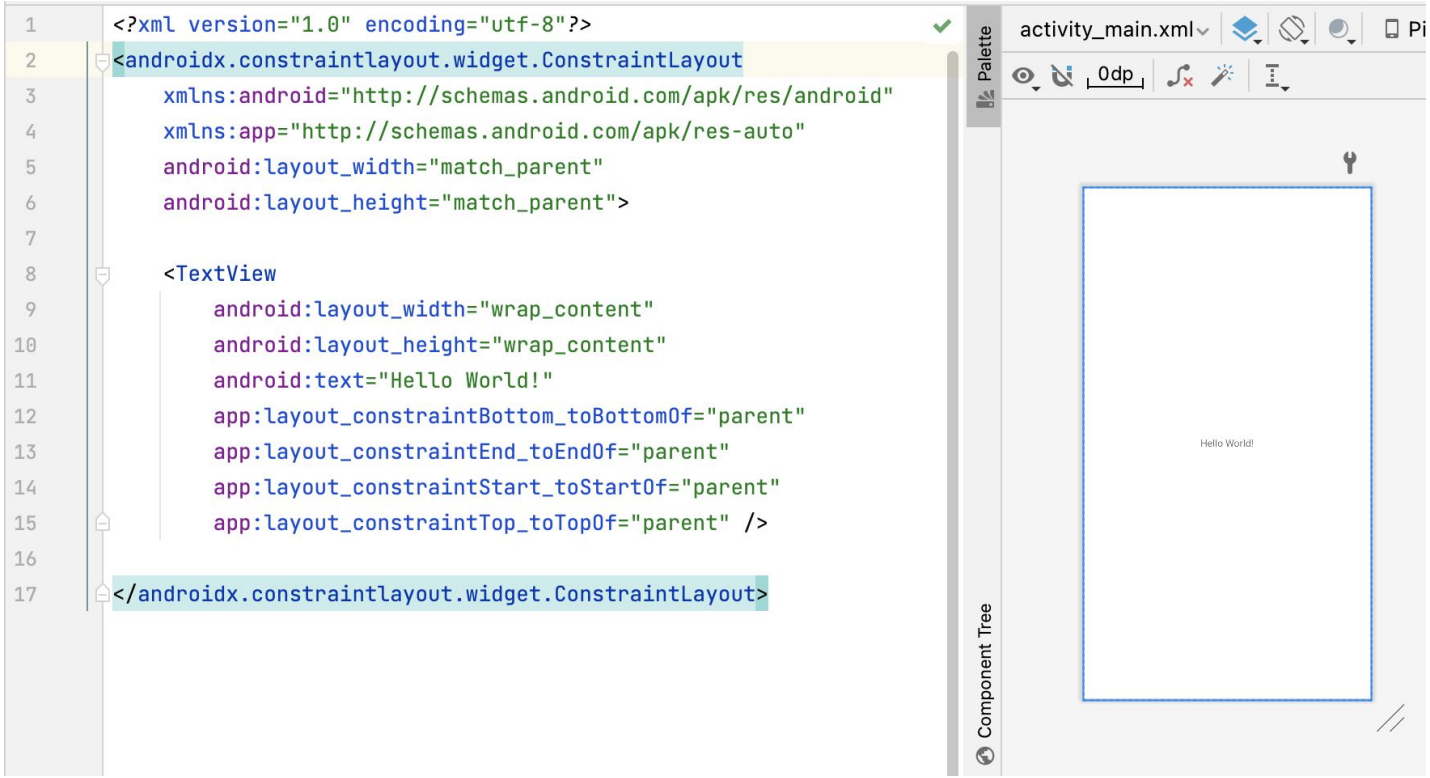
- **java** : contient le code Java et/ou Kotlin de l'application, ainsi que les tests unitaires

- **res** : contient les ressources de l'application
  - layout : les vues au format xml
  - drawables : les icônes et images
  - values
    - strings.xml
    - colors.xml
    - dimens.xml
    - les styles et thèmes

- **build.gradle (app)** :
  - contient les configurations de l'application
  - les dépendances et librairies utilisées dans l'application

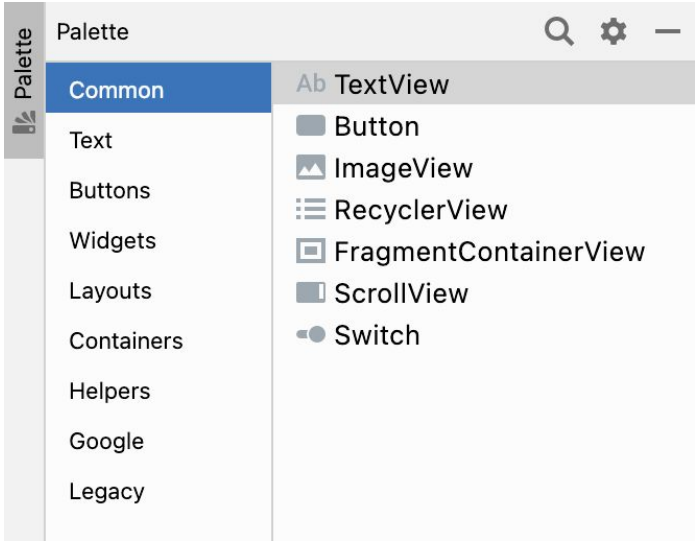
3. Android : Outils et plateforme

Les layouts - Edition d'un fichier XML



3. Android : Outils et plateforme

Les layouts - Palette de composants



Il existe un ensemble de composants à utiliser pour:

- organiser des vues:
  - ConstraintLayout
  - RelativeLayout

- Afficher des informations et interagir avec l'utilisateur:
  - TextView
  - Button
  - ImageView
  - RecyclerView

3. Android : Outils et plateforme

App components

Activités (Activities)

- classe *Activity*
- interface graphique pour l'utilisateur (UI) unique
- une activité peut envoyer, déclencher d'autres activités
- plusieurs points d'entrées, il n'y a pas de main()
- Exemple : entrées claviers, clic tactile sur écran, affichage

Services (Services)

- classe *Service*
- tâche en arrière plan sans interaction direct avec l'utilisateur (pas d'interface graphique, programme sans interface)
- Exemple : écouter de la musique, download, upload de gros fichiers

Fournisseurs de contenu (Content providers)

- classe ContentProvider
- fournisseur de contenus aux autres applications

3. Android : Outils et plateforme

App components

Écouteurs d’intention (Broadcast receivers)

- classe BroadcastReceiver
- Réagir à des messages systèmes ou à d’autres applications

Éléments d’interaction : Intent

- Pour lancer une autre activité au sein de l’application (explicit intent)
- Pour exécuter une action via une autre application (implicit intent)

3. Android : Outils et plateforme

Android Manifest

Toutes les applications disposent d'un AndroidManifest.xml.
Ce fichier contient :

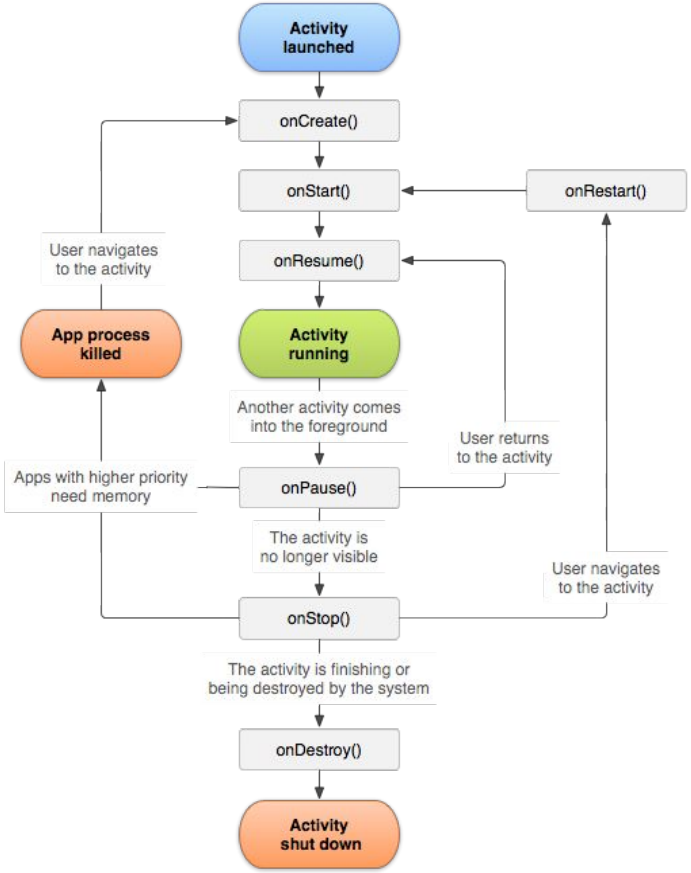
- Les informations propres à l'application
  - Nom de package
  - Nom de l'application
  - API cible
- Les app components
  - Activities
  - Services
  - BroadcastReceiver
  - Content providers
- Les permissions
  - Elles sont nécessaires pour réaliser certaines actions
    - Accéder à internet / la géolocalisation / etc.

3. Android : Outils et plateforme

Activities

Aucune méthode main()
Contrôlé par 7 méthodes de la classe ***android.app.Activity***

- **onCreate() :** Allocation de ressource, appelée quand l'activité est créée pour la première fois, c'est ici qu'on lie la vue à l'activité.
- **onStart() :** Affichage à l'écran, appelée lorsque l'activité devient visible pour l'utilisateur
- **onResume() :** Passage au premier plan, appelée quand l'activité commencera à interagir avec l'utilisateur
- **onPause() :** Passage en arrière plan, appelée quand l'activité n'est plus visible pour l'utilisateur
- **onStop() :** Arrêt de l'exécution, appelée quand l'activité n'est plus visible pour l'utilisateur
- **onRestart() :** Affichage de nouveau à l'écran, appelée quand l'activité est stoppée et redevient visible pour l'utilisateur
- **onDestroy() :** Libération des ressources, appelée avant que l'activité soit détruites



3. Android : Outils et plateforme

ViewModel

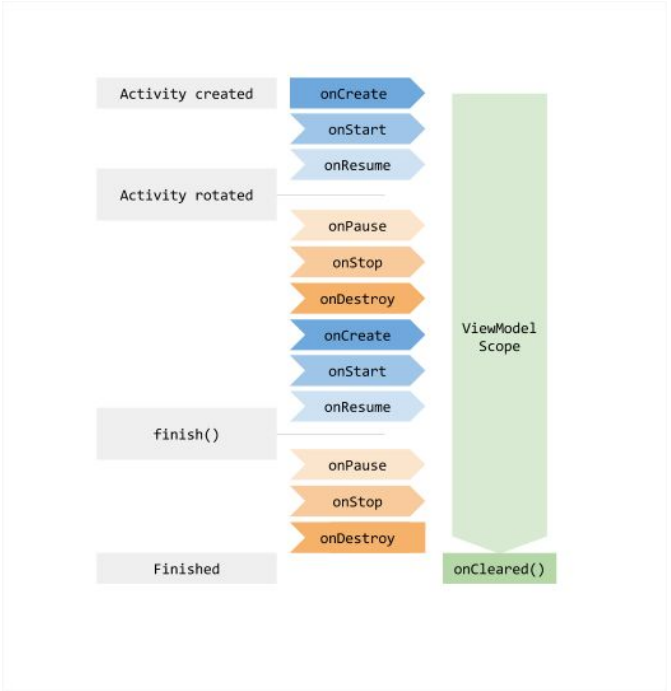
La classe ViewModel présente principalement deux avantages :

- Elle vous permet de conserver l'état de l'UI.
- Elle donne accès à la logique métier.

ViewModel permet la survie à la fois par l'état qu'il détient et par les opérations qu'il déclenche. Cette mise en cache évite d'avoir à récupérer à nouveau les données lors des modifications de configuration courantes comme la rotation de l'écran.

Un ViewModel reste en mémoire jusqu'à ce que le ViewModelStoreOwner auquel il s'applique disparaisse. Cela peut se produire dans les cas suivants :

- Dans le cas d'une activité, lorsqu'elle se termine.
- Dans le cas d'une entrée de navigation, lorsqu'elle est supprimée de la BackStack



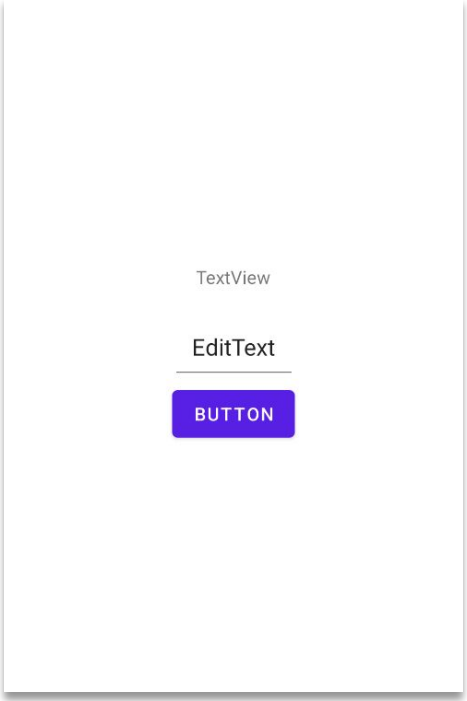
3. Android : Outils et plateforme

Views

Les Views sont des “blocks” à destination de l’interface utilisateur.

Ils permettent d’afficher ou de saisir des informations.

```
<androidx.constraintlayout.widget.ConstraintLayout xmlns:android="http://schemas.android.com,
    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <TextView
        android:id="@+id/line1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_weight="1"
        android:text="TextView"
        app:layout_constraintBottom_toTopOf="@+id/line2"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintHorizontal_bias="0.5"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent"
        app:layout_constraintVertical_chainStyle="packed" />

    <EditText
        android:id="@+id/line2"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_weight="1"
        android:text="EditText"
        app:layout_constraintBottom_toTopOf="@+id/button"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintHorizontal_bias="0.5"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toBottomOf="@+id/line1" />
```



3. Android : Outils et plateforme

ViewGroups

Un ViewGroup (sous classe de ***View***) est une vue pouvant contenir des Views ou des ViewGroups. Quelques exemples :

Framelayout : c'est la base la plus simple des mises en page Android. FrameLayout contient en général une seule vue.

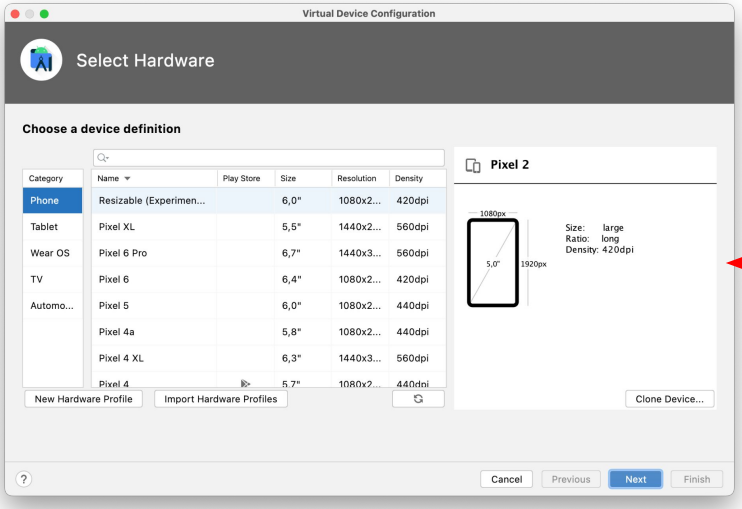
Linearlayout (horizontal) et Linearlayout (Vertical) : LinearLayout organise des vues dans une seule colonne ou une seule ligne. Les vues des enfants peuvent être agencées horizontalement ou verticalement, ce qui explique la nécessité de deux configurations différentes : l'une pour les lignes de vues horizontales et l'autre pour les colonnes de vues verticales.

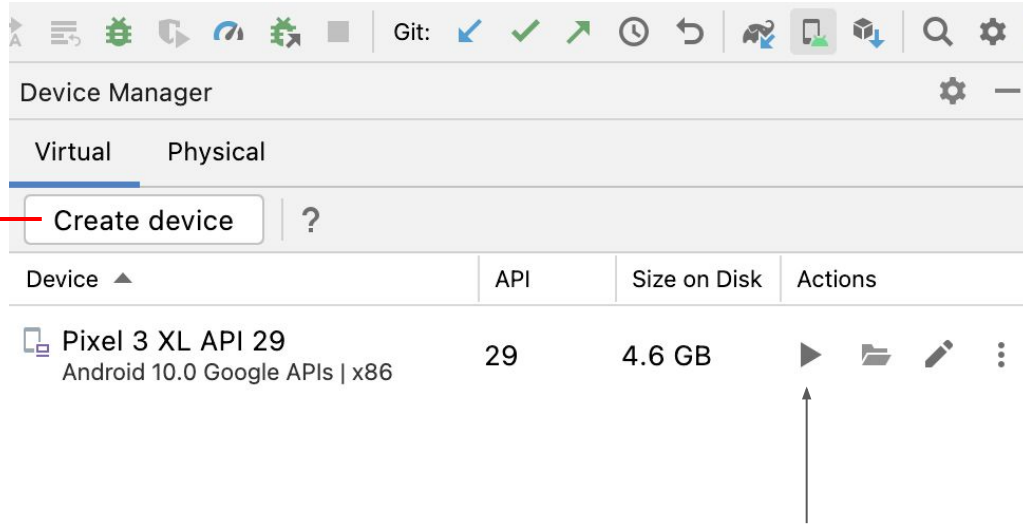
RelativeLayout : permet de disposer les vues enfants par rapport aux autres ou par rapport au parent.

ConstraintLayout : permet de créer des configurations complexes avec une hiérarchie de vue plate (pas de groupes de vues imbriqués), similaire à RelativeLayout dans la mesure où toutes les vues sont définies en fonction des relations entre les vues de frères et sœurs et la mise en page parentale, mais elle est plus flexible que RelativeLayout et plus facile à utiliser (disponible depuis API 9 et plus).
Il a été construit à partir de la base avec l'éditeur de mise en page, donc tout est accessible depuis l'éditeur de conception et vous n'avez jamais besoin d'éditer le XML à la main. Mieux encore, son système de disposition basé sur des contraintes vous permet de créer la plupart des mises en page sans groupes de vues imbriqués.

3. Développement mobile Android : Outils et plateforme

Les émulateurs





Lance l'émulateur

3. Android : Outils et plateforme

Les ressources

colors.xml

```xml
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <color name="purple_200">#FFBB86FC</color>
    <color name="purple_500">#FF6200EE</color>
    <color name="purple_700">#FF3700B3</color>
    <color name="teal_200">#FF03DAC5</color>
    <color name="teal_700">#FF018786</color>
    <color name="black">#FF000000</color>
    <color name="white">#FFFFFFFF</color>
</resources>
```

Les ressources sont accessibles via la classe R qui est générée automatiquement.

Exemples :

- R.id
- R.drawable
- R.string
- R.dimen
- etc.

values/strings.xml

Edit translations for all locales in the translations editor.

```xml
<resources>
    <string name="app_name">Starter</string>
    <string name="starter_mainScreen_hello_text">Hello</string>
</resources>
```

dimens.xml

```xml
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <dimen name="small_padding">6dp</dimen>
</resources>
```

3. Android : Outils et plateforme

Les ressources

R.id: Référence les ids des vues xml ayant un id enregistré.

```
<TextView
    android:id="@+id/line1"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
```

R.string: Référence les strings de l'application.

- <string/>: fournit un string unique
- <string-array/>: fournit une liste de string
- <plurals/>: fournit un string en fonction de la quantité demandée

```
<resources>
    <string name="starter_mainScreen_hello_text">Hello</string>

    <plurals name="qte_pieces">
        <item quantity="zero">pièce</item>
        <item quantity="one">pièce</item>
        <item quantity="few">pièces</item>
        <item quantity="many">pièces</item>
        <item quantity="other">pièces</item>
    </plurals>

    <string-array name="days">
        <item>Lundi</item>
        <item>Mardi</item>
        <item>Mercredi</item>
        <item>Jeudi</item>
        <item>Vendredi</item>
        <item>Samedi</item>
        <item>Dimanche</item>
    </string-array>
</resources>
```

3. Android : Outils et plateforme

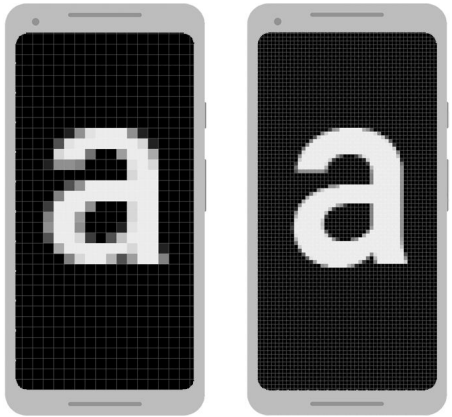
Les dimensions

Pour spécifier la taille d'un élément sur une interface utilisateur Android Studio, les unités de mesure suivantes :

- **px : pixel**
  1px correspond à 1 pixel sur l'écran sur l'écran.
  ⚠️ L'utilisation du pixel n'est pas recommandée, car l'interface utilisateur sera différente sur un device ayant une résolution d'écran différente.

- **dp : Density-independent pixel**
  *px = dp * (dpi / 160)*
  C'est l'unité de mesure recommandée lorsque vous spécifiez la dimension des vues dans votre mise en page car elle s'adapte à ma densité.
  One dp is a virtual pixel unit that's roughly equal to one pixel on a medium-density screen (160dpi; the "baseline" density). Android translates this value to the appropriate number of real pixels for each other density.

- **sp : Scale-independent pixel**
  1 sp est similaire à dp et est recommandé pour spécifier les tailles de police.



Deux écrans de même taille peuvent avoir un nombre différent de pixels

4. Kotlin

Depuis 2019, Android est devenu “Kotlin-first”, il devient le langage officiel supporté par Google pour développer des applications natives Android et remplace le Java.

Il est open-source et s’intègre aux standards de Java et de la JVM.

Avantages de Kotlin

- **Expressive and concise:** il permet une meilleure productivité.
- **Safer code:** le langage permet d’éviter les erreurs classiques du type null pointer exception.
- **Interoperable:** Le Kotlin est compatible avec le Java, la migration vers Kotlin peut se faire progressivement.
- **Structured Concurrency:** Les coroutines apportés avec Kotlin rendent les traitements asynchrone beaucoup plus simple qu’en Java.

Plus d’info sur Kotlin : https://kotlinlang.org/docs/basic-syntax.html

4. Kotlin

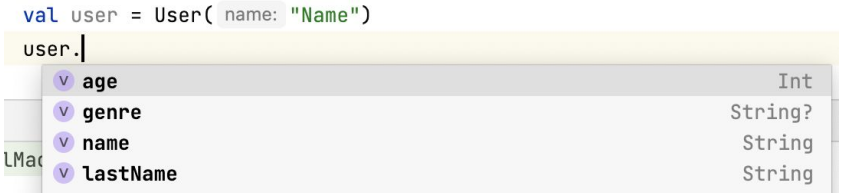
Classes

Une classe se définit avec le mot clé **class**.
Ses propriétés peuvent être listées directement dans la déclaration de la classe ou dans le corps de la classe.

```
class User(
    var name: String,
    var lastName: String,
    var age: Int
) {
    var genre: String? = null

    constructor(name: String) : this(name, lastName: "", age: 0) {}

    constructor(name: String, lastName: String, age: Int, genre: String) : this(name, lastName, age) {
        this.genre = genre
    }
}
```

4. Kotlin

class / data class / interface / abstract class / enum



La définition d'accesseurs (getter / setter) n'est plus obligatoire en Kotlin.

```kotlin
interface Forme {
    fun draw()
}

class Rectangle : Forme {
    override fun draw() {
    }
}
```

```kotlin
abstract class Forme {
    open fun draw() {
        //Do something
    }
}

class Rectangle : Forme() {
    override fun draw() {
        super.draw()
        //Do something
    }
}
```

```kotlin
data class Person(
    val name: String,
    val age: Int,
    val language: Language
)
```

```kotlin
enum class Language {
    FR, EN
}
```

4. Kotlin

Variables

Toutes les variables se déclarent à l'aide des termes ***var*** et ***val***.
Une variable déclarée à l'aide de ***var*** est une variable modifiable.
Une variable déclarée à l'aide de ***val*** est une variable finale et n'est pas modifiable.

```
fun main() {
    var maVariable: String = "Hello world"

    maVariable = "Another Value"
}
```

```
fun main() {
    val maVariable: String = "Hello world"

    maVariable = "Another Value"
}
```

4. Kotlin

Fonctions

Déclaration d'une fonction

Une fonction en kotlin est déclarée avec le mot clé **fun**.

Dans cet exemple, la fonction prend deux ***Double*** en paramètres et retourne un ***Double*** comme résultat.

```
fun powerOf(number: Double, exponent: Double): Double {...}
```

Appel d'une fonction

```
val result = powerOf( number: 2.0,  exponent: 2.0)
```

Default arguments

Dans cet exemple, des valeurs par défaut sont attribuées aux paramètres.

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

```
fun main() {
    powerOf() //result = 1
    powerOf( number: 2) //result = 2
    powerOf( number: 2,  exponent: 2) //result = 4
}
```

4. Kotlin

Fonctions

Named arguments

Les named arguments permettent de clarifier le code et lorsqu'ils sont utilisés l'ordre des paramètres importe peu.

```
fun main() {
    powerOf(number = 2)//result = 2
    powerOf(exponent = 2)//result = 1
    powerOf(number = 2, exponent = 2)//result = 4
    powerOf(exponent = 2, number = 2)//result = 4
}
```

Unit-returning functions

Lorsqu'une fonction ne retourne rien, elle est de type **Unit**. Dans ce cas, il n'est pas obligatoire de le spécifier.

```
fun executeSomeCode() { /*...*/ }
fun executeSomeCode(): Unit { /*...*/ }
```

Single-expression functions

Dans le cas de Single-expression functions les accolades peuvent être remplacées par un **=**, comme pour les variables ou une fonction de type Unit, le type est inféré, il n'est pas obligatoire de le spécifier.

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

4. Kotlin

Types et Inférence de type

En kotlin, il n'existe pas de primitives a proprement parler comme en Java, mais il existe des objets qui s'en rapprochent:

- Boolean
- String
- Int
- Long
- Float
- …

Bien que kotlin soit un langage typé, il n'est pas toujours nécessaire de spécifier le type d'une variable ou d'une fonction. Le compilateur est capable de déduire le type de la variable, ainsi :

```
val maVariable: String = "Hello world"
```

revient au même que :

```
val maVariable = "Hello world"
```

4. Kotlin

Types et Inférence de type

Pour la déclaration de nombres, l'inférence de type n'est pas aussi évident pour le compilateur, il faut l'aider.
Par exemple, un nombre, s'il est suivi d'un **f** sera un **Float** et sera reconnu par Android comme tel sans avoir à spécifier le type.
D'autres cas sont possibles :

```
var int = 1 //Int
var float = 1f //Float
var double = 1.0 //Double
var long = 1L //Long
```

4. Kotlin

Les listes

Les listes peuvent prendre tout type d'objets en paramètre.

```
//La liste est figée, on ne peut pas ajouter d'éléments
val list: List<String> = listOf("apple", "banana", "kiwifruit")

//La liste peut être modifiée
val arraylist: ArrayList<String> = arrayListOf("apple","banana", "kiwifruit")
arraylist.add("pear")
```

Il est également possible de réaliser des opérations sur les listes :

```
list.filter { it.startsWith( prefix: "a") }
    .sortedBy { it }
    .map { it.uppercase() }
    .forEach { println(it) } //Result = APPLE
```

Ici, on filtre les éléments de la liste commençant par la lettre "a", puis on les trie par ordre alphabétique, et enfin affiche le résultat en majuscule.

4. Kotlin

Les boucles et les conditions

On retrouve en Kotlin les mêmes boucles et conditions qu'en Java.

La boucle **for**:

```
val items: List<String> = listOf("apple","banana", "kiwifruit")
for(item in items){
    println(item)
}
```

Tout comme en Java, il est possible d'utiliser les boucles **while**, et **do…while**.

If expression:

```
val person = Person(name = "Nom", age = 30, language = Language.FR)

if (person.language == Language.FR) {
    println("Vous parlez français !")
} else {
    println("You speak another language !")
}
```

4. Kotlin

Les boucles et les conditions

when expression:

```
val person = Person(name = "Nom", age = 30, language = Language.FR)

when(person.language) {
    Language.FR ->  println("Vous parlez français !")
    Language.EN -> println("You speak english !")
    else -> println("Vous parlez un autre langage !")
}
```

4. Kotlin

Les lambdas

Kotlin vient avec la possibilité de passer des fonctions en paramètre d'autres fonctions.

```kotlin
fun main() {
    calculate({ 2.0 * 2.0 }) //4.0
    calculate { 2.0 + 2.0 } //4.0
    calculate { powerOf( number: 2.0,  exponent: 4.0) } //16.0
}

fun calculate(operation: () -> Double) {
    println("Le resultat est : ${operation()}")
}
```

4. Kotlin

Les coroutines

Les coroutines sont une manière de lancer du code pour qu'il soit non bloquant et asynchrone.

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

    lifecycleScope.launch(Dispatchers.Main) {  this: CoroutineScope
        waitAction()
    }
}

suspend fun waitAction() {
    repeat( times: 10) {  it: Int
        delay( timeMillis: 1000)
        println("Thread launched ... ")
    }

    println("Thread ended ...")
}
```

launch permet de lancer la coroutine.
suspend définit une fonction ayant une action asynchrone.

5. Android : Développement

Activité et interaction

ℹ️ On n'est pas obligé d'override toutes les fonctions du cycle de vie

MainActivity.kt

```kotlin
package fr.unilasalle.starter

import android.os.Bundle
import androidx.appcompat.app.AppCompatActivity

class MainActivity : AppCompatActivity() {

    override fun onCreate(savedInstanceState: Bundle?) {...}

    override fun onStart() {...}

    override fun onResume() {...}

    override fun onPause() {...}

    override fun onStop() {...}

    override fun onDestroy() {...}

    override fun onRestart() {...}

}
```

Attacher une vue à une activity

Solution 1 :

```kotlin
lateinit var line1: TextView

override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    setContentView(R.layout.activity_main)

    line1 = findViewById(R.id.line1)
    line1.text = "New Text Value"
}
```

Solution 2 :

build.gradle (:app)

```
android {
    //...

    buildFeatures {
        viewBinding = true
    }

    //...
}
```

```kotlin
private lateinit var binding: ActivityMainBinding

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

    binding.line1.text = "New Text Value"
}
```

5. Android : Développement

Interagir avec la vue

Réagir au clic sur un bouton

```
binding.button.setOnClickListener {  it: View!
    //Do Something
}
```

Modifier le texte d'un TextView

```
binding.line1.text = "New text value"
binding.line1.text = resources.getString(R.string.starter_mainScreen_hello_text)
binding.line1.text = resources.getQuantityString(R.plurals.qte_pieces,  quantity: 4)
```

On utilise la valeur définie dans le fichier strings.xml

Etre notifié d'un changement de texte

```
binding.myEditText.doOnTextChanged { text, start, before, count ->

}
```

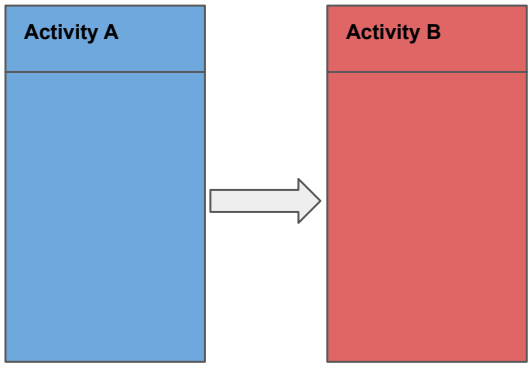
5. Android : Développement

Navigation

Une application peut être constituée de plusieurs Activities. Il est nécessaire de pouvoir naviguer et communiquer entre celles-ci. Pour naviguer d'une Activity à une autre, nous utilisons des **Intents**.

Deux cas se présentent :

- Transmission de données d'une activity A vers une activity B uniquement




```
val intent = Intent(applicationContext, SecondActivity::class.java)
startActivity(intent)
```

ℹ️ Il est possible d'utiliser les extras pour transmettre des données d'une Activity à une autre.

```
val intent = Intent(applicationContext, SecondActivity::class.java)
intent.putExtra("key", value)
startActivity(intent)
```

Les données se récupèrent depuis SecondActivity de cette manière :

```
intent.extras?.getString("key")
intent.extras?.getBoolean("key")
intent.extras?.getLong("key")
```

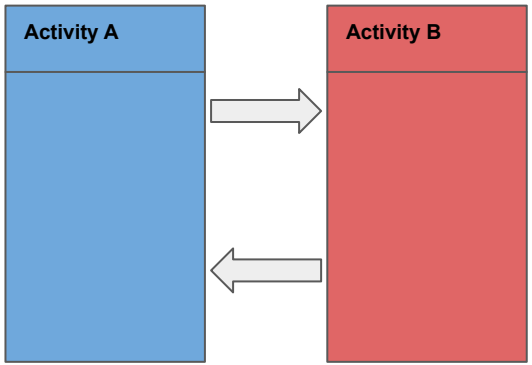
5. Android : Développement

Navigation

- Transmission de données d'une activity A vers une activity B puis transmission de données de B vers A

```
private val getResult = registerForActivityResult(
    ActivityResultContracts.StartActivityForResult()
) { it: ActivityResult!
    if (it.resultCode == RESULT_OK) {
        val returnValue = it.data?.getStringExtra( name: "return_value")
        //Do something with the returned value
    }
}
```

```
val intent = Intent( packageContext: this, SecondActivity::class.java)
getResult.launch(intent)
```



Les données se récupèrent depuis SecondActivity de la même manière que précédemment.

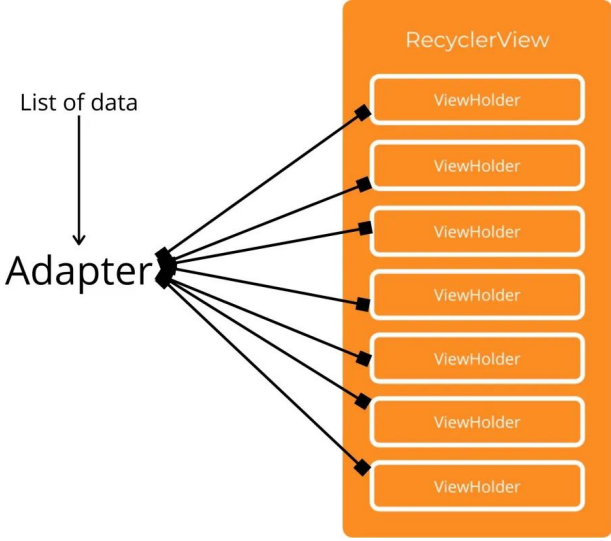
Depuis l'activity B, on appelle setResult pour communiquer des informations.

```
val resultIntent = Intent()
intent.putExtra( name: "return_value",  value: "Hello World")
setResult(RESULT_OK, resultIntent)
```

5. Android : Développement

Afficher une liste - RecyclerView

Sur Android les listes sont affichées dans des conteneurs, le **RecyclerView** en est un.
Comme son nom l'indique, il recycle ses vues, ce qui lui donne de meilleures performances et évite les fuites de mémoire.




```
class MyListAdapter() : RecyclerView.Adapter<MyListAdapter.ItemViewHolder>() {

    var data: List<ItemEntity> = listOf()
        set(value) {
            field = value
            this.notifyDataSetChanged()
        }

    override fun getItemCount(): Int = data.size

    override fun onCreateViewHolder(parent: ViewGroup, viewType: Int): ItemViewHolder {
        val view = LayoutInflater.from(parent.context).inflate(R.layout.layout_item, parent,  attachToRoot: false)
        return ItemViewHolder(view)
    }

    override fun onBindViewHolder(holder: ItemViewHolder, position: Int) {
        val item = data.get(position)
        holder.bind(item)
    }

    class ItemViewHolder(itemView: View) : RecyclerView.ViewHolder(itemView) {

        private val binding: LayoutItemBinding = LayoutItemBinding.bind(itemView)

        fun bind(item: ItemEntity) {
            binding.myItem.text = item.itemName
        }
    }
}
```

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

    <TextView...>

</androidx.constraintlayout.widget.ConstraintLayout>
```

5. Android : Développement

Afficher une liste - RecyclerView

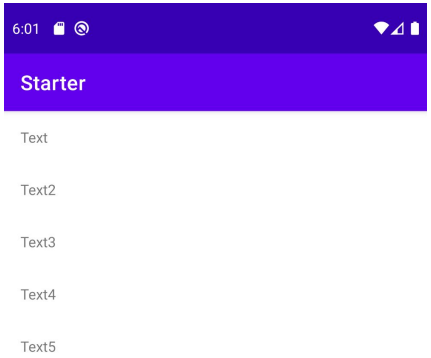
Le RecyclerView s'intègre ensuite de cette manière :

Activity:

```
adapter = MyListAdapter()
binding.myList.adapter = adapter
adapter.data = data
```

Layout lié à l'Activity:

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

    <androidx.recyclerview.widget.RecyclerView
        android:id="@+id/myList"
        android:layout_width="match_parent"
        android:layout_height="0dp"
        app:layoutManager="androidx.recyclerview.widget.LinearLayoutManager"
        app:layout_constraintBottom_toTopOf="@id/inputText"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent"
        tools:listitem="@layout/layout_item" />
```



5. Android : Développement

ViewModel - LiveData

Quelques prérequis
Ajouter les dépendances suivantes dans le build.gradle (:app) :

```
//ViewModel
implementation("androidx.lifecycle:lifecycle-viewmodel-ktx:2.5.1" )

//LiveData
implementation("androidx.lifecycle:lifecycle-livedata-ktx:2.5.1" )
```

5. Android : Développement

ViewModel - LiveData

```
class ListViewModel() : ViewModel() {

    private var _item = MutableLiveData<List<String>>()
    val item: LiveData<List<String>> = _item

    init {
        fetchData()
    }

    private fun fetchData() {
        _item.value = listOf(
            "Bonjour",
            "Test"
        )
    }

}

class ListViewModelFactory() : ViewModelProvider.Factory {
    override fun <T : ViewModel> create(modelClass: Class<T>): T {
        return if (modelClass.isAssignableFrom(ListViewModel::class.java)) {
            ListViewModel() as T
        } else {
            throw IllegalArgumentException("ViewModem Not Found")
        }
    }
}
```

```
class ListActivity : AppCompatActivity() {


    private lateinit var binding: ActivityViewmodelBinding
    private lateinit var adapter: MyListAdapter

    private lateinit var viewModel: ListViewModel

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

        viewModel = ViewModelProvider( owner: this, ListViewModelFactory())[ListViewModel::class.java]

        populateView()
        addObservers()
    }

    private fun populateView() {
        adapter = MyListAdapter()
        binding.myList.adapter = adapter

    }

    private fun addObservers() {
        viewModel.item.observe( owner: this) {  it: List<String>!
            adapter.data = it
        }
    }
}
```

5. Android : Développement

Stockage de données - SharedPreferences

Shared Preferences : stockent les données privées par l'utilisation de paires de clé/valeur.

```
var sharedPreferences: SharedPreferences = getSharedPreferences(SHARED_PREF_NAME, MODE_PRIVATE)
```

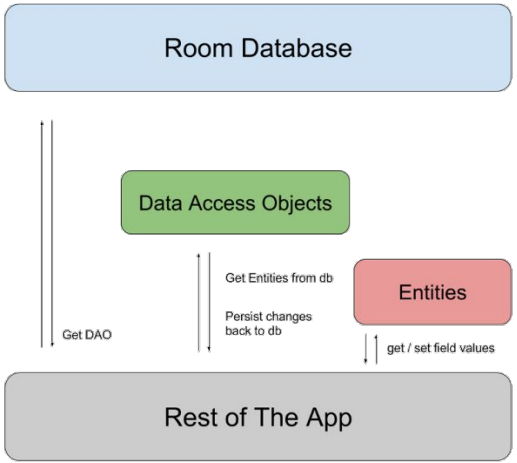
```
private fun saveData(input: String) {
    val editor = sharedPreferences.edit()
    editor.putString(STRING_DATA_KEY, input)
    editor.apply()
}

private fun loadData(): String =
    sharedPreferences.getString(STRING_DATA_KEY, "No value") ?: ""
```

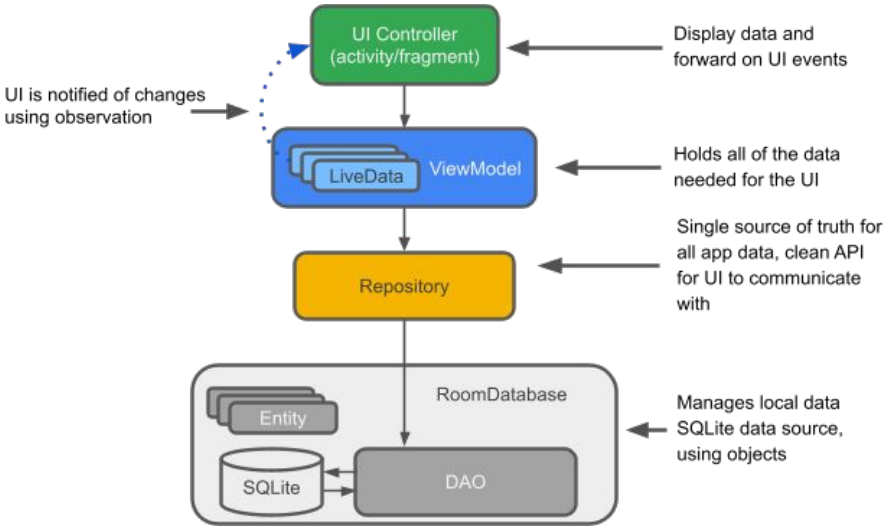
5. Android : Développement

Stockage de données - BDD (Room) - ViewModel - Coroutines

Qu'est-ce qu'une base de données Room ?

- Room est une librairie de persistance de données, qui propose une abstraction de SQLite.
- Room gère les tâches routinières que vous effectuiez auparavant avec un SQLiteOpenHelper.
- Room utilise le DAO pour envoyer des requêtes à sa base de données.
- Par défaut, pour éviter de mauvaises performances de l'UI, **Room ne vous autorise pas à émettre des requêtes sur le thread principal.**





5. Android : Développement

Stockage de données - BDD (Room) - ViewModel - Coroutines

Quelques prérequis :

Ajouter les dépendances suivantes dans le build.gradle (:app) :

```
//Room
implementation "androidx.room:room-runtime:2.4.3"
kapt "androidx.room:room-compiler:2.4.3"
implementation "androidx.room:room-ktx:2.4.3"
```

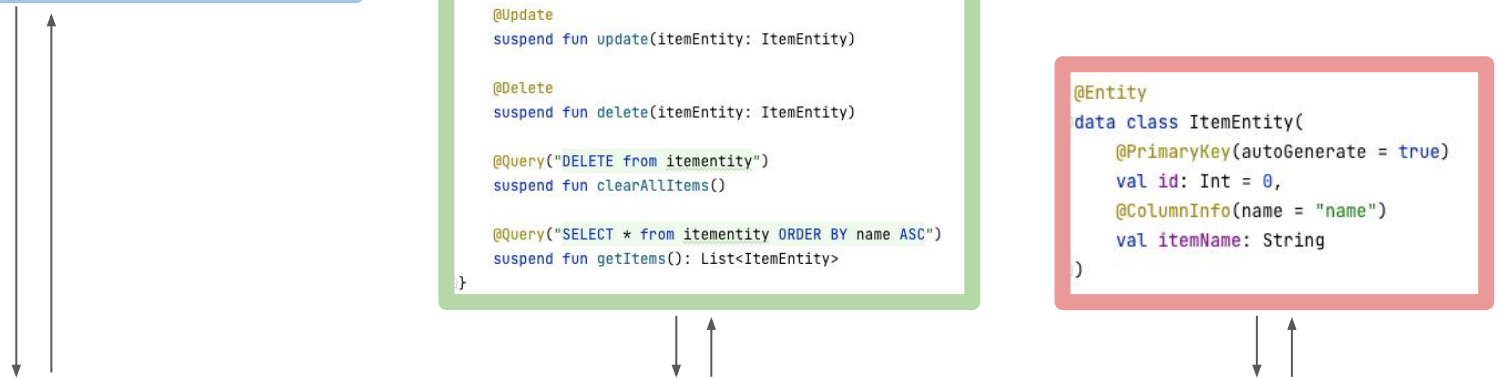
5. Android : Développement

Stockage de données - BDD (Room) - ViewModel - Coroutines

```kotlin
@Database(
    version = 1,
    entities = [
        ItemEntity::class,
    ]
)
abstract class MyDatabase : RoomDatabase() {
    abstract fun getItemDao(): ItemDao
}
```

```kotlin
@Dao
interface ItemDao {
    @Insert
    suspend fun insert(itemEntity: ItemEntity)

    @Update
    suspend fun update(itemEntity: ItemEntity)

    @Delete
    suspend fun delete(itemEntity: ItemEntity)

    @Query("DELETE from itementity")
    suspend fun clearAllItems()

    @Query("SELECT * from itementity ORDER BY name ASC")
    suspend fun getItems(): List<ItemEntity>
}
```

```kotlin
@Entity
data class ItemEntity(
    @PrimaryKey(autoGenerate = true)
    val id: Int = 0,
    @ColumnInfo(name = "name")
    val itemName: String
)
```



5. Android : Développement

Stockage de données - BDD (Room) - ViewModel - Coroutines

Initialisation de la BDD

```
private fun initDatabase() {
    database = Room.databaseBuilder(
        /* context = */ this,
        /* klass = */ MyDatabase::class.java,
        /* name = */ "my-database"
    ).build()
}
```

Factory du ViewModel

```
class RoomViewModelFactory(val database: MyDatabase) : ViewModelProvider.Factory {
    override fun <T : ViewModel> create(modelClass: Class<T>): T {
        return if (modelClass.isAssignableFrom(RoomViewModel::class.java)) {
            RoomViewModel(database) as T
        } else {
            throw IllegalArgumentException("ViewModem Not Found")
        }
    }
}
```

```
class RoomViewModel(
    private val database: MyDatabase
) : ViewModel() {

    private var _item = MutableLiveData<List<ItemEntity>>()
    val item: LiveData<List<ItemEntity>> = _item

    init {
        fetchData()
    }

    fun insertData(itemEntity: ItemEntity) {
        viewModelScope.launch { this: CoroutineScope
            database.getItemDao().insert(itemEntity)
            fetchData()
        }
    }

    fun clearDatabase() {
        viewModelScope.launch { this: CoroutineScope
            database.getItemDao().clearAllItems()
            fetchData()
        }
    }

    private fun fetchData() {
        viewModelScope.launch { this: CoroutineScope
            _item.value = database.getItemDao().getItems()
        }
    }

}
```

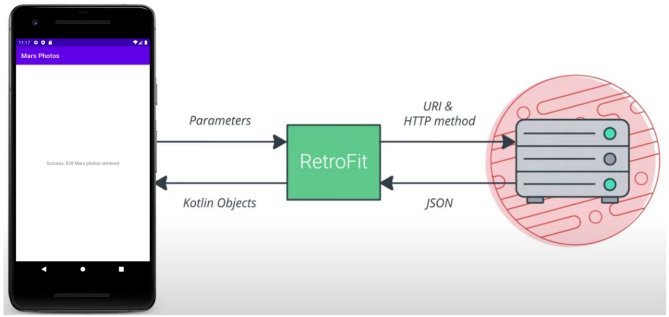
5. Android : Développement

Call API - Retrofit

Aujourd'hui, la plupart des serveurs Web exécutent des services Web à l'aide d'une architecture Web sans état courant, appelée REST, qui signifie REpresentational StateTransfer. Les services Web qui proposent cette architecture sont appelés services RESTful.

Les requêtes sont envoyées aux services Web RESTful de manière standardisée via des URI. Un URI (Uniform Resource Identifier, ou identifiant de ressource uniforme) identifie une ressource sur le serveur grâce à son nom, sans indiquer son emplacement ni la manière d'y accéder.

Retrofit est une librairie tierce, qui permet de simplifier les communications vers un serveur.



5. Android : Développement

Call API - Retrofit

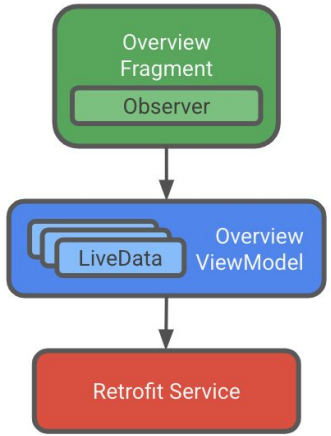
Quelques prérequis :

Ajouter les permissions dans AndroidManifest:

```
<uses-permission android:name="android.permission.INTERNET" />
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
```

Ajouter les dépendances suivantes dans le build.gradle (:app):

```
implementation("com.squareup.retrofit2:retrofit:2.9.0")
implementation("com.squareup.retrofit2:retrofit-adapters:2.8.1")
implementation("com.squareup.retrofit2:converter-gson:2.9.0")

implementation("org.jetbrains.kotlinx:kotlinx-serialization-json:1.1.0")
```



5. Android : Développement

Call API - Retrofit

```
interface RetrofitService {
    @GET("/realestate")
    suspend fun getData(): List<RealEstateResponse>
}
```

```
class RetrofitViewModel(
    private val retrofitService: RetrofitService
) : ViewModel() {

    private val _data = MutableLiveData<String>()
    val data: LiveData<String> = _data

    init {
        viewModelScope.launch { this: CoroutineScope
            _data.value = retrofitService.getData().size.toString()
        }
    }
}
```

```
object RetrofitApi {
    fun getService(
    ): RetrofitService {
        val retrofitBuilder = Retrofit.Builder()

        val okHttpClient = OkHttpClient.Builder()
            .build()

        retrofitBuilder.client(okHttpClient)

        val retrofit = retrofitBuilder
            .baseUrl(BASE_URL)
            .addConverterFactory(GsonConverterFactory.create())
            .build()

        return retrofit.create(RetrofitService::class.java)
    }

    private const val BASE_URL = "https://android-kotlin-fun-mars-server.appspot.com/"
}
```

5. Android : Développement

Log et Toast

Log

L’ajout manuel de logs pour déboguer le code se fait à partir de android:util.Log. Chaque message est précédé d’une étiquette. Les différentes méthodes sont :

- Log.v (‘’étiquette’’, ‘’message’’), pour les messages communs.
- Log.d (‘’étiquette’’, ‘’message’’), pour les messages de debug.
- Log.i (‘’étiquette’’, ‘’message’’), pour les messages à caractère informatif.
- Log.w (‘’étiquette’’, ‘’message’’), pour les warnings.
- Log.e (‘’étiquette’’, ‘’message’’), pour les erreurs.

Toast

Un Toast fournit des commentaires simples sur une opération dans une petite fenêtre (popup). Il ne remplit que la quantité d'espace requise pour le message et l'activité en cours reste visible et interactive. Les toasts disparaissent automatiquement après un timeout.

```
Toast.makeText( context: this,  text: "Message to display", Toast.LENGTH_SHORT).show()
```