



Programmation Android

Ce que vous devez savoir

Plan

- 1 **Connaissances essentielles**
- 2 **Définition d'une GUI et adaptation au contexte**
- 3 **Aspect mobile : adaptation à différentes configurations**
- 4 **Cycle de vie d'une activité**
- 5 **L'objet Intent**
- 6 **Stockage de données et de fichiers**

Plan

- 1 **Connaissances essentielles**
- 2 Définition d'une GUI et adaptation au contexte
- 3 Aspect mobile : adaptation à différentes configurations
- 4 Cycle de vie d'une activité
- 5 L'objet Intent
- 6 Stockage de données et de fichiers

Sur la plate-forme Android

Sur l'OS Android

- noyau Linux
- développement d'app avec un SDK Java/Kotlin en évolution régulière

Le Android SDK contient (entre autres)

- les **utilitaires** nécessaires au développement (**tools**) : adb, SDK manager, etc.
- les bibliothèques nécessaires au dev (pas toutes) : **APIs**, organisées par version
- le **Android Virtual Device Manager** (création et gestion des émulateurs)

Contenu du projet

Dossier App

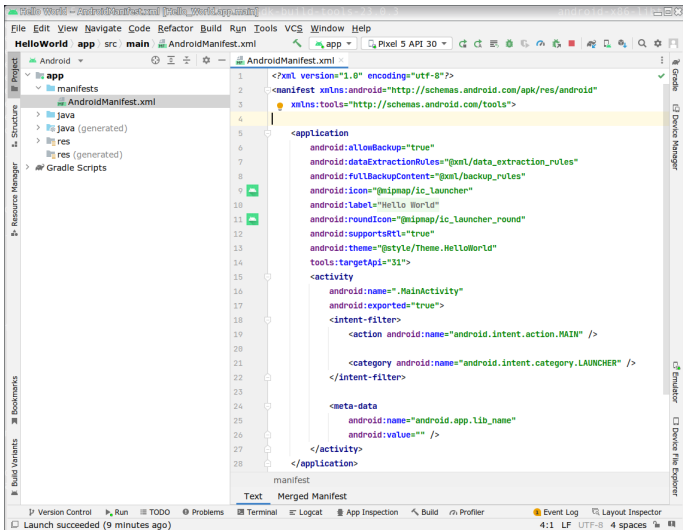
- **build/** : fichiers générés par le projet
- **libs/** : librairies additionnelles ("à la main")
- **src/** : intégralité des sources pour le développement (appli + tests)
- **build.gradle** : définition des options du build
- **proguard-rules.pro** : règles additionnelles pour le build
- **.gitignore** : définition des fichiers ignorés par git

Contenu du projet

Dossier App/src/

- **androidTest/** : tests de l'application dans l'environnement android
- **test/** : tests internes à l'application
- **main** : sources de l'application
 - **main/java** : sources Java, e.g. la classe qui lance l'activité
 - **main/res** : les ressources de l'application
 - **drawable-(h)(m)(l)dpi/** images dans différentes résolutions
 - **layout/** GUI design général (xml)
 - **values/** valeurs des variables (xml)
 - **menu/** définition des menus (xml)
- **main/AndroidManifest.xml** : description et éléments-clés de l'application (nom, activité principale, intents, etc.)

app/src/main/AndroidManifest.xml : fichier indispensable



The screenshot shows an IDE window with the following details:

- Project:** HelloWorld
- Package:** app
- File:** src/main/AndroidManifest.xml
- API Level:** Pixel 5 API 30
- Code Editor:** Contains the XML code for the AndroidManifest.xml file.

```
1 <?xml version="1.0" encoding="utf-8"?>
2 <manifest xmlns:android="http://schemas.android.com/apk/res/android"
3         xmlns:tools="http://schemas.android.com/tools">
4
5     <application
6         android:allowBackup="true"
7         android:dataExtractionRules="@xml/data_extraction_rules"
8         android:fullBackupContent="@xml/backup_rules"
9         android:icon="@mipmap/ic_launcher"
10        android:label="Hello World"
11        android:roundIcon="@mipmap/ic_launcher_round"
12        android:supportRtl="true"
13        android:theme="@style/Theme.HelloWorld"
14        tools:targetApi="31">
15        <activity
16            android:name=".MainActivity"
17            android:exported="true">
18            <intent-filter>
19                <action android:name="android.intent.action.MAIN" />
20
21                <category android:name="android.intent.category.LAUNCHER" />
22            </intent-filter>
23
24            <meta-data
25                android:name="android.app.lib_name"
26                android:value="" />
27        </activity>
28    </application>

```

At the bottom of the IDE, the status bar shows: "Launch succeeded (9 minutes ago)" and "4:1 LF UTF-8 4 spaces".

Manifest complet : manifest.xml + build.gradle

The screenshot shows the Android Studio IDE with the following components:

- Project Structure:** Shows the hierarchy: app > manifests > AndroidManifest.xml.
- AndroidManifest.xml:** Contains the following XML code:


```

<?xml version="1.0"
    android:versionCode="1"
    android:versionName="1.0"
    package="org.fisichel.helloworld"
    xmlns:android="http://schemas.android.com/apk/res/android" >

    <uses-sdk
        android:minSdkVersion="23"
        android:targetSdkVersion="32" />

    <application
        android:allowBackup="true"
        android:appComponentFactory="androidx.core.app.CoreComponentFacto
        android:dataExtractionRules="@xml/data_extraction_rules"
        android:fullBackupContent="@xml/backup_rules"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportRtl="true"
        android:theme="@style/Theme.HelloWorld" >

        <activity
            android:exported="true"
            android:name="org.fisichel.helloworld.MainActivity" >

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

                <category
                    android:name="android.intent.category.LAUNCHER" />

            <meta-data
                android:name="android.app.lib_name"
                android:value="" />
      
```
- build.gradle (Hello World):** Shows the Gradle configuration for the app module, including dependencies and build tools.
- Manifest Sources:** Lists the sources used in the manifest: core:1.7.0 manifest, Hello_World.app main manifest (this file), and build.gradle manifest.
- Other Manifest Files:** Lists other manifest files included in the merge, such as recyclerview:1.1.0 manifest, lifecycle-livedata-core:2.3.1 manifest, etc.
- Bottom Bar:** Shows the status bar with "Launch succeeded (34 minutes ago)" and various tool icons.

Points essentiels du manifest

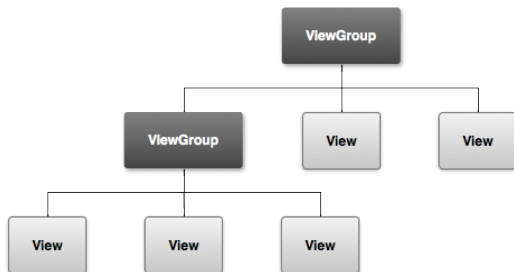
Caractéristiques de l'application

- 1 **le nom de package** `build.gradle`
- 2 les numéros d'API min et cible `build.gradle`
- 3 attributs de l'application : balise **<application ...**
 - nom : **<application ... android:label="@string/app_name" >**
 - caractéristiques de chaque activité définie :
 - classe : **<activity android:name=".AfficheURL" >**
 - filtre(s) pour les **<intent-filter>**

Plan

- 1 Connaissances essentielles
- 2 Définition d'une GUI et adaptation au contexte**
- 3 Aspect mobile : adaptation à différentes configurations
- 4 Cycle de vie d'une activité
- 5 L'objet Intent
- 6 Stockage de données et de fichiers

Organisation d'une GUI Android



- **View** : élément de type *widget* (boutons, champ texte, etc.) [▶ View](#)
- **ViewGroup** : un type de **View** gérant d'autres View, par un mécanisme de mise en page : grille, liste verticale/horizontale, contraintes etc. [▶ ViewGroup](#)

La classe `android.view.View`

Principes d'utilisation

- Toutes les *widgets* sont des sous classes de `View`
- Il est possible de créer une UI dans le code Java, mais on définit généralement une UI grâce à des fichiers XML
- À chaque type de `View` correspond ainsi une balise XML

▶ classe Java `Button` en XML ⇒

```
<Button
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="@string/button_text"
    ... />
```

La classe `android.view.ViewGroup`

Principes

- Un `ViewGroup` est un type (hérite de) de `View` pouvant contenir d'autres `View`, appelés *children*
- C'est la classe de base dont hérite les *layouts* et les *view containers*
- À chaque type de `ViewGroup` correspond une balise XML
- La différence est qu'elle pourra elle-même contenir des balises de type `View`

▶ [plus d'information](#)

Clic sur bouton, solution 1 : XML

```
<Button
    android:id="@+id/button_send"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:onClick="sendMessage"
    android:text="@string/button_send" />
```

- Ajout de l'attribut **android:onClick** à l'élément *Button*
- valeur : méthode définie dans l'activité contenant la *view*
- signature standardisée : **public void** et un paramètre de type *View*

```
/** Called when the user touches the button */
public void sendMessage(View view) {
    // Do something in response to button click
}
```

Solution 2

dans le code Java (à préférer)

Ajout d'un écouteur (listener) au bouton, par exemple au moment de la création de l'activité, dans la méthode `onCreate` :

```
Button button = (Button) findViewById(R.id.button_send);
button.setOnClickListener(new View.OnClickListener() {
    public void onClick(View v) {
        // Do something in response to button click
    }
});
```

Solution 3, Avec la librairie de Data Binding (Android Jetpack)

Jetpack est une surcouche des API Android qui facilite le développement

► [Android Jetpack](#)

Pour profiter du data binding, il faut modifier, si nécessaire, le fichier de build pour activer cette fonctionnalité

► [plus d'information](#)

To configure your app to use data binding, enable the `dataBinding` build option in your `build.gradle` file in the app module, as shown in the following example:

```
android {
    ...
    buildFeatures {
        dataBinding true
    }
}
```


Plan

- 1 Connaissances essentielles
- 2 Définition d'une GUI et adaptation au contexte
- 3 Aspect mobile : adaptation à différentes configurations**
- 4 Cycle de vie d'une activité
- 5 L'objet Intent
- 6 Stockage de données et de fichiers

Aadaptation de l'UI au contexte

Propriétés d'un écran

- **size** → **small**, **normal**, **large** ou **xlarge**
- **density** → **low** (ldpi), **medium** (mdpi), **high** (hdpi), **extra high** (xhdpi)

Principe et gestion de l'adaptation

- Chaque layout ou bitmap est placé dans un sous répertoire de **res** ayant un nom lié à la taille et/ou à la résolution correspondantes.
- Note : le **changement d'orientation** (portrait ou paysage) est une **modification de la taille de l'écran**

Gestion de différents layout

```
MyProject/  
  res/  
    layout/  
      main.xml  
    layout-large/  
      main.xml
```

Un layout par configuration

- Pour chaque taille à supporter : **un fichier layout de même nom.**
- Chaque configuration est placée dans un sous répertoire de **res** correspondant à la taille : **res/layout-*< screen_size >***
e.g. res/layout-large.
- Par défaut, **layout/** est utilisé pour l'orientation portrait.

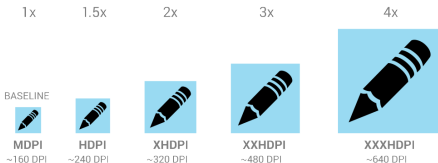
Gestion de différentes orientation

```
MyProject/  
  res/  
    layout/                # default (portrait)  
      main.xml  
    layout-land/          # landscape  
      main.xml  
    layout-large/         # large (portrait)  
      main.xml  
    layout-large-land/    # large landscape  
      main.xml
```

Gestion de différentes résolutions

Exemple pour les images

- Il est important de fournir des icônes avec différentes résolutions
- À partir d'une image vectorielle, on génère 4 images ayant les rapports suivants : xhdpi: 2.0, hdpi: 16, mdpi: 1.0, ldpi: 0.75
- e.g. 200x200 xhdpi, 150x150 hdpi, 100x100 mdpi, 75x75 ldpi



Gestion de différentes résolutions

Idem, pour les sous répertoires de **res** :

```
MyProject/  
  res/  
    drawable-xhdpi/  
      awesomeimage.png  
    drawable-hdpi/  
      awesomeimage.png  
    drawable-mdpi/  
      awesomeimage.png  
    drawable-ldpi/  
      awesomeimage.png
```

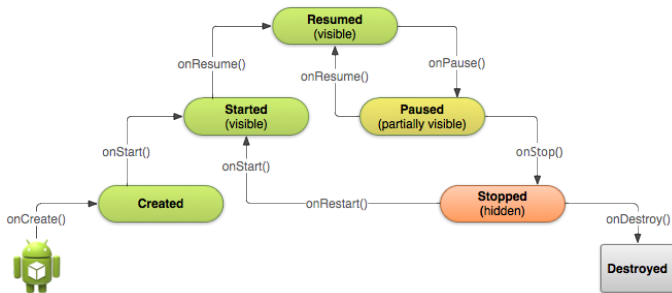
Accès automatique à la ressource correspondante

- Dans le code, `@drawable/awesomeimage` sélectionnera automatiquement la ressource correspondant à la résolution courante.

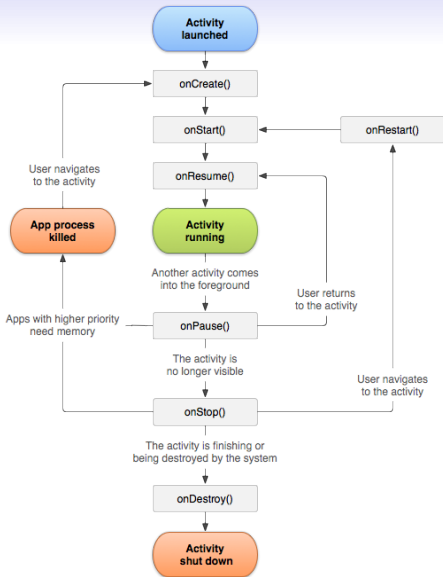
Plan

- 1 Connaissances essentielles
- 2 Définition d'une GUI et adaptation au contexte
- 3 Aspect mobile : adaptation à différentes configurations
- 4 Cycle de vie d'une activité**
- 5 L'objet Intent
- 6 Stockage de données et de fichiers

Méthodes du cycle de vie : définies dans la classe Activity



Cycle de vie : interactions utilisateurs



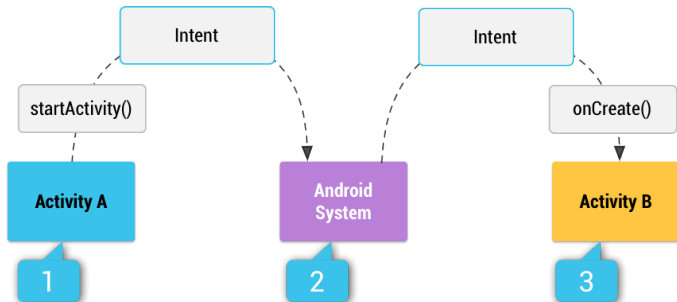
Plan

- 1 Connaissances essentielles
- 2 Définition d'une GUI et adaptation au contexte
- 3 Aspect mobile : adaptation à différentes configurations
- 4 Cycle de vie d'une activité
- 5 L'objet Intent**
- 6 Stockage de données et de fichiers

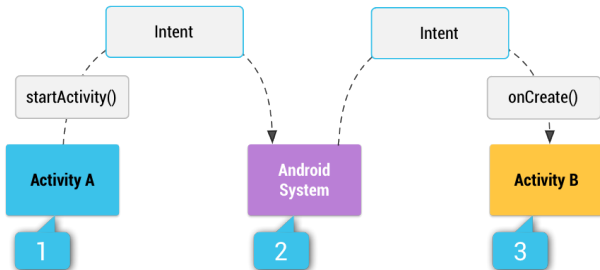
L'objet `android.content.Intent`

- **explicite** : lancement d'une activité spécifique
- lancement : **`android.app.Activity.startActivity(Intent)`**
- communication entre activités : **`android.content.Intent.putExtra(K,V)`**
- **implicite** : demande à l'OS de lancement d'une action (activité inconnue)

L'objet android.content.Intent



Intent implicite : cycle de vie



- 1** activité A : `startActivity` avec un Intent implicite
- 2** l'OS parcourt les applications pour trouver les **intent-filter** correspondant à l'action demandée
- 3** l'OS démarre l'activité trouvée, B

Plan

- 1 Connaissances essentielles
- 2 Définition d'une GUI et adaptation au contexte
- 3 Aspect mobile : adaptation à différentes configurations
- 4 Cycle de vie d'une activité
- 5 L'objet Intent
- 6 **Stockage de données et de fichiers**

Android storage

	Type de contenu	Méthodes	Permissions requises	Accès depuis autre app	Suppression lors de la désinstallation
Fichiers App	App seule	internal storage : getFilesDir() getCacheDir() external storage : getExternalFilesDir() getExternalCacheDir()	internal storage : Aucune external storage : Aucune API 19+	Aucun	Oui
Médias	images audio vidéos	API MediaStore	Depuis une autre app : API 30+ : READ_EXTERNAL_STORAGE API 29 : READ_EXTERNAL_STORAGE WRITE_EXTERNAL_STORAGE API 28- : Permissions pour tous les fichiers	Suivant les permissions	Non
Documents utilisateurs et autres	autre contenu partagé et téléchargements	Storage Access Framework	Aucune	via le system file picker	Non
Préférences	Paires clé-valeur	Jetpack Preferences SharedPreferences API	Aucune	Aucun	Oui
Base de données	données structurées	Room persistence library	Aucune	Aucun	Oui

Quelques ressources web

LE site pour les développeurs Android

[▶ Android developer](#)

- Accès direct à la documentation [▶ Docs](#)
- → [▶ Guides](#)
- → [▶ API](#)
- → [▶ Samples](#)

The Busy Coder's Guide to Android Development

[▶ Site](#)

- Et sa base d'exemples open source très fournie [▶ GitHub](#)