



# Android

Développement d'applications mobiles

Formation de base





# Développement d'applications mobiles avec Android (base)

- Introduction
- Le SDK Android
- Android Studio
- Un projet Android
- L'interface utilisateur
- Intent et navigation
- Listes et adaptateurs
- La persistance des données
- Les BroadcastReceivers
- Annexe



# Introduction



# Introduction : Présentation

- Android est la première plate-forme mobile open source et entièrement paramétrable.
  - Les sources sont accessibles à tous et chacun peut les consulter, voire même créer sa propre variante d'Android.
  - Le code source disponible n'inclut cependant pas les services phares comme Maps, Gmail ou Google Play, qui restent la propriété de Google.
  - Bien qu'open source, Android ne peut donc pas être considéré "libre".



# Introduction :

## Liens utiles

- Sources
  - <http://source.android.com>
- Site officiel et site développeurs (Doc, guides, references,...)
  - <http://www.android.com>
  - <http://developer.android.com>
- Actualités concernant Android
  - <http://www.pointgphone.com>
  - <http://www.androidpit.fr>
  - <http://www.frandroid.com>
  - <http://www.phonandroid.com>



# Introduction : Android vs. iOS

## iOS

- Le concurrent numéro un d'Android. Présent sur tous les produits mobiles Apple (iPhone, iPad, iPod touch,...).
- Système propriétaire de Apple.
- Coûts non-négligeable pour le développeur :
  - Mac obligatoire pour publier une application sur l'AppStore.
  - Un compte développeur coûte une centaine d'euro par an.
  - Les applications publiées doivent être validées par Apple (utilité, qualité,...), ce qui peut prendre plusieurs jours.
- Cependant, comme tout système "fermé", iOS offre un certain confort de développement mais qui laisse peu de place à l'improvisation.

## Android

- Présent sur de nombreux smartphones de marques diverses (Samsung, LG, Motorola, OnePlus, Huawei,...).
- Code open-source basé sur le noyau Linux.
- Coûts négligeables pour le développeur :
  - Développement possible sur n'importe quel OS avec le même IDE.
  - Un compte développeur coûte 25\$ à vie, et les services offerts par Google sont gratuits.
  - Les applications publiées sont validées rapidement, sans que l'utilité de votre application ne soit mise en cause.
- Comme tout système "ouvert", Android laisse plus de liberté au développeur, mais certains lui reprochent un manque d'encadrement.



# Introduction : Windows Phone et autres

## Windows Phone

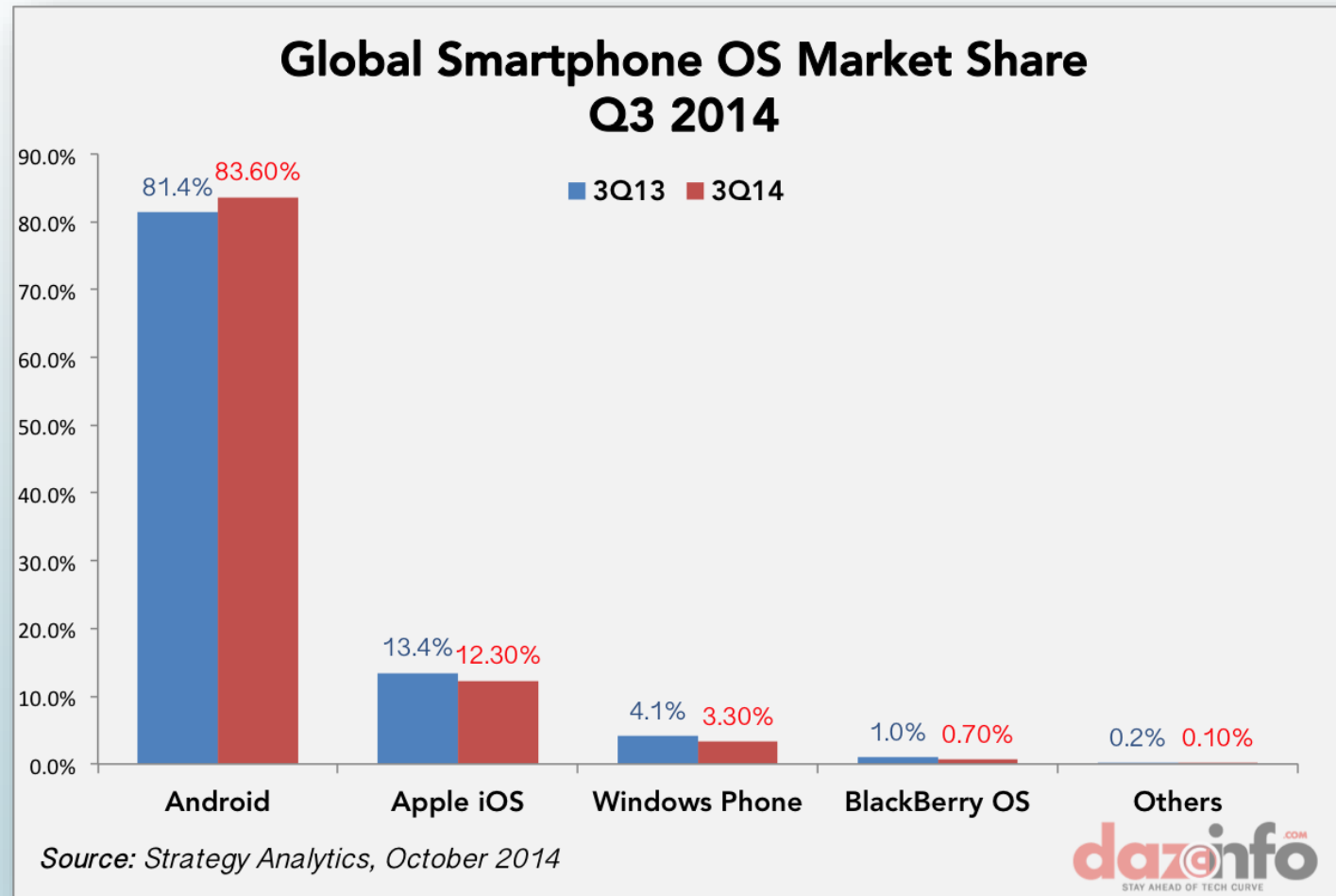
- Après des débuts difficile, Windows a commencé à remonter la pente. Présent sur certaines marques de smartphones (Nokia, HTC, certains Samsung,...)
- Système propriétaire de Microsoft
- Coûts variables selon les cas :
  - Compte individuel à environ 19\$ mais ne permettant pas d'utiliser toutes les fonctionnalités.
  - Compte entreprise à environ 99\$.
- Le développement Windows Phone utilise la technologie Silverlight. Microsoft fournit une documentation abondante.

## Autres

- Blackberry OS
- Autres systèmes plus marginaux

# Introduction :

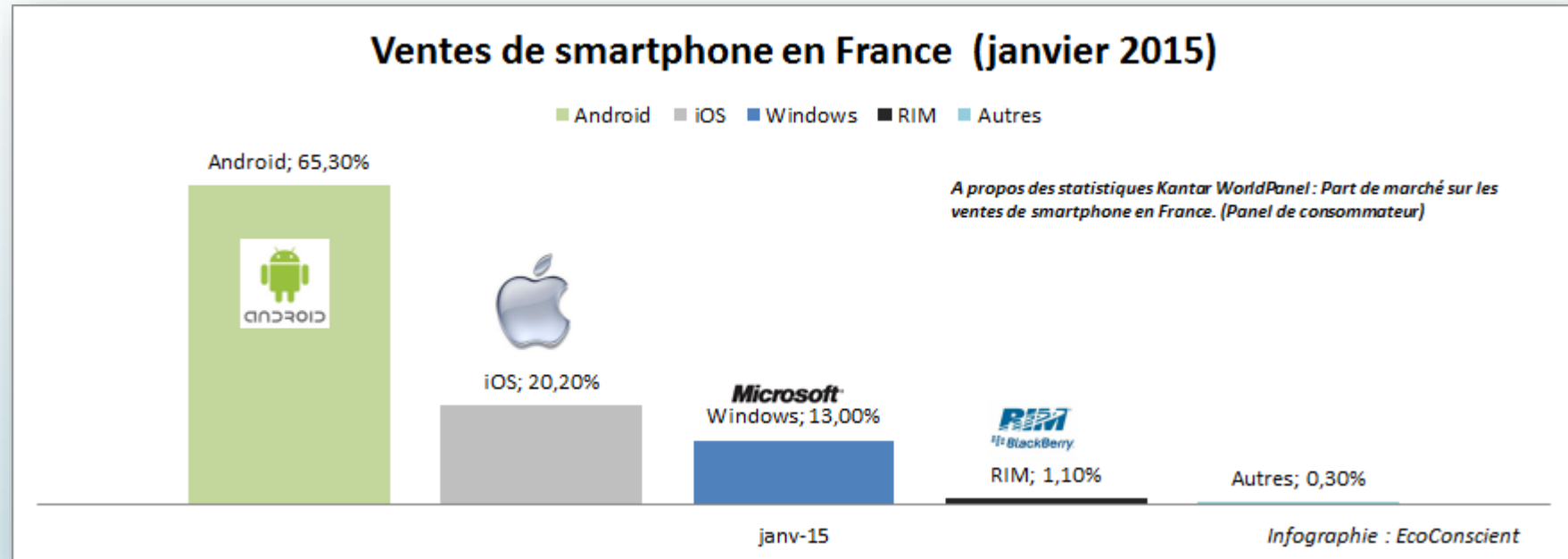
## Face aux concurrents





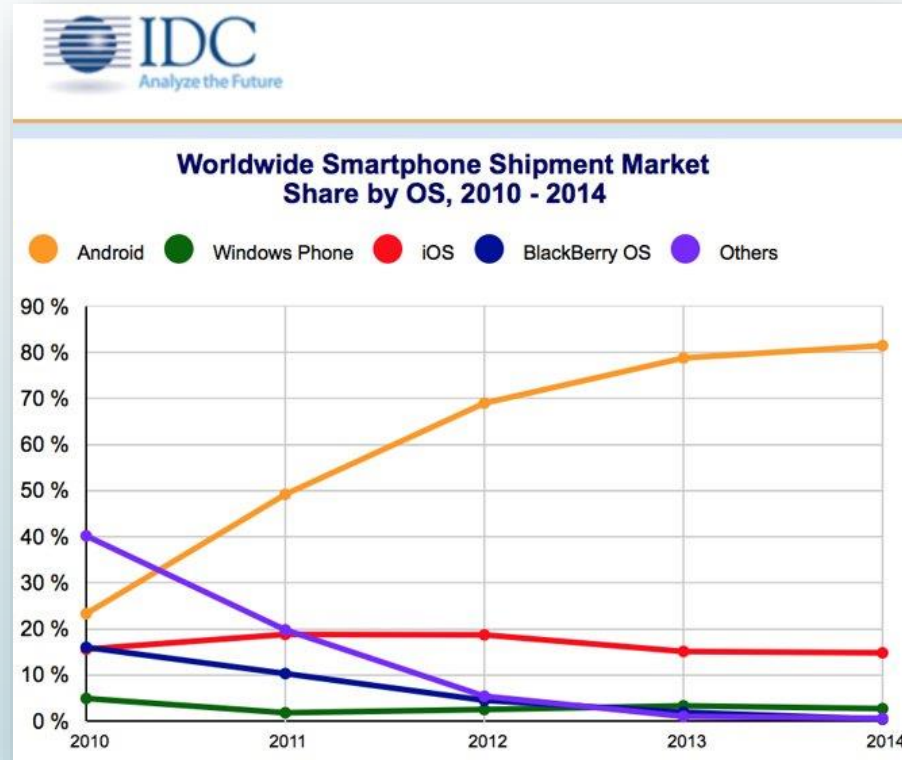
# Introduction :

## Face aux concurrents



# Introduction :

## Face aux concurrents



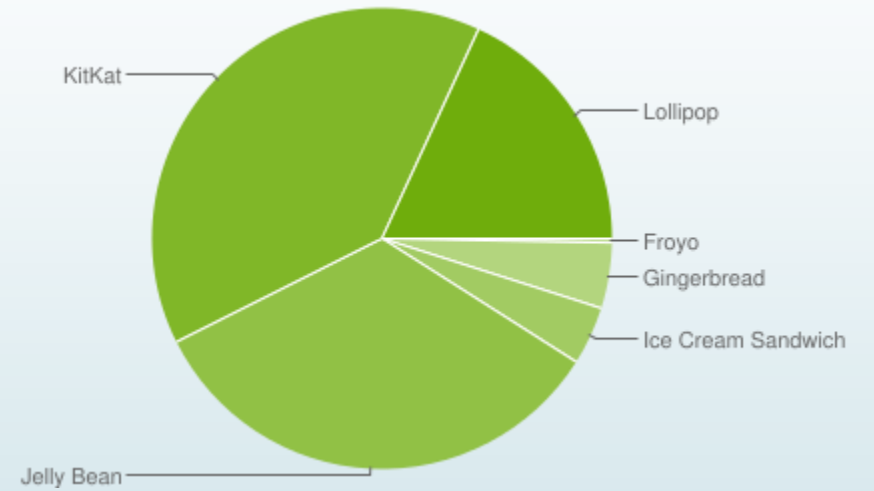


# Le SDK Android

# Le SDK Android :

## Les différentes versions (août 2015)

API	Name	Distribution (août 2015)
8	Froyo	0,3%
10	Gingerbread	4,6%
15	IceCreamSandwich	4,1%
16	JellyBean	13,0%
17		15,9%
18		4,7%
19	KitKat	39,3%
21	Lollipop	15,5%
22		2,6%
23	Marshmallow	N/A





# Le SDK Android : Architecture

- La plate-forme Android est composée de différentes couches :
  - Un noyau Linux
    - Gestion des ressources matérielles, communications réseaux,...
  - Des bibliothèques
    - SQLite (base de donnée interne), WebKit (rendu HTML), OpenGL (graphisme),...
  - Un framework applicatif qui propose au développeur des services de gestion de vues, de contenu, de ressources ainsi que les accès à la téléphonie, ou encore à la localisation.
  - Des applications natives, dont un navigateur web, un gestionnaire de contacts, un calendrier,...



# Le SDK Android : Architecture

- Note : Dalvik Virtual Machine et Java Virtual Machine
  - Sur Android, la DVM (Dalvik Virtual Machine) remplace la JVM.
  - Pourquoi ?
    - La JVM coûte cher.
    - La DVM est plus légère et consomme moins de ressources.



# Le SDK Android : Outils

- Google fournit, en plus du système d'exploitation, un kit de développement (SDK). Celui-ci contient :
  - Les différentes APIs
  - Des Samples (exemples d'applications)
  - Des ressources (documentation)
  - Des outils de debug, de gestion, d'émulation,...
- Pour utiliser ces différents outils et développer en Android, nous utiliserons un IDE (integrated development environment) : Android Studio.



# Android Studio





# Android Studio : Présentation

- **Android Studio** est l'**IDE** officiel pour développer sur **Android**. Il est basé sur l'IDE IntelliJ, et hérite donc de ses caractéristiques (raccourcis, environnement,...).
- Note : Auparavant le développement se faisait sur Eclipse, grâce à l'ADT Bundle. Cette solution est aujourd'hui fortement déconseillée bien qu'elle reste possible.



# Android Studio : Installation

- Sont nécessaires :
  - Le JDK 7
  - L'Android Studio Bundle
    - Android Studio
    - Android SDK

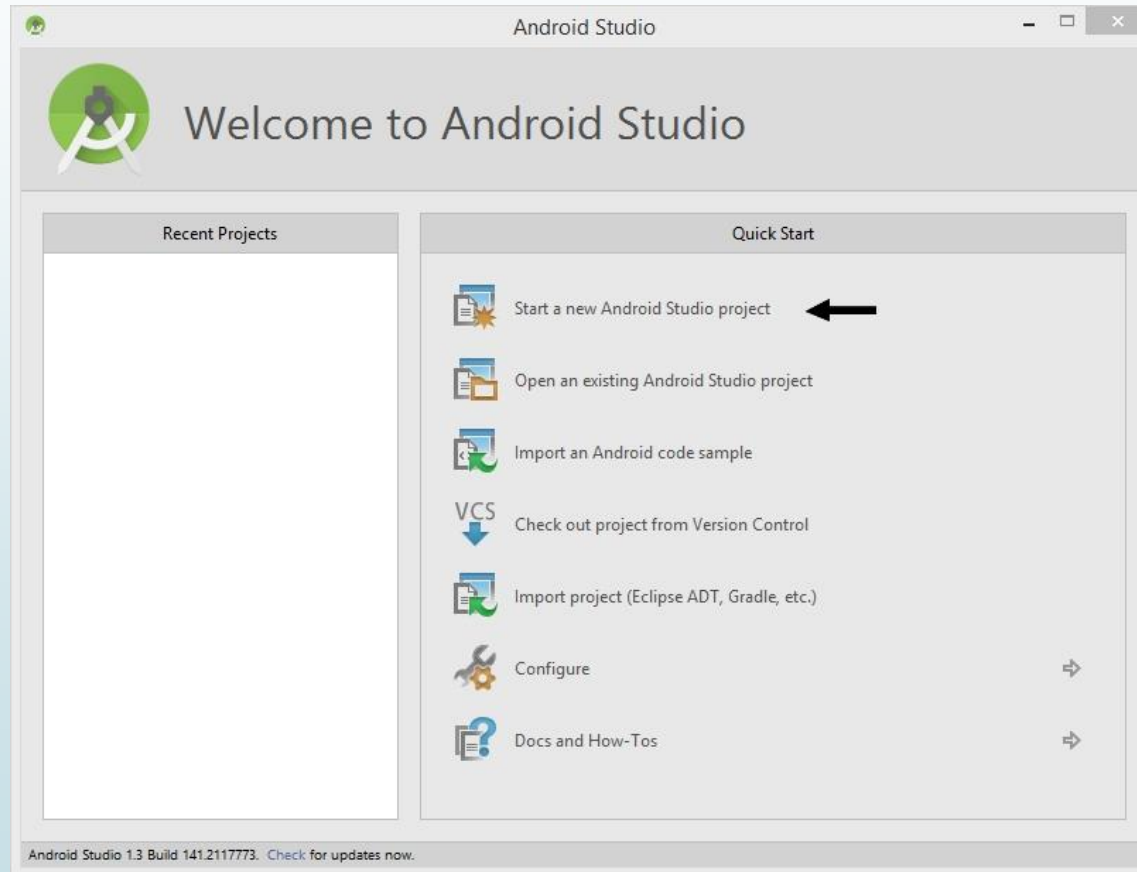
<https://developer.android.com/sdk/index.html>



# Android Studio : Utilisation

- **Android Studio** affiche un environnement assez classique divisé en **fenêtres** que l'on peut afficher, réduire ou fermer au choix.
- Certaines fenêtres possèdent plusieurs **onglets** (par exemple lorsqu'on a le choix entre utiliser un éditeur WYSIWYG ou éditer un fichier nous-même).
- L'utilisation des **outils** du SDK dans Android Studio est détaillée en annexe.

# Android Studio : Un Hello World



# Android Studio : Un Hello World

Create New Project

New Project  
Android Studio

Configure your new project

Application name: Hello World ← 1

Company Domain: app.formation.be ← 2

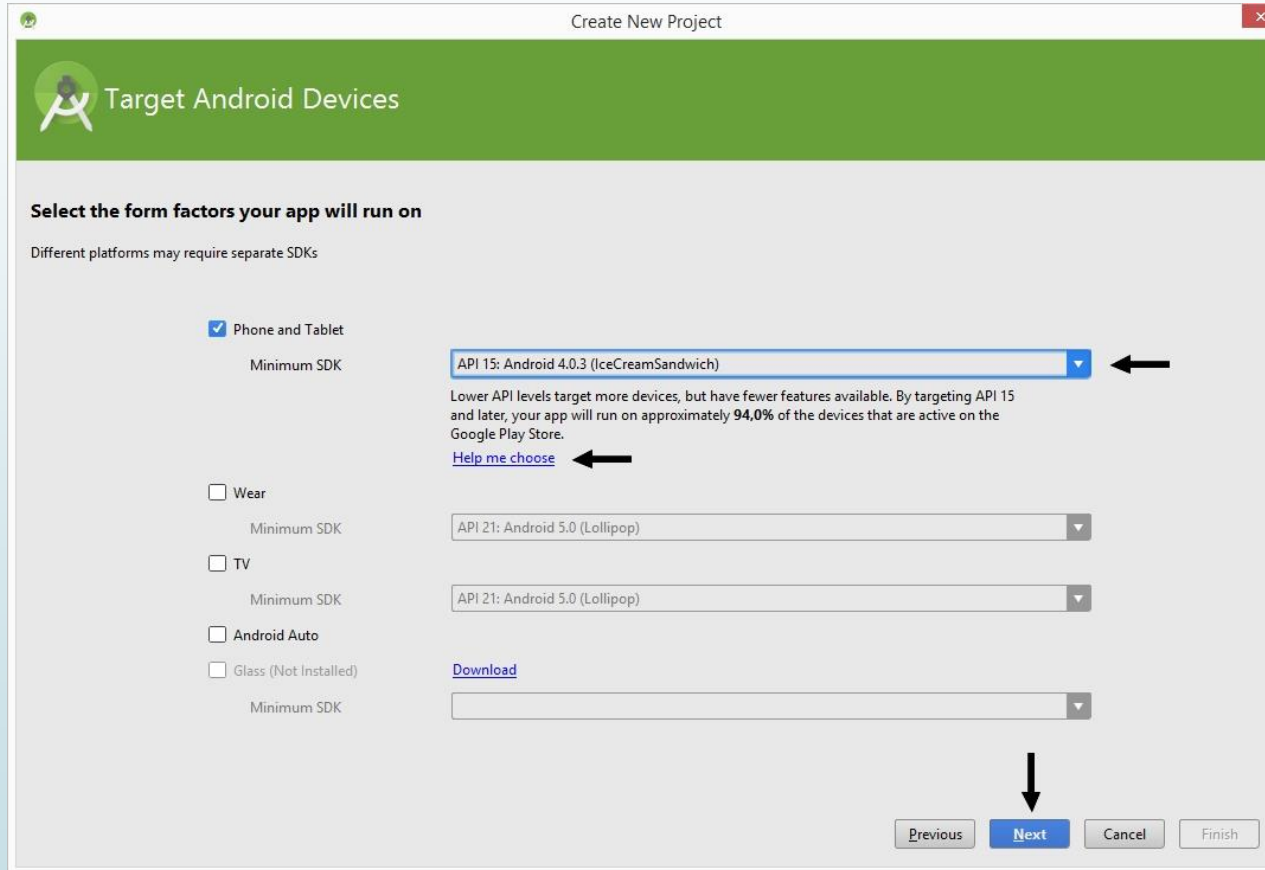
Package name: be.formation.app.helloworld [Edit](#)

Project location: D:\Android\Projects\HelloWorld ← 3

Previous Next Cancel Finish

4

# Android Studio : Un Hello World



Create New Project

Target Android Devices

Select the form factors your app will run on

Different platforms may require separate SDKs

☒ Phone and Tablet

Minimum SDK

API 15: Android 4.0.3 (IceCreamSandwich)

Lower API levels target more devices, but have fewer features available. By targeting API 15 and later, your app will run on approximately 94,0% of the devices that are active on the Google Play Store.

[Help me choose](#)

☐ Wear

Minimum SDK

API 21: Android 5.0 (Lollipop)

☐ TV

Minimum SDK

API 21: Android 5.0 (Lollipop)

☐ Android Auto

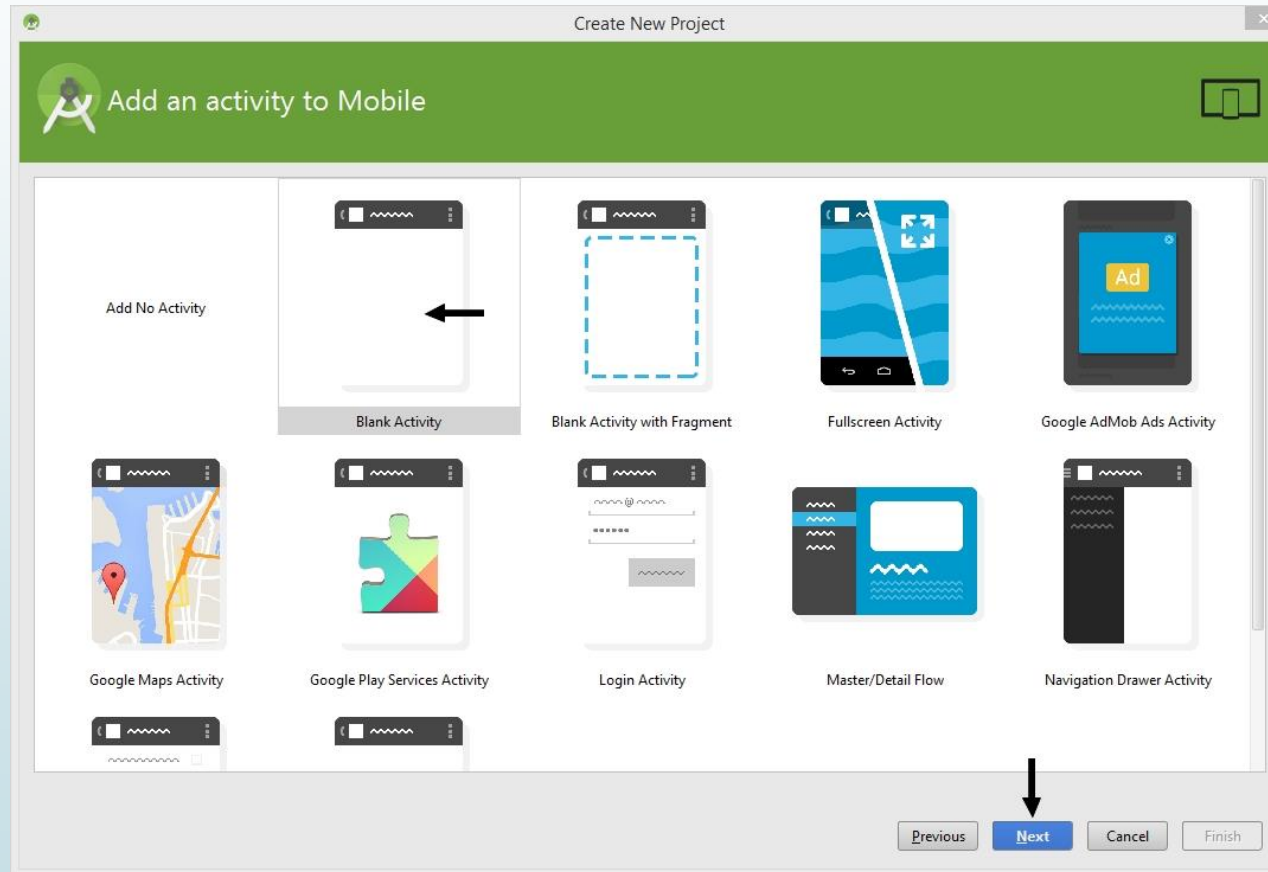
☐ Glass (Not Installed)

Minimum SDK

[Download](#)

Previous Next Cancel Finish

# Android Studio : Un Hello World



# Android Studio : Un Hello World

Create New Project

Customize the Activity

Creates a new blank activity with an action bar.

Blank Activity

Activity Name: MainActivity

Layout Name: activity\_main

Title: MainActivity

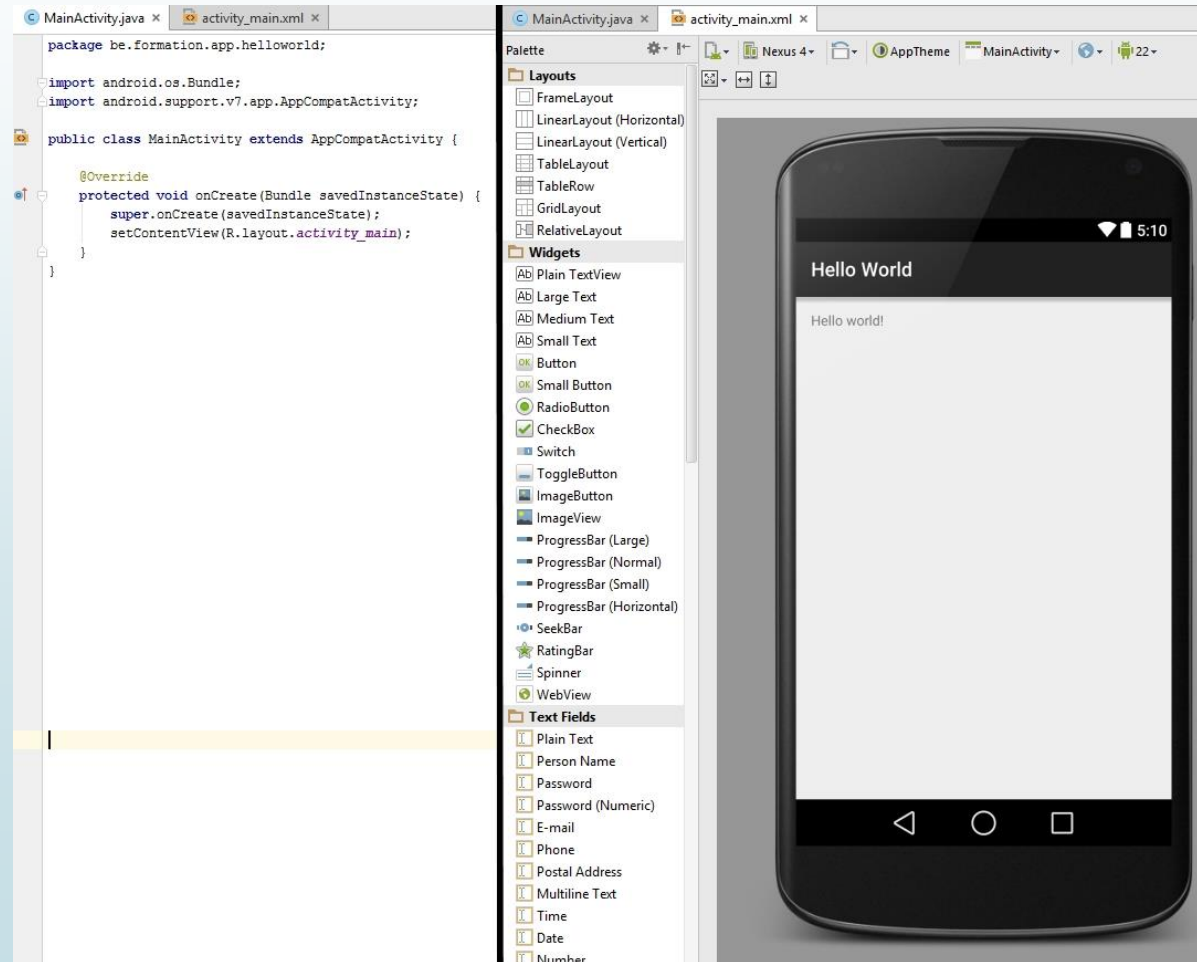
Menu Resource Name: menu\_main

The name of the activity class to create

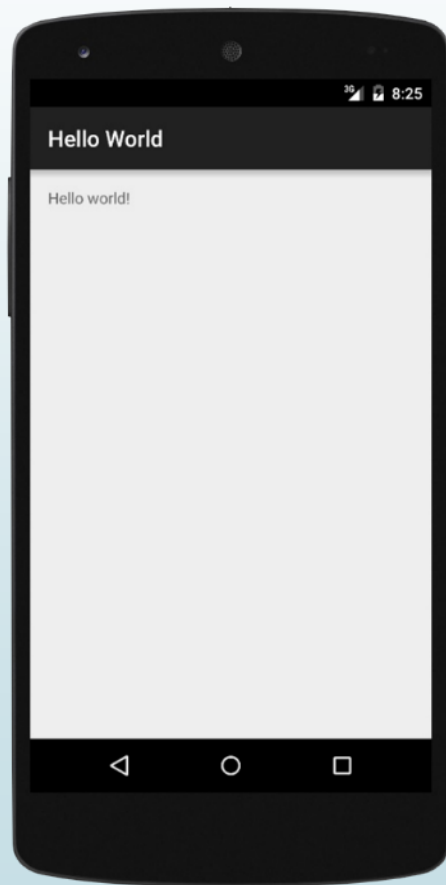
Previous Next Cancel Finish



# Android Studio : Un Hello World



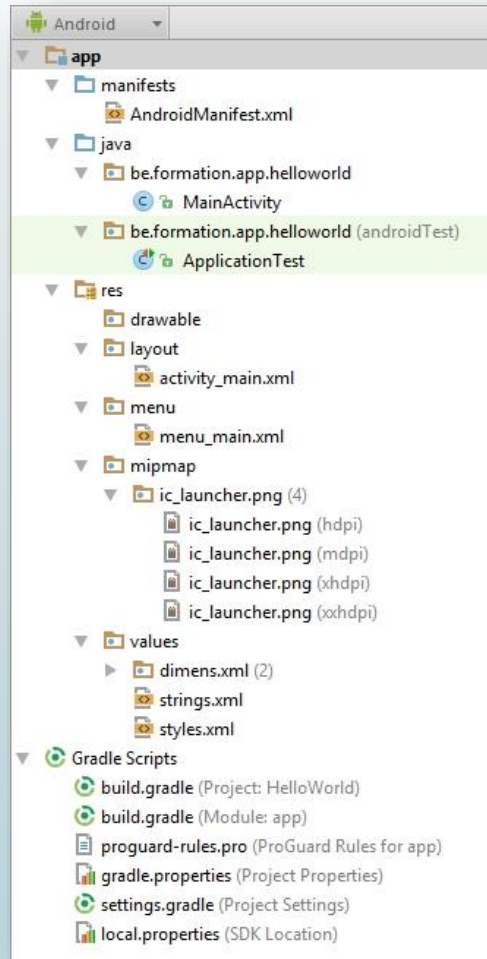
# Android Studio : Un Hello World



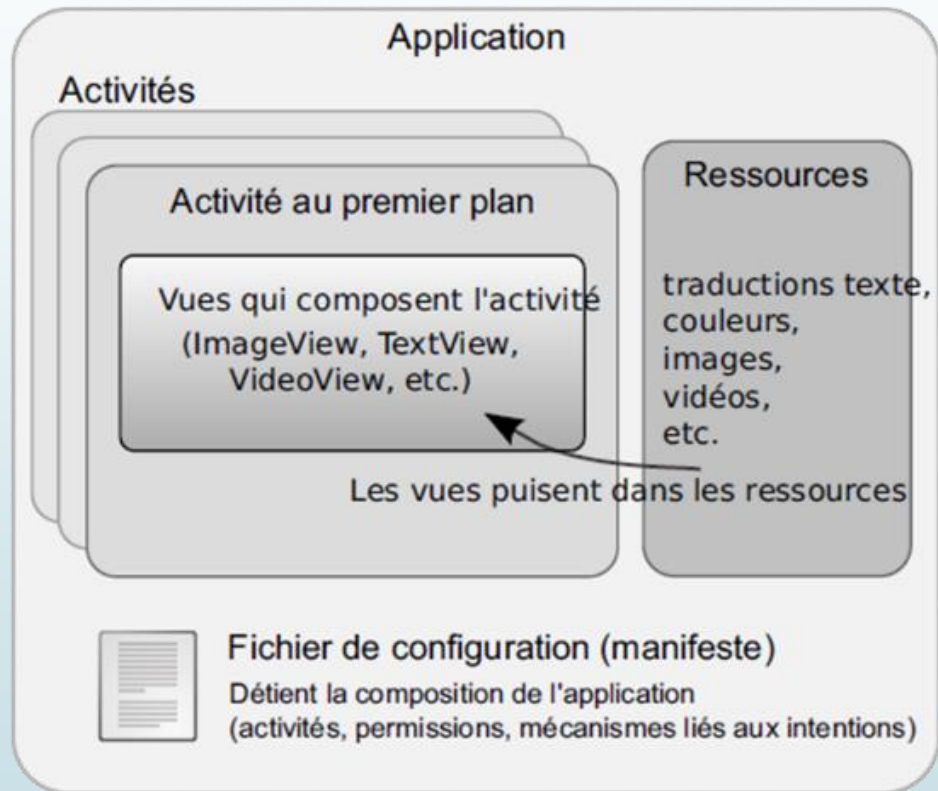


Un project Android

# Un project Android : Structure du projet



# Un project Android : Architecture





# Un project Android : Le manifeste

Le fichier manifeste (**AndroidManifest.xml**) est un fichier indispensable à chaque application qui décrit entre autres :

- Le **point d'entrée** de votre application (le code qui doit être exécuté au démarrage de l'application).
- Les composants qui constituent votre application.
- les **permissions nécessaires** à l'exécution du programme
  - accès à Internet
  - accès à l'appareil photo
  - accès au GPS
  - ...



# Un project Android : Le manifeste

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example"
    android:versionCode="1"
    android:versionName="1.0" >
    <uses-sdk android:minSdkVersion="14" />
    <application
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name" >
        <activity
            android:name=".Android_ExampleActivity"
            android:label="@string/app_name" >
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>
</manifest>
```



# Un projet Android :

## Les activités

- **Composant principal** de la plateforme Android sur lequel viendront se greffer des éléments graphiques (Views).
- Détermine le **comportement** d'une page bien spécifique.
- Peut être assimilée à **un écran** que l'application propose à l'utilisateur.
- Doit étendre la classe **Activity**, ou une de ses classes enfants.
- Doit être déclaré dans le manifeste de l'application
- Exemples :
  - Un formulaire
  - Une liste des conversations SMS
  - Un plan Google Map
  - ...





# Un project Android :

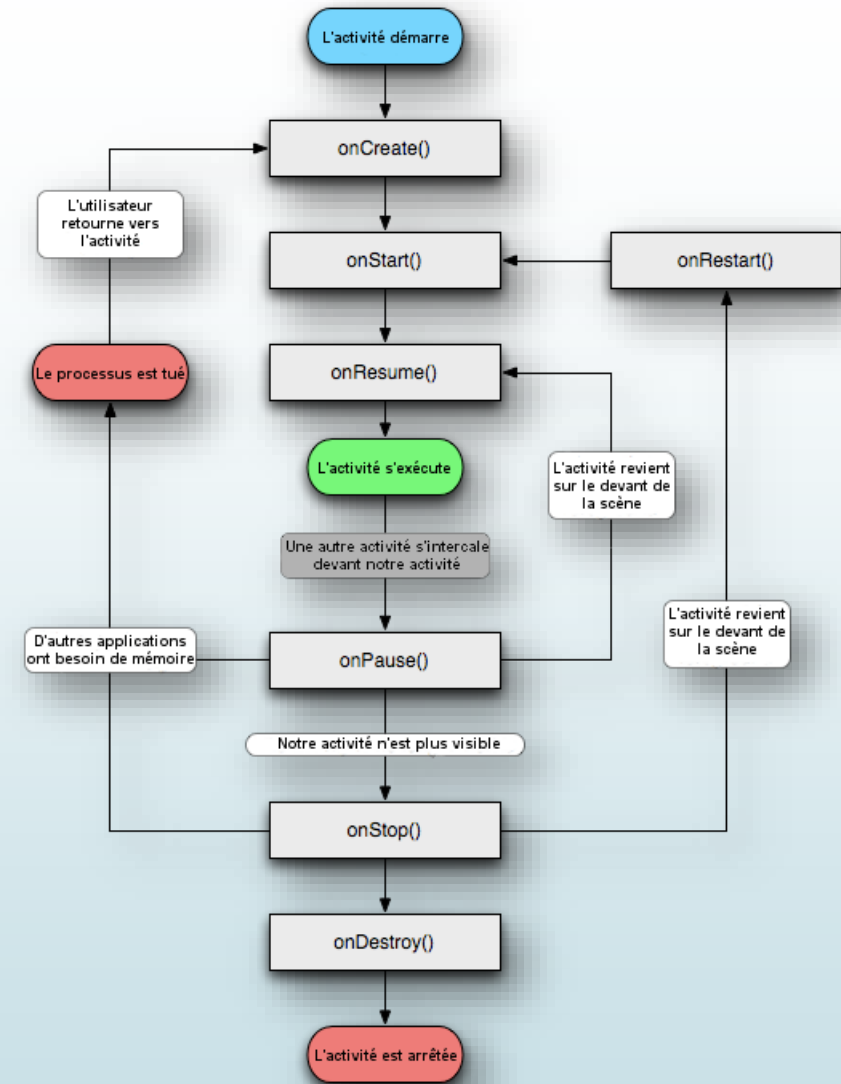
## Les activités

- Une **activité** est composée de **deux parties** :
  - Une **logique d'activité** et la gestion du cycle de vie de cette activité. Le tout géré dans une classe Java héritant de la classe Activity.
  - Une **interface utilisateur** que l'on définira soit dans le code ou plus couramment dans un fichier ressource de type XML.

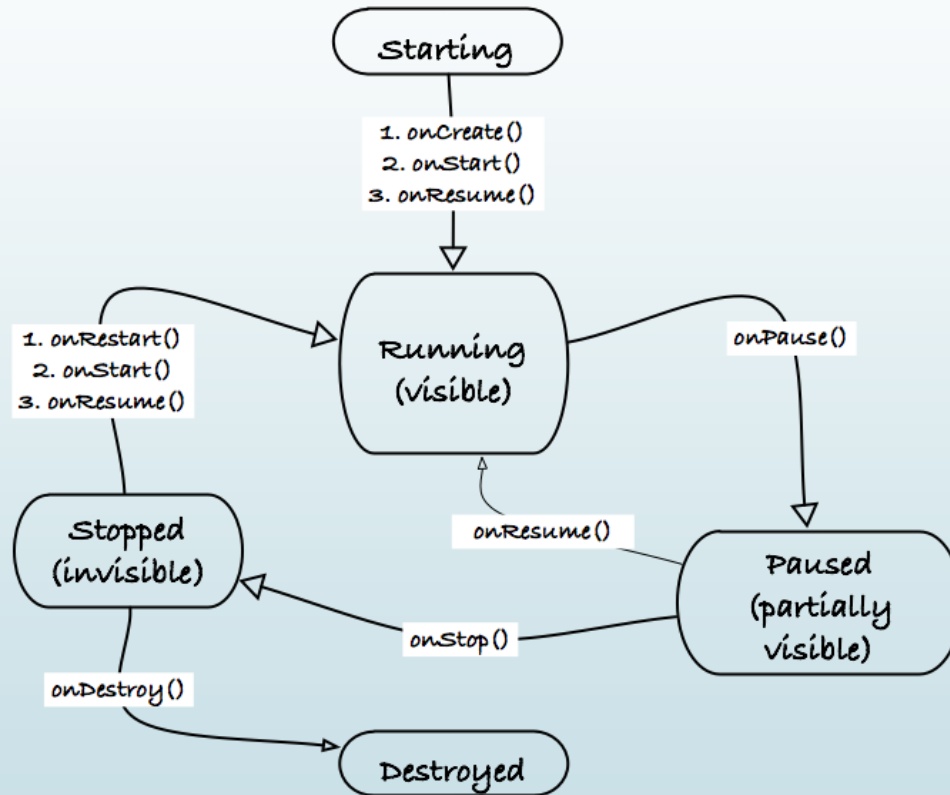
## Cycle de vie d'une activité

Les Principaux états d'une activité :

- **Active**  
Détenant le focus utilisateur grâce à la méthode `onResume()` ou `onStart()`.
- **Paused**  
Ne détenant pas le focus et attendant un resume.
- **Stopped**  
Activité non visible, toutes les variables ont été vidées et l'application n'est plus en marche.



# Cycle de vie d'une activité



# Un projet Android :

## Les activités

```
public class MainActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        // Appelée une seule fois, à la création de l'activité
    }

    @Override
    protected void onDestroy() {
        super.onDestroy();
        // Appelée une seule fois, à la destruction de l'activité
    }

    @Override
    protected void onStart() {
        super.onStart();
        // Appelée quand l'activité devient visible
    }

    @Override
    protected void onStop() {
        super.onStop();
        // Appelée quand l'activité n'est plus visible
    }
}
```

# Un projet Android :

## Les activités

```
@Override
protected void onRestart() {
    super.onRestart();
    // Appelée quand l'activité redevient visible après un stop
}

@Override
protected void onResume() {
    super.onResume();
    // Appelée quand l'activité prend ou reprend le focus utilisateur
}

@Override
protected void onPause() {
    super.onPause();
    // Appelée quand l'activité perd le focus utilisateur
}
}
```



# Un project Android :

## Les activités

- Notes :

- Une activité hérite de la classe **Activity**, ou d'une de ses sous-classes.

- Exemples de spécialisations :

- AppCompatActivity, ListActivity, PreferenceActivity,...

- Toutes les activités, en tant que composants Android, doivent être déclarées dans le **manifeste** de l'application.



# Un project Android : Exercices

1. Modifier l'application "Hello World" afin qu'elle affiche un autre texte.
2. Créer un nouveau projet, et utiliser les logs pour observer les différentes méthodes qui sont appelées. (Explication de Logcat en annexe !)



# Un project Android :

## Les ressources

- Les **ressources** sont des **fichiers** externes ne contenant pas d'instructions.
- Elles sont utilisées par le code et liées à l'application au moment de sa construction.
- Elles sont placées et triées dans le dossier "**res**" de votre projet.
- Android supporte un grand nombre de fichiers :
  - JPEG, PNG, XML, MPEG,...





# Un project Android : Les ressources

- On distingue plusieurs **types** de **ressources** différentes, triées par **dossiers**
  - anim
  - animator
  - color
  - drawable
  - interpolator
  - layout
  - menu
  - mipmap
  - raw
  - transition
  - values
  - xml

# Un projet Android :

## Les ressources

- Pour créer des fichiers de **ressources**, il faut utiliser le clic droit sur le dossier "res", puis *New/Android Resource File*
- Certains fichiers sont déjà générés à la **création** de votre projet, comme le fichier *strings.xml*, qui contient des ressources de type **chaînes de caractères**.
- Par convention, les fichiers dans le dossier *values* sont nommés par le pluriel de la ressource qu'ils contiennent.
  - strings.xml contient les éléments `<string>`
  - colors.xml contient les éléments `<color>`
  - etc,...
- Note : les fichiers de ressources doivent être nommés en minuscules, sans espaces, accents ni autres caractères spéciaux. De plus, les caractères spéciaux qu'ils contiennent doivent être **échappés** grâce au caractère "\\".



# Un project Android : Les ressources

- Habituellement, les fichiers *values* s'écrivent comme suit :

```
<resources>
```

```
    <type name="res_name">valeur</type>
```

```
</resources>
```



# Un projet Android :

## Les ressources

- Les **ressources** sont accessibles différemment depuis une classe java ou depuis un autre fichier de ressource XML.
- Dans un code java, les ressources sont accessibles et utilisées depuis le code grâce à la **classe statique R**. Cette classe est **automatiquement générée** en fonction des ressources présentes dans votre projet au moment de la compilation et de la construction de l'application.

- Exemple d'utilisation d'une ressource :

```
setContentView(R.layout.mon_fichier_xml);
```

- Si vous voulez manipuler des instances, vous pouvez utiliser cette syntaxe :

```
Resources resources = getResources();
```



# Un project Android : Les ressources

- Dans un code XML, les ressources sont accessibles et utilisées grâce à la notation `@{type}/{id}`
  - Exemple :
    - `@color/couleur_boutons`
    - `@string/hello_world`



# Un project Android :

## Exercices

1. En utilisant les ressources, récupérer la string « J'ai compris le système des ressources » et l'afficher dans le layout de l'activité.
2. En utilisant les ressources, récupérer la couleur #F4CCAA et la définir en tant que couleur de fond de votre TextView

### **Pour aller plus loin :**

1. Créer une activité qui affiche une image
2. Créer un objet « Application » et récupérer une variable via celui-ci



# Un project Android : L'application

- L'objet **Application** représente l'application courante.
- Tout comme une activité, l'application possède un **cycle de vie**.
- Pour créer une application, il faut étendre la classe Application et préciser son "name" dans le manifeste.
- On peut récupérer cet objet depuis une activité grâce à la méthode **getApplication()** (conversion nécessaire !);
- Quelques exemples d'utilisation :
  - Initialisation de variables « globales »
  - Utilisation des préférences
  - Action à effectuer lorsque la mémoire allouée est faible



# L'interface utilisateur

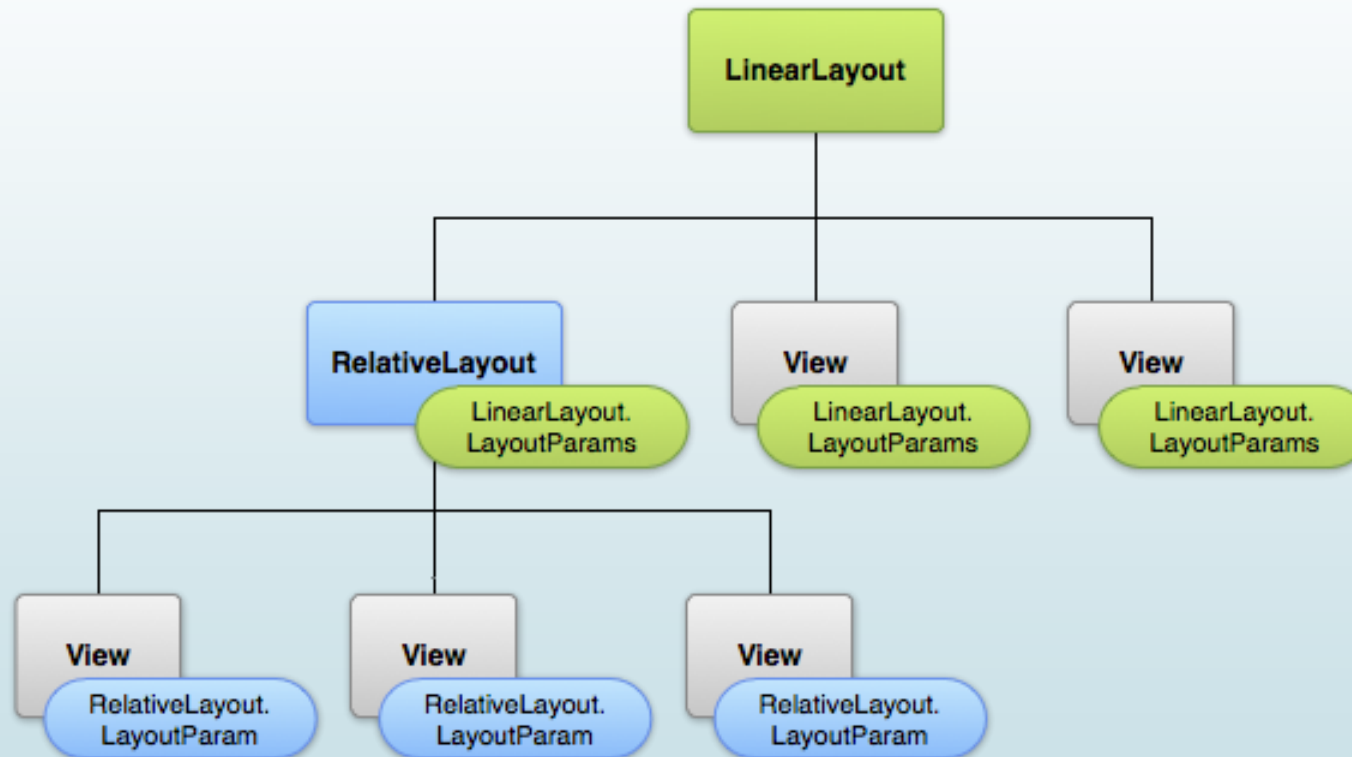




# L'interface utilisateur : Concepts

- **View** : élément graphique. Ce composant de l'interface permet souvent d'**interagir** avec l'**utilisateur**, et est paramétrable. On en trouve de nombreux types, du plus simple au plus complexe :
  - TextView : simple affichage de texte
  - EditText : entrée utilisateur
  - CheckBox : boîte à cocher
  - ...
- **Layout** : conteneur de Views. C'est le layout qui est responsable de la majeure partie du **positionnement** de ses enfants. Il en existe différents types :
  - LinearLayout (horizontal ou vertical)
  - RelativeLayout
  - GridLayout

# L'interface utilisateur : Composition



# L'interface utilisateur : Views

+	+	+
06	août	2015
-	-	-

★ ★ ★ ★ ★

Username

☐ Oui

☐ Non

☐ J'accepte les conditions

# L'interface utilisateur : Exemple

Ce que nous souhaitons obtenir.



Structure réelle





# L'interface utilisateur : Premier layout

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    tools:context=".MainActivity">

    <!-- ADD VIEWS OR OTHER LAYOUTS HERE -->

</LinearLayout>
```



# L'interface utilisateur : Premier layout

- La racine est **LinearLayout**.
- La propriété **orientation**=« vertical » définit le sens dans lequel les éléments seront positionnés.
- Attributs communs à tous les layouts (**layout\_width**, **layout\_height**)
  - La valeur `match_parent` spécifie que le layout doit prendre toute la place disponible sur la largeur/hauteur.
  - La valeur `wrap_content` précise que le layout ne doit prendre que la place nécessaire.
- Attributs spécifiques à chaque layout.



# L'interface utilisateur : Premier layout

- Les **dimensions** des éléments peuvent être précisées en plusieurs unités (px, in, mm, pt, dp, sp, ...).
- Il est préférable d'utiliser le **dp** pour les éléments et images, et le **sp** pour la police.
- Ces deux unités **s'adaptent** en effet à la résolution de l'écran et rendent les applications plus **portables**.
- Utilisez **toujours** les dp et sp sauf s'il vous est impossible de faire autrement.



# L'interface utilisateur :

## Premier layout

### ► Note :

- Px : Pixels - corresponds to actual pixels on the screen.
- In : Inches - based on the physical size of the screen.  
1 Inch = 2.54 centimeters
- Mm : Millimeters - based on the physical size of the screen.
- Pt : Points - 1/72 of an inch based on the physical size of the screen.
- Dp : Density-independent Pixels - an abstract unit that is based on the physical density of the screen. These units are relative to a 160 dpi screen, so one dp is one pixel on a 160 dpi screen. The ratio of dp-to-pixel will change with the screen density, but not necessarily in direct proportion.
- Sp : Scale-independent Pixels - this is like the dp unit, but it is also scaled by the user's font size preference. It is recommend you use this unit when specifying font sizes, so they will be adjusted for both the screen density and user's preference.



## L'interface utilisateur : UI et logique

Cette interface définit un layout de type `LinearLayout` vertical, et un composant graphique d'affichage de texte de type `TextView`.

Vous remarquerez que nous avons défini un attribut `android:id` avec la valeur `@+id/monText`. Cette valeur nous permettra de faire référence à cet élément dans le code avec la constante `R.id.monText`.

Pour le moment nous n'allons pas détailler les éléments et les attributs : nous y reviendrons plus loin dans ce chapitre.

```
MainActivity.java x activity_main.xml x
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools" android:layout_width="match_parent"
    android:layout_height="match_parent" android:paddingLeft="16dp"
    android:paddingRight="16dp"
    android:paddingTop="16dp"
    android:paddingBottom="16dp"
    android:orientation="vertical"
    android:gravity="center_horizontal"
    tools:context=".MainActivity">

    <TextView
        android:id="@+id/tv_main_hello"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="@string/hello_world"
        android:textSize="18sp"
        android:layout_margin="10dp"/>

</LinearLayout>
```

## L'interface utilisateur : UI et logique

L'interface a défini un composant graphique TextView vide.

Pour y accéder depuis le code et pouvoir le manipuler, vous devez d'abord récupérer une instance de l'objet avec la méthode findViewById de l'activité.

Une fois l'objet récupéré, vous pourrez le manipuler de la même façon que n'importe quel objet, par exemple pour modifier son texte. L'affichage répercutera le changement.

```
MainActivity.java x activity_main.xml x
package be.formation.app.helloworld;

import ...

public class MainActivity extends AppCompatActivity {

    private TextView tv_main_hello;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

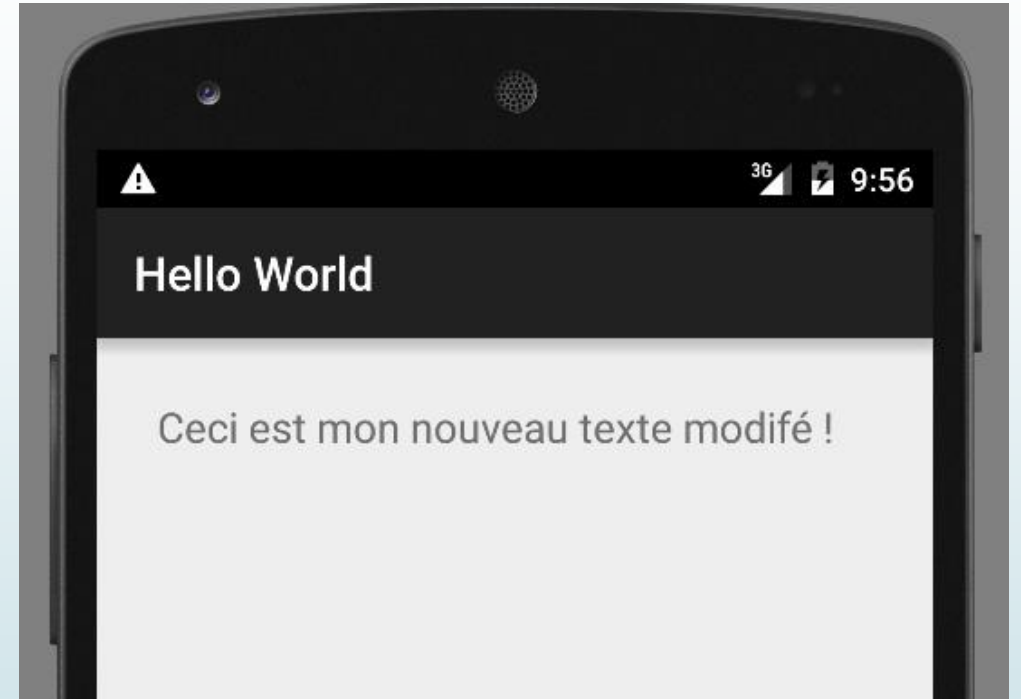
        tv_main_hello = (TextView) findViewById(R.id.tv_main_hello);
        tv_main_hello.setText("Ceci est mon nouveau texte modifié !");
    }
}
```

## L'interface utilisateur : UI et logique

Ce résultat pouvait être obtenu sans utiliser de définition XML, directement en écrivant du code Java.

Plusieurs problèmes avec cette méthode :

- Non-séparation de la logique et de la présentation.
- Perte de maintenabilité (Graphiste <> Développeur).





## L'interface utilisateur : UI et logique

Exemple de définition d'une interface en Java.

Attention ! Ceci est une pratique à éviter.

```
LinearLayout monLinearLayout = new LinearLayout(this);  
monLinearLayout.setOrientation(LinearLayout.HORIZONTAL);  
  
TextView myText = new TextView(this);  
monLinearLayout.addView(myText);  
  
setContentView(monLinearLayout);  
  
myText.setText("Bonjour tout le monde !");
```



# L'interface utilisateur :

## Les évènements

- Sous Android, toute **action effectuée par un utilisateur** est considérée comme un évènement.
- Un évènement peut être **intercepté** par un élément de l'interface.
- L'interception s'effectue grâce à des objets appelés « **Listeners** ».
- Un listener permet d'associer un évènement à une méthode qui sera appelée lors de l'apparition de l'évènement.



# L'interface utilisateur :

## Les évènements

- `onClick()` de `View.OnClickListener`.
  - Elle est appelée lorsque l'utilisateur touche n'importe quel élément (en mode Touch), ou place le focus sur l'élément grâce aux touches de navigation.
- `onLongClick()` de `View.OnLongClickListener`.
  - Même chose que la méthode précédente, mais l'action doit durer au moins une seconde ( $\geq 1$  sec).
- `onFocusChange()` de `View.OnFocusChangeListener`.
  - Elle est appelée lorsque l'utilisateur place ou enlève le focus sur l'élément en utilisant les touches directionnelles.
- `onKey()` de `View.OnKeyListener`.
  - Elle est appelée lorsque l'utilisateur a mis le focus sur l'élément et qu'il presse ou relance une touche de l'appareil.
- `onTouch()` de `View.OnTouchListener`.
  - Elle est appelée lorsque l'utilisateur opère une action qualifiée d'événement de toucher, incluant une pression (press), un relachement (release), ou tout autre mouvement ou trajectoire (opérée dans les limites de l'élément).



# L'interface utilisateur : Les évènements

► En pratique :

```
Button btn_main_send = (Button) findViewById(R.id.btn_main_send);  
btn_main_send.setOnClickListener(new OnClickListener() {  
    @Override  
    public void onClick() {  
        // TODO  
    }  
});
```



# L'interface utilisateur :

## Quelques attributs

### ■ EditText

- hint : texte qui s'affichera tant que le champs est vide, pour donner des indications à l'utilisateur (équivalent à un placeholder en HTML)
- inputType : choix du type de texte qui sera entré par l'utilisateur. Cela permet d'adapter le clavier au contenu de l'EditText (équivalent à un type en HTML)

### ■ ImageView

- src : c'est évidemment la source du fichier à afficher dans l'ImageView. Utilisez la notation `@drawable/nom_du_fichier` pour retrouver une ressource drawable.
- scaleType : permet de définir le comportement de l'image si sa taille ou son ratio ne correspond pas à l'ImageView. Essayez "centerInside" ou "centerCrop".

### ■ Global

- gravity : définit le comportement du positionnement des éléments enfants.
- layout\_gravity : définit le comportement du positionnement de l'élément lui-même.





# L'interface utilisateur : Exercices

1. Créer un formulaire de login utilisateur. Il contiendra 4 views :

- Username
- Password
- Boutons de Login et de Reset

A la validation, vous loguerez ce que l'utilisateur a entré.

2. Créer un formulaire d'inscription d'un utilisateur. Le choix des éléments vous revient !



# L'interface utilisateur : Conclusion

- L'interface utilisateur est un **élément clé** de la réussite d'une application.
- Android met pour cela en place des mécanismes simples pour l'interfaçage.
- La formation avancée montre comment créer ses propres composants, styles, et thèmes.
- Une interface **épurée** et proche des standards reste la meilleure des solutions L'utilisateur aura ainsi l'impression d'utiliser une application native.



# Intent et navigation



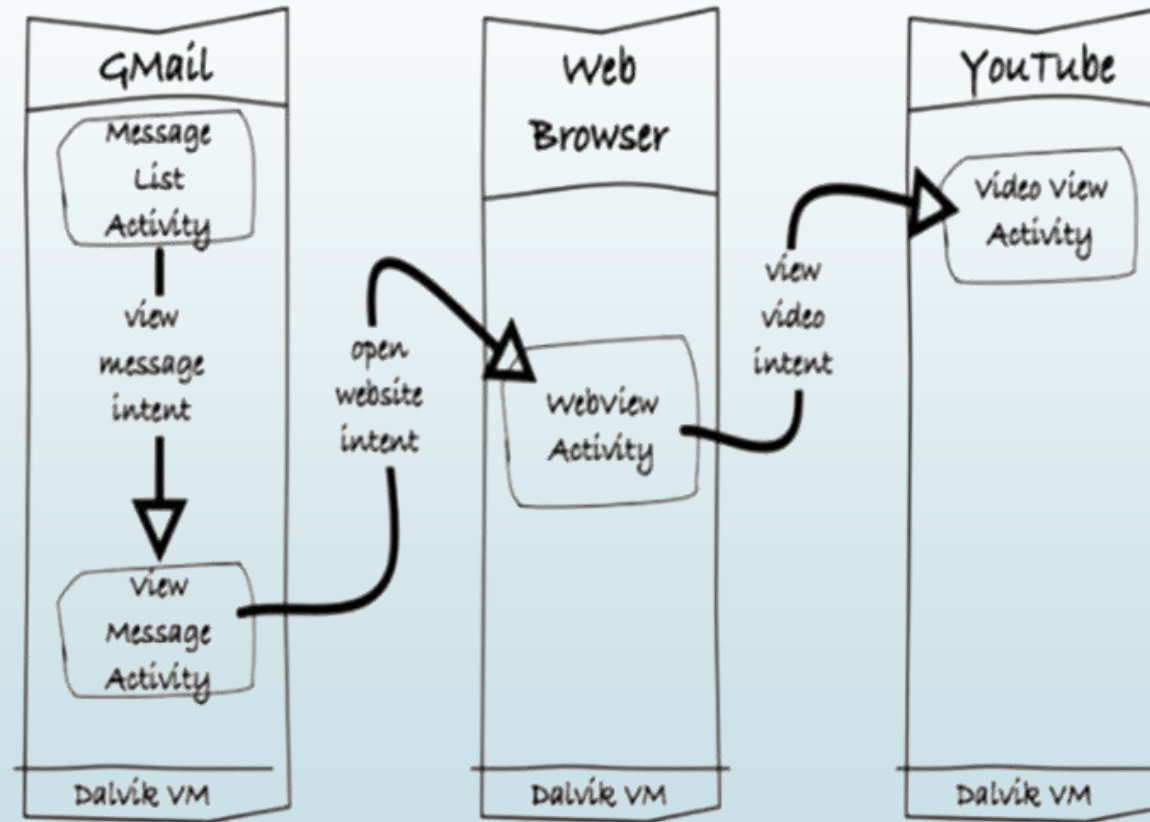
# Intent et navigation :

## Le concept d'Intent

- La **communication** interne du système android se base sur l'envoi et la réception de **messages**.
- Ceux-ci expriment **l'intention** d'une **action** et représentent une description abstraite d'une opération à effectuer.
- Un message peut être utilisé pour communiquer avec plusieurs **composants**
  - Une activité
  - Un service
  - Une autre application
- Un message peut également contenir des **données**, appelées "**Extra**", sous la forme d'un objet **Bundle**

# Intent et navigation :

## Le concept d'Intent





# Intent et navigation :

## Le concept d'Intent

- Le message est issu de la classe **Intent** et véhicule toutes les **informations** nécessaires à la réalisation de l'action.
- Le démarrage d'une activité ou d'un service, par exemple, est effectué au moyen d'un objet Intent.
- L'objet correspond au **souhait d'une action**, mais sans en expliquer les modalités. Le **système** récepteur s'occupera de choisir l'application à lancer.
  - Note : Impossible par exemple de demander au système de lancer un navigateur particulier ! C'est le système qui se charge de résoudre notre intention, éventuellement en demandant à l'utilisateur sa préférence.



# Intent et navigation : Le concept d'Intent

- Deux façons d'envoyer des **Intents** au système :
  - En **ciblant** un **composant précis** de l'application (mode **explicite**).
  - En **déléguant** au système le traitement de la demande (mode **implicite**).
- C'est à chaque application de **filtrer** et de gérer les **Intent** qui sont pertinents pour celle-ci.
  - Une application peut être inactive tout en restant à l'écoute des intentions circulant dans le système.
  - Les filtres d'Intent sont abordés en formation avancée.



# Intent et navigation : Principe

- Trois utilisations pour les objets **Intent** :
  - Démarrage d'une **activité** au sein d'une même application.
  - Solliciter une autre **application**.
  - Démarrage d'un **service** (tâche s'effectuant en arrière-plan).
- L'objet **Intent** :
  - Véhicule toutes les informations nécessaires à la réalisation d'une **action** ou à la réception d'une information.
  - Prend une série de **paramètres** à sa construction qui seront détaillés tout au long de ce chapitre.





# Intent et navigation :

## Entre activités dans une application

- Une activité peut être assimilée à un **écran** de l'application, il est cependant assez rare qu'une application soit composée d'un seul écran. Il faut un moyen de **démarrer** une **nouvelle activité**.
- Chaque composant d'une application nécessitera un **Intent** pour être démarré.
- Deux façons de faire :
  - Avec valeur de retour
  - Sans valeur de retour

# Intent et navigation :

## Entre activités dans une application

- Démarrer une **activité** sans attendre de retour se fait au moyen de la méthode `startActivity()`;

```
Intent intent = new Intent(this,ActiviteADemarrer.class);  
startActivity(intent);
```

- Deux paramètres pour la construction d'un objet **Intent** :
  - **Context** `PackageContext` : le contexte à partir duquel l'Intent est créé et sera renvoyé. La plupart du temps, il s'agit de l'activité en cours, `this`.
  - **Class**<?> `cls` : Une classe Java héritant de la classe `Activity`.
- N'oubliez pas de déclarer l'activité dans le manifeste... Sinon c'est une `ActivityNotFoundException`



# Intent et navigation :

## Exercices

1. Créer une activité démarrant une sous-activité à l'aide d'un bouton.



# Intent et navigation :

## Entre activités dans une application

- Démarrer une **activité** avec une valeur de **retour** se fait au moyen de la méthode `startActivityForResult()`;
- Grâce à cette méthode, lorsque l'activité enfant aura terminé sa tâche, elle en avertira l'activité **parent**.



# Intent et navigation : Entre activités dans une application

```
Private static final int CODE_ACTIVITE = 1;
```

```
Intent intent = new Intent(this, maClasse.class);
```

```
startActivityForResult(intent, CODE_ACTIVITE);
```

- La constante CODE\_ACTIVITE permettra d'identifier quelle activité a provoqué le retour de valeur.



## Intent et navigation : Entre activités dans une application

Utilisation de la méthode  
`setResult()`; dans la sous-activité  
en précisant le code de retour.

```
@Override
public void onClick(View v) {
    switch(v.getId()){
        case R.id.button1:
            setResult(RESULT_OK);
            finish();
            break;

        case R.id.button2:
            setResult(RESULT_CANCELED);
            finish();
            break;
    }
}
```

## Intent et navigation : Entre activités dans une application

Utilisation de la méthode `onActivityResult()` de l'activité dans laquelle a été appelée la méthode `startActivityForResult()`.

Trois paramètres pour la méthode `onActivityResult()` :

- `int requestCode` : code identifiant l'activité qui a été appelée par la méthode `startActivity()` ou `startActivityForResult()`.
- `int resultCode` : Valeur de retour renvoyée à l'activité parente.
- `Intent data` : Permet l'échange de données.

```
@Override
protected void onActivityResult(int requestCode
    , int resultCode
    , Intent data) {
    // Le code de requête est utilisé pour identifier l'activité enfant
    switch(requestCode){
    case CODE_MON_ACTIVITE:
        switch(resultCode){
        case RESULT_OK:
            Toast.makeText(this
                , "Action validée"
                , Toast.LENGTH_LONG).show();
            return;
        case RESULT_CANCELED:
            Toast.makeText(this
                , "Action annulée"
                , Toast.LENGTH_LONG).show();
            return;
        default:
            // Faire quelque chose
            return;
        }
    default:
        // Faire quelque chose
        return;
    }
}
```



# Intent et navigation :

## Exercices

1. Créer une activité et une sous activité affichant un choix (deux boutons cliquables : valider et annuler), récupérer la valeur du choix et l'afficher dans l'activité parente.





# Intent et navigation : Embarquer des données

- Il arrive constamment que l'on doive passer des **informations** à l'activité que l'on souhaite démarrer.
- La classe **Intent** dispose de méthodes **putExtra()** et **getExtras()** grâce auxquelles un objet de type **Bundle** véhiculera vos données d'une activité à une autre.
- Ces méthodes permettent d'envoyer et de récupérer tous les **types primitifs**, ainsi que les objets qui implémentent les interfaces **Serializable** ou **Parcelable**.
- L'interface Parcelable, plus complexe et propre à Android, est abordée en formation avancée.

# Intent et navigation :

## Embarquer des données

➤ Activité parent

```
Intent intent = new Intent(this, MonActivite.class);  
intent.putExtra("maclé", "hello !");  
startActivity(intent);
```

➤ Activité enfant

```
Bundle extra = this getIntent().getExtras();  
if(extra != null)  
    String data = extra.getString("maclé");
```



# Intent et navigation :

## Exercices

1. Créer une activité qui propose un EditText et un Button. Lorsqu'on clique sur le bouton, un Intent est envoyé à une deuxième activité qui affiche simplement la valeur entrée dans l'EditText.

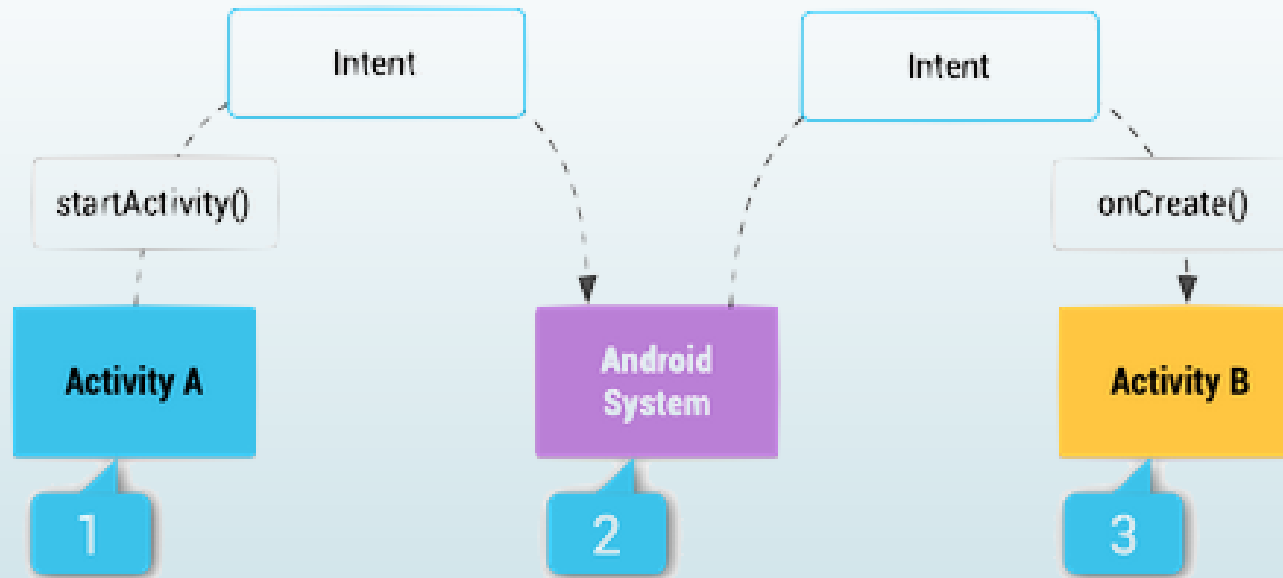


# Intent et navigation : Solliciter d'autres applications

- L'envoi d'un **Intent** permet également de demander à un composant d'une autre application de réaliser l'**action** voulue.
- Dans ce cas, le système prend la décision de l'application à utiliser.
  - Pour prendre cette décision, le système va utiliser l'objet Intent que l'on va lui passer.
- Nous exprimons donc une intention au système et il se chargera de proposer le composant à utiliser. Le système choisit, à l'**exécution** et non à la compilation, le composant à utiliser.

# Intent et navigation :

## Solliciter d'autres applications





# Intent et navigation : Solliciter d'autres applications

- Pour demander au système de réaliser une action, nous utilisons simplement un objet **Intent** auquel on spécifie :
  - Le type d'**action** à réaliser.
  - Des valeurs complémentaires.
- Il ne faut pas préciser la classe ciblée !
- Par exemple, pour demander à composer un numéro de téléphone :
  - On précisera l'action ACTION\_DIAL.
  - Un URI comportant le numéro à appeler.
- **Attention !** Certains types d'action nécessitent l'ajout d'une **permission** dans le manifeste.

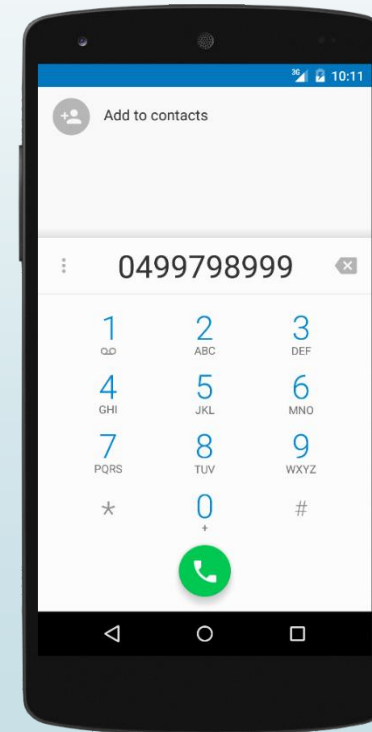
Ex : `<uses-permission android:name="android.permission.CALL_PHONE" />`

# Intent et navigation : Solliciter d'autres applications

► Code de l'exemple précédent :

```
Uri phoneNumber = Uri.parse("tel:0499798999");  
Intent phoneIntent = new Intent(Intent.ACTION_DIAL, phoneNumber);  
startActivity(phoneIntent);
```

► Et voici le résultat :



# Intent et navigation :

## Solliciter d'autres applications

- Quelques exemples d'Intent communs :

- Créer une alarme

- Action : [ACTION SET ALARM](#)

- Extra : [EXTRA\\_HOUR](#), [EXTRA\\_MINUTES](#), [EXTRA\\_MESSAGE](#), [EXTRA\\_DAYS](#),...

- Permission nécessaire : [SET\\_ALARM](#)

- Exemple :

```
public void createAlarm(String message, int hour, int minutes) {  
    Intent intent = new Intent(AlarmClock.ACTION_SET_ALARM)  
        .putExtra(AlarmClock.EXTRA_MESSAGE, message)  
        .putExtra(AlarmClock.EXTRA_HOUR, hour)  
        .putExtra(AlarmClock.EXTRA_MINUTES, minutes);  
    if (intent.resolveActivity(getPackageManager()) != null) {  
        startActivity(intent);  
    }  
}
```



# Intent et navigation :

## Solliciter d'autres applications

- Ajouter un évènement au calendrier

- Action : [ACTION\\_INSERT](#)

- Data URI : [Events.CONTENT\\_URI](#)

- Extra : [EXTRA\\_EVENT\\_BEGIN\\_TIME](#), [EXTRA\\_EVENT\\_END\\_TIME](#), [TITLE](#), [DESCRIPTION](#),...

- Exemple :

```
public void addEvent(String title, String location, Calendar begin, Calendar end) {  
    Intent intent = new Intent(Intent.ACTION_INSERT)  
        .setData(Events.CONTENT_URI)  
        .putExtra(Events.TITLE, title)  
        .putExtra(CalendarContract.EXTRA_EVENT_BEGIN_TIME, begin)  
        .putExtra(CalendarContract.EXTRA_EVENT_END_TIME, end);  
    if (intent.resolveActivity(getPackageManager()) != null) {  
        startActivity(intent);  
    }  
}
```

# Intent et navigation : Solliciter d'autres applications

- Appeler un numéro

- Action : [ACTION\\_CALL](#)

- Data URI : `tel:<phone-number>`

- Permission : [CALL\\_PHONE](#)

- Exemple :

```
public void dialPhoneNumber(String phoneNumber) {  
    Intent intent = new Intent(Intent.ACTION_DIAL);  
    intent.setData(Uri.parse("tel:" + phoneNumber));  
    if (intent.resolveActivity(getPackageManager()) != null) {  
        startActivity(intent);  
    }  
}
```

# Intent et navigation : Solliciter d'autres applications

- Faire une recherche sur le Web
  - Action : [ACTION\\_WEB\\_SEARCH](#)
  - Extra : [SearchManager.QUERY](#)
  - Exemple :

```
public void searchWeb(String query) {  
    Intent intent = new Intent(Intent.ACTION_SEARCH);  
    intent.putExtra(SearchManager.QUERY, query);  
    if (intent.resolveActivity(getPackageManager()) != null) {  
        startActivity(intent);  
    }  
}
```

# Intent et navigation : Solliciter d'autres applications

- Ouvrir une page Web

- Action : [ACTION\\_VIEW](#)

- Data URI : http:<URL> OU https:<URL>

- Extra : [SearchManager.QUERY](#)

- Exemple :

```
public void openWebPage(String url) {  
    Uri webpage = Uri.parse(url);  
    Intent intent = new Intent(Intent.ACTION_VIEW, webpage);  
    if (intent.resolveActivity(getPackageManager()) != null) {  
        startActivity(intent);  
    }  
}
```



# Intent et navigation : Solliciter d'autres applications

- D'autres Intent existent et permettent d'ouvrir d'autres types d'application et/ou d'effectuer d'autres actions.
- Plus d'informations :

<https://developer.android.com/guide/components/intents-common.html>

<http://developer.android.com/reference/android/content/Intent.html>

# Intent et navigation :

## Les permissions

- À partir de l'API 23, il est obligatoire d'effectuer certaines demandes de permission en RunTime.
- Cette requête s'effectue sur base de deux étapes, celles-ci représentées par des conditions

```
if (ActivityCompat.checkSelfPermission(this, Manifest.permission.CALL_PHONE) != PackageManager.PERMISSION_GRANTED) {  
    if (!ActivityCompat.shouldShowRequestPermissionRationale(this, Manifest.permission.CALL_PHONE)) {  
        ActivityCompat.requestPermissions(this, new String[] {Manifest.permission.CALL_PHONE}, MY_PERMISSIONS_REQUEST_CALL_PHONE);  
    }  
}
```



# Intent et navigation :

## Exercices

1. Créer une application qui appellera le numéro de téléphone entré par l'utilisateur.
2. Créer une application qui va permettre de faire une recherche Google.



# Intent et navigation :

## Conclusion

- La classe **Intent** a un rôle **crucial** dans le système android, elle permet de:
  - Lancer une **activité**
  - Démarrer un **service**
  - Passer des **paramètres** à une activité
  - Solliciter d'autres **applications** pour exécuter une **action** spécifique
- La maîtrise des Intents est absolument **obligatoire** pour tout développeur de la plate-forme android.





# Listes et adaptateurs



# Listes et adaptateurs

## La ListView

- En Android, les listes d'éléments cliquables sont gérés par une **ListView**. C'est une View comme les autres, qu'il suffit d'insérer dans son XML.
- Contrairement aux autres Views, la ListView permet automatiquement de scroller si les éléments sont trop nombreux pour s'afficher.
- Les éléments d'une ListView sont généralement ajoutés dynamiquement dans le code, pour cela on utilise un **adaptateur**.



# Listes et adaptateurs

## L'adaptateur

- L'**adaptateur** est un modèle de conception offrant une couche d'abstraction avec des méthodes fortement typées pour accéder aux données.
- Celui-ci va gérer les **sources de données** qui seront affichées dans les différentes listes de l'interface utilisateur.
- Techniquement, il s'agit d'un objet Adapter qui est une interface entre les **données** et leur **affichage**.



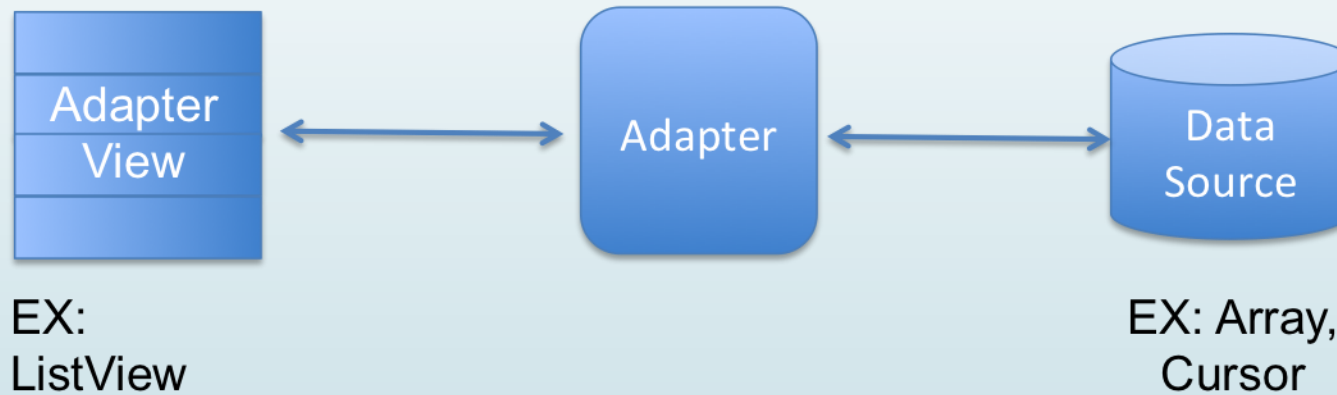
# Listes et adaptateurs

## Les adaptateurs

- Pour afficher une **liste** d'éléments cliquables, il nous faut :
  - Des **données** (Array, ArrayList, Cursor, ...)
  - Un **Adapter** qui fera l'interface entre les données et les vues
  - Une **ListView** qui affichera les données, ainsi disponibles pour l'utilisateur
- Adapter étant une interface, il est impossible d'avoir un objet Adapter utilisable en l'état.
- Il existe plusieurs implémentations permettant de traiter les types les plus courants :
  - ArrayAdapter<T> : Tableaux avec T générique
  - SimpleCursorAdapter : Traitement des données de type Cursor
  - BaseAdapter : Adaptateurs personnalisés
  - SimpleAdapter : ArrayList de Maps
  - ...

# Listes et adaptateurs

## Introduction



# Listes et adaptateurs

## Exemple

1. Créer une activité qui contient une ListView définie en XML.

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    tools:context=".MainActivity">

    <ListView
        android:id="@+id/lv_main_list"
        android:layout_width="match_parent"
        android:layout_height="match_parent"/>

</LinearLayout>
```

# Listes et adaptateurs

## Exemple

2. Récupérer cette ListView dans votre code java, et créer des données sous la forme d'un tableau de String.

```
public class MainActivity extends AppCompatActivity {  
  
    ListView lv_main_list;  
  
    @Override  
    protected void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        setContentView(R.layout.activity_main);  
  
        lv_main_list = (ListView) findViewById(R.id.lv_main_list);  
  
        String[] data = new String[] { "The Lord of the Rings", "Matrix", "Interstellar", "Fight Club" };  
    }  
}
```

# Listes et adaptateurs

## Exemple

3. Enfin, créer un ArrayAdapter de String et le passer à la méthode `setAdapter()` de la `ListView`. Le constructeur de l'`ArrayAdapter` prend un `Context`, un `Layout`, l'id d'une `TextView` de ce `Layout` ainsi que vos données.

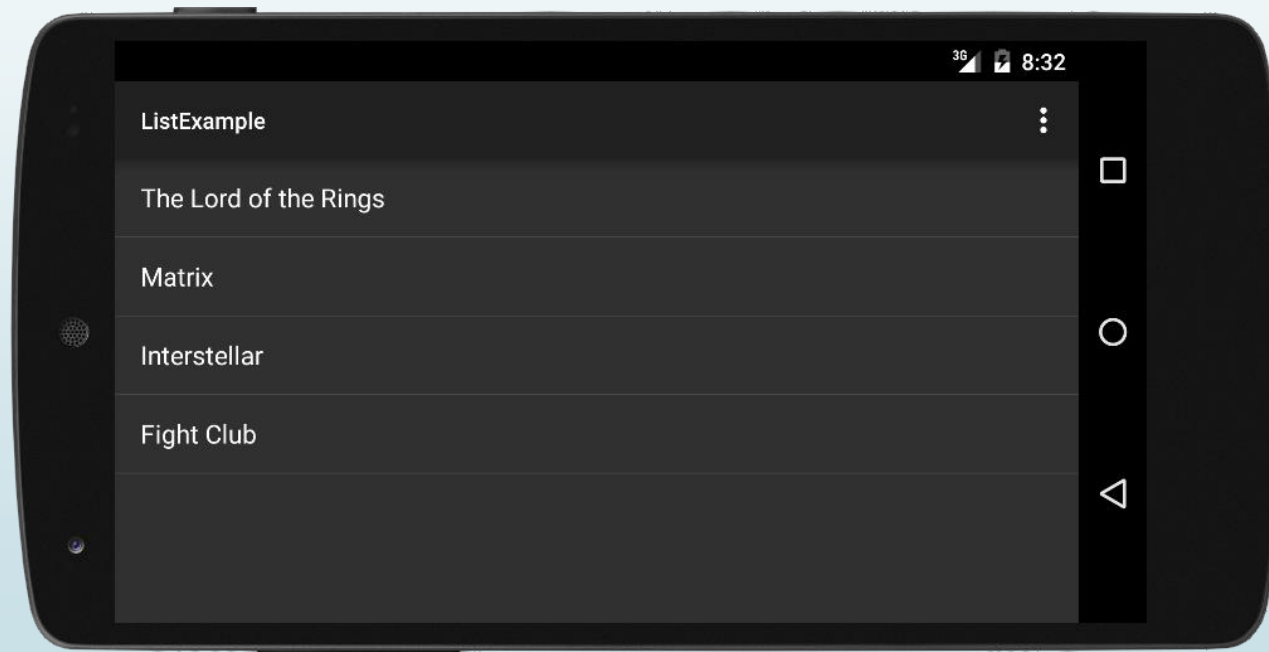
```
ArrayAdapter<String> adapter = new ArrayAdapter<String>(getApplicationContext(),  
    android.R.layout.simple_list_item_1,  
    android.R.id.text1,  
    data);  
  
lv_main_list.setAdapter(adapter);
```



# Listes et adaptateurs

## Exemple

4. Voici le résultat :





# La persistance des données



# La persistance des données

## Introduction

- Toute application se doit de pouvoir **conserver** ou **charger** des données : préférences de l'utilisateur, dernière localisation, fichiers créés par l'application,...
- Il existe plusieurs façons d'assurer la persistance des données en Android :
  - Les objets "Bundle" qui permettent de conserver l'état d'une activité.
  - Les "Preferences" qui permettent de conserver les préférences de l'utilisateur sous la forme de couples clé/valeur.
  - Les fichiers, stockage brut de données depuis votre application.
  - Les bases de données internes, qui permettent d'enregistrer et de récupérer des données structurées.
  - Le stockage externe au téléphone (abordé en formation avancée), via un webservice ou une librairie.



# La persistance des données

## Le Bundle

- Un Bundle est un moyen d'enregistrer l'état d'une activité.
- La persistance se fera la plupart du temps via les méthodes de cycle de vie de l'activité.
- Toutes les informations relatives à l'état de l'activité seront stockées dans un objet Bundle.



# La persistance des données

## Le Bundle

- La méthode `onSaveInstanceState()` est appelée lors de la sauvegarde de l'état d'une activité.
- L'objet ainsi créé est de type `Bundle` :
  - Passé à la méthode `onCreate()`
  - Passé à la méthode `onRestoreInstanceState()`



# La persistance des données

## Le Bundle

- La méthode `putString("key", "value")` permet de sauvegarder un couple clé/valeur grâce au Bundle.
- La méthode `getString("key")` récupère la valeur placée dans la clé.
- La méthode `clone()` retourne un objet Bundle exactement identique au précédent.
- La méthode `remove("key")` supprime le couple dont la clé est indiquée en paramètre.



# La persistance des données

## Le Bundle

- Avantages de cette méthode :
  - **Simplicité** extrême, aucun traitement de la part du développeur n'est requis.
  - Mécanisme **natif** et implémenté par défaut.
- Inconvénients :
  - Les informations ne sont pas accessibles à d'autres applications.
  - On ne peut conserver avec cette méthode, que les informations relatives à **une** interface graphique.
  - Quand l'utilisateur presse le bouton « Back », les variables sont détruites.



# La persistance des données

## Les Shared Preferences

- Les préférences sont stockées dans un fichier XML, sous la forme de couples clé/valeur.
- Deux façons d'obtenir un fichier de préférence :
  - La méthode `getSharedPreferences(nomFichier, mode)`.  
mode peut prendre trois valeurs :
    - `MODE_PRIVATE` (accessibles uniquement à l'activité créatrice).
    - `MODE_WORLD_READABLE` (accessibles à toute l'application en lecture seule).
    - `MODE_WORLD_WRITABLE` (accessibles à toute l'application en lecture/écriture).
  - La méthode `PreferenceManager.getDefaultSharedPreferences(context)`.
    - Plus simple à utiliser (pas de nom de fichier, moins de risque d'erreur).
    - Utilisé par les menu de préférences (voir slides suivants).
    - À privilégier !





# La persistance des données

## Les Shared Preferences

- Modification des Shared Preferences :
  - L'enregistrement se fait via l'interface *Editor*, récupérée grâce à la méthode *edit()* de la classe *SharedPreferences*.
  - Utilisation des différentes méthodes *put* (*putInt()*, *putString()*,...) avec en paramètre le couple clé/valeur à sauvegarder.
  - Cette méthode permet également de modifier un couple déjà sauvegardé.
  - Une fois les modifications effectuées, appel de la méthode *commit()* ou *apply()* sur l'objet *Editor*.
    - *commit()* sauvegarde les modifications directement, alors que *apply()* fait les modifications en background.



# La persistance des données

## Les Shared Preferences

► Exemple :

```
SharedPreferences prefs = PreferenceManager.getDefaultSharedPreferences(getApplicationContext());  
SharedPreferences.Editor editor = prefs.edit();  
editor.putBoolean("newsletter", true);  
editor.putString("email", "user@mail.com");  
editor.apply();
```

# La persistance des données

## Les Shared Preferences

- Récupération des Shared Preferences :
  - On utilise simplement les différentes méthodes *get* correspondantes au type souhaité (*getInt()*, *getString()*,...).
    - Ces méthodes prennent en paramètre la clé, ainsi qu'une valeur par défaut renvoyée dans le cas où aucune valeur n'est trouvée.
- Exemple :

```
SharedPreferences prefs = PreferenceManager.getDefaultSharedPreferences(getApplicationContext());  
boolean newsletter = prefs.getBoolean("newsletter", false);  
String email = prefs.getString("email", "");
```



# La persistance des données

## Exercice

### 1. Créer un petit formulaire

- EditText : Email
  - EditText : Password
  - CheckBox : Remember me
  - Button : Login
- Sauver ces informations dans les préférences partagées, et tentez de les récupérer :
- Lors du lancement de l'application.
  - Dans une seconde activité.



# La persistance des données

## Les fichiers

- Le fichier est l'élément de base du système Android.
- Même s'il existe différents moyens de stockage, il reste difficile de se passer de données brutes, et donc de fichiers.
  - Données sérialisées, texte brut, fichiers binaires,...
- Android prévoit donc des classes et des méthodes pour gérer ces fichiers.



# La persistance des données

## Les fichiers

- Première méthode :
  - Création d'un objet `File` à partir du constructeur, auquel on passe en paramètre le dossier et le nom du fichier.
    - `getFilesDir()` vous retourne le chemin vers votre espace de stockage interne.
    - `getCacheDir()` vous retourne le chemin vers votre espace de cache (limité !).
  - On peut ensuite utiliser ce fichier, par exemple avec un `FileWriter` ou un `FileReader` afin d'y écrire ou d'y lire du contenu.

# La persistance des données

## Les fichiers

```
File file = new File(getFilesDir(), "info.dat");
FileWriter fileWriter = new FileWriter(file);
fileWriter.write("Hello file !");
fileWriter.close();
```

```
File file2 = new File(getFilesDir(), "info.dat");
FileReader fileReader = new FileReader(file2);
CharBuffer charBuffer = CharBuffer.allocate(24);
fileReader.read(charBuffer);
fileReader.close();

StringBuilder sb = new StringBuilder();

for (int i = 0; i < charBuffer.length(); i++) {
    sb.append(charBuffer.get(i));
}

Log.i("FILE CONTENT", sb.toString());
```

# La persistance des données

## Les fichiers

- Deuxième méthode :
  - Créer ou ouvrir un fichier :
    - `openFileOutput(nomFichier, mode)`  
mode peut prendre quatre valeurs :
      - `MODE_PRIVATE`
      - `MODE_WORLD_READABLE` (déconseillé !)
      - `MODE_WORLD_WRITABLE` (déconseillé !)
      - `MODE_APPEND` (ajoute les données au fichier au lieu d'écraser son contenu)
    - Cette méthode renvoie un objet `FileOutputStream`, sur lequel on peut appliquer la méthode `write()`, qui prend en paramètre un tableau de *byte*.
    - La lecture de ce fichier se fait ensuite via la méthode `openFileInput(nomFichier)` et un `FileInputStream`.





# La persistance des données

## Les fichiers

```
FileOutputStream fos = openFileOutput("myfile.dat", MODE_PRIVATE);  
fos.write("myText".getBytes());  
fos.close();
```

```
FileInputStream fis = openFileInput("myfile.dat");  
BufferedReader reader = new BufferedReader(new InputStreamReader(fis));  
StringBuilder out = new StringBuilder();  
String line;  
while ((line = reader.readLine()) != null) {  
    out.append(line);  
}  
reader.close();  
fis.close();  
  
Log.i("File content", out.toString());
```



# La persistance des données

## Les fichiers

- Troisième méthode :
  - Embarquer les fichiers dans l'application (et donc directement dans l'apk).
  - Il suffit de placer ces fichiers dans le répertoire res/raw de votre projet.
  - **Attention !** Cette méthode, bien que parfois indispensable, n'est pas à utiliser à l' légère ! L'apk peut vite devenir très lourd.
  - Alternative : télécharger les fichiers nécessaires après l'installation de l'application.



# La persistance des données

## Les fichiers

► Plus d'informations :

<http://developer.android.com/training/basics/data-storage/files.html>



# La persistance des données

## Exercice

1. Créer une activité qui contient :

- EditText : texte
- Button : sauvegarder le texte dans un fichier
- Button : afficher le contenu du fichier dans un Toast
- Button : supprimer le fichier



# La persistance des données

## SQLite - Introduction

- SQLite est une base de données relationnelle qui offre plusieurs avantages très utiles aux applications mobiles :
  - Légère
  - Gratuite
  - OpenSource
- SQLite est une base de données interne, et s'exécute donc sans serveur.
  - Les requêtes s'effectuent donc dans le même processus que l'application.
- La base de données appartient à l'application créatrice, elle seule peut donc y accéder.



# La persistance des données

## SQLite - Introduction

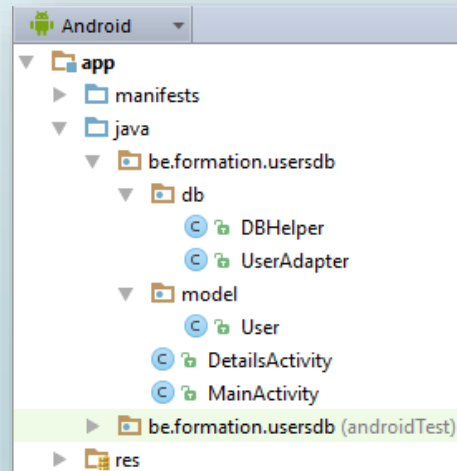
### ► **Attention !**

- Pour l'instant, tous nos appels à SQLite se feront dans le thread principal. La bonne pratique est de faire ces appels de façon asynchrones (voir formation avancée).
- Les appareils pour lesquels nous développons sont de puissance très variable, et ont des caractéristiques très diverses (CPU, RAM, Espace disque,...). Il faut donc éviter d'effectuer de trop nombreuses opérations (insertions et autres transactions).
- De même, plutôt que d'insérer des données binaires en base de données, il vaut mieux utiliser le système de fichiers, et faire référence à ce fichier dans votre db.

# La persistance des données SQLite – Mise en place

- Création d'une base de données SQLite dans un projet Android :
  - Créer un nouveau package, "db" par exemple.
  - Dans ce package, créer :
    - Une classe de gestion de votre base de données, qui étendra SQLiteOpenHelper.
    - Autant de classes DAO que de tables dans votre base de données.

➤ Exemple :





# La persistance des données

## SQLite - Le SQLiteOpenHelper

- Notre classe "DbHelper" étend SQLiteOpenHelper. Elle doit donc implémenter plusieurs méthodes :
  - *Un constructeur qui renverra les informations pertinentes au constructeur de la super classe (le contexte, le nom de la db, une éventuelle factory ainsi que le numéro de version).*
  - *onCreate() : qui sera appelée à la création de votre base de données. Elle exécutera donc tous les scripts de création des tables.*
  - *onUpgrade() : qui sera appelée à chaque incrémentation du numéro de version. Elle exécutera donc tous les scripts d'upgrade des tables.*
- *Habituellement, on définit en public static final le nom de la db, ainsi que le numéro de version.*



# La persistance des données SQLite - Le SQLiteOpenHelper

► Exemple :

```
public class DBHelper extends SQLiteOpenHelper {

    public static final String DB_NAME = "app_db";
    public static final int DB_VERSION = 1;

    public DBHelper(Context context) {
        super(context, DB_NAME, null, DB_VERSION);
    }

    @Override
    public void onCreate(SQLiteDatabase db) {
        db.execSQL(UserDAO.CREATE_REQUEST);
    }

    @Override
    public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
        db.execSQL(UserDAO.UPGRADE_REQUEST);
        onCreate(db);
    }
}
```



# La persistance des données

## SQLite - Le Design Pattern DAO

- Le design pattern DAO est un modèle de conception offrant :
  - Une couche d'abstraction avec des **méthodes fortement typées** pour accéder aux données.
  - Des méthodes pour la **connexion** à la base de données.
- Le Data Access Object (DAO) va en fait jouer le rôle d'**intermédiaire** entre la classe qui va appeler les données (généralement, notre activité), et la base de données elle-même.
- En général, on aura donc
  - Une classe *model*, sous la forme d'un POJO.
  - Une table en base de données
  - Un DAO qui fera le lien entre la table et l'objet java



# La persistance des données

## SQLite - Les Data Access Objects

- Chaque DAO est propre à une table de votre base de données. On va donc y mettre tout ce qui ne se rapporte qu'à cette table (en public static final) :
  - Le nom de la table ainsi que les noms des différentes colonnes.
  - Le script de création, d'upgrade, ou de drop de cette table.
- De plus, on y ajoutera des variables private, un constructeur, et des méthodes d'accès à la base de données, en lecture et en écriture.

# La persistance des données SQLite - Les Data Access Objects

► Exemple (1) :

```
public class UserDao {

    public static final String TABLE_USER = "user";

    public static final String COLUMN_ID = "_id";
    public static final String COLUMN_LAST_NAME = "last_name";
    public static final String COLUMN_FIRST_NAME = "first_name";
    public static final String COLUMN_REG_NAT = "reg_nat";

    public static final String CREATE_REQUEST = "CREATE TABLE " + UserDao.TABLE_USER
        + " (" + UserDao.COLUMN_ID + " INTEGER PRIMARY KEY AUTOINCREMENT, "
        + UserDao.COLUMN_LAST_NAME + " TEXT NOT NULL, "
        + UserDao.COLUMN_FIRST_NAME + " TEXT NOT NULL, "
        + UserDao.COLUMN_REG_NAT + " TEXT UNIQUE NOT NULL"
        + ");";

    public static final String UPGRADE_REQUEST = "DROP TABLE " + UserDao.TABLE_USER;

    private DBHelper dbHelper;
    private Context context;
    private SQLiteDatabase db;
```

# La persistance des données

## SQLite - Les Data Access Objects

► Exemple (2) :

```
public UserDAO(Context context) {  
    this.context = context;  
}  
  
public UserDAO openWritable() {  
    dbHelper = new DBHelper(context);  
    db = dbHelper.getWritableDatabase();  
    return this;  
}  
  
public UserDAO openReadable() {  
    dbHelper = new DBHelper(context);  
    db = dbHelper.getReadableDatabase();  
    return this;  
}  
  
public void close() {  
    db.close();  
    dbHelper.close();  
}
```



# La persistance des données

## SQLite - Les Data Access Objects

- Enfin, on pourra ajouter les méthodes nécessaires à la manipulation des données (méthodes du C.R.U.D.) :
  - CREATE
  - READ
  - UPDATE
  - DELETE
- Et, éventuellement, d'autres méthodes dont on ressent le besoin.



# La persistance des données

## SQLite - Les Data Access Objects

### ► Create :

- Pour créer une entrée en base de données depuis un objet Java, on utilise l'objet ContentValues.
  - Celui-ci fonctionne comme un dictionnaire (couple clé/valeur).
  - Il prend en paramètre le nom de la colonne, et la valeur à attribuer à cette colonne.
- Cet objet ContentValues est simplement passé en paramètre à la méthode insert de notre db.

# La persistance des données

## SQLite - Les Data Access Objects

► Create (exemple) :

```
public long insert(User u) {  
  
    ContentValues cv = new ContentValues();  
    cv.put(COLUMN_LAST_NAME, u.getLastName());  
    cv.put(COLUMN_FIRST_NAME, u.getFirstName());  
    cv.put(COLUMN_REG_NAT, u.getRegNat());  
    return db.insert(TABLE_USER, null, cv);  
}
```





# La persistance des données

## SQLite - Les Data Access Objects

### ► Read :

- Lorsqu'on veut lire une ou plusieurs entrées en base de données SQLite, on doit utiliser la méthode `query`. Cette méthode nous rend un objet `Cursor`.
- L'objet `Cursor` est en fait un "extrait" de votre base de données, dans lequel on peut naviguer grâce à des méthodes.
  - `moveToFirst()`
  - `moveToNext()`
  - `moveToPosition()`
  - `getCount()`
  - ...
- **Attention !** Le `Cursor` ne commence pas à l'index 0 comme un tableau, mais se place avant le premier résultat. D'où la méthode `moveToFirst()` qu'il ne faut pas oublier d'appeler, mais seulement si un résultat existe !



# La persistance des données

## SQLite - Les Data Access Objects

### ► Read :

- Pour passer d'un *Cursor* à des valeurs en Java, on utilise la méthode *get* appropriée au type de données que l'on souhaite récupérer (*getInt()*, *getString()*, *getLong()*,...).
  - Ces méthodes prennent en paramètre le nom de la colonne dont on veut connaître la valeur.
  - Habituellement, on créera donc une méthode qui traduira notre objet *Cursor*, positionné à une ligne donnée, en objet Java.

# La persistance des données

## SQLite - Les Data Access Objects

► Read :

```
public Cursor getUserCursorById(int id) {  
    Cursor c = db.query(TABLE_USER, null, COLUMN_ID + "=" + id, null, null, null, null);  
  
    if (c.getCount() > 0) {  
        c.moveToFirst();  
        return c;  
    } else {  
        return null;  
    }  
}
```

# La persistance des données

## SQLite - Les Data Access Objects

► Read :

```
public static User cursorToUser(Cursor c) {  
    User u = new User();  
    u.setUserID(c.getInt(c.getColumnIndex(COLUMN_ID)));  
    u.setLastName(c.getString(c.getColumnIndex(COLUMN_LAST_NAME)));  
    u.setFirstName(c.getString(c.getColumnIndex(COLUMN_FIRST_NAME)));  
    u.setRegNat(c.getString(c.getColumnIndex(COLUMN_REG_NAT)));  
    return u;  
}
```

# La persistance des données

## SQLite - Les Data Access Objects

► Read :

```
public User getUserId(int id) {  
  
    Cursor c = getUserCursorById(id);  
  
    if (c != null) {  
        return cursorToUser(c);  
    } else {  
        return null;  
    }  
}
```

# La persistance des données

## SQLite - Les Data Access Objects

► Update :

```
public int update(User u) {  
  
    ContentValues cv = new ContentValues();  
    cv.put(COLUMN_LAST_NAME, u.getLastName());  
    cv.put(COLUMN_FIRST_NAME, u.getFirstName());  
    cv.put(COLUMN_REG_NAT, u.getRegNat());  
    return db.update(TABLE_USER, cv, COLUMN_ID + "=" + u.getUserID(), null);  
}
```

# La persistance des données

## SQLite - Les Data Access Objects

► Delete :

```
public void delete(User u) {  
    db.delete(TABLE_USER, COLUMN_ID + "=" + u.getUserID(), null);  
}
```



# Les BroadcastReceivers





# Les BroadcastReceiver

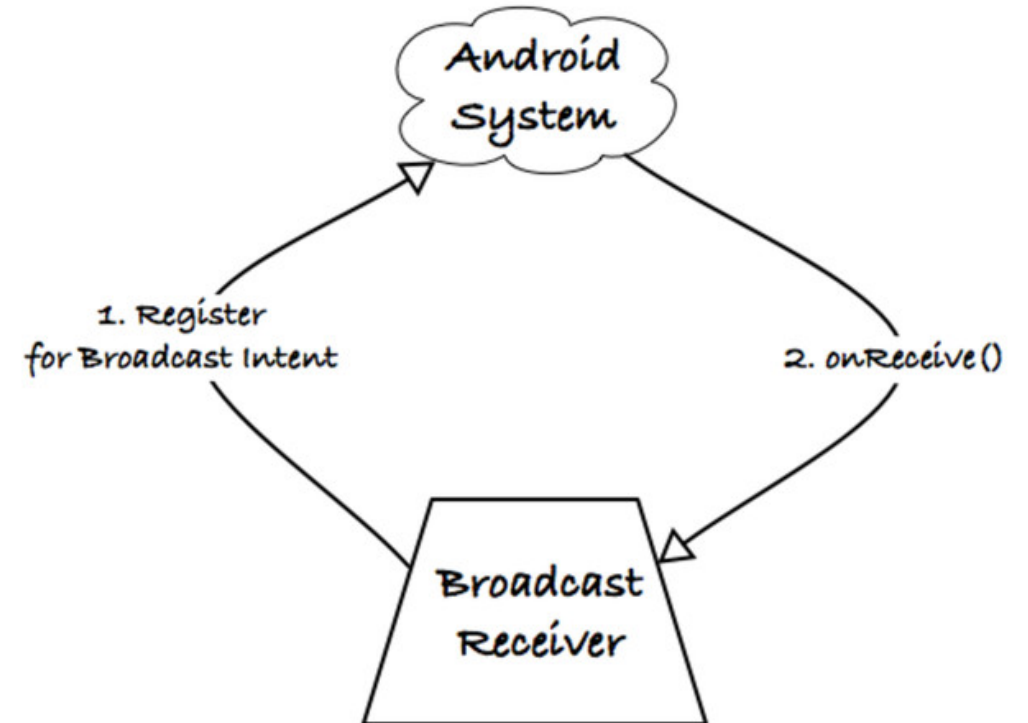
## Introduction

- Le BroadcastReceiver est l'un des composants logiciels d'Android. À ce titre, il doit donc être déclaré dans le manifeste.
- Son rôle est de s'enregistrer auprès du système, de façon à recevoir une notification lorsqu'un évènement surgit.
- Il s'agit simplement d'un code qui « dort » et qui se réveille lorsque l'action à laquelle il s'est enregistré survient.
- Contrairement aux activités et aux services, le BroadcastReceiver n'est pas un *Context*, mais en reçoit un via la méthode `onReceive`.

## Les BroadcastReceivers Introduction

- Lors de l'installation de votre application, le Broadcast Receiver s'enregistre auprès du système pour un certain type d'évènement.
- Lorsque cet évènement se produit, le système appelle la méthode `onReceive()` de notre Broadcast Receiver.

## Broadcast Receivers





# Les BroadcastReceivers

## Mise en place

- Créer une classe héritant de `BroadcastReceiver` et implémenter la méthode `onReceive()`.
  - On reçoit deux éléments :
    - Un contexte (qui nous permettra d'appeler des méthodes, ou de faire des traitements pour lesquels un contexte est nécessaire).
    - Un intent (la méthode `intent.getAction()` nous donne l'action à l'origine de l'appel du receiver).
- Déclarer cette classe dans le manifeste, en tant que `<receiver>`
- Spécifier, toujours dans le manifeste, les actions auxquelles ce receiver devra réagir.

# Les BroadcastReceivers

## Exemple

```
public class CustomBroadCast extends BroadcastReceiver {  
    @Override  
    public void onReceive(Context context, Intent intent) {  
  
        Log.i("CustomBroadCast", intent.getAction());  
    }  
}
```

```
<receiver android:name=".CustomBroadCast">  
    <intent-filter>  
        <action android:name="android.intent.action.AIRPLANE_MODE" />  
    </intent-filter>  
</receiver>
```



# Les BroadcastReceiver Exemple

- Autres types d'évènements systèmes :
  - `android.intent.action.`
    - `ACTION_BOOT_COMPLETED`
    - `ACTION_POWER_CONNECTED`
    - `ACTION_BATTERY_LOW`
    - `DOCK_EVENT`
    - `NEW_OUTGOING_CALL`
    - `SCREEN_ON`
    - ...

# Les BroadcastReceivers

## Mise en place

- Il est également possible de définir des actions personnalisées, et appeler ainsi votre BroadcastReceiver :

```
<receiver android:name=".CustomBroadCast">  
    <intent-filter>  
        <action android:name="be.formation.android.CUSTOM_ACTION" />  
    </intent-filter>  
</receiver>
```

```
Intent intent = new Intent();  
intent.setAction("be.formation.android.CUSTOM_ACTION");  
sendBroadcast(intent);
```



# Les BroadcastReceivers

## Exercice

1. Créer un Broadcast Receiver qui réagit au passage en mode avion et qui affiche un Toast à l'utilisateur.
2. Créer un Broadcast Receiver qui ouvre votre application à la réception de cet évènement. Attention ! Vous aurez besoin d'ajouter le flag `FLAG_ACTIVITY_NEW_TASK` à votre intent.



Annexe





# Android Studio :

## Logcat

- **Logcat** est un outil qui permet d'envoyer du **log** depuis application. Grâce à cela, vous allez pouvoir obtenir des **informations** au cours de l'exécution, et vérifier le bon fonctionnement de votre code.
- Pour logger un message depuis l'application, on utilise simplement une **méthode**, qui varie selon le **niveau** de votre log. Il en existe 6, du plus négligeable (verbose), au plus grave (assert).

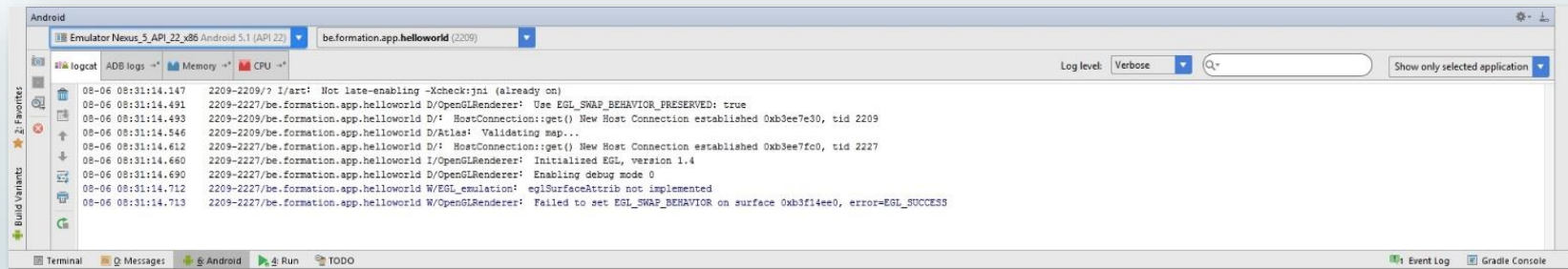


# Android Studio : Logcat

Level	Log. ... ("tag", "msg");	Level code
VERBOSE	Log.v	Log.VERBOSE
DEBUG	Log.d	Log.DEBUG
INFO	Log.i	Log.INFO
WARN	Log.w	Log.WARN
ERROR	Log.e	Log.ERROR
ASSERT	Log.wtf	Log.ASSERT

# Android Studio : Logcat

➡ Résultat :



➡ **Attention !** Tout appel aux méthodes de log doit se faire dans un bloc conditionnel :

```
String tag = "MainActivity";
String message = "Mon message de log";
int level = Log.DEBUG;

if (Log.isLoggable(tag, level)) {
    Log.d(tag, message);
}
```



# Android Studio : SDK Manager

- Le **SDK Manager** un outil **indispensable** : c'est grâce à lui que vous pouvez mettre à jour vos **API** ou en télécharger de nouvelles, mais aussi profiter d'**outils** plus spécifiques, comme les Google Play Services, l'Android Support Library,...
- Quelques conseils :
  - Installez toujours la dernière API (non *Preview*).
  - Téléchargez et mettez à jour les outils suivants :
    - Android Support Library
    - Android SDK BuildTools
    - Android SDK Tools
    - Android SDK Platform-Tools
    - Google USB Drivers
    - Intel x86 Emulator Accelerator
    - Google Play Services

# Android Studio : SDK Manager

The screenshot displays the 'Default Settings' dialog box in Android Studio, specifically the 'Appearance & Behavior > System Settings > Android SDK' section. The 'Android SDK' tab is selected, showing a list of SDK platforms and tools. The 'Android SDK Location' is set to 'D:\Android\Android SDK'. Below the list, there is a section for 'Available SDK developer tools' with a table of installed and available packages.

**Default Settings**

**Appearance & Behavior > System Settings > Android SDK**

Manager for the Android SDK and Tools used by Android Studio

Android SDK Location: D:\Android\Android SDK

**SDK Platforms** | **SDK Tools** | **SDK Update Sites**

Each Android SDK Platform package includes the Android platform default. Once installed, Android Studio will automatically check for updates to display individual SDK components.

Name	API
<input checked="" type="checkbox"/> Android 6.0 (Marshmallow)	23
<input checked="" type="checkbox"/> Android 5.1 (Lollipop)	22
<input type="checkbox"/> Android 5.0 (Lollipop)	21
<input type="checkbox"/> Android 4.4 (KitKat Wear)	20
<input type="checkbox"/> Android 4.4 (KitKat)	19
<input type="checkbox"/> Android 4.3 (Jelly Bean)	18
<input type="checkbox"/> Android 4.2 (Jelly Bean)	17
<input type="checkbox"/> Android 4.1 (Jelly Bean)	16
<input type="checkbox"/> Android 4.0.3 (IceCreamSandwich)	15
<input type="checkbox"/> Android 2.3.3 (Gingerbread)	10
<input type="checkbox"/> Android 2.2 (Froyo)	8

[Launch Standalone SDK Manager](#)

**Default Settings**

**Appearance & Behavior > System Settings > Android SDK**

Manager for the Android SDK and Tools used by Android Studio

Android SDK Location: D:\Android\Android SDK

**SDK Platforms** | **SDK Tools** | **SDK Update Sites**

Below are the available SDK developer tools. Once installed, Android Studio will automatically check for updates. Check "show package details" to display available versions of an SDK Tool.

Name	Version	Status
<input checked="" type="checkbox"/> Android SDK Build Tools		Installed
<input checked="" type="checkbox"/> Android SDK Tools 24.3.4	24.3.4	Installed
<input checked="" type="checkbox"/> Android SDK Platform-Tools 23	23.0.0	Installed
<input checked="" type="checkbox"/> Documentation for Android SDK	1	Installed
<input checked="" type="checkbox"/> Android Support Repository, rev 17	17.0.0	Installed
<input checked="" type="checkbox"/> Android Support Library, rev 23	23.0.0	Installed
<input type="checkbox"/> Android Auto Desktop Head Unit emulator	1.0.0	Not installed
<input checked="" type="checkbox"/> Google Play services, rev 26	26.0.0	Installed
<input checked="" type="checkbox"/> Google Repository, rev 21	21.0.0	Installed
<input checked="" type="checkbox"/> Google Play APK Expansion Library, rev 3	3.0.0	Installed
<input type="checkbox"/> Google Play Billing Library	5.0.0	Not installed
<input type="checkbox"/> Google Play Licensing Library	2.0.0	Not installed
<input checked="" type="checkbox"/> Android Auto API Simulators	1.0.0	Installed
<input checked="" type="checkbox"/> Google USB Driver, rev 11	11.0.0	Installed
<input checked="" type="checkbox"/> Google Web Driver, rev 2	2.0.0	Installed
<input checked="" type="checkbox"/> Intel x86 Emulator Accelerator (HAXM installer), rev 5.4	5.4.0	Installed
<input type="checkbox"/> Android NDK	1.0.0	Not installed

[Launch Standalone SDK Manager](#)

☐ Show Package Details

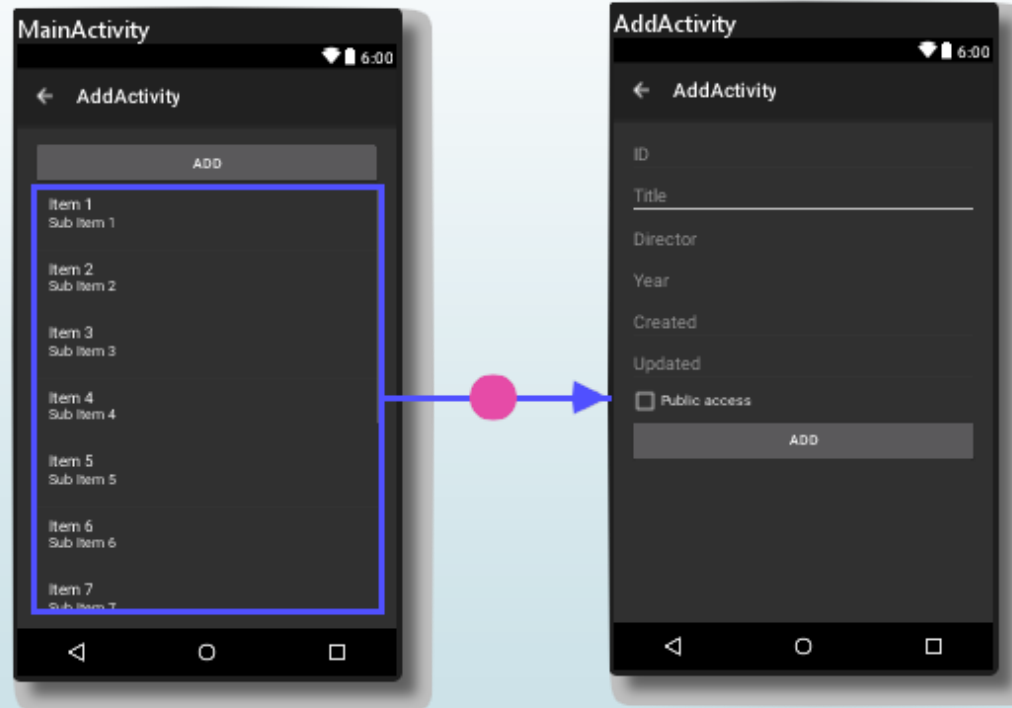
OK Cancel Apply Help



# Android Studio : Navigation Editor

- Le **Navigation Editor** est un outil qui permet d'**éditer** ainsi que d'avoir un **aperçu** visuel de la **navigation** dans votre application.
- On peut donc voir tous nos **écrans**, et savoir vers quelle activité redirige chaque **bouton**.
- Les **modifications** faites dans cet outil sont répercutées dans votre code.
- **Attention !**
  - Comme pour tout code généré, il faut impérativement aller voir ce que cet outil a écrit pour vous et le valider.
  - De plus, tous les types de navigation ne sont pas pris en compte (menus de navigations, fragments,...)

# Android Studio : Navigation Editor



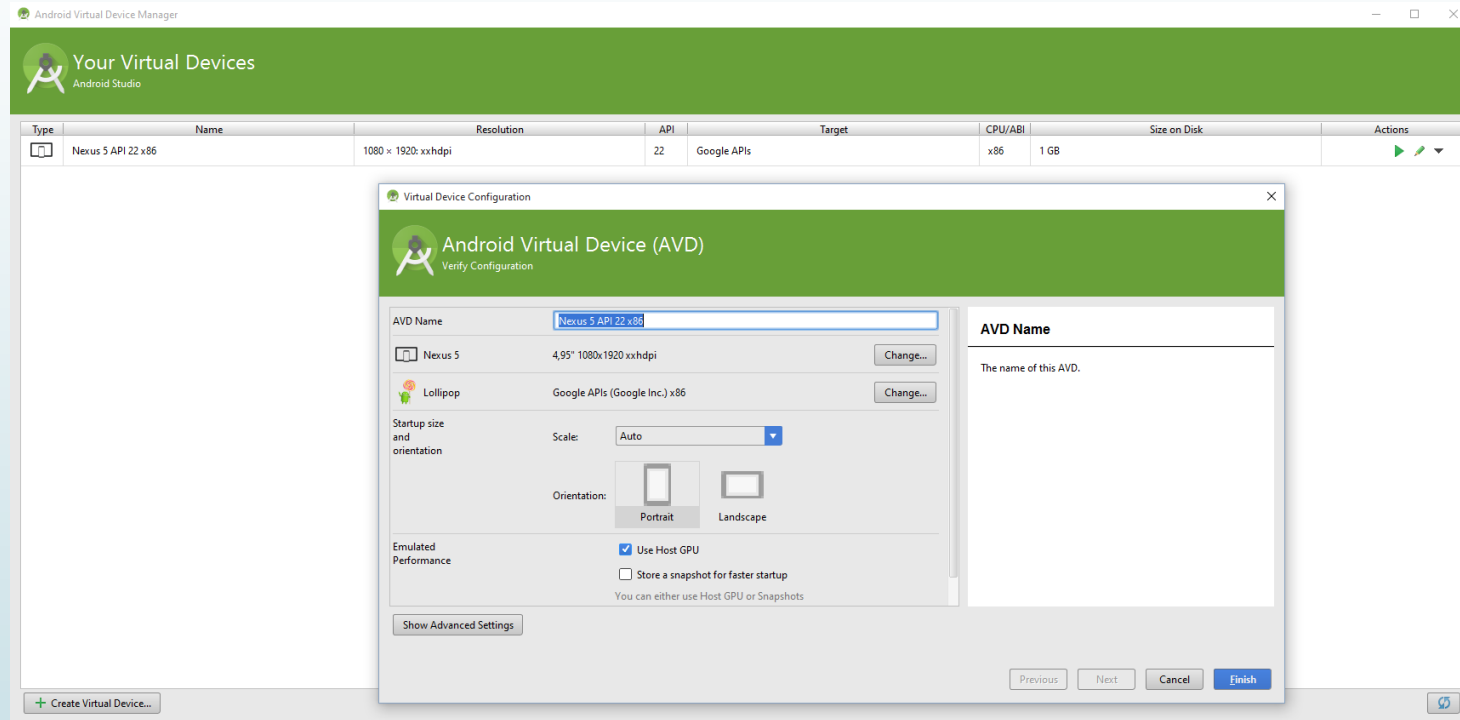


# Android Studio : AVD Manager

- L'**AVD** (Android Virtual Device) **Manager** permet de créer et de gérer vos **émulateurs** Android.
- Après l'installation Android Studio vous crée habituellement un émulateur à jour, mais vous pouvez, si vous le souhaitez, le modifier afin de simuler une **API** plus ancienne, ou une résolution différente.
- Options importantes :
  - Use Host GPU : permet d'accélérer l'exécution de l'émulateur.
  - Enable Keyboard Input (dans les options avancées) : permet de taper directement le texte au clavier. Attention de ne pas oublier de tester le bon fonctionnement du clavier intégré à l'émulateur !



# Android Studio : AVD Manager

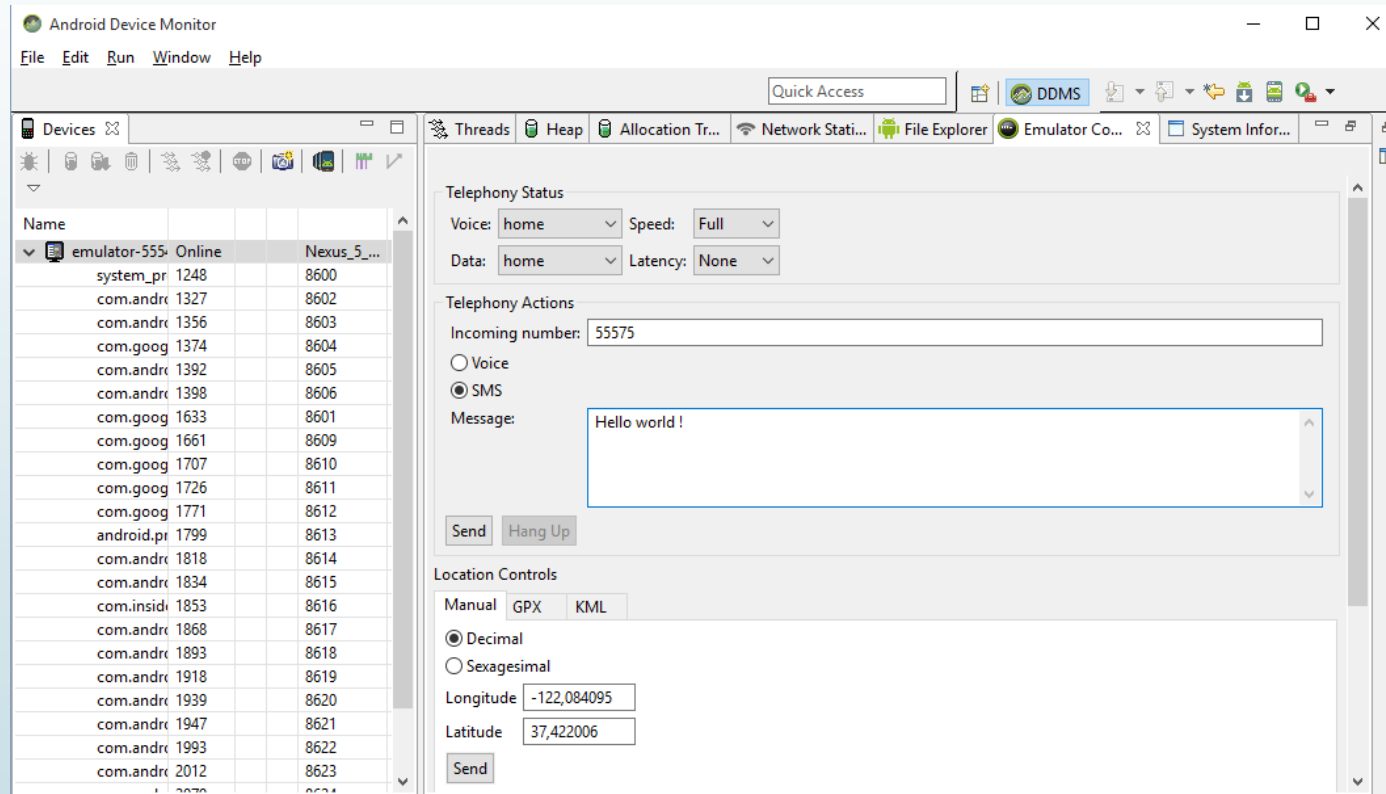




# Android Studio : Android Device Monitor

- **L'Android Device Monitor** est un outil puissant qui permet de simuler certains comportements sur votre **émulateur** :
  - Simulation d'**appels** entrants
  - Simulation de **SMS** entrants
  - Simulation de géolocalisation **GPS**
- Il permet également de naviguer parmi les **fichiers** de votre émulateur, d'en ajouter, ou d'en récupérer. C'est par exemple utile lorsqu'on travaille avec une base de données interne **SQLite**.

# Android Studio : Android Device Monitor





# Android Studio :

## Informations utiles

### ► Quelques raccourcis :

- Alt + Entrée      Action principale (importer, compléter, accepter,...)
- Ctrl + Espace    Auto-Complétion
- Alt + Insert      Générer
- Ctrl + Alt + O    Réorganiser les imports
- Ctrl + D          Dupliquer la ligne
- Ctrl + Alt + L    Reformater
- Ctrl + P          Voir les paramètres
- Maj + F10        Compiler et exécuter



# Contact

Raphaël Jungers

*[raphael.jungers@bstorm.be](mailto:raphael.jungers@bstorm.be)*

