INSTI-LOK 2021 - 2022

Développement d'applications mobiles

Ing. Judicael ZANNOU

Objectif

- **Objectif Général**
  - Ce cours vise à faire acquérir à l'apprenant, les technologies de développement mobile tel que Android et connaître les spécificités du développement mobile. Le but de ce cours est de découvrir la programmation sous une plateforme mobile, sa plate-forme de développement et les spécificités du développement embarqué sur téléphone mobile.
- **Objectifs Spécifiques** : A l'issue de ce cours, les étudiants auront appris à:
  - Comprendre les plateformes mobiles et leurs contraintes.
  - Apprendre à développer une application sur une plateforme mobile
  - Comprendre les fonctionnalités d'un Smartphone.

Plan

- Introduction sur les applications mobiles
- Introduction à la plateforme Android
- Quelques rappels du langage Java
- Installation et configuration des outils de développement Android
- Le cycle de vie d’une activité
- Éléments d'une Application Android
- IHM et composants graphiques sous la plateforme Android
- Les contrôles et la gestion des évènements
- Les processus en arrière plan
- Quelques fonctionnalités d’Android
- Publier une application Android

Introduction sur les applications mobiles

Généralité

- Logiciel applicatifs pour appareil électronique
- Une application mobile est un programme téléchargeable de façon gratuite ou payante et exécutable à partir du système d'exploitation d'un dispositif mobile (PDA, smartphone, tablette...).
- Elles sont pour la plupart distribuées depuis des plateformes de téléchargement (parfois elles mêmes contrôlées par les fabricants de smartphones) telles que l'App Store (plateforme d'Apple), le Google Play (plateforme de Google / Android), ou encore le Windows Phone Store(plateforme de Microsoft).

Introduction sur les applications mobiles

Généralité

- Peut fonctionner sans Internet
- Exige une connexion
- Interaction avec autre équipement électronique

Introduction sur les applications mobiles

Types d’application mobile

Techniquement parlant, il y a trois types d’application mobile que tout utilisateur peut rencontrer

- **Application Native** : Il s’agit d’application conçue pour un ou plusieurs systèmes d’exploitation mobiles bien particulier(s) basé sur un langage propre à chacun d’eux.
- **Application web** : Toute application conçue avec HTML et CSS de plus opérationnelle sur navigateur internet pour un smartphone est appelée application web.
- **Application hybride** : Il s’agit d’une application mobile qui fusionne entre les caractéristiques de web application (développement en HTML 5) et celles de l’application native. De cette manière, l’application mobile sera accessible sur toutes les plateformes d’application.

Application Native

Application hybride

Introduction à la plateforme Android

- À l'origine, « Android » était le nom d'une PME américaine, Android Incorporated, créée en 2003 puis rachetée par Google en 2005 ;
- En novembre de l'année 2007 l'Open Handset Alliance (OHA), et qui comptait à sa création 35 entreprises (actuelle 80 membres) évoluant dans l'univers du mobile, dont Google, fait la sortie officielle de Android 1.0 ;
- La popularité d'Android a toujours été croissante et au quatrième trimestre 2010 qu'Android devient le système d'exploitation mobile le plus utilisé au monde.

La philosophie et les avantages d'Android

- **Open source** : Le contrat de licence pour Android respecte les principes de l'open source ;
- **Gratuit (ou presque)** : Android est gratuit, autant pour vous que pour les constructeurs. Pour publier une application sur Play Store il faut créer un compte qui coûte $25 ;
- **Facile à développer** : Toutes les API mises à disposition facilitent et accélèrent grandement le travail ;
- **Facile à vendre** : Le Play Store est une plateforme immense et très visitée ;
- **Flexible** : Le système est extrêmement portable, il s'adapte à beaucoup de structures différentes ;
- **Complémentaire** : L'architecture d'Android est inspirée par les applications composites

Quelques rappels du langage Java

- **Les variables** : Les variables permettent de conserver dans la mémoire de l'appareil des informations avec lesquelles on va pouvoir faire des opérations ;
- **Les primitives** : Les primitives permettent de retenir des informations simples telles que des nombres sans virgule (auquel cas la variable est un entier, *int*), des chiffres à virgule (des réels, *float*) ou des booléens (variable qui ne peut valoir que vrai (*true*) ou faux (*false*), avec les *boolean*) ;
- **Les objets** : A l'opposé des primitives (variables simples), les objets sont des variables compliquées. Une primitive ne peut contenir qu'une information, tandis qu'un objet est constitué d'une ou plusieurs autres variables, et par conséquent d'une ou plusieurs valeurs. Ainsi, un objet peut lui-même contenir un objet !
    - Construire un objet s'appelle l'**instanciation**.

Quelques rappels du langage Java

- **L'héritage :** Une classe Java dérive toujours d'un autre classe, Object quand rien n'est spécifié. Pour spécifier de quelle classe hérite une classe on utilise le mot-clé extends ;
- **La compilation et l'exécution** : pour qu'un programme fonctionne, il doit d'abord passer par une étape de compilation, qui consiste à traduire votre code Java en bytecode. Dans le cas d'Android, ce bytecode sera ensuite lu par un logiciel qui s'appelle la machine virtuelle Dalvik. Cette machine virtuelle interprète les instructions bytecode et va les traduire en un autre langage que le processeur pourra comprendre, afin de pouvoir exécuter votre programme.

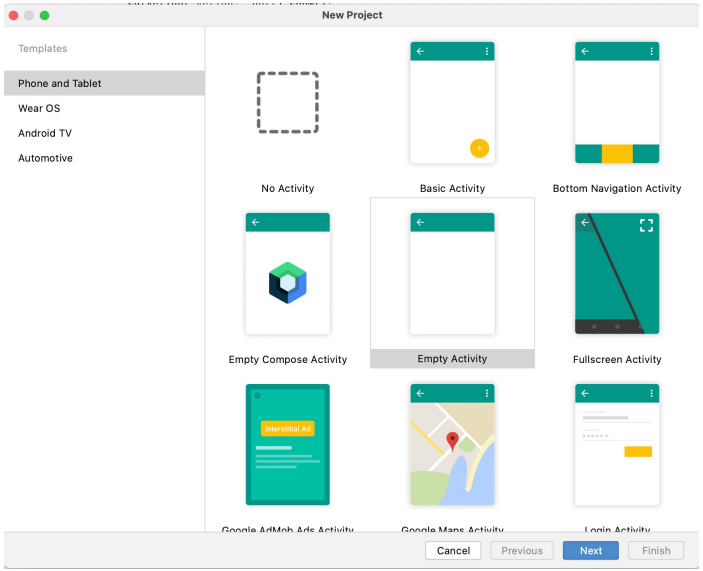
TP : Héritage en Java

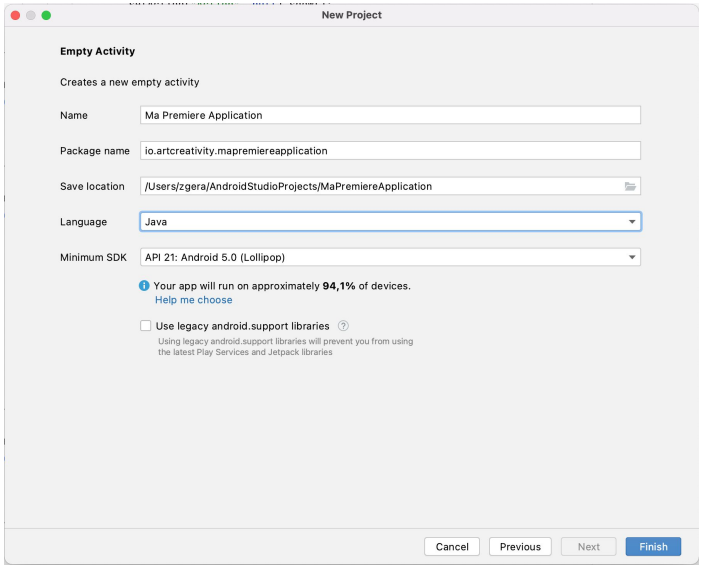
https://icwww.epfl.ch/~sam/ipo/series/serie07/ex1/task.html

Installation et configuration des outils de développement Android

- Pour télécharger Android Studio rendez vous sur https://developer.android.com/studio
- Suivez les instructions selon votre système d'exploitation pour installer et configurer Android Studio sur votre système sur https://developer.android.com/studio/install
- Installer et configurer un émulateur Android ou avoir un Smartphone Android

Votre première application





Éléments d'une Application Android

Une application Android peut être composée des éléments suivants:

- des activités (**android.app.Activity**): il s'agit d'une partie de l'application présentant une vue à l'utilisateur
- des services (**android.app.Service**): il s'agit d'une activité tâche de fond sans vue associée
- des fournisseurs de contenus (**android.content.ContentProvider)** permettent le partage d'informations au sein ou entre applications
- des widgets(**android.appwidget.***): une vue accrochée au bureau d'Android
- des Intents(**android.content.Intent**): permet d'envoyer un message pour un composant externe sans le nommer explicitement
- des récepteurs d'Intents(**android.content.BroadcastReceiver**): permet de déclarer être capable de répondre à des Intents
- des notifications (**android.app.Notifications**): permet de notifier l'utilisateur de la survenue d'événements

Éléments d'une Application Android

- **Activity**
  - Une activité (activity) gère l'affichage et les interactions utilisateurs sur un écran . Les activités sont indépendantes les unes des autres.
  - Une activité est une sous classe de **android.app.Activity**
  - Une application peut avoir plusieurs activités pouvant lancer cette application
- **Service**
  - Un service (service) est un composant qui est exécuté en tâche de fond. Il ne fournit pas d'interface graphique.
  - Exemple de service : jouer de la musique, rechercher des données sur le réseau.
  - Un service est une sous classe de **android.app.Service**

Éléments d'une Application Android

- **ContentProvider**
  - Un fournisseur de contenu (content provider) gère des données partageables. C'est le seul moyen d'accéder à des données partagées entre applications
  - Exemple de fournisseur de contenu : les informations de contacts de l'utilisateur du smartphone
  - On peut créer un fournisseur de contenus pour des données qu'on veut partager
  - On récupère un fournisseur de contenu pour des données partageables en demandant le **ContentResolver** du système et en donnant ensuite, dans les requêtes, l'URI des données

Éléments d'une Application Android

- **BroadcastReceiver**
  - Un récepteur d'informations est une sous classe de **android.content.BroadcastReceiver**
  - Un récepteur d'informations est un composant à l'écoute d'informations qui lui sont destinées. Un tel récepteur indique le type d'informations qui l'intéressent et pour lesquelles il se mettra en écoute. Exemple :
    - appel téléphonique entrant, réception d'un SMS, réseau Wi-Fi connecté, informations diffusées par des applications.
  - Les informations peuvent être envoyées par le système (réception de la liste des réseaux Wi-Fi, ...)
  - L'application réceptrice d'informations (c'est à dire possédant un récepteur d'informations) n'a pas besoin d'être lancée. Si elle ne l'est pas, Android la démarre automatiquement
  - Un récepteur n'est pas une IHM mais peut en lancer une (éventuellement petite dans la barre de notification), ou peut lancer un service traitant l'arrivée de l'information

Éléments d'une Application Android

- **Intent**
  - Un intent est une sous classe de **android.content.Intent**
  - Un événement (intent) est une "intention" à faire quelque chose contenant des informations destinées à un composant Android. C'est un message asynchrone
  - Les activités, services et récepteurs d'informations utilisent les Intent pour communiquer entre eux
  - Un Intent ne contient pas obligatoirement explicitement le composant qui va le gérer. Dans ce cas c'est un Intent implicite. Si l'Intent indique explicitement la classe du composant qui va le gérer c'est un Intent explicite

Éléments d'une Application Android

- **Le fichier Manifest**
  - Toute application doit contenir le fichier XML **AndroidManifest.xml**
  - Ce fichier déclare, entre autre, les différents composants de l'application
  - Parmi les composants, seuls les récepteurs d'évènements (BroadcastReceiver) ne sont pas forcément dans le manifeste.
  - Les autres (**Activity**, **Service**, **ContentProvider**) doivent l'être sinon ils ne seront jamais lancés quel que soit le code écrit !
  - Ce fichier contient aussi :
    - les permissions nécessaires à l'application (accès internet, accès en lecture/écriture aux données partagées)
    - le numéro minimum de la version API utilisée, des logiciels et matériels utilisés par l'application (caméra) et des bibliothèques supplémentaires à l'API de base (Google maps library)

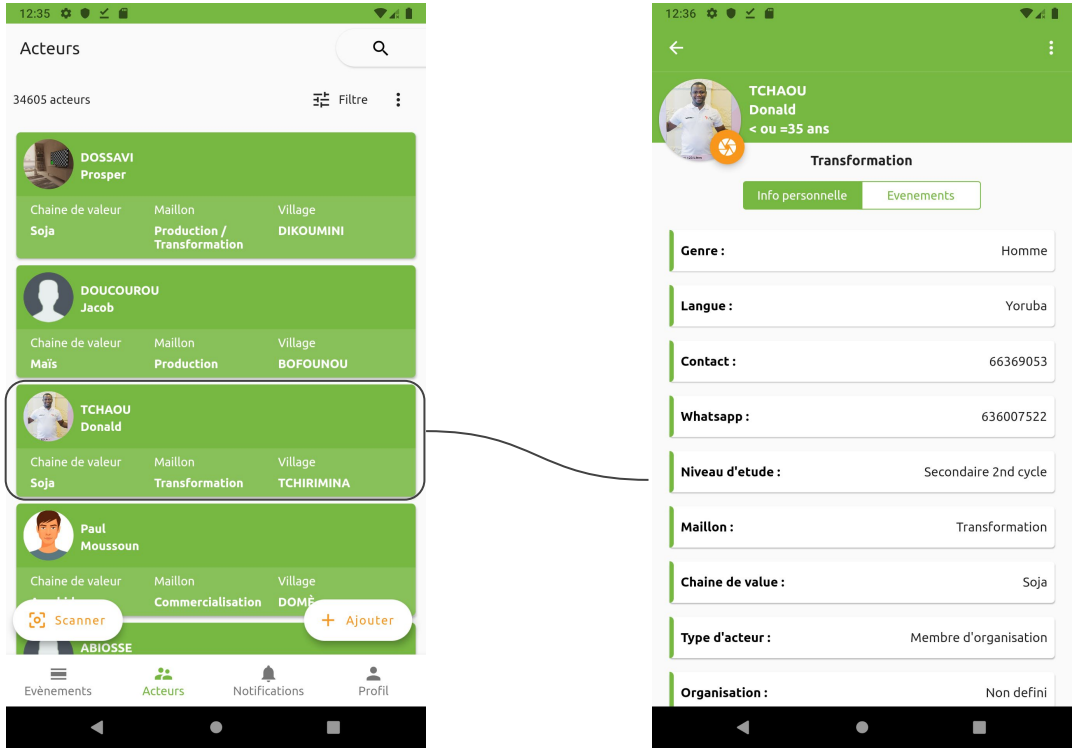
Le cycle de vie d’une activité

Qu’est ce que c’est qu’une activité?

Une **activité** est la composante principale d'une application sous Android. L'**activity** est le métier de l'application et possède généralement une **View** au minimum, c'est-à-dire un écran graphique.

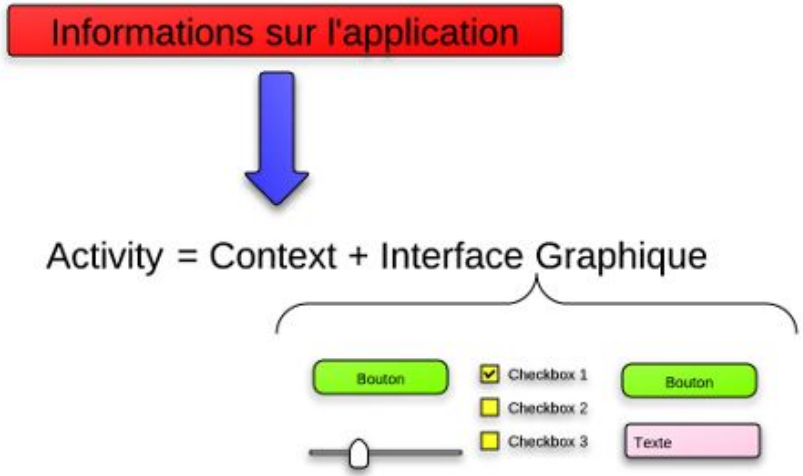
Ainsi dans une application standard, on pourrait trouver une activité qui liste des contacts, une activité qui ajoute un nouveau contact, et une activité qui affiche le détail d'un contact. Le tout forme un ensemble cohérent, mais chaque activité pourrait fonctionner de manière autonome.

Le cycle de vie d’une activité



Le cycle de vie d’une activité

Une activité contient des informations sur l'état actuel de l'application : ces informations s'appellent le ***context***. Ce ***context*** constitue un lien avec le système Android ainsi que les autres activités de l'application, comme le montre la figure suivante.



Le cycle de vie d’une activité

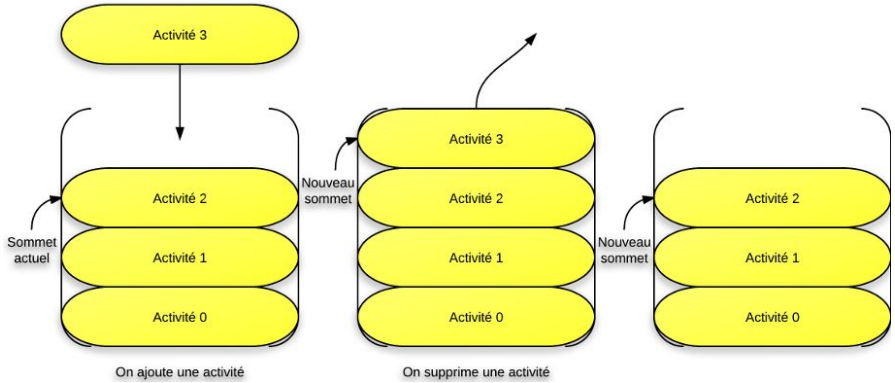
État d’une activité

- À tout moment votre application peut laisser place à une autre application, qui a une priorité plus élevée. Si votre application utilise trop de ressources système, alors elle empêchera le système de fonctionner correctement et Android l'arrêtera sans vergogne.
- Votre activité existera dans plusieurs états au cours de sa vie, par exemple un état actif pendant lequel l'utilisateur l'exploite, et un état de pause quand l'utilisateur reçoit un appel.

Le cycle de vie d’une activité

Pour être plus précis, quand une application se lance, elle se met tout en haut de ce qu'on appelle la pile d'activités.

Une pile est une structure de données de type «LIFO», c'est-à-dire qu'il n'est possible d'avoir accès qu'à un seul élément de la pile, le tout premier élément, aussi appelé sommet

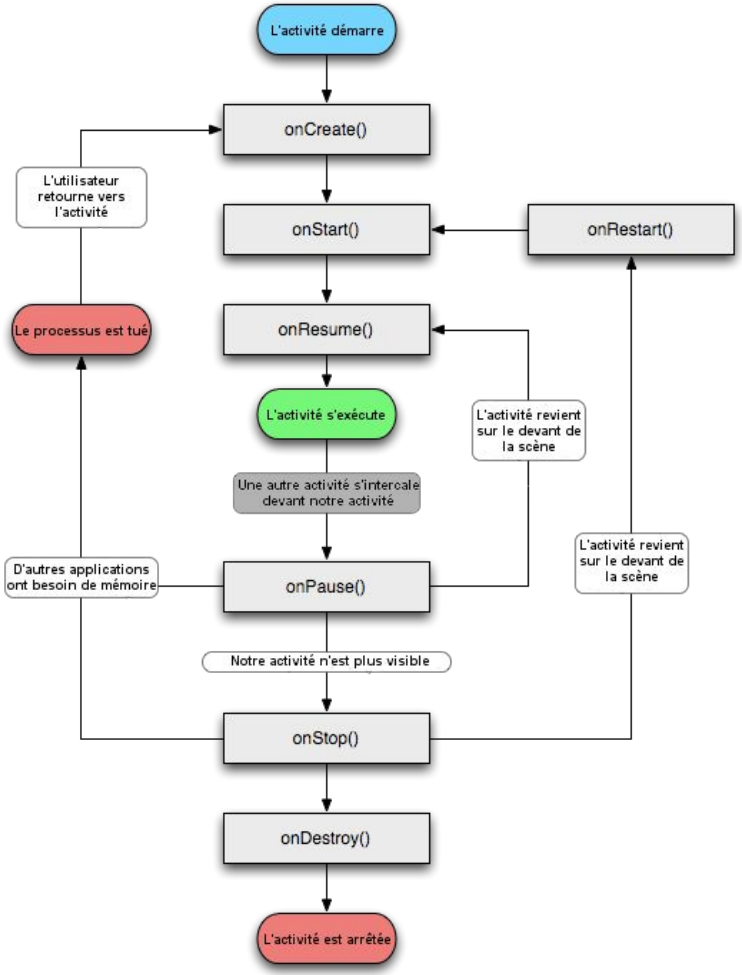


Le cycle de vie d'une activité

Etat	Visibilité	Description
Active («active» ou «running»)	L'activité est visible en totalité.	Elle est sur le dessus de la pile, c'est ce que l'utilisateur consulte en ce moment même et il peut l'utiliser dans son intégralité. C'est cette application qui a le focus, c'est-à-dire que l'utilisateur agit directement sur l'application.
Suspendue («paused»)	L'activité est partiellement visible à l'écran. C'est le cas quand vous recevez un SMS et qu'une fenêtre semi-transparente se pose devant votre activité pour afficher le contenu du message et vous permettre d'y répondre par exemple.	Ce n'est pas sur cette activité qu'agit l'utilisateur. L'application n'a plus le focus, c'est l'application au-dessus qui l'a. Pour que notre application récupère le focus, l'utilisateur devra se débarrasser de l'application partiellement au-dessus pour que l'utilisateur puisse à nouveau interagir avec notre activité. Si le système a besoin de mémoire, il peut très bien tuer l'application.
Arrêtée («stopped»)	L'activité est tout simplement invisible pour l'utilisateur, car une autre activité prend toute la place sur l'écran.	L'application n'a évidemment plus le focus, et puisque l'utilisateur ne peut pas la voir, il ne peut pas agir dessus. Le système retient son état pour pouvoir reprendre, mais il peut arriver que le système tue votre application pour libérer de la mémoire système.

Le cycle de vie d’une activité

Une activité n'a pas de contrôle direct sur son propre état (et par conséquent vous non plus en tant que programmeur), il s'agit plutôt d'un cycle rythmé par les interactions avec le système et d'autres applications.



Les ressources

Les ressources sont des fichiers qui contiennent des informations qui ne sont :

- Pas en Java (ce n'est donc pas du code).
- Pas dynamique (le contenu d'une ressource restera inchangé entre le début de l'exécution de votre application et la fin de l'exécution).

Android est destiné à être utilisé sur un très grand nombre de supports différents, et il faut par conséquent s'adapter à ces supports. L'avantage des ressources, c'est qu'elles nous permettent de nous adapter facilement à toutes ces situations différentes.

Les ressources

- **Le format XML**

  Le XML est un langage de balisage un peu comme le HTML — le HTML est d'ailleurs indirectement un dérivé du XML. Le langage de balisage ne donne pas des ordres é l’ordinateur comparer aux langages de développement (C++, Java, etc.), il se contente de définir une façon de mettre en forme l'information de façon à ce qu'on sache comment lire cette information.

Les ressources

- **La syntaxe XML**

  Comme pour le format HTML, un fichier XML débute par une déclaration qui permet d'indiquer qu'on se trouve bien dans un fichier XML.

  ```
  <?xml version="1.0" encoding="utf-8"?>
  ```

  Cette ligne permet d'indiquer que :

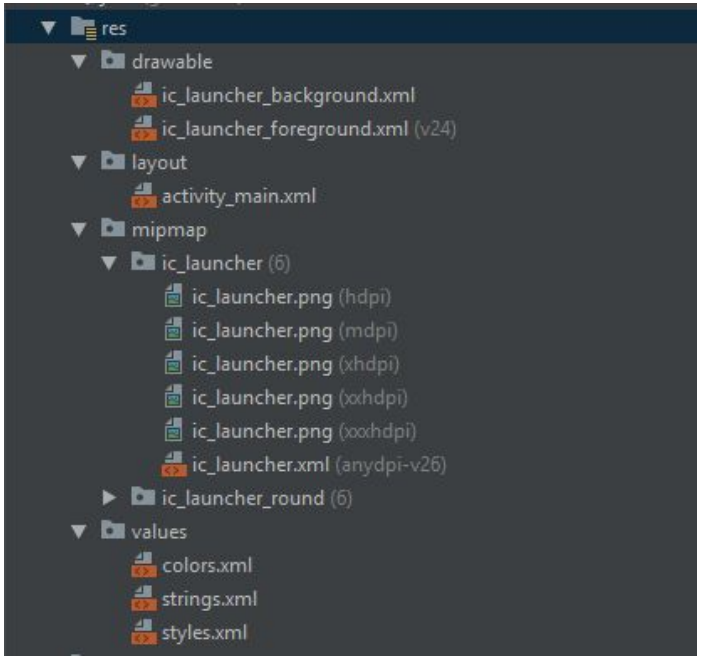
  - On utilise la version 1.0de XML.
  - On utilise l'encodage des caractères qui s'appelle utf-8; c'est une façon de décrire les caractères que contiendra notre fichier.

Les ressources

- La syntaxe XML

```xml
<?xml version="1.0" encoding="utf-8"?>
<universite>
  <ecole style="fantaisie">
    <nom>EPAC</nom>
    <adresse>Abomey-Calavi</adresse>
    <nb-etudiants>2530</nb-etudiants>
  </ecole>
  <ecole style="aventure">
    <nom>INSTI-LOK</nom>
    <adresse>Lokossa</adresse>
    <nb-etudiants>563</nb-etudiants>
  </ecole>
</universite>
```

Les ressources



Les ressources sont des éléments capitaux dans une application Android. On y trouve par exemple des chaînes de caractères ou des images. Comme Android est destiné à être utilisé sur une grande variété de supports, il fallait trouver une solution pour permettre à une application de s'afficher de la même manière sur un écran 7" que sur un écran 10", ou faire en sorte que les textes s'adaptent à la langue de l'utilisateur. C'est pourquoi les différents éléments qui doivent s'adapter de manière très précise sont organisés de manière tout aussi précise, de façon à ce qu'Android sache quels éléments utiliser pour quels types de terminaux.

Les ressources

Type	Description	Présence de fichiers XML
Dessin et image (*res/drawable*)	On y trouve les images matricielles (les images de type PNG, JPEG ou encore GIF) ainsi que des fichiers XML qui permettent de décrire des dessins (ce qui donne des images vectorielles qui ne se dégradent pas quand on les agrandit).	Oui
Mise en page ou interface graphique (*res/layout*)	Les fichiers XML qui représentent la disposition des vues.	Exclusivement
Menu (res/menu)	Les fichiers XML pour pouvoir constituer des menus.	Exclusivement
Donnée brute (*res/raw*)	Données diverses au format brut. Ces données ne sont pas des fichiers de ressources standards, on pourrait y mettre de la musique ou des fichiers HTML par exemple.	Le moins possible
Différentes variables (*res/values*)	Il est plus difficile de cibler les ressources qui appartiennent à cette catégorie tant elles sont nombreuses. On y trouve entre autre des variables standards, comme des chaînes de caractères, des dimensions, des couleurs, etc.	Exclusivement

Les ressources

```
public final class R {
    public static final class attr {
    }
    public static final class dimen {
      public static final int padding_large=0x7f040002;
      public static final int padding_medium=0x7f040001;
      public static final int padding_small=0x7f040000;
    }
    public static final class drawable {
      public static final int ic_action_search=0x7f020000;
      public static final int ic_launcher=0x7f020001;
    }
    public static final class id {
      public static final int menu_settings=0x7f080000;
    }
    public static final class layout {
      public static final int activity_main=0x7f030000;
    }
    public static final class menu {
      public static final int activity_main=0x7f070000;
    }
    public static final class string {
      public static final int app_name=0x7f050000;
      public static final int hello_world=0x7f050001;
      public static final int menu_settings=0x7f050002;
      public static final int title_activity_main=0x7f050003;
    }
    public static final class style {
      public static final int AppTheme=0x7f060000;
    }
  }
```

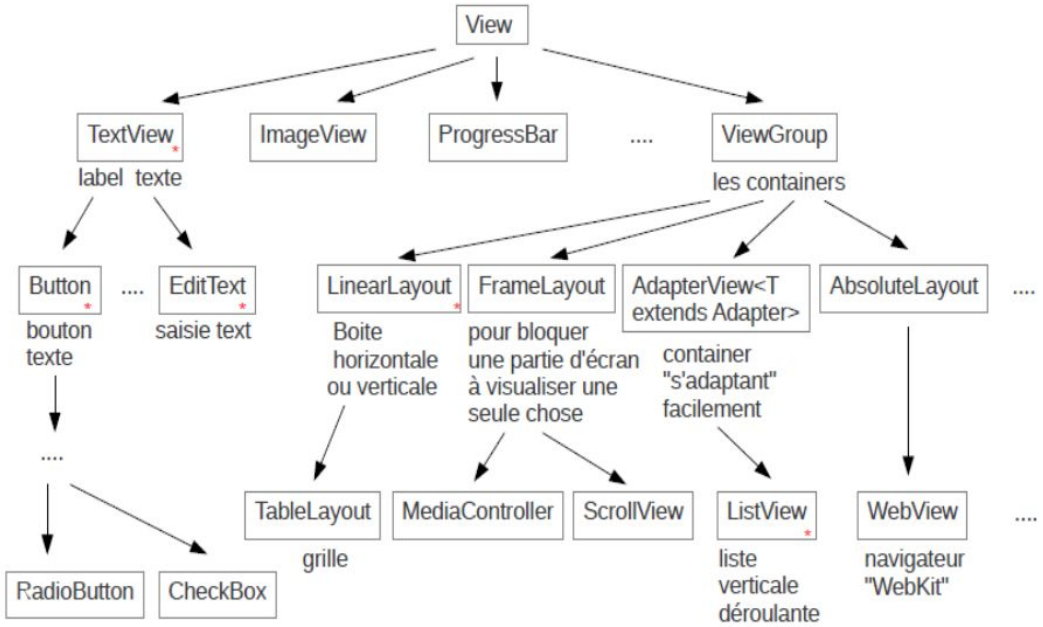
IHM sous la plateforme Android

- **Construction d'une IHM**
  - Construire une IHM, c'est mettre des composants graphiques les uns à l'intérieur des autres
  - Les ensembles de composants graphiques sont des classes. On aura la classe des boutons, la classe des cases à cocher, etc.
  - Un composant graphique particulier sera une instance particulière d'une classe. Par exemple le bouton "Quitter" et le bouton ''Sauvegarder" d'une IHM seront deux instances de la classe des boutons
  - Un composant graphique a 3 parties :
    - les données qu'il représente : le modèle (model)
    - le dessin d'affichage : la vue (view)
    - ce qui prend en charge les actions de l'utilisateur sur ce composant : le controleur

IHM sous la plateforme Android

- **Construction d'une IHM**
  - Les interfaces sont composées d'objets héritant des classes View et ViewGroup;
  - Une vue se dessine et permet les interactions avec l'utilisateur (button, textView...)
  - Un groupe de vues contient d'autres vues (les gabarits conteneurs des vues)
    - Un groupe de vues organise l'affichage des vues qu'il contient
    - Un groupe de vues est une vue.
  - Android fournit une collection de vues et de groupes de vues qui offrent :
    - les principales entrées (textes, champs de texte, listes, etc.) ;
    - différents modèles d'organisation (linéaire, etc.).
  - Une classe est associée à chaque type de vue ou groupe de vue.

IHM sous la plateforme Android



IHM sous la plateforme Android

- **Construction d'une IHM**
  - On peut tout coder en Java dans l'activité
  - Mais, en général, pour l'IHM on utilise plutôt les fichiers XML
  - Par exemple, créer une nouvelle application Android
    - Ouvrir le fichier ***activity_main.xml*** (dans *res/layout/activity_main.xml*)
  - Et on peut construire l'IHM en glisser déposer à partir d'une palette graphique.
  - L'IHM est alors générée en XML
  - On peut changer la largeur et la hauteur d'un composant à l'aide de ses propriétés
    - Exemple: ***layout:width*** et ***layout:height***
  - Les valeurs de ces propriétés peuvent être
    - ***match_parent***(anciennement nommé ***fill_parent*** avant l'API 8) indique que le composant graphique sera aussi grand en largeur ou hauteur que son conteneur le lui autorise
    - ***wrap_content*** indiquant que le composant prend seulement la taille qui lui faut en largeur ou hauteur pour s'afficher correctement

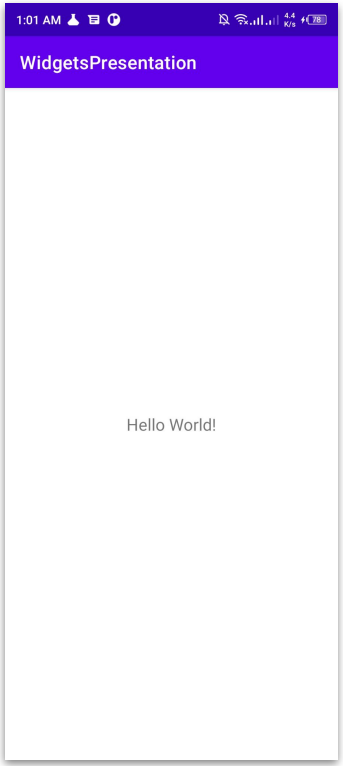
IHM sous la plateforme Android

- **Construction d'une IHM**
  - Pour positionner une propriété (= valeur d'attribut = données = ...) d'un composant graphique on peut le faire soit en XML soit par appel d'une méthode appropriée en java
  - Dans le fichier XML, on positionne une propriété d'un composant graphique à l'aide d'une valeur d'attribut de sa balise XML
  - Donc il y a une correspondance entre les noms d'attribut d'une balise XML et une méthode, pour positionner une propriété d'un composant graphique. On a, par exemple :
    - android:background pour l'arrière plan
    - android:alpha pour la transparence d'une vue (0 pour complètement transparent et 1 pour complètement opaque)

Quelques Widgets sous la plateforme Android

- TextView

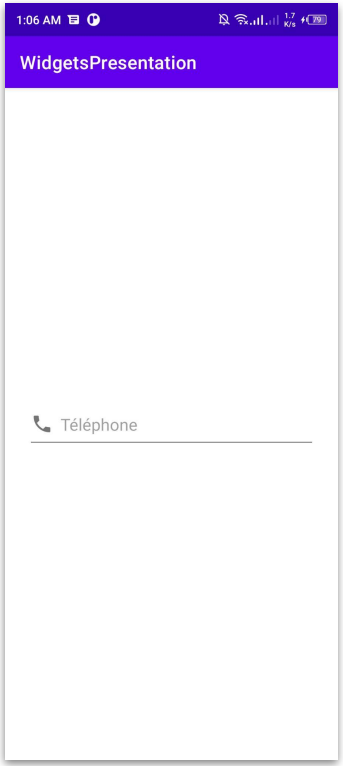
```
<TextView
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text="Hello World!"
      android:textAppearance="@style/TextAppearance.AppComp
      at.Medium" />
```



Quelques Widgets sous la plateforme Android

- EditText

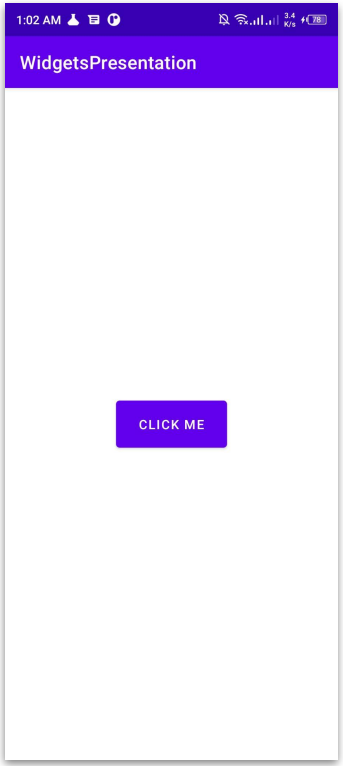
```
<EditText
      android:layout_width="match_parent"
      android:layout_height="wrap_content"
      android:hint="Téléphone"
      android:inputType="phone"
      android:drawableStart="@drawable/ic_phone"
      android:drawablePadding="8dp"
      android:layout_margin="24dp"
      android:paddingVertical="16dp"
      android:textAppearance="@style/TextAppearance.AppComp
      at.Medium" />
```



Quelques Widgets sous la plateforme Android

- Button

```
<Button
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:layout_margin="24dp"
      android:paddingHorizontal="24dp"
      android:paddingVertical="16dp"
      android:text="Click me" />
```



Quelques Widgets sous la plateforme Android

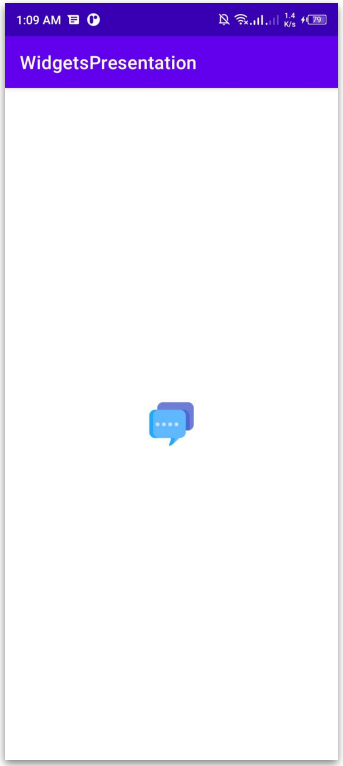
- ImageView

```
<ImageView
      android:layout_width="match_parent"
      android:layout_height="wrap_content"
      android:layout_margin="48dp"
      android:scaleType="center"
      android:src="@drawable/ananas_neutre" />
```

Quelques Widgets sous la plateforme Android

- ImageButton

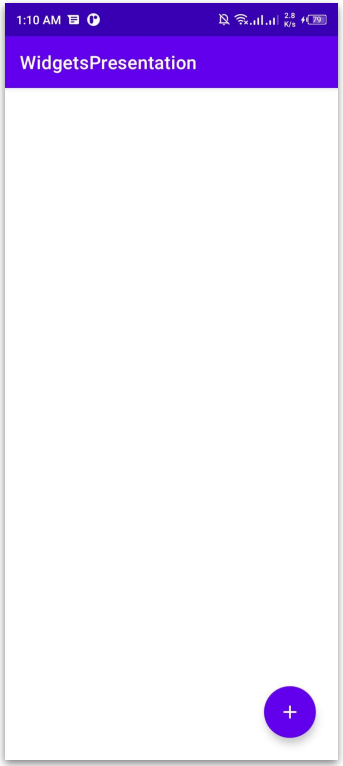
```
<ImageButton
      android:layout_width="48dp"
      android:layout_height="48dp"
      android:layout_margin="48dp"
      android:scaleType="centerInside"
      android:background="@android:color/transparent"
      android:src="@drawable/comments" />
```



Quelques Widgets sous la plateforme Android

- FloatingActionButton

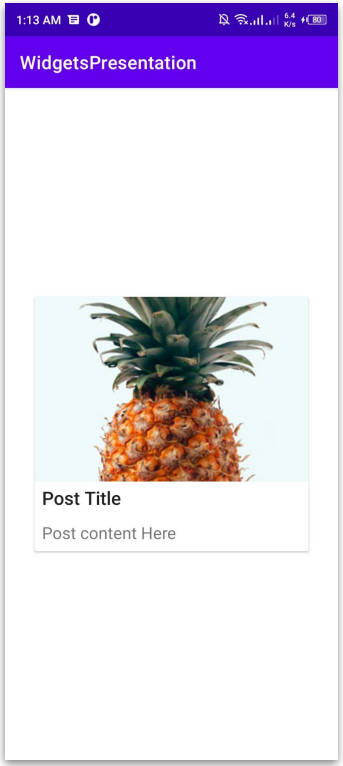
```
<com.google.android.material.floatingactionbutton.Floating
ActionButton
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:layout_margin="24dp"
      android:src="@drawable/ic_add_white"
      app:backgroundTint="@color/purple_500"
      app:layout_constraintBottom_toBottomOf="parent"
      app:layout_constraintRight_toRightOf="parent"
      app:tint="@color/white" />
```



Quelques Widgets sous la plateforme Android

- CardView

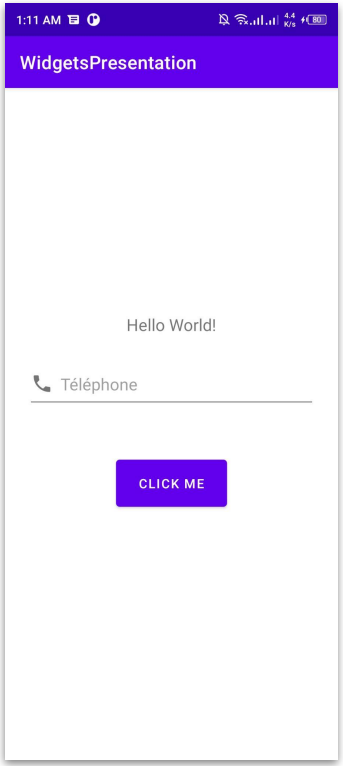
```
<androidx.cardview.widget.CardView
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:layout_margin="32dp"
      android:padding="8dp">
      <LinearLayout
            android:layout_width="match_parent"
            android:layout_height="match_parent"
            android:orientation="vertical">

            …

      </LinearLayout>
</androidx.cardview.widget.CardView >
```



Quelques Widgets sous la plateforme Android

- LinearLayout

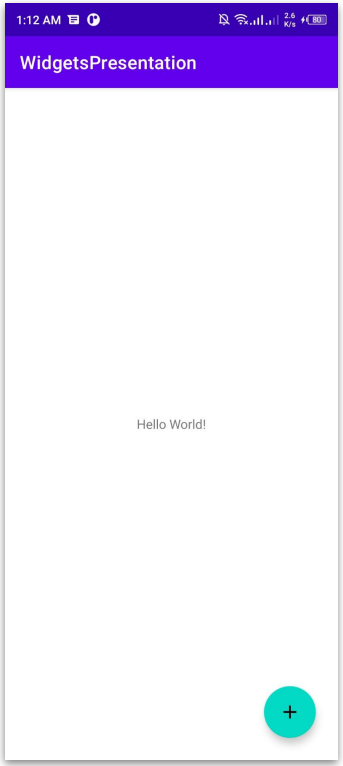
```
<LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android "
xmlns:tools="http://schemas.android.com/tools"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:gravity="center"
        android:orientation="vertical"
        tools:context=".MainActivity"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

           …

</LinearLayout >
```



Quelques Widgets sous la plateforme Android

- RelativeLayout

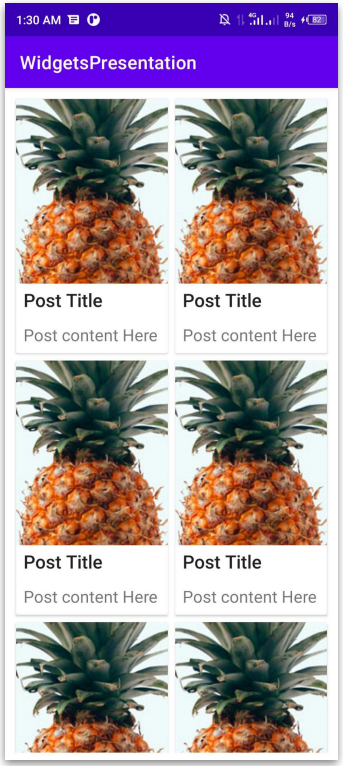
```
<RelativeLayout
xmlns:android="http://schemas.android.com/apk/res/android"
          xmlns:tools="http://schemas.android.com/tools"
          android:layout_width="match_parent"
          android:layout_height="match_parent">
       <TextView
          android:layout_width ="wrap_content"
          android:layout_height ="wrap_content"
          android:layout_centerHorizontal ="true"
          android:layout_centerVertical ="true"
          android:text="Hello World!"  />
       <com.google.android.material.floatingactionbutton.FloatingActionButton
          android:layout_width ="wrap_content"
          android:layout_height ="wrap_content"
          android:layout_alignParentEnd ="true"
          android:layout_alignParentBottom ="true"
          android:layout_margin ="24dp"
          android:src="@drawable/ic_add_white"  />
</LinearLayout>
```



Quelques Widgets sous la plateforme Android

- RecyclerView

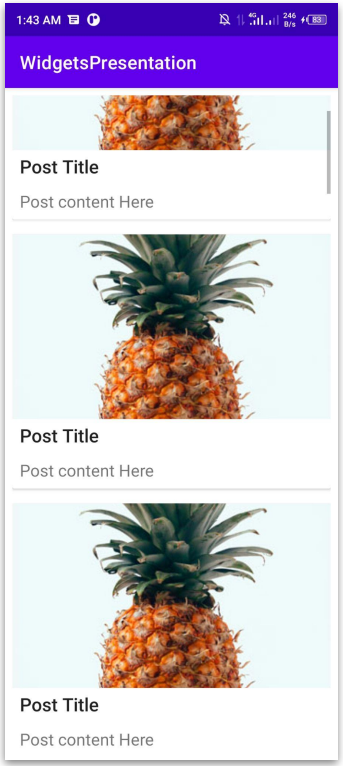
```
<androidx.recyclerview.widget.RecyclerView
      android:layout_width="match_parent"
      android:layout_height="match_parent"
      android:padding="8dp"
      app:layoutManager="androidx.recyclerview.widget.GridL
ayoutManager"
      app:spanCount="2"
      tools:itemCount="8"
      tools:listitem="@layout/card_item"  />
```



Quelques Widgets sous la plateforme Android

- ListView

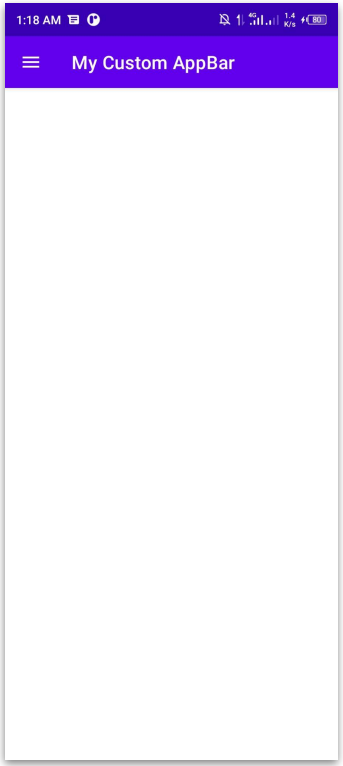
```
<ListView
      android:layout_width="match_parent"
      android:layout_height="match_parent"
      android:padding="8dp"
      tools:listitem="@layout/card_item"  />
```



Quelques Widgets sous la plateforme Android

- AppBar

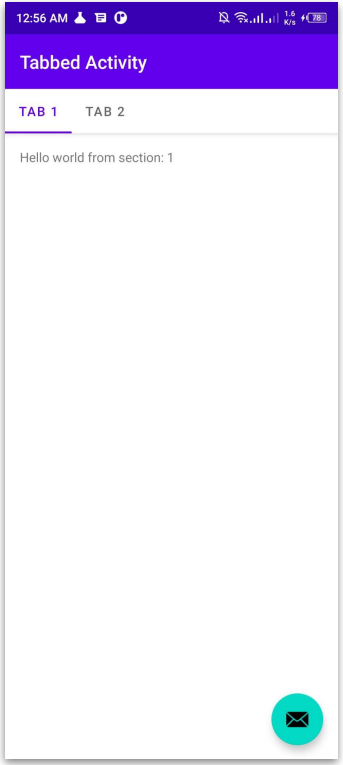
```
<com.google.android.material.appbar.AppBarLayout
      android:layout_width="match_parent"
      android:layout_height="wrap_content">

      <com.google.android.material.appbar.MaterialToolbar
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            app:navigationIcon="@drawable/ic_menu_white"
            app:title="My Custom AppBar"
            app:titleTextColor="@color/white" />
</com.google.android.material.appbar.AppBarLayout >
```



Quelques Widgets sous la plateforme Android

- TabLayout

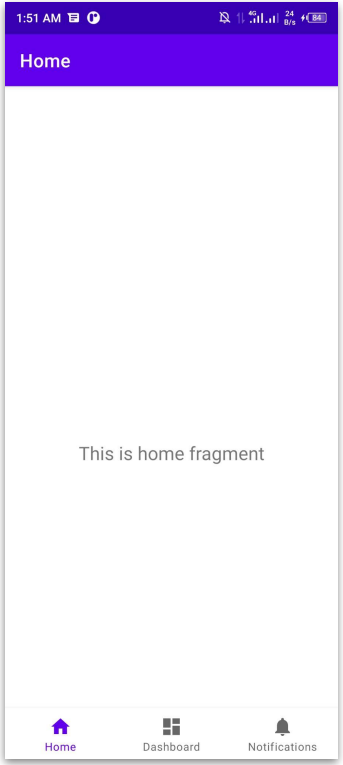
```
<androidx.coordinatorlayout.widget.CoordinatorLayout
   android:layout_width="match_parent"
   android:layout_height="match_parent" >
       <com.google.android.material.appbar.AppBarLayout
              android:layout_width="match_parent"
              android:layout_height="wrap_content"
              android:theme="@style/Theme.WidgetsPresentation.AppBarOverlay"  >
              <TextView
                     android:id="@+id/title"
                     android:layout_width="wrap_content"
                     android:layout_height="wrap_content"
                     android:gravity="center"
                     android:minHeight="?actionBarSize"
                     android:text="Tabbed Activity"/>
              <com.google.android.material.tabs.TabLayout
                     android:id="@+id/tabs"
                     android:layout_width="match_parent"
                     android:layout_height="wrap_content"
                     app:tabMode="scrollable"  />
       </com.google.android.material.appbar.AppBarLayout  >
       <androidx.viewpager.widget.ViewPager
              android:id="@+id/view_pager"
              android:layout_width="match_parent"
              android:layout_height="match_parent"
              app:layout_behavior="@string/appbar_scrolling_view_behavior"   />
       <!-- fab here -->
 </androidx.coordinatorlayout.widget.CoordinatorLayout  >
```



Quelques Widgets sous la plateforme Android

- BottomNavigation

```
<com.google.android.material.bottomnavigation.BottomNavigationView
    android:id="@+id/nav_view"
    android:layout_width="0dp"
    android:layout_height="wrap_content"
    android:layout_marginStart="0dp"
    android:layout_marginEnd="0dp"
    android:background="?android:attr/windowBackground"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintLeft_toLeftOf="parent"
    app:layout_constraintRight_toRightOf="parent"
    app:menu="@menu/bottom_nav_menu"  />
```



Les contrôles et la gestion des évènements

Deux moyens de gérer les évènements:

- créer un auditeur d'événements (classe qui implémente une interface connue) et l'enregistrer auprès du composant (View)
- les View sont elles mêmes auditrices de certains événements: (touché de l'écran par exemple). Il suffit donc de spécialiser la méthode adaptée et lancée lorsque l'évènement survient
    - 1°) est classique (Java SE, Java ME). Les interfaces sont des interfaces internes à la classe View et de nom OnXXXListener(donc des interfaces de nom View.OnXXXListener). Cela nécessite d'implémenter une méthode de nom onXXX(). On enregistre un auditeur par setOnXXXListener(View.OnXXXListenerl)
    - 2°) permet d'écrire directement la gestion de certains événements qui peuvent se produire dans la View

Les contrôles et la gestion des évènements

Il existe plusieurs interfaces, une pour chaque type d'actions:

- L'interface utilisée pour contrôler les simples clics : **View.OnClickListener**, il nous oblige à définir la méthode **void onClick(View vue)**.
- **View.OnLongClickListener** pour les longs clics, avec la méthode:
  - **boolean onLongClick(View vue) :** elle doit retourner **true** une fois que l'action associée a été effectuée.
- **View.OnKeyListener** pour gérer l'appui de touches avec la méthode:
  - **boolean onKey(View vue, int codeTouche, KeyEvent inforEvent) :** Cette méthode doit retourner **true** une fois que l'action associée a été effectuée.

Les contrôles et la gestion des évènements

Il existe plusieurs interfaces, une pour chaque type d'actions:

- **View.OnTouchListener** correspond à une simple touche qui s'effectue sur l'élément. On observe une touche au moment d'un clique puis au moment du relâchement. Rester appuyé après un clic est toujours une touche, même si on sort de l'élément. On y associe la méthode:
    - **boolean onTouch(View vue, MotionEvent inforEvent)** : Cette méthode doit retourner true une fois que l'action associée a été effectuée.

Les contrôles et la gestion des évènements

Intent

- Un intent est une "intention" à faire quelque chose contenant des informations destinées à un composant Android. C'est un message asynchrone.
- Les activités, services et récepteurs d'informations utilisent les Intent pour communiquer entre eux.
- La navigation entre les écrans peut se faire de deux façons différentes : soit explicitement, soit implicitement. Dans les deux cas, l'ordre de changement d'activité est véhiculé par un objet de type *Intent*.
- Un **Intent** ne contient pas obligatoirement explicitement le composant qui va le gérer. Dans ce cas c'est un Intent implicite. Si l'**Intent** indique explicitement la classe du composant qui va le gérer c'est un Intent explicite
- Un événement est une sous classe de **android.content.Intent**

Les contrôles et la gestion des évènements

Intent

- Les Intents permettent de gérer l'envoi et la réception de messages afin de faire coopérer les applications. Le but des Intents est de déléguer une action à un autre composant, une autre application ou une autre activité de l'application courante.
- Un objet Intent contient les information (champs) suivantes:
  - le nom du composant ciblé (facultatif)
  - l'action à réaliser, sous forme de chaîne de caractères
  - les données: contenu MIME et URI des données supplémentaires (extra) sous forme de paires clef/valeur
  - une catégorie pour cibler un type d'application
  - des drapeaux (informations supplémentaires)

Les contrôles et la gestion des évènements

Intent

- La façon dont sont renseignés ces champs détermine la nature ainsi que les objectifs de l'intent. Ainsi, pour qu'un intent soit dit « explicite », il suffit que son champ composant soit renseigné. Ce champ permet de définir le destinataire de l'intent, celui qui devra le gérer. Ce champ est constitué de deux informations : le package où se situe le composant, ainsi que le nom du composant. Ainsi, quand l'intent sera exécuté, Android pourra retrouver le composant de destination de manière précise.

Les contrôles et la gestion des évènements

Intent

- À l'opposé des intents explicites se trouvent les intents« implicites ». Dans ce cas, on ne connaît pas de manière précise le destinataire de l'intent, c'est pourquoi on va s'appliquer à renseigner d'autres champs pour laisser Android déterminer qui est capable de réceptionner cet intent. Il faut au moins fournir deux informations essentielles :
    - Une action : ce qu'on désire que le destinataire fasse.
    - Un ensemble de données : sur quelles données le destinataire doit effectuer son action.

Les contrôles et la gestion des évènements

Intent

- Il existe aussi d'autres informations, pas forcément obligatoires, mais qui ont aussi leur utilité propre le moment venu :
  - La catégorie : permet d'apporter des informations supplémentaires sur l'action à exécuter et le type de composant qui devra gérer l'intent.
  - Le type : pour indiquer quel est le type des données incluses. Normalement ce type est contenu dans les données, mais en précisant cet attribut vous pouvez désactiver cette vérification automatique et imposer un type particulier.
  - Les extras : pour ajouter du contenu à vos intents afin de les faire circuler entre les composants.
  - Les flags : permettent de modifier le comportement de l'intent.

Les contrôles et la gestion des évènements

Intent

- L'action d'un intent est une chaîne de caractères qui symbolise le traitement à déclencher.
- De nombreuses constantes sont définies dans le SDK pour les actions nativement disponibles comme:
  - **Intent.ACTION_WEB_SEARCH** pour demander de réaliser une recherche sur Internet
  - **Intent.ACTION_CALL** pour passer un appel téléphonique.
- Les données détaillent l'action de l'Intent dont elle dépend directement. Elles peuvent être d'ailleurs nulle, certaines actions n'appelant pas forcément à devoir être précisées. Pour illustration, dans l'exemple **ACTION_CALL**, la donnée sera le numéro de téléphone à composer.

Les contrôles et la gestion des évènements

Intent

- La donnée est couplée avec un autre attribut qui est le type **MIME** de la donnée à traiter.
- La catégorie, quant à elle, apporte une classification à l'action. La catégorie **Intent.CATEGORY_LAUNCHER** positionne l'activité comme étant exécutable, cette catégorie est utilisée de pair avec l'action **Intent.ACTION_MAIN**. Android positionne les activités de cette catégorie dans le lanceur d'applications. La catégorie **CATEGORY_HOME** marque l'activité à afficher au démarrage du téléphone.

Les Menus

- Un menu est un endroit privilégié pour placer certaines fonctions tout en économisant l’espace dans une fenêtre.
- Tous les anciens terminaux sous Android possédaient un bouton physique pour afficher le menu.
- Cette pratique est devenue un peu plus rare depuis que les constructeurs essaient au maximum de dématérialiser les boutons. Mais depuis Android 2.3, il existe un bouton directement dans l'interface du système d'exploitation, qui permet d'ouvrir un menu.

Les Menus

- Il existe deux sortes de menu dans Android:
  - Le menu d'options, celui qu'on peut ouvrir avec le bouton Menu sur le téléphone. Si le téléphone est dépourvu de cette touche, Android fournit un bouton dans son interface graphique pour y accéder.
  - Les menus contextuels qui sont comme les menus qui s'ouvrent à l'aide d'un clic droit sous Windows et Linux? Eh bien, dans Android ils se déroulent dès lorsqu'on effectue un long clic sur un élément de l'interface graphique
- Ces deux menus peuvent contenir des sous-menus, qui peuvent contenir des sous-menus,etc.
- On peut définir un menu de manière programmatique en Java, mais la meilleure façon de procéder est en XML.

Les Menus

- Un menu doit être peuplé avec des éléments, et c'est sur ces éléments que cliquera l'utilisateur
- Ces éléments s'appellent en XML des **<item>** et peuvent être personnalisés à l'aide de plusieurs attributs :
    - **android:id**, que vous connaissez déjà. Il permet d'identifier de manière unique un<item>. Autant d'habitude cet attribut est facultatif, autant cette fois il est vraiment indispensable.
    - **android:icon**, pour agrémenter votre<item>d'une icône.
    - android:**title**, qui sera son texte dans le menu.android:enabled="false«qui permet de désactiver par défaut un<item>.
    - **android:checkable** auquel vous pouvez mettre true si vous souhaitez que l'élément puisse être coché
    - De plus, si vous souhaitez qu'il soit coché par défaut, vous pouvez mettre **android:checked** à true.

Les Menus

- Il peut arriver que plusieurs éléments se ressemblent beaucoup ou fonctionnent ensemble, c'est pourquoi il est possible de les regrouper avec **<group>**.
- Si on veut que tous les éléments du groupe soient des CheckBox, on peut mettre au groupe l'attribut **android:checkableBehavior="all"**
- sachant que les seuls **<item>** que l'on puisse cocher sont ceux qui se trouvent dans un sous-menu.
- si on veut qu'ils soient tous des RadioButton, on peut mettre l'attribut **android:checkableBehavior="single"**

Les Menus

Côté Java:

- Il va falloir dire à Android qu'il doit parcourir le fichier **XML** pour construire le menu.
- Pour cela, on va surcharger la méthode **boolean onCreateOptionsMenu(Menu menu)** de la classe **Activity**.
- Cette méthode est lancée au moment de la première pression du bouton qui fait émerger le menu.
- Pour parcourir le **XML**, on va l'inflater
- L'identifiant d'un **<item>** permet de déterminer comment il réagit aux clics au sein de la méthode **boolean onOptionsItemSelected(MenuItem item)**.

Les Menus

Menu Contextuel:

- Le menu contextuel est très différent du menu d'options, puis qu'il n’apparaît pas quand on appuie sur le bouton d'options, mais plutôt quand on clique sur n'importe quel
- Alors, on ne veut peut-être pas que tous les objets aient un menu contextuel, c'est pourquoi il faut déclarer quels widgets en possèdent, et cela se fait dans la méthode de la classe Activity **void registerForContextMenu(View v)**
- Dès que l'utilisateur fera un clic long sur cette vue, un menu contextuel s'ouvrira (s’il est défini)

Les Menus

Menu Contextuel:

- Ce menu se définit dans la méthode suivante :
  - **void onCreateContextMenu(ContextMenu m, View v, ContextMenu.ContextMenuInfo menuInfo)**
  - m est le menu à construire, v la vue sur laquelle le menu a été appelé et menuInfo indique sur quel élément d'un adaptateur a été appelé le menu, si on se trouve dans une liste par exemple
- Il n'existe pas de méthode du type onPrepare cette fois-ci, par conséquent le menu est détruit puis reconstruit à chaque appel.
- C'est pourquoi il n'est pas conseillé de conserver le menu dans un paramètre comme nous l'avions fait pour le menu d'options.

Les Menus

Menu Contextuel:

- Pour écrire des identifiants facilement, la classe **Menu** possède une constante **Menu.FIRST** qui permet de désigner le premier élément, puis le deuxième en incrémentant, etc.
- Avec **super.onCreateContextMenu(m, v, menuInfo)** on passe les paramètres à la super classe pour permettre de récupérer le **ContextMenuInfo** dans la méthode qui gère l'évènementiel des menus contextuels, la méthode **boolean onContextItemSelected(MenuItem item)**.
- On peut récupérer des informations sur la vue qui a appelé le menu avec la méthode **ContextMenu.ContextMenuInfo getMenuInfo()** de la classe **MenuItem**.

Les Menus (Résumé)

- La création d'un menu se fait en XML pour tout ce qui est statique et en Java pour tout ce qui est dynamique.
- La déclaration d'un menu se fait obligatoirement avec un élément menu à la racine du fichier et contiendra des item.
- Un menu d'options s'affiche lorsque l'utilisateur clique sur le bouton de menu de son appareil. Il ne sera affiché que si le fichier XML représentant votre menu est désérialisé dans la méthode **boolean onCreateOptionsMenu(Menu menu)** et que vous avez donné des actions à chaque item dans la méthode **boolean onOptionsItemSelected(MenuItem item)**
- Un menu contextuel s'affiche lorsque vous appuyez longtemps sur un élément il se construit avec **void onCreateContextMenu(ContextMenu menu, View vue, ContextMenu.ContextMenuInfo menuInfo)**
- La vue qui a fait appel à ce menu contextuel est récupérée à partir de la méthode **boolean onContextItemSelected(MenuItem item)**.

ListView

- **ListView** est un view group, affiche des éléments selon une liste et peut être déplacé verticalement. **ListView** est une vue importante et est largement utilisé dans les applications Android.
- Un simple exemple de ListView est votre carnet de contacts, où vous avez une liste de vos contacts affichés dans un **ListView**.
- En plus de **ListView**, Android vous fournit un autre view similaire qui est **ExpandableListView**

ListView

ListItem

- Un **ListView** est fait à partir d'un groupe de **ListItem**. **ListItem** est une ligne (row) individuelle dans listview où les données seront affichées. Toutes les données dans listview sont affichées uniquement via listItem. Considérez Listview comme groupe défilable de ListItems.
- Un **ListItem** est une pièce de l'interface qui peut être créée par un nombre de **View**.

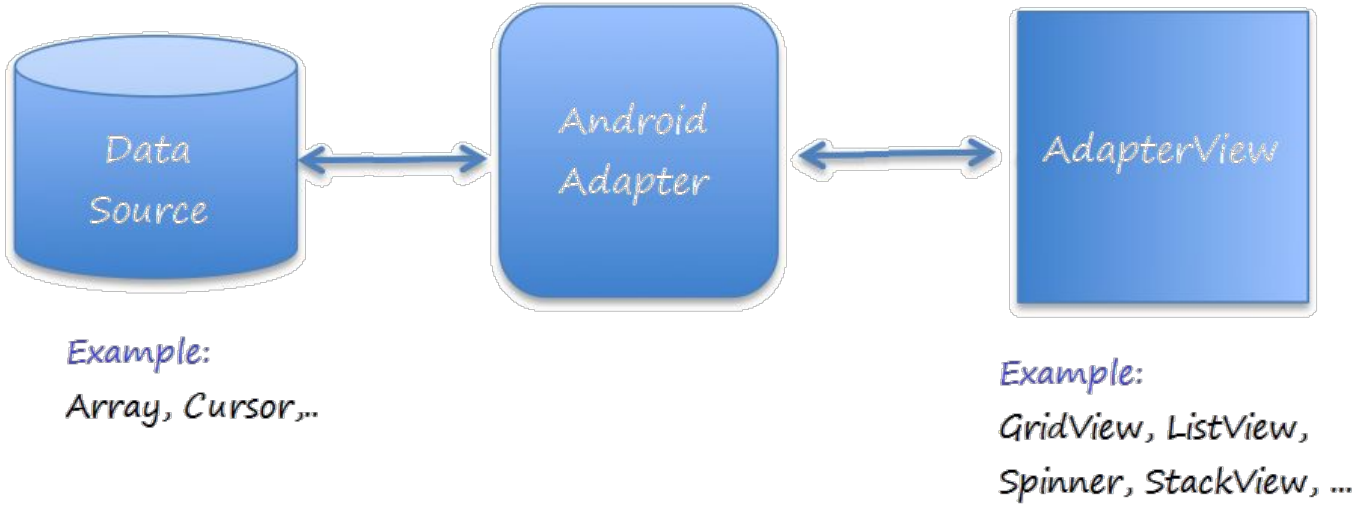
ListView

Adapter (L'adaptateur)

- **Android Adapter** (L'adaptateur) est un pont entre des **View** (par exemple comme **ListView**) et les données sous-jacentes pour ce View. Un Adapter gère des données et adapte les données dans les lignes individuelles (**ListItem**) du view.
- Vous pouvez lier des **Adapter** avec **Android ListView** via la méthode **setAdapter**. Maintenant, nous allons voir comment **Adapter** fonctionne à l'aide de l'image suivante.

ListView

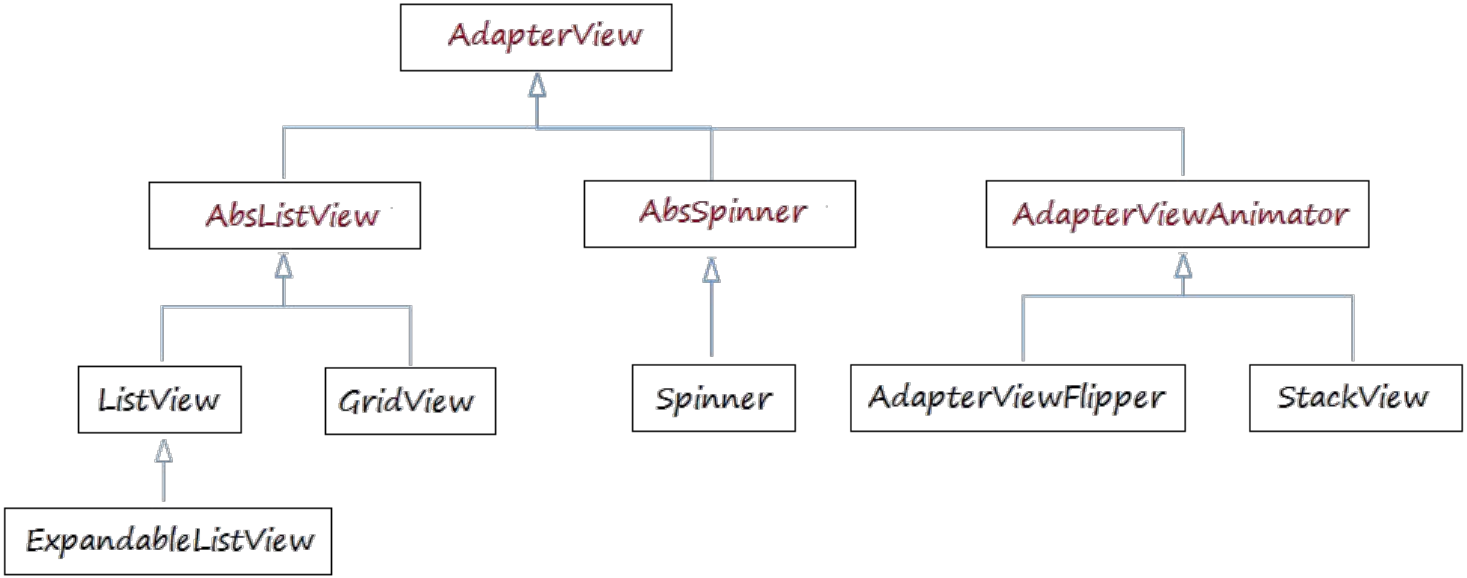
Adapter (L'adaptateur)



ListView

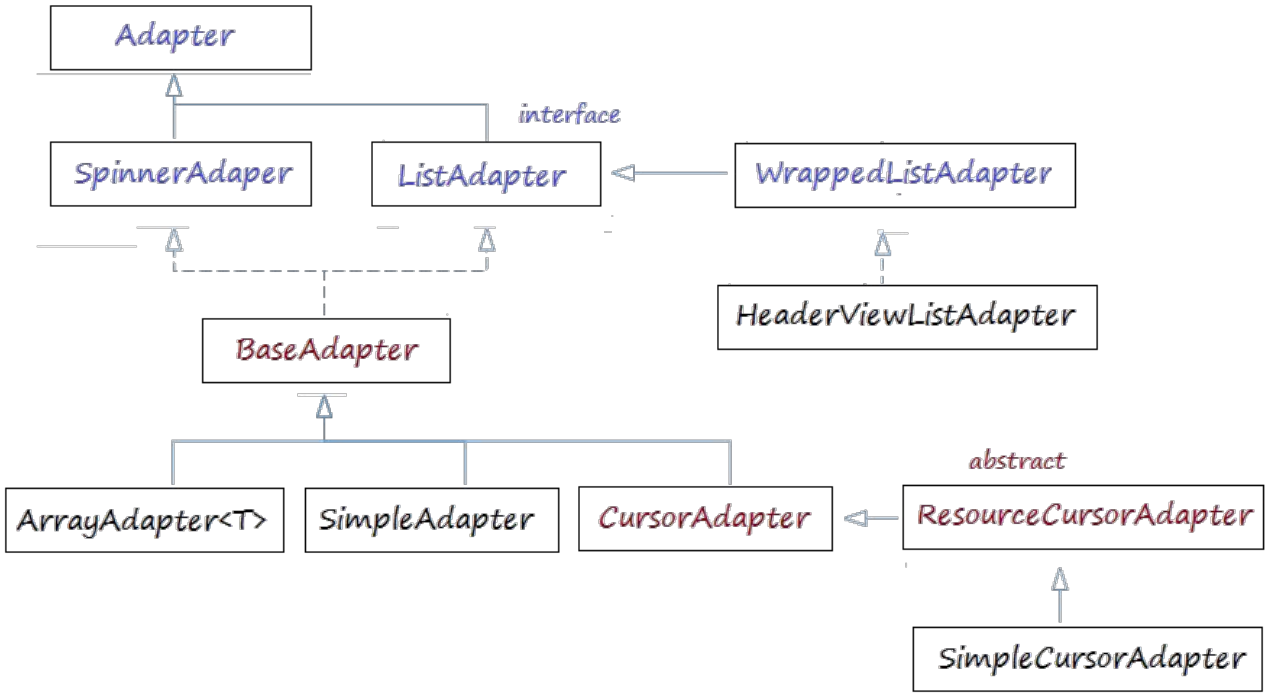
AdapterView

- Il existe plusieurs **View** qui ont eu besoin d'Android **Adapter** en vue de gérer des données affichées, ces View sont les sous-classes de la classe **AdapterView**, vous pouvez voir l'illustration suivante:



ListView

Android Adapter



SQLite et Android

- SQLite est une base de données open source, qui supporte les fonctionnalités standards des bases de données relationnelles comme la syntaxe SQL, les transactions et les prepared statement. La base de données nécessite peu de mémoire lors de l'exécution (env. 250 ko), ce qui en fait un bon candidat pour être intégré dans d'autres environnements d'exécution.
- SQLite est intégrée dans chaque appareil Android. L'utilisation d'une base de données SQLite sous Android ne nécessite pas de configuration ou d'administration de la base de données.
- L'accès à une base de données SQLite implique l'accès au système de fichiers. Cela peut être lent. Par conséquent, il est recommandé d'effectuer les opérations de base de données de manière asynchrone.
- Si votre application crée une base de données, celle-ci est par défaut enregistrée dans le répertoire DATA /data/APP_NAME/databases/FILENAME.

SQLite et Android

Architecture de SQLite

- Pour créer et mettre à jour une base de données dans votre application Android, vous créez une classe qui hérite de **SQLiteOpenHelper**. Dans le constructeur de votre sous-classe, vous appelez la méthode super() de **SQLiteOpenHelper**, en précisant le nom de la base de données et sa version actuelle.
- **onCreate**() - est appelée par le framework pour accéder à une base de données qui n'est pas encore créée.
- **onUpgrade**() - est appelée si la version de la base de données est augmentée dans le code de votre application. Cette méthode vous permet de mettre à jour un schéma de base de données existant ou de supprimer la base de données existante et la recréer par la méthode **onCreate**().

SQLite et Android

Architecture de SQLite

- Les deux méthodes reçoivent en paramètre un objet **SQLiteDatabase** qui est la représentation Java de la base de données.
- La classe **SQLiteOpenHelper** fournit les méthodes **getReadableDatabase**() et **getWriteableDatabase**() pour accéder à un objet **SQLiteDatabase** en lecture, respectif en écriture.
- Les tables de base de données doivent utiliser l'identifiant _id comme clé primaire de la table. Plusieurs fonctions Android s'appuient sur cette norme.

SQLite et Android

```
public class DataBaseHelper extends SQLiteOpenHelper {

    private static final String DB_NAME = "project27.db";
    private static final int DB_VERSION = 1;
    private static final String TAG = "DataBaseHelper";

    private static DataBaseHelper dataBaseHelper;

    private DataBaseHelper(@Nullable Context context) {
        super(context, DB_NAME, null, DB_VERSION);
    }
    public static DataBaseHelper getInstance(Context context) {
        if(dataBaseHelper==null){
            dataBaseHelper = new DataBaseHelper(context);
        }
        return dataBaseHelper;
    }
    @Override
    public void onCreate(SQLiteDatabase sqLiteDatabase) {
        sqLiteDatabase.execSQL(Product.CREATE_TABLE);
    }
    @Override
    public void onUpgrade(SQLiteDatabase sqLiteDatabase, int i, int i1) {
    }
}
```

SQLite et Android

Insertion

```
SQLiteDatabase database  = dataBaseHelper.getWritableDatabase();
ContentValues contentValues  = new ContentValues();
contentValues.put("name", product.name);
contentValues.put("description", product.description);
contentValues.put("price", product.price);
contentValues.put("quantityInStock", product.quantityInStock);
contentValues.put("alertQuantity", product.alertQuantity);
long id = database.insert(Product.TABLE_NAME, null, contentValues);
```

SQLite et Android

Recuperation

```
SQLiteDatabase database  = dataBaseHelper.getReadableDatabase();
String[] column = new String[]{"id", "name", "description", "price",
"quantityInStock", "alertQuantity"};

String where = "id=?";
String[] whereArgs = new String[]{id+""};
@SuppressLint("Recycle")
Cursor cursor = database.query(Product.TABLE_NAME, column, where,
       whereArgs, "", "", "");
```

SQLite et Android

Mise à jour

```
SQLiteDatabase database = dataBaseHelper.getWritableDatabase();
ContentValues contentValues = new ContentValues();
contentValues.put("name", product.name);
contentValues.put("description", product.description);
contentValues.put("price", product.price);
contentValues.put("quantityInStock", product.quantityInStock);
contentValues.put("alertQuantity", product.alertQuantity);

String where = "id=?";
String[] whereArgs = new String[]{id+""};

database.update(Product.TABLE_NAME, contentValues, where, whereArgs);
```

SQLite et Android

Suppression

```
SQLiteDatabase database  = dataBaseHelper.getWritableDatabase();
int rowDelete = database.delete(Product.TABLE_NAME, "id=?", new
String[]{""+id});
```