File -> Settings -> Editor -> Auto Import -> Java

**Cours Android : initiation**

1. **Introduction (wikipédia)**

**Android** est un [système d'exploitation mobile](https://fr.wikipedia.org/wiki/Syst%C3%A8me_d%27exploitation_mobile) fondé sur le [noyau Linux](https://fr.wikipedia.org/wiki/Noyau_Linux) et développé par [Google](https://fr.wikipedia.org/wiki/Google).

Lancé en juin 2007 à la suite du rachat par [Google](https://fr.wikipedia.org/wiki/Google) en 2005 de la [*startup*](https://fr.wikipedia.org/wiki/Jeune_pousse_(entreprise)) du même nom, le système avait d'abord été conçu pour les [smartphones](https://fr.wikipedia.org/wiki/Smartphone) et [tablettes tactiles](https://fr.wikipedia.org/wiki/Tablettes_tactiles), puis s'est diversifié dans les objets connectés et ordinateurs comme les télévisions ([Android TV](https://fr.wikipedia.org/wiki/Android_TV)), les voitures ([Android Auto](https://fr.wikipedia.org/wiki/Android_Auto)), les [Chromebook](https://fr.wikipedia.org/wiki/Chromebook) ([Chrome OS](https://fr.wikipedia.org/wiki/Chrome_OS) qui utilise les applications Android) et les [*smartwatch*](https://fr.wikipedia.org/wiki/Montre_intelligente) ([*Wear OS*](https://fr.wikipedia.org/wiki/Wear_OS)).

En 2015, Android est devenu le système d'exploitation mobile le plus utilisé dans le monde. Android est diffusé sous trois formes :

* Il peut être modifié par les constructeurs qui y ajoutent leurs surcouches, apportant ainsi des fonctionnalités supplémentaires mais au détriment du délais d'obtention des nouvelles mises à jour qui est parfois important.
* Il peut également être installé sans surcouche comme sur les smartphones [Android One](https://fr.wikipedia.org/wiki/Android_One) qui bénéficient de ce fait rapidement des nouvelles versions du système ou encore sur les smartphones Android Go, qui disposent quant à eux d'une version allégée capable de fonctionner de manière rapide sur du matériel d'entrée de gamme.
* Android existe enfin aussi sous la forme de différentes versions alternatives également appelées *ROM Custom* (ou [*forks*](https://fr.wikipedia.org/wiki/Fork_(d%C3%A9veloppement_logiciel))).

Android est défini comme étant une *pile de logiciels* destinés à fournir une solution pour les appareils mobiles. Cette pile comporte un système d'exploitation (comprenant un [noyau Linux](https://fr.wikipedia.org/wiki/Noyau_Linux)), les applications clés telles que le navigateur web, le téléphone et le carnet d'adresses ainsi que des logiciels intermédiaires entre le système d'exploitation et les applications. L'ensemble est organisé en cinq couches distinctes :

* le [noyau Linux](https://fr.wikipedia.org/wiki/Noyau_Linux) avec les [pilotes](https://fr.wikipedia.org/wiki/Pilote_informatique) ;
* des [bibliothèques logicielles](https://fr.wikipedia.org/wiki/Biblioth%C3%A8que_logicielle) telles que [WebKit](https://fr.wikipedia.org/wiki/WebKit)/[Blink](https://fr.wikipedia.org/wiki/Blink_(moteur_de_rendu)), [OpenGL ES](https://fr.wikipedia.org/wiki/OpenGL_ES), [SQLite](https://fr.wikipedia.org/wiki/SQLite) ou [FreeType](https://fr.wikipedia.org/wiki/FreeType) ;
* un [environnement d'exécution](https://fr.wikipedia.org/wiki/Environnement_d%27ex%C3%A9cution) et des bibliothèques permettant d'exécuter des programmes prévus pour la [plate-forme Java](https://fr.wikipedia.org/wiki/Plate-forme_Java) ;
* un [framework](https://fr.wikipedia.org/wiki/Framework) — kit de développement d'applications ;
* un lot d'applications standard qui comprend un [environnement de bureau](https://fr.wikipedia.org/wiki/Environnement_de_bureau), un carnet d'adresses, un navigateur web et un téléphone.

Les services offerts par Android facilitent l'exploitation des réseaux de télécommunications [GSM](https://fr.wikipedia.org/wiki/Global_System_for_Mobile_Communications), [Bluetooth](https://fr.wikipedia.org/wiki/Bluetooth), [Wi-Fi](https://fr.wikipedia.org/wiki/Wi-Fi) et [UMTS](https://fr.wikipedia.org/wiki/Universal_Mobile_Telecommunications_System), la manipulation de médias, notamment de la vidéo [H.264](https://fr.wikipedia.org/wiki/H.264), de l'audio [MP3](https://fr.wikipedia.org/wiki/MPEG-1/2_Audio_Layer_3) et des images [JPEG](https://fr.wikipedia.org/wiki/JPEG) ainsi que d'autres formats, l'exploitation des [senseurs](https://fr.wikipedia.org/wiki/Senseurs) tels que les capteurs de mouvements, la caméra, la boussole et le récepteur [GPS](https://fr.wikipedia.org/wiki/Global_Positioning_System), l'utilisation de l'écran tactile, le stockage en [base de données](https://fr.wikipedia.org/wiki/Base_de_donn%C3%A9es), le [rendu](https://fr.wikipedia.org/wiki/Rendu) d'images en 2D ou 3D en utilisant le [processeur graphique](https://fr.wikipedia.org/wiki/Processeur_graphique), l'affichage de [page web](https://fr.wikipedia.org/wiki/Page_web), l'exécution [multitâche](https://fr.wikipedia.org/wiki/Multit%C3%A2che) des applications et l'envoi de messages [SMS](https://fr.wikipedia.org/wiki/Short_Message_Service).

Android et la plateforme Java

Jusqu'à sa version [4.4](https://fr.wikipedia.org/wiki/Android_KitKat), Android comporte une [machine virtuelle](https://fr.wikipedia.org/wiki/Machine_virtuelle) nommée [Dalvik](https://fr.wikipedia.org/wiki/Dalvik_(machine_virtuelle)), qui permet d'exécuter des programmes prévus pour la [plate-forme Java](https://fr.wikipedia.org/wiki/Plate-forme_Java). C'est une machine virtuelle conçue dès le départ pour les appareils mobiles et leurs ressources réduites — peu de puissance de calcul et peu de mémoire.

Le [*bytecode*](https://fr.wikipedia.org/wiki/Bytecode) de Dalvik est différent de celui de la [machine virtuelle Java](https://fr.wikipedia.org/wiki/Machine_virtuelle_Java) d’Oracle (JVM) et le processus de construction d'une application est différent : le code source de l'application, en langage [Java](https://fr.wikipedia.org/wiki/Java_(langage)), est tout d'abord compilé avec un compilateur standard qui produit un [*bytecode*](https://fr.wikipedia.org/wiki/Bytecode) pour JVM (*bytecode* standard de la plateforme Java) puis ce dernier est traduit en *bytecode* pour Dalvik par un programme inclus dans Android.

L'ensemble de la [bibliothèque standard](https://fr.wikipedia.org/wiki/Biblioth%C3%A8que_standard) d'Android ressemble à [J2SE](https://fr.wikipedia.org/wiki/Java_SE) (*Java Standard Edition*) de la plateforme Java. La principale différence est que les bibliothèques d'[interface graphique](https://fr.wikipedia.org/wiki/Interface_graphique) [AWT](https://fr.wikipedia.org/wiki/Abstract_Window_Toolkit) et [Swing](https://fr.wikipedia.org/wiki/Swing_(Java)) sont remplacées par des bibliothèques d'Android[7](https://fr.wikipedia.org/wiki/Android#cite_note-smith-7).

Le développement d'applications pour Android s'effectue avec un ordinateur personnel sous [Mac OS](https://fr.wikipedia.org/wiki/Mac_OS), [Windows](https://fr.wikipedia.org/wiki/Microsoft_Windows) ou [Linux](https://fr.wikipedia.org/wiki/Linux) en utilisant le [JDK](https://fr.wikipedia.org/wiki/Java_Development_Kit) de la plate-forme Java et des outils pour Android.

Des outils qui permettent de manipuler le téléphone ou la tablette, de la simuler par une [machine virtuelle](https://fr.wikipedia.org/wiki/Machine_virtuelle) et de créer des fichiers APK (les fichiers de [paquet](https://fr.wikipedia.org/wiki/Paquet_(logiciel)) d'Android).

La bibliothèque d'Android permet la création d'[interfaces graphiques](https://fr.wikipedia.org/wiki/Interface_graphique) selon un procédé similaire aux frameworks de quatrième génération que sont [XUL](https://fr.wikipedia.org/wiki/XML-based_User_interface_Language), [JavaFX](https://fr.wikipedia.org/wiki/JavaFX) ou [Silverlight](https://fr.wikipedia.org/wiki/Silverlight) : l'interface graphique peut être construite par déclaration.

La programmation consiste à déclarer la composition de l'interface dans des fichiers [XML](https://fr.wikipedia.org/wiki/Extensible_Markup_Language) ; la description peut comporter des *ressources*.

Ces déclarations sont ensuite transformées en objets tels que des fenêtres et des boutons, qui peuvent être manipulés par Java.

Les écrans ou les fenêtres (*activités* dans d'Android), sont remplis de plusieurs *vues* ; chaque vue étant une pièce d'interface graphique (bouton, liste, case à cocher…).

Android 3.0, destiné aux tablettes, introduit la notion de *fragments* : des panneaux contenant plusieurs éléments visuels.

Une tablette ayant — contrairement à un téléphone — généralement suffisamment de place à l'écran pour plusieurs panneaux

*Android Runtime* (ART)

À partir de la version [5.0](https://fr.wikipedia.org/wiki/Android_Lollipop) sortie en 2014, l'[environnement d'exécution](https://fr.wikipedia.org/wiki/Environnement_d%27ex%C3%A9cution) [ART](https://fr.wikipedia.org/wiki/ART_(Android)) (*Android RunTime*) remplaça la machine virtuelle [Dalvik](https://fr.wikipedia.org/wiki/Dalvik_(machine_virtuelle)). Cet environnement d'exécution plus performant est développé par Google pour pallier le potentiel limité de Dalvik, créé en 2007.

Avec ART, contrairement à la machine virtuelle java, les fichiers .apk ne sont plus lancés directement mais décompressés et lancés avec de nouvelles bibliothèques et API ; les applications prennent ainsi plus de place (+20 %), mais les gains en performance et en autonomie des batteries sont conséquents (+20 à 30 %).

Identité visuelle : logo

[](https://commons.wikimedia.org/wiki/File:Android_robot.svg?uselang=fr)

Le personnage nommé *Bugdroid* est le petit robot vert utilisé par Google pour présenter Android. Ce personnage est sous la licence « *Creative Commons* by (3.0) » et peut donc être utilisé librement.

Le site *Engadget* annonce que Bugdroid, le logo d’Android, serait en fait un personnage d’un jeu des années 1990 sur Atari : [*Gauntlet: The Third Encounter*](https://fr.wikipedia.org/wiki/Gauntlet:_The_Third_Encounter).

En 2019 Google fait évoluer l'identité visuelle d'android, et abandonne la numérotation des versions avec des noms de desserts

**Histoire**

Android doit son nom à la [*startup*](https://fr.wikipedia.org/wiki/Jeune_pousse_(entreprise)) éponyme spécialisée dans le développement d’applications mobiles rachetée par Google en août 2005, nom venant lui-même d'« [androïde](https://fr.wikipedia.org/wiki/Andro%C3%AFde) » qui désigne un robot construit à l'image d'un être humain. Le logiciel, qui avait été surnommé *gPhone*  et qui était initialement prévu pour être un [système d'exploitation](https://fr.wikipedia.org/wiki/Syst%C3%A8me_d%27exploitation) pour appareil photo, fut proposé gratuitement et laissé librement modifiable par les fabricants de téléphones mobiles. Le gPhone a été lancé en octobre 2008 aux États-Unis dans un partenariat de distribution exclusif entre [Google](https://fr.wikipedia.org/wiki/Google) et [T-Mobile](https://fr.wikipedia.org/wiki/T-Mobile).

Les crises de sécurité d'Android

Lors de l'été 2015, Android a fait face à plusieurs crises nuisant à la sécurité de tous ses utilisateurs.

* La première crise concernait la faille *Stagefright* qui pouvait perturber jusqu'à 95 % des terminaux fonctionnant avec le système d’exploitation de Google par un simple [MMS](https://fr.wikipedia.org/wiki/Multimedia_Messaging_Service).

Le pirate peut ainsi avoir accès à quasiment toutes les données sur le téléphone. Depuis l'apparition de cette faille, Google distribue des mises-à-jour de sécurité pour tous les appareils sous Android, et celles-ci se déroulent au début de chaque mois.

* La deuxième est un bug découvert par des chercheurs de [*Trend Micro*](https://fr.wikipedia.org/wiki/Trend_Micro) qui paralysa les téléphones et tablettes avec le [système d’exploitation](https://fr.wikipedia.org/wiki/Syst%C3%A8me_d%E2%80%99exploitation) de Google.
* En novembre 2016, un logiciel espion chinois fut découvert dans des [smartphones](https://fr.wikipedia.org/wiki/T%C3%A9l%C3%A9phone_intelligent) Android. Il avait été installé nativement par la société chinoise [*AdUps*](https://fr.wikipedia.org/w/index.php?title=AdUps&action=edit&redlink=1) sur 700 millions d'appareils Android pour collecter les données de leurs utilisateurs.

**Versions**

Android a connu plusieurs [mises à jour](https://fr.wikipedia.org/wiki/Mise_%C3%A0_jour_(informatique)) depuis sa première version. Ces mises à jour servent généralement à corriger des [bugs](https://fr.wikipedia.org/wiki/Bug_(informatique)), à améliorer l'aspect graphique ou à ajouter de nouvelles fonctionnalités. Dans l’ensemble, chaque version est développée sous un [nom de code](https://fr.wikipedia.org/wiki/Nom_de_code) basé sur des [desserts](https://fr.wikipedia.org/wiki/Dessert) et suivent une logique alphabétique :

| **Version** | **Dernière révision** | **Nom de code** | **Date de sortie** | **Version du noyau linux** |
| --- | --- | --- | --- | --- |
| 1.0 |  | Aucun | 11 novembre [2007](https://fr.wikipedia.org/wiki/2007) | ? |
| 1.1 |  | *Petit Four* | [22](https://fr.wikipedia.org/wiki/22_octobre) [octobre](https://fr.wikipedia.org/wiki/Octobre_2008) [2008](https://fr.wikipedia.org/wiki/2008) |
| 1.5 |  | *Cupcake* | [30](https://fr.wikipedia.org/wiki/30_avril) [avril](https://fr.wikipedia.org/wiki/Avril_2009) [2009](https://fr.wikipedia.org/wiki/2009) | Linux  2.6.27 |
| 1.6 | *Donut* | [15](https://fr.wikipedia.org/wiki/15_septembre) [septembre](https://fr.wikipedia.org/wiki/Septembre_2009) [2009](https://fr.wikipedia.org/wiki/2009) | Linux  2.6.29 |
| 2.X |  | Éclair | [26](https://fr.wikipedia.org/wiki/26_octobre) [octobre](https://fr.wikipedia.org/wiki/Octobre_2009) [2009](https://fr.wikipedia.org/wiki/2009) |
| 2.2.x |  | [Froyo](https://fr.wikipedia.org/wiki/Android_FroYo) | [20](https://fr.wikipedia.org/wiki/20_mai) [mai](https://fr.wikipedia.org/wiki/Mai_2010) [2010](https://fr.wikipedia.org/wiki/2010) | Linux 2.6.32 |
| 2.3.x | 10 | [*Gingerbread*](https://fr.wikipedia.org/wiki/Android_Gingerbread) | [6](https://fr.wikipedia.org/wiki/6_d%C3%A9cembre) [décembre](https://fr.wikipedia.org/wiki/D%C3%A9cembre_2010) [2010](https://fr.wikipedia.org/wiki/2010) | Linux 2.6.35 |
| 3.x.x |  | [*Honeycomb*](https://fr.wikipedia.org/wiki/Android_Honeycomb) | [22](https://fr.wikipedia.org/wiki/22_f%C3%A9vrier) [février](https://fr.wikipedia.org/wiki/F%C3%A9vrier_2011) [2011](https://fr.wikipedia.org/wiki/2011) | Linux 2.6.36 |
| 4.0.x | 15 | [*Ice Cream Sandwich*](https://fr.wikipedia.org/wiki/Android_Ice_Cream_Sandwich) | [19](https://fr.wikipedia.org/wiki/19_octobre) [octobre](https://fr.wikipedia.org/wiki/Octobre_2011) [2011](https://fr.wikipedia.org/wiki/2011) | Linux 3.0.1 |
| 4.1.x | 16 | [*Jelly Bean*](https://fr.wikipedia.org/wiki/Android_Jelly_Bean) | [9](https://fr.wikipedia.org/wiki/9_juillet) [juillet](https://fr.wikipedia.org/wiki/Juillet_2012) [2012](https://fr.wikipedia.org/wiki/2012) | Linux 3.0.31 |
| 4.2.x | 17 | [13](https://fr.wikipedia.org/wiki/13_novembre) [novembre](https://fr.wikipedia.org/wiki/Novembre_2012) [2012](https://fr.wikipedia.org/wiki/2012) | Linux 3.0 – 3.1 |
| 4.3.x | 18 | [24](https://fr.wikipedia.org/wiki/24_juillet) [juillet](https://fr.wikipedia.org/wiki/Juillet_2013) [2013](https://fr.wikipedia.org/wiki/2013) |
| 4.4.x | 19 | [*KitKat*](https://fr.wikipedia.org/wiki/Android_KitKat) | [31](https://fr.wikipedia.org/wiki/31_octobre) [octobre](https://fr.wikipedia.org/wiki/Octobre_2013) [2013](https://fr.wikipedia.org/wiki/2013_en_informatique) | Linux 3.4 – 3.10 |
| 5.0.x | 21 | [*Lollipop*](https://fr.wikipedia.org/wiki/Android_Lollipop) | [3](https://fr.wikipedia.org/wiki/3_novembre) [novembre](https://fr.wikipedia.org/wiki/Novembre_2014) [2014](https://fr.wikipedia.org/wiki/2014) |
| 5.1.x | 22 | [9](https://fr.wikipedia.org/wiki/9_mars) [mars](https://fr.wikipedia.org/wiki/Mars_2015) [2015](https://fr.wikipedia.org/wiki/2015) |
| 6.0 | 23 | [*Marshmallow*](https://fr.wikipedia.org/wiki/Android_Marshmallow) | [5](https://fr.wikipedia.org/wiki/5_octobre) [octobre](https://fr.wikipedia.org/wiki/Octobre_2015) [2015](https://fr.wikipedia.org/wiki/2015) | Linux 3.10 – 3.18 |
| 7.x | 24,25 | [*Nougat*](https://fr.wikipedia.org/wiki/Android_Nougat) | [22](https://fr.wikipedia.org/wiki/22_ao%C3%BBt) [août](https://fr.wikipedia.org/wiki/Ao%C3%BBt_2016) [2016](https://fr.wikipedia.org/wiki/2016) | Linux 3.18 – Linux 4.4 |
| 8.0.x | 26 | [Oreo](https://fr.wikipedia.org/wiki/Android_Oreo) | [21](https://fr.wikipedia.org/wiki/21_ao%C3%BBt) [août](https://fr.wikipedia.org/wiki/Ao%C3%BBt_2017) [2017](https://fr.wikipedia.org/wiki/2017) | Linux 4.4  – Linux 4.9 |
| 8.1.x | 27 | [5](https://fr.wikipedia.org/wiki/5_d%C3%A9cembre) [décembre](https://fr.wikipedia.org/wiki/D%C3%A9cembre_2017) [2017](https://fr.wikipedia.org/wiki/2017) |
| 9.0.x | 28 | [*Pie*](https://fr.wikipedia.org/wiki/Android_Pie) | [1er](https://fr.wikipedia.org/wiki/1er_d%C3%A9cembre) [décembre](https://fr.wikipedia.org/wiki/D%C3%A9cembre_2018) [2018](https://fr.wikipedia.org/wiki/2018) | Linux 4.14 |
| 10.0.x | 29 | [Android 10](https://fr.wikipedia.org/wiki/Android_10) | [3](https://fr.wikipedia.org/wiki/3_septembre) [septembre](https://fr.wikipedia.org/wiki/Septembre_2019) [2019](https://fr.wikipedia.org/wiki/2019) | Linux x.xx |

**Android Studio**

C'est un [environnement de développement](https://fr.wikipedia.org/wiki/Environnement_de_d%C3%A9veloppement) pour développer des applications mobiles [Android](https://fr.wikipedia.org/wiki/Android). Il est basé sur [IntelliJ IDEA](https://fr.wikipedia.org/wiki/IntelliJ_IDEA) et utilise le [moteur de production](https://fr.wikipedia.org/wiki/Moteur_de_production) [Gradle](https://fr.wikipedia.org/wiki/Gradle).

Android studio a été créé et est maintenu par Google, les versions récentes du système d'exploitation Android sont mises à jour dans l'IDE à chaque nouvelle version.

Avant Android Studio, de 2009 à 2014, [Google](https://fr.wikipedia.org/wiki/Google) propose comme environnement de développement officiel une distribution spécifique de l'environnement [Eclipse](https://fr.wikipedia.org/wiki/Eclipse_(projet)), contenant le [SDK](https://fr.wikipedia.org/wiki/Kit_de_d%C3%A9veloppement) d'Android.

Android Studio est annoncé en mai 2013, et en décembre 2014, Android Studio devient conseillé par Google, et Eclipse est délaissé

Android Studio permet principalement d'éditer les fichiers [Java](https://fr.wikipedia.org/wiki/Java_(technique))/[Kotlin](https://fr.wikipedia.org/wiki/Kotlin_(langage)) et les fichiers de configuration [XML](https://fr.wikipedia.org/wiki/Extensible_Markup_Language) d'une application Android.

Il propose entre autres des outils pour gérer le développement d'applications multilingues et permet de visualiser rapidement la mise en page des écrans sur des écrans de résolutions variées simultanément. Il intègre un [émulateur](https://fr.wikipedia.org/wiki/%C3%89mulation) permettant de faire tourner un système Android virtuel sur un ordinateur.

La dernière version d'Android studio est 3.5 de la date 20 août 2019.

**1) Téléchargement et Installation**

Télécharger Android studio sur sa page principale, <https://developer.android.com/studio/index.html>

Vous allez télécharger un fichier qui contient un ensemble d'outils indispensables pour développer les applications Android. Ce paquet contient Android Studio et un outil pour gérer l'installation du SDK Android sur votre système.

**Installation**

Assurez vous d'abord que vous avez déjà installé **Java**version 8 ou +.

Si vous avez téléchargé un fichier .exe (recommandé), double-cliquez dessus pour le lancer.

Si vous avez téléchargé un fichier .zip, décompressez le fichier ZIP, copiez le dossier android-studio dans votre dossier Program Files, ouvrez le dossier android-studio🡪 bin et lancez studio64.exe (pour les ordinateurs 64 bits) ou studio.exe. (pour les machines 32 bits).

Suivez l’assistant de configuration d’Android Studio et installez les packages SDK recommandés.

**Configuration de Android studio**

Dans la configuration vous avez deux étapes l'une pour le SDK (software development kit) et l'autre pour le AVD Android Virtual Device.

* **Android SDK (software development kit)**

C'est un ensemble d'outils de développement utilisés pour développer des applications sur la plate-forme Android. Le SDK d'Android comprend les éléments suivants:

* Des bibliothèques requises
* Débogueur (Debugger)
* Un émulateur (emulator)
* Documentation pertinente pour les interfaces de programme d'application Android (API)
* Exemple de code source
* Des tutoriels pour le système d'exploitation Android

Il se peut qu'on trouve plusieurs fois des paquets avec le même nom, dans ce cas c'est qu'il s'agit de versions différentes du même paquet, comme vous pouvez le voir dans la colonne Rev. : on trouve la version 19.1 d'Android SDK Platform-tools mais aussi la version 17, … par exemple. On essayera toujours d'avoir la dernière version de la plateforme.

* **Android Virtual Device (AVD)**

C'est une configuration de périphérique exécutée avec l'émulateur Android (Android emulator). Il fonctionne avec l'émulateur pour fournir un environnement virtuel spécifique au périphérique dans lequel installer et exécuter des applications **Android**.

## ****2) Créer une application Android****

**Présentation générale**

**Application :**

Une application est un ensemble de fenêtres entre lesquelles il est possible de naviguer.

**Activité**

Une activité est une fenêtre de l'application, elle remplit tout l'écran ainsi l'application ne peut afficher qu'une seule activité à la fois.

Une activité existe 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 par exemple un appel.

**Contexte**

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.

**Interface graphique**

C'est la fenêtre associée à une activité.

La conception de l'interface graphique d'une application pour  **Android**  peut se faire de trois manières :

* par programmation en **language XML**
* par programmation en langage Java via la création d'objets (Button,TextView etc..).
* par glisser-déposer d'éléments depuis la boite à outils(**button,TextView** etc..).

## Première application avec Android : Helloworl

Android Studio est un IDE célèbre pour sa superbe collection de raccourcis claviers. Je vous recommande d'apprendre beaucoup plus de raccourcis, la souris est lente !

* Sélectionnez Start a new Android Studio Project et renseignez les informations
* **Application name** : c’est le nom qui va apparaitre dans la liste des applications sur l’appareil et dans le Play Store.
* **Company domain** : on se base sur le nom de domaine de son entreprise pour constituer ce champ, il permet à Android Studio de déduire automatiquement un Package Name.
* **Package name** : il est utilisé comme identifiant de l’application, il permet de considérer différentes versions d’une application comme étant une même application. Il doit être unique parmi tous les packages installés sur le système.
* **Minimum required SDK** : c’est la version Android la plus ancienne sur laquelle l’application peut tourner.
* En fin Sélectionnez Blank Activity

**Les dossiers du Projet**

A la création du projet, Android Studio crée automatiquement des dossiers pour contenir les fichiers de code Java, les fichiers XML, et les fichiers multimédias.

L’explorateur de projet permet de naviguer dans ces dossiers. Les dossiers que nous utiliserons le plus sont java et res.

* **java** situé dans le répertoire du projet (sous app\) contient les classes Java
* **res** (dans app\src\main\res) contient des sous dossiers où sont stockés les ressources qui définissent l’interface de l’application comme :
* **layout** regroupe les fichiers XML qui définissent la disposition des composants sur l’écran.
* **Drawable** contient tout élément qui peut être dessiné sur l’écran : images (en PNG de préférence), formes, animations, ….
* **menu** contient les fichiers XML définissant les menus
* **mipmap** contient les images de l’icône de l'application sous différentes résolutions.
* **values** contient les fichiers XML qui définissent des valeurs constantes (des chaines de caractères, des dimensions, des couleurs, des styles etc.)
* **gradle** Android Studio utilise un système qu’on appelle Gradle pour compiler et générer les applications. Pour fonctionner le Gradle a besoin d’un script qui définit les règles de compilation et génération (configuration et dépendances).
* **manifests** contient le fichier AndroidManifest.xml, fichier de configuration de l'app

**Exécution du projet**

On peut exécuter le projet soit directement sur le téléphone avec certaine configuration ou sur un émulateur (AVD).

**Configuration du téléphone**

Activez l’option de débogage USB sur l’appareil :

Paramètres🡪sécurité et activer "source inconue"

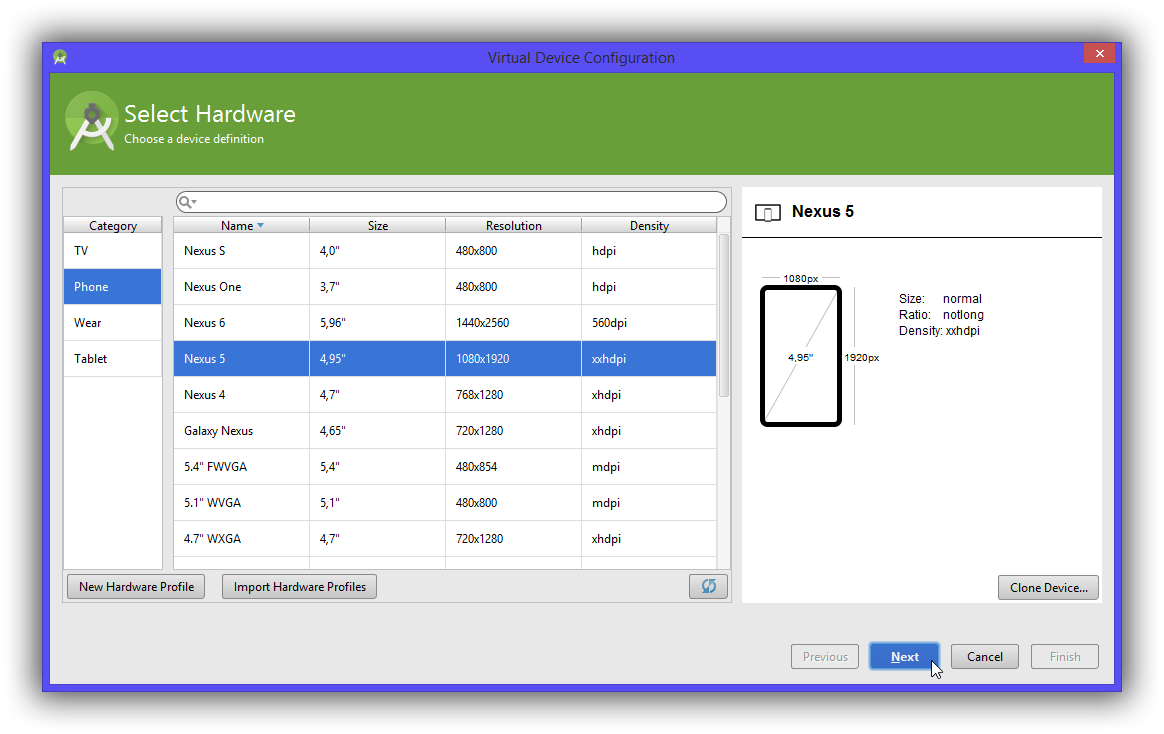
Paramètres🡪 A propos .. et touchez sept fois "Numéro de build".

Retournez ensuite à l’écran Paramètres, vous verrez apparaitre "Options développeurs", rentrez y et activez "débogage USB".

**Configuration de l'émulateur**

Menu tools🡪AVD manager

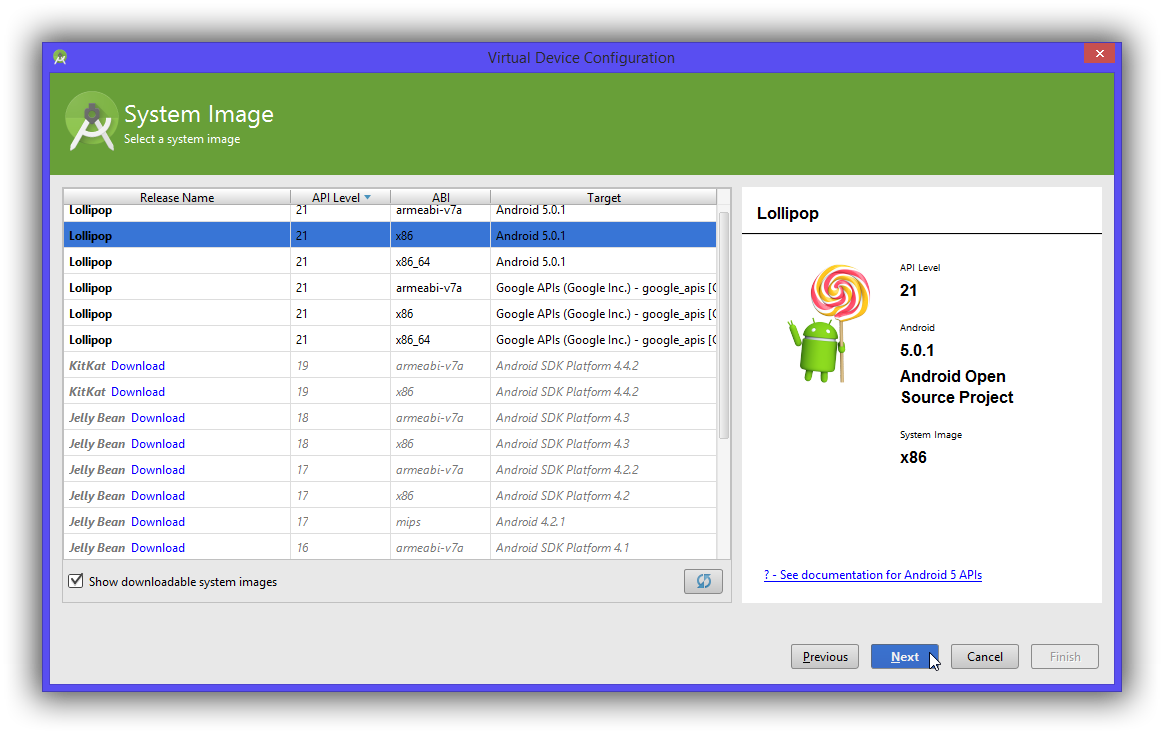
Cliquez sur le bouton Create a new virtual device.



**Category :**

permet de sélectionner si vous souhaitez émuler un téléviseur, un téléphone, ou une tablette.

* choisir un téléphone puis cliquer sur Next



Sur cet écran, chaque ligne correspond à une version d'Android.

La colonne de gauche Release Name : est le nom commercial de la version.

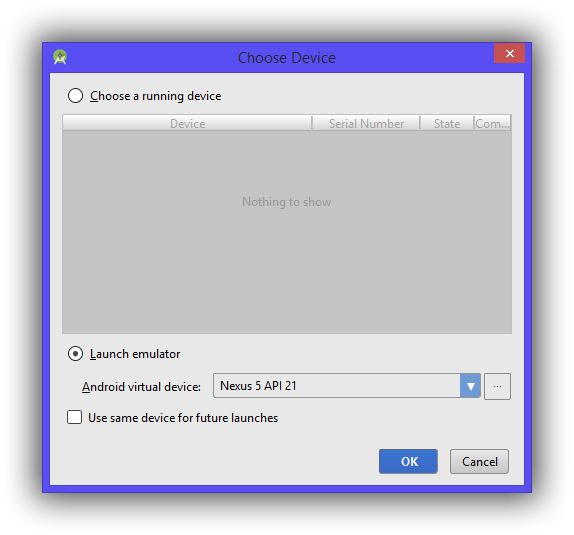
la colonne 2 AIP Level : son numéro d'API.

La colonne 3 ABI : représente l'ABI (Application binary interface), l'interface de communication entre le processus de l'AVD et la version d'Android. Si vous sélectionnez x86, ce sera comme si votre émulateur avait un processus x86 sur 32 bits, si vous sélectionnez x86\_64, idem mais sur 64 bits.

La dernière colonne représente la cible. Soit une version standard d'Android, soit une version d'Android sur laquelle sont installés les outils de Google (Google Maps par exemple), cliquez sur Next puis terminer.

**Exécution**

Sans aucune modification du projet, cliquer sur le bouton d'exécution pour obtenir une fenêtre.



Si vous avez un téléphone configuré, dont les drivers sont installés sur l'ordinateur et qui est connecté à l'ordinateur, alors cliquer sur l'option "Choose a running device" et sélectionner le téléphone, sinon, sélctionner l'émulateur créé avec "Launch emulator" puis cliquer sur OK.

**Remarque**

La méthode setContentView (View vue)permet d'indiquer l'interface graphique de notre activité. Si nous lui donnons un TextView, alors l'interface graphique affichera ce TextView et rien d'autre.

**Exemple : Remplacer le code de la méthode OnCreate par le code suivant :**

**super**.onCreate(savedInstanceState);  
*//setContentView(R.layout.activity\_main);*TextView text = **new** TextView(**this**);  
text.setText(**"Bonjour, tout le monde."**);  
setContentView(text);

1. **Applications et activité**

**Application Android**

• Une activité = un programme + une interface

• Un service = un programme sans interface

• Une application =

– Une activité principale

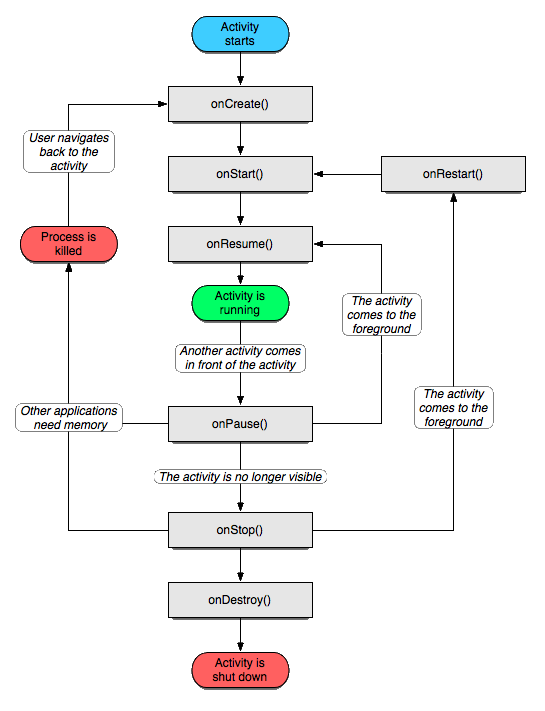
– Eventuellement une ou plusieurs activités secondaires

– Eventuellement un ou plusieurs services

– Eventuellement un ou plusieurs écouteurs d’intentions diffusées

– Eventuellement un ou plusieurs fournisseurs de contenu

## ****Cycle de vie d’une application Android****

[](http://www.mti.epita.fr/blogs/wp-content/uploads/2010/05/activity_lifecycle1.png)

* **onCreate**

Cette méthode est appelée à la création de l'activité  (Activity). Elle sert à initialiser l'activité ainsi que toutes les données nécessaires.

Quand la méthode **OnCreate** est appelée, on lui passe un **Bundle** en argument. Ce Bundle contient l’état de sauvegarde enregistré lors de la dernière exécution de l'activité.

* **onStart**

C'est le début d’exécution de l'activité (début du passage au premier plan).

Si l'activité ne peut pas aller en avant plan quelque soit la raison, l’activité sera transférée à **OnStop**.

* **onResume**

Cette méthode est appelée après **OnStart**. A la fin de l’appel de la méthode **onResume** l'application se trouve au premier plan et reçoit les interactions utilisateurs.

* **onPause**

Si une autre activité passe au premier plan, la méthode **onPause** est appelée sur l'activité. Afin que vous sauvegardiez l’état de l’activité et les différents traitements effectués par l’utilisateur.

**onStop**

Appelée quand l'activité n’est plus visible. Dans cette méthode vous devez arrêter tous les traitements et services exécutés par l'application.

**onDestroy**

Appelée quand l'application est totalement fermée (Processus terminé). Toutes les données non sauvegardées sont perdues.

## ****Configuration****

**Unités de mesure dans les fichiers XML**

• Dans les fichiers XML, les dimensions des éléments d’interface (taille, marges, …) peuvent être exprimées en diverses unités :

– Pixels (px)

– Pouces (in)

– Millimètres (mm)

– Points (pt) = 1/72 pouce

– Pixel à densité indépendante (dp) 1 dp = 1 pixel pour un écran de 160 dpi

– Pixel à taille indépendante (sp) relatif à la taille des polices de caractères

**Couleurs dans les fichiers XML**

• pour XML, les couleurs sont exprimées sous la forme d’une chaîne de caractères codant les composantes en hexadécimal : "#AARRVVBB"

– AA est l’opacité (00 totalement transparent, FF opaque)

– RR est la composante rouge (00 à FF)

– VV est la composante verte (00 à FF)

– BB est la composante bleue (00 à FF)

Si AA est omis la couleur est opaque

**Exemple**

**Android:backGround="#FF0000"**

**Les conteneurs**

• FrameLayout (un seul élément)

• AbsoluteLayout (plusieurs éléments placés par leur coordonnées)

• LinearLayout (plusieurs éléments placés horizontalement ou verticalement sans ascenseurs)

• TableLayout (plusieurs éléments en tableau sans ascenseurs)

• RelativeLayout (plusieurs éléments placés relativement les uns aux autres)

• ScrollView (un seul élément avec ascenseur vertical)

• HorizontalScrollView (un seul élément avec ascenseur horizontal)

**L'interface graphique : les vues simples**

En Android, on construit les IHM avec un fichier XML rangé dans res\layout

* Ce fichier est repéré dans le code Java par R.layout.*nom\_Fichier*
* et on positionne l'IHM de l'activité à l'aide de ce fichier par

setContentView(R.layout.*nom\_Fichier*);

* On récupère un composant graphique de l'IHM par la méthode findViewById(R.id.*id\_du\_Composant*);
* Certains composants graphiques permettent d'afficher beaucoup d'items par des mécanismes de défilement : les Spinner, Gallery, GridView, et ListView

**Conseils**

• Ecran tactile de petite taille alors :

– Eviter les interfaces trop denses (on ne peut pas agrandir l’écran comme on agrandit une fenêtre)

– Eviter les éléments cliquables trop petits (il faut pouvoir cliquer avec le doigt même avec de gros doigts)

– Eviter les éléments cliquables trop tassés (il faut pouvoir cliquer sur le bon élément même si on vise mal)

• Le défilement se fait par touché/glissé

– Pas trop d’ascenseurs (on ne peut pas faire défiler un conteneur entier ET des éléments de ce conteneur dans le même sens)

– Pas d’ascenseurs mal placés (si tous les éléments sont cliquables comment faire défiler sans cliquer ?)

**Création d’interfaces**

Une interface est un arbre dont la racine est l’écran et les feuilles sont les éléments de l’interface (boutons, textes, cases à cocher, …)

Une interface peut être créée en utilisant un fichier xml ou en utilisant une classe java.

• **Par programme java**: comparable à java swing, mais avec des classes propres à Android

– Définition de conteneurs

– Définition d’éléments d’interaction (widgets) + placement et ajout dans les conteneurs

• **Par fichiers xml** (forme déclarative statique)

**Création d'une interface avec un fichier XML**

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

**<!--** Commentaire

**-->**

**<Classe du conteneur principal**

**xmlns:android="http://schemas.android.com/apk/res/android"**

Propriétés du conteneur principal **>**

**<Classe d'éléments d'interface ou de conteneur**

propriétés du conteneur ou de l'élément d'interface

**/>**

…

**<Classe d'élément d'interface ou de conteneur**

propriétés du conteneur ou de l'élément d'interface

**/>**

**</Classe du conteneur principal>**

**Exemple**

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

<LinearLayout

xmlns:android=*"http://schemas.android.com/apk/res/android"*

android:layout\_width=*"fill\_parent"*

android:layout\_height=*"fill\_parent"*

android:orientation=*"vertical"* >

<TextView

android:id=*"@+id/text"*

android:layout\_width=*"wrap\_content"*

android:layout\_height=*"wrap\_content"*

android:text=*"Hello, I am a TextView"* />

<Button

android:id=*"@+id/button"*

android:layout\_width=*"wrap\_content"*

android:layout\_height=*"wrap\_content"*

android:text=*"Hello, I am a Button"* />

</LinearLayout>

**Utilisation de l'interface**

• Dans l’activité principale

**setContentView(R.layout.*nom\_du\_fichier\_xml*)**

**Exemple**

1. Créer un projet android
2. clic droit sur **res**. puis **New / Androïd XML File**
3. Choisir comme nom : **layout\_file\_name.xml**

****

****

1. Saisir le texte suivant dans le fichier **layout\_file\_name.xml**

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

**<LinearLayout xmlns:android=*"http://schemas.android.com/apk/res/android"***

**android:layout\_width=*"fill\_parent"***

**android:layout\_height=*"fill\_parent"***

**android:orientation=*"vertical"* >**

**<TextView**

**android:id=*"@+id/text"***

**android:layout\_width=*"wrap\_content"***

**android:layout\_height=*"wrap\_content"***

**android:text=*"Hello, I am a TextView"* />**

**<Button**

**android:id=*"@+id/button"***

**android:layout\_width=*"wrap\_content"***

**android:layout\_height=*"wrap\_content"***

**android:text=*"Hello, I am a Button"* />**

**</LinearLayout>**

Le projet doit se présenter comme suit :

****

1. Ouvrir ensuite le projet
2. Dans **onCreate,** remplacer le code par le code suivant :

**public** void onCreate(Bundle savedInstanceState) {

super.onCreate(savedInstanceState);

setContentView(R.layout.layout\_file\_name);

}}

1. Exécuter.
2. **Les widgets**

Un widget est un élément de base qui permet d'afficher le contenu ou permet d'interagir avec l'application. Chaque widget possède un nombre important d'attributs XML et de méthodes Java.

**Création des widgets**

Avec la diversité des machines sous lesquelles fonctionne Android, il faut vraiment exploiter toutes les opportunités offertes par les ressources pour développer des applications qui fonctionneront sur la majorité des terminaux.

Une application Android polyvalente possède un fichier XML pour chaque type d'écran, de façon à pouvoir s'adapter. **En effet, si vous développez une application uniquement à destination des petits écrans, les utilisateurs de tablettes trouveront votre travail illisible et ne l'utiliseront pas du tout.**

**En plus, avec Eclipse, on peut créer des interfaces graphiques à la souris. Il est en effet possible d'ajouter un élément et de le positionner avec sa souris.**

Ouvrez le fichier (**activity\_main.xml**) qui se trouve dans le répertoire res/layout. Il s'agit du fichier **activity\_main.xml**.

On peut placer différentes vues en cliquant dessus depuis le menu de gauche, puis en les déposant sur l'activité.

**Pour accéder au fichier XML du projet, on a deux onglets Graphical Layout et activity\_main.xml.**

|  |  |
| --- | --- |
|  | <RelativeLayout  xmlns:android="http://schemas.android.com/apk/res/android"  xmlns:tools="http://schemas.android.com/tools"  android:layout\_width="fill\_parent"  android:layout\_height="fill\_parent" >  …..  </RelativeLayout> |

Avec **Swing** les objets graphiques qui englobent d'autres objets graphiques s'appellent des layouts (en français des gabarits).

Un layout est une vue spéciale qui peut contenir d'autres vues et qui n'est pas destinée à fournir du contenu ou des contrôles à l'utilisateur.

Les layouts se contentent de disposer les vues d'une certaine façon.

Les vues sont les enfants, le layout (la vue englobante) est le parent. Une vue qui ne peut pas englober d'autres est appelée un widget.

**Propriété communes (des conteneurs et des widgets** )

Une vue peut avoir des attributs, qui permettent de modifier certains de ses aspects. Certains de ces attributs sont spécifiques à certaines vues, d'autres sont communs.

* **Identifiant**

Un identifiant est un attribut qui peut être associé à chaque élément décrit dans un fichier XML, un tel identifiant permet d'accéder à l'objet.

**Avec xml**

android:id="@+id/*id\_element*"

* @ signifie qu'on va parler d'un identifiant,
* id est la classe où se situe l'identifiant dans R.java,
* *id\_element* : est le nom de l'identifiant.

**android:id=*"@+id/text"***

**Avec java : setId(int)**

* ***Placement des éléments :*** place prise par l’élément dans le conteneur
* layout\_width : définit la largeur que prend la vue
* layout\_height : définit la hauteur qu'elle prend.

Ces deux attributs peuvent prendre une valeur parmi les trois suivantes :

* fill\_parent : la vue prend autant de place que son parent
* wrap\_content : la vue prend la taille suffisante pour écrire le texte ;
* Une valeur numérique précise avec une unité.

**Unités de mesure dans les fichiers** XML

• Dans les fichiers XML, les dimensions des éléments d’interface (taille, marges, …) peuvent être exprimées en diverses unités :

– Pixels (px)

– Pouces (in)

– Millimètres (mm)

– Points (pt) = 1/72 pouce

– Pixel à densité indépendante (dp) : 1 dp = 1 pixel pour un écran de 160 dpi. Il s'agit d'une unité qui est indépendante de la résolution de l'écran. En effet, les autres unités comme le pixel (px) ou le millimètre (mm) varient d'un écran à l'autre…

Par exemple si vous mettez une taille de 500 dp pour un widget, il aura toujours la même dimension quelque soit la taille de l'écran. Si vous mettez une dimension de

80 mm pour un widget, il sera grand pour un grand écran… et énorme pour un petit écran.

– Pixel à taille indépendante (sp) relatif à la taille des polices de caractères

• Forme : Dans les fichiers XML les unités sont exprimées sous la forme : “24.5mm” ou “65px” …

**Remarque**

Vous pouvez remplacer fill\_parent par match\_parent.

* **Positionnement des éléments : android:layout\_gravity**

Les valeurs possibles sont : top, bottom, left, right, center\_vertical, fill\_vertical, center\_horizontal, fill\_horizontal, center, fill

* **Marges internes**

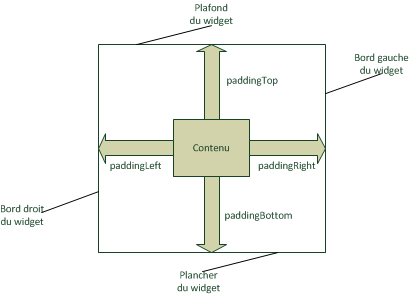
android:padding

android:layout\_paddingBottom , android:layout\_paddingLeft ,

android:layout\_paddingRight , android:layout\_paddingTop

**Exemple :**

android:padding="10.5dp"

****

**Avec java**

void setPadding (int left, int top, int right, int bottom).

textView.setPadding(15, 105, 21, 105);

* ***Marges externes (défini pour chaque élément) :* layout\_margin**🡨🡪 **padding**

android:layout\_marginBottom , android:layout\_marginLeft ,

android:layout\_marginRight , android:layout\_marginTop.

* android:visibility : Rend l’élément visible, invisible ou absent (avec invisible la place est conservée, avec absent la place n'est pas conservée.

#### android:background : couleur ou une image de fond.

**Prise en compte des clics sur l’élément**

* *avec xml : android:clickable :* Autorise ou interdit la prise en compte des clics
* *avec java : setClickable(boolean)*

**Prise en compte des clics longs sur l’élément**

*avec xml : android:longClickable :* **Autorise** ou interdit des clics longs

*avec java : setLongClickable(boolean)*

**Récupération de l'identifiant : findViewById**

L'identifiant défini en XML peut être utilisé dans le code java. Pour cela, on utilise la méthode View **findViewById (int id).**

Attention, cette méthode renvoie un objet de type ***View***, il faut donc la "caster" dans le type de destination.

**Exemple**

Dans l'exemple suivant, l'interface graphique est référencée par **R.layout.activity\_main**, il s'agit du layout d'identifiant main.

**Fichier xml**

xmlns:android="http://schemas.android.com/apk/res/android"

xmlns:tools="http://schemas.android.com/tools"

android:layout\_width="fill\_parent"

android:layout\_height="fill\_parent" **>**

**<TextView**

android:id="@+id/text"

android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content"

android:padding="25.7dp"

android:text="@string/hello\_world"

**</RelativeLayout>**

**Classe java**

import android.app.Activity;

import android.os.Bundle;

import android.widget.TextView;

public class exActivity extends Activity {

TextView monTexte = null;

public void onCreate(Bundle savedInstanceState) {

super.onCreate(savedInstanceState);

setContentView(R.layout.activity\_main);

monTexte = (TextView)findViewById(R.id.text);

monTexte.setText("Le texte de notre TextView");

monTexte.setPadding(50, 60, 70, 90);

} }

**Quelques vues**

**TextView**

Permet d'afficher un texte qu'on ne peut modifier. On peut aussi y insérer des textes formatés, à l'aide de balises HTML.

**Exemple**

<TextView

android:layout\_width="fill\_parent"

android:layout\_height="wrap\_content"

android:text="@string/textView" // signifie qu'on utilise une ressource de type string

android:textSize="8sp"

android:textColor="#112233" />

**Quelques propriétés**

android:textColorHighlight="#00ff00"

android:textIsSelectable="true"

android:lines="2"

android:digits : indique si la saisie n'accepte que du numérique ou pas

android:numerics="x" (où x peut être integer, signed, decimal) définit le mode de saisie numérique

android:password : pour cacher ou non le texte lors de la saisie

android:textColor RGB hexadécimal #FF0000

android:textStyle : en gras (bold), en italique (italic) ou les deux (bold\_italic) ;

android:fontFamily="Arial"

android:typeface="serif"

android:textSize="20sp"

|  |
| --- |
|  |

**En java**

textView.setText(R.string.textView);

textView.setTextSize(8);

textView.setTextColor(0x112233);

**Les boutons**

**<Button**

**android:id=*"@+id/button"***

**android:layout\_width=*"wrap\_content"***

**android:layout\_height=*"wrap\_content"***

**android:text=*"Hello, I am a Button"* />**

**ImageView : p**ermet d’afficher des images

Android dispose de deux widgets permettant d’intégrer des images dans les activités : ImageView et ImageButton.

Chacun d’eux possède un attribut **android:src** permettant de préciser l’image utilisée. Cet attribut désigne généralement une ressource graphique.

**Quelques propriétés :**

– Contenu

android:src définit une couleur ou une image.

android:tint définit une couleur qui teinte l'image

**Couleurs dans les fichiers XML**

• Dans les fichiers XML, les couleurs sont exprimées sous la forme d’une chaîne de caractères codant les composantes en hexadécimal : "#AARRVVBB"

– AA est l’opacité (00 totalement transparent, FF opaque, si AA est omis la couleur est opaque)

– RR est la composante rouge (00 à FF)

– VV est la composante verte (00 à FF)

– BB est la composante bleue (00 à FF)

– Position et dimensions de l’image

android:adjustViewBounds : La taille de l‘ImageView sera modifiée ou non

android:baselineAlignBottom :Cadrage ou pas de l'image en bas de la zone

android:cropToPadding : L'image sera ou pas coupée si elle est plus grande que la taille disponible

android:scaleType : mode de redimensionnement de l'image avec ou sans déformation (*centerCrop, fitXY")*.

– Taille

android:maxHeight : hauteur maximale

android:maxWidth : largeur maximale

#### Exemple

<ImageView android:id=*"@+id/image"*

android:src=*"@drawable/image1"*

android:layout\_width=*"fill\_parent"*

android:layout\_height=*"wrap\_content"*

android:maxHeight=*"200px"*

android:adjustViewBounds=*"true"*

android:scaleType=*"centerCrop"*

#### />

**imageButton** 

Mêmes paramètres que ImageView

android:src="couleur" : définir une couleur ou une image

android:adjustViewBounds : indiquer si la taille du bouton doit ou pas être ajustée à celle de l'image.

android:baselineAlignBottom : indiquer que l'image est placée ou pas en bas de la zone

android:cropToPadding : indiquer si l'image sera coupée ou pas si elle est plus grande que la taille disponible.

android:scaleType="s" (où s peut prendre les valeurs : matrix, fitXY,fitStart, fitCenter, fitEnd, center, centerCrop, centerInside) permet de redimensionner ou pas l'image à la taille disponible et/ou de la déformer.

android:maxHeight : définir la hauteur disponible

android:maxWidth : définir la largeur disponible

android:tint : définir une couleur qui teinte l'image

**La classe Toast**

C'est une classe qui permet à un texte d'apparaître en premier plan puis disparaît au bout d'un temps donné.

**• Création d'un Toast**

Toast.makeText(Context, String, int) renvoie l'objet de classe Toast créé.

– Le premier paramètre est l'activité

– Le deuxième est le message à afficher

– Le dernier indique la durée d'affichage, les seules valeurs possibles sont : Toast.**LENGTH\_SHORT** (2 secondes) ou Toast.**LENGTH\_LONG** (5 secondes).

**• Affichage d'un Toast**

La méthode show() affiche le message pour la durée définie lors de sa création

**Exemple**

                Toast.makeText(this,  "Bonjour " , Toast.LENGTH\_LONG).show();

1. **Interaction entre l'interface graphique et l'utilisateur**

**Les événements généraux (click, longclick, onkey et onTouch).**

**Evénement click dans xml**

* On ajoute l’attribut **android:onClick** à la déclaration XML du bouton :

<Button

android:onClick="uneMethode"

...

/>

* On définit une méthode qui prend un seul paramètre View :

public void uneMethode(View b) {

                Toast.makeText(this,  "Bonjour " , Toast.LENGTH\_LONG).show();

}

**Les listeners**

Pour pouvoir réagir à un évènement, il faut utiliser un objet qui va détecter l'évènement et permet de le traiter. Ce type d'objet s'appelle un **listener**.

Un listener est une interface qui permet de redéfinir des méthodes de *callback* et chaque méthode sera appelée au moment où se produira l'évènement associé.

Par exemple, pour intercepter l'évènement **clic** sur un **Button**, on applique l'interface View.OnClickListener sur ce bouton.

Cette interface contient la méthode de callback onClick(View vue)

Le paramètre de type View est la vue sur laquelle le clic a été effectué, qui sera appelée à chaque clic et qu'il faudra implémenter pour déterminer que faire en cas de clic.

Le package à utiliser pour OnClickListener est android.view.View.OnClickListener.

**Autres listeners**

* View.OnLongClickListener pour les clics qui durent longtemps, avec la méthode boolean onLongClick(View vue). Cette méthode doit retourner True une fois que l'action associée a été effectuée.
* View.OnKeyListener pour gérer l'appui sur une touche. On y associe la méthode boolean onKey(View vue, int code, KeyEvent event). Cette méthode doit retourner true une fois que l'action associée a été effectuée.
* View.OnTouchClickListener lorsque vous touchez avec vos doigts la vue concernée, avec la méthode public boolean onTouch(View v, MotionEvent event). Cette méthode doit retourner True une fois que l'action associée a été effectuée.

Pour associer un listener à une vue, on utilisera une méthode du type setOn[Evenement]Listener(On[Evenenement]Listener listener) avec Evenement l'évènement concerné, par exemple pour détecter les clics sur un bouton on fera :

**Exemple**

setOnClickListener(notre\_listener);

Pour intercepter ces événements on a au moins trois méthodes, par héritage, par classe anonyme ou par attribut.

**Par héritage**

Notre classe doit implémenter un listener, ce qui veut dire que l'activité interceptera d'elle-même les évènements.

Il n'est pas indispensable de gérer tous les évènements d'une interface, on peut laisser une méthode vide si on n'a pas besoin de tels évènements.

**Exemple d'implémentation :**

Afficher un message avec un Toast comme réponse au clic sur un bouton.

**//Implementer l'interface OnClickLister**

public class Main extends Activity implements View.OnClickListener {

private Button b = null;

public void onCreate(Bundle savedInstanceState) {

super.onCreate(savedInstanceState);

setContentView(R.layout.main);

b = (Button) findViewById(R.id.boutton);

b.setOnClickListener(this); **//associer un listener (qui est notre interface graphique) à un bouton.**

}

**//redéfinir la méthode onClick**

public void onClick(View v) {

/\* Réagir au clic \*/

}

}

La vue passée dans la méthode onClick(View) permet de différencier les boutons de l'application. En effet, il est possible de récupérer l'identifiant de la vue sur laquelle le clic a été effectué. Ainsi, la réaction diffère en fonction de cet identifiant :

public void onClick(View v) {

// On récupère l'identifiant de la vue, et en fonction de cet identifiant…

switch(v.getId()) {

case R.id.bouton1:

/\* Agir pour bouton 1 \*/

break;

case R.id.bouton2:

/\* Agir pour bouton 2 \*/

break;

}

}

**Par une classe anonyme**

L'inconvénient principal de la technique précédente est qu'elle peut très vite allonger les méthodes des listeners, ce qui fait qu'on s'y perd un peu s'il y a beaucoup d'éléments à gérer. C'est pourquoi il est préférable de passer par une classe anonyme dès qu'on a un nombre élevé d'éléments qui réagissent au même évènement.

Pour rappel, une classe anonyme est une classe interne qui dérive d'une superclasse ou implémente une interface, et dont on ne précise pas le nom. Par exemple pour créer une classe anonyme qui implémente View.OnTouchListener()

**widget.setTouchListener(new View.OnTouchListener() {**

//Contenu de ma classe et comme on implémente une interface, il y aura des méthodes à redéfinir

public boolean onTouch(View v, MotionEvent event)

}**);**

**Exemple**

public class AnonymousExampleActivity extends Activity {

private Button b1 = null;

public void onCreate(Bundle savedInstanceState) {

super.onCreate(savedInstanceState);

setContentView(R.layout.main);

b1 = (Button)findViewById(R.id.b1);

b1.setOnLongClickListener (new OnLongClickListener() {

public boolean onLongClick(View v) {

text1.setText("ddddddddddd");

return false;

}

});

}}

**Par un attribut**

C'est un dérivé de la méthode précédente : on implémente des classes anonymes dans des attributs de façon à pouvoir les utiliser dans plusieurs éléments graphiques différents qui auront la même réaction pour le même évènement. C'est la méthode conseillée dès qu'il y a, par exemple, plusieurs boutons qui utilisent le même code.

**Exemple**

public class Main extends Activity {

private OnClickListener b\_ecout = new View.OnClickListener() {

public void onClick(View v) {

// Réagir au clic

} };

Button b1 = null;

public void onCreate(Bundle savedInstanceState) {

super.onCreate(savedInstanceState);

setContentView(R.layout.main);

b1 = (Button) findViewById(R.id.bouton1);

b1.setOnClickListener(b\_ecout);

}}

**Exercice**

Mettre trois boutons et pour un clic sur chaque bouton, changer la couleur de l'arrière plan.

public void onClick(View v) {

// donner d'abord un identifiant à votre layout…

LinearLayout layout1= (LinearLayout) findViewById(R.id.L1);

switch(v.getId()) {

case R.id.bouton1:

layout1.setBackgroundColor(Color.*RED*);

break;

case R.id.bouton2:

layout1.setBackgroundColor(Color.*Green*);

break;

}}

**Les événements**

Tous les éléments d’interface (conteneurs et widgets) possèdent les méthodes suivantes :

– setOnClickListener(View.OnClickListener) associe un écouteur d'événements aux clics sur la vue.

– setOnLongClickListener(View.OnLongClickListener) associe un écouteur d'événements aux clics longs sur la vue.

– setOnKeyListener(View.OnKeyListener) associe un écouteur d'événements aux actions clavier sur la vue.

– setOnTouchListener(View.OnTouchListener) associe un écouteur d'événements aux touchés sur la vue.

**Evénements généraux**

|  |  |  |  |
| --- | --- | --- | --- |
| **Evénement** | **Listener** | **Méthode (pour lier la source au Listener)** | **Méthodes à redéfinir** |
| Clic | View.OnClickListener | setOnClickListener | onClick(View) |
| Long clic | View.OnLongClickListener | setOnLongClickListener | onLongClick(View) |
| Touché | View.OnTouchListener | setOnTouchListener | onTouch(View, MotionEvent) |
| Clavier | View.OnKeyListener | setOnKeyListener | onKey(View, int, KeyEvent) |

**EditText**

Ce composant est utilisé pour permettre à l'utilisateur d'écrire des textes. Il s'agit en fait d'un TextView éditable.

Il hérite de TextView, alors il peut prendre les mêmes attributs que TextView en XML et on peut utiliser avec les mêmes méthodes Java.

|  |  |  |  |
| --- | --- | --- | --- |
| **Evènement** | **Interface** | **Méthode de lien** | **Méthodes à redéfinir** |
| Modification | **TextWatcher** | addTextChangedListener | -beforeTextChanged(CharSequence, int, int, int)  -afterTextChanged(CharSequence, int, int, int)  • Texte  • Point de départ de la modification  • Nombre de cars remplacés  • Nombre de cars de remplacement  -onTextChanged (CharSequence, int, int, int) |

##### **Exemple en XML**

<EditText

android:layout\_width="fill\_parent"

android:layout\_height="wrap\_content"

android:**hint**="taper votre nom"

android:inputType="textMultiLine"

android:lines="5" />

Au lieu d'utiliser android:text, on utilise android:hint.

Le problème avec android:text est qu'il remplit l'EditText avec le texte demandé, alors qu'android:hint affiche juste un texte d'indication, qui n'est pas pris en compte par l'EditText en tant que valeur.

**Avec java**

EditText editText = new EditText(this);

editText.setHint("taper votre nom");

editText.setInputType(InputType.TYPE\_TEXT\_FLAG\_MULTI\_LINE);

editText.setLines(5);

En plus des propriétés standard de TextView, EditText possède de nombreuses autres propriétés dédiées à la construction des champs :

* android:autoText pour indiquer si le champ doit fournir une correction automatique de l’orthographe.
* android:capitalize pour demander que le champ mettra automatiquement en majuscule la première lettre de son contenu.
* android:digits pour indiquer que le champ n’acceptera que certains chiffres.
* android:singleLine pour indiquer si la saisie s’effectue sur une seule ou plusieurs lignes (i.e, <Enter> vous place-t-il sur le widget suivant ou ajoute une nouvelle ligne ?).

**Exercice : Copier le contenu d'un edit text dans un label**

1. suite à un click de bouton

e=(EditText)findViewById(R.id.*e1*);

e.setTextColor(Color.*RED*);

t1=(TextView)findViewById(R.id.*t1*);

t1.setText(e.getText()) ;

1. par implémentation de l'interface **TextWatcher**

**public** **void** onTextChanged(CharSequence arg0, **int** arg1, **int** arg2, **int** arg3){

**try** {

H\_TAX = Double.*parseDouble*(h\_tax.getText().toString());

TTC = H\_TAX \* (1+ *TVA*);

ttc.setText(String.*valueOf*(TTC));

} **catch** (NumberFormatException e) {

}}

**Les éléments à deux états**

Ils ont les mêmes paramètres que TextView auxquels vient s’ajouter la définition de l’état initial à l'aide de la propriété checked :

android:checked="b" où b vaut true ou false.

**Cases à cocher : checkBox**

La case à cocher classique peut être dans deux états : cochée ou décochée. Un clic sur la case permet d'inverser la valeur de son état.

<CheckBox

android:id="@+id/check"

android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content"

android:checked="true"

android:text="Cette case est : cochée" />

Dans le code Java :

* isChecked() : pour savoir si la case est cochée ;
* setChecked() : pour forcer la case dans l’état coché ou décoché ;
* toggle() : inverse l’état de la case.

**Exemple**

* Implementer l'interface OnCheckedChangeListener
* Appeler la méthode **setOnCheckedChangeListener**

public void **onCreate**(Bundle icicle) {

super.**onCreate**(icicle);

**setContentView**(R.layout.main);

cb=(CheckBox)**findViewById**(R.id.check);

cb.**setOnCheckedChangeListener**(this);

}

public void **onCheckedChanged**(CompoundButton buttonView, boolean isChecked) {

if (cb.isChecked()) {

cb.**setText**("Cette case est : cochee");

}

else {

cb.**setText**("Cette case est : decochee");

}}}

**Boutons radio**

Avec Android, on peut utiliser la classe “[**android.widget.RadioButton**](http://developer.android.com/reference/android/widget/RadioButton.html)” pour un radio button, et ces radio buttons sont souvent groupés par un [**android.widget.RadioGroup**](http://developer.android.com/reference/android/widget/RadioGroup.html).

**Quelques méthodes :**

* check() pour sélectionner un bouton radio à partir de son identifiant
* clearCheck() pour décocher tous les boutons du groupe ;
* getCheckedRadioButtonId() pour obtenir l’identifiant du bouton radio coché (cette méthode renvoie –1 si aucun bouton n’est coché).

**Exemple**

**<RadioGroup**

**android:id=*"@+id/radio1"***

**android:layout\_width=*"wrap\_content"***

**android:layout\_height=*"wrap\_content"***

**android:orientation=*"horizontal"* >**

**<RadioButton**

**android:id=*"@+id/b1"***

**android:layout\_width=*"wrap\_content"***

**android:layout\_height=*"wrap\_content"***

**android:checked=*"true"* />**

**<RadioButton**

**android:id=*"@+id/b2"***

**android:layout\_width=*"wrap\_content"***

**android:layout\_height=*"wrap\_content"* />**

**</RadioGroup>**

**Code java associé**

RadioGroup rd = new RadioGroup(this);

RadioButton rb1 = new RadioButton(this);

RadioButton rb2 = new RadioButton(this);

// On ajoute les boutons au RadioGroup

rd.addView(rb1, 0);

rd.addView(rb2, 1);

// On sélectionne le premier bouton

rd.check(0);

// On récupère l'identifiant du bouton coché

int id = rd.getCheckedRadioButtonId();

**Interaction : clic sur un bouton**

public void onClick(View v) {

int id = rd.getCheckedRadioButtonId();

b = (RadioButton) findViewById(id);

Toast.makeText(this, b.getText(), Toast.LENGTH\_SHORT).show();

}

rd.setOnCheckedChangeListener(new rd.OnCheckedChangeListener() {

public void onCheckedChanged(RadioGroup group, int checkedId) {

rb=(RadioButton)findViewById(id);

textView.setText("You Selected "+rb.getText());

}

});

**Propriétés utiles**

Parmi les propriétés les plus utiles de View, citons celles qui contrôlent la séquence de focus :

* android:nextFocusDown ;
* android:nextFocusLeft ;
* android:nextFocusRight ;
* android:nextFocusUp.

La propriété android:visibility contrôle la visibilité initiale du widget.

**Méthodes utiles**

* setEnabled() permet de basculer entre l’état actif et l’état inactif du widget,
* isEnabled() permet de tester si un widget est actif.

On utilise souvent ces deux méthodes pour désactiver certains widgets en fonction des choix effectués à l’aide de CheckBox ou de RadioButton.

* requestFocus() donne le focus à un widget
* isFocused() permet de tester s’il a le focus.

En utilisant les méthodes évoquées plus haut, on peut donc donner le focus à un widget précis après une opération de désactivation.

**ToggleButton **

android:disabledAlpha définit la transparence appliquée lorsque le bouton est inactif

android:textOff Pour définir le texte quand le bouton n'est pas allumé

android:textOn Pour définir le texte quand le bouton est allumé

Exemple

<ToggleButton

android:id=*"@+id/toggleButton1"*

android:layout\_width=*"wrap\_content"*

android:layout\_height=*"wrap\_content"*

android:textOn=*" on"*

android:textOff=*" off"*

android:checked*="true"* />

Les méthodes

* setChecked(boolean checked)
* isChecked
* setTextOn
* setTextOff
* setTextColor(Color.RED);
* setBackgroundColor(Color.BLACK);

**public** **void** onClick(View v) {

StringBuffer result = **new** StringBuffer();

result.append("toggleButton1 : ").append(toggleButton1.getText());

result.append("**\n**toggleButton2 : ").append(toggleButton2.getText());

Toast.makeText(**this**, result.toString(), Toast.LENGTH\_SHORT).show();

}

##### **Les listes :** ArrayAdapter

Les adaptateurs se chargent de fournir la liste des données d’un widget de sélection et de convertir les différents éléments en vues spécifiques pour qu'elles s’affichent dans ce widget de sélection.

Si l'activité est pilotée par une seule liste, il peut être préférable que cette activité soit une sous-classe de ListActivity plutôt que de la classe de base Activity. Si la vue principale est uniquement constituée de la liste, vous n’avez même pas besoin de fournir de layout.

ListActivity construira une zone de liste par défaut qui occupera tout l’écran.

On peut toutefois personnaliser cette présentation à condition d’identifier cette ListView par @android:id/list, afin que ListActivity sache que c'est la liste principale de l’activité.

Les constructeurs :

public ArrayAdapter (Context contexte, int id, T[] objects)

public ArrayAdapter (Context contexte, int id, List<T> objects).

* contexte : dans lequel notre adapter va fonctionner (généralement c'est l’instance de l’activité).
* id : l’identifiant de ressource de la vue qui contiendra un élément de la liste ( **android.R.layout.simple\_list\_item\_1**)
* objects : la liste ou le tableau des éléments à afficher.

**Exemple**

## Affichage avec ListActivity

Il faut faire attention à l'identifiant du composant ListView, il doit être : "@android:id/list".

On ne peut pas mettre un identifiant personnalisé comme on peut le faire pour les autres composants, il faut mettre celui indiqué pour que les mécanismes de gestion de vues (ceux présent dans ListActivity) fonctionnent.

**Exemple**

**Code main.xml**

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

<LinearLayout

…

<TextView

android:id="@+id/selection"

android:layout\_width="fill\_parent"

android:layout\_height="wrap\_content"/>

<ListView

android:id="@android:id/list"

android:layout\_width="fill\_parent"

android:layout\_height="fill\_parent"

android:drawSelectorOnTop="false"

/>

</LinearLayout>

**Code Java**

public class ListViewDemo extends ListActivity {

String[] items={"élément1", " élément2", " élément3", " élément4", " élément5"};

@Override

public void onCreate(Bundle icicle) {

super.onCreate(icicle);

**// ATTENTION: pas de setContentView**

ArrayAdapter<String> aa new ArrayAdapter<String>(this,

android.R.layout.simple\_list\_item\_1, items));

setListAdapter(**aa**);

} }

**Remarque**

Par défaut, un ArrayAdapter affichera pour chaque objet de la liste, le résultat de la méthode String toString() associée et l'insérera dans une TextView.

L'avantage d'utiliser une ListActivity par rapport à une Activity est dans le fait qu'il n'y a pas besoin de récupérer l'instance du composant ListView.

## Affichage sans ListActivity

Le widget classique d’Android pour les listes s’appelle ListView. Pour disposer d’une liste complètement fonctionnelle, il suffit d’inclure un objet ListView dans votre présentation, d’appeler setAdapter() pour fournir les données et les vues filles, puis d’attacher un écouteur *via* setOnItemSelectedListener() pour être prévenu de toute modification de la sélection.

**Code XML**

Créer une ListView dans le fichier xml :

**<ListView**

**android:id="@+id/list1"**

**android:layout\_height="wrap\_content"**

**android:layout\_width="match\_parent">**

**</ListView>**

**Code java**

ArrayAdapter(Context, type) : le second paramètre est un type prédéfini :

* + **android.R.layout.simple\_list\_item\_1** pour une liste à choix unique
  + **android.R.layout.simple\_list\_item\_multiple\_choice** pour une liste à choix multiple, (une case à cocher apparaît à coté de chaque élément de la liste)

public class ListViewAndroidExample extends Activity {

        ListView **L**;

 String[] items={"élément1", " élément2", " élément3", " élément4", " élément5"};

      public void onCreate(Bundle savedInstanceState) {

            super.onCreate(savedInstanceState);

setContentView(R.layout.**activity\_main**);

**L** = (ListView) findViewById(R.id.list1);

**ArrayAdapter<String> adapter = new ArrayAdapter<String>(this,**

**android.R.layout.simple\_list\_item\_1, items);**

**//**Le deuxième paramètre "android.R.layout.simple\_list\_item\_1" permet d'indiquer la //présentation utilisée pour les items de notre liste. Ici il s'agit d'une présentation simple pour les chaines de caractères incluse dans le SDK.

**// Assign adapter to ListView**

**L**.setAdapter(**adapter**);

        }

    }

**Interaction**

* onListItemClickListener
* onListItemSelectedListener

**Exemple**

public class MyListView extends Activity {

ListView list1;

private String items[] = { "Iphone", "Tutorials", "Gallery", "Android","item 1", "item 2", "item3", "item 4" };

protected void onCreate(Bundle savedInstanceState) {

super.onCreate(savedInstanceState);

setContentView(R.layout.main);

list1 = (ListView) findViewById(R.id.list);

list1.setAdapter(new ArrayAdapter<String>(this, android.R.layout.simple\_list\_item\_1, items));

list1.setOnItemClickListener(new OnItemClickListener(){

public void onItemClick(AdapterView<?> arg0, View view, int position, long id) {

Object o = list1.getItemAtPosition(position);

String ch = o.toString();

Toast.makeText(getApplicationContext(), "You have chosen the pen: " + " " + ch, Toast.LENGTH\_LONG).show();

}

}

);

}}

**Liste de choix (Spinner)**

Comme pour ListView, on fournit l’adaptateur pour les données et les vues filles *via* setAdapter() et on accroche un écouteur avec setOnItemSelectedListener().

Affiche le choix actuel et affiche un RadioGroup quand on clique dessus pour le changer.

**Propriétés :**

android:prompt : définit le titre de la fenêtre qui s'ouvre lorsque l'on fait un choix

android:entries="@array/maliste" définit le contenu de la liste à partir du contenu d'un

fichier xml placé dans res/values/ qui a la forme suivante :

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

<resources>

<string-array name=*"maliste"*>

<item>element1</item>

…

<item>element2</item>

</string-array>

</resources>

**Exemple**

Dans le dossier res/values, créer le fichier xml suivant :

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

<resources>

<string-array name=*"liste1"*>

<item>Mercure</item>

<item>soleil</item>

<item>Neptune</item>

</string-array>

</resources>

Dans le layout de l'application ajouter le spinner suivant :

<Spinner

android:id=*"@+id/spinner1"*

android:layout\_width=*"122dp"*

android:layout\_height=*"wrap\_content"*

android:entries=*"@array/maliste1"* />

ArrayAdapter<String> dataAdapter = new ArrayAdapter<String>(this, android.R.layout.simple\_spinner\_item, items);

spinner.setAdapter(dataAdapter);

**Interface**

**OnItemSelectedListener**

**public void onItemSelected(AdapterView<?> p, View view, int position, long id) {**

**// On selecting a spinner item**

**String item = p.getItemAtPosition(position).toString();**

**// Showing selected spinner item**

**Toast.makeText(parent.getContext(), "Selected: " + item, Toast.LENGTH\_LONG).show();**

**}**

public void onNothingSelected(AdapterView<?> arg0) {

// TODO Auto-generated method stub

}

**AutoCompleteTextView**

C’est une spécialisation de EditText pour apporter l’auto complétion.

**AutoCompleteTextView** est une sorte hybride de EditText et de Spinner.

Puisque c'est une sous-classe de EditText, on peut donc utiliser toutes les propriétés de cette dernière pour contrôler son aspect.

android:completionHint="texte" : texte affiché en titre du menu déroulant

android:completionThreshold : Pour définir le nombre de caractères à taper avant que la complétion n'entre en action.

android:dropDownHeight="unité" : définit la hauteur du menu déroulant, on peut aussi utiliser les constantes fill\_parent et wrap\_content

android:dropDownWidth="unité" : on peut aussi utiliser les constantes fill\_parent et

wrap\_content.

android:dropDownHorizontalOffset : définit le décalage horizontal du menu déroulant

android:dropDownVerticalOffset : définit le décalage vertical du menu déroulant

**Remarque**

**AutoCompleteTextView** ne permet pas d’utiliser les écouteurs de sélection. Il est donc préférable d’enregistrer un TextWatcher comme n’importe quel EditText pour être prévenu lorsque le texte a été modifié. Ce type d’événement est déclenché par une saisie manuelle ou par une sélection dans la liste des propositions.

Notre activité implémente l’interface TextWatcher, alors on doit redéfinir onTextChanged(), beforeTextChanged() et afterTextChanged().

**Exemple**

<TextView

android:id=*"@+id/selection"*

android:layout\_width=*"fill\_parent"*

android:layout\_height=*"wrap\_content"*

/>

<AutoCompleteTextView

android:id=*"@+id/edit"*

android:layout\_width=*"fill\_parent"*

android:layout\_height=*"wrap\_content"*

android:completionThreshold=*"1"*/>

</LinearLayout>

**Code Java:**

**public** **class** MainActivity **extends** Activity **implements** TextWatcher{

TextView selection;

AutoCompleteTextView edit;

String[] items={"lorem", "ipsum", "dolor", "dolar", "dodo", "consectetuer", "adipiscing", "elit", "porttitor", "sodales", "pellentesque", "augue", "purus"};

@Override

**public** **void** onCreate(Bundle icicle) {

**super**.onCreate(icicle);

setContentView(R.layout.*main*);

selection=(TextView)findViewById(R.id.*selection*);

edit=(AutoCompleteTextView)findViewById(R.id.*edit*);

**edit**.addTextChangedListener(**this**);

**edit**.setAdapter(**new** ArrayAdapter(**this**, android.R.layout.*simple\_dropdown\_item\_1line*, items));

}

**public** **void** onTextChanged(CharSequence s, **int** start, **int** before, **int** count) {

selection.setText(edit.getText());

}

**public** **void** beforeTextChanged(CharSequence s, **int** start, **int** count, **int** after) {

// imposée par l’interface, mais inutilisée ici

}

**public** **void** afterTextChanged(Editable s) {

// imposée par l’interface, mais inutilisée ici

}

}

**Gallery**

Utilisé pour faire des galeries d’images avec défilement horizontal, où chaque choix défile selon l’axe horizontal et où l’élément sélectionné est mis en surbrillance.

Du point de vue code, un objet Gallery fonctionne comme un Spinner ou un GridView.

**Propriétés**

**android:spacing** : nombre de pixels séparant les différents éléments de la liste.

**android:spinnerSelector** : précise ce qui indiquera une sélection – il peut s’agir d’une référence à un objet Drawable ou une valeur RGB de la forme #RRGGBB ou équivalente.

**android:drawSelectorOnTop** : indique si la barre de sélection (ou le Drawable) doit être dessinée avant (false) ou après (true) le dessin du fils sélectionné.

Si cette propriété vaut true, le sélecteur est suffisamment transparent pour que l’on puisse apercevoir le fils derrière lui ; sinon les utilisateurs ne pourront pas voir ce qu’ils ont choisi.

**android:animationDuration** : définit la durée de la transition (en ms) lorsque l'on passe d'un élément à l'autre.

**android:unselectedAlpha** : définit la transparence des éléments non sélectionnés.

Pour remplir une galerie il faut un Adapter (comme pour ListView) mais que l’on doit écrire par héritage de la classe BaseAdapter puis l’associer à la galerie par la méthode setAdapter

<Gallery android:id="@+id/magalerie"

android:layout\_width="fill\_parent"

android:layout\_height="wrap\_content"

android:unselectedAlpha="0.5"

/>

**Exemple**

<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">  
  
    <Gallery  
        android:id="@+id/gallery1"  
        android:layout\_width="fill\_parent"  
        android:layout\_height="wrap\_content" />  
      
     <ImageView  
        android:id="@+id/imageView1"  
        android:layout\_marginTop="100dp"  
        android:layout\_width="250dp"  
        android:layout\_gravity="center\_horizontal"  
        android:layout\_height="250dp"  
        android:src="@drawable/image1" />

 </LinearLayout>

**Code java**

**public** **class** MainActivity **extends** Activity {

**private** Integer[] mymageIds = {

R.drawable.*fr*,

R.drawable.*ic\_launcher*,

R.drawable.*ico\_ok*,

R.drawable.*sym\_keyboard\_ok*,

R.drawable.*us*,

};

**public** **void** onCreate(Bundle icicle) {

**super**.onCreate(icicle);

setContentView(R.layout.*main*);

Gallery gallery = (Gallery) findViewById(R.id.*gallery1*);

gallery.setSpacing(1);

GalleryImageAdapter **aa =new** GalleryImageAdapter(**this,** mymageIds)

gallery.setAdapter(aa);

} }