Université d'Alger 1 Benyoucef Ben Kheda
Faculté des Sciences
Département MI

L3 - SI

Développement d Applications Mobiles

DR. AICHA BOUTORH

2017/2018

Contenu de la matière

1) Introduction
2) Les systèmes d'exploitation mobiles (iOS, WindowsPhone, Android)
3) Architecture de la plate-forme Android
4) Installation de l'environnement de développement
5) Structure et composants des applications mobiles
6) Construction de l'interface utilisateur
7) Utilisation des ressources : XML, images, fichiers, ..
8) Programmation mobile avec Android (SDK Android, XML et JSON, Elément d'interface, SQLite, Connectivité, ….)
9) Développement d'une application simple
10) Déploiement d'une application mobile.

Fiche de Contact et Modalité d'Evaluation

- **Coefficient : 3**
- **Crédits : 5**
- **1 Cours, 1 TD, 1 TP/ Semaine**
- **Activités Pratiques (TP)**
- **Contrôle continu**
- **Examen final**

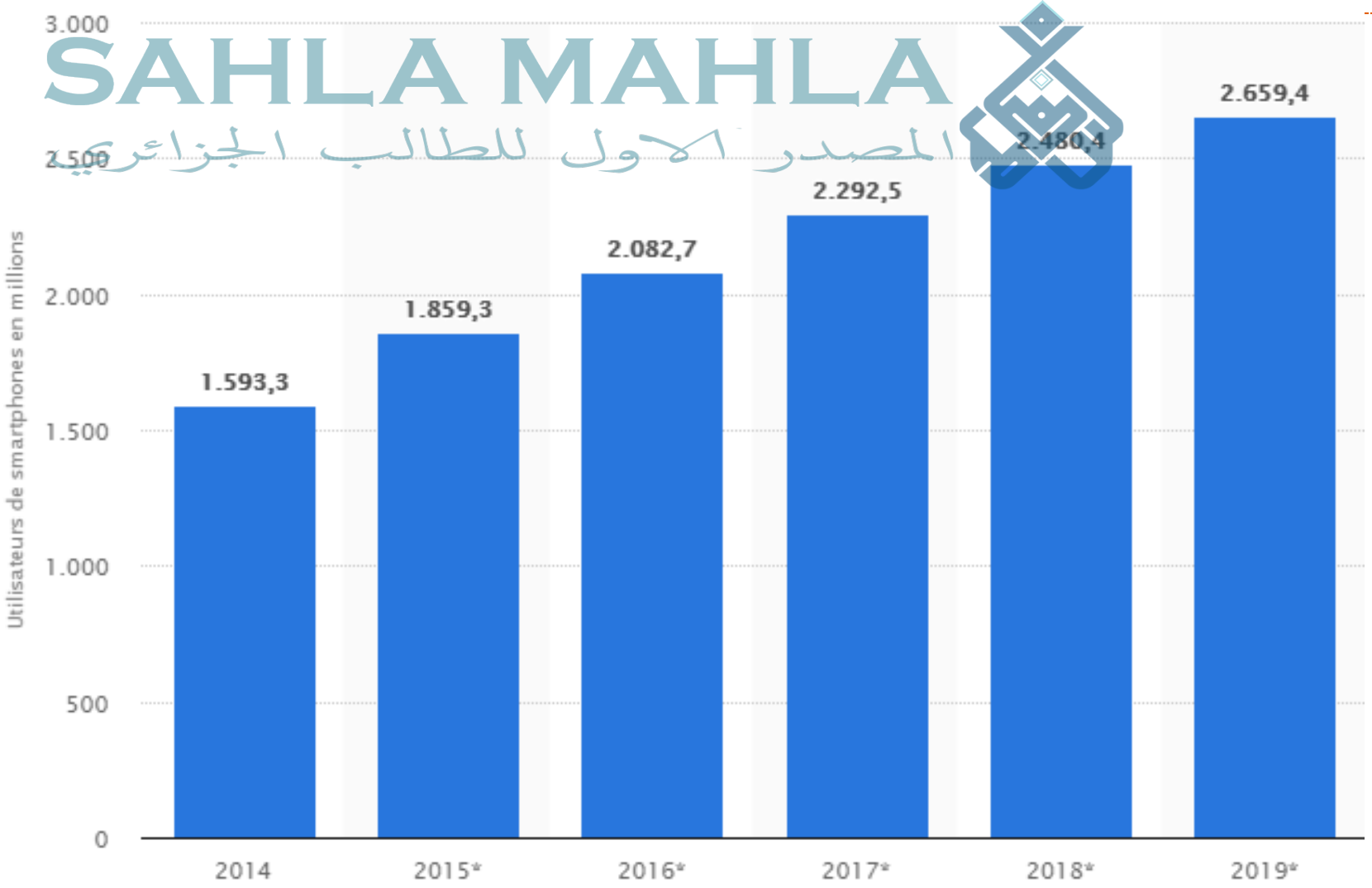
Introduction

Un marché en explosion

- **Vente de "terminaux mobiles" évolues**
- En 2012, 5 milliards de téléphones mobiles étaient en circulation dont 1 milliard de Smartphones
- L'explosion des ventes doit beaucoup a l'énorme succès de l'iPhone d'Apple en 2007 puis de l'iPad
- L'internet des objets (IoT) va faire exploser les chiffres !
- Des utilisateurs de plus en plus accros

- **Vente d'applications associées**
- 102 milliards de téléchargements ont eu lieu en 2013, soit 194 app téléchargées par minute
- C'est la nouvelle ruée vers l'or pour les développeurs !

Nombre d'utilisateurs de smartphone dans le monde entre 2014 et 2019 (en millions)



https://fr.statista.com/statistiques/574542/utilisateurs-de-smartphone-dans-le-monde--2019/

En 2016, 90 milliards d'apps ont été téléchargées

https://www.igen.fr/app-store/2017/01/en-2016-90-milliards-dapps-ont-ete-telechargees-98494

Qu'est-ce qu'un Smartphone?

- Un **téléphone intelligent** est un ordinateur personnel portatif doté d'un **système d'exploitation mobile** et d'une connexion réseau cellulaire mobile à large bande intégrée pour la communication de données vocales, SMS et Internet; La plupart des smartphones, ou tous, prennent également en charge le **Wi-Fi**.

- Les smartphones sont généralement **pocket-sized**, par opposition aux tablettes, qui sont beaucoup plus grandes. Ils sont capables d'exécuter une variété de composants logiciels, appelés "**apps**".

- La plupart des applications de base (*calendrier d'événements, appareil photo, navigateur Web, etc.*) sont pré-installées avec le système, tandis que d'autres peuvent être téléchargées à partir de **Google Play Store** ou d'**Apple App Store.**

Smartphone, Phablette, Tablette ?

Smartphones
5-inch and less

Phablets
Between 5-7 inches

Tablets
7-inches and above

Qu'est-ce qu'une Mobile App?

- Une application mobile est un programme informatique conçu pour fonctionner sur un appareil mobile tel qu'un téléphone / une tablette ou une montre.

- Le terme "**app**" est un raccourci du terme "**application logicielle**". Il est devenu très populaire, et en 2010 a été répertorié comme «**Word of the Year**» par l'*American Dialect Society*. En 2009, le chroniqueur de technologie **David Pogue** a déclaré que les nouveaux téléphones intelligents pourraient être surnommés «**Mobile App**» pour les distinguer des smartphones moins sophistiqués.

MobileApp *vs* WebApp

	Application mobile (native)	Application web
Portabilité	Développement spécifique a chaque plateforme	Navigateur Web
Développement /coût	Nécessite un SDK + connaissance d'un langage spécifique	Langage du Web (HTML / JS / CSS /PHP…)
Mises à jour	Soumission a un magasin d'applications et éventuelle validation + retéléchargement par le Client	Mise a jour rapide en mettant a jour tout simplement les fichiers sur le serveur Web
Disponibilité	Mode online et offline	Nécessite obligatoirement une connexion internet
Fonctionnalités	Utilise toutes les fonctionnalites du mobile (GPS, voix, notifications, contacts…)	Limitées aux possibilités du navigateur

HybridApp : le modele hybride

- Encapsulation d'une WebApp dans une MobileApp

SmartPhone

- Les Smartphones et les tablettes peuvent remplacer dans différentes situations, l'ordinateur portable, ils sont considérés comme des petits ordinateurs.
- Ils deviennent de plus en plus fréquents et cela grâce aux différents systèmes d'exploitation mobile tel que :

- **iOS**
- **Windows Phone,**
- **Android**

qui ne cessent de se développer ….

Les systèmes d'Exploitation Mobiles « Mobile OS »

Qu’est ce qu’un Mobile OS?

- **Mobile Operating System** (or **Mobile OS**)
- Un système d'exploitation mobile est un système d'exploitation pour téléphones, tablettes, smartwatches ou autres appareils mobiles. Les ordinateurs portables typiques sont «mobiles», les systèmes d'exploitation habituellement utilisés sur eux ne sont pas considérés comme mobiles, car ils ont été conçus à l'origine pour les ordinateurs de bureau qui n'ont pas besoin de fonctionnalités mobiles spécifiques. Cette distinction devient floue dans certains systèmes d'exploitation plus récents qui sont des hybrides fabriqués pour les deux utilisations.
- Un Mobile OS s’agit d’un système d’exploitation qui exploite un appareil mobile (*smartphone, tablette,* ...). Une plate-forme qui contrôle toutes les opérations et fonctionnalité de base du téléphone mobile comme option d’écran tactile, cellulaires, Bluetooth, Wifi, appareil photo, lecteur de musique et d’autres fonctionnalités. **Exemple:** iOS, Windows Phone, Android, BlackBerryOS, Symbian, ...

Mobile OS

- Avec le nombre de Smartphones disponibles sur le marché, ça devient difficile de choisir un mobile.

- Certains Smartphones sont mieux en termes de vitesse de processeur tandis que d'autres sont mieux en termes de leur appareil photo.

- Toutefois, le **Mobile OS** fait la différence. Et donc la guerre des mobiles passe essentiellement par leurs systèmes d'exploitation

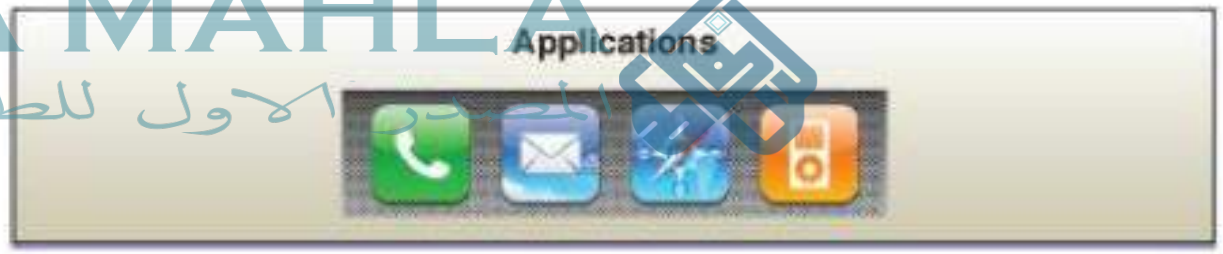
Mobile OS : développements

Plateforme	Programmation	IDE recommandé(s)
Windows Phone	VB.Net, C#	Visual studio .Net
iOS	Objective-C, Swift	X-CODE
Blackberry OS	Java	MDS Studio
Java ME	Java	EclipseME (CLDC, MIDP)
Android	Java, code natif C++	Android Studio / Eclipse + plug-in ADT
Palm WebOS	JavaScript, C/C++	Eclipse + webOS plug-in
Symbian OS	C++	Performance
Brew MP	C++	Visual Studio + plug-in
Bada	C++	badaIDE
Sailfish OS (MeeGo)	Qt C++	QtCreator
Firefox OS (B2G)	HTML5/CSS3/JavaScript	Notepad++ et Firefox OS Simulator
Ubuntu Mobile	C/C++, JavaScript	Qt Creator
Tizen	HTML5/JavaScript, code natif C++	Eclipse + plug-in Tizen

iOS (iPhoneOperating System)

- iOS est un système d'exploitation mobile créé et développé par Apple Inc. exclusivement pour son matériel. C'est le système d'exploitation qui alimente actuellement de nombreux appareils mobiles de la société, notamment l'**iPhone**, l'**iPad**, l'**iPod Touch** et **l'Apple TV**.
- Initialement dévoilé en 2007 pour l'iPhone, iOS a été étendu pour prendre en charge d'autres appareils Apple tels que l'**iPod Touch** (sept 2007) et l'**iPad** (jan 2010). En janvier 2017, l'App Store d'Apple contient plus de **2,2 millions** d'applications iOS, dont 1 million sont natives pour iPad.
- iOS Disponible en 34 langues dans le monde
- Il est programmé en C, C ++, Objective-C.
- iOS est surveillé très étroitement par apple, ce qui signifie que les utilisateurs ne peuvent pas apporter des modifications.
- Apple n'acquiert pas les licences iOS pour l'installation sur du matériel non Apple.

Architecture de iOS



Contient les frameworks clés pour la construction d'une application iOS

Permet de gérer le multimédia sur l'appareil mobile (les technologies graphiques, audio et vidéo).

Contient les interfaces fondamentales pour iOS, pour accéder aux fichiers, aux sockets réseau, ...

Gère la mémoire virtuelle, les threads, le système de fichiers, le réseau, la communication interprocessus ...

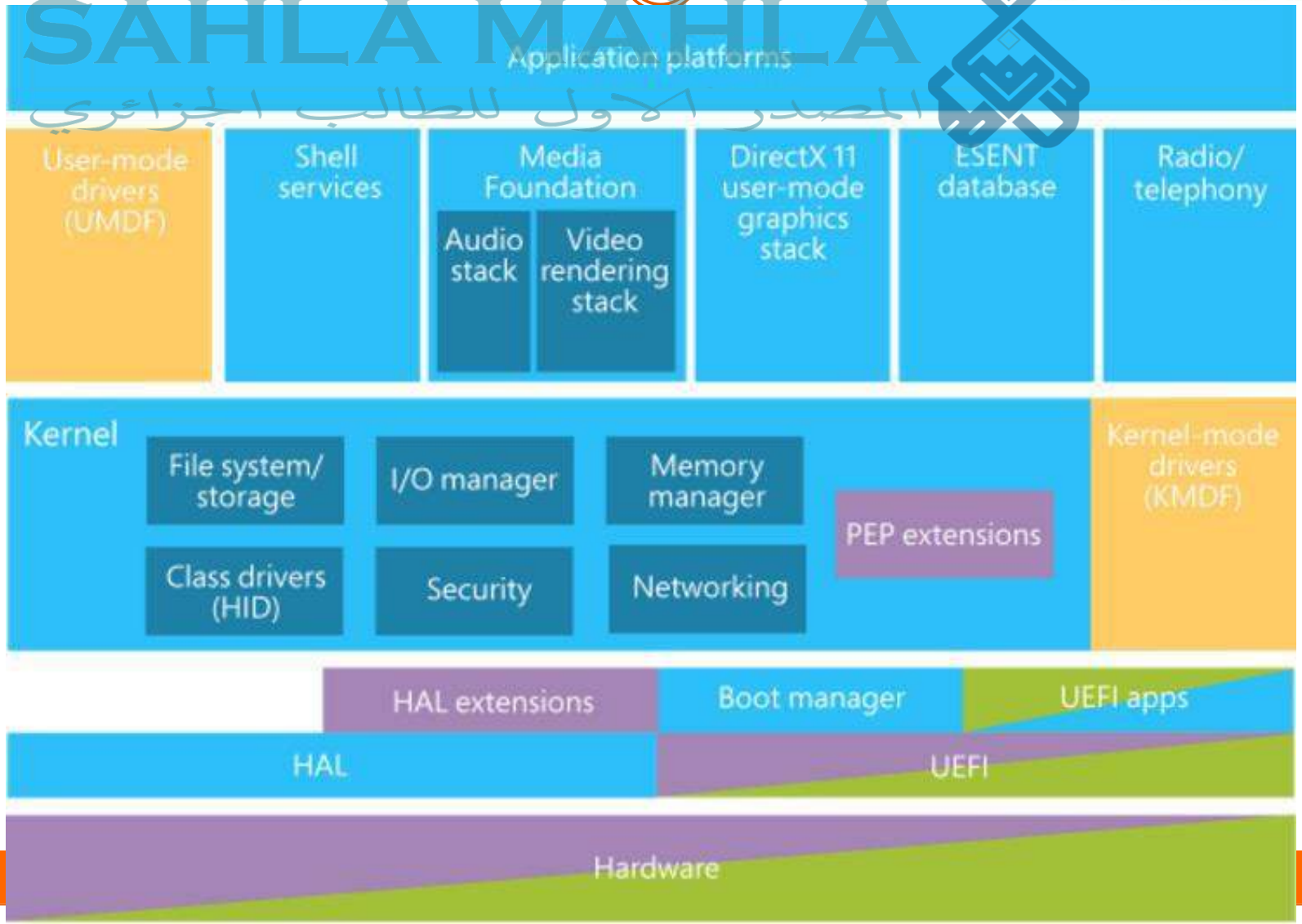
Windows Phone

- Windows Phone (WP) est une famille de systèmes d'exploitation mobiles développés par **Microsoft** pour les smartphones en remplacement de **Windows Mobile**.
- Utilisé dans des dispositifs faites par **Nokia**, **Samsung** et **HTC**
- Il a été lancé en octobre 2010 avec **Windows Phone 7**. **Windows Phone 8** est la dernière version publique du système d'exploitation, publiée le 14 avril 2014.
- WP ne trouvaient pas beaucoup de place sur le marché.

- ***Environnement de développement:***
- Visual Studio IDE + Windows Phone SDK
- Langages : les Application Windows mobile peuvent être développées en:
  - ***XAML:*** Langage de balise basé sur du XML Utilisé pour définir les interfaces graphiques des applications
  - ***C# :***Langage Orienté Objet très proche de la syntaxe Java Développé et maintenu par Microsoft(Support aussi VB, C++)

Architecture de

Windows Phone



Android

- Android est un système d'exploitation mobile développé par **Google**, basé sur une version modifiée du **noyau Linux** et d'autres logiciels open source et conçu principalement pour les appareils mobiles à écran tactile tels que les smartphones et les tablettes. Il a été annoncé en 2007 , il est devenu une plateforme ouverte en 2008.

- C'est un **OS gratuit et complètement ouvert**. C'est-à-dire que le code source et les API sont ouvertes. Ainsi, les développeurs obtiennent la permission d'intégrer, d'agrandir et de replacer les composants existants.

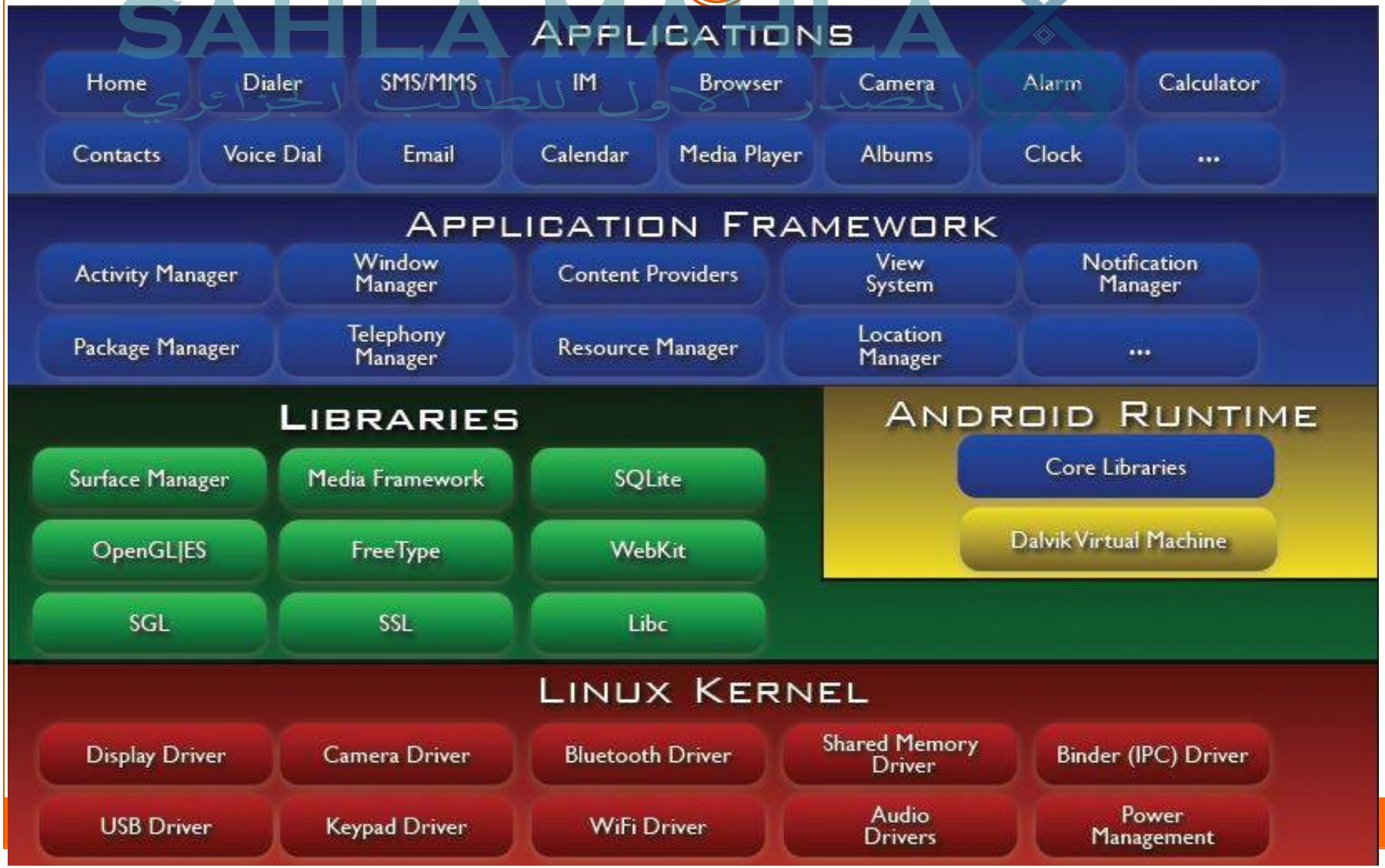
- Android est le système d'exploitation **le plus vendu** sur les smartphones depuis **2011** et sur tablettes depuis 2013. En mai 2017, il compte plus de **deux milliards** d'utilisateurs actifs par mois, soit la plus grande base installée de tous les systèmes d'exploitation. fonctions de magasin de plus de **3,5 millions d'applications**

Android

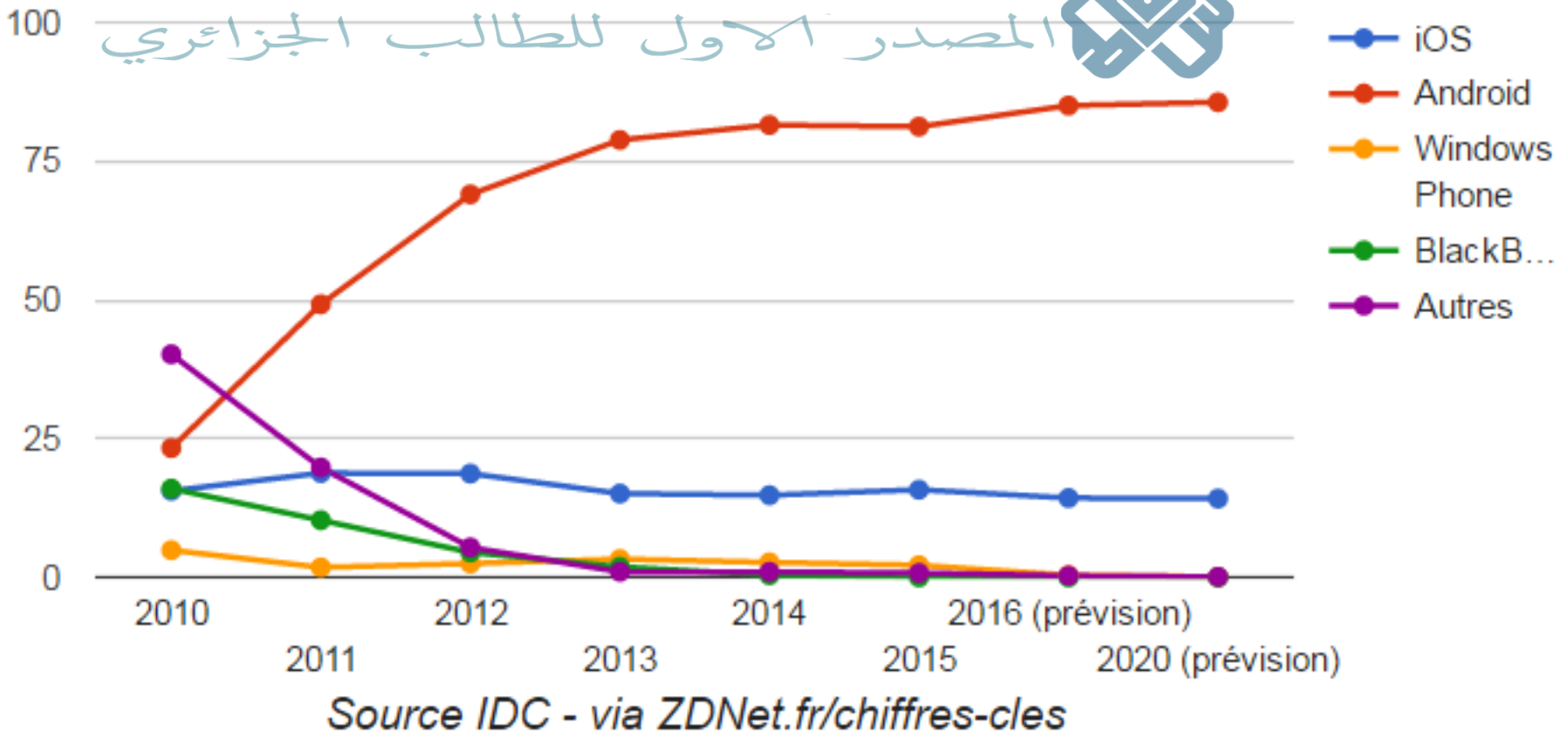
- ***Environnement de développement:***

- **AndroidStudio**, Eclipse
- **Android SDK Manager**
- AVD Manager
- Langages : les Applications Mobiles peuvent être développées en:
- ***XML:*** *Utilisé pour créer des interfaces graphiques des applications*
- ***JAVA:*** *Langage Orienté Objet*

Architecture Android



Mobile OS



Source IDC - via ZDNet.fr/chiffres-cles

[Ref]

Université d'Alger 1 Benyoucef Ben Kheda
Faculté des Sciences
Département MI

L3 - SI

Développement d Applications Mobiles

DR. AICHA BOUTORH

2017/2018

La Plate-Forme Android

Qu'est ce qu'Android?

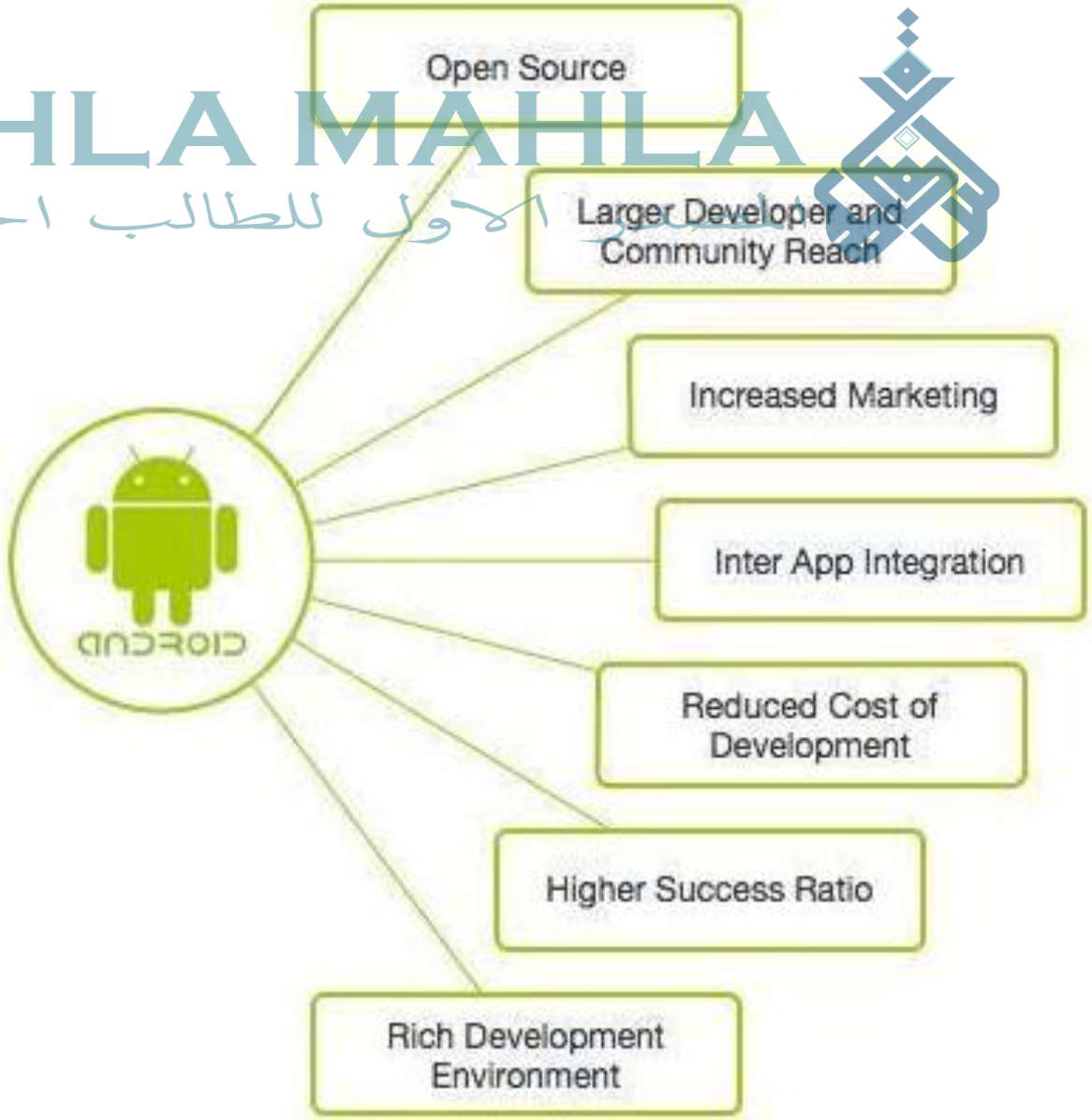
- Android est un système d'exploitation Open Source et basé sur Linux pour les appareils mobiles tels que les smartphones et les tablettes. Android a été développé par l'***Open Handset Alliance***, dirigée par ***Google***, et d'autres entreprises.

- Plateforme pour les appareils mobiles
- Gratuite
- Open source
- Flexible

➢ **Android inclue :**

- Un système d'exploitation basé sur Linux.
- Des applications basiques (téléphones, contacts, …).
- Un ensemble d'API avancées
- SDK basé sur un sous-ensemble de JAVA (existe aussi dans d'autres langages).

Pourquoi Android?



Qu'est-ce que “API”

- **Application programming interface**
  - En programmation informatique, une interface de programmation d'application (API) est un ensemble de définitions de sous-programmes, de protocoles et d'outils pour la construction de logiciels d'application.
  - En termes généraux, il s'agit d'un ensemble de méthodes de communication clairement définies entre différents composants logiciels.
  - Une bonne API facilite le développement d'un programme informatique en fournissant tous les blocs de construction, qui sont ensuite assemblés par le programmeur.

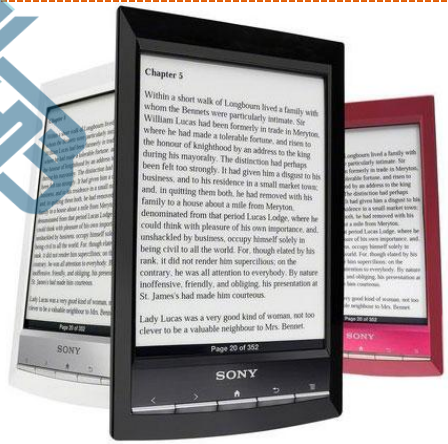
Android OS

- Le système d'exploitation Android est actuellement le système d'exploitation le plus populaire au monde. **Le système d'exploitation Android fonctionne sur :**

Smartwatches "AndroidWear"

HD or UHD Smartphones

Touchscreen Tablets



e-book readers

Game Consoles

Smart Glasses

Auto Dashboards

Smart TV "AndroidTV"

Si vous souhaitez que vos applications s'exécutent partout, Android est la solution optimale

- Comme il y a des milliards d'appareils électroniques Android possédés par des milliards de personnes partout dans le monde, Il va de soi que développer de formidables applications Android pour tous ces gens pourrait être une entreprise extrêmement rentable, en supposant que vous avez le bon concept et le bon design.

L'histoire de l'OS Android: une croissance impressionnante

- L'OS Android a été créé à l'origine par Andy Rubin pour être un OS pour les téléphones mobiles.
- En juillet 2005, Google a racheté **Android** et fait d'Andy Rubin le vice-président des plateformes mobiles pour Google, où il est resté jusqu'en novembre 2014.
- Beaucoup pensent que cette acquisition d'Android OS par Google répondait en grande partie à l'apparition de l'***iPhone d'Apple***.
- Cependant, il y avait assez d'autres grands acteurs, tels que ***RIM Blackberry***, ***Nokia Symbian*** **et** ***Microsoft Windows Mobile***, que c'était une décision d'affaires avisée pour Google d'acheter le talent d'ingénierie d'Android Incorporé avec sa propriété intellectuelle **Android OS**.
- Cela a permis à Google d'insérer leur entreprise de moteur de recherche Internet dans le marché mobile émergent, que beaucoup appellent maintenant **Internet 2.0**.

Versions d'Android

Cupcake
Andriod 1.5

Donut
Android 1.6

Eclair
Android 2.0/2.1

FROYO
2.2

Gingerbread
Android 2.3/2.3.7

Honeycomb
3.0

Ice Cream Sandwich
4.0

Jellybean
Android 4.1-4.3

KitKat
Android 4.4

Android 5.0, Lollipop

android
v6.0 Marshmallow

Android 7.0 Nougat

Niveau de l'API

- Niveau API est une valeur entière qui identifie de manière unique la révision de l'API de structure offerte par une version de la plate-forme Android.

Platform Version	API Level	VERSION_CODE
Android 6.0	23	MARSHMALLOW
Android 5.1	22	LOLLIPOP_MR1
Android 5.0	21	LOLLIPOP
Android 4.4W	20	KITKAT_WATCH
Android 4.4	19	KITKAT
Android 4.3	18	JELLY_BEAN_MR2
Android 4.2, 4.2.2	17	JELLY_BEAN_MR1
Android 4.1, 4.1.1	16	JELLY_BEAN

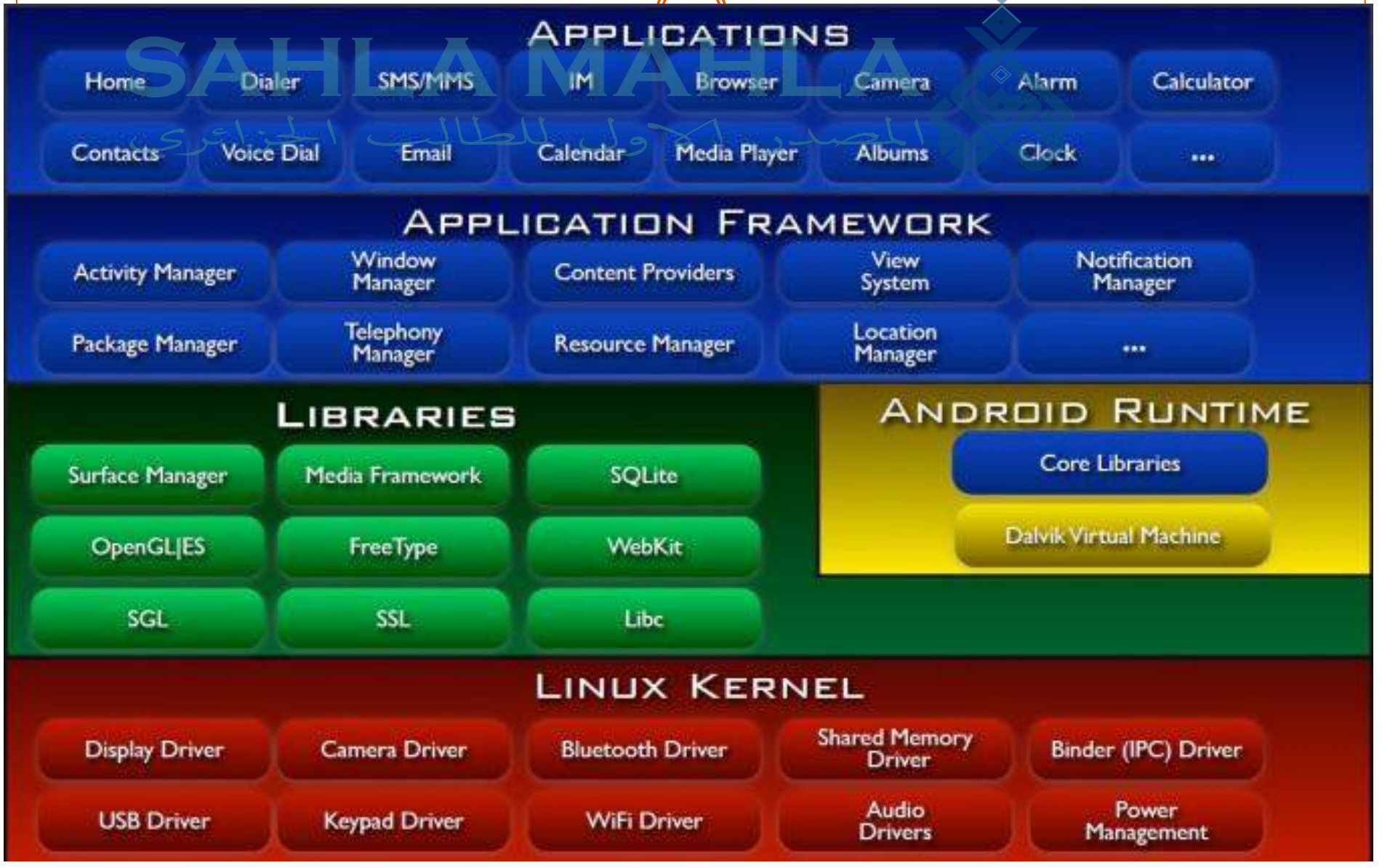
Niveau de l'API

Platform Version	API Level	VERSION_CODE
Android 4.0.3, 4.0.4	15	ICE_CREAM_SANDWICH_MR1
Android 4.0, 4.0.1, 4.0.2	14	ICE_CREAM_SANDWICH
Android 3.2	13	HONEYCOMB_MR2
Android 3.1.x	12	HONEYCOMB_MR1
Android 3.0.x	11	HONEYCOMB
Android 2.3.4 Android 2.3.3	10	GINGERBREAD_MR1
Android 2.3.2 Android 2.3.1 Android 2.3	9	GINGERBREAD
Android 2.2.x	8	FROYO

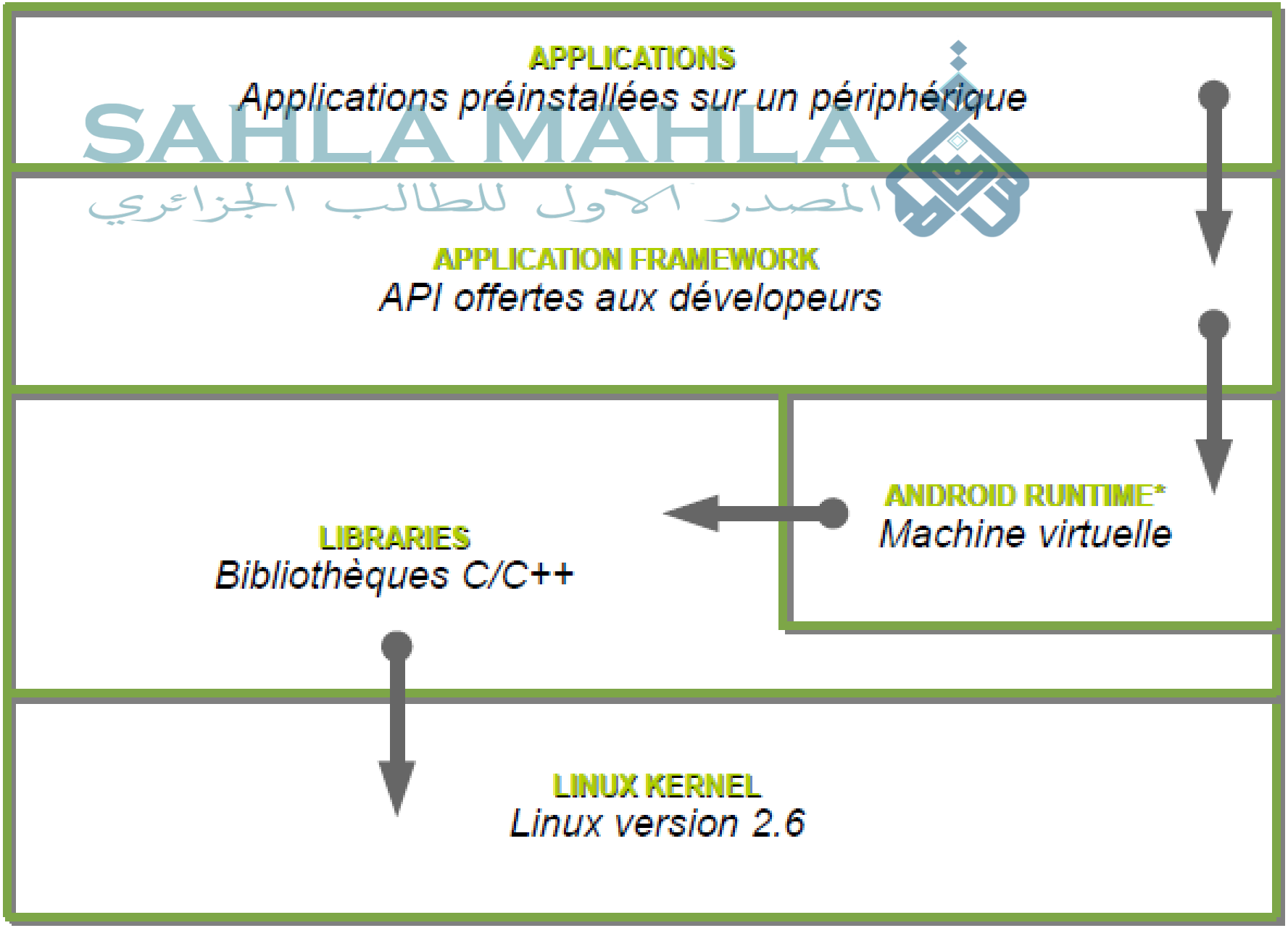
Niveau de l'API

Platform Version	API Level	VERSION_CODE
Android 2.1.x	7	ECLAIR_MR1
Android 2.0.1	6	ECLAIR_0_1
Android 2.0	5	ECLAIR
Android 1.6	4	DONUT
Android 1.5	3	CUPCAKE
Android 1.1	2	BASE_1_1
Android 1.0	1	BASE

Architecture logicielle



Couches logicielles



Environnement de développement pour Android

(AndroidStudio, SDK Android, AVD Manager)

AndroidStudio

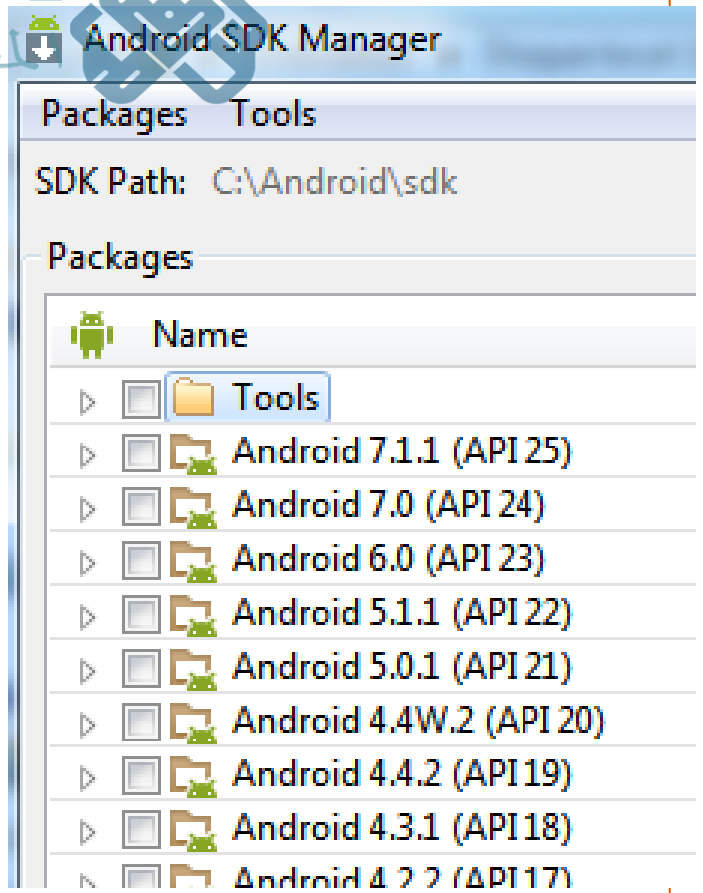
- L'IDE officiel pour Android
- Il fournit les outils pour créer des applications sur chaque type d'appareil Android.
- Disponible gratuitement sur le lien:
- **https://developer.android.com/studio/index.html**

- Langage de développement : Java, XML

SDK Android
(Software Development Kit)

- est un ensemble d'outils de développement logiciel destinés aux développeurs permettant la création d'un logiciel sur la plateforme Android, pour enrichir les applications avec des fonctionnalités avancées, des publicités, des notifications et plus, la plupart des développeurs d'applications mettent en œuvre des kits de développement de logiciels spécifiques.

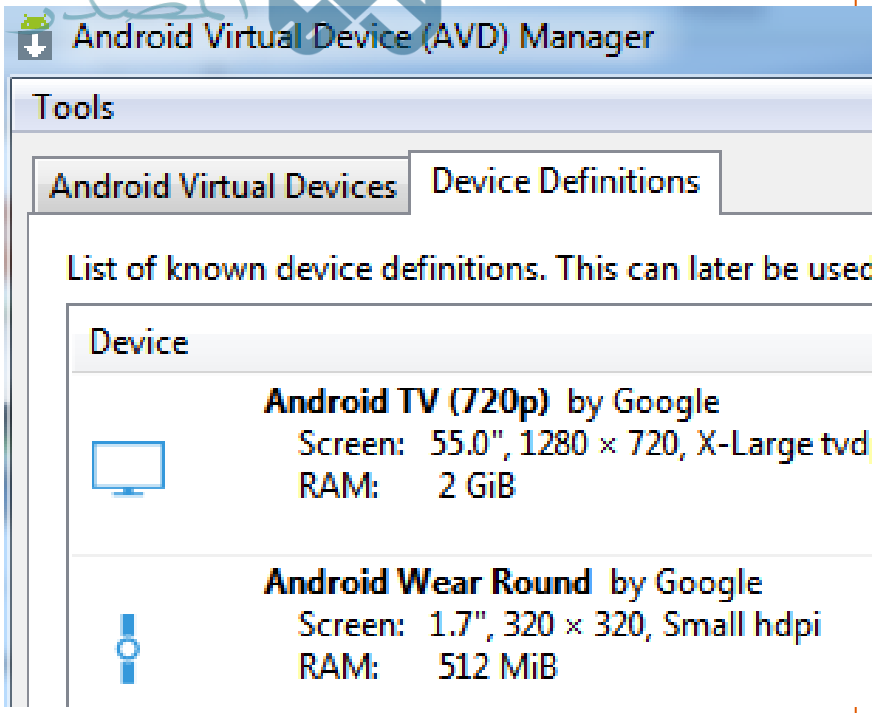
- SDK d'Android fournit :
  - Les librairies Java d'Android
  - Un émulateur pour tester les applications
  - Des images du système Android



Android Virtual Device (AVD) Manager

- Un AVD est une configuration d'émulateur qui vous permet de modéliser un périphérique réel en définissant les options matérielles et logicielles à émuler par l'émulateur Android.
  Le gestionnaire AVD fournit une interface utilisateur graphique dans laquelle vous pouvez créer et gérer des périphériques virtuels Android (AVD), requis par l'émulateur Android.

- L'utilisation d'un émulateur nous évite d'avoir à charger à chaque fois l'application dans un appareil pour la tester.



Installation de l'IDE

1. Télécharger et installer le JDK 8 (*Java Development Kit*)

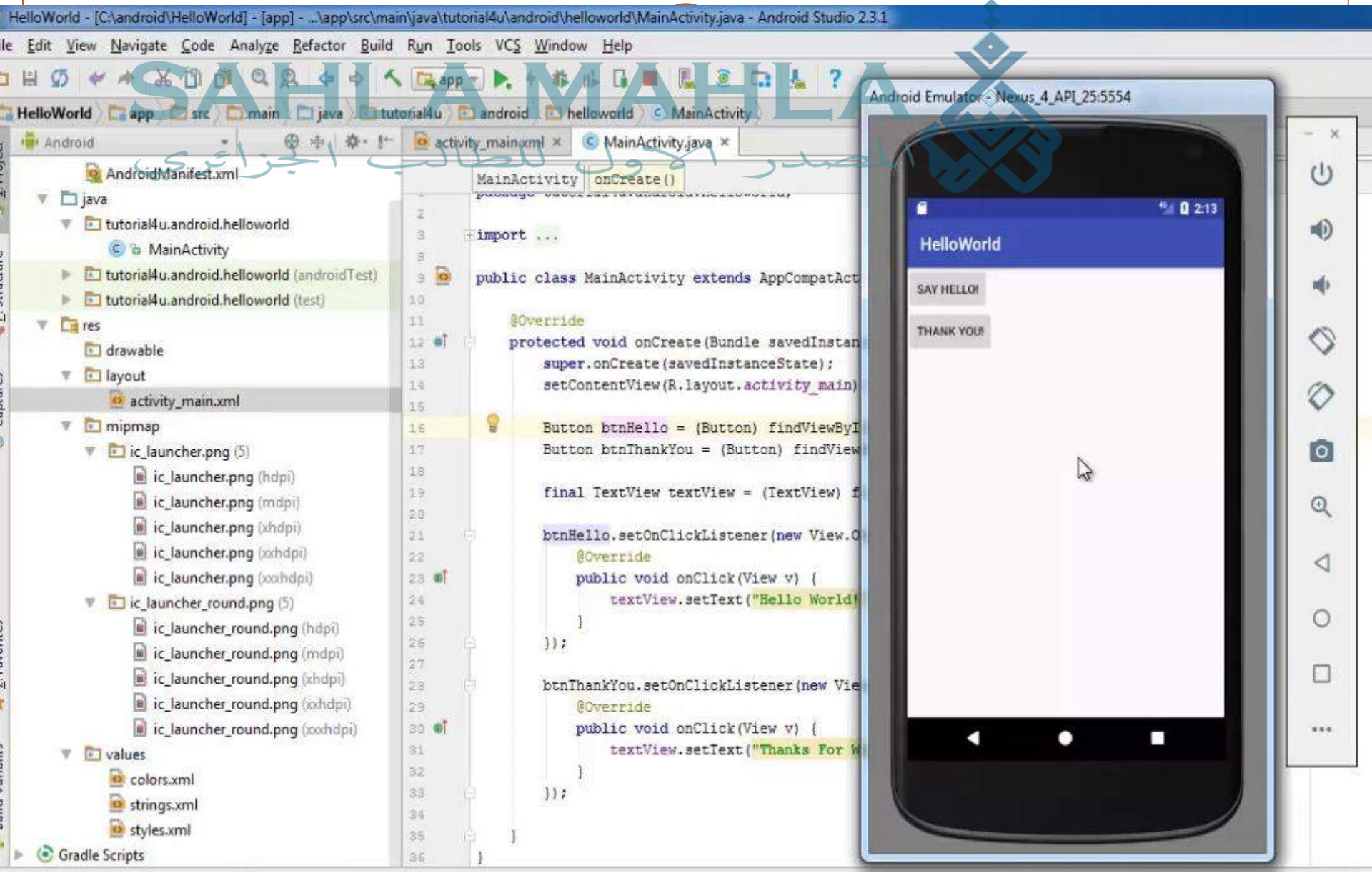
http://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html

2. Téléchargez 'AndroidStudio. Il contient l'environnement de développement, le SDK (*Software DevelopmentKit*) Androidavec la dernière version de la plateforme, ainsi qu'un émulateur. https://developer.android.com/studio/index.html

3. Lancez l'exécutable pour démarrer l'installation et suivez le wizard

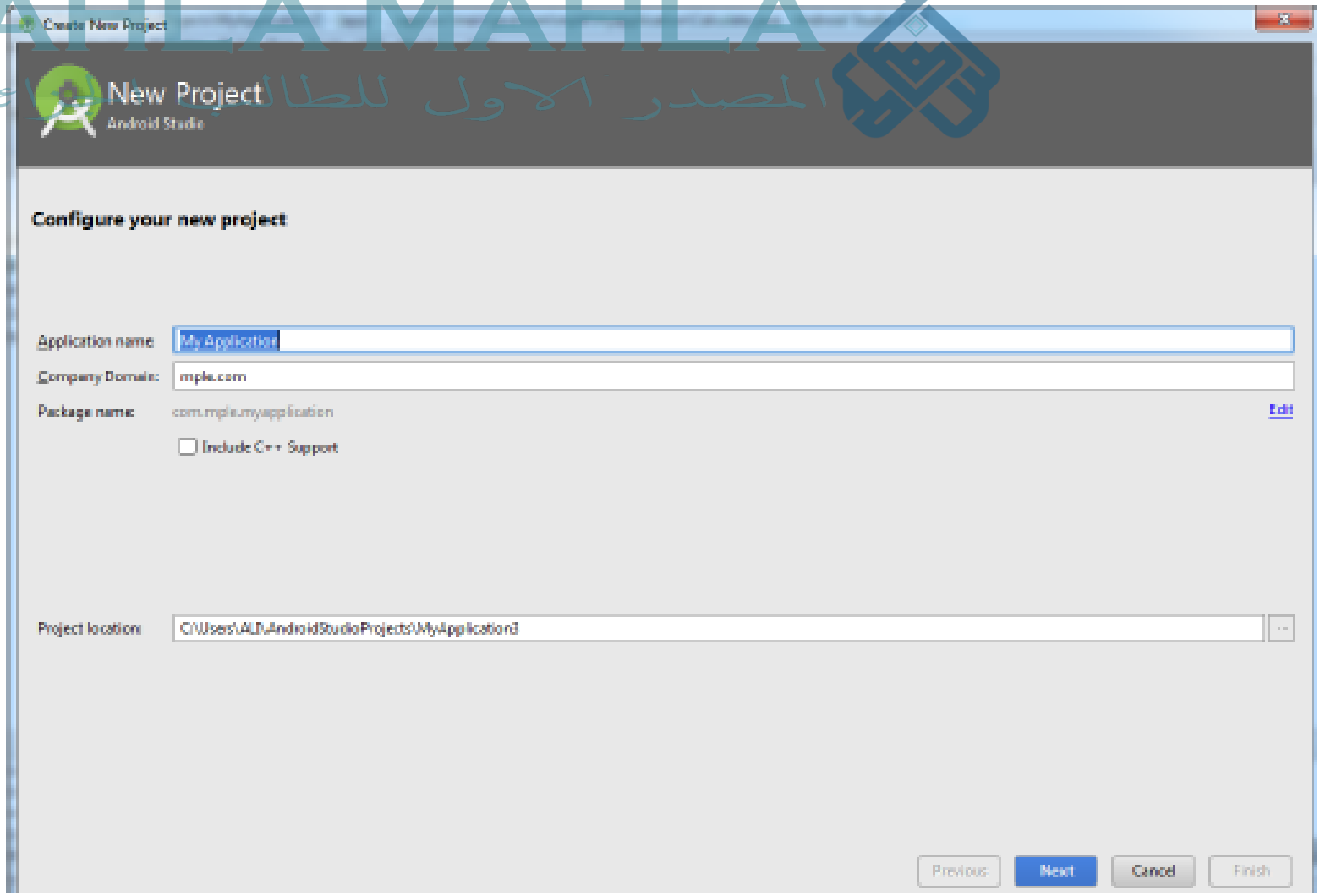
4. Installer AndroidStudio

Première Application Android

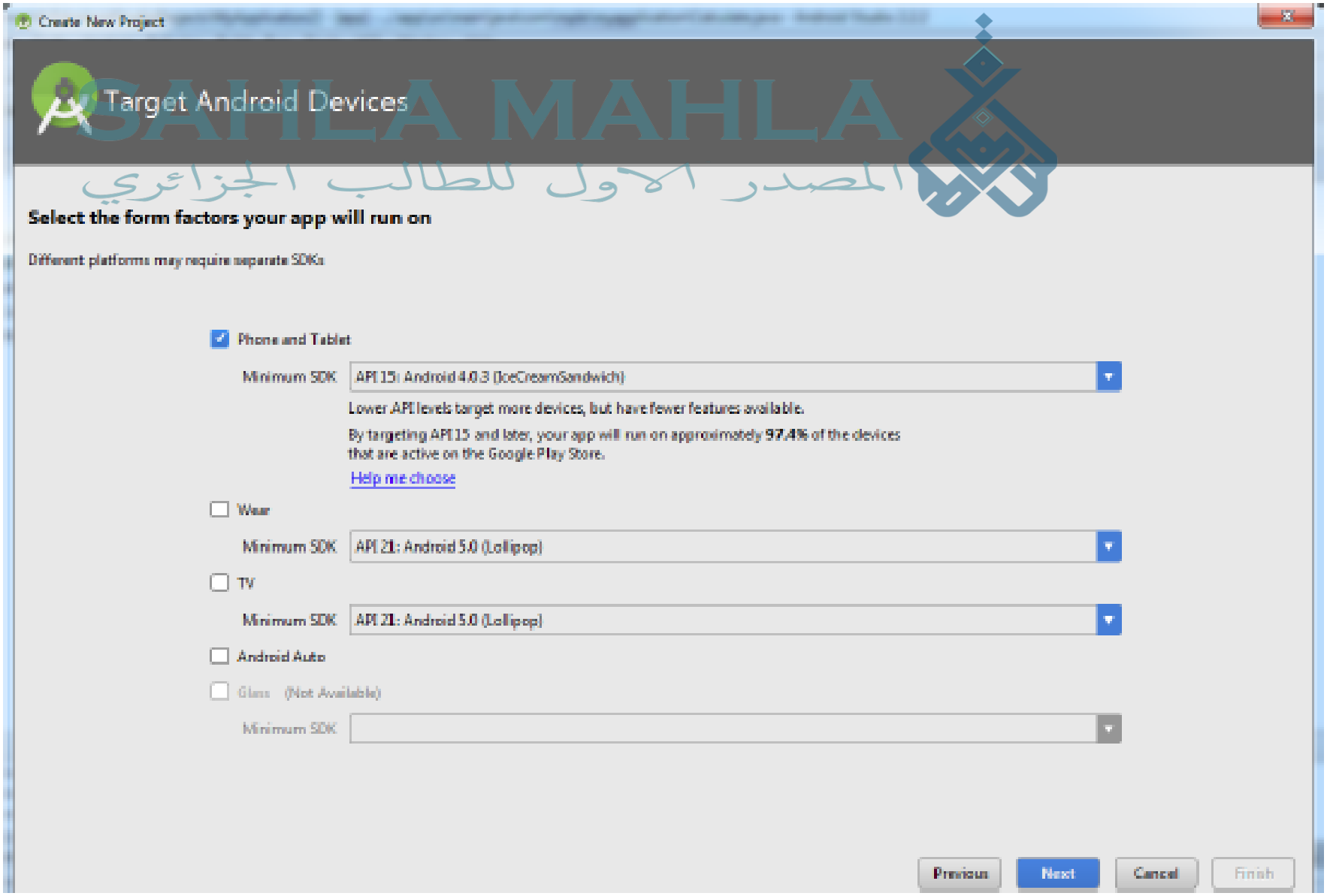


Création d'un nouveau projet:

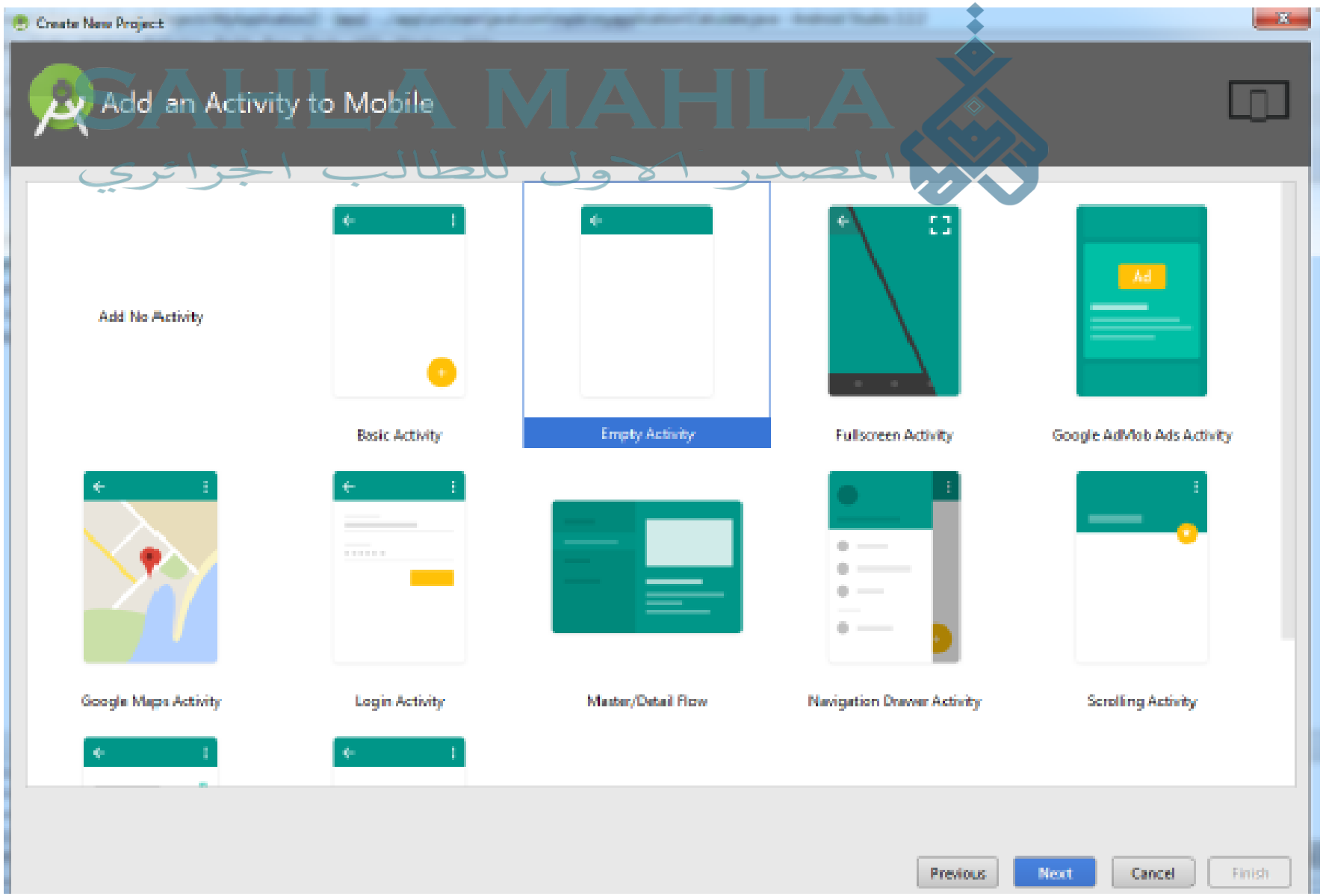
File/New/New Project



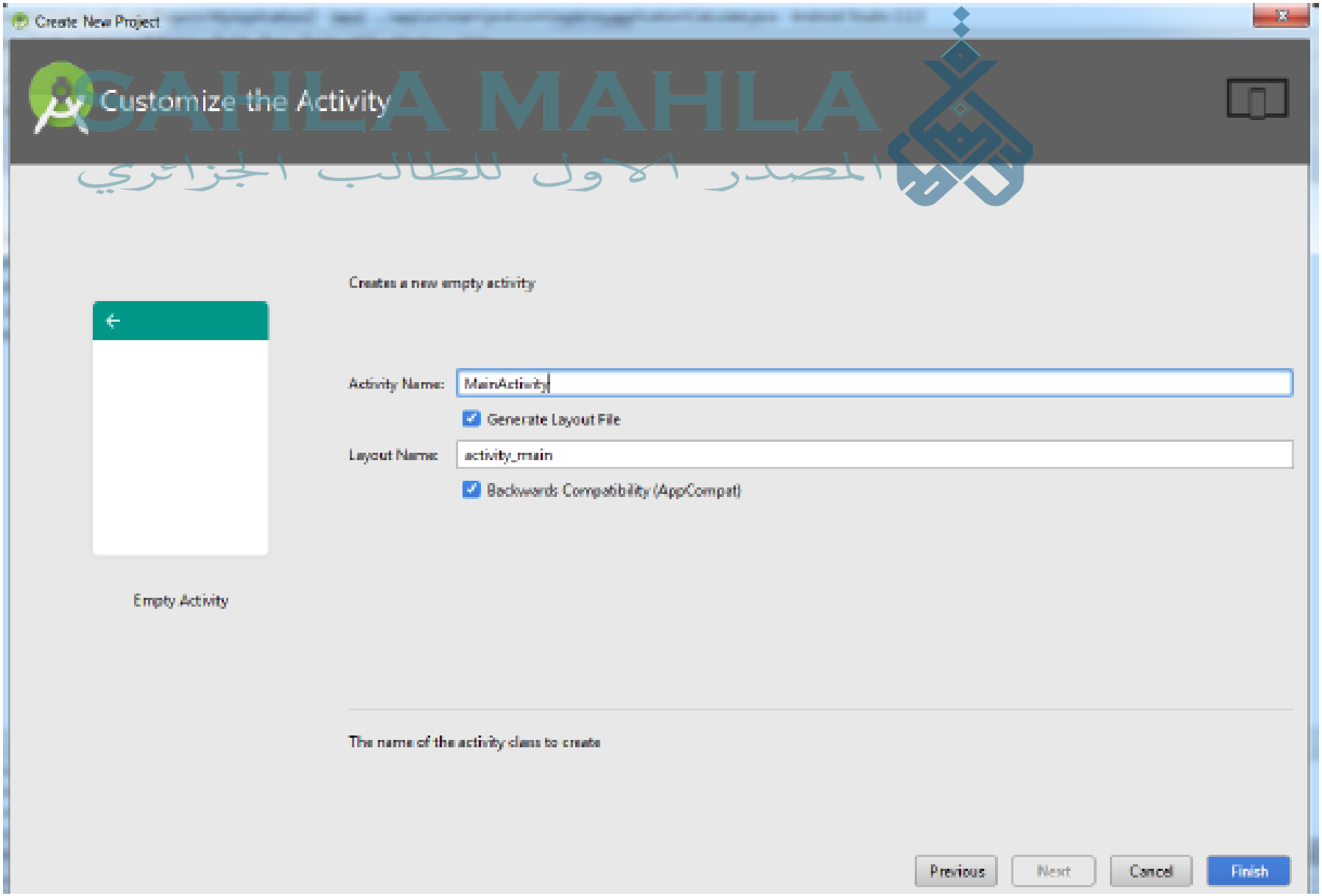
Choix de la version



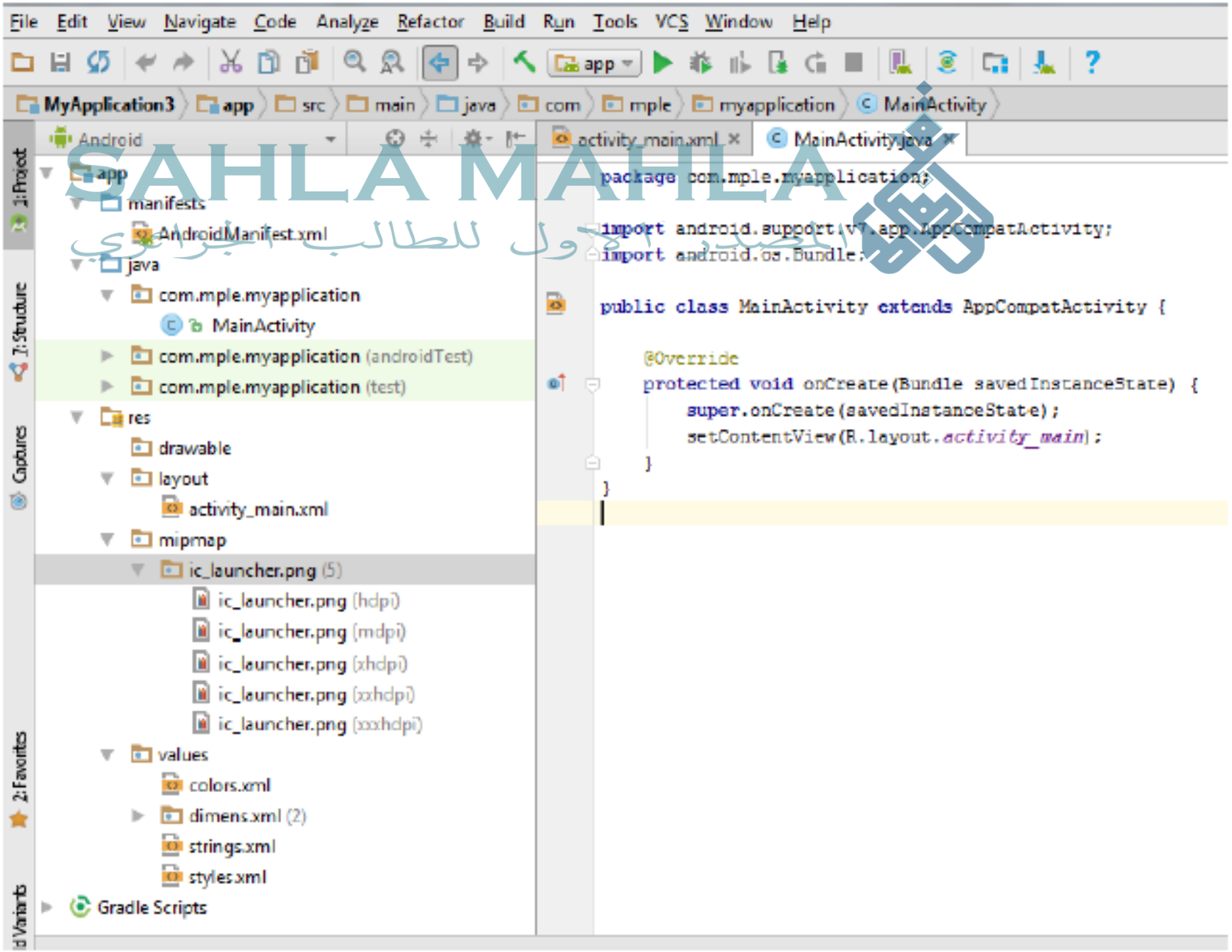
Choix du type d'activité



Introduire le nom de l'activité et de layout



Résultat de l'assistant de de création de projet



Anatomie de l'application Android

AndroidManifeste.xml : décrit les caractéristiques fondamentales de l'application et définit chacune de ses composants.

Java : contient les fichiers source .java pour le projet. Par défaut, il inclut un fichier source MainActivity.java

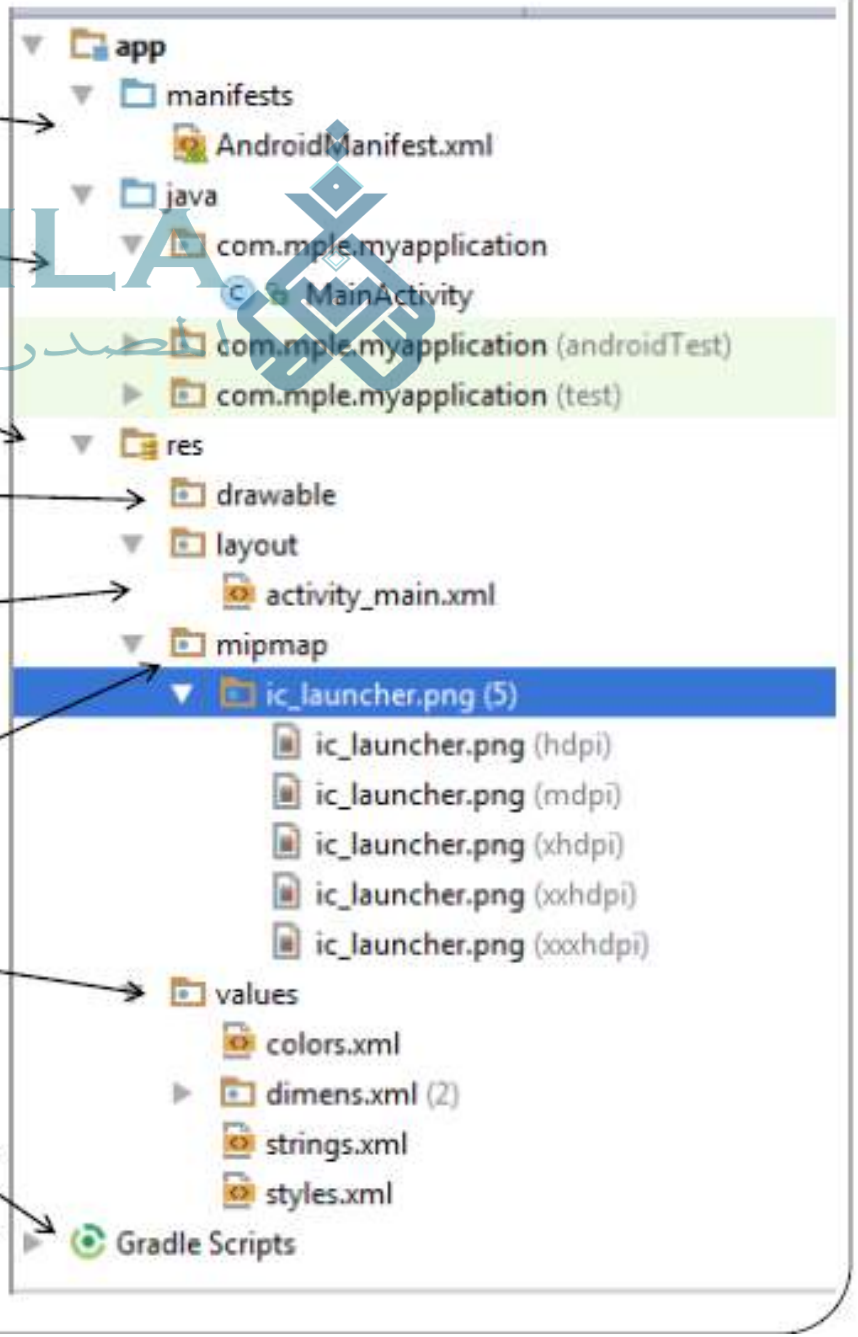
res/drawable : Fichiers XML et images, icones utilisés dans l'interface. répertoire pour les objets conçus pour les écrans haute densité.

Res/layout: un répertoire pour les fichiers qui définissent l'interface utilisateur de l'application

mipmap: images, icônes utilisés dans l'interface

Values: un répertoire pour d'autres fichiers XML divers contenant une collection de ressources, telles que des définitions de chaînes et de couleurs.

Gradle scripts: l'outil de compilation du projet. Un fichier généré automatiquement qui contient compileSdkVersion, buildToolsVersion, applicationId, minSdkVersion, targetSdkVersion, versionCode et versionName



Le fichier AndroidManifest.xml

android:versionCode : un nombre entier (positif et sans virgule) qui indique quelle est la version actuelle de l'application.
N'est montré à l'utilisateur, mais considéré par le Play Store.
Si on met une application avec un code de version supérieur à celui de l'ancienne, alors le Play Store considère que l'application a été mise à jour.

android:versionName : peut être une chaîne de caractères et sera montré à l'utilisateur.
Exemple android:versionName="Première version alpha - 0.01a"

La version d'Android (API 15) ou supérieure pour pouvoir utiliser cette application (cette application ne sera proposée sur Google Play uniquement si un utilisateur utilise cette version d'Android ou une version supérieure).

```
<uses-sdk android:minSdkVersion="15"
          android:targetSdkVersion="23" />
```

La version à partir de laquelle on pourra exploiter à fond l'application

Il décrit les attributs et les composants de l'application.
Par défaut, l'application n'a qu'un composant, l'activité principale.

```
<application
    android:allowBackup="true"
    android:icon="@mipmap/ic_launcher"
    android:label="My Application"
    android:supportsRtl="true"
    android:theme="@style/AppTheme">
    <activity android:name=".MainActivity">
        <intent-filter>
            <action android:name="android.intent.action.MAIN" />

            <category android:name="android.intent.category.LAUNCHER" />
        </intent-filter>
    </activity>
</application>
```

Définition de l'icône, du nom et du thème de l'application

Déclaration des activités

Indique que l'activité est le point d'entrée de l'application et doit s'afficher dans le lanceur d'applications

Exécution de l'application

- Le SDK Android permet de :

  - Installer l'application sur un appareil réel connecté par USB

  - Simuler l'application sur une tablette virtuelle *AVD*

AVD = Android Virtual Device

Université d'Alger 1 Benyoucef Ben Kheda
Faculté des Sciences
Département MI

L3 - SI

Développement d Applications Mobiles

DR. AICHA BOUTORH

2017/2018

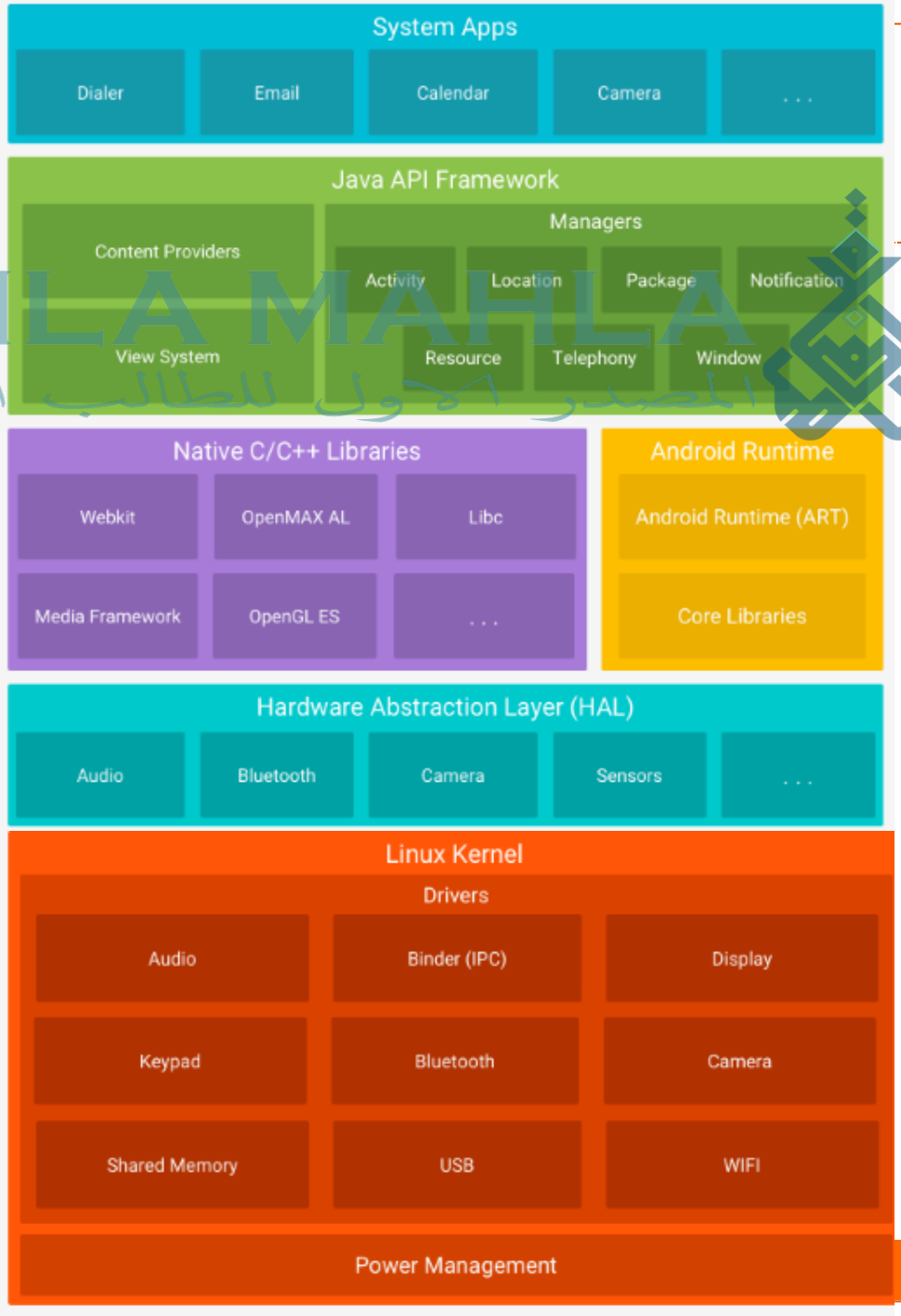
Architecture et Concepts d'Android

- Android avait d'abord été conçu pour les *smartphones* et tablettes tactiles, puis s'est diversifié dans les objets connectés et ordinateurs comme:

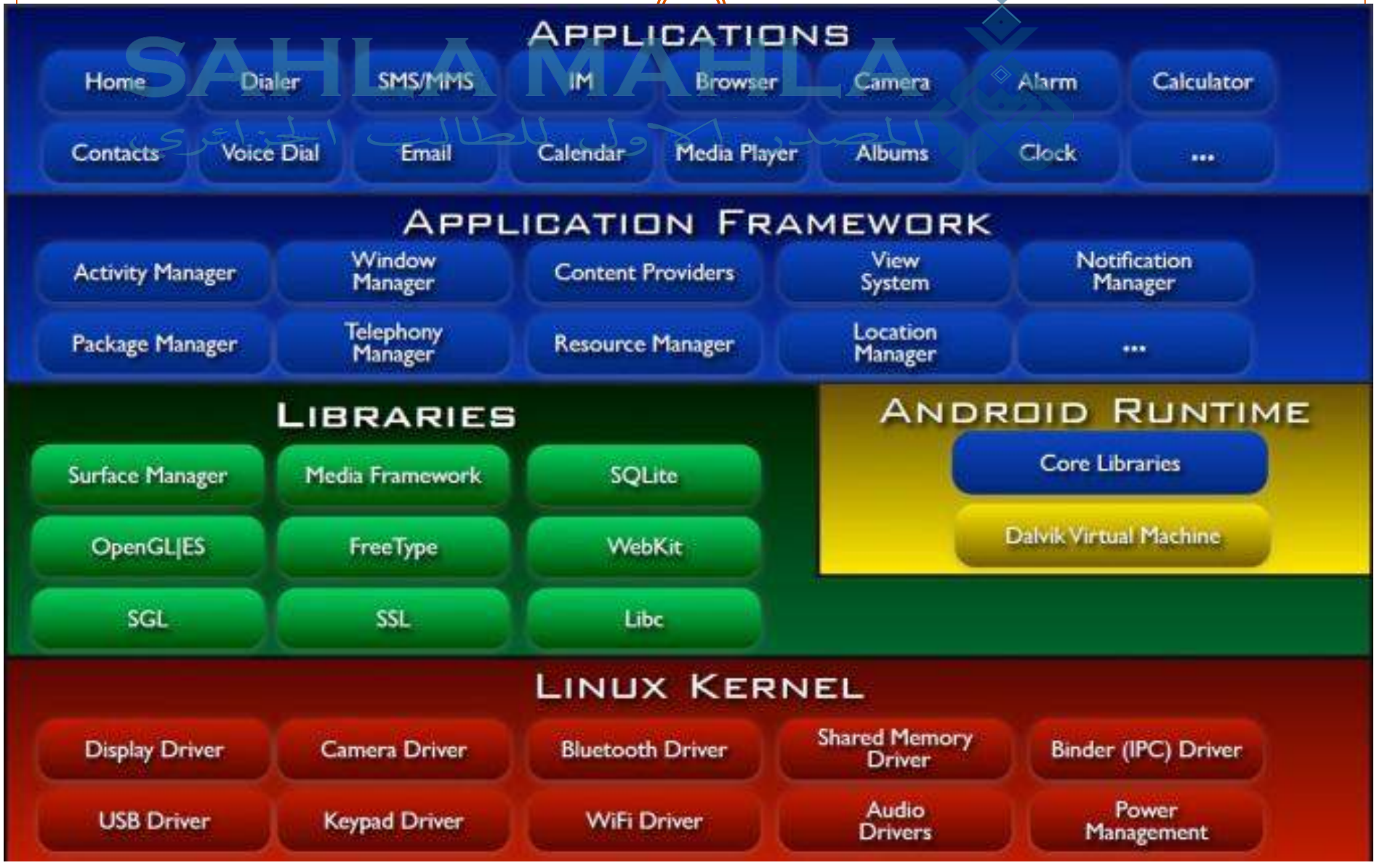
- les télévisions (AndroidTV),
- les voitures (AndroidAuto),
- les ordinateurs (Android-x86) et
- les smartwatch(AndroidWear).

Il offre:

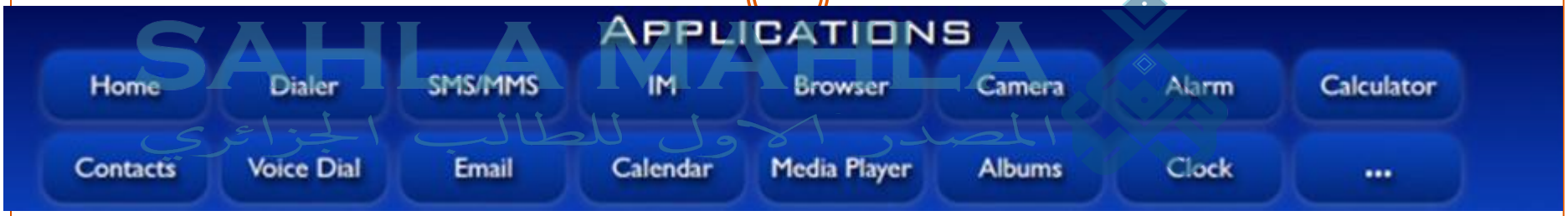
- Environnement de développement complet:–émulateur de téléphone, outils de débogage, mesure des performances, …
- Framework applicatif
- Machine virtuelle Dalvik–
- Navigateur intégré–WebKit
- Graphisme 2D et 3D
- Base de données SQLite
- CODEC audio et vidéo (MPEG4, MP3, AAC, PNG, …)
- Options matérielles : téléphonie GSM, caméra, GPS, accéléromètre, Wifi, Bluetooth, …



Architecture logicielle de la plate forme Android



Applications



- Vous trouverez toute l'application Android dans la couche supérieure. Vous allez écrire votre application à installer sur cette couche uniquement.
- Android est fourni avec un ensemble de programmes de base (également nommés applications natives) permettant d'accéder à des fonctionnalités comme les photos, les cartes géographiques, le Web, les SMS, le téléphone, le calendrier, ...
- Ces applications sont développées à l'aide du langage de programmation Java.
- Pour l'utilisateur final, c'est la seule couche accessible et visible

Application Framework

- En fournissant une plateforme de développement ouverte, Android offre aux développeurs la possibilité de créer des applications extrêmement riches et innovantes.
- Android introduit la notion de **Services**.
  Un service est une application qui n'a aucune interaction avec l'utilisateur et qui tourne en arrière plan pendant un temps indéfini
- La couche Application Framework fournit de nombreux services de niveau supérieur aux applications sous la forme de classes Java. Les développeurs d'applications sont autorisés à utiliser ces services dans leurs applications.
- Les développeurs sont libres de profiter du matériel périphérique, les informations de localisation d'accès, exécutez les services d'arrière-plan, définir des alarmes, ajouter des notifications de la barre d'état, et beaucoup, beaucoup plus...

Les services cœurs de la plateforme (**Core Plateform Services**) fournissent des services essentiels au fonctionnement de la plateforme :

- **Activity Manager** : gère le cycle de vie des applications et maintient une "pile de navigation" (navigation backstack) permettant d'aller d'une application à une autre et de revenir à la précédente quand la dernière application ouverte est fermée.
- **Package Manager** : utiliser par l'*Activity Manager* pour charger les informations provenant des fichiers **.apk** (android packaga file)
- **Window Manager :** gère les fenêtres des applications --> quelle fenêtre doit être afficher devant une autre à l'écran.
- **Resouce Manager :** gère tous ce qui n'est pas du code, toutes les ressources --> *images, fichier audio, etc.*
- **Notification Manager :** Permet aux applications d'afficher des alertes et des notifications à l'utilisateur
- **Content Provider :** gère le partage de données entre applications, comme par exemple la base de données de contact, qui peut être consultée par d'autres applications que l'application Contact. Les Données peuvent être partagées à travers une *base de données (**SQLite**), des fichiers, le réseau, etc.*
- **View System** : fournit tous les composants graphiques : listes, grille, texte box, boutons et même un navigateur web embarqué.

Les bibliothèques (Libraries)

- Android inclut un ensemble de bibliothèques **C/C++** utilisées par de nombreux composants de la plateforme Android. Cette catégorie englobe les bibliothèques spécifiques au développement Android. Des exemples incluent les bibliothèques de framework d'application en plus de celles qui facilitent la création d'interface utilisateur, le design graphique et l'accès à la base de données. Ils sont accessibles au développeur par l'intermédiaire du ***framework Android*:**

- **Bibliothèques multimédias :** permettent le support de nombreux formats ***audio*** et ***vidéo***, tels que ***MPEG4***, ***MP3***, ***JPG***, ***PNG*** ….
- **Skia:** Moteur graphique 2D.
- **OpenGL:** Bibliothèques 3D.
- **SQLit.** Stockage et partage des données des applications
- **WebKit:** Moteur de navigation Web

Moteur d'exécution Android (AndroidRuntime)

- Cette section fournit un composant clé appelé **Dalvik Virtual Machine** qui est une sorte de machine virtuelle Java spécialement conçue et optimisée pour Android.
- **Dalvik VM**: fournit une machine virtuelle alternative, adaptée aux limitations des appareils mobiles, permet l'exécution simultanée de plusieurs applications
- La machine virtuelle Dalvik utilise des fonctionnalités de base Linux telles que la gestion de la mémoire et le multithreading, qui est intrinsèque au langage Java. La machine virtuelle Dalvik permet à chaque application Android de s'exécuter dans son propre processus, avec sa propre instance de la machine virtuelle Dalvik.
- Apartir de la version Android5.0 (Lollipop), **Dalvik** a été remplacé par **ART (AndroidRunTime)**

Les services matériels (**Hardware Services**) fournissent un accès vers les *API matérielles* de bas niveau :

- **Telephony Service** : permet d'accéder aux interfaces "téléphonique" (gsm, 3G, etc.)
- **Location Service** : permet d'accéder au GPS.
- **Bluetooth Service** : permet d'accéder à l'interface bluetooth.
- **WiFi Service** : permet d'accéder à l'interface Wifi.
- **USB Service** : permet d'accéder aux interfaces USB.
- **Sensor Service** : permet d'accéder aux détecteurs (détecteurs de luminosité, etc.)

Noyau Linux (Linux Kernel)

- Au bas des couches est Linux - Android repose sur un noyau Linux qui fournit un niveau d'abstraction entre le matériel de l'appareil et la pile logicielle, il contient tous les pilotes matériels essentiels et gère les services du système, comme la sécurité, la gestion de la mémoire et des processus, la pile réseau et les pilotes, la caméra, le clavier, l'affichage, etc.

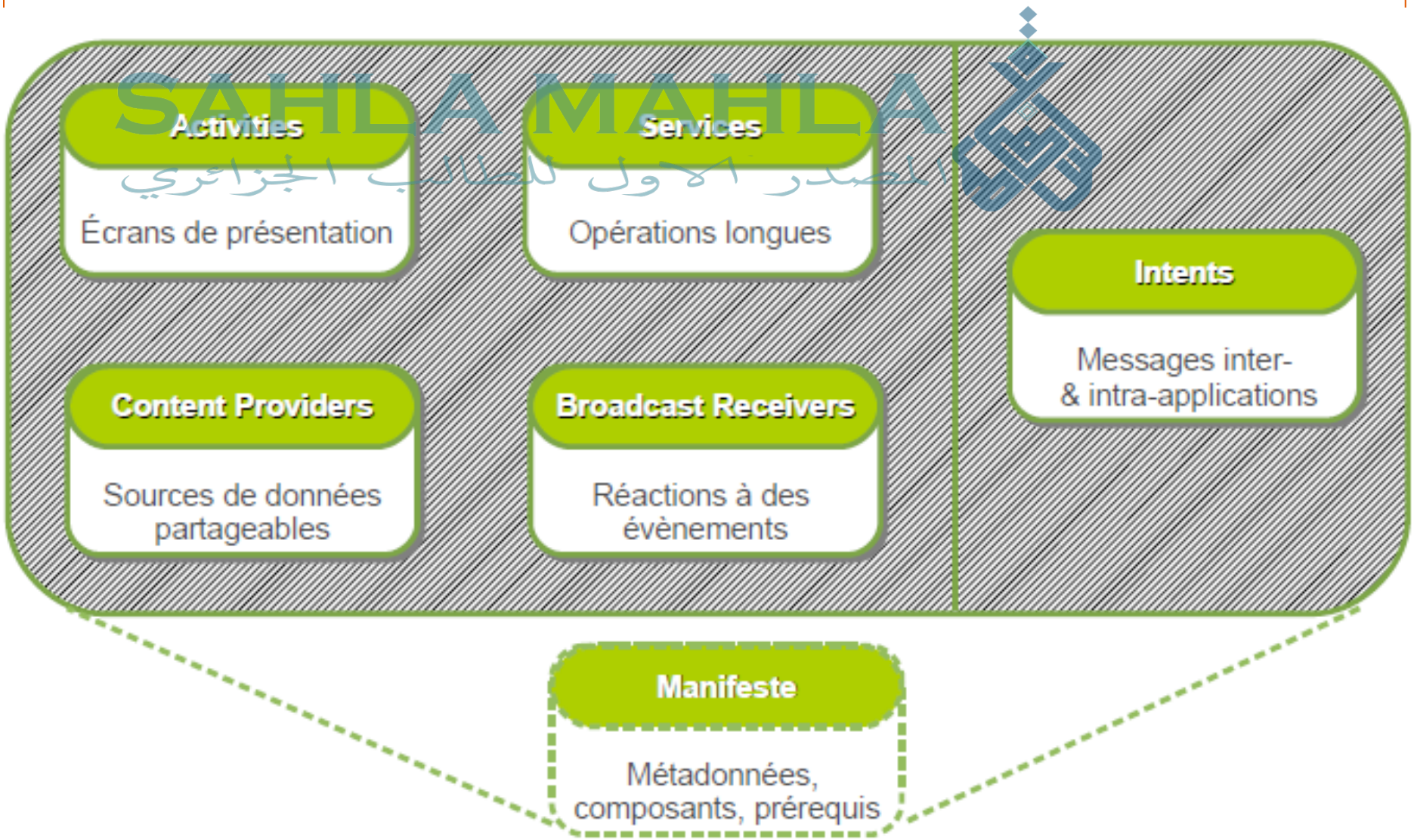
- Le noyau de linux permet de:
  - Gérer la mémoire
  - Gérer les processus
  - Gérer le matériel (écran clavier ...)
  - Gérer les capteurs (appareil photo, GPS, wifi...)

Construction d'une Application Android

Composants applicatifs

- Une application Android est un groupe de composants
- Elle a plusieurs points d'entrées pour s'exécuter –
- Il n'y a pas qu'un seul "main" du point de vue d'un développeur d'application Android
- Les composants d'application sont les éléments constitutifs essentiels d'une application Android. Ces composants sont faiblement couplés par le fichier manifeste d'application AndroidManifest.xml qui décrit chaque composant de l'application et leur interaction.

Composants applicatifs

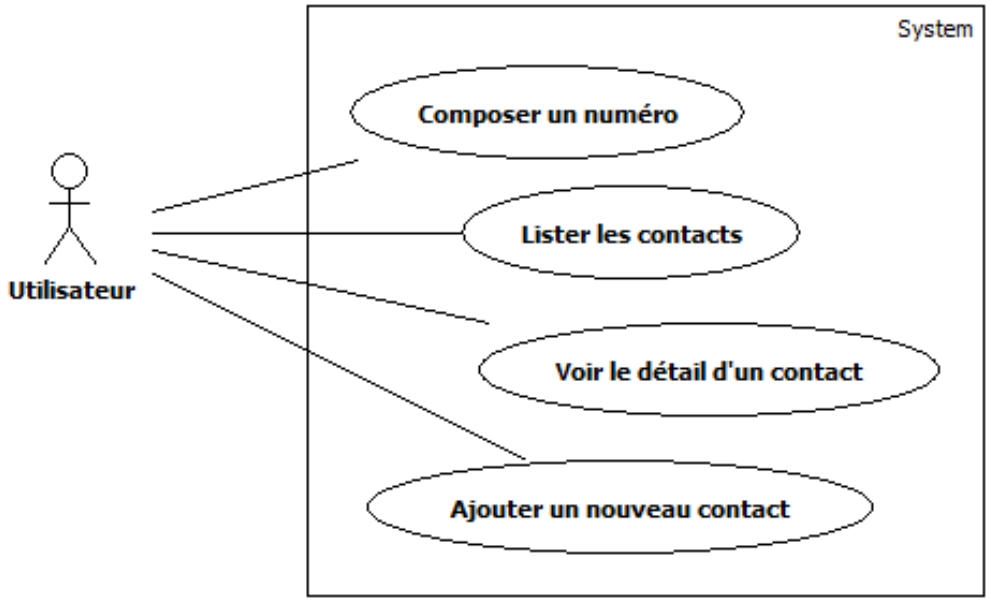


1)Activities

- Une activité représente une seule actions sur l'écran avec une interface utilisateur.
- – Le seul composant qui soit visible à l'utilisateur final
- – Le composant dans lequel est programmé la logique IHM(Interaction Homme Machine)
- – Permet d'interagir avec l'utilisateur final
- Par exemple, une application de messagerie peut avoir une activité qui affiche une liste de nouveaux e-mails, une autre activité pour composer un e-mail et une autre activité pour lire des e-mails. Si une application a plus d'une activité, l'une d'entre elles doit être marquée comme activité présentée lors du lancement de l'application.

1) Activities

- Une application est formée de *n* activities
- Exemple : **application de téléphonie**
- 1) Numéroteur
- 2) Annuaire des contacts
- 3) Fiche d'un contact
- 4) Ajout d'un contact



- Etend ***android.app.Activity***

2) Services

- – Ils gèrent le traitement en arrière-plan associé à une application
- Conçus pour exécuter des opérations qui sont longues ou qui ont besoin de tourner indéfiniment
- Par exemple : *Téléchargement et upload de gros fichiers*
- Un service peut lire de la musique en arrière-plan pendant que l'utilisateur se trouve dans une application différente, ou il peut extraire des données sur le réseau sans bloquer l'interaction de l'utilisateur avec une activité.
- Un service ≈ une activité sans GUI
- Sert a effectuer des opérations ou des calculs en dehors de l'interaction utilisateur

2) Services

- Deux types de services :
- **Local** : service qui s'éxecute dans le même processus que votre application
- **Distant** (IPC) : service qui s'éxecute dans des processus indépendants de votre application
- Etend ***android.app.Service***

3) Broadcast Receiver

- – Autre composant applicatif d'Android qu'ils gèrent la communication entre Android OS et les applications.
- Les récepteurs de diffusion répondent simplement aux messages de diffusion provenant d'autres applications ou du système.
- Par exemple, les applications peuvent également initier des diffusions pour permettre à d'autres applications de savoir que certaines données ont été téléchargées sur l'appareil et qu'elles peuvent être utilisées. C'est donc le récepteur de diffusion qui interceptera cette communication et lancera l'action appropriée.
- – Activation/Désactivation du Wifi
- – Rétablissement de connexion réseau
- – Boot Completed, pour notifier la fin du démarrage du système » Permet de lancer automatiquement son application au démarrage du système

3) Broadcast Receiver

- Réagit aux annonces diffusées
  - System-defined : la batterie est faible, un SMS vient d'arriver, etc.
  - User-defined : solde bancaire negatif, etc.
- Ne nécessite pas une interface graphique
- Un broadcast receiver est une classe qui étend ***android.content.BroadcastReceiver***
- Un receiver s'abonne/desabonne via le fichier manifest ou par programme.

4) Content Provider

- Ils gèrent les problèmes de gestion des données et des bases de données. Sert a rendre une partie des données d'une application accessibles aux autres applications

- Seul moyen pour un partage de données inter-applications

- Un composant fournisseur de contenu fournit des données d'une application à d'autres sur demande. Ces demandes sont gérées par les méthodes de la classe ContentResolver. Les données peuvent être stockées dans le système de fichiers, la base de données ou ailleurs entièrement.

- Un content provider est une classe qui etend

android.content.ContentProvider

*) Intent

- – Composant du système Android faisant office d'IPC (Inter Prrocess Communication) entre applications soit Activités ou Services
- – Permet d'invoquer d'autres activités ou services
- – Exposer des fonctionnalités aux autres activités en guise de réutilisation de code
- • L'activité Camera permet de prendre des photos
- • Le "browser" peut ouvrir une page web via une URL
- • L'application email peut envoyer des emails

SAHLA MAHLA

المصدر الاول للطالب الجزائري

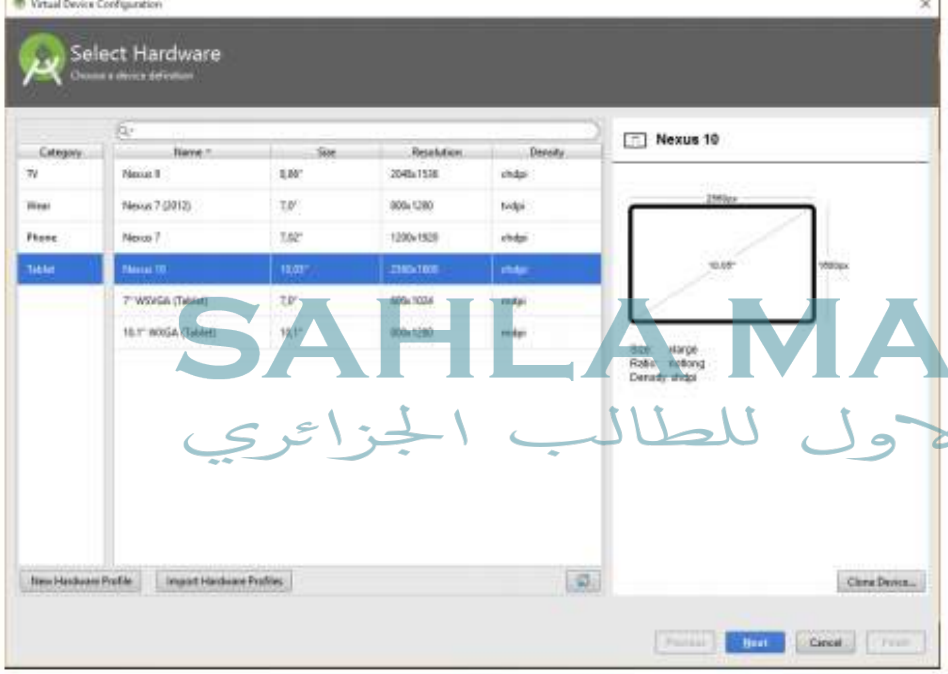
Exécution de l'Application

- Le test de l'ensemble du projet permet de simuler le lancement de l'application par l'utilisateur (exécutable, package, ...).

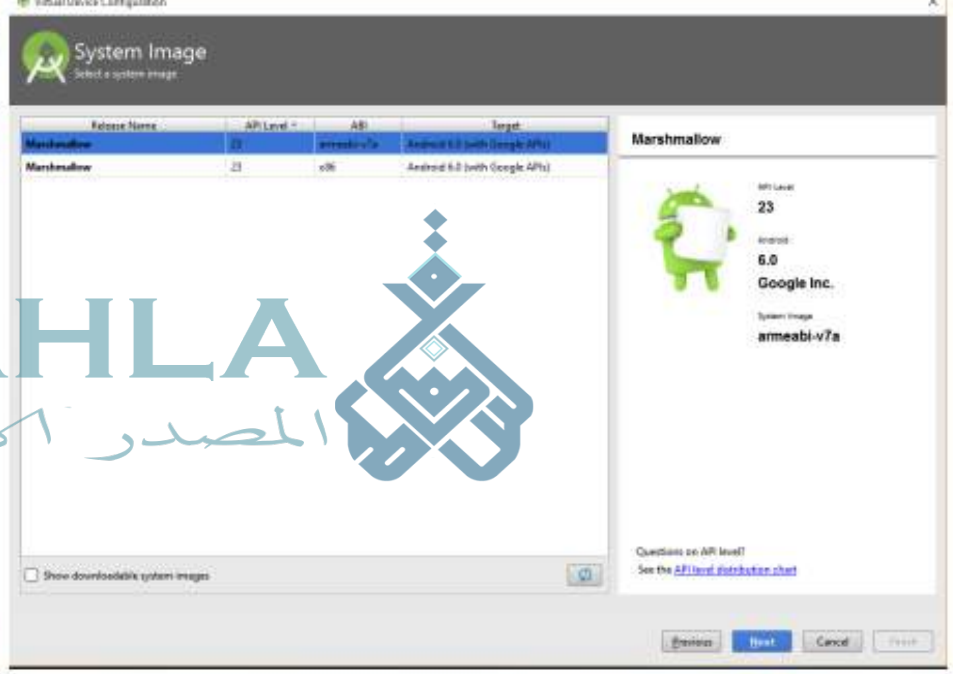
- Il est ainsi possible de tester l'application dans son ensemble, même si son développement n'est pas terminé.

Sur un émulateur

- Un émulateur permet de reproduire le comportement d'un appareil réel d'une façon virtuelle.
- L'utilisation d'un émulateur nous évite d'avoir à charger à chaque fois l'application dans un appareil pour la tester.
- On pourra ainsi lancer l'application dans l'IDE et elle s'exécutera sur un appareil virtuel appelé Android Virtual Device AVD qui émule le comportement d'un téléphone, une tablette ou autre.



(a) Choix de l'appareil



(b) Choix du processeur

- Pour configurer un émulateur, allez dans Tools > Android > AVD Manager, un émulateur existe par défaut, mais vous pourriez rajouter d'autres appareils en cliquant sur *Create Virtual Device* .
- Dans l'exemple de la figure rajouter une tablette Nexus 10,

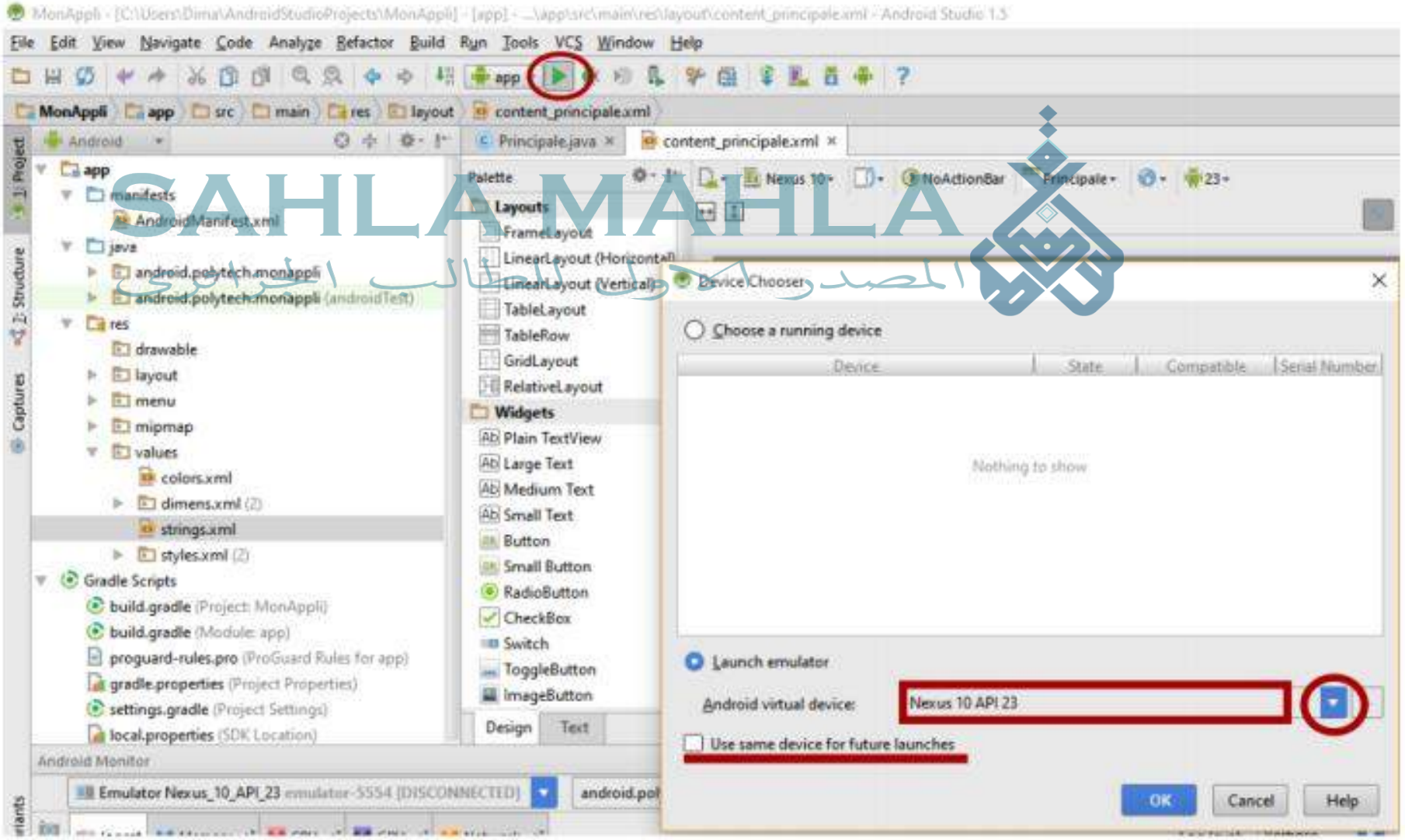
Sélectionner ensuite le processeur qui sera émulé.

- En cochant Show downloadable system images vous pouvez télécharger d'autres images systèmes avec d'autres versions Android. En cliquant sur Next Vous avez ensuite la possibilité de configurer d'autres paramètres.

- Pour lancer l'exécution sur l'émulateur, appuyez sur le bouton d'exécution et sélectionnez l'émulateur sur lequel vous souhaitez lancer l'application.

- Vous pouvez cochez « Use same device for future launches » pour éviter d'avoir à sélectionner l'appareil à chaque lancement.

- L'émulateur se lance. Rassurez-vous, vous n'aurez pas à le relancer à chaque fois que vous compilez votre projet, laissez-le ouvert et à chaque fois que vous compilez et relancez votre application, elle pourra être chargée dans l'émulateur en cours



Sur un appareil réel

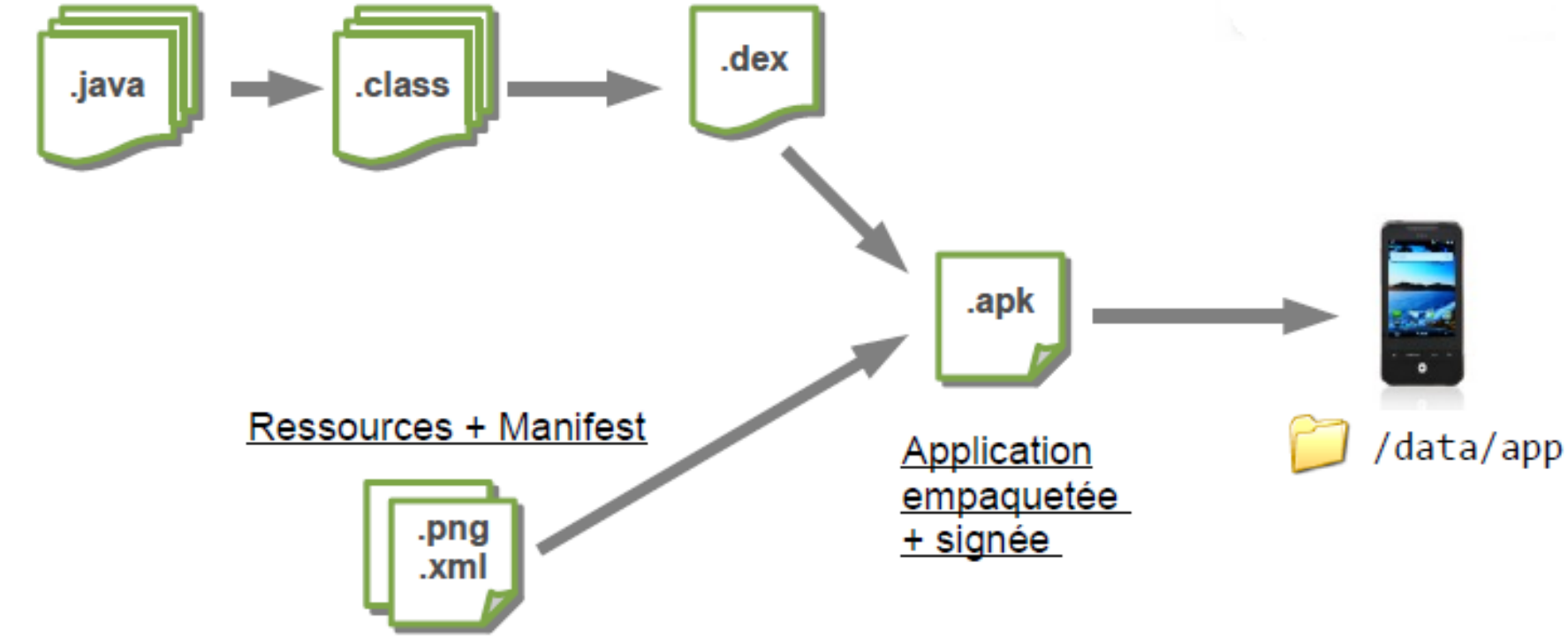
- Connectez votre appareil par câble USB à l'ordinateur et installez le pilote si nécessaire
- Activez l'option de débogage USB sur l'appareil en allant dans les paramètres, sous développement ou options pour les développeurs.
- Lancez l'application depuis Android Studio comme précédemment. Si on vous demande de choisir l'appareil, sélectionnez « Choose a running device », puis votre téléphone ou tablette. Android Studio installera l'application sur votre appareil et la lancera.
- Si au moment de la sélection de l'appareil vous recevez un message indiquant que l'appareil n'est pas compatible, assurez vous d'abord que la version Sdk de l'appareil est bien supérieure au minSdk version de votre projet.

- **Une fois que votre application est compilée, un fichier .apk est créé dans le dossier app\build\outputs\apk de votre répertoire de travail.**

- **C'est l'exécutable de votre application.**

- **C'est ce fichier que vous devez déployer pour distribuer votre application.**

Compilation & Déploiement



Université d'Alger 1 Benyoucef Ben Kheda
Faculté des Sciences
Département MI

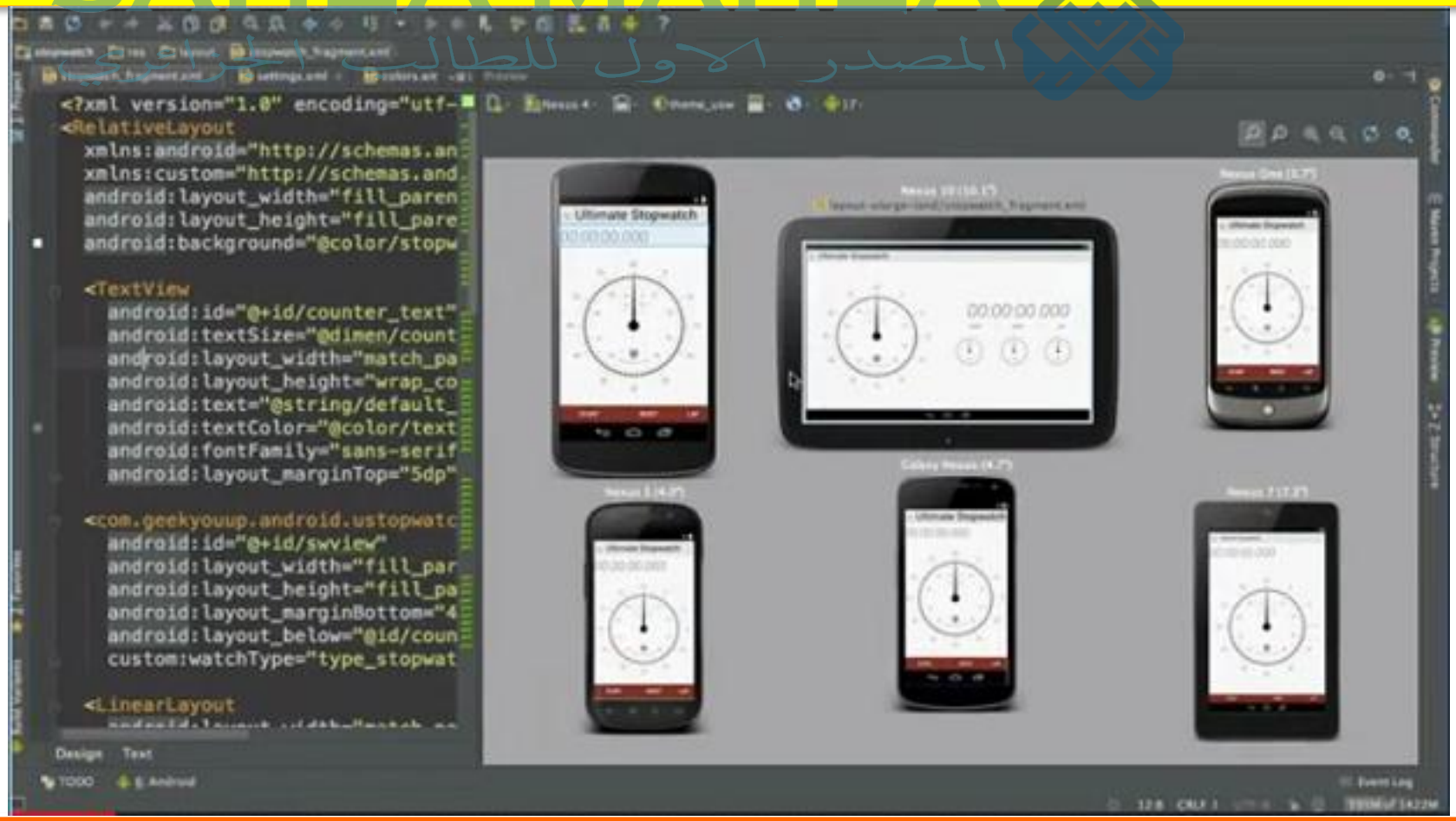
L3 - SI

Développement d Applications Mobiles

DR. AICHA BOUTORH

2017/2018

Interface Utilisateur Graphique



Graphical User Interface (GUI)

Interfaces

- La mise en place d'une interface peut se faire de deux façons :

- – Par description de l'interface dans des fichiers **XML**

- – Par programme

Extensible Markup Language: XML Overview

- **langage de balisage extensible**
- **Extensible:** (pensez "personnalisable") Vous pourriez créer votre propre ensemble de balises XML pour n'importe quel but que vous souhaitez.
- C'est un langage de balisage, car il utilise de simples "balises" pour définir ce que vous voulez faire.
- Un autre langage de balisage appelé **HTML**, ou **Hypertext Markup Language**, qui est utilisé pour créer des sites Web

Pourquoi xml?

- La raison de cette approche, à savoir l'utilisation de XML pour tout ce qui peut être considéré comme orienté conception, est que l'utilisation de XML libère les membres de votre équipe de développement d'applications qui conçoivent:
  - la convivialité de votre application,
  - l'accès aux fonctionnalités,
  - l'interface utilisateur,
  - les styles,
  - le thème,
  - les graphiques, etc.,

de devoir apprendre (c'est-à-dire, comprendre) comment fonctionne la programmation Java.

Fonctionnalités Avancées

- Quelques exemples de fonctionnalités avancées liées à la conception Android que vous pouvez implémenter principalement à l'aide de «définitions» de balisage XML incluent principalement:
  - des graphiques multi-états,
  - des éléments d'interface utilisateur personnalisés,
  - des images, des animations,
  - les menus d'options, les menus contextuels,
  - les boîtes de dialogue, les boîtes de dialogue d'alerte,
  - les styles, les thèmes
  - et le manifeste de votre application.

- Vous pouvez également implémenter des fonctions de conception stratégiques moins avancées en utilisant XML, y compris:
  - des valeurs constantes de chaîne (texte) pour votre application,
  - des constantes de valeurs entières (numériques),
  - des constantes de valeur état (booléen),
  - les valeurs d'espacement d'écran (dimension) pour vos conceptions d'interface utilisateur.
  - Les tableaux, qui sont des collections de données utilisées dans votre application (comme une simple base de données), peuvent également être créés et chargés avec leurs valeurs de données, en utilisant des fichiers XML.

- Le balisage XML est contenu dans de simples fichiers au format texte identifiés à l'aide de l'extension de fichier **.xml**.

- Ces fichiers XML peuvent ensuite être lus ou "analysés" par le système d'exploitation Android, ou le code Java de votre application, et transformés en structures d'objet Java.

Fichier .XML

- Les fichiers XML qui décrivent une interface sont:
  - placés dans le répertoire ***res/layout***.
  - référencés par ***R.layout.nom_du_fichierXML***.
- Les activités peuvent utiliser la méthode ***setContentView(R.layout.nom_du_fichierXML)*** pour mettre en place l'interface décrite par un tel fichier.
- Le bloc de base pour la création d'interfaces utilisateur est un objet ***View*** créé à partir de la classe ***View*** qui occupe une zone à l'écran et est responsable du dessin et de la gestion des événements.

View --- ViewGroup

- Dans une application Android, l'interface utilisateur est construite à l'aide d'objets de type *View* et *ViewGroup*.

- ***View*** est la classe de base des ***widgets*** « **composant d'interface graphique interactifs** » on cite comme exemples Champs de texte, Boutons, etc…

- ***ViewGroup*** est une sous-classe de ***View,*** fournit un conteneur invisible qui contient d'autres ***Views*** ou d'autres ***ViewGroups***, et organise la manière dont sont présents les composants.

- En utilisant le vocabulaire XML d'Android, vous pouvez rapidement concevoir des mises en page d'interface utilisateur et les éléments d'écran qu'elles contiennent, de la même manière que vous créez des pages Web en HTML, avec une série d'éléments imbriqués.

- Chaque fichier ***layout*** doit contenir exactement un élément racine, qui doit être un objet *View* ou *ViewGroup*.

- Une fois que vous avez défini l'élément racine, vous pouvez ajouter des objets *layout* en page ou des *widgets* supplémentaires en tant qu'*éléments enfants* pour créer progressivement une hiérarchie *View* qui définit votre *layout*.

Forme générale d'un fichier XML

- Un document XML est constitué par des éléments ayant chacun une balise de début, un contenu et une balise de fin.

- Un document XML doit avoir exactement un élément racine (une balise qui renferme toutes les autres balises). XML fait la distinction entre les lettres majuscules et minuscules.

- Un fichier XML « bien formé » doit remplir les conditions suivantes :
  • un document XML commence toujours par un prologue
  • chaque balise d'ouverture a une balise de fermeture ;
  • toutes les balises sont complètement imbriquées.

- Un fichier XML est dit *valide* s'il est bien formé et s'il contient un lien vers un **schéma XML**. Il est valide par rapport à ce schéma.

Forme générale d'un fichier XML

- Un document XML commence toujours par un **prologue** qui décrit le fichier XML.

- Ce prologue peut être minimal, par exemple: **<?xml version="1.0"?>**

- Ou peut contenir d'autres informations, par exemple l'encodage : **<?xml version="1.0" encoding="UTF-8" standalone="yes"?>**

Forme générale d'un fichier XML

<Classe_du_conteneur_principal
xmlns:android="http://schemas.android.com/apk/res/android"
propriétés du conteneur principal>

<Classe de conteneur ou d'élément d'interface
propriétés du conteneur ou de l'élément d'interface**/>**
...

<Classe de conteneur ou d'élément d'interface
propriétés du conteneur ou de l'élément d'interface**/>**

</Classe_du_conteneur_principal>

Par exemple, voici un **Layout XML** qui utilise un **LinearLayout** *vertical* pour contenir un ***TextView*** et un ***Button***:


```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
        android:layout_width="match_parent"
        android:layout_height="match_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>
```

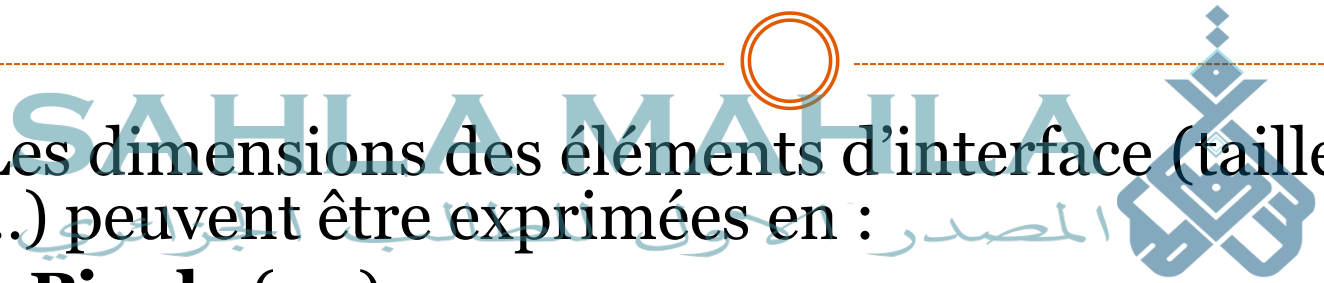
Après avoir déclaré votre layout au format XML, enregistrez le fichier avec l'extension **.xml** dans le répertoire ***res / layout /*** de votre projet Android afin qu'il soit correctement compilé.

Charger la ressource XML

- Lorsque vous compilez votre application, chaque fichier layout XML est compilé dans une ressource ***View***.
- Vous devez charger la ressource layout à partir du code de votre application dans votre implémentation ***Activity.onCreate ()***.
- Pour ce faire, appelez ***setContentView ()***, en lui passant la référence de votre ressource layout sous la forme: ***R.layout.layout_file_name***.
- Par exemple, si votre Layout XML est enregistrée en tant que **main_layout.xml**, vous devrez le charger pour votre activité comme ceci:

```
public void onCreate (Bundle savedInstanceState) {
   super.onCreate(savedInstanceState);
   setContentView(R.layout.main_layout);
}
```

Unités de mesure dans les fichiers XML



- Les dimensions des éléments d'interface (taille, marges, …) peuvent être exprimées en :
- – **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
- • Dans les fichiers **XML** les unités sont exprimées sous la forme : **"30.5mm"** ou **"74px"** …

Propriétés des éléments d'interface

- **Identifiant:** Tout objet View peut être associé à un identifiant entier, afin d'identifier de manière unique la vue dans l'arborescence.
- Lorsque l'application est compilée, cet ID est référencé en tant qu'entier, mais l'ID est généralement affecté dans le fichier XML de présentation sous la forme d'une chaîne, dans l'attribut id. C'est un attribut qui peut être associé à chaque élément décrit dans un fichier XML, et permet de **référencer l'objet** créé dans d'autres fichiers XML
- Les éléments ne devant pas être référencés peuvent ne pas avoir d'identifiant.
- La syntaxe d'un ID: ***android:id="@+id/mon_id"***
- Pour retrouver cet élément ont utilise la méthode ***findViewById (R.id.mon_id)***.

Identifiant (ID)

- Le symbole at (@) au début de la chaîne indique que l'analyseur XML doit analyser et développer le reste de la chaîne d'ID et l'identifier comme ressource d'ID.

- Le signe plus (+) signifie qu'il s'agit d'un nouveau nom de ressource qui doit être créé et ajouté à nos ressources (dans le fichier R.java).

- Il existe un certain nombre d'autres ressources d'identification offertes par le framework Android.

Afin de créer des vues (views) et de les référencer à partir de l'application, un modèle commun est de:

- 1- Définissez une vue / un widget dans le fichier layout et attribuez-lui un identifiant unique:

```
<Button android:id="@+id/my_button"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/my_button_text"/>
```

- 2- Créez ensuite une instance de l'objet de vue et capturez la à partir de layout (généralement dans la méthode onCreate ()):

```
Button myButton = (Button) findViewById(R.id.my_button);
```

Paramètres de Layout

- Chaque classe ViewGroup implémente une classe imbriquée qui étend ***ViewGroup.LayoutParams***.
- Cette sous-classe contient les types de propriété qui définissent **la taille** et **la position** de chaque vue enfant, selon le groupe de vues.

- Notez que chaque sous-classe LayoutParams a sa propre syntaxe pour définir les valeurs. Chaque élément enfant doit définir LayoutParams qui convient à son parent, bien qu'il puisse également définir LayoutParams différent pour ses propres enfants.

Paramètres de Layout

- **La taille:** Tous les éléments qui composent une interface graphique doivent fournir une taille, grâce aux deux attributs :
  - ***android:layout_height***
  - ***android:layout_width***

- Ces deux propriétés peuvent prendre 3 types de valeur :
  - **Une taille fixe :** par exemple 30px (pixels).
  - **match_parent**: indique à la vue de devenir aussi grande que son groupe de vue parent permettra i.e L'élément occupe tout l'espace disponible chez son conteneur parent.
  - **wrap_content**: indique à la vue de se dimensionner aux dimensions requises par son contenu i.e. L'élément occupe la place nécessaire (la taille de son contenu).

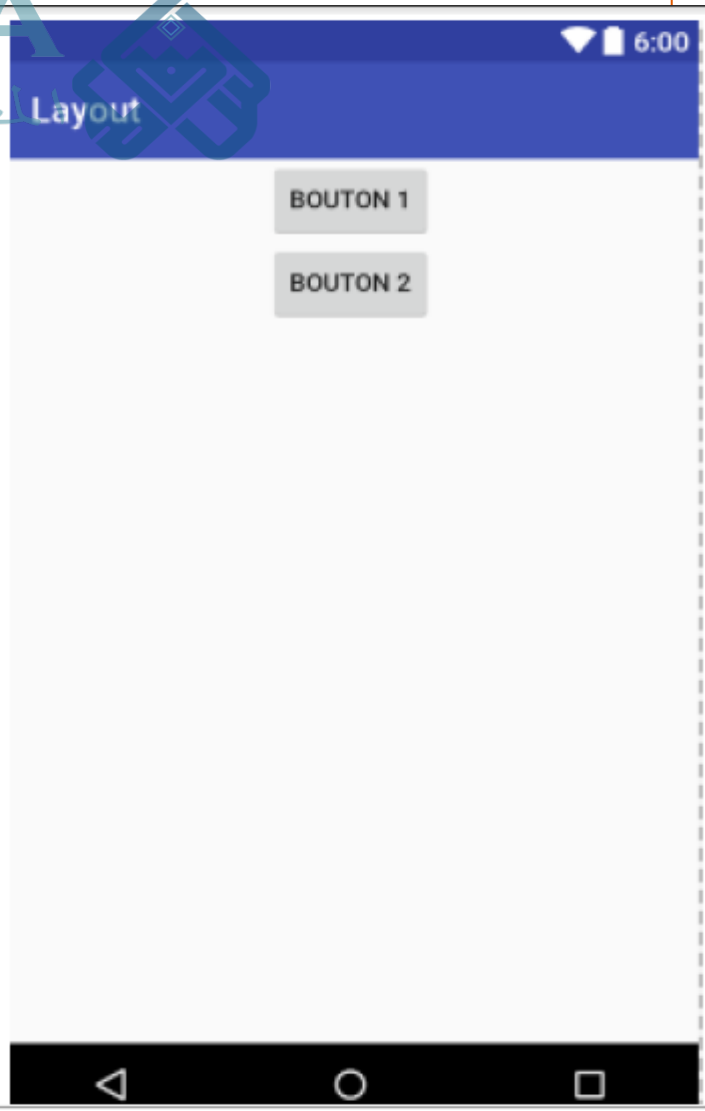
Position de Layout

- **Gravité des éléments:** Les éléments sont placé dans le layout en haut et à gauche par défaut.
- Pour positionner un élément autre part, on utilise l'attribut : **"android:gravity** qui peut prendre différentes valeurs :

–android:gravity=*"top"*
–*android:gravity="bottom"*
–android:gravity="left"
–*android:gravity="right“*
–android:gravity=*"center_horizontal"*
–android:gravity=*"center_vertical"*

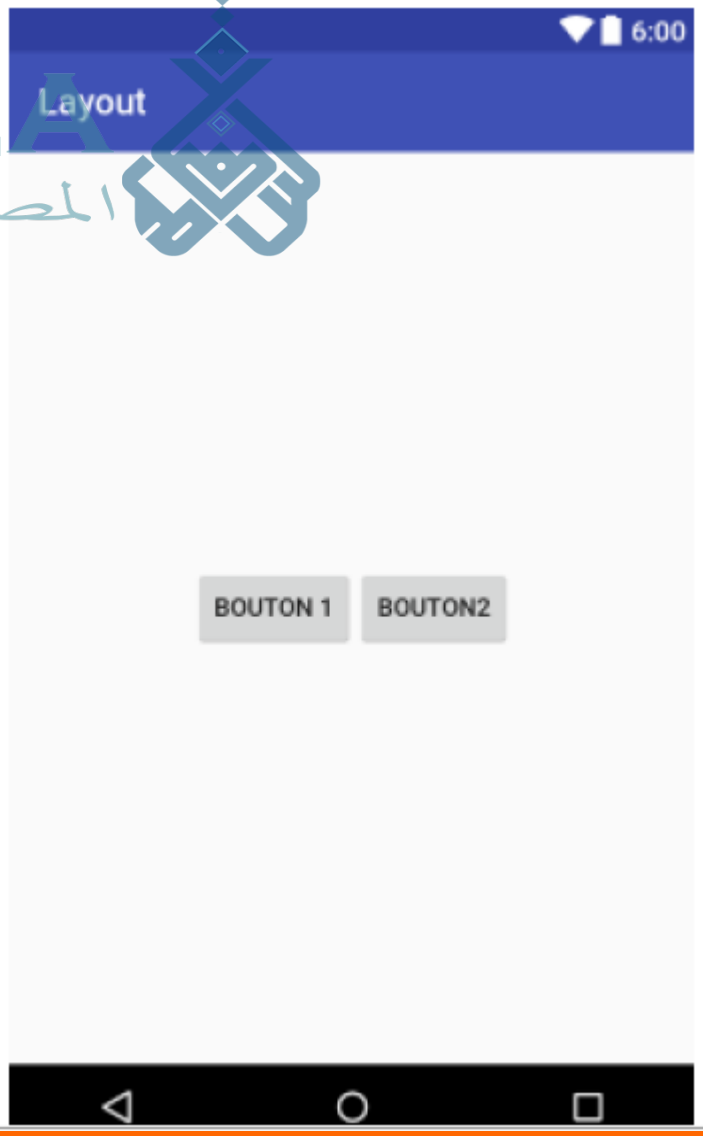
Il est possible de combiner les valeurs grâce au symbole | (touche 6 + Alt Gr) : android:gravity=*“bottom|right"*

```xml
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
android:gravity="center_horizontal"
android:orientation="vertical"
android:layout_width="match_parent"
android:layout_height=" match_parent ">
<Button
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:text="Bouton 1"/>
<Button
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:text="Bouton 2"/>
</LinearLayout>
```



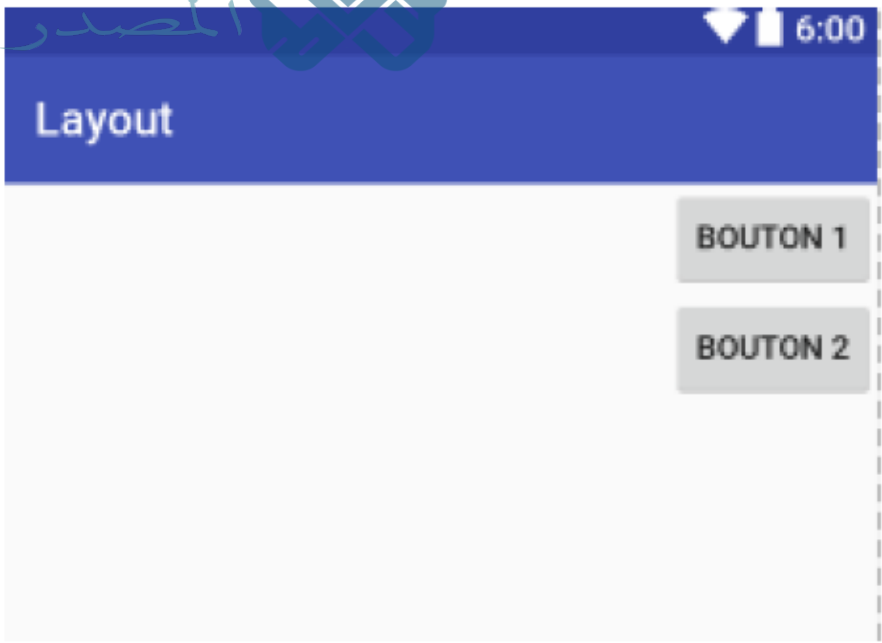
[Ref]

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
android:gravity="center"
android:orientation="horizontal"
android:layout_width=" match_parent "
android:layout_height=" match_parent ">
<Button
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:text="Bouton 1"/>
<Button
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:text="Bouton2"/>
</LinearLayout>
```

[Ref]

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
android:orientation="vertical"
android:gravity="right"
android:layout_width="match_parent"
android:layout_height="match_parent">
<Button
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:text="Bouton 1"/>
<Button
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:text="Bouton 2"/>
</LinearLayout>
```

[Ref]

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
android:orientation="horizontal"
android:layout_width=" match_parent "
android:layout_height=" match_parent ">
<Button
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:layout_gravity="center"
android:text="Bouton 1"/>
<Button
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:text="Bouton 2"/>
</LinearLayout>
```

[Ref]

Espacement

- Les différents éléments créés sont par défaut serrés les uns contre les autres.
- On peut augmenter l'espacement à l'aide de la propriété
  - ***android:padding*** *(**Marges internes**)*
  - ***android:layout_margin*** *(**Marges externes**)*

- **Exemple:**
- ***android:padding="9px"*** *(la valeur précise l'espace situé entre le contour de l'élément et son contenu).*
- ***android:layout_margin="9px"*** *(la valeur précise l'espace situé entre le contour de l'élément et le contour de ses voisins).*

Espacement

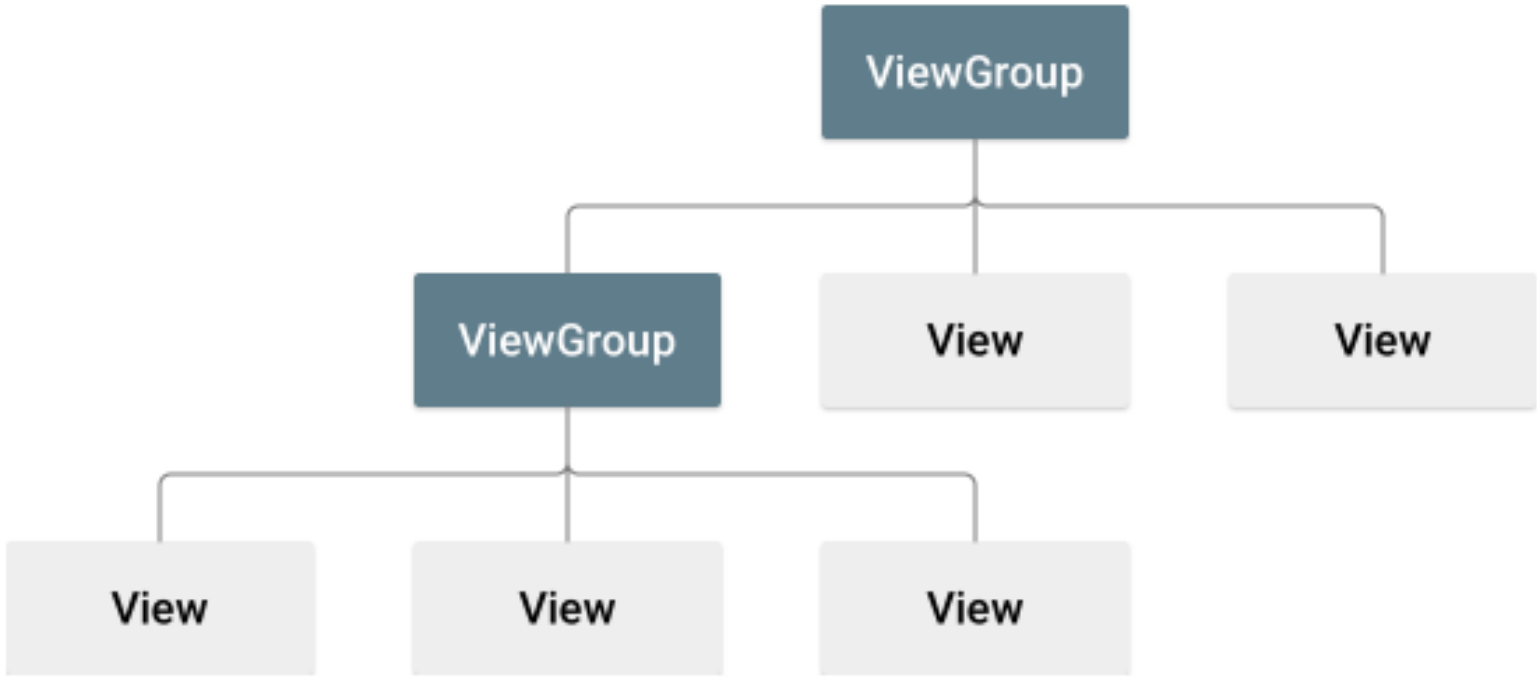
- Il est aussi possible de créer un décalage uniquement sur l'un des côtés du widget

- en **haut** avec ***paddingTop*** ***ou*** ***layout_marginTop***
- ou en **bas** avec ***paddingBottom*** ***ou*** ***layout_marginBottom***
- à **gauche** avec ***paddingLeft*** ***ou*** ***layout_marginLeft***,
-
- à **droite** avec ***paddingRight*** ***ou*** ***layout_marginRight***,

[Ref]

Les Conteneurs (les layouts)

- Une mise en page définit la structure d'une interface utilisateur dans votre application, par exemple dans une activité. Tous les éléments de layout sont construits en utilisant une hiérarchie d'objets *View* et *ViewGroup*.
- Une vue (*view*) est généralement quelque chose que l'utilisateur peut voir et interagir avec. Alors qu'un *ViewGroup* est un conteneur invisible qui définit la structure de disposition pour View et d'autres objets ViewGroup,



Les conteneurs (les Layouts)

- Un layout est un objet, qui va contenir d'autres layouts et/ou les éléments de l'interface graphique comme les **boutons**, les **Textview** ..... Et qui permet d'organiser la disposition de ces éléments sur l'écran.

- Les layouts sont des classes qui héritent toutes d'une même classe parente, la classe ***ViewGroup***.

- Il existe plusieurs types de layout :
  - **Linear Layout**,
  - **Relative Layout**,
  - **Table Layout**,
  - **Absolute Layout**
  - **Frame Layout**
  - **List View**
  - **Grid View**

chacun avec des caractéristiques différentes.

LinearLayout

- Linear Layout est un groupe de vues ***(ViewGroup)*** qui aligne tous les éléments enfants (conteneurs ou widgets) appartenant à ce Layout verticalement ou horizontalement, Le choix de la direction se fait avec l'attribut ***android: orientation.***

Attributs de LinearLayout

Attribut	Description
android:id	C'est l'identifiant qui identifie de façon unique le layout
android:baseline Aligned	Cela doit être une valeur booléenne, "true" ou "false" et empêche le layout d'aligner les lignes de base de ses enfants.
android:baselineAlig nedChildIndex	Lorsque LinearLayout fait partie d'un autre layout alignée sur la ligne de base, il peut spécifier lequel de ses enfants doit être aligné sur la ligne de base.
android:divider	Ceci est à utiliser comme un diviseur vertical entre les boutons. Vous utilisez une valeur de couleur.
android:gravity	Ceci spécifie comment un objet doit positionner son contenu, à la fois sur les axes X et Y.
android:orientation	Ceci spécifie la direction de l'arrangement. Vous utilisez "horizontal" pour une ligne, "vertical" pour une colonne.
android:weightSum	Somme des poids de l'enfant

LinearLayout : Exemple

- Suivez les étapes suivantes pour modifier l'application Android « Hello World » et pour montrer comment créer votre propre application Android en utilisant Linear Layout.

Etape	Description
1	Utiliserez Android Studio pour créer une application Android sous le nom '*Demo*' par exemple
2	Modifiez le contenu par défaut du fichier ***res / layout / activity_main.xml*** pour inclure quelques boutons dans le conteneur linéaire (LinearLayout).
3	Exécutez l'application pour lancer l'émulateur Android et vérifier le résultat des modifications effectuées dans l'application.

Exemple 1

- **MainActivity.java.**

```
public class MainActivity extends Activity {
   @Override
   protected void onCreate(Bundle savedInstanceState)
{ super.onCreate(savedInstanceState);
setContentView(R.layout.activity_main);
  }
}
```

Exemple 1

- **res/layout/activity_main.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" >

    <Button android:id="@+id/btnStartService"
        android:layout_width="270dp"
        android:layout_height="wrap_content"
        android:text="start_service"/>

    <Button android:id="@+id/btnPauseService"
        android:layout_width="270dp"
        android:layout_height="wrap_content"
        android:text="pause_service"/>

    <Button android:id="@+id/btnStopService"
        android:layout_width="270dp"
        android:layout_height="wrap_content"
        android:text="stop_service"/>

</LinearLayout>
```

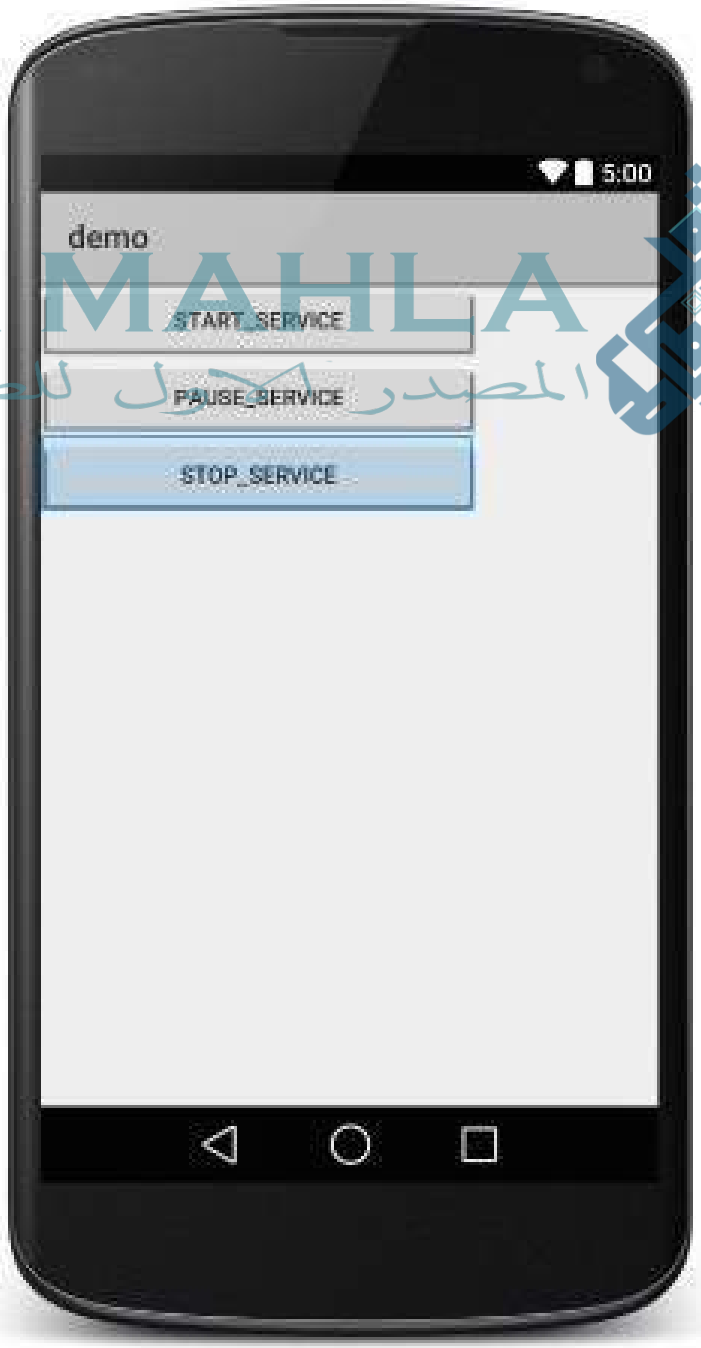
Exemple 1

- Essayons d'exécuter notre Hello World modifié!.

- Pour exécuter l'application à partir de Android studio, ouvrez l'un des fichiers d'activité de votre projet et cliquez sur l'icône *Run*.

- Android studio installe l'application sur votre *AVD* et la démarre et si tout va bien avec votre configuration et application, il affichera la fenêtre de l'émulateur suivante -

Exemple 1



Exemple 2

- Maintenant, changez l'orientation de Layout en **android: orientation = "horizontal"**

et essayez d'exécuter la même application, cela donnera l'écran suivant -

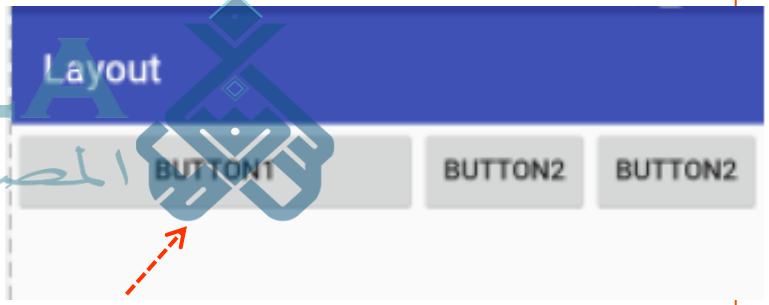
Poids

- ***android: layout_weight*** Permet d’attribuer un poids à élément en termes de la quantité d'espace qu‘il doit occuper sur l'écran.

- Le poids par défaut est **zéro**.

- Un poids plus important permet de se développer pour remplir tout espace restant dans la vue parent.

```xml
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android
"
    android:orientation="horizontal"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <Button
        android:text="Button1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_weight="1"/>

        <Button
            android:text="Button2"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content" />

        <Button
            android:text="Button2"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content" />
</LinearLayout>
```



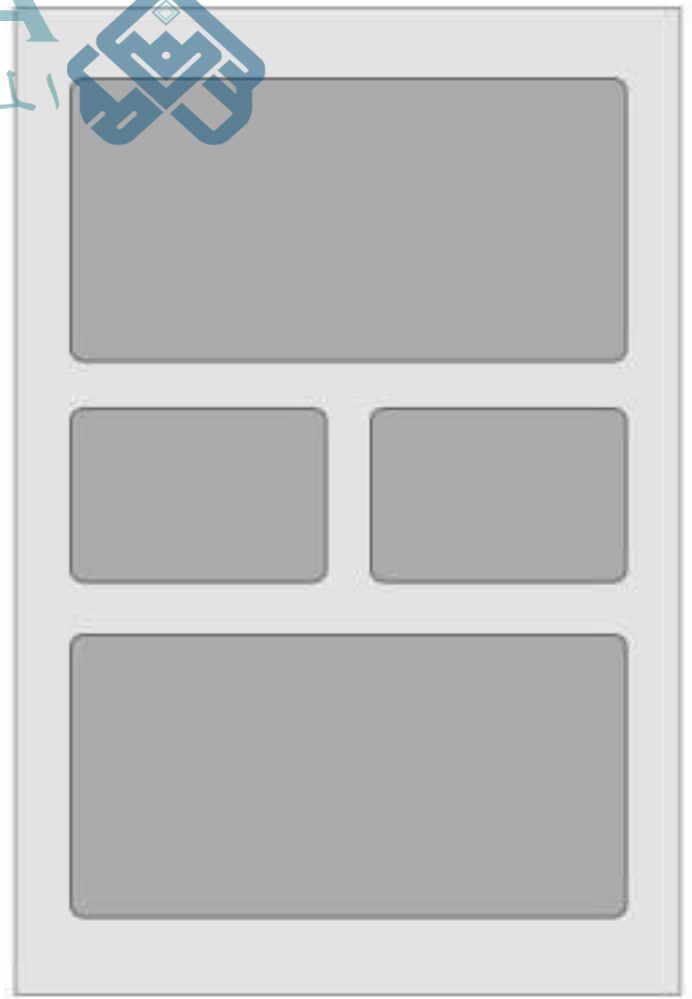
[Ref]

RelativeLayout

- permet de spécifier comment les vues enfants sont positionnées:

- les unes par rapport aux autres (par rapport aux éléments frères).

- ou relatifs au parent.



Attributs de RelativeLayout

Attribut	Description
android:id	C'est l'identifiant qui identifie de façon unique le layout
android:gravity	Ceci spécifie comment un objet doit positionner son contenu, à la fois sur les axes X et Y.
android:ignoreGravity	Cela indique quelle vue ne devrait pas être affectée par la gravité.

RelativeLayout

- En utilisant RelativeLayout, vous pouvez:
  - aligner deux éléments par une bordure droite
  - mettre un en dessous d'un autre,
  - centré sur l'écran,
  - centré à gauche, etc.

- Par défaut, toutes les vues enfants sont dessinées en haut à gauche de layout. Vous devez donc définir la position de chaque vue à l'aide des différentes propriétés disponibles dans *RelativeLayout.LayoutParams*

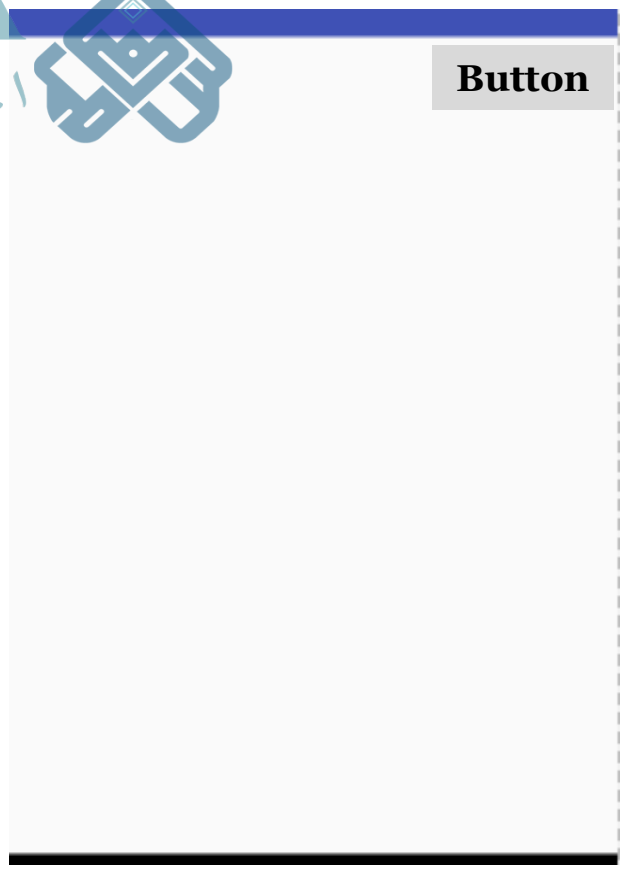
- Dans ce type de relation on lie un élément à son conteneur à l'aide d'un ensemble d'attributs.

Positionnement relatif au conteneur

Attribut	Description
android:layout_alignParentTop	Si la valeur est ***true***, le bord **supérieur** de l'élément correspond au bord **supérieur** du parent (son conteneur).
android:layout_alignParentBottom	Si la valeur est ***true***, correspondance des bords **inférieurs**.
android:layout_alignParentLeft	Si la valeur est ***true***, correspondance des bords **gauches**.
android:layout_alignParentRight	Si la valeur est ***true***, correspondance des bords **droits**.
android:layout_centerHorizontal	Si la valeur est ***true,*** centrer l'enfant **horizontalement** dans son conteneur.
android:layout_centerInParent	Si la valeur est ***true***, centrer l'enfant **horizontalement** et **verticalement** dans son conteneur.
android:layout_centerVertical	Si la valeur est ***true***, centrer l'enfant **verticalement** dans son conteneur.
android:layout_alignParentEnd	Si la valeur est ***true***, le bord de **fin** de l'élément correspond au bord de **fin** de son conteneur.
android:layout_alignParentStart	Si la valeur est ***true***, le bord de **début** de l'élément correspond au bord de **début** de son conteneur.

Exemple 1


```
<?xmlversion="1.0" encoding="utf-8"?>
<RelativeLayout
  xmlns:android=
  http://schemas.android.com/apk/res/android
  xmlns:tools=
   http://schemas.android.com/tools
  android:id="@+id/activ"
  android:layout_width="match_parent"
  android:layout_height="match_parent"
  tools:context="com.td1.cour4.MainActivity">
  <Button
    android:text="Button"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:id="@+id/button"
    android:layout_alignParentTop="true"
    android:layout_alignParentRight="true" />
</RelativeLayout>
```

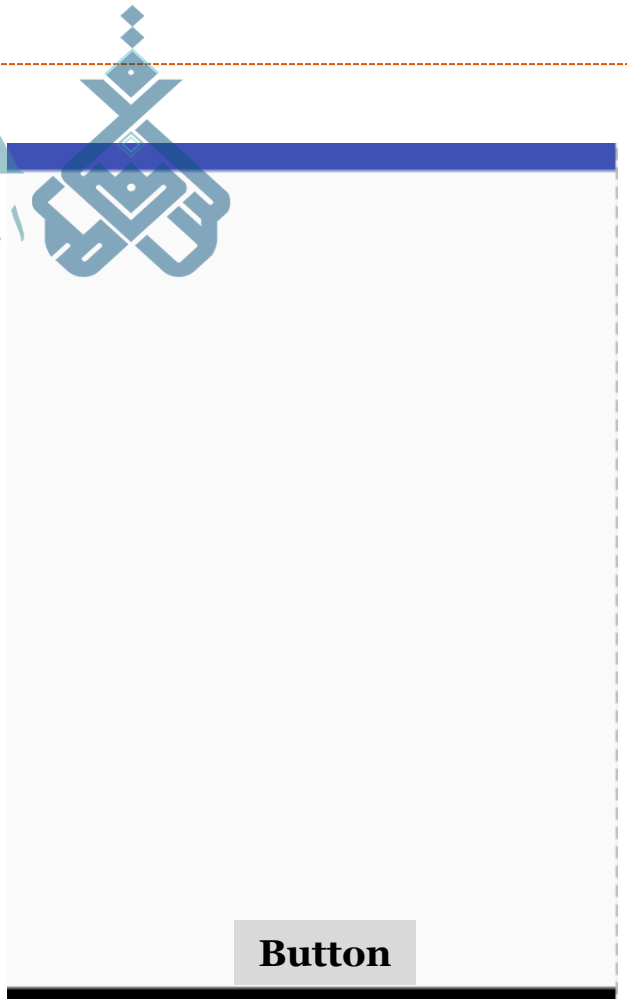


Exemple 2


```
<Button
  android:id="@+id/button"
  android:text="Button"
  android:layout_width="wrap_content"
  android:layout_height="wrap_content"
  android:layout_centerInParent="true"/>
```

Exemple 3


```
<Button
  android:id="@+id/button"
  android:text="Button"
  android:layout_width="wrap_content"
  android:layout_height="wrap_content"
  android:layout_centerHorizontal="true"
  android:layout_alignParentBottom="true"/>
```



Positionnement relatif aux autres éléments

Attribut	Description
android:layout_above="@+id/e"	Positionner l'élément **au-dessus** de l'élément référencé par son ID.
android:layout_below="@+id/e"	Positionner l'élément **au-desous** de l'élément référencé par son ID.
android:layout_toLeftOf="@+id/e"	Positionner l'élément **à gauche** de l'élément référencé par son ID.
android:layout_toRightOf="@+id/e"	Positionner l'élément **à droite** de l'élément référencé par son ID.
android:layout_alignTop="@+id/e"	Le bord **supérieur** de l'élément est aligné avec le bord supérieur de l'élément référencé par son ID.
android:layout_alignBottom="@+id/e"	Le bord **inférieur** de l'élément est aligné avec le bord **inférieur** de l'élément référencé par son ID.

Positionnement relatif aux autres éléments

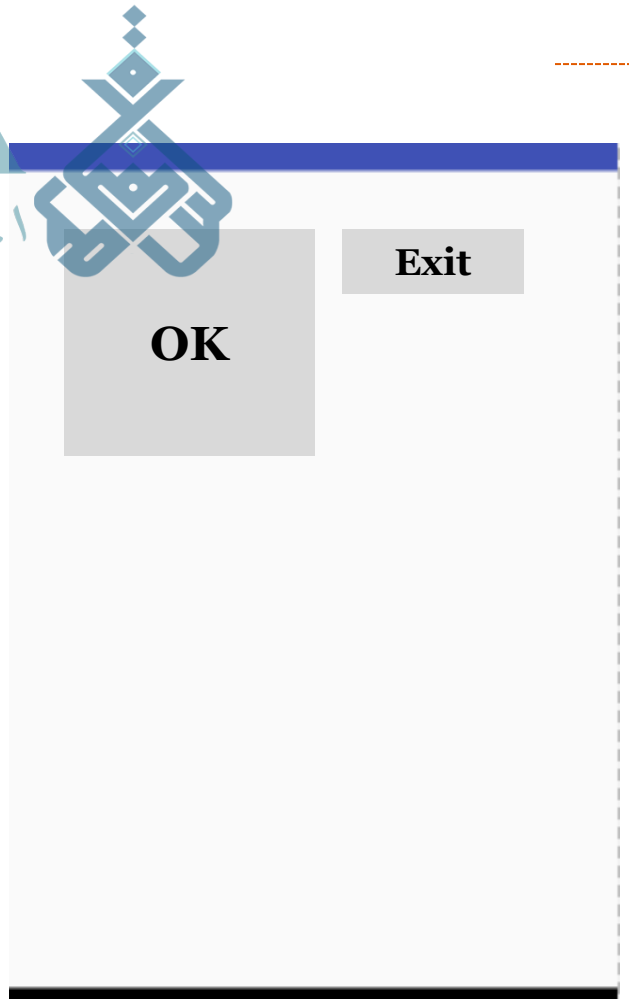
Attribut	Description
android:layout_alignLeft="@+id/e"	Le bord **gauche** de l'élément est aligné avec le bord **gauche** de l'élément référencé par son ID.
android:layout_alignRight="@+id/e"	Le bord **droit** de l'élément est aligné avec le bord **droit** de l'élément référencé par son ID.
android:layout_alignBaseLine="@+id/e"	Indique que les **lignes de base** des 2 éléments sont alignées
android:layout_alignStart="@+id/e"	le bord de **début** de l'élément corresponde au bord de **début** de l'élément référencé par son ID.
android:layout_toEndOf ="@+id/e"	le bord de **fin** de l'élément corresponde au bord de **fin** de l'élément référencé par son ID.

Exemple 1

```
<Button
  android:id="@+id/Ok_butt"
  android:text="OK"
  android:layout_width="wrap_content"
  android:layout_height="wrap_content"
  android:layout_alignParentLeft="true"
  android:layout_alignParentStart="true"
  android:layout_margingTop="20dp"
  android:padding="50dp"/>
<Button
  android:id="@+id/Exit_butt"
  android:text="Exit"
  android:layout_width="wrap_content"
  android:layout_height="wrap_content"
 android:layout_toRightOf="@+id/Ok_butt"  />
```

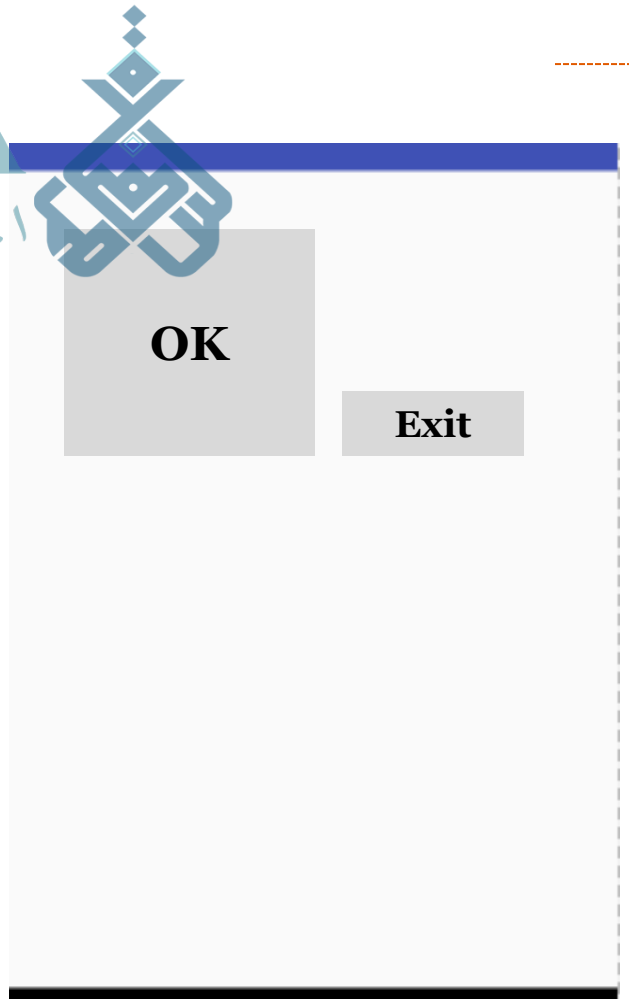
Exemple 2

```
<Button
  android:id="@+id/Ok_butt"
  android:text="OK"
  android:layout_width="wrap_content"
  android:layout_height="wrap_content"
  android:layout_alignParentLeft="true"
  android:layout_alignParentStart="true"
  android:layout_margingTop=“20dp“
  android:padding=“50dp"/>
<Button
  android:id="@+id/Exit_butt"
  android:text=“Exit"
  android:layout_width="wrap_content"
  android:layout_height="wrap_content“
 android:layout_toRightOf="@+id/Ok_butt“
 android:layout_alignTop="@+id/Ok_butt"  />
```



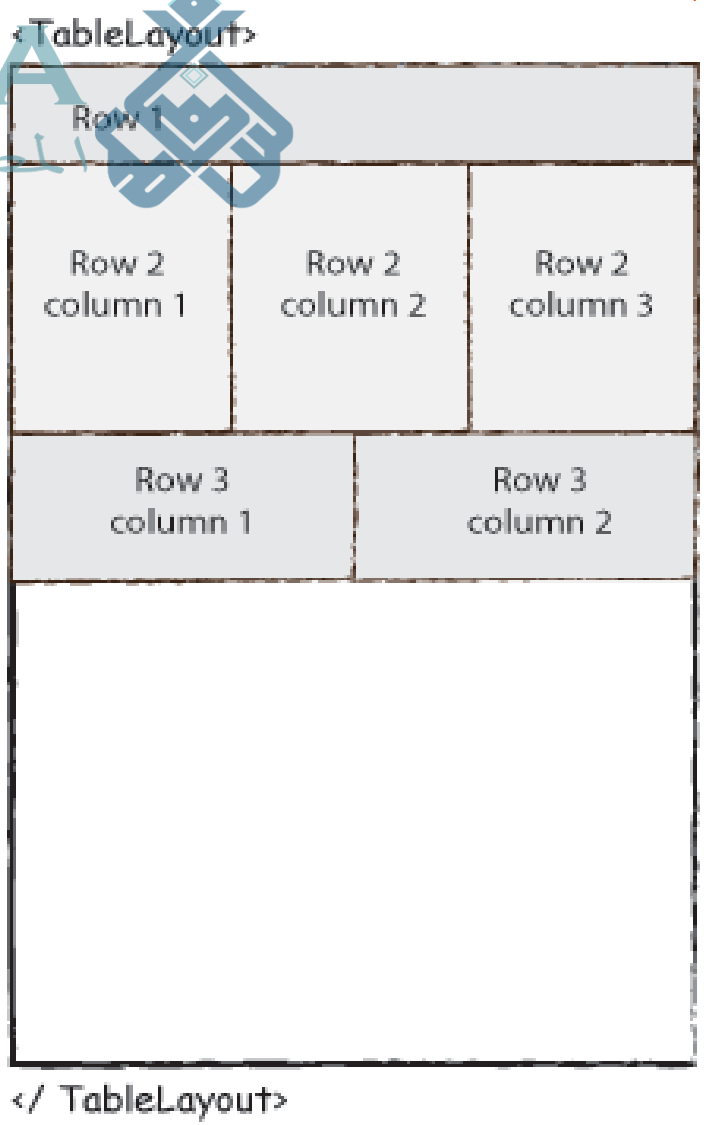
Exemple 3

```
<Button
  android:id="@+id/Ok_butt"
  android:text="OK"
  android:layout_width="wrap_content"
  android:layout_height="wrap_content"
  android:layout_alignParentLeft="true"
  android:layout_alignParentStart="true"
  android:layout_margingTop="20dp"
  android:padding="50dp"/>
<Button
  android:id="@+id/Exit_butt"
  android:text="Exit"
  android:layout_width="wrap_content"
  android:layout_height="wrap_content"
 android:layout_toRightOf="@+id/Ok_butt"
 android:layout_alignBottom="@+id/Ok_butt"  />
```



TableLayout

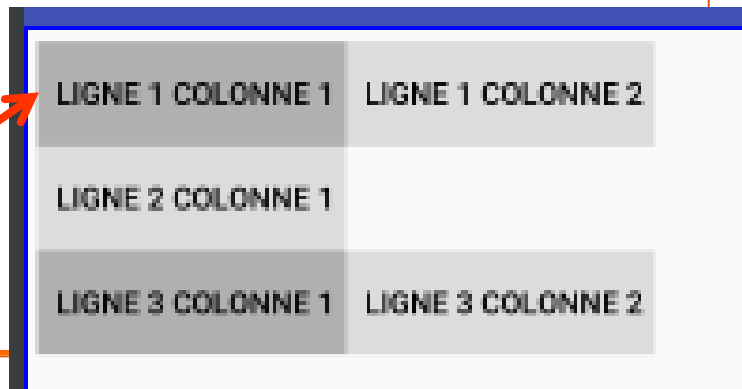
- **Android TableLayout** permet d'organiser des groupes de vues en lignes et en colonnes.
- L'attribut ***<TableRow>*** est utilisé pour créer une ligne dans la table.
- Chaque ligne a zéro ou plusieurs cellules; chaque cellule peut contenir un objet ***View***.
- Les conteneurs ***TableLayout*** n'affichent pas de lignes de bordure pour leurs lignes, colonnes ou cellules.
- Nous pouvons également imbriquer un autre TableLayout dans une cellule de tableau.



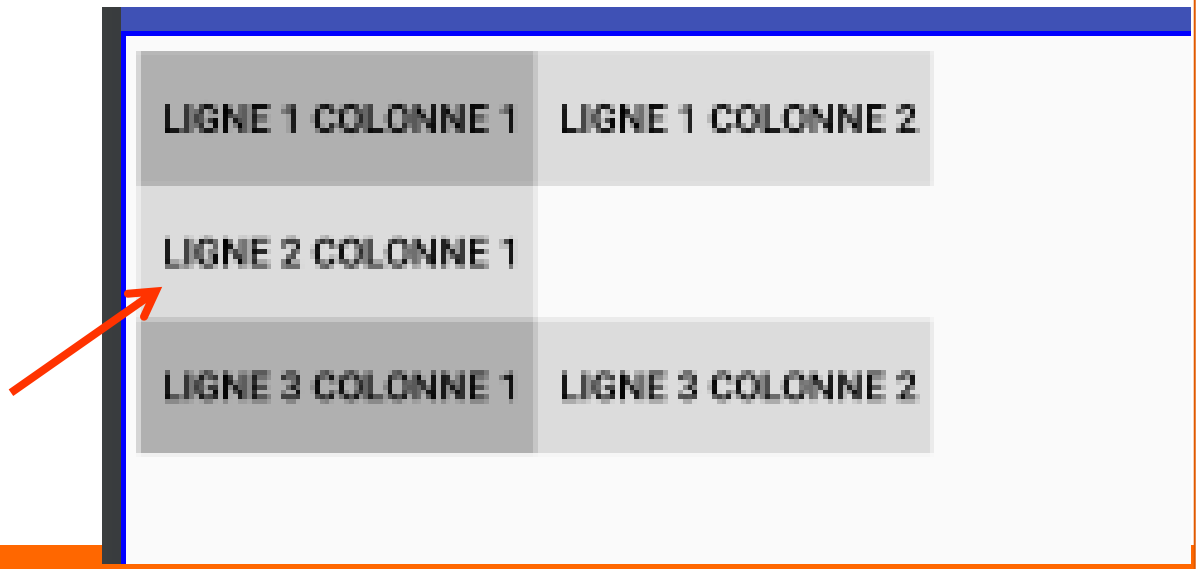
Attributs de TableLayout

Attribut	Description
android:id	C'est l'identifiant qui identifie de façon unique le layout
android:collapseColumns	Permet à la (aux) colonne (s) d'être réduite ou de devenir invisible . Les index de colonne doivent être séparés par une virgule: 1, 3, 6.
android:shrinkColumns	Permet de réduire la largeur de la colonne. Les index de colonne doivent être séparés par une virgule: 1, 3, 6.
android:stretchColumns	Permet d'étendre la (les) colonne (s) pour occuper l'espace libre disponible. Les index de colonne doivent être séparés par une virgule: 1, 3, 6.

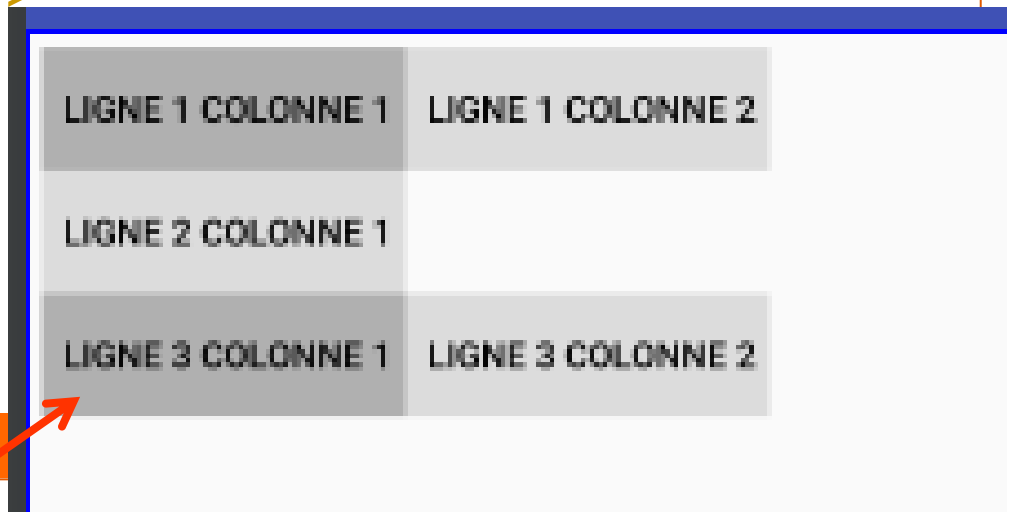
```xml
<?xml version="1.0" encoding="utf-8"?>
<TableLayout
xmlns:android="http://schemas.android.com/apk/res/android"
   android:layout_width="match_parent"
   android:layout_height="match_parent"
   android:padding="6dp">

   <TableRow
      android:layout_height="wrap_content"
      android:layout_width="match_parent">

      <Button
         android:layout_width="match_parent"
         android:layout_height="wrap_content"
         android:text=" Ligne 1 Colonne 1"
         android:background="#bobobo"
         android:padding="5dp"/>

      <Button
         android:layout_width="match_parent"
         android:layout_height="wrap_content"
         android:text=" Ligne 1 Colonne 2"
         android:background="#dcdcdc"
         android:padding="5dp"/>

   </TableRow>
```

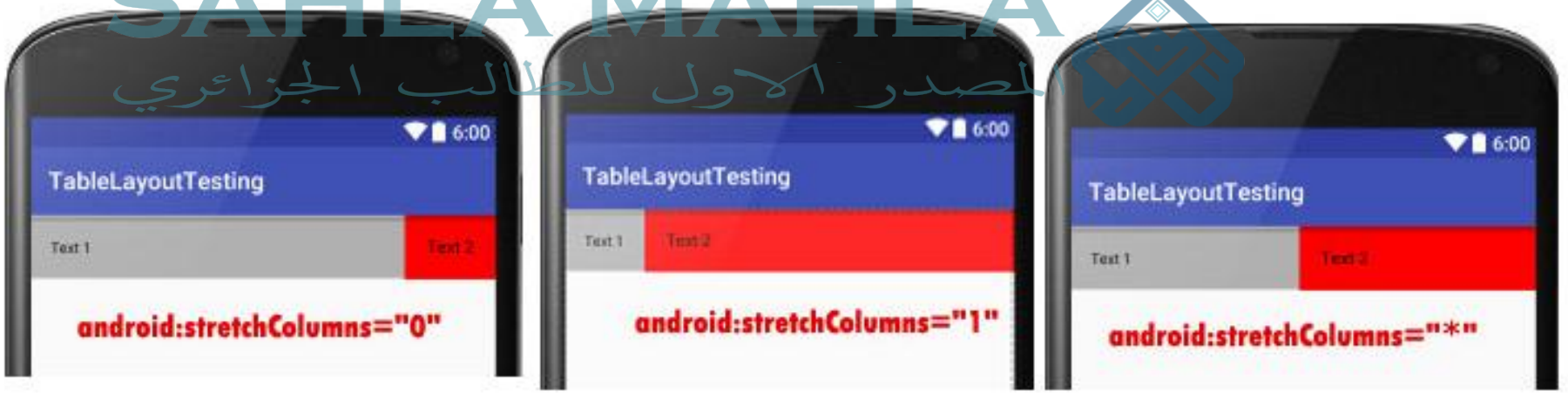
```
<TableRow
    android:layout_height="wrap_content"
    android:layout_width="match_parent">

    <Button
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text=" Ligne 2 Colonne 1"
        android:background="#dcdcdc"
        android:padding="5dp"/>

</TableRow>
```

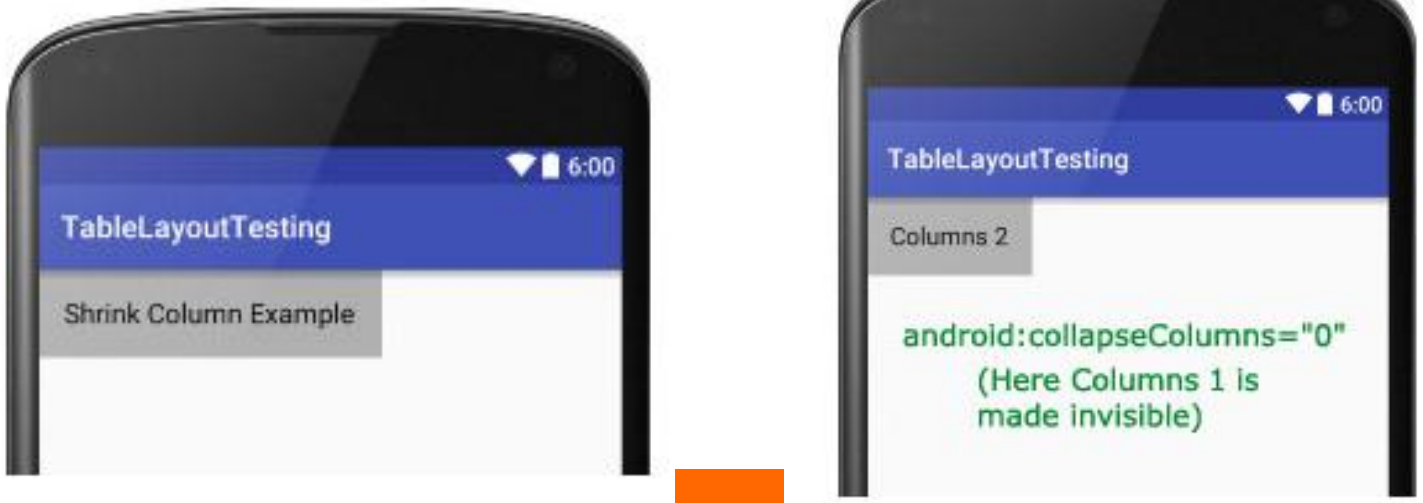



```
<TableRow
    android:layout_height="wrap_content"
    android:layout_width="match_parent">

    <Button
      android:layout_width="match_parent"
      android:layout_height="wrap_content"
      android:text=" Ligne 1 Colonne 1"
      android:background="#bobobo"
      android:padding="5dp"/>

    <Button
      android:layout_width="match_parent"
      android:layout_height="wrap_content"
      android:text=" Ligne 1 Colonne 2"
      android:background="#dcdcdc"
      android:padding="5dp"/>

  </TableRow>

</TableLayout>
```

Exemples

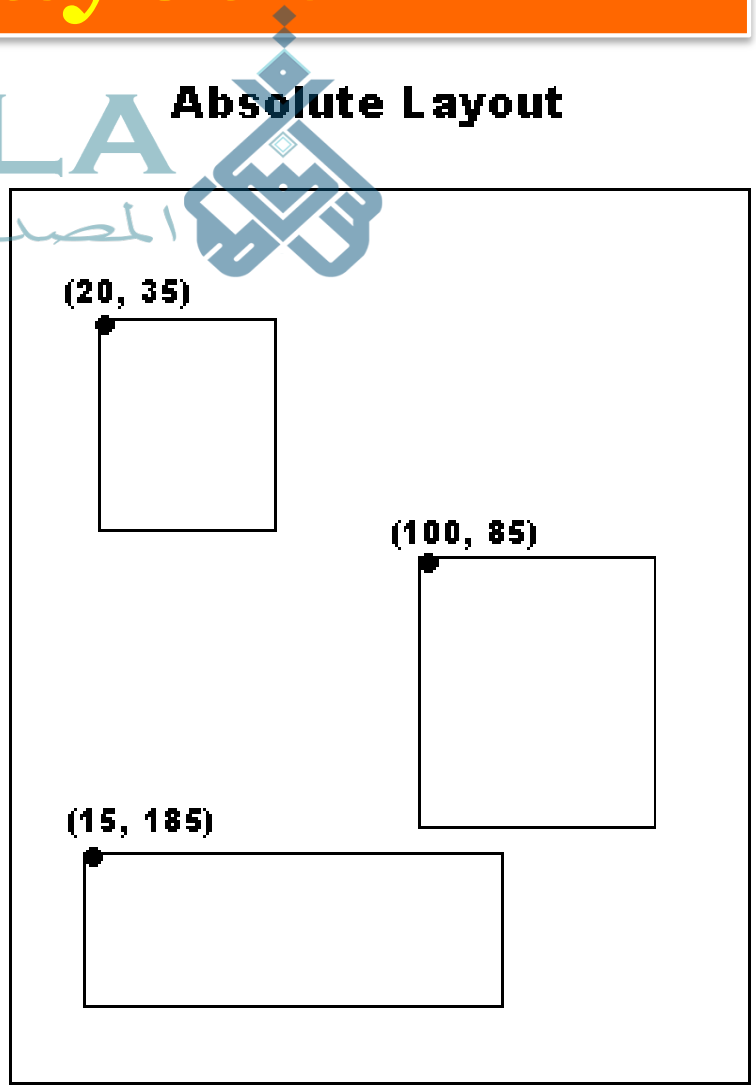




AbsoluteLayout

- Une mise en page absolue vous permet de spécifier les emplacements exacts (***coordonnées x / y***) de ses enfants. Les mises en page absolues sont moins flexibles et plus difficiles à maintenir que les autres types de mise en page sans positionnement absolu.

- c'est une mise en page utilisée pour concevoir les mises en page personnalisées. Dans cette disposition, vous pouvez spécifier l'emplacement exact de ses enfants en utilisant les coordonnées x et y.



Remarque importante:

- Absolute Layout est plus difficile à maintenir pour les différentes tailles d'écran mobiles car nous définissons l'emplacement exact d'une vue enfant ou d'un composant.

- Le positionnement est basé sur les coordonnées ***x*** et ***y*** et ce positionnement n'est pas aussi utile dans le monde de diverses résolutions d'écran.

Attributs de AbsoluteLayout

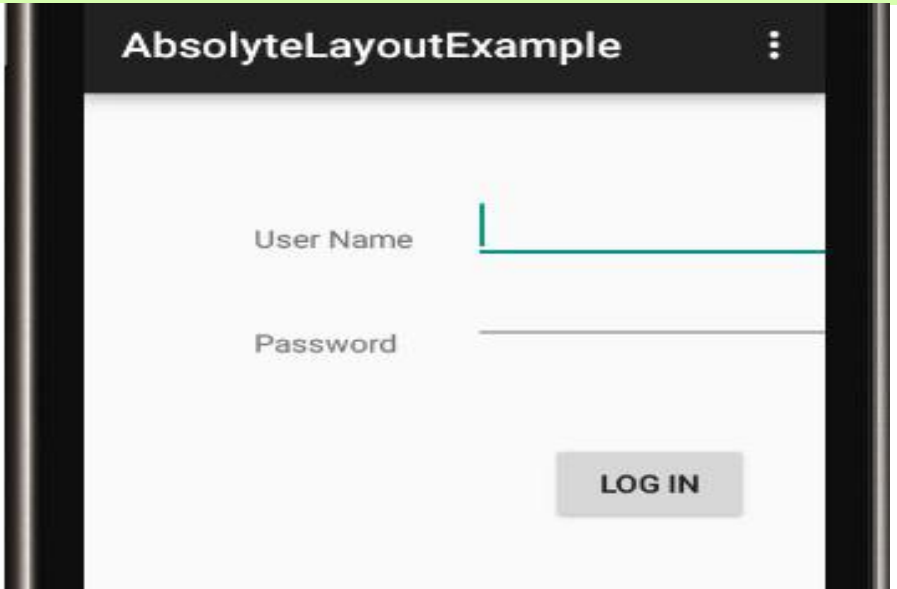
Attribut	Description
android:id	C'est l'identifiant qui identifie de façon unique le layout
android:layout_x	Ceci spécifie la coordonnée **x** de la vue.
android:layout_y	Ceci spécifie la coordonnée **y** de la vue.

Exemple 1

```
<AbsoluteLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent">
    <Button
        android:layout_width="100dp"
        android:layout_height="wrap_content"
        android:text="OK"
        android:layout_x="50px"
        android:layout_y="361px" />
    <Button
        android:layout_width="100dp"
        android:layout_height="wrap_content"
        android:text="Cancel"
        android:layout_x="225px"
        android:layout_y="361px" />

</AbsoluteLayout>
```


```
<AbsoluteLayout

xmlns:android="http://schemas.android.com/
apk/res/android"
   android:layout_width="fill_parent"
   android:layout_height="fill_parent">
    <TextView
       android:layout_x="110px"
       android:layout_y="110px"
       android:text="User Name"
       android:layout_width="wrap_content"
       android:layout_height="wrap_content" />
   <EditText
      android:layout_x="250px"
      android:layout_y="80px"
      android:width="100px"
      android:layout_width="200dp"
      android:layout_height="wrap_content" />
  <TextView
     android:layout_x="110px"
     android:layout_y="200px"
     android:text="Password"
     android:layout_width="wrap_content"
     android:layout_height="wrap_content" />
```

```
<EditText
    android:layout_x="250px"
    android:layout_y="150px"
    android:width="100px"
    android:layout_width="200dp"
    android:layout_height="wrap_content"
/>
 <Button
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Log In"
    android:layout_x="300px"
    android:layout_y="300px"/>
</AbsoluteLayout>
```



FrameLayout

- Frame Layouts sont l'un des types de conteneur les plus simples et les plus efficaces utilisés par les développeurs Android pour organiser les contrôles d'affichage. Ils sont utilisés moins souvent que d'autres layouts, simplement parce qu'ils sont généralement utilisés pour afficher une seule vue ou des vues qui se chevauchent.
- FrameLayout est souvent utilisée comme une disposition de conteneur, car elle n'a généralement qu'une seule vue enfant (souvent un autre layout, utilisé pour organiser plus d'une vue)

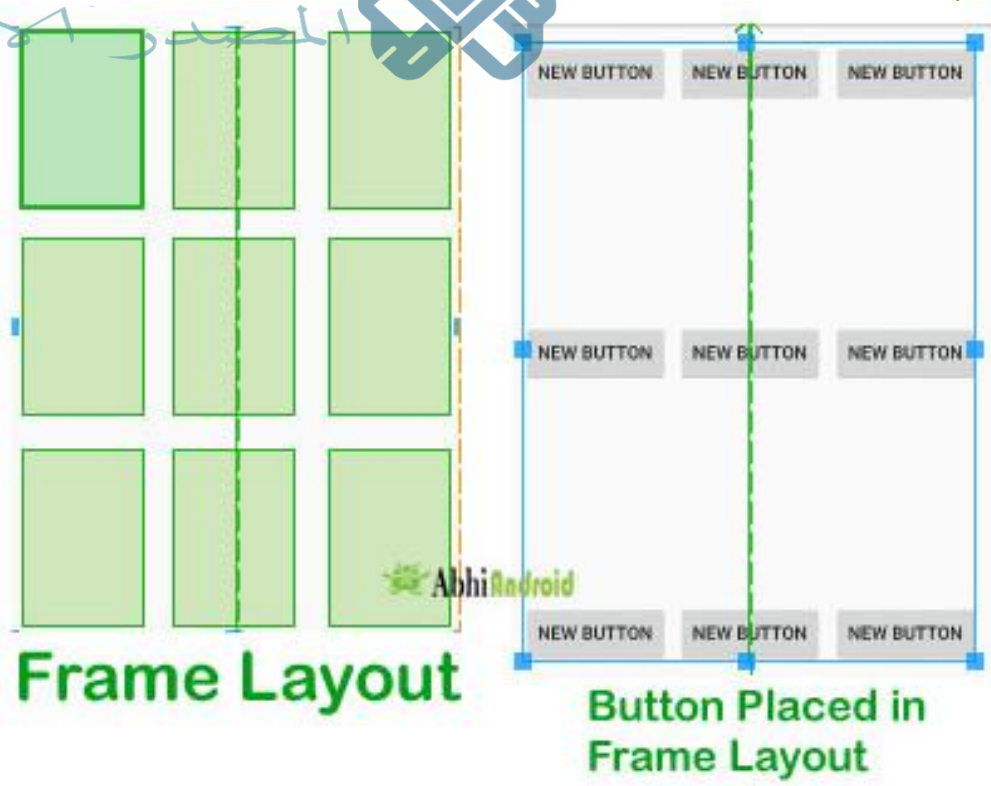
FRAME LAYOUT

- Vous pouvez cependant ajouter plusieurs enfants à un FrameLayout et contrôler leur position dans le FrameLayout en affectant la gravité à chaque enfant.

FrameLayout

- FrameLayouts sont conçus pour bloquer une zone sur l'écran. FrameLayout doit être utilisé pour maintenir la vue enfant, car il peut être difficile d'afficher des vues uniques dans une zone spécifique de l'écran sans se chevaucher.

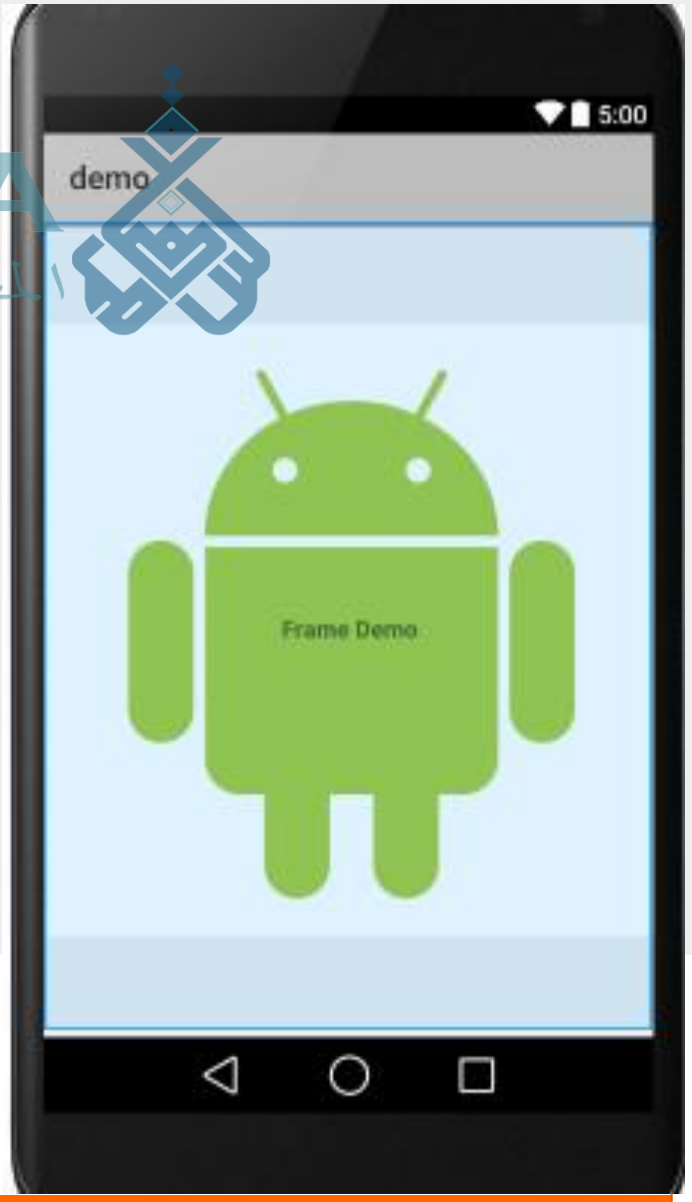
- Nous pouvons ajouter plusieurs enfants à un FrameLayout et contrôler leur position en affectant la gravité à chaque enfant, en utilisant l'attribut *android: layout_gravity*.



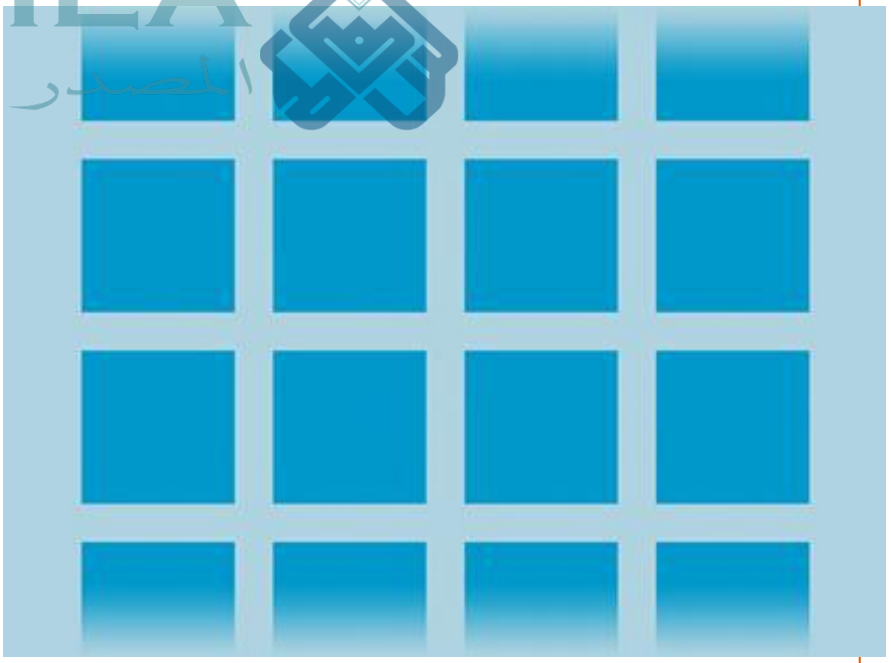
Attributs de FrameLayout

Attribut	Description
android:id	C'est l'identifiant qui identifie de façon unique le layout
android:foreground	Définit le drawable pour dessiner le contenu et les valeurs possibles peuvent être une valeur de couleur, sous la forme de "#rgb", "#argb", "#rrggbb" ou "#aarrggbb".
android:foregroundGravity	Définit la gravité à appliquer au premier plan drawable. Les valeurs possibles sont top, bottom, left, right, center, center_vertical, center_horizontal etc.
android:measureAllChildren	Détermine s'il faut mesurer tous les enfants ou uniquement ceux qui sont dans l'état VISIBLE ou INVISIBLE lors de la mesure. Par défaut '*false*'.

```xml
<FrameLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent">

    <ImageView
        android:src="@drawable/ic_launcher"
        android:scaleType="fitCenter"
        android:layout_height="250px"
        android:layout_width="250px"/>

    <TextView
        android:text="Frame Demo"
        android:textSize="30px"
        android:textStyle="bold"
        android:layout_height="fill_parent"
        android:layout_width="fill_parent"
        android:gravity="center"/>
</FrameLayout>
```



GridLayout



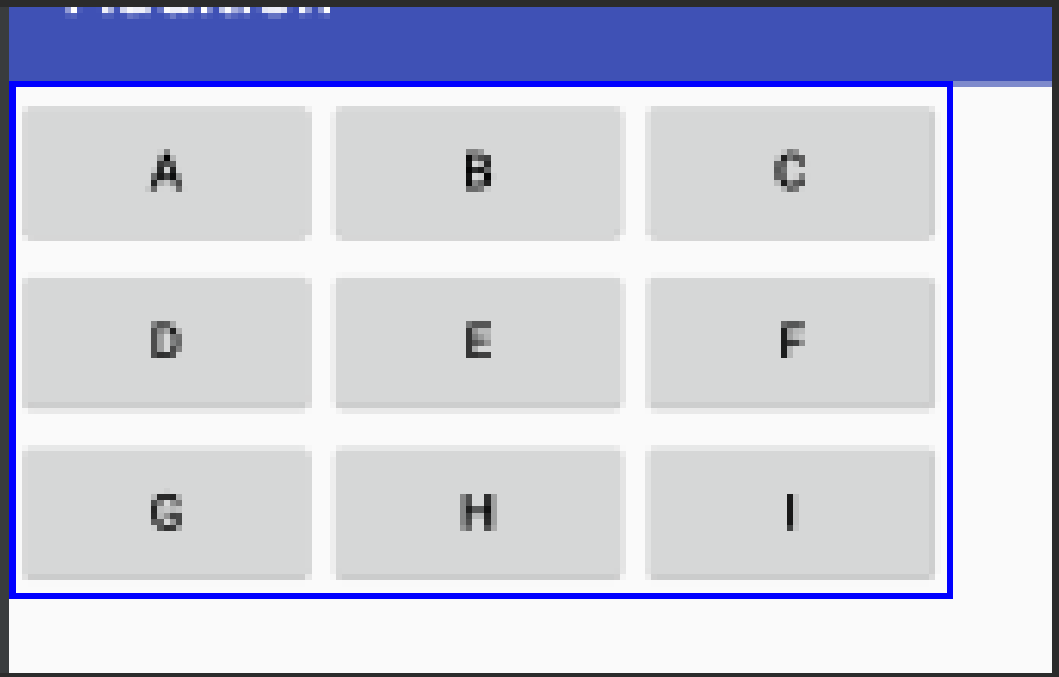
- Affiche les éléments dans une grille bidimensionnelle (lignes et colonnes)
- Les éléments de la grille ne sont pas nécessairement prédéterminés, ils sont automatiquement insérés dans la mise en page à l'aide d'un **ListAdapter**.

- Deux options nécessaire:
  - **android:columnCount** correspond au nombre de lignes sur la grille.
  - **android:rowCount** correspond au nombre de colonnes sur lagrille.

Attributs de GridLayout

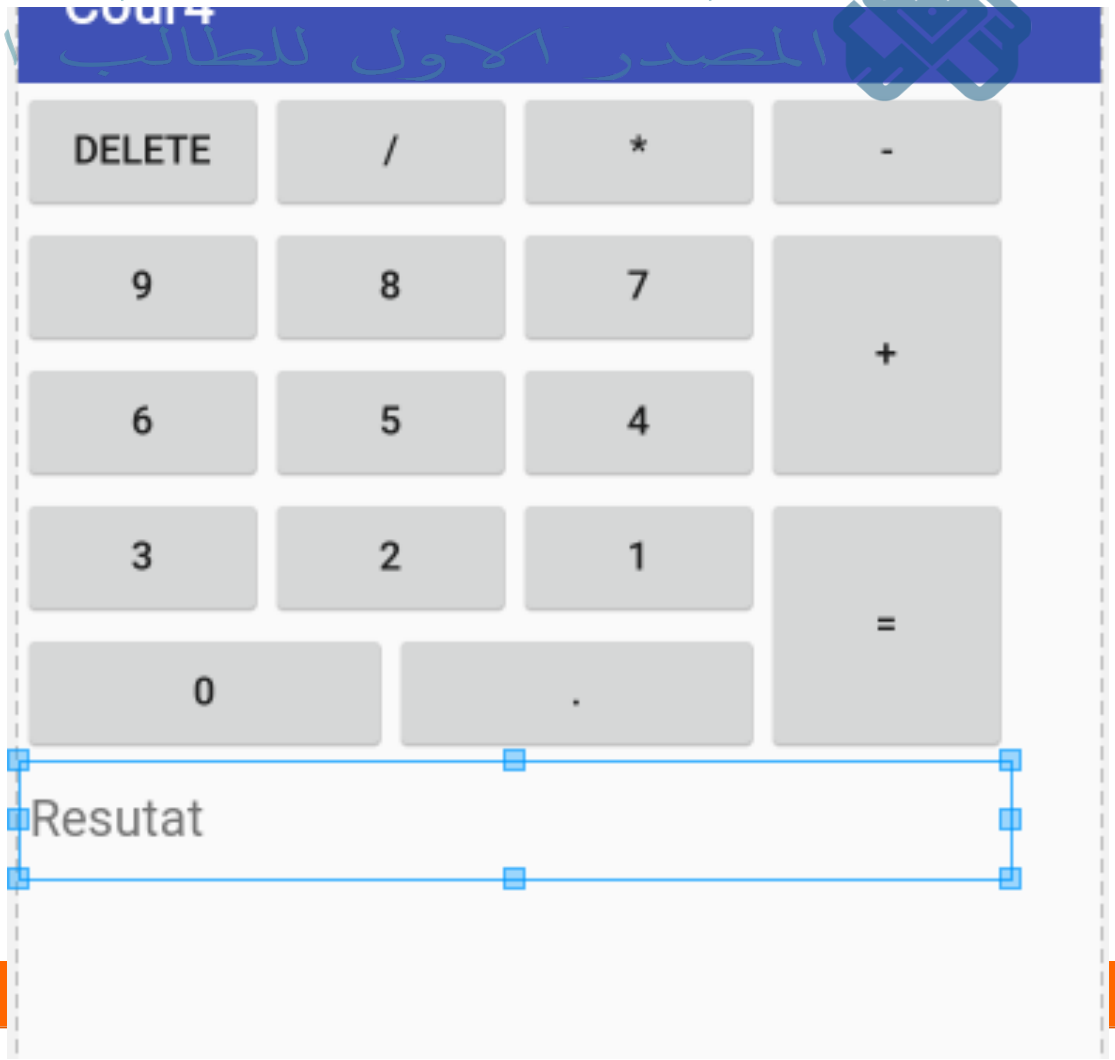
Attribut	Description
android:id	C'est l'identifiant qui identifie de façon unique le layout
android:columnWidth	Ceci spécifie la largeur pour chaque colonne: Pourrait être en px, dp, sp, in, mm.
android:gravity	Spécifie la gravité dans chaque cellule.
android:horizontalSpacing	Définit l'espacement horizontal par défaut entre les colonnes. Cela pourrait être en px, dp, sp, in ou mm.
android:numColumns	Définit le nombre de colonnes à afficher. soit une valeur entière, telle que « 50 » ou « *auto_fit* », ce qui signifie afficher autant de colonnes que possible pour remplir l'espace disponible.
android:stretchMode	Définit comment les colonnes doivent s'étirer pour remplir l'espace vide disponible. Cela doit être l'une des valeurs - *none*, *spacingWidth* , *columnWidth*, *spacingWidthUniform*
android:verticalSpacing	Définit l'espacement vertical par défaut entre les lignes. Cela pourrait être en px, dp, sp, in ou mm.

```xml
<?xml version="1.0" encoding="utf-8"?>
<GridLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/GridLayout1"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:columnCount="3"
    android:rowCount="3"
    android:orientation="horizontal">

    <Button android:text="A" />
    <Button
        android:text="B" />
    <Button
        android:text="C" />
    <Button
        android:text="D" />
    <Button
        android:text="E" />
    <Button
        android:text="F" />
    <Button
        android:text="G" />
    <Button
        android:text="H" />
    <Button
        android:text="I" />

</GridLayout>
```

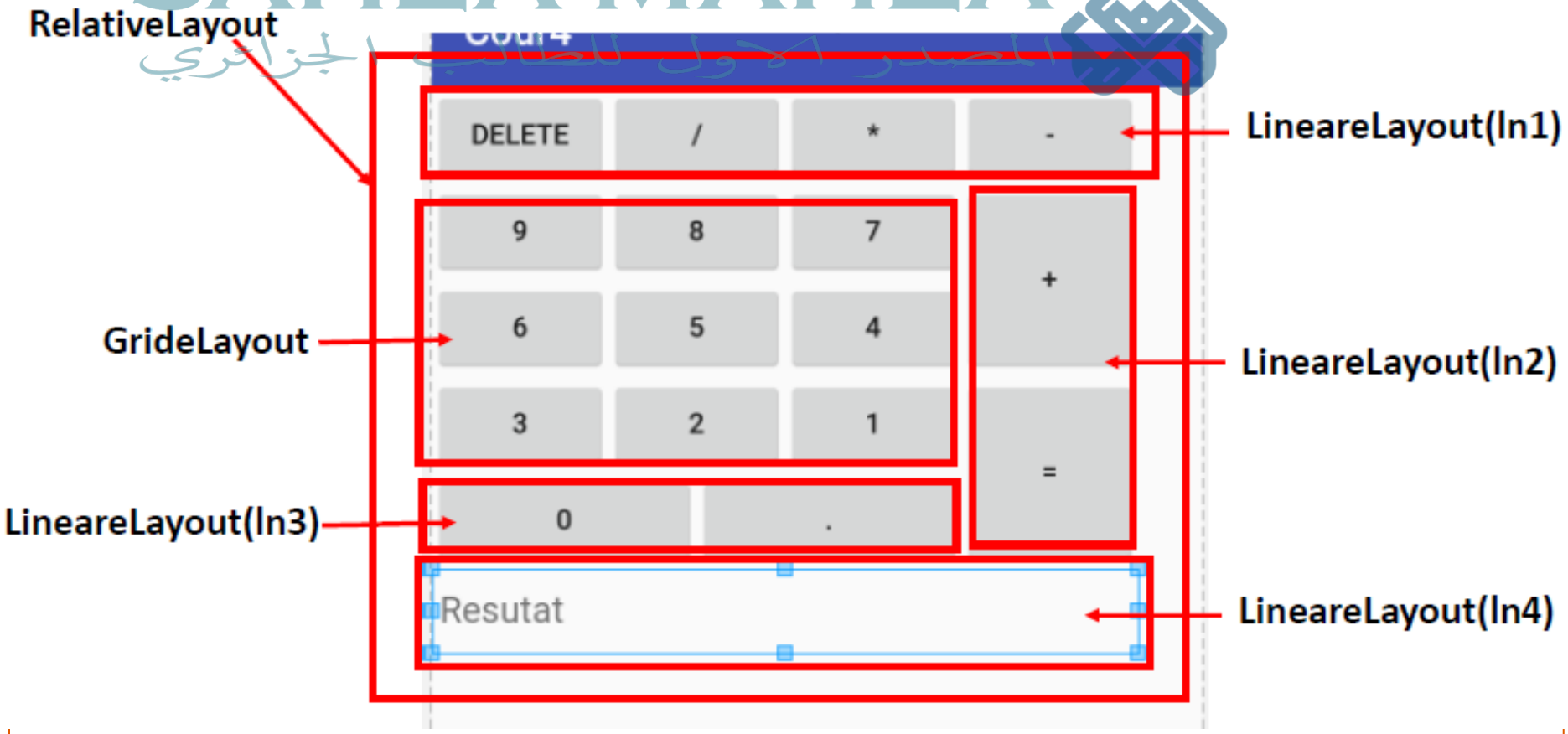



Exercice

- Ecrire le code XML décrivant l'interface graphique



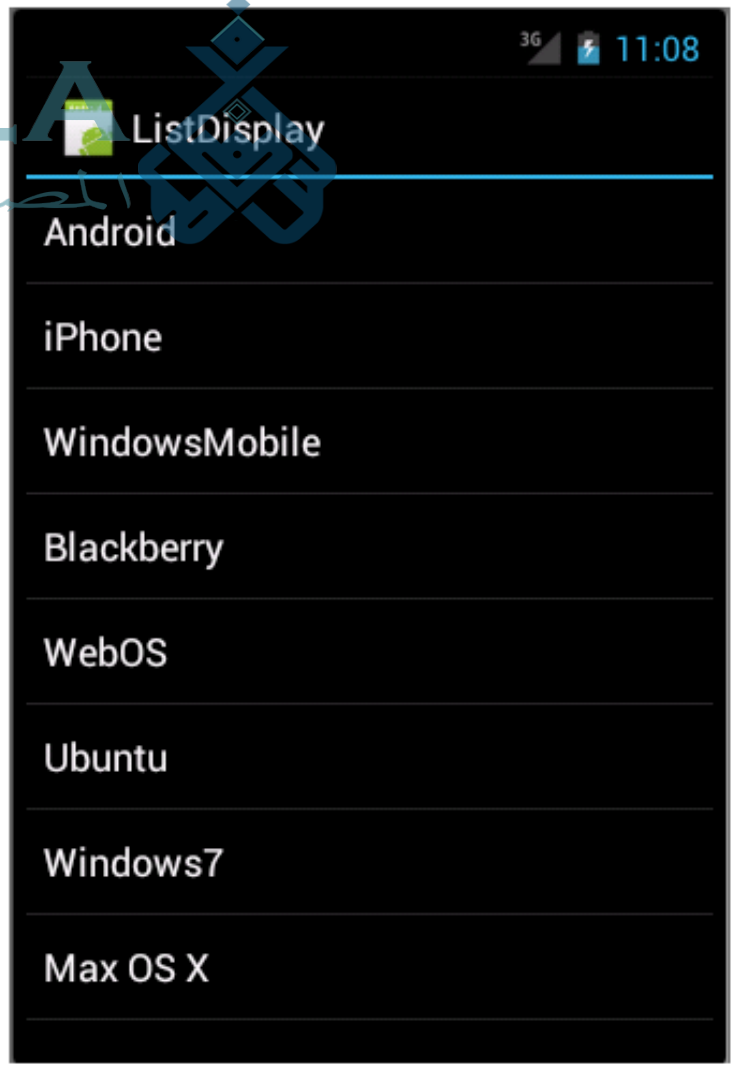
Exercice



ListView

- Android ListView est une vue qui regroupe plusieurs éléments et les affiche dans une liste déroulante verticale.

- Les éléments de la liste sont automatiquement insérés dans la liste à l'aide d'un adaptateur qui extrait le contenu d'une source telle qu'un tableau ou une base de données.



LIST VIEW

ListView

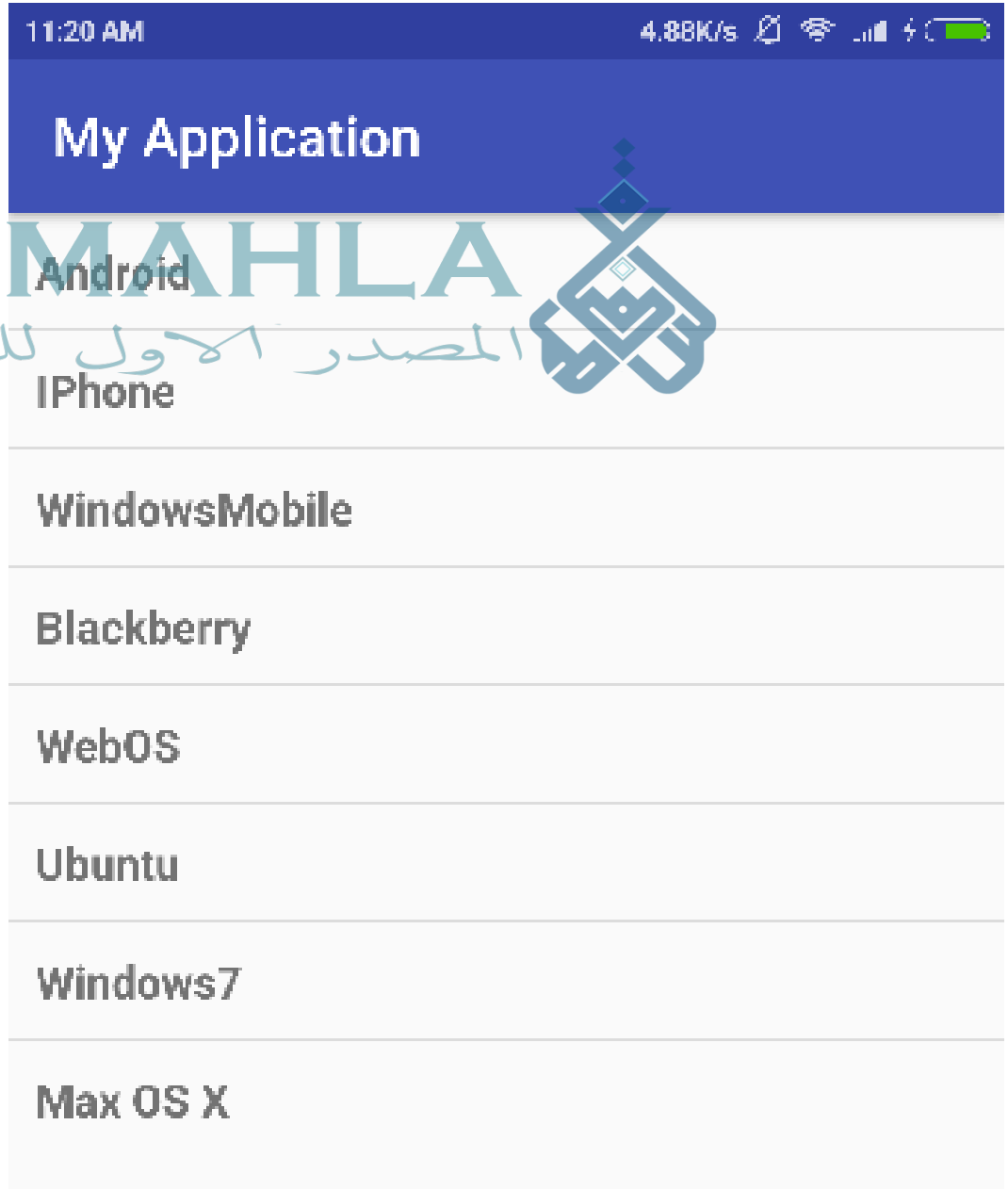
- Un adaptateur établit un pont entre les composants de l'interface utilisateur et la source de données qui remplit les données dans le composant de l'interface utilisateur.

- **ListView** et **GridView** sont des sous-classes de **AdapterView** et peuvent être remplies en les liant à un **adaptateur**, qui récupère les données d'une source externe et crée une vue qui représente chaque entrée de données.

- Android fournit plusieurs sous-classes d'adaptateur qui sont utiles pour récupérer différents types de données et créer des vues pour un AdapterView (par exemple ListView ou GridView).

- Les adaptateurs communs sont **ArrayAdapter**, **BaseAdapter**, **CursorAdapter**, **SimpleCursorAdapter**, **SpinnerAdapter** et **WrapperListAdapter**.

Exemple 1



Les Widgets de base

- Les widgets sont un aspect essentiel de la personnalisation de l'écran d'accueil. Ils sont des objets des données et des fonctionnalités les plus importantes d'une application, accessibles directement à partir de l'écran d'accueil de l'utilisateur.

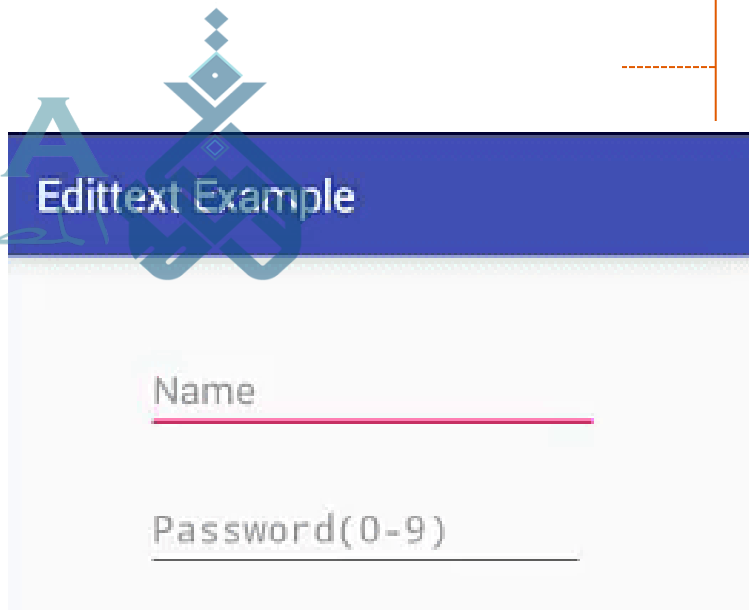
- Les utilisateurs peuvent déplacer des widgets sur leurs panneaux d'écran d'accueil et, s'ils sont pris en charge, les redimensionner pour adapter la quantité d'informations dans un widget à leur préférence.

- Chaque type de widgets est une classe qui dérive de la classe "View".

- Ils sont placé systématiquement dans des layouts.

○ EditText

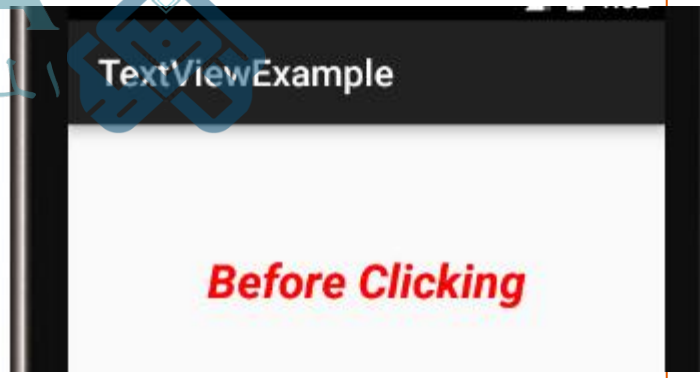
```
<EditText android:id="@+id/editText1“
    android:layout_width="wrap_content“
    android:layout_height="wrap_content"
    android:ems="10"
    android:hint="name"
    android:inputType="textPersonName" />

<EditText android:id="@+id/editText2"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:ems="10"
    android:hint="password_0_9"
    android:inputType="numberPassword" />
```

- L'attribut **"android:hint"** permet d'afficher un commentaire
- L'attribut **"android:inputType"** permet de choisir un type d'entrée (text, mot de passe, nombre, date, ...)
- L'attribut **"android:tex"** permet d'initialiser le champ avec une valeur par défaut
- La méthode ***getText()*** permet de récupérer le texte saisi
- La méthode ***setText()*** permet de placer un texte

○ TextView

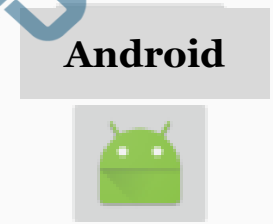
```
<TextView android:id="@+id/TextView1“
    android:layout_width="wrap_content“
    android:layout_height="wrap_content"
    android:layout_centerHorizontal="true"
    android:text="Before Clicking"
    android:textColor="#f00"
    android:textSize="25sp"
    android:textStyle="bold|italic"
    android:layout_marginTop="50dp" />
```



- L'attribut " **android:textStyle** " permet de choisir le style du texte, peut prendre les valeurs **“normal"** , **"bold"** , **"italic"**
- L'attribut **"android:typeface** " permet de choisir la police de caractères, peut prendre les valeurs **"sans"**, **"serif"**, **"monospace"**
- L'attribut " **android:textSize** " permet de choisir la taille du texte
- L'attribut " **android:textColor** " permet de choisir la couleure du texte
- .....

➢ Button/ImageButton


```
<Button android:id="@+id/But1 "
    android:layout_width="wrap_content "
    android:layout_height="wrap_content"
     android:gravity= " center"
     android:text= " Android" />

<ImageButton android:id="@+id/ImgBut1 "
    android:layout_width="wrap_content "
    android:layout_height="wrap_content"
     android:gravity= " center"
     android:srcCompat= " @mipmap/ic_launcher" />
```



○ CheckBox

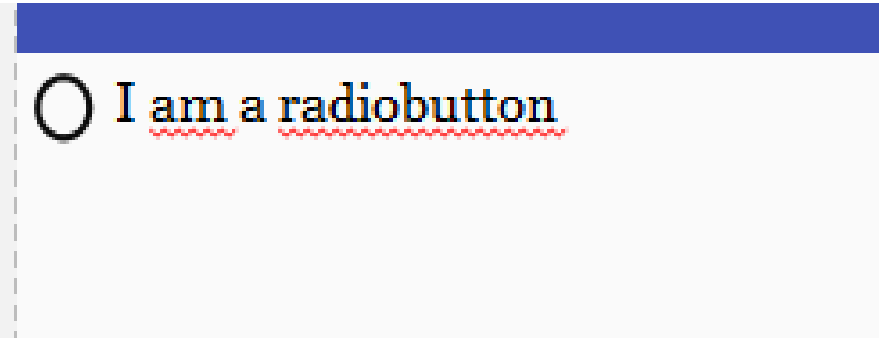
```
<CheckBox  android:id="@+id/ChBox1"
    android:layout_width=“wrap_content "
    android:layout_height="wrap_content"
    android:text=" Android"
    android:textColor="#44f“
    android:textStyle="bold|italic" />

<CheckBox  android:id="@+id/ChBox2"
    android:layout_width=“wrap_content "
    android:layout_height="wrap_content"
    android:text= " Java"
    android:textColor="#f44“
    android:textStyle="bold|italic" />
```


- Pour vérifier si un CheckBox est coché:

```
CheckBox ch= (CheckBox) this.findViewById (R.id.ChBox1);

boolean     b=ch.isChecked();
```

○ RadioButton


```
<RadioButton android:id="@+id/RBut1 "
    android:layout_width=“match_parent "
    android:layout_height="wrap_content"
    android:text= " I am a radiobutton" />
```

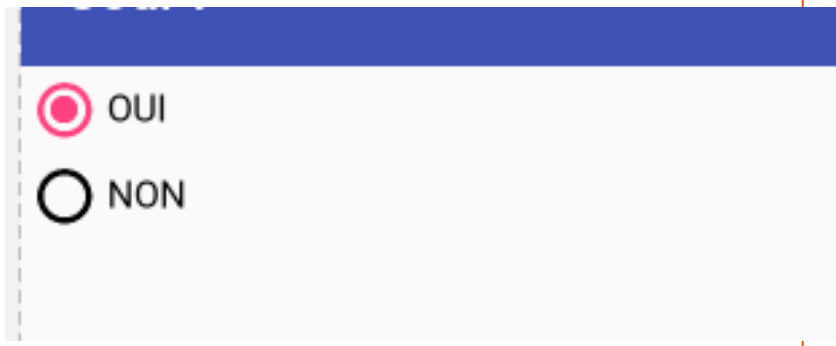


- Pour vérifier si un RadioButtonest coché:

```
RadioButton ch= (RadioButton) this.findViewById (R.id.RBut1);

boolean     b=ch.isChecked();
```

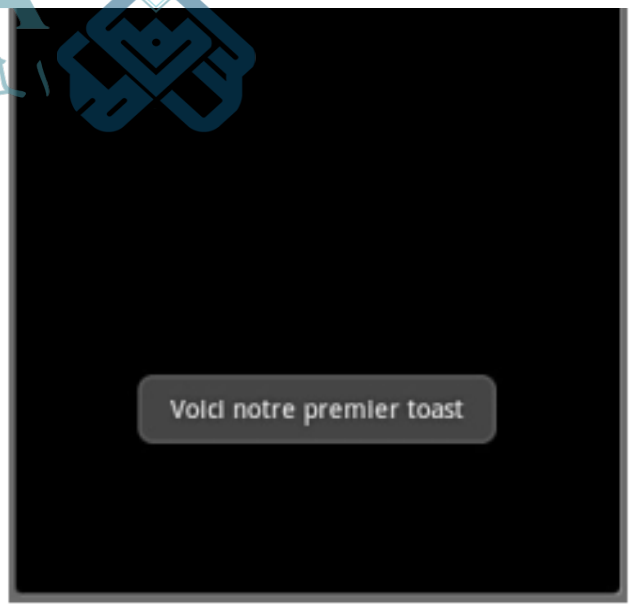
○ RadioGroup


```
<RadioGroup
    android:layout_width=“wrap_content "
    android:layout_height="wrap_content"
    android:orientation= " vertical" >

RadioButton
    android:layout_width=“wrap_content"
    android:layout_height="wrap_content"
    android:text= "OUI"
    android:checked= “true" />

RadioButton
    android:layout_width=“wrap_content"
    android:layout_height="wrap_content"
    android:text= “NON" />

</RadioGroup>
```



Toast

- Andorid Toast peut être utilisé pour afficher des informations à l'utilisateur pour une courte période de temps.
- Un toast contient un message à afficher rapidement et disparaît après un certain temps sans intervention de l'utilisateur.
- Vous pouvez également créer un toast personnalisé, par exemple un toast affichant une image.
- Un "toast" est une instance de la classe "Toast" ***(android.widget.Toast)***




```
Toast toast= Toast.makeText(this,"Voici notre premier toast",
Toast.LENGTH_LONG);


toast.show();
```

Gérer l’ évènement de clic

- **1.Utiliser un click listener(l’interface** OnClickListener)
- Package *android.view.View.OnClickListener*

- **2. Utiliser l’attribut**

android:onClick dans un fichier xml

1. Utiliser un click listener


```
public class
MainActivity extends AppCompatActivity implements
OnClickListener
{
@Override
Protected void onCreate (Bundle savedInstanceState)
{ ……
  ……
 Button b = (Button) this.findViewById(R.id.button);
b.setOnClickListener(this);}

public void onClick(Viewv)
{
//insérer votre code ici
 }
}
```

2.Utiliser l'attribut android:onClickdans un fichier xml


```
public class MainActivity extends AppCompatActivity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        .....
        .....

    }
```

```
<Button
    android:text="Add"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:id="@+id/button"
    android:onClick="onClickAdd"
    />
```

```
public void onClickAdd(View v){
```

Université d'Alger 1 Benyoucef Ben Kheda
Faculté des Sciences
Département MI

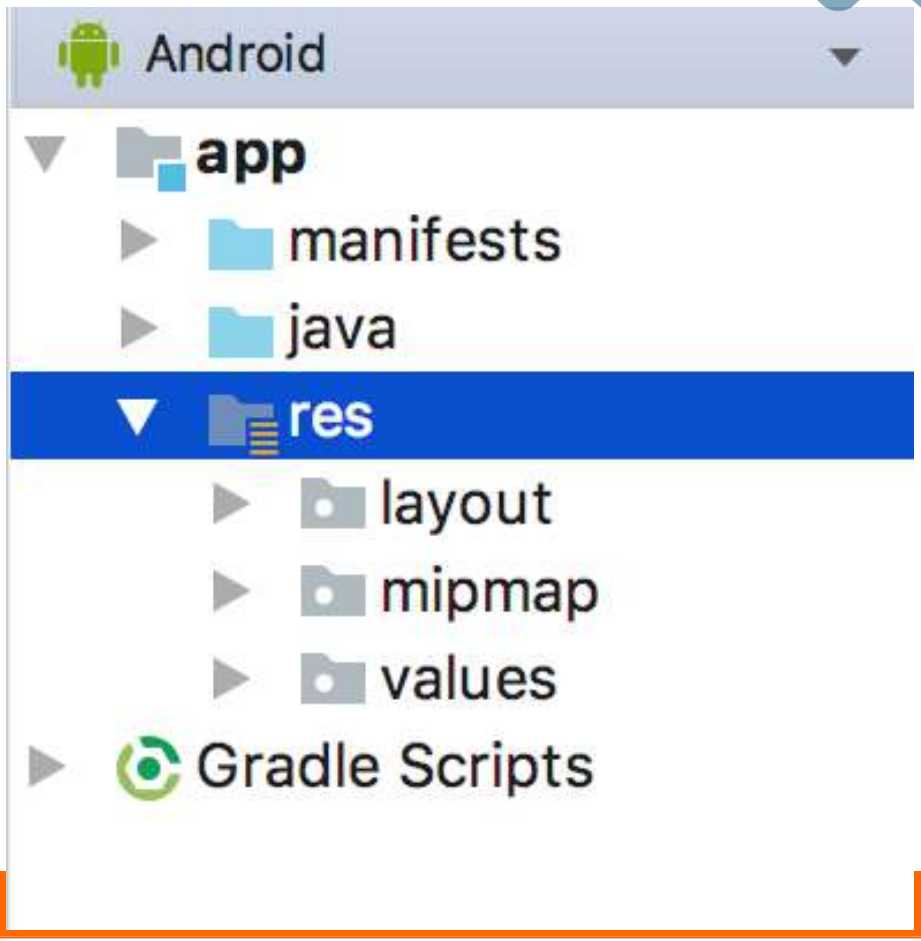
L3 - SI

Développement d Applications Mobiles

DR. AICHA BOUTORH

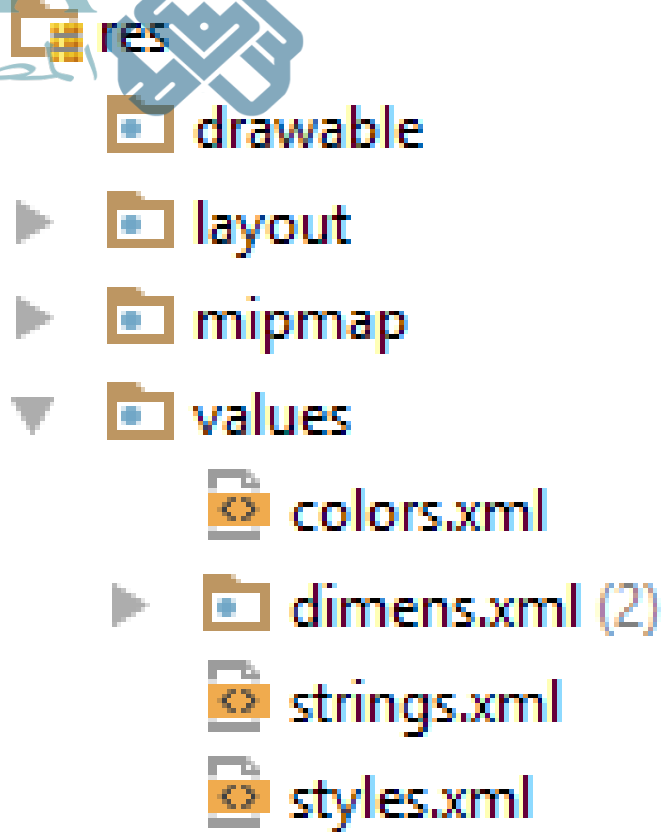
2017/2018

Les Ressources



Les Ressources

- Android est utilisé sur un très grand nombre de supports différents,
- Les ressources sont des fichiers externes organisés d'une manière particulière
- Les fichiers de ressources ne contenant pas d'instruction, et sont utilisés par le code
- Les ressources liés à l'application au moment de sa construction.



- Android offre un support d'un grand nombre de fichiers ressources:
  - les fichiers XML...
  - les fichiers images JPEG et PNG,
  - ....

La classe R

- Les ressources sont accessibles et utilisées depuis le code grâce à la classe statique **R**.
- Cette classe est automatiquement générée en fonction des ressources présentes dans votre projet au moment de la compilation et de la construction de l'application.

Un Exemple:

```
public final class R {
    public static final class string {
        public static final int bienvenue=0x7f040000;
        public static final int texte_bouton_quitter=0x7f040001;
        public static final int texte_titre_ecran=0x7f040002;
    };
    public static final class layout {
        public static final int ecran_de_demarrage=0x7f030001;
        public static final int ecran_principal=0x7f030000;
    };
    public static final class drawable {
        public static final int image_android=0x7f020000;
    };
};
```

L'Utilisation d'une Ressource

- La syntaxe pour utiliser une ressource:

R.type_de_ressource.nom_ressource

- ***Exemples:***

```
String name = resources.getString (R.string.AppName);
setContentView (R.layout.Start_Screen);
Button BT= (Button) this.findViewById (R.id.BT1);
```

Ressource Drawable

- Une ressource drawable est un concept général pour un graphique qui peut être dessiné à l'écran et qui construit les images de l'application

- Pour créer une ressource drawable:
  copier l'image dans le dossier ***drawable***

- Pour référencer une ressource drawable: utiliser la syntaxe **"@drawable/nom de l'image"**

- ***Expl:*** android:background="@drawable/My_Image«

- Pour référencer une ressource drawable dans un programme java: **R.drawable.nomde l'image**

Ressource mipmap

- L' icone de l'application
- Pour creéer une ressorce mipmap, copier l'icon dans le dossier **mipmap**

- Pour référencer une ressource mipmap, utiliser la syntaxe **"@mipmap/nom de l'image«**

- ***Expl***: **android:icon="@mipmap/My_Icon"**

- Pour référencer une ressource mipmap dans un programme java: **R.mipmap.nom de l'icone**;

Ressource de type Color

- **Création de la Ressource Color: res/values/ color.xml**


```
<?xml version="1.0" encoding="utf-8"?>
<resources>
<color name="Green"> #51B000 </color>
<color name="Blue"> #303F9F </color>
<color name="Pink"> #FF4081 </color>
</resources>
```

- **Utilisation de la Ressource Color:**

```
<Button
  android:id="@+id/BT1"
  android:text="Button1"
  android:textColor="@color/Green"
  android:background="@color/Blue"
  android:layout_width="wrap_content"
  android:layout_height="wrap_content"/>
```


```
<Button
  android:id="@+id/BT2"
  android:text="Button2"
  android:textColor="@color/Pink"
  android:background="@color/Green"
  android:layout_width="wrap_content"
  android:layout_height="wrap_content"/>
```

Ressource de type Dimension

- **Création de la Ressource Dimension: res/values/ Dimens.xml**

```
<resources>
    <dimen name="marge_intern">16dp</dimen>
    <dimen name="marge_extern">16dp</dimen>
    <dimen name="large_button">120dp</dimen>
    <dimen name="haut_button">60dp</dimen>
</resources>
```

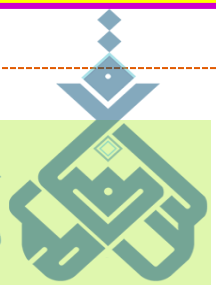
• Utilisation de la Ressource Dimension:

```
<Button
  android:id="@+id/BT1"
  android:text="Button1"
  android:textColor="@color/Green"
  android:background="@color/Blue"
  android:layout_width="@dimen/large_button"
  android:layout_height="@dimen/haut_button“
  android:layout_margin="@dimen/marge_extern“
  android:Padding="@dimen/marge_intern"/>
```



```
<Button
  android:id="@+id/BT2"
  android:text="Button2"
  android:textColor="@color/Pink"
  android:background="@color/Green"
   android:layout_width="@dimen/large_button"
  android:layout_height="@dimen/haut_button“
  android:layout_margin="@dimen/marge_extern“
  android:Padding="@dimen/marge_intern"/>
```

Ressource de type String



- **Création de la Ressource String: res/values/ Strings.xml**

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
< string name="App_Name"> CourseAndroid</string>
< string name="Msg"> Welcome </string>
< string name="Qst"> Give your Question </string>
</resources>
```

- **Référencement des ressources Stringres**

- Affecter une ressource String à une vue (View) en Java :

*EditText txt= (EditText) **this**.findViewById(R.id.**t**);*
*txt.setText(R.string.**Qst**);*

- Pour spécifier un titre de label dans un layout.xml : **@string/nom**

```
<TextView
    android:text="@string/Qst"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:id="@+id/textView" />
```

Ressource de type Style

- Les styles permettent de changer l'aspect général de l'application, d'un élément ou d'une partie (l'apparence de tous les textes de l'application).
- Un style est une collection de propriétés qui spécifie le design d'une vue, d'un élément ou d'une application.
- Un style peut spécifier différentes propriétés : *Padding, Margin, Style de texte, Couleur du texte, Taille du texte, etc…*
- Un style se défini dans un fichier ressource XML (styles.xml)

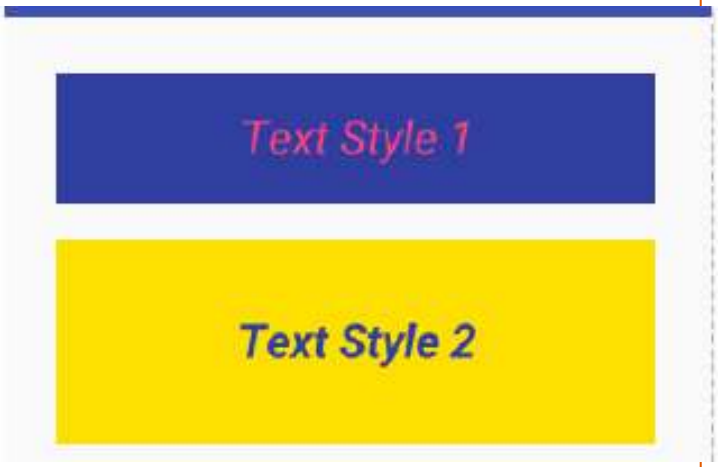
- **Création de la Ressource Style: res/values/ Styles.xml**


```
<style name="TextStyle1">
    <item name="android:textColor">@color/textColor1</item>
    <item name="android:textStyle">italic</item>
    <item name="android:textSize">@dimen/textSize</item>
    <item name="android:background">@color/btnBackground1</item>
    <item name="android:layout_gravity">center_horizontal</item>
    <item name="android:gravity">center_horizontal</item>
    <item name="android:padding">20dp</item>
</style>

<style name="TextStyle2">
    <item name="android:textColor">@color/textColor2</item>
    <item name="android:textStyle">italic|bold</item>
    <item name="android:textSize">@dimen/textSize</item>
    <item name="android:background">@color/btnBackground2</item>
    <item name="android:layout_gravity">center_horizontal</item>
    <item name="android:gravity">center_horizontal</item>
    <item name="android:padding">40dp</item>
</style>
```

[Ref]

- **Utilisation de la Ressource Style:**

```
<TextView
    android:theme="@style/TextStyle1"
    android:text="Text Style 1"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:id="@+id/textView4"
    android:layout_margin="10dp"/>


<TextView
    android:theme="@style/TextStyle2"
    android:text="Text Style 2"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:id="@+id/textView8"
    android:layout_margin="10dp"/>
```



Université d'Alger 1 Benyoucef Ben Kheda
Faculté des Sciences
Département MI

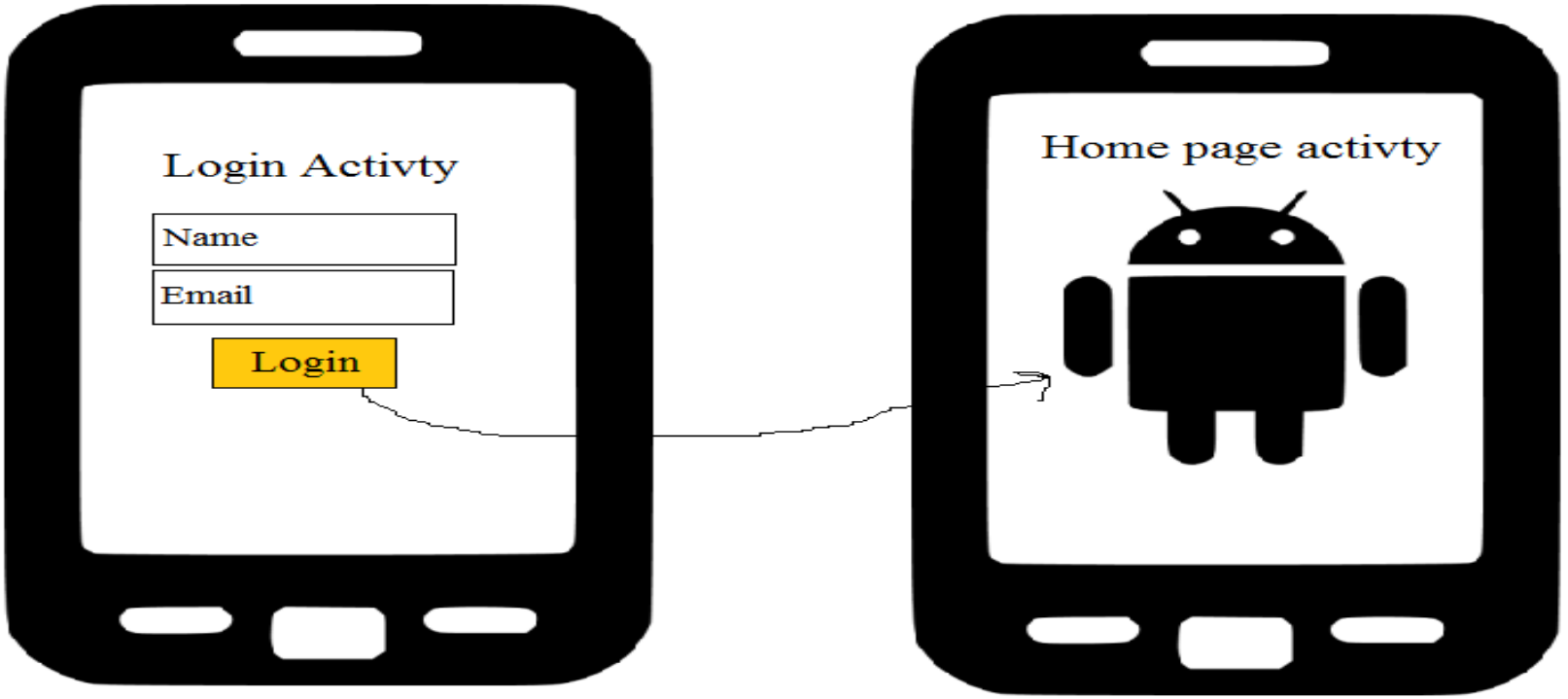
L3 - SI

Développement d Applications Mobiles

DR. AICHA BOUTORH

2017/2018

La Communication Entre Composants Intent



- Afin de pouvoir insérer plusieurs activités au sein d'une même application; le fichier *Manifest* doit être bien maîtrisé.

- Dans le fichier ***Manifest*** vous trouverez un ensemble de nœuds pour décrire votre projet.

- Le nœud ***<application>*** peut-être le nœud le plus important dans le ***Manifest.***

Code : XML

```xml
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
  package="sdz.chapitreTrois"
  android:versionCode="1"
  android:versionName="1.0" >

  <uses-sdk android:minSdkVersion="7" />

  <application
    android:icon="@drawable/ic_launcher"
    android:label="@string/app_name" >

    <activity
      android:name=".ManifestActivity"
      android:label="@string/app_name" >

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

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

    </activity>

  </application>

</manifest>
```


[Ref]

Le nœud : <application>

- Le nœud **« application »** décrit les attributs et les différents composants qui caractérisent votre application.
- Par défaut, votre application n'a qu'un composant, l'**Activité Principale**.
- **Les composants:** sont les éléments qui composeront vos projets.
- Votre application sera au final un ensemble de composants qui interagissent, entre eux et avec le reste du système.

<Activity>

```
<activity
    android:name=".ManifestActivity"
    android:label="@string/app_name" >
```


- Le nœud « ***Activity*** » décri toutes les activités contenues dans l'application.

- ***Rappel:*** une activité correspond à un écran d'une application. Plusieurs écrans, signifie, plusieurs activités.

- **android:name** un attribut indispensable, pour indiquer la classe qui implémente l'activité. Il représente également un **identifiant** pour Android qui permet de repérer ce composant parmi d'autre composants.

- **android:label** permet de préciser un nom pour chaque activité, c'est le mot qui s'affichera en haut de l'écran.

<intent-filter>

```
<activity
  android:name=".ManifestActivity"
  android:label="@string/app_name" >
  <intent-filter>
    <action android:name="android.intent.action.MAIN" />

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

</activity>
```


- Indique comment cette activité se lancera.
- L'activité qui sera lancée depuis le menu principal d'Android contiendra toujours ces deux lignes dans son Manifest.

Communication entre Applications la Classe Intent

- Comment lancer une activité depuis une autre activité ?
- Il existe un mécanisme puissant qui permet de faire exécuter certaines actions et de faire circuler des messages entre applications ou à l'intérieur d'une même application.
- Ce mécanisme est les **Intents.**
- **Intents** forment un mécanisme complet permettant aux activités et aux applications d'interagir entre elles.

Communication entre Applications la Classe Intent

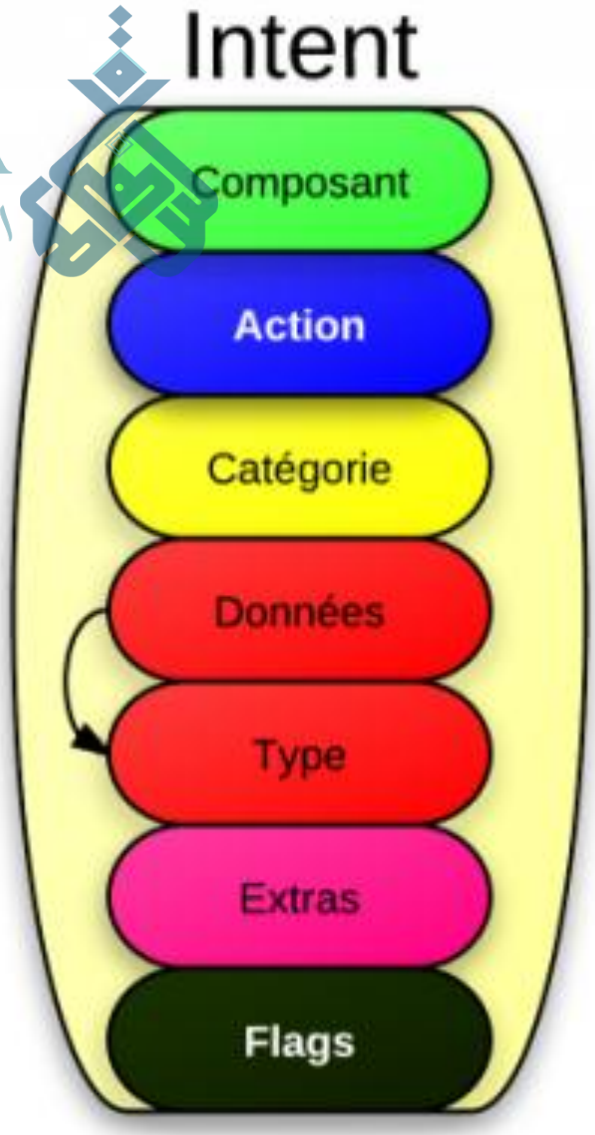
- **Exemple:**

- Si l'utilisateur clique sur un numéro de téléphone dans une application, tout le système sera informé qu'il y a un numéro qu'il faut appeler grâce à un **intent**.

Le système par la suite, trouvera les applications qui peuvent prendre en charge cette demande.

Intent

- Un intent est un objet qui contient plusieurs champs.
- La façon dont sont renseignés ces champs détermine la nature ainsi que les objectifs de l'intent.



- **Composant:** permet de définir le destinataire de l'intent. Si le champ est renseigné alors l'intent est **explicite,** sinon il est **implicite** i.e le destinataire de l'intent n'est pas précisé, c'est pourquoi d'autres champs seront renseignés pour laisser Android déterminer qui est capable de réceptionner cet intent.

❑ Il faut au moins fournir deux informations:

➢ **Une action** : ce qu'on désire que le destinataire fasse.

➢ **Un ensemble de données** : sur quelles données le destinataire doit effectuer l'action.

- Il existe aussi d'autres informations utiles, pas forcément obligatoires.

Autres Informations des Intents

- **La catégorie** : fournit des informations supplémentaires sur l'action à exécuter et le type de composant qui devra gérer l'intent.
- **Le type** : indique le type des données incluses. En précisant cet attribut vous pouvez 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.

Intent Explicite

- Pour créer un intent explicite, il suffit de donner un Context qui appartient au package où se trouve la classe de destination

```
Intent I = new Intent (Context context, Class<?> cls);
```

Code : Java

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

- Pour lancer l'intent il existe **deux façons:**
  - Sans Retour
  - Avec Retour (le composant de destination nous renvoie une réponse).

1) Démarrer une activité Sans Retour

- Pour démarrer une activité *(sans retour)*, la méthode utilisée est **void startActivity (Intent intent)** , avec comme paramètre une instance de la classe Intent spécifiant l'activité à exécuter :

```
Intent intent= new I (this, Activity2.class);
startActivity (I);
```


- Avant de pouvoir démarrer l'activité, il faut au préalable la déclarer dans le fichier **AndroidManifest.xml** de l'application.
- Sans cette déclaration, une erreur du type ***ActivityNotFoundException*** sera générée lors de l'appel de la méthode *startActivity*.

```
<application ...>
<activity android:name=".Activity2"></activity>
</application>
```

Exemple: dans une première activité, vous allez mettre un bouton de sorte qu' appuyer sur ce bouton lance une seconde activité.

```
public class MainActivity extends Activity {
  public final static String AGE =
"sdz.chapitreTrois.intent.example.AGE";

  private Button mPasserelle = null;

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

    mPasserelle = (Button) findViewById(R.id.passerelle);

    mPasserelle.setOnClickListener(new View.OnClickListener() {
      @Override
      public void onClick(View v) {
        // Le premier paramètre est le nom de l'activité actuelle
        // Le second est le nom de l'activité de destination
        Intent secondeActivite = new Intent(MainActivity.this,
IntentExample.class);



        // Puis on lance l'intent !
        startActivity(secondeActivite);
      }
    });
  }
```


[Ref]

La seconde activité ne fera rien de particulier, elle affiche un layout différent

Code : Java

```java
package sdz.chapitreTrois.intent.example;

import android.app.Activity;
import android.os.Bundle;

public class IntentExample extends Activity {
  @Override
  public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.layout_example);

    // On récupère l'intent qui a lancé cette activité
    Intent i = getIntent();
```

[Ref]

Le Manifest.xml

- N'oubliez pas de préciser dans le Manifest les deux activités qui composent votre application

```
<application
  android:icon="@drawable/ic_launcher"
  android:label="@string/app_name"
  android:theme="@style/AppTheme" >
  <activity
    android:name=".MainActivity"
    android:label="@string/title_activity_main" >
    <intent-filter>
      <action android:name="android.intent.action.MAIN" />
      <category android:name="android.intent.category.LAUNCHER" />
    </intent-filter>
  </activity>

  <activity
    android:name=".IntentExample"
    android:label="@string/title_example" >
  </activity>
</application>
```

[Ref]

bouton de la première activité, on passe à la seconde

2) Démarrer une activité Avec Retour

- Pour démarrer une activité qui renvoie un petit *feedback* au retour, on utilise la méthode
  void startActivityForResult(Intent intent, int requestCode)

- La méthode ***startActivityForResult*** permet de communiquer une valeur de retour à l'activité parent

- La *'sous-activité'* est identifiée avec un **requestCode.**
  i.e identifier de manière unique un Intent

- la méthode **setResult** de la classe ***Activity*** est utilisée pour renvoyer une valeur de retour, en indiquant comme paramètre le code de retour.

- Android prévoit plusieurs valeurs par défaut telles que:
  RESULT_OK; *RESULT_CANCELED ;* ...

Les valeurs constantes (codes de retours)

Constante	Valeur
DEFAULT_KEYS_DIALER	1
DEFAULT_KEYS_DISABLE	0
DEFAULT_KEYS_SEARCH_GLOBAL	4
DEFAULT_KEYS_SEARCH_LOCAL	3
DEFAULT_KEYS_SHORTCUT	2
RESULT_CANCELED	0
RESULT_FIRST_USER	1
RESULT_OK	-1

2) Démarrer une activité Avec Retour

- La méthode **void onActivityResult (int requestCode, int resultCode, Intent data)** est la première méthode de *callback* appelée dans l'activité initiale quand l'activité appelée s'arrêtera.

- La méthode ***onActivityResult*** utilise trois paramètres pour identifier l'activité et ses valeurs de retour :

  - **requestCode**: valeur identifiant quelle activité a appelé la méthode (le même code que celui passé dans *startActivityForResult*)

  - **resultCode**: valeur de retour envoyée par la sous-activité pour indiquer comment elle s'est terminée. C'est une constante définie dans la classe *Activity* (RESULT_OK *« si l'activité s'est terminée normalement »*, RESULT_CANCELED *« s'il y a eu un problème ou qu'aucun code de retour n'a été précisé »*, etc.)

  - **Intent data :** cet objet permet d'échanger des données

2) Démarrer une activité Avec Retour

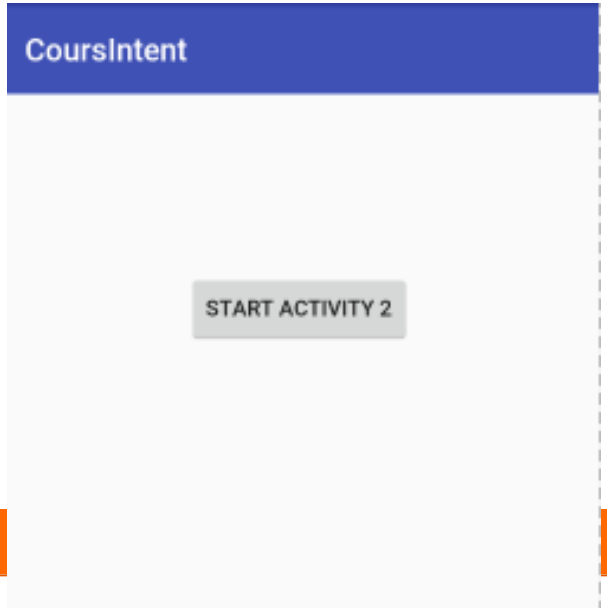
- Dans la seconde activité, le résultat est défini avec la méthode

void setResult (int resultCode, Intent data)

- **Exemple:**
  - La première activité contient un bouton qui lance une deuxième activité.
  - La seconde activité proposera à l'utilisateur de cliquer sur deux boutons.
  - Cliquer sur un de ces boutons retournera à l'activité précédente en lui indiquant lequel des deux boutons a été choisit.

Main Activity

```
private static final int CODE_ACTIVITY = 1;
@Override
protected void onCreate(Bundle savedInstanceState) {
.....
}

@Override
public void onClick(View v) {
    if(v.getId()==R.id.btn1){
        Intent intent = new Intent(this,Activity2.class);
        startActivityForResult(intent, CODE_ACTIVITY);
    }
}
```

Main Activity


```
public class MainActivity .....{
    private static final int CODE_ACTIVITY = 1;
    @Override
    protected void onCreate(.....) {.....}
    @Override
    public void onClick(View v) {.....}
    @Override
    protected void onActivityResult(int requestCode, int resultCode, Intent data)
// Le code de requête est utilisé pour identifier l'activité enfant
        if(requestCode==CODE_ACTIVITY){
        if (resultCode == RESULT_OK)
            Toast.makeText(this, "Action validée", Toast.LENGTH_LONG).show();

        if (resultCode == RESULT_CANCELED)
            Toast.makeText(this, "Action non validée", Toast.LENGTH_LONG).show();
        }
    }
```

Second Activity


```
public class Activity2 extends AppCompatActivity implements OnClickListener{
    @Override
    protected void onCreate(.....) {

      .....
    }
    @Override
    public void onClick(View v) {
        if(v.getId()==R.id.btnYes)
            setResult(RESULT_OK);

        if(v.getId()==R.id.btnNo)
            setResult(RESULT_CANCELED);
    }
}
```

Intent Implicite

- **Déléguer au système le choix de l'application**

❖ Envoyer une requête à un destinataire, sans savoir qui est.

❖ Ainsi, les applications destinataires sont soit fournies par Android, soit par d'autres applications téléchargées sur le Play Store par exemple.

- Nous pouvons envoyer une intention (intent) et demander au système de choisir le composant le plus approprié pour exécuter l'action transmise.

- C'est le système qui décide alors de l'application à utiliser pour exécuter l'action.

- Pour décider du composant le plus approprié, le système se base sur les informations qu'on spécifie dans l'objet Intent: ***action, données, ….***

- Ainsi on exprime l'intention au système et le système se chargera de résoudre l'intention en proposant le composant de l'application le plus approprié.

Une action est une constante qui se trouve dans la classe Intent

Quelques actions natives:

ACTION_ANSWER	Prendre en charge un appel entrant.
ACTION_CALL	Appeler un numéro de téléphone. Cette action lance une activité affichant l'interface pour composer un numéro puis appelle le numéro contenu dans l'URI spécifiée en paramètre.
ACTION_DELETE	Démarrer une activité permettant de supprimer une donnée identifiée par l'URI spécifiée en paramètre.
ACTION_DIAL	Afficher l'interface de composition des numéros. Celle-ci peut être pré-remplie par les données contenues dans l'URI spécifiée en paramètre.
ACTION_EDIT	Éditer une donnée.
ACTION_SEARCH	Démarrer une activité de recherche. L'expression de recherche de la pourra être spécifier dans la valeur du paramètre SearchManager.QUERY envoyé en extra de l'action.
ACTION_SEND	Envoyer des données texte ou binaire par courriel ou SMS. Les paramètres dépendront du type d'envoi.
ACTION_SENDTO	Lancer une activité capable d'envoyer un message au contact défini par l'URI spécifiée en paramètre.
ACTION_VIEW	Démarrer une action permettant de visualiser l'élément identifié par l'URI spécifiée en paramètre. C'est l'action la plus commune. Par défaut les adresses commençant par http: lanceront un navigateur web, celles commençant par tel: lanceront l'interface de composition de numéro et celles débutant par geo: lanceront Google Map.
ACTION_WEB_SEARCH	Effectuer une recherche sur Internet avec l'URI spécifiée en paramètre comme requête.

Exemple d'intent qui lance l'interface de composition de numéro

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    Button b=(Button)this.findViewById(R.id.btn1);
    b.setOnClickListener(this);
}

@Override
public void onClick(View v) {
    if(v.getId()==R.id.btn1){
        Uri uri = Uri.parse("tel:0774482316");
        Intent intent = new Intent(Intent.ACTION_DIAL, uri);
        startActivity(intent);
    }
}
```


- ***URI*: *Uniform Resource Identifier***
- est une courte chaîne de caractères identifiant une ressource [Ref]

Exemple d'intent qui lance le navigateur Web


```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    Button b=(Button)this.findViewById(R.id.btn1);
    b.setOnClickListener(this);
}

@Override
public void onClick(View v) {
    if(v.getId()==R.id.btn1){
        Uri uri = Uri.parse("http://www.google.fr/");
       Intent intent = new Intent(Intent.ACTION_VIEW, uri);
       startActivity(intent);
    }
}
```

« Extra »

Transport d'informations dans un Intent

- Les Intents servent à transporter des informations d'une activité à l'autre
- Le champ « **extra** » dans un intent contient les données à véhiculer entre les applications.
- Un extra est ***une clé*** à laquelle on associe une valeur.
- la méthode **Intent putExtra (String key, X value)** est utilisée pour insérer un extra, avec ***key*** *la clé de l'extra* et ***value*** *la valeur associée*.
- Pour le type de la valeur on peut y mettre n'importe quel type de base (*int, String*, …)

« Extra »

- Pour récupérer tous les extras d'un intent, on utilise la méthode **Bundle getExtras()**, auquel cas vos couples **clé-valeurs** sont contenus dans le Bundle.

- Pour récupérer un extra précis à l'aide de sa clé et de son type, on utilise la méthode

X get{X}Extra(String key, X defaultValue),

X étant le type de l'extra et ***defaultValue*** la valeur qui sera retournée si la clé passée ne correspond à aucun extra de l'intent.

Placer des données dans un Intent:


```
Intent intent= new  Intent (this,  Activity2.class);
intent.putExtra ("user",  "User_Name" );
startActivity (intent);
```

putExtra (nom, valeur) rajoute un couple *(nom, valeur)* dans l'intent.

La valeur peut être *nombres*, *chaîne*, *booléen*, *réel*, ...

Extraction d'informations d'un Intent


```
Bundle b= getIntent().getExtras();
String user = b.getString("user");
```

- Ou bien:

```
Intent intent= getIntent();
String user = intent.getExtras().getString("user");
```

- Ou bien:

```
Intent intent = getIntent();
String user = intent.getStringExtra ("user");
```

- getIntent() retourne l'Intent qui a démarré cette activité.

Contexte d'une application

- Le contexte d'une application est un objet global vivant pendant tout le fonctionnement d'une application

- Pour le récupérer :

```
Context context= this.getApplicationContext();
```

- Il contient des informations sur l'état actuel de l'application et constitue un lien avec le système Android ainsi que les autres activités de l'application.

Université d'Alger 1 Benyoucef Ben Kheda
Faculté des Sciences
Département MI

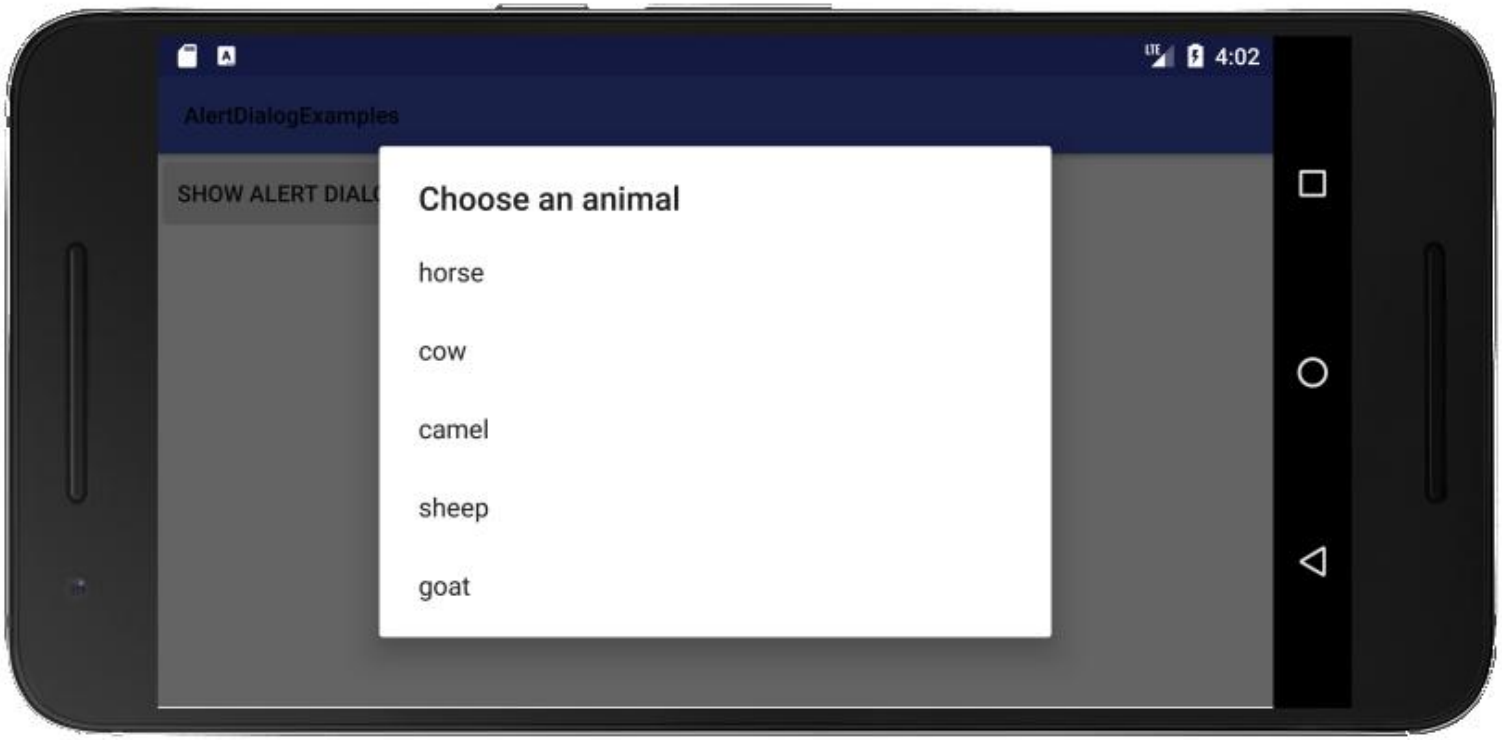
L3 - SI

Développement d Applications Mobiles

DR. AICHA BOUTORH

2017/2018

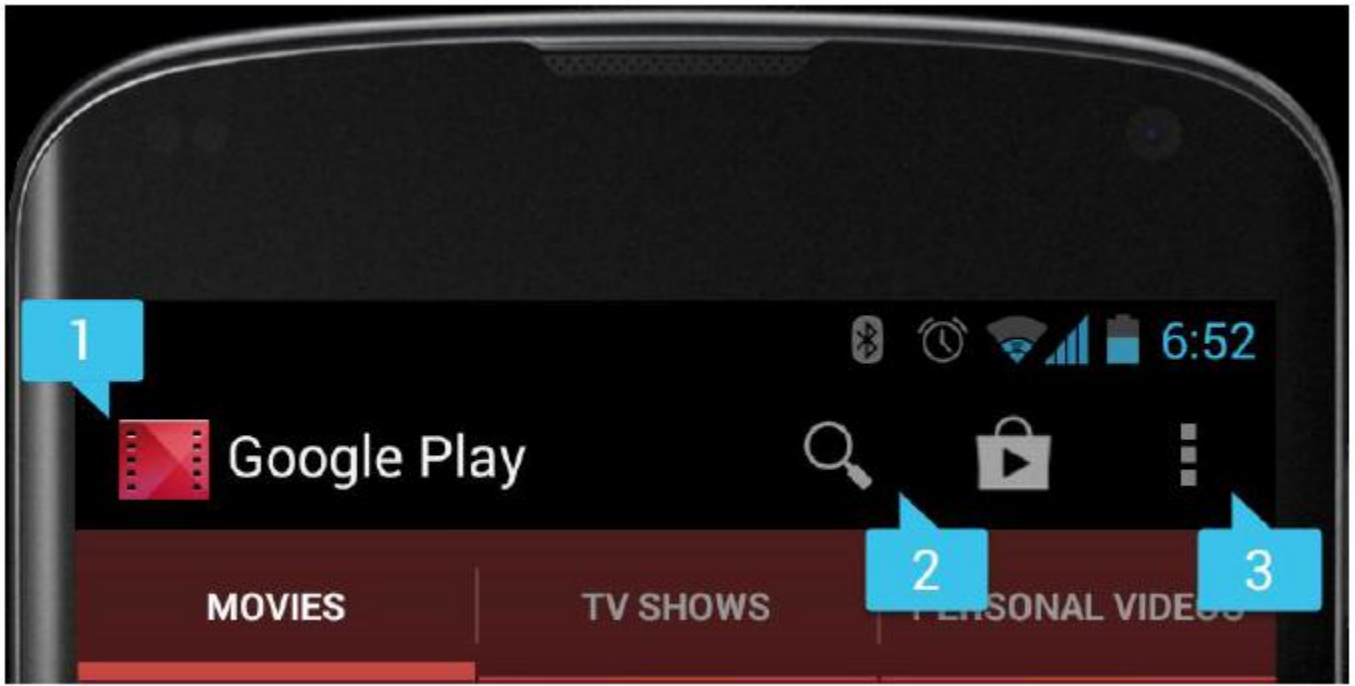
Menus, Dialogues et ListView



Barre d'Actions

- La barre d'actions (***ActionBar*) *se situe en haut d'une activité***. Elle peut afficher le titre de l'activité, l'icône d'application (1), des actions qui peuvent être lancées (items de menus) (2), des vues additionnelles et un bouton pour avoir d'autres choses encore (3).


```
getSupportActionBar().setTitle("titre");
getSupportActionBar().setDisplayShowHomeEnabled(true);
getSupportActionBar().setIcon(R.mipmap.icon);
```



Les Menus

- Un menu est une liste d'items qui apparaît soit sur la barre d'action, soit quand on appuie sur le bouton « *menu* ».

- Certains de ces items sont présents en permanence dans la barre d'action. La sélection d'un item déclenche une callback.

- Il existe deux sortes de menu dans Android :
  - **Le menu d'options**: peut être ouvert avec le bouton Menu sur le téléphone.
  - **Le menu contextuel**: peut être ouvert en effectuant un long clic sur un élément de l'interface graphique. Le même principe des menus qui s'ouvrent à l'aide d'un clic droit sous Windows et Linux

- **Création de menu:** **un fichier** ***res/menu/nom_du_menu.xml***

Spécification d'un menu:

* Créer: res/menu/nom_du_menu.xml :
* Chaque **<item>** : identifiant, icône, titre, enabled….
* L'attribut **showAsAction** = "***always***" ou "***ifRoom***" ou "***never***" selon la visibilité qu'on souhaite dans la barre d'action.


```
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto">
    <item android:id="@+id/itemsave"
        android:title="new"
        app:showAsAction="ifRoom"
        android:icon="@drawable/iconsave"/>

    <item android:id="@+id/itemdelete"
        android:title="delete"
        app:showAsAction="always"
        android:icon="@ drawable/icondelete"/>

   <item android:id="@+id/help"
        android:title="help"
        app:showAsAction="never"
        android:icon="@ drawable/iconhelp" />
</menu>
```

[Ref]

- Afin de gagner un peu de place, il est possible d'avoir un **<item>** qui ouvre un sous-menu, et ce sous-menu sera à traiter comme tout autre menu.


```
<item>
  <menu>
    <item />
    <!-- d'autres items-->
    <item />
  </menu>
</item>
```

Menu d'Options

1) création d'un menu d'options:

* Surcharger la méthode ***onCreateOptionsMenu (Menu menu)*** pour rajouter les items du menu défini dans le XML.
* Un ***MenuInflater*** est un lecteur/traducteur de fichier XML en vues ; sa méthode '*inflate*' crée les Vues (instancie le menu).

"To inflate, c'est désérialiser en français, et dans notre cas c'est transformer un objet qui n'est décrit qu'en XML en véritable objet qu'on peut manipuler."

```
//Creating an Options Menu
@Override
public boolean onCreateOptionsMenu(Menu menu) {
    MenuInflater inflater = getMenuInflater();
    inflater.inflate(R.menu.nom_de_menu, menu);
    return true;
}
```

2) Réactions aux clic

* l'identifiant d'un **<item>** permet de déterminer comment il réagit aux clics au sein de la méthode boolean onOptionsItemSelected (MenuItem item).
* Voici la seconde callback selon l'item choisi :


```
//Handling click events
@Override
public boolean onOptionsItemSelected(MenuItem item) {
    // Handle item selection
    switch (item.getItemId()) {
        case R.id.itemsave: Toast.makeText(this,"Save Item",
       Toast.LENGTH_SHORT).show(); return true;
        case R.id.itemdelete: Toast.makeText(this,"DeleteItem",
       Toast.LENGTH_SHORT).show(); return true;
        case R.id.itemhelp: Toast.makeText(this,"help Item",
       Toast.LENGTH_SHORT).show();  return true;
       default:
            return super.onOptionsItemSelected(item);
    }
}
```

Menu Contextuel

- Un menu contextuel apparait si l'utilisateur effectue un ***appui long*** sur un élement

- Déclarer quels ***widgets*** possèdent un menu contextuel.
- Cela se fait dans la méthode de la classe **Activity** **void registerForContextMenu (View vue).**

- Dès que l'utilisateur fera un clic long sur cette vue, un menu contextuel s'ouvrira

- Appeler cette méthode après *setContentView*, lorsque la vue est construite

- La méthode reçoit l'identifiant du widget sur lequel ajouter le menu contextuel

Création de menu contextuel:

- Quand l'utilisateur fait un clic long, cela déclenche la méthode

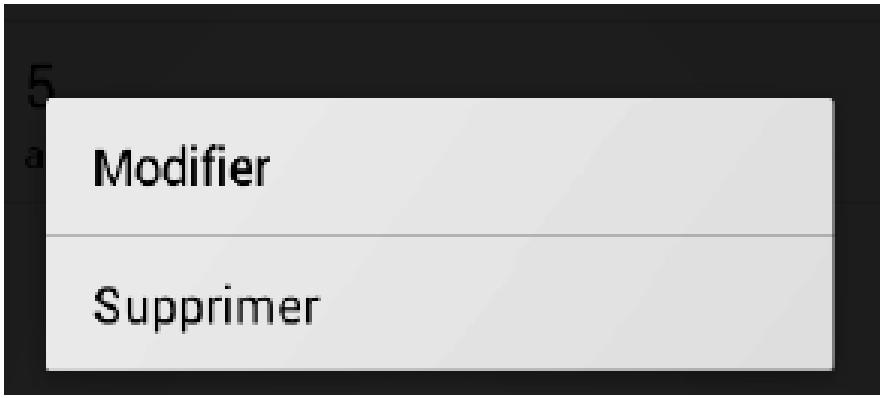
```
void onCreateContextMenu (ContextMenu menu, View vue,
ContextMenu.ContextMenuInfo menuInfo)
```


- Son rôle est d'instancier le menu res/menu/nom_de_menu
  - **menu** est le menu à construire,
  - **vue** la vue sur laquelle le menu a été appelé
  - **menuInfo** indique sur quel élément d'un adaptateur a été appelé le menu

```
//Creating Contextual Menus
@Override
public void onCreateContextMenu(ContextMenu menu, View v,
                                ContextMenu.ContextMenuInfo menuInfo) {
    super.onCreateContextMenu(menu, v, menuInfo);
    MenuInflater inflater = getMenuInflater();
    inflater.inflate(R.menu.nom_de_menu, menu);
}
```

Associer un menu contextuel à une vue:


```
@Override
protected void onCreate(Bundle savedInstanceState){
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        ListView lv = (ListView)findViewById(android.R.id.list);
        registerForContextMenu(lv);
```



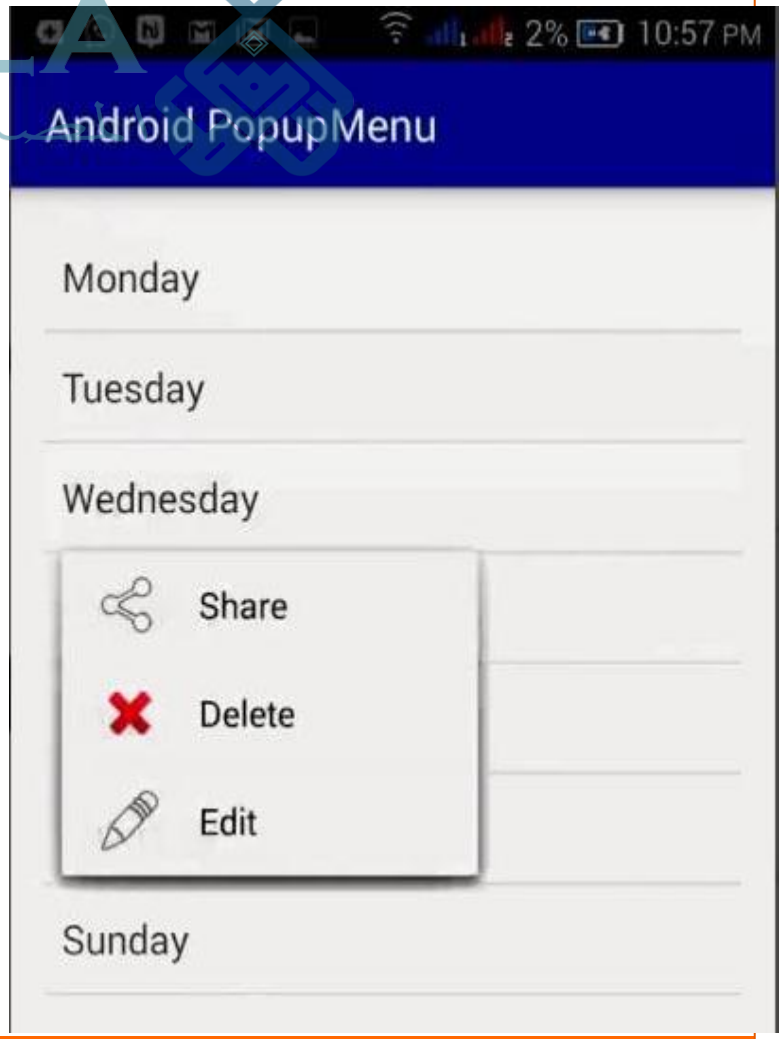
Réactions aux sélections d'items de menu contextuel


```
@Override
public boolean onContextItemSelected(MenuItem item) {
 switch (item.getItemId()) {
        case R.id.itemsave: Toast.makeText(this,"Save Item",
            Toast.LENGTH_SHORT).show(); return true;
        case R.id.itemdelete: Toast.makeText(this,"DeleteItem",
      Toast.LENGTH_SHORT).show(); return true;
        case R.id.itemhelp: Toast.makeText(this,"help Item",
      Toast.LENGTH_SHORT).show();  return true;
      default: return super.onOptionsItemSelected(item);
    }
}
```

Menu Popup

- Un menu popup s'affiche lorsqu'on click sur une vue

L'activité doit implémenter la classe:
PopupMenu.OnMenuItemClickListener
(pour gérer le click)



Exemple:
Afficher un popup
lorsqu'on click sur un bouton

```
<Button
    android:text="Button"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:id="@+id/button"
    android:onClick="showMenu"/>
```



```
//Création d'un menu popup
public void showMenu(View v) {
    PopupMenu popup = new PopupMenu(this, v);
    popup.setOnMenuItemClickListener(this);
    popup.inflate(R.menu.nom_du_menu);
    popup.show();
}
// Gérer les clics
@Override
public boolean onMenuItemClick(MenuItem item) {
    switch (item.getItemId()) {
        case R.id.item1: ……; return true;
        case R.id.item2: ……; return true;
        ……
        default:
            return super.onOptionsItemSelected(item);
    }
}
```

[Ref]

Annonces et Dialogues

- **Annonces : *toasts***
- Un « *toast* » n'affiche aucun bouton.
- C'est un message apparaissant à l'écran pendant un instant.
- La durée d'affichage peut être allongée avec *LENGTH_LONG*.

```
Toast.makeText(getApplicationContext(),
"Ajouté", Toast.LENGTH_SHORT).show();
```

Annonces personnalisées

- Pour personnaliser une annonce, définir un layout dans ***res/layout/toast_layout.xml***.
- La racine de ce layout doit avoir un identifiant, ex : *toast_layout_id*

qui est mentionné dans la création :

```
// expanser le layoutdu toast
LayoutInflater inflater= getLayoutInflater();
Viewlayout= inflater.inflate (R.layout.toast_layout,
(ViewGroup) findViewById(R.id.toast_layout_id));

// créer le toast et l'afficher
Toast toast= new Toast (getApplicationContext());
toast.setDuration(Toast.LENGTH_LONG);
toast.setView (layout);
toast.show();
```

Les Boîtes de Dialogue

- Une boîte de dialogue est une petite fenêtre qui passe au premier plan pour demander à l'utilisateur une confirmation de ce qu'il veut faire.

- Les boîtes de dialogue sont utilisées pour:
  - annoncer des erreurs,
  - indiquer un état d'avancement d'une tâche
  - donner une information ou
  - ...

- Les boîtes de dialogue héritent de la classe **Dialog** et on les trouve dans le package **android.app.Dialog**.

Dialogue d'Alerte

- Un dialogue est une petite fenêtre qui apparait à l'écran pour afficher ou demander quelque chose d'urgent à l'utilisateur.

- Un dialogue d'alerte **AlertDialog** affiche un texte et peut contenir jusqu'à trois boutons pour demander à l'utilisateur ce qu'il souhaite faire.

- Exemples: ok, annuler, oui, non, aide. . .

- Un dialogue d'alerte est construit à l'aide de la classe **AlertDialog.Builder**.

Méthodes pour AlertDialog.Builder

- **setCancelable (boolean x)** : Si le paramètre 'x' vaut **true**, alors on pourra sortir de la boîte grâce au bouton retour de notre appareil.

- **setIcon (int r)** ou **setIcon (Drawable icon)**: Permet d'ajouter une icône à la boîte de dialogue.

- **setMessage (int r)** ou **setMessage (String mssg)** : 'mssg' doit être une ressource de type String ou une String.

- **setTitle (int r)** ou **setTitle (String title)** : le paramètre 'title' doit être une ressource de type String ou une String.

- **setView (View view)** ou **setView (int r)**: le paramètre 'view' doit être une vue. Il s'agit de l'équivalent de setContentView pour un objet de type Context.

Méthodes pour Ajouter des Boutons

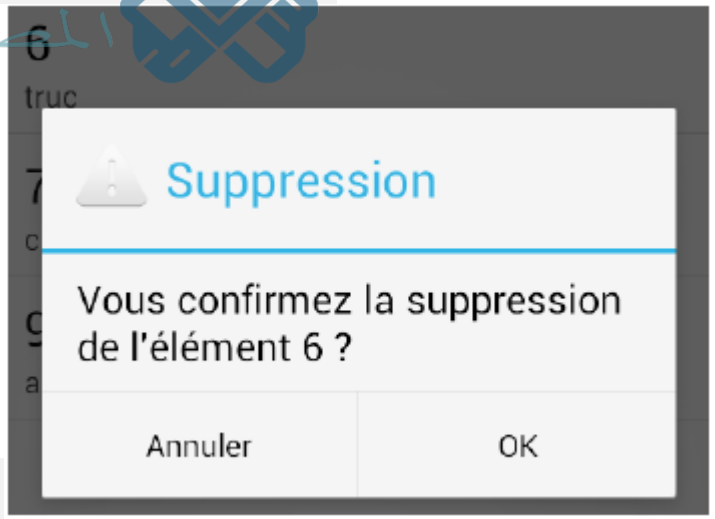
- **setPositiveButton (text, DialogInterface.OnClickListener listener)**
  - **text** qui doit être une ressource de type String ou une String,
  - **listener** qui définira que faire en cas de clic.
  - Ce bouton se trouvera tout à gauche.

- **setNegativeButton (text, DialogInterface.OnClickListener listener)**.
  - Ce bouton se trouvera tout à droite.

- **setNeutralButton (text, DialogInterface.OnClickListener listener)**
  - Ce bouton se trouvera entre les deux autres boutons.

Exemple

```
Builder choix= new AlertDialog.Builder(this);
choix.setTitle("Suppression");
choix.setIcon(android.R.drawable.ic_dialog_alert);
choix.setMessage("Vous confirmez la suppression ?");
```


```
// rajouter un bouton "oui" qui supprime
choix.setPositiveButton(android.R.string.yes,
new DialogInterface.OnClickListener() {
Public  void onClick (DialogInterfacedialog, int idBtn)
{ // methodede supprission} });
// rajouter un bouton "non" qui ne fait rien
choix.setNegativeButton (android.R.string.no, null);
// affichage du dialogue
choix.show();
```

Dialogues Personnalisés

- Un dialogue personnalisé sert à demander à l'utilisateur une information plus complexe.
- Il faut définir le layout du dialogue incluant au moins un bouton pour valider.
- Ensuite attacher le layout de dialogue à un objet de type Dialog

Exemple:

```
Dialog dialog = new Dialog(this); // créer le dialogue
dialog.setContentView(R.layout.edit_dialog);
dialog.setTitle("Création d'un type");
// bouton valider
Button btnValider = (Button) dialog.findViewById(R.id.dialog_btn_valider);
btnValider.setOnClickListener(new OnClickListener() {
@Override public void onClick(View v) {
            ..................
} });
dialog.show(); // afficher le dialogue
```



```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="..." ...>
<TextView android:id="@+id/dialog_titre" .../>
<EditText android:id="@+id/dialog_libelle" .../>
<Button android:id="@+id/dialog_btn_valider" ... />
</LinearLayout>
```

Gestion d'une liste d'items "ListView"

- Dans une application qui affiche et édite une liste d'items:
  - La liste est stockée dans un tableau dynamique appelé ***ArrayList***.
  - L'écran est occupé par un ***ListView***. Une vue qui permet d'afficher des listes quelconques.

- **Le container Java ArrayList<type>**

C'est un type de données générique, c'est à dire paramétré par le type des éléments mis entre <. . .> ;

ce type doit être un objet.

Création d'un tableau ArrayList


```
import java.util.ArrayList;
ArrayList<TYPE> liste = new ArrayList<TYPE>();
```

- Quelques méthodes utiles :

- **liste.size()** : retourne le nombre d'éléments présents.
- **liste.clear()** : supprime tous les éléments.
- **liste.add(x)** : ajoute l'élément '**x**' à la liste.
- **liste.add(i, x)** : insérer l'élément '**x**' à l'indice '**i**' dans la liste.
- **liste.remove(x ou i)** : retire cet élément.
- **liste.get(i)** : retourne l'élément présent à l'indice '**i**'.
- **liste.contains(x)** : true si elle contient l'élément '**x**'.
- **liste.indexOf(x)** : indice de l'élément '**x**', s'il existe.

Pour Afficher une Liste

1- Exemple de layout de l'activité

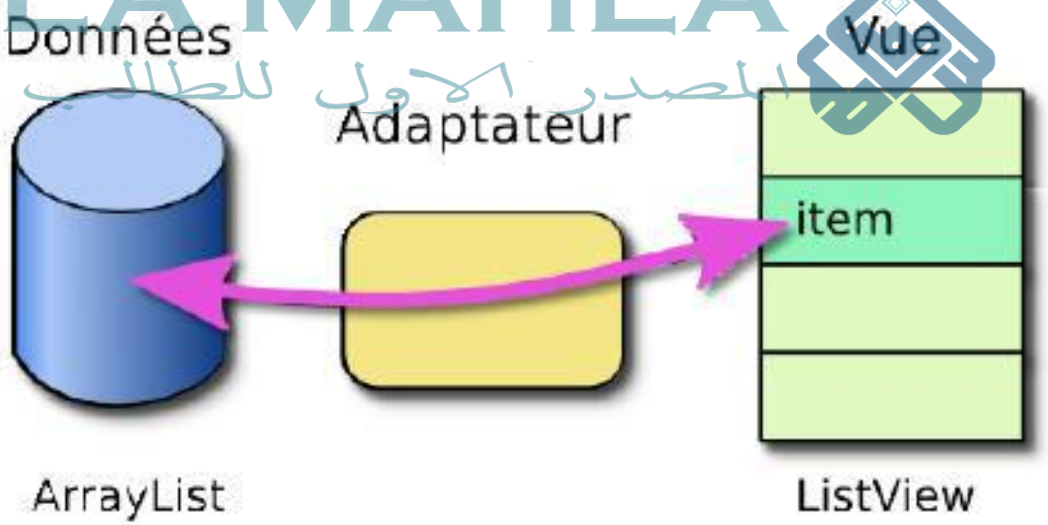
```
<LinearLayout xmlns:android="..."  ... ...>
<ListView android:id="@android:id/list"
android:layout_width="match_parent"
android:layout_height="match_parent"/>
</LinearLayout>
```

2- Exemple de layout pour un item

```
<LinearLayout xmlns:android="..."  ... ...>
<TextView android:id="@android:id/text"/>
</LinearLayout>
```

android.R.layout.simple_list_item_1: C'est un layout qui affiche un seul *TextView*. Son identifiant est android.R.id.text,

3- adaptateur pour accéder aux données



- Un adaptateur est un objet qui affiche un item dans une ListView.
- Une ListView affiche les items à l'aide d'un adaptateur (adapter).

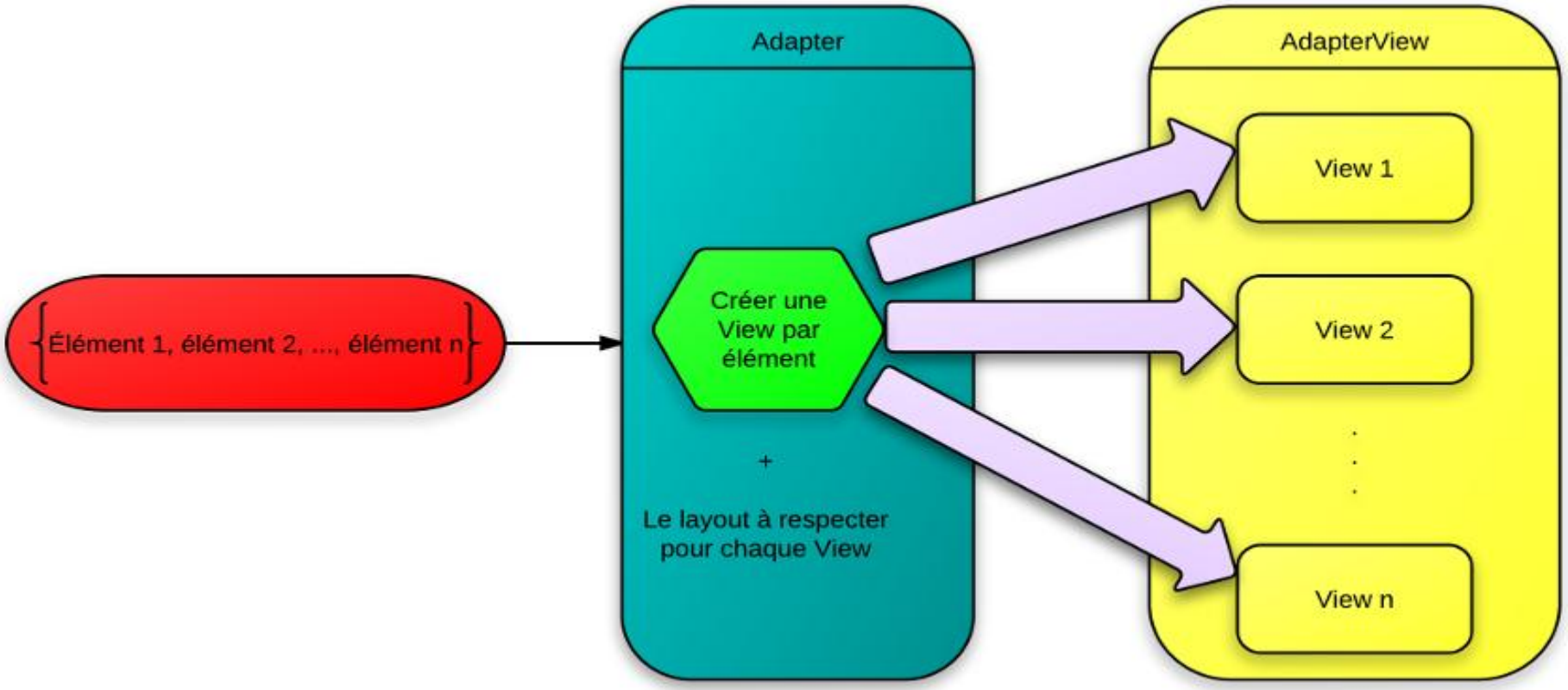
Listes et Adaptateurs

- La gestion des listes se divise en deux parties:

➢ **Adapter** (**adaptateurs**): objets qui gèrent les données, mais pas leur affichage ou leur comportement en cas d'interaction avec l'utilisateur. Un adaptateur peut être considéré comme un intermédiaire entre les données et la vue qui représente ces données. Il accède aux données quelque soit le type de stockage des éléments : tableau, BDD

➢ **AdapterView**: objets qui gèrent l'affichage et l'interaction avec l'utilisateur, mais sur lesquels on ne peut pas effectuer d'opération de modification des données.

- **Pour afficher une liste depuis un ensemble de données:**
  - **on donne à l'adaptateur une liste d'éléments à traiter et la manière dont ils doivent l'être**
  - **on passe cet adaptateur à un AdapterView.**
- **Dans ce dernier, l'adaptateur va créer un widget pour chaque élément en fonction des informations fournies en amont.**



https://openclassrooms.com/courses/creez-des-applications-pour-android/des-widgets-plus-avances-et-des-boites-de-dialogue

Les adaptateurs

- Adapter est une interface qui définit les comportements généraux des adaptateurs.

- 3 principaux adaptateurs pour construire un widget simple:

1. ArrayAdapter: permet d'afficher les informations simples ;

2. SimpleAdapter: utile dès qu'il s'agit d'écrire plusieurs informations pour chaque élément;

3. CursorAdapter: pour adapter le contenu qui provient d'une base de données.

(1) Les listes simples : ArrayAdapter

- La classe **ArrayAdapter** se trouve dans le package **android.widget.ArrayAdapter**.
- Permet d'afficher les données d'un ArrayList. Son constructeur:

public ArrayAdapter (Context contexte,
int item_layout_id, int textview_id, List<T> données).

- **context:** c'est l'activité qui crée cet adaptateur (this)
- **item_layout_id:** identifiant du layout des items qui déterminera la mise en page de l'élément. (exemple; android.R.layout.simple_list_item_1)
- **textview_id:** identifiant du TextViewdans de layout,
- **Données:** c'est la liste ou le tableau contenant les données (éléments) à afficher (ArrayList)

- List<T> signifie qu'e les objets de la liste peuvent être de n'importe quel type.

Créer un adaptateur standard pour liste (arraylist)

- ArrayAdapter<String> adapter = new ArrayAdapter<Planete> (this, android.R.layout.simple_list_item_1, android.R.id.text1, liste);

Associer ListView et l'adaptateur

- ListView lv = (ListView) findViewById (android.R.id.list);
- lv.setOnItemClickListener (this);

Le ArrayAdapter est limité à une seule chaîne par item

Exemple

- Le style d'affichage est minimaliste, seulement la liste des noms des langages de programmation.
- On ne peut pas afficher deux informations avec un ArrayAdapter.

(2) ListView

- Dans le package android.widget.ListView.
- Elles affichent les éléments les uns après les autres

Exemple

1- PYTHON

2- ANDROID

3- JAVA


```
import java.util.ArrayList;
import android.app.Activity;
import android.os.Bundle;
import android.widget.ArrayAdapter;
import android.widget.ListView;
public class ListesActivityExp extends Activity {
ListView liste = null;
  public void onCreate (Bundle savedInstanceState) {
     super.onCreate(savedInstanceState);
     setContentView(R.layout.main);
     liste = (ListView) findViewById(R.id.listView);
     List<String> exemple = new ArrayList<String>();
     exemple.add("1- PYTHON");
     exemple.add("2- ANDROID");
     exemple.add("3- JAVA");
     ArrayAdapter<String> adapter = new ArrayAdapter<String>(this,
 android.R.layout.simple_list_item_1, exemple);
     liste.setAdapter(adapter);
} }
```

Clic sur un élément de ListView:

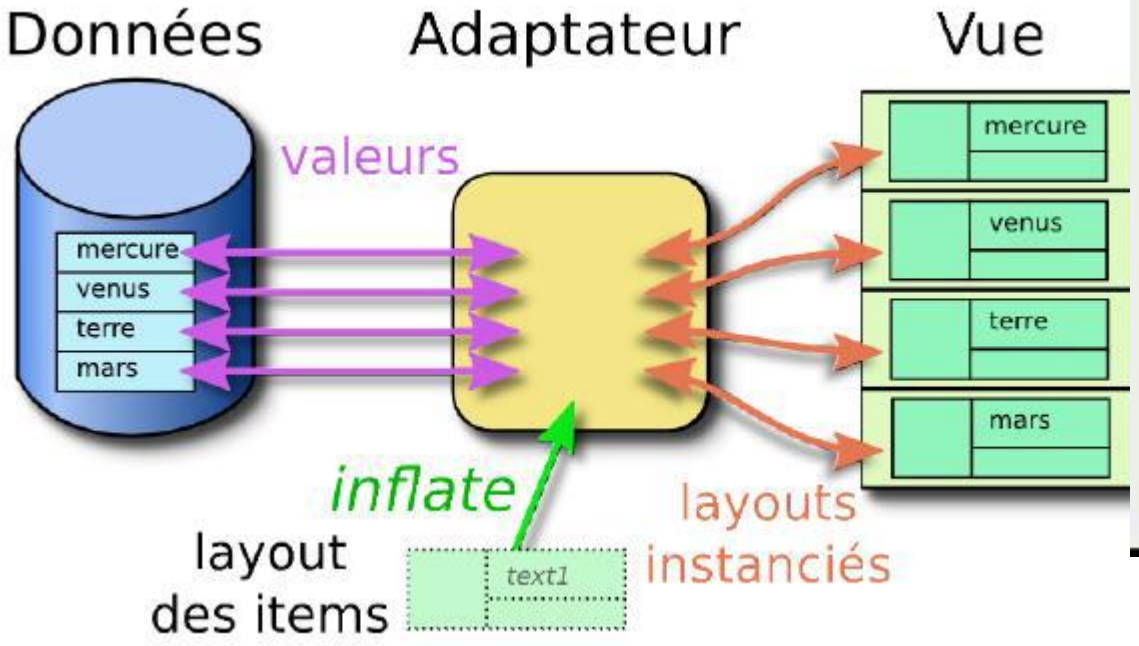
- Implementer l'interface *OnItemClickListener* et surcharger la méthode *onItemClick*


```
public class MainActivity extends Activity implements OnItemClickListener {
@Override
protected void onCreate(Bundle savedInstanceState){
        // appeler la méthode surchargée dans la superclasse
        super.onCreate(savedInstanceState);
        // mettre en place le layout contenant le ListView
        setContentView(R.layout.main);
        ListView lv=(ListView)findViewById(android.R.id.list);
        lv.setOnItemClickListener(this);
}


@Override
public void onItemClick(AdapterView<?> parent, View v, int position, long id){
        // gérer un clic sur l'item identifié par id
        ...
}
}
```

[Ref]

(2) ListView personnalisée

- Ces listes sont utilisées pour afficher les éléments les uns après les autres avec plus d'information.
- Un adaptateur personnalisé sera créé (surcharger la méthode getView() de la classe ArrayAdapter).



[Ref]

- Un Adaptateur est le conteneur des informations d'une liste.
- Un AdapterView affiche les informations et régit ses interactions avec l'utilisateur.
- C'est donc dans l'adaptateur que se trouve la structure de données qui détermine comment sont rangées les données.
- Dans notre adaptateur se trouvera une liste de contacts sous forme de ArrayList.

Université d'Alger 1 Benyoucef Ben Kheda
Faculté des Sciences
Département MI

L3 - SI

Développement d Applications Mobiles

DR. AICHA BOUTORH

2017/2018

Composition d'une Application

Une Application

- La majorité des applications (si ce n'est toutes) contiendront plusieurs activités.
- Une application complexe peut aussi contenir plusieurs composants
- Les composants de l'application sont les éléments constitutifs essentiels d'une application Android.
- Chaque composant est un point d'entrée à travers lequel le système ou un utilisateur peut entrer dans votre application.
- Certains composants dépendent des autres.

Composants d'Une Application

- Il existe quatre types de composants d'application:

  - **Activités:** chaque Activité gère un écran d'interaction avec l'utilisateur et est définie par une classe Java.
  - **Services:** ce sont des processus qui tournent en arrière-plan
  - **Récepteurs d'annonces** : pour gérer des événements globaux envoyés par le système à toutes les applications.
  - **Fournisseurs de contenu:** représentent une sorte de base de données.

- Chaque type sert à un objectif distinct et a un cycle de vie distinct qui définit comment le composant est créé et détruit.

Autorisation d'Une Application

- Si une application Android a besoin d'utiliser une information endors de son scope elle aura besoin d'une **permission**.
- Une application doit déclarer les autorisations dont elle a besoin : *caméra, carnet d'adresse, accès à internet, GPS, …*
- pour demander un accès à internet, on indiquera dans le manifeste la ligne:

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

Liste des permissions existantes:
https://developer.android.com/reference/android/Manifest.permission.html

Démarage d'Une Application

- Le système Android lance l'activité qui est marquée dans **AndroidManifest.xml** :
  - **action=MAIN**
  - **catégorie=LAUNCHER** dans

- D'autres activités par la suite peuvent être démarrées. Chacune se met « devant » les autres comme sur une pile.

- Une activité marquée comme **démarrable**:

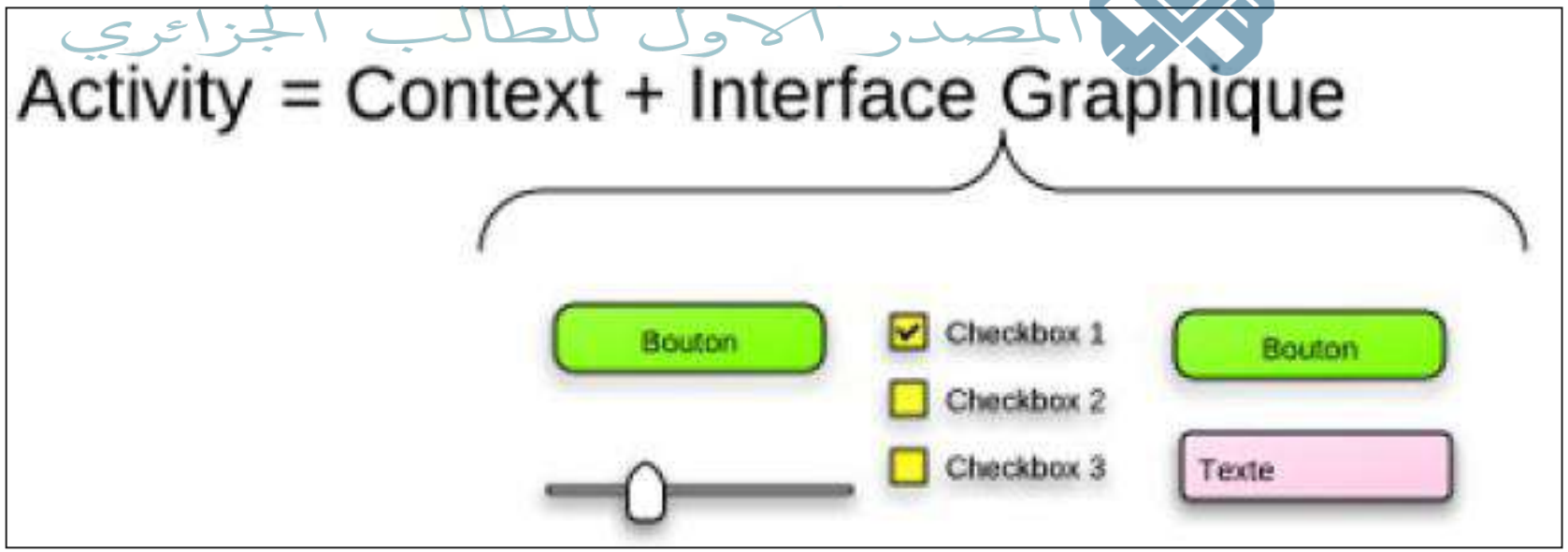
```
<activity android:name=".MainActivity" ...>
  <intent-filter>
    <action android:name= "android.intent.action.MAIN" />
    <category android:name= "android.intent.category.LAUNCHER" />
  </intent-filter>
</activity>
```

Navigation entre Activités

- Une application est un assemblage de fenêtres entre lesquelles il est possible de naviguer.
- Les différentes fenêtres sont appelées des activités.
- Un moyen efficace de différencier des activités: si les interfaces graphiques sont différentes, alors il s'agit d'activités différentes.
- L'application ne peut en afficher qu'une activité à la fois.
- Pour naviguer entre les activités:
  (Lancer Act2 à partir de Act1)

```
Intent intent= new Intent (this, Act2.class);
startActivity (intent);
```

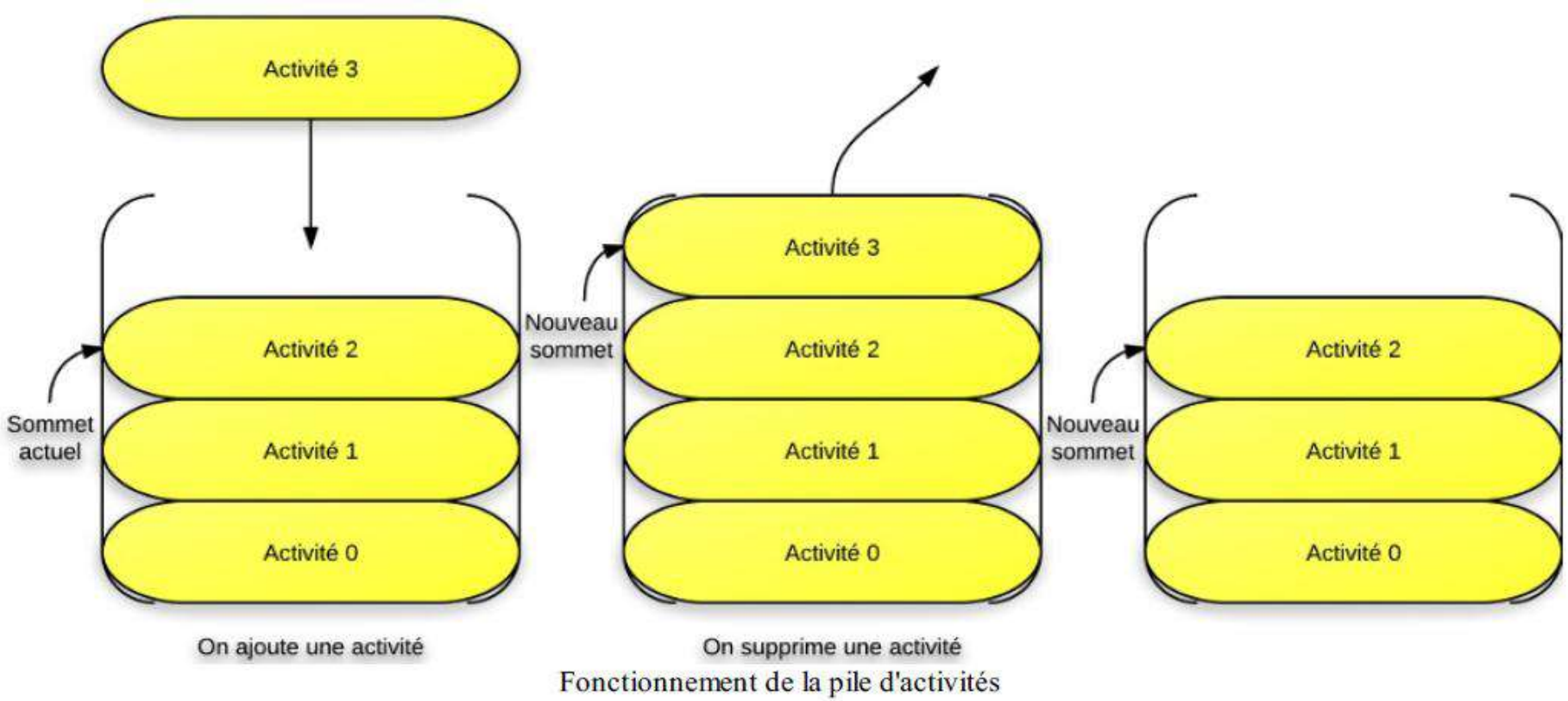
ACTIVITÉ



Une activité est constituée du contexte de l'application et d'une seule et unique interface graphique

Etats d'une Acitvité

- À tout moment votre application peut laisser place à une autre application, qui a une priorité plus élevée.
- Votre activité existera dans plusieurs états au cours de sa vie
- Quand une application se lance, elle se met tout en haut de la pile d'activités.



Fonctionnement de la pile d'activités

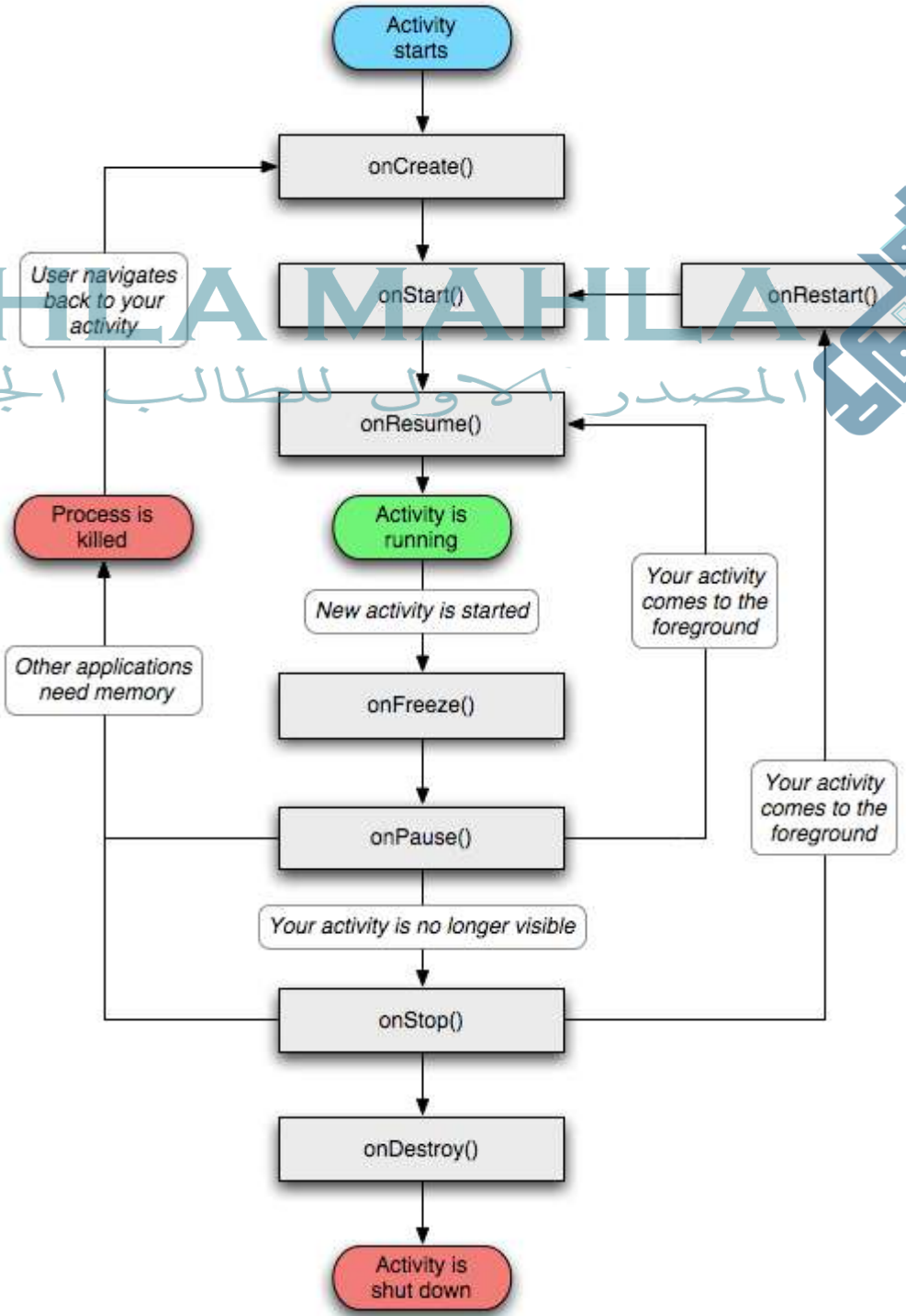
https://openclassrooms.com/courses/creez-des-applications-pour-android/votre-premiere-application-1

- **Une activité se trouve dans l'un de ces états :**
- **Active** (**active** ou **running** ou **resumed**) : elle est sur le devant, visible en totalité. C'est ce que l'utilisateur utilise pour le moment.
- **En pause** (**paused**) : partiellement visible et inactive. Ce n'est pas sur cette activité qu'agit l'utilisateur, car une autre activité est venue devant
- **Stoppée** (**stopped**) : totalement invisible et inactive on ne peut plus la voir du tout, ses variables sont préservées mais elle ne tourne plus. Puisque l'utilisateur ne peut pas la voir, il ne peut pas agir dessus.

- **Événements de changement d'état:**
  "La classe **Activity** reçoit des événements de la part du système Android par l'appel des fonctions appelées ***callbacks***. "

- Exemples :
- **onCreate** Déclenchement de la création d'une activité par l'arrivée d'un Intent.
- **onPause** Le système prévient l'activité qu'une autre activité est passée devant, Les informations doivent être enregistrées au cas où l'utilisateur ne revient pas.

Cycle de Vie d'une Acitvité



Cycle de Vie et Callbacks

- La transition entre chaque étape implique qu'une méthode est appelée par votre application. Cette méthode partage le même nom que l'état traversé.
- Par exemple la méthode ***onCreate*** est appelée à la création**.**
- La période **active** peut être interrompue par la période **suspendue**.
- La période **suspendue** peut être interrompue par la période **arrêtée**.

- **onCreate**: appelée à la création de l'activité

- **onStart:** appelée après la méthode **onCreate** ou **onRestart**

- **onResume:** appelée après la méthode **onStart** ou **onPause**.
  Cette méthode est exécutée à chaque fois que votre activité retourne au premier plan. Aussi à chaque lancement, c'est pourquoi on l'utilise pour initialiser les ressources qui seront coupées dans le **onPause()**.

- **onPause:** appelée pour arrêter l'activité lorsque une autre activité prend la main pour passer en premier plan. On utilise la méthode onPause() pour arrêter des animations, libérer des ressources « *GPS ou la caméra* », arrêter des tâches en arrière-plan ...
  Android conserve une copie fonctionnelle de l'activité qui sera restaurée au retour, il n'est pas nécessaire de sauvegarder des données dans cette méthode.

- **onStop:** permet de libérer certaines ressources

- **onRestart:** appelée si l'activité revient au premier plan (suivi d'un appel à **onStart**)

- **onDestroy:** appelée après un appel de la méthode ***finish()*** ou par le système si celui-ci a besoin de ressources (permet de libérer des ressources liées à l'activité).

Cycle de Vie et Callbacks


```
public class CycleActivity extends AppCompatActivity {
    String msg = "Android : ";
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        Log.d(msg, "The onCreate() event");
    }
```

```
@Override
protected void onStart() { super.onStart(); Log.d(msg, "The onStart() event");}
@Override
protected void onResume() {super.onResume();Log.d(msg, "The onResume() event");}
@Override
protected void onPause() {super.onPause();Log.d(msg, "The onPause() event");}
@Override
protected void onStop() {super.onStop();Log.d(msg, "The onStop() event");}
@Override
public void onDestroy() {super.onDestroy();Log.d(msg, "The onDestroy() event");}
```

[Ref]

De la naissance à la mort

- ***onCreate():*** Première méthode qui est lancée au démarrage de l'activité, c'est l'endroit pour:
  - initialiser l'interface graphique,
  - démarrer les tâches d'arrière-plan
  - récupérer des éléments de la base de données, ...
- L'appel de la méthode **public void finish ()** permet de passer directement à la méthode **onDestroy()**, qui symbolise la mort de l'activité.

- ***onDestroy()*** : Il existe trois raisons pour lesquelles l'application atteint la méthode ***onDestroy*** et sera terminée :
  - Si le téléphone manque de mémoire et qu'il ait besoin d'arrêter l'application pour en récupérer.
  - Si l'utilisateur presse le bouton Arrière et il n'y a pas une activité précédente ou l'activité actuelle ne permet pas de retourner à l'activité précédente, alors l'application sera quittée.
  - Enfin, si vous faites appel à la méthode **public void finish()**

- Android passera d'abord par **onPause()** et **onStop()** dans tous les cas, à l'exception de l'appel de la méthode ***finish()*** ! où on passe directement au ***onDestroy().***

Enregistrement de valeurs d'une exécution à l'autre

- Si l'application est quittée correctement, alors Android ne garde pas en mémoire de traces des activités.
- Si Android a dû tuer le processus, alors il va garder en mémoire une trace des activités afin de pouvoir les restaurer. Au prochain lancement de l'application, le paramètre de type **Bundle** de la méthode **onCreate(Bundle)** sera peuplé d'informations enregistrées sur l'état des vues de l'interface graphique.

- Il est possible de sauver des informations d'un lancement à l'autre de l'application dans un **Bundle**.
- C'est un container de données quelconques, sous forme de couples **("nom", valeur).**

- Il est possible de sauvegarder d'autres informations qui, elles, ne sont pas sauvegardées par défaut grâce à une méthode qui est appelée à chaque fois qu'il y a des chances pour que l'activité soit tuée (après **onPause**).
- Cette méthode **protected** void **onSaveInstanceState**(Bundle State).


```
static final String ETAT_SCORE = "ScoreJoueur"; // nom
private int mScoreJoueur = 0; // valeur
@Override
public void onSaveInstanceState(Bundle etat) { // enregistrer l'état courant
etat.putInt(ETAT_SCORE, mScoreJoueur);
super.onSaveInstanceState(etat); }
```

- **Restaurer l'état au lancement**

```
@Override
protected void onRestoreInstanceState(Bundle etat) {
super.onRestoreInstanceState(etat); // restaurer l'état précédent
mScoreJoueur = etat.getInt(ETAT_SCORE); }
```

- La méthode onRestoreInstanceState(Bundle) est appelée après onStart()

Les Fragments

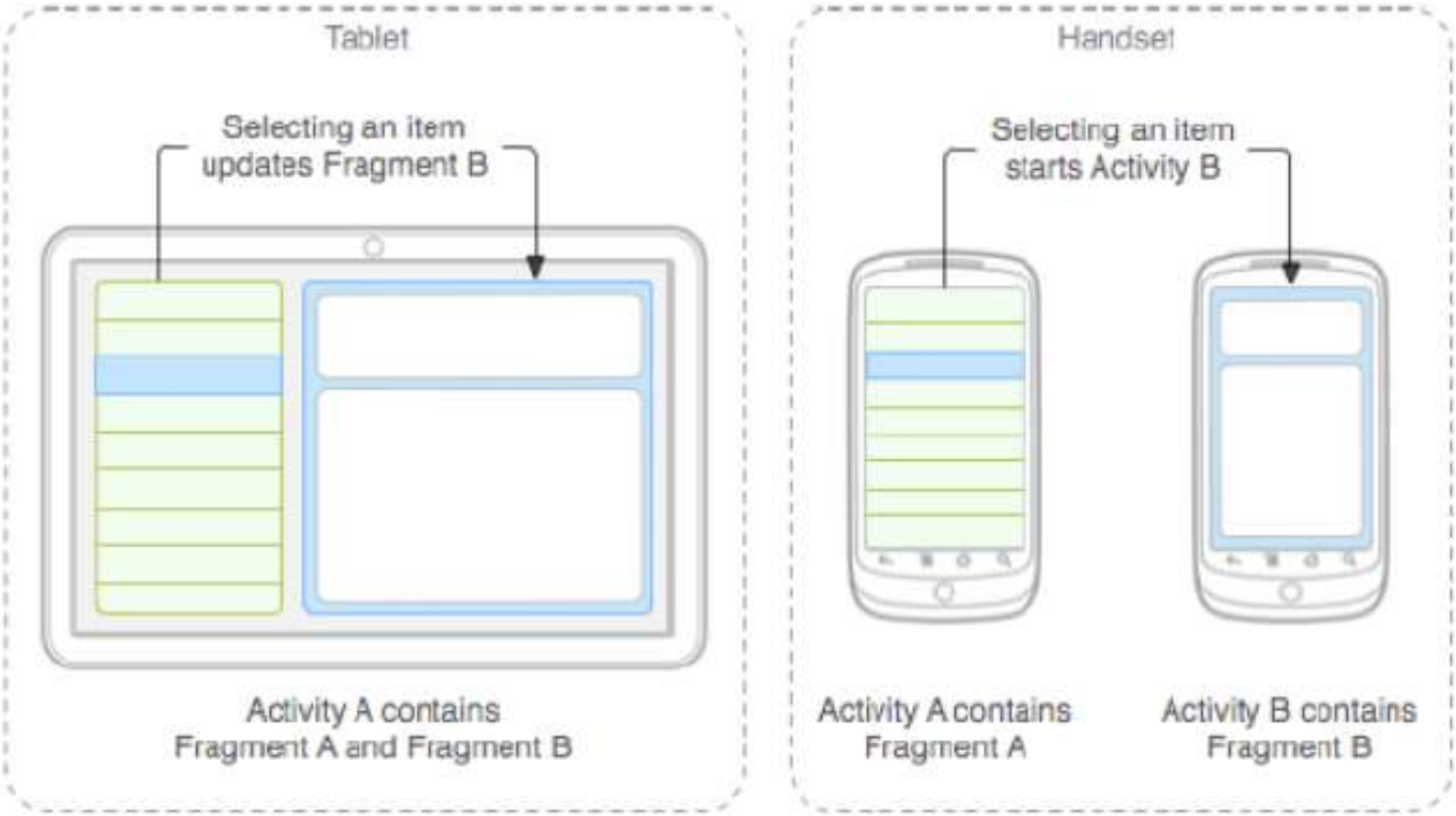
- La philosophie découle du problème qui consiste à adapter une application Android à toutes les tailles d'appareils existantes.

- Un fragment est un code qui est :
  - lié à une interface,
  - possède un cycle de vie.

- Un fragment est une sorte de mini-activité (portion d'une IHM + portion d'un comportement)
- Permet la réutilisation
- Ajout statique ou dynamique
- Permet de mieux gérer les écrans de tailles différentes

- Comparant aux activités, les fragments ne sont pas liés à un écran, mais **à un fragment de l'écran**.
- Une activité peut donc afficher un ou plusieurs fragments.
- Le fragment est embarqué dans une activité et suit le cycle de vie de l'activité

- Une interface devient plus souple avec des fragments.
- Selon la taille d'écran, on peut afficher une liste et les détails, ou séparer les deux:
  - un fragment pour afficher une liste,
  - un fragment pour afficher les détails des éléments d'une liste)

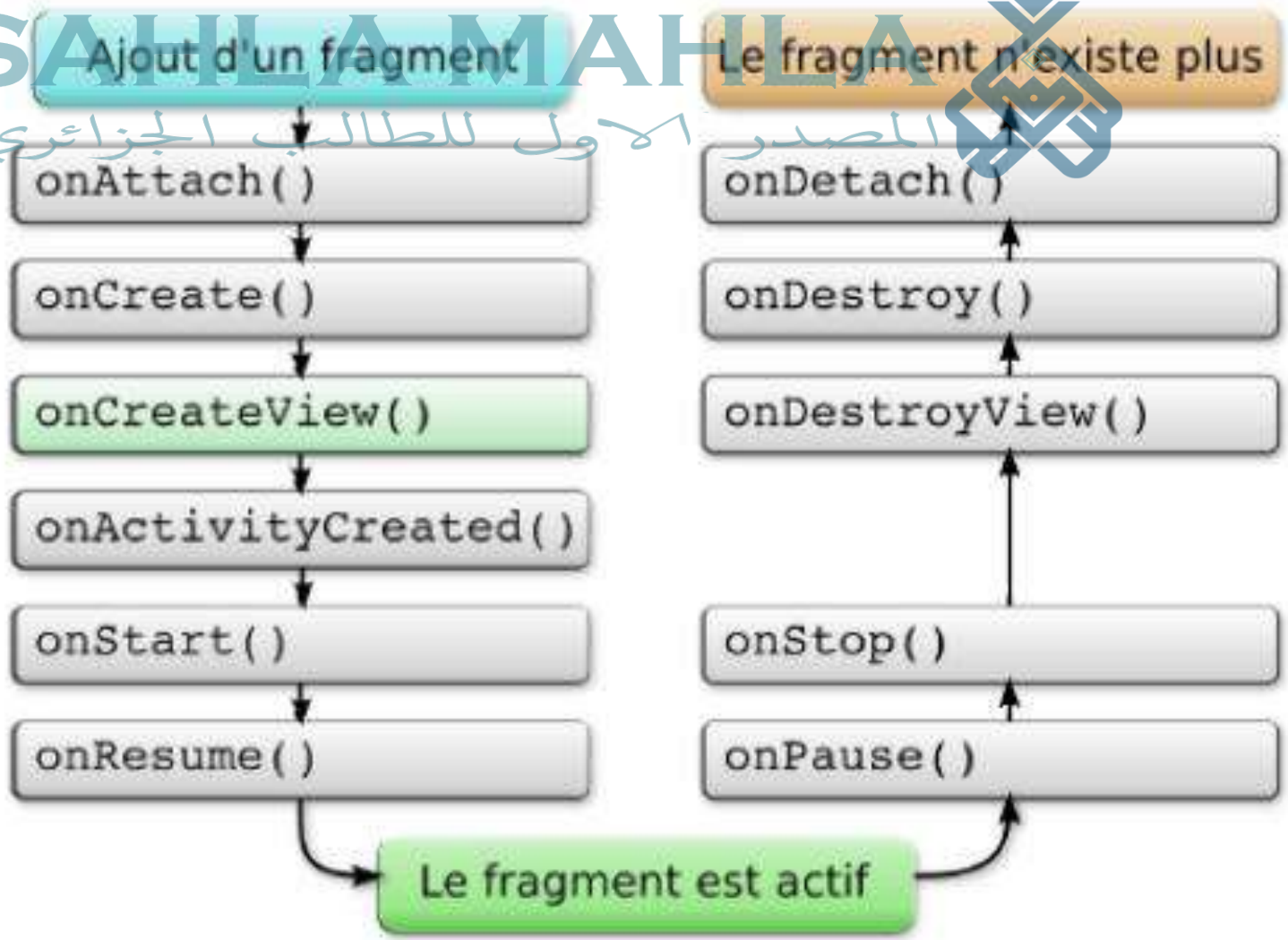


Structure d'un Fragment

- Un fragment est un layout, et un comportement (code java) et considéré comme une activité très simplifiée.
- Un fragment minimal est :

```
public class MyFragment extends Fragment {
@Override
public View onCreateView(LayoutInflater inflater, ViewGroup container, Bundle savedInstanceState){
return inflater.inflate(R.layout.my_fragment, container, false);
}
}
```

Sycle de Vie des Fragments



- **onAttach:** pour récupèrer un pointeur vers l'activité contenante;
- **onCreate:** pour instancier les objets non graphiques;
- **onCreateView:** pour instancier la vue et lescomposants graphiques;
- **onActivityCreate:** pour récupèrer un pointeur sur l'activité (qui est construite),
- **onStart:** pour lancer les traitements;
- **onResume:** Fragment actif;
- **onPause:** Fragment en pause
- **onStop:** pour arrêter les traitements et libèrer les objets;
- **onDestroyView:** la vue est détachée de celle de l'activité, pour libirer la mémoire des objets graphiques;
- **onDestroy, onDetach:** fragment tué et détaché.

Intégrer un fragment dans une activité

- Un fragment ne peut apparaître que dans le cadre d'une activité, de lui-même il n'est pas capable de s'afficher.
- Un fragment dans une activité est comme une sorte de vue interne.

- Un fragment est intégré dans une activité de deux manières :

➢ Statiquement: les fragments à afficher sont prévus dans le layout de l'activité.

➢ Dynamiquement: les fragments sont ajoutés, enlevés ou remplacés selon les besoins.

Comment utiliser les Fragments ?

- Pour créer différentes mises en page avec des fragments, vous pouvez :

  - Utiliser une activité qui affiche deux fragments sur les tablettes et un seul sur les téléphones. Cela nécessite que le fragment ne soit pas déclaré dans le fichier de mise en page pour basculer les fragments dans l'activité en cas de besoin, et il peut être supprimé de façon dynamique.

  - Utiliser des activités séparées pour héberger chacun des fragments sur un téléphone. Changer un fragment nécessite de démarrer une autre activité qui héberge l'autre fragment.

Exemple de fragment statique dans une activité

Fragment

- *fichier layout/my_fragment.xml*

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
android:layout_width="match_parent"
android:layout_height="match_parent"
android:orientation="vertical" >
<Button
android:id="@+id/button1"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:text="Clic" />
</LinearLayout>
```

Exemple de fragment statique dans une activité

Fragment

- *fichier MyFragment.java*

```
public class MyFragment extends Fragment {
        @Override
        public View onCreateView(LayoutInflater inflater, ViewGroup container,
        Bundle savedInstanceState){
        View view = inflater.inflate(R.layout.my_fragment, container,false);
        Button btn= (Button) view.findViewById(R.id.button1);
        btn.setOnClickListener(new OnClickListener() {
        public void onClick(View v) {
        Toast.makeText(getActivity(), "Clic !!!", Toast.LENGTH_LONG).show();
        }
        });
        return view;
        }
}
```

Exemple de fragment statique dans une activité

Activité

```
<LinearLayout xmlns:android="....."
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:orientation="vertical" >
<fragment
android:id="@+id/simple_fragment"
android:name="com.td.myfragment.MyFragment"
android:layout_width="wrap_content"
android:layout_height=" wrap_content "/>
</LinearLayout>
```

```
public class MainActivity extends Activity {
@Override
public void onCreate(Bundle savedInstanceState) {
super.onCreate(savedInstanceState);
setContentView(R.layout.main);
}
}
```

- Pour utiliser votre nouveau fragment, vous pouvez l'ajouter de manière statique au fichier de mise en page XML.
- Si un fragment est défini dans le fichier XML, l'attribut android:name indique le nom de la classe correspondante.

[Ref]

Fragment Dynamiquement

- Pour définir des fragments dynamiquement, on fait appel au **FragmentManager** (Il gère l'affichage des fragments).
- L'ajout et la suppression de fragments se fait à l'aide de ***transactions*** (c'est l'association entre un ***FrameLayout*** vide et un **fragment**).

Exemple:

Soit un layout d'une activité contenant deux FrameLayout vides :

```
<LinearLayout xmlns:android="..."
android:orientation="horizontal" ... >
        <FrameLayout android:id="@+id/fragmentA" ... />
        <FrameLayout android:id="@+ id/fragmentB" ... />
</LinearLayout>
```

On peut dynamiquement attribuer un fragment à chacun.

- **Exemple (Suite)**
- Soient les deux fragments ***MyFragment1*** et ***MyFragment2***
- Dans l'activité, on peut remplacer dynamiquement les FrameLayout par les fragments comme suite:


```
// gestionnaire de fragment
FragmentManager manager = getFragmentManager();
// transaction
FragmentTransaction trans = manager.beginTransaction();
// mettre les fragments dans les réceptacles
trans.replace(R.id.fragmentA, new MyFragment1());
trans.replace(R.id.fragmentB, new MyFragment2());
trans.commit();
```

Composition d'une Application

SERVICE

- Composant applicatif indépendant qui s'exécute en arrière-plan, et qui ne possède pas d'interface graphique pour interaction directe avec l'utilisateur

Service

- Deux états possibles pour un service:

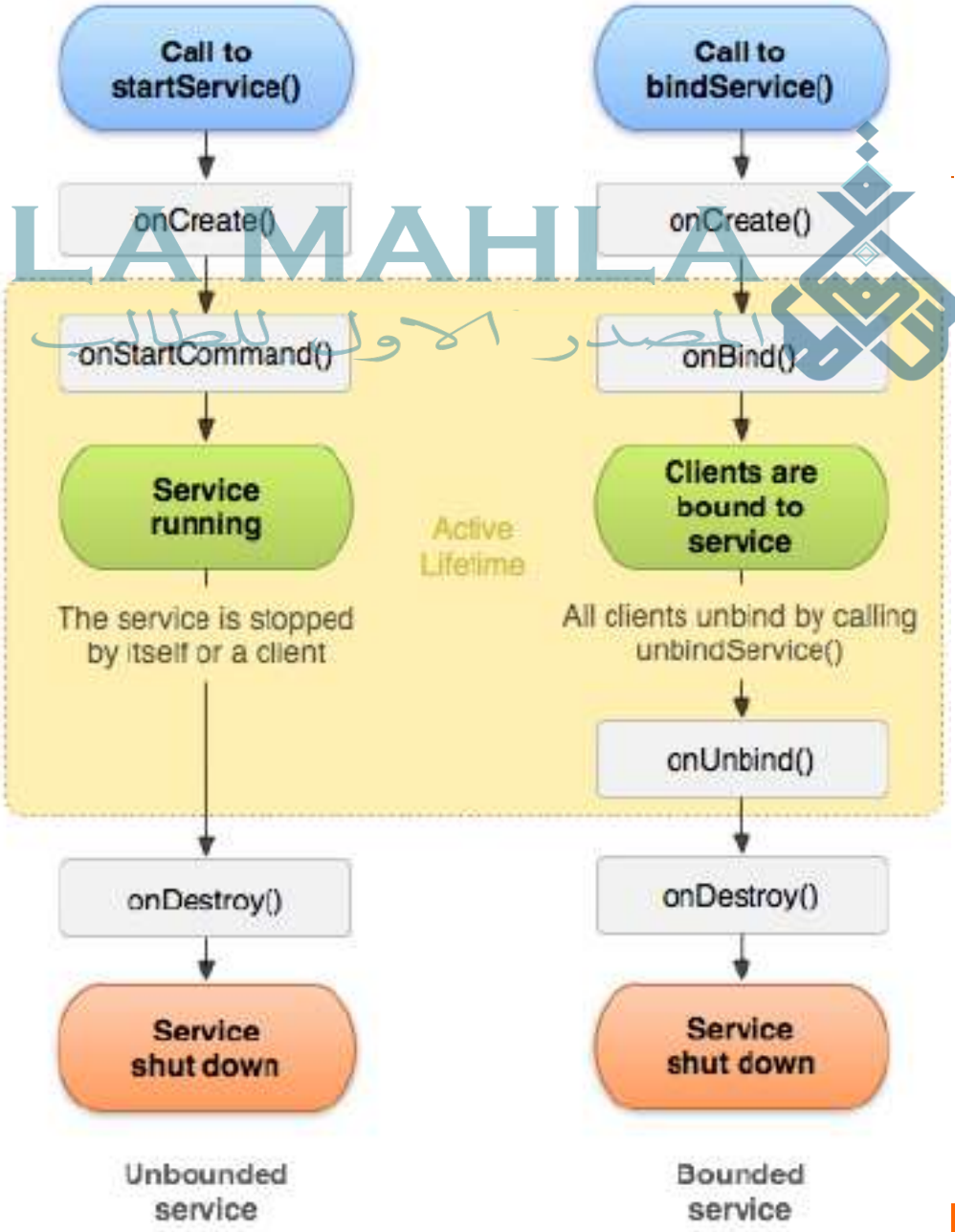
- **Unbounded**: état **Démarré,** en appelant la méthode startService(). Un service peut s'exécuter en arrière-plan indéfiniment après son lancement, même si le composant qui l'a démarré est détruit.

- **Bounded**: état **Connecté,** en appelant la méthode bindService(). Un service connecté offre une interface qui permet d'interagir avec le service, envoyer des requêtes et obtenir des résultats.

- La déclaration du service dans le Manifest:

```
<service
    android:name=".MyService"
    android:enabled="true"
    android:exported="true">
</service>
```

- Cycle de vie d'un service non connecté



- Cycle de vie d'un service connecté

Démarrer le service depuis l'activité avec la méthode *startService()*


```
Intent I = new Intent (this, MyService.class);

startService(I);
```

- Méthode de la classe Activity
- invoque les mèthodes:
- ***onCreate()***

  puis
- ***onStartCommand()***

Arrêter un service démarré

avec la méthode *startService()*


```
Intent I = new Intent (this, MyService.class);
stopService(I);
```

- Le service est arrêté par le système si ce dernier a besoin de mémoire
- Un composant appel ***stopService()*** pour arrêter un service
- Le service peut s'arrêter lui-même avec la méthode ***stopSelf()***
- si d'autres composants sont connectés au service, le service n'est pas arrêté.

Exemple 1:

Code de service

```
public class MyService extends Service {
    @Override
    public void onCreate(){
        super.onCreate();
        Log.d(this.getClass().getName(), "Création du service: onCreate");
    }
    @Override
    public int onStartCommand(Intent intent, int flags, int startId){
        Log.d(this.getClass().getName(), "Démarrage du service: onStartCommand");
         .. .. .. .. ..
        return START_NOT_STICKY;
    }
    @Override
    public void onDestroy(){
        Log.d(this.getClass().getName(), "Arrete du service: onDestroy");
    }
    @Override
    public IBinder onBind(Intent intent) {
        return null;
    }
    @Override
    public boolean onUnbind(Intent intent) {
        return true;
    }
}
```


[Ref]

Exemple 1:

Code de l'Activité (démarrer arrêter le service)


```
public class MainActivity extends AppCompatActivity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
    }
    public void demarrerService(View v){
        Intent intent=new Intent(this, MyService.class);
        startService(intent);
    }
    public void arreterService(View v){
        Intent intent=new Intent(this, MyService.class);
        stopService(intent);
    }
}
```

Démarrer le service depuis l'activité

avec la méthode *bindService()*

- Invoque les méthodes:
  - ***onCreate()*** puis ***onBind()***
- Le service est démarré et lié avec l'activité
- L'activité peut interagir avec le service à travers la **connexion (Bind)**

Arrêter un service démarré

avec la méthode *bindService()*

- Le service est arrêté par un **unbindService**
- Le service est réellement détruit si aucun client n'est lié au service

Créer un Service Connecté

- Implémenter la méthode **onBind** au niveau du service.
- La méthode **onBind** retourne un objet de type **Ibinder** propre au service
- Le **Ibinder** sera utilisé par l'activité pour récupérer la référence vers le service.
- La classe qui implémente **Ibinder** possède une méthode renvoyant une interface vers le service

Connecter l'Activité au Service

- Implémenter l'interface ***ServiceConnection***

- Créer la connexion au niveau de l'activité, en redéfinissant les méthodes:
  - ***onServiceConnected***: appelée lors de la connexion de l'activité au service
  - ***onServiceDisconnected***: appelée lors de la déconnexion de l'activité du service

Exemple 2:

Code de service

```
public class MyService extends Service {

    //********** L'implémentation de l'interface Ibinder *********
    private final IBinder mBinder = new MyServiceBinder();
    public class MyServiceBinder extends Binder {
        MyService getService() {
            return MyService.this;
        }
    }
    //*****************************************************************

        .. .. ..

    @Override
    public IBinder onBind(Intent intent) {
        Log.d(this.getClass().getName(), "Connexion au service: onBind");

        return mBinder;
    }
    @Override
    public boolean onUnbind(Intent intent) {
        Log.d(this.getClass().getName(), "Deconnexion de service: onBind");
        return true;
    }
}
```


[Ref]

Exemple 2:

Code de l'Activité (se connecter et se déconnecter du service)


```
private MyService monService;
// L'objet ServiceConnection gère la connexion entre l'activité et le service.
private ServiceConnection maConnexion = new ServiceConnection() {
// Méthode appelée lorsque la connexion est établie. Récupère la référence vers le
service associé.
public void onServiceConnected(ComponentName className, IBinder service) {
     monService = ((MyService.MyServiceBinder)service).getService();}
        // Méthode appelée lorsque la connexion est terminée.
public void onServiceDisconnected(ComponentName className) {
       monService = null; }};

 public void connectService(View v){
    Intent i=new Intent(this, MyService.class);
    bindService(i,maConnexion, Context.BIND_AUTO_CREATE);
 }

 public void deconecterService(View v){
    Intent i=new Intent(this, MyService.class);
    unbindService(maConnexion);
 }
```

Réception de diffusion «BroadcastReceiver »

- Ecouter et réagir aux événements.
- Les événements sont des Intents envoyées par ***sendBroadcast()***
- les événements) sont réceptionnées par une classe héritant de **BroadcastReceiver**

- *Evènements Système* : batterie faible, mise en veille, réception d'un SMS, etc.
- *Evènements entre applications*.

- Déclaration de BroadcastReceiver dans le fichier *AndroidManifest.xml*


```
<receiver
   android:name=".MyReceiver"  ... ... >
   <intent-filter>
     <action android:name ="android.intent.action.BOOT_COMPLETED" />
   </intent-filter>
</receiver>
```

- ACTION_BOOT_COMPLETED: permet de déclencher un service lorsque l'appareil a complètement démarré

Evénements système

- Quelques événements système importants:

Evénement	Description
Intent.ACTION_BOOT_COMPLETED	Boot terminé. Nécessite la permission android.permission.RECEIVE_BOOT_COMPLETED
Intent.ACTION_POWER_CONNECTED	L'alimentation est connectée à l'appareil.
Intent.ACTION_POWER_DISCONNECTED	L'alimentation a été déconnectée de l'appareil.
Intent.ACTION_BATTERY_LOW	Déclenché sur une batterie faible.
Intent.ACTION_BATTERY_OKAY	État de la batterie bien nouveau.

Exemple: Auto démarrage d'un service

- Démarrer un service au démarrage du système.
- Pour cela, il faut créer un **BroadcastReceiver** qui va réagir à l'action **BOOT_COMPLETED** et lancer l'**Intent** au travers de la méthode **startService**.
- L'application nécessite la permission: **"android.permission.RECEIVE_BOOT_COMPLETED"**
- Le fichier Manifest :

```
<uses-permission android:name="android.permission.RECEIVE_BOOT_COMPLETED" />
<application
        .. .. .. >
<receiver
    android:name=".MyReceiver"
    android:enabled="true"
    android:exported="true">
    <intent-filter>
        <action android:name="android.intent.action.BOOT_COMPLETED" />
    </intent-filter>
</receiver>
<application>
```

[Ref]

Exemple (Suite): Le BoadcastReceiver:


```
public class MyReceiver extends BroadcastReceiver {
    public MyReceiver() {}
    @Override
    public void onReceive (Contextcontext, Intentintent)
      {
       Intent i = new Intent(context, MyService.class);
       context.startService(i);
       }
}
```

Les fournisseurs de contenu « ContentProvider »

- Un fournisseur de contenu gère l'accès à un référentiel central de données.
- Un fournisseur fait partie d'une application Android, qui fournit souvent sa propre interface utilisateur pour travailler avec les données

ContentProvider

- Parfois, il est nécessaire de partager des données entre les applications. C'est là que les fournisseurs de contenu deviennent très utiles.

- ContentProvider est le moyen le plus connu de partage de données entres différentes applications, il sert à:
  - Stocker,
  - Récupérer, et
  - Partager des données.

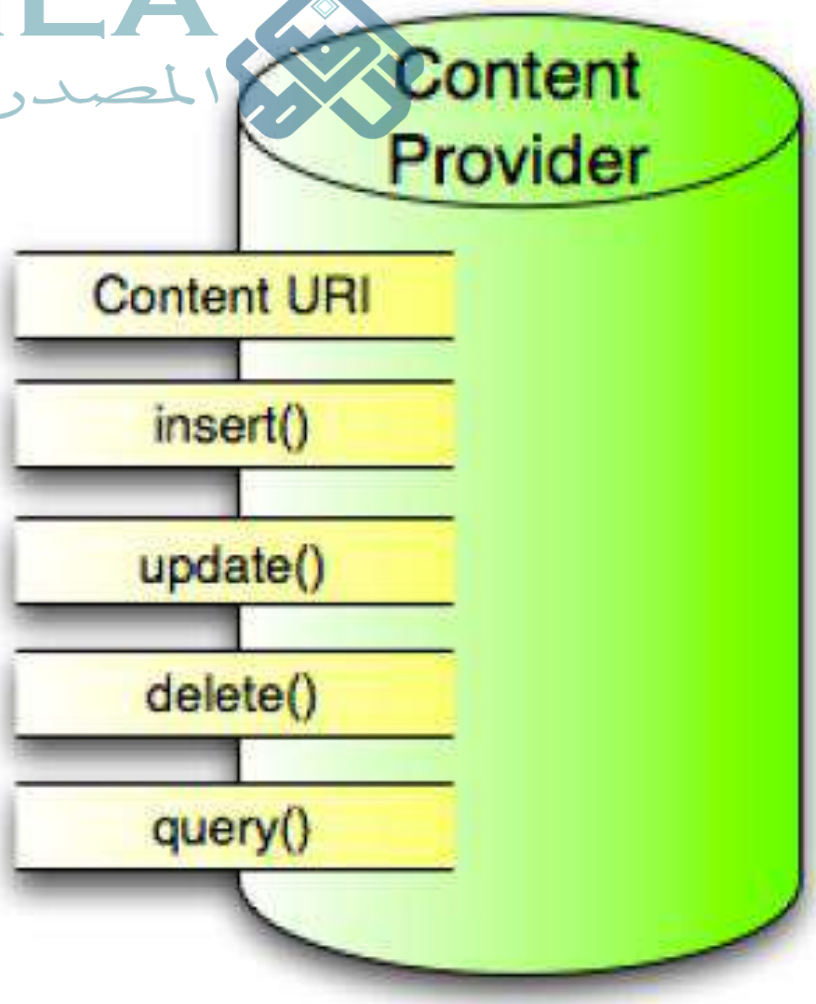
- Il rend les données accessibles à toutes les applications, une application peut alors lire et modifier les données d'une autre application.

- Par défaut Android expose plusieurs contentProvider gérants les contacts d'un téléphone, images, vidéos, audio, ...

- Un **contentProvider** se compose de:

- •**Uri** : Pour interroger un fournisseur de contenu, vous spécifiez la chaîne de requête sous la forme d'un URI “séquence de caractères qui identifie une ressource”

- •**Méthodes** : Insert, Update, Delete, Query



•Pour créer un ContentProvider, on doit:

- Mettre en place un système pour stocker les données (généralement le **SQLite**)

- Etendre la classe ***ContentProvider***.

- Déclarer le **Content Provider** dans le manifest.

- Avec la balise provider:

```
<provider
    android:name=".MyContentProvider"
    android:authorities="myuri"
    ..........></provider>
```

Un Content Provider doit surcharger **les 6 méthodes** suivantes:

- **query():** retourne un objet **Cursor** sur lequel nous pouvons itérer pour récupérer les données.
- **insert():** est utilisé pour rajouter des données.
- **update():** est utilisé pour mettre à jour une données déjà existante.
- **delete():** permet de supprimer une donnée du ContentProvider.
- **getType():** retourne le type des données contenues dans le ContentProvider.
- **onCreate():** Appeler afin d'initialiser le Content Provider

Université d'Alger 1 Benyoucef Ben Kheda
Faculté des Sciences
Département MI

L3 - SI

Développement d Applications Mobiles

DR. AICHA BOUTORH

2017/2018

Bases de Données SQLite

SQLiteDatabase.

- Nous avons besoin d'un moyen efficace pour stocker des données complexes et d'y accéder.
- Nous avons besoin des bases de données, qui sont optimisées.
- **SQLite** joue un grand rôle pour sauvegarder des données complexes et répétées

Bases de Données

- Une base de données permet de stocker un ensemble d'informations de manière structurée et organisée suivant un **schéma**.
- La base de données est constituée de **Tables**
- Une table regroupe des ensembles d'informations similaires et composée de:
  - Tuples ou Enregistrements *(lignes)*
  - Attributs qui caractérise chaque tuple *(Colonnes)*

- *Exemple:* ***Table Utilisateur***

Nom	Age	Métier
Ahmed	45	Auteur
Khadidja	22	Etudiante
Karim	36	Médecin
Asma	30	Enseignante
.....	.....	.....

SQLiteDatabase.

- Les bases de données pour Android sont fournies à l'aide de **SQLite**.

- **SQLite** est un SGBD très compact et très efficace pour les applications embarquées, on le trouve aussi dans Skype, Adobe Reader, Firefox, etc.

- **SQLite** est un **SGBD relationnel**, léger, gratuit et Open Source. Il fournit un support de bases de données relationnelles simplifiée pour tenir sur une tablette.

SQLite

- Contrairement à MySQL par exemple, SQLite ne nécessite pas de serveur pour fonctionner, ce qui signifie que son exécution se fait dans le même processus que celui de l'application, il stocke les données dans un fichier portable .
- Il possède peu de types de données
- Pas de réglages pour améliorer les performances
- Pas de gestion des utilisateurs (pas de sécurité)

SQLite pour Android

- SQLite a été inclus dans le coeur d'Android
- Chaque application peut avoir sa propre base.
- Une application peut stocker des données dans une ou plusieurs bases de données SQLite
- Une base de données est un fichier ***.db** créé par une application. Les bases de données sont stockées, de manière générale, dans les répertoires de la forme **/data/data/<package>/databases/*NOM_BDD*** .
- Les bases de données sont privées à l'application qui les a créées
- Pour les partagées (les rendre publiques), on doit passer par un fournisseur de contenu (ContentProvider)

Type de données

- Cinq types de données pour **SQLite** :

1) ● **INTEGER** : nombres entiers (sans virgule), signés ou non

2) ●**REAL** : nombres réels (avec virgule)

3) ●**TEXT** : données textuelles - les chaînes de caractères-

4) ●**BLOB** : stockage de données sous forme binaire

5) ●**NULL** : donnée nulle

Création d'une Table

Code : SQL

```
CREATE TABLE nom_de_la_table (
   nom_du_champ_1 type {contraintes},
   nom_du_champ_2 type {contraintes},
   ...);
```

Contraintes:

- **PRIMARY KEY:** désigner la clé primaire sur un attribut ;
- **NOT NULL:** indiquer que cet attribut ne peut être **NULL** ;
- **CHECK:** vérifier que la valeur de cet attribut est cohérente ;
- **DEFAULT:** préciser une valeur par défaut.

Suppression d'une Table

DROP TABLE table_name;

Insertion

INSERT INTO TABLE_NAME [(*column1*, *column2*, *column3*,...)] **VALUES** (*value1*, *value2*, *value3*,...);

INSERT INTO TABLE_NAME **VALUES** (*value1*,*value2*,*value3*,...);

Sélection

SELECT Attribut1, … **FROM** table_name;

SELECT Attribut1, … **FROM** table_name
WHERE [*CONDITION*];

SELECT Attribut1, … **FROM** table_name
WHERE [*condition1*] **AND** [*condition2*] …
OR [*conditionN*];

Mise à Jour

UPDATE table_name
SET attribut1 = value1,....
WHERE [*condition*];

Suppression

DELETE FROM table_name
WHERE [*condition*];

Toute autre requête fonctionne

SELECT AVG (age) **FROM** Person **GROUP BY** ……;

SELECT COUNT(*) **FROM** Person **WHERE** name= “M”;

DELETE FROM Book **WHERE** Title **NOT LIKE** “*DB*” ;

SELECT * **FROM** Product **WHERE** Price> 5 **ORDER BY** Price;

……

SQLite dans une application Android

- **La classe SQLiteDatabase:**
  - Représente une BDD.
  - Ses méthodes permettent d'exécuter une requête.
    - –**void execSQL (String sql)**
    - –**Cursor rawQuery (String sql, …)**

- **La classe Cursor:**
  - Permet de parcourir le résultat d'une requête SELECT.

Les méthodes de la classe SQLiteDatabase

La méthode execSQL

- Pour exécuter une requête SQL pour laquelle on ne souhaite pas de réponse: *CREATE, INSERT*. . .

➢ **Void execSQL (String sql)** :

- Expl : **bdd.execSQL("DROP TABLE Person");**

➢ **Void execSQL (String sql, String[] Args)** :

- Expl : **bdd.execSQL ("DELETE FROM Person WHERE FirstName=? AND SecondName=?", new String[] { Ali, Ilyes });**

Les méthodes de la classe SQLiteDatabase

La méthode rawQuery

- Pour exécuter une requête SQL pour laquelle on souhaite de réponse , des requêtes de type *SELECT*.
- *rawQuery* retourne un objet de type **Cursor** qui permet de parcourir les **n-uplets** un à un :

```
Cursor cursor= bdd.rawQuery ("SELECT * FROM Person WHERE...");
```

Méthodes spécialisées

- On utilise notre objet SQLiteDatabase pour faire l'insert/ update/delete en lui passant, le nom de la table, l'objet contenant des paires (colonne, valeur).

- **int insert (String table-Name, null, ContentValues valeurs).** Méthode pratique pour insérer une ligne dans la base de données, le nom d'une colonne (souvent null), elle retourne l'identifiant du nouveau n-uplet.
- **int update (String table-Name, ContentValues valeurs, String whereClause, String[] Args)**
- **int delete (String table-Name, String whereClause, String[] Args)**
- *update* et *delete* retournent le nombre de lignes affectées i.e le nombre de *n-uplets* modifiées.

- **Les paramètres:**
  - **Table-Name** : le nom de la table
  - **valeurs** : On définit un objet ***ContentValues*** dans lequel on place des paires *(NomDeColonne, Valeur)* à créer.
  - **whereClause**: une condition contenant des jokers **?**
  - **whereArgs**: chaînes à mettre à la place des **?**

Exemple


```
ContentValues valeurs = new ContentValues();

valeurs.put (“FirstName", "Ahmed");
valeurs.put(“SecondName", "Mounir");

bdd.update ("Person", valeurs, "Id=?", new String[]
{"0001" });

bdd.delete ("Person", "weight
 BETWEEN ? AND ?",new String[]{"45", "53"});
```

Les méthodes de la classe Cursor

- Nous pouvons récupérer des données de a base en utilisant un objet de la classe Cursor. L'appel à la méthode ***rawQuery*** retournera un ensemble de results avec le **curseur** pointant vers la table.
- Nous pouvons avancer le curseur et récupérer les données.

- **Fonctions disponibles dans la classe Cursor:**

- **getColumnCount()** : retourne le nombre total de colonnes de la table.
- **getCount()** : retourne le nombre total de de lignes de la table.
- **getPosition()** : retourne la position actuelle du curseur dans la table
- **getInt(i)**, **getLong(i)**, **getFloat(i)**, **getString(i)**, ... : valeur de la colonne *i*.
- **moveToFirst()** : positionner le curseur sur le premier
- **moveToNext()** : passe à la ligne suivante.
- **isAfterLast()** : retourne vrai si le parcours est fini

La Classe SQLiteOpenHelper

- Pour gérer toutes les opérations liées à la base de données, une classe abstraite est fournit par Android « SQLiteOpenHelper ».
- Elle aide à maîtriser toutes les relations avec la base de données.
- Elle gère automatiquement la création et la mise à jour de la base de données. Elle possède deux méthodes à surcharger
- Sa syntaxe est donnée ci-dessous:

```
public class DBHelper extends SQLiteOpenHelper {
   public DBHelper(){
      super(context,DATABASE_NAME,null,1);
   }
   public void onCreate(SQLiteDatabase db) {}
   public void onUpgrade(SQLiteDatabase database, int oldVersion, int newVersion) {}
}
```

Les méthodes de la classe SQLiteOpenHelper

La méthode onCreate

- La méthode de *callback*
  void onCreate (SQLiteDatabase db)
  est automatiquement appelée au moment de la création de la base de données.
  Le paramètre **db** représente la base.

- Cette méthode est appelée quand la base de données n'existe pas encore. C'est dans cette méthode que vous devrez lancer les instructions pour créer les différentes tables et éventuellement les remplir avec des données initiales.

Les méthodes de la classe SQLiteOpenHelper

La méthode onUpgrade

- La méthode de mise à jour, qui est déclenchée à chaque fois que l'utilisateur met à jour son application.

Void onUpgrade (SQLiteDatabase db, int oldVersion, int newVersion)

- Cette méthode est appelée quand la version de l'application est supérieure à celle de la base.
- Son rôle est de mettre à jour les tables de la base de données.

- **Paramètres:**

●**db** : la base de données manipulée
●**oldVersion** : la version précédente de la base
●**newVersion** : la nouvelle version de la base

La méthode onUpgrade

- En général, le contenu de cette méthode est assez constant puisqu'on se contente de supprimer les tables déjà existantes pour les reconstruire suivant le nouveau schéma :

```
public static final String METIER_TABLE_DROP = "DROP TABLE IF EXISTS
" + METIER_TABLE_NAME + ";";

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

Les méthodes de la classe SQLiteOpenHelper

- Les méthodes

  - **getReadableDatabase**

  - **getWritableDatabase**

Ces Méthodes de **SQLiteOpenHelper** ouvrent la base et appellent automatiquement **onCreate** et **onUpgrade** si nécessaire.

Recommandation

- Il est recommandé de définir une classe associée à chaque table.
- Les instances de la classe sont les n-uplets de la table.
- Définir une classe qui regroupe toutes les requêtes SQL la concernant : création, suppression, mise à jour, parcours, insertions. . . sous forme de méthodes de classe.

CursorAdapter

- Un **CursorAdapter** est un Adaptateur qui expose les données d'un **Cursor** à un **ListView**.
- Android fournit des classes d'adaptateur pour afficher les données d'une requête de base de données SQLite.
- Cursor est une classe qui permet de parcourir le résultat d'une requête **SELECT**.

- Il suffit d'étendre la classe ***CursorAdapter*** et redéfinir les méthodes ***newView()*** et ***bindView()*.**

- Constructeur de la classe héritant de la classe ***CursorAdapter*** possède deux paramètres:
  - **context**: le contexte de l'application.
  - **cursor**: résultat d'une requête SELECT.

```
public MyCursorAdapter(Contextcontext, Cursor cursor)
  { super (context, cursor, 0);}
```

CursorAdapter

La méthode *newView()*

utilisée pour instancier une nouvelle vue et la renvoyer

```
@Override
public View newView (Context context, Cursor cursor,
ViewGroup parent) {
Return LayoutInflater.from (context).inflate
(R.layout.item_layout, parent, false); }
```

- Avec: item_layoutest le fichier décrivant les items de ListView (*res/layout/item_layout.xml*)

CursorAdapter

La méthode *bindView()*

Utilisée pour lier toutes les données à une vue donnée.

```
@Override
public void bindView(View view, Context context, Cursor cursor) {
// Les champs à remplir dans la vue de l'item
TextView edit_nom= (TextView) view.findViewById(R.id.editnom);
TextView edit_prenom= (TextView) view.findViewById(R.id.editprenom);

// Extraction des données du curseur
String str_nom= cursor.getString(cursor.getString(0));      // 0 est la position de colonne nom
String str_prenom= cursor.getString(cursor.getString(1)); // 1 est la position de colonne prenom

// Remplir les champs par les données extraites
edit_nom.setText(str_nom);
edit_prenom.setText(str_prenom);
}
```

[Ref]

CursorAdapter

Récupération du curseur

- **CursorAdapter** est utilisé après la récupération d'un **Cursor** représentant le résultat d'une requête **SQLite**

- Nous pouvons utiliser **SQLiteOpenHelper** qui donne accès à la base de données sous-jacente.

- Nous utilisons la méthode **rawQuery** qui renvoie un **Cursor**.

```
// DatabaseHelper est une sous classe de la classe SQLiteOpenHelper
DatabaseHelper  dbhelper =new DatabaseHelper(this);
SQLiteDatabase db= dbhelper.getReadableDatabase();
Cursor cursor= db.rawQuery ("SELECT * FROM Student", null);
```

CursorAdapter

Attacher l'adaptateur à un ListView:

- L'adaptateur est utilisé dans l'activité pour afficher un ensemble d'éléments dans un ListView:

```
// Trouver ListView à remplir (lv est l'identifiant de ListView)
ListView lv= (ListView) findViewById(R.id.lv);
// Setup cursor adapter using cursor from last step
MyCursorAdapter adapter= new MyCursorAdapter(this, cursor);
// Attach cursor adapter to the ListView
lv.setAdapter(adapter);
```