Développement sous Android

Le but de cette formation est de découvrir la programmation sous Android, sa plate-forme de développement et les spécificités du développement embarqué sur *smartphone, Tablette, TV*.

Les grandes notions abordées dans ce cours sont:

- Présenter le système Android ;
- Créer les interfaces d'une application ;
- Naviguer et faire communiquer des applications ;
- Manipuler des données (préférences, fichiers, ...) ;
- Manipuler Services, threads et programmation concurrente ;
- Utiliser les capteurs, le réseau.

Plan de la formation :

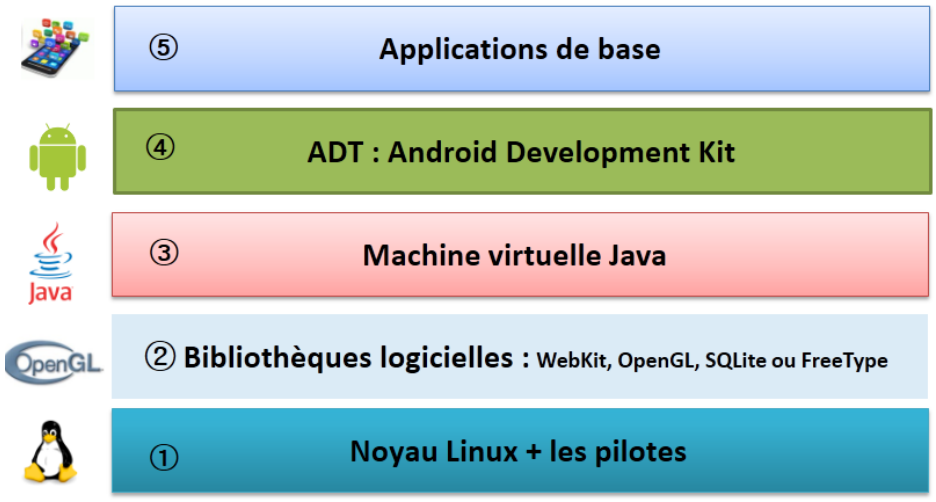
1. Le SDK Android
2. Interfaces graphiques (Layout)
3. Les styles
4. Material Designs
5. Les Intents
6. Persistance des données
7. Programmation concurrente
8. Connectivité
9. Développement client serveur
10. Divers
11. Workshop

1.Android

1.1. Définition :

Android est un système d'exploitation qui va donner vie aux appareils, il sait parler aux composants et vous pouvez également directement lui parler via une interface graphique. Ce "programme" va servir ainsi de liaison, le plus emblématique des OS reste Microsoft Windows installé sur plus de 90% du parc informatique. Android est quant à lui dédié aux appareils mobiles, smartphones et tablettes, mais aussi aux télévisions connectés avec la solution Google TV. Ces appareils mobiles ont nécessité de développer un OS qui leur est dédié, la raison est simple : l'expérience utilisateur d'un smartphone ou d'une tablette nécessite des interactions et un fonctionnement particulier.

1.2. Les couches Android :



1.3.Historique des versions

Le nombre de ***release*** est impressionnant [Version] :

Nom	***Version***	***Date***
Android	1.0	09/2008
Petit Four	1.1	02/2009
Cupcake	1.5	04/2009
Donut	1.6	09/2009
Gingerbread	2.3	12/2010
Honeycomb	3.0	02/2011

Ice Cream Sandwich	4.0.1	10/2011
Jelly Bean	4.1	07/2012
KitKat	4.4	10/2013
Lollipop	5.0	10/2014
	5.0.1	12/2014
	5.0.2	12/2014
	5.1	03/2015
	5.1.1	04/2015
Marshmallow	6	10/2015
	6.0.1	12/2015
Nougat	7.0	08/2016
	7.1	10/2016

1.4.SDK Android

L'écosystème d'Android s'appuie sur deux piliers :

- Le langage Java
- Le SDK qui permet d'avoir un environnement de développement facilitant la tâche du développeur.

Le kit de développement donne accès à des exemples, de la documentation mais surtout à l'API de programmation du système et à un émulateur pour tester ses applications.

Stratégiquement, Google utilise la licence Apache pour Android ce qui permet la redistribution du code sous forme libre ou non et d'en faire un usage commercial.

Le plugin **Android Development Tool** permet d'intégrer les fonctionnalités du SDK à Eclipse/ Android Studio. Il faut l'installer comme un plugin classique en précisant l'URL du plugin. Ensuite, il faut renseigner l'emplacement du SDK (préalablement téléchargé et décompressé) dans les préférences du plugin ADT.

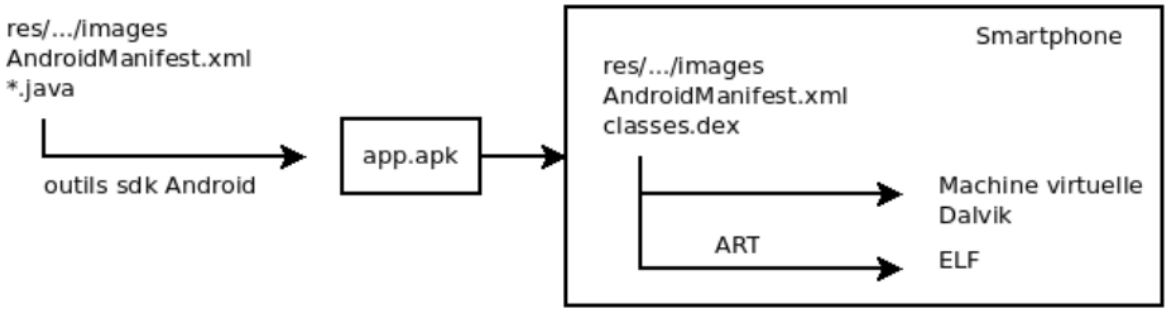
a. L'Operating System

Android est en fait un système de la famille des Linux, pour une fois sans les outils GNU. L'OS s'appuie sur :

- Un noyau Linux (et ses drivers)
- Une couche d'abstraction pour l'accès aux capteurs (HAL)
- Une machine virtuelle : Dalvik Virtual Machine (avant Lollipop)

- Un compilateur de bytecode vers le natif Android Runtime (pour Lollipop)
- Des applications (navigateur, gestion des contacts, application de téléphonie...)
- Des bibliothèques (SSL, SQLite, OpenGL ES, etc.…)
- Des API d'accès aux services Google

b. Anatomie d'un déploiement :



Dalvik et ART

[Dalvik]est le nom de la machine virtuelle open-source utilisée sur les systèmes Android. Cette machine virtuelle exécute des fichiers .dex, plus ramassés que les .class classiques. Ce format évite par exemple la duplication des String constantes. La machine virtuelle utilise elle-même moins d'espace mémoire et l'adressage des constantes se fait par un pointeur de 32 bits.

[Dalvik]n'est pas compatible avec une JVM du type Java SE ou même Java ME. La librairie d'accès est donc redéfinie entièrement par Google.

A partir de Lollipop, Android dispose d'ART qui compile l'application au moment du déploiement (Ahead-of-time compilation).

c. Le plugin de développement d'Eclipse: ADT

Un projet basé sur le plugin ADT est décomposé de la manière suivante :

- **src/**: les sources Java du projet
- **libs/**: bibliothèques tierces
- **res/**:
  - **res/drawable**: ressources images
  - **res/layout**: description des IHMs en XML
  - **res/values**: chaines de caractères et dimensions

- **gen/** : les ressources auto générées par ADT
- **assets/:** ressources brutes (raw bytes)
- **bin/**:
  - bin/classes: les classes compilées en .class
  - bin/classes.dex: exécutable pour la JVM Dalvik
  - bin/myapp.zip: les ressources de l'application
  - bin/myapp.apk: application empaquetée avec ses ressource et prête pour le déploiement.

d. Les éléments d'une application

Une application Android peut être composée des éléments suivants :

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

e. Le Manifest de l'application :

Le fichier **AndroidManifest.xml** déclare l'ensemble des éléments de l'application.

```xml
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.hassan.myapplication">
    <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>
    <service>...</service>
    <receiver>...</receiver>
    <provider>...</provider>
</manifest>
```

1.5.Les ressources

Les ressources de l'application sont utilisées dans le code au travers de la classe statique R. ADT régénère automatiquement la classe statique R à chaque changement dans le projet. Toutes les ressources sont accessibles au travers de R, dès qu'elles sont déclarées dans le fichier XML ou que le fichier associé est déposé dans le répertoire adéquat. Les ressources sont utilisées de la manière suivante :

```
android.R.type_ressource.nom_ressource
```

Qui est de type int. Il s'agit en fait de l'identifiant de la ressource. On peut alors utiliser cet identifiant ou récupérer l'instance de la ressource en utilisant la classe Ressources:

```
Resources res = getResources();
String hw = res.getString(R.string.hello);
XXX o = res.getXXX(id);
```

Une méthode spécifique pour les objets graphiques permet de les récupérer à partir de leur id, ce qui permet d'agir sur ces instances même si elles ont été créées via leur définition XML :

```
TextView texte = (TextView)findViewById(R.id.le_texte);
texte.setText("Here we go !");
```

Les chaines

Les chaines constantes de l'application sont situées dans res/values/strings.xml. L'externalisation des chaines permettra de réaliser l'internationalisation de l'application. Voici un exemple :

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="hello">Bonjour les dev IAM </string>
    <string name="app_name">Andro_TP0 </string>
</resources>
```

La récupération de la chaine se fait via le code :

```
Resources res = getResources();
String hw = res.getString(R.string.hello);
```

Internationalisation :

Le système de ressources permet de gérer très facilement l'internationalisation d'une application. Il suffit de créer des répertoires values-XX où XX est le code de la langue que l'on souhaite implanter. On place alors dans ce sous répertoire le fichier xml strings.xml contenant les chaines traduites associées aux même clefs que dans values/strings.xml. On obtient par exemple pour les langues es et fr l'arborescence :

```
MyProject/
  res/
     values/
          strings.xml
     values-es/
          strings.xml
     values-fr/
          strings.xml
```

Android chargera le fichier de ressources approprié en fonction de la langue du système.

Autres valeurs simples

Plusieurs fichiers xml peuvent être placés dans res/values. Cela permet de définit des chaines, des couleurs, des tableaux.

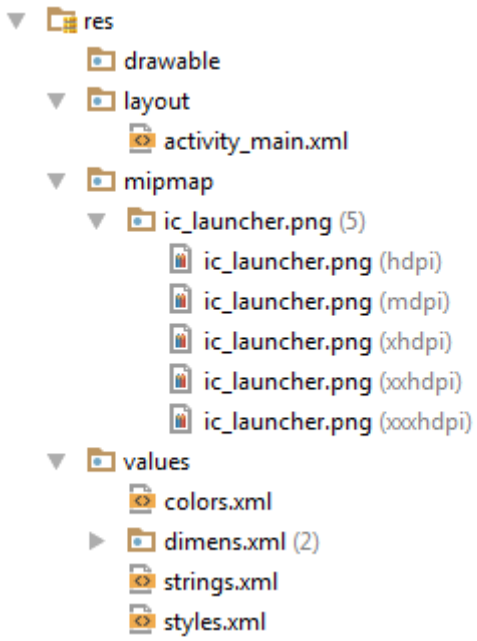
L'assistant de création permet de créer de nouveaux fichiers de ressources contenant des valeurs simples, comme par exemple un tableau de chaines :

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
<string-array name="test">
  <item>it1</item>
  <item>it2</item>
</string-array>
</resources>
```

Autres ressources

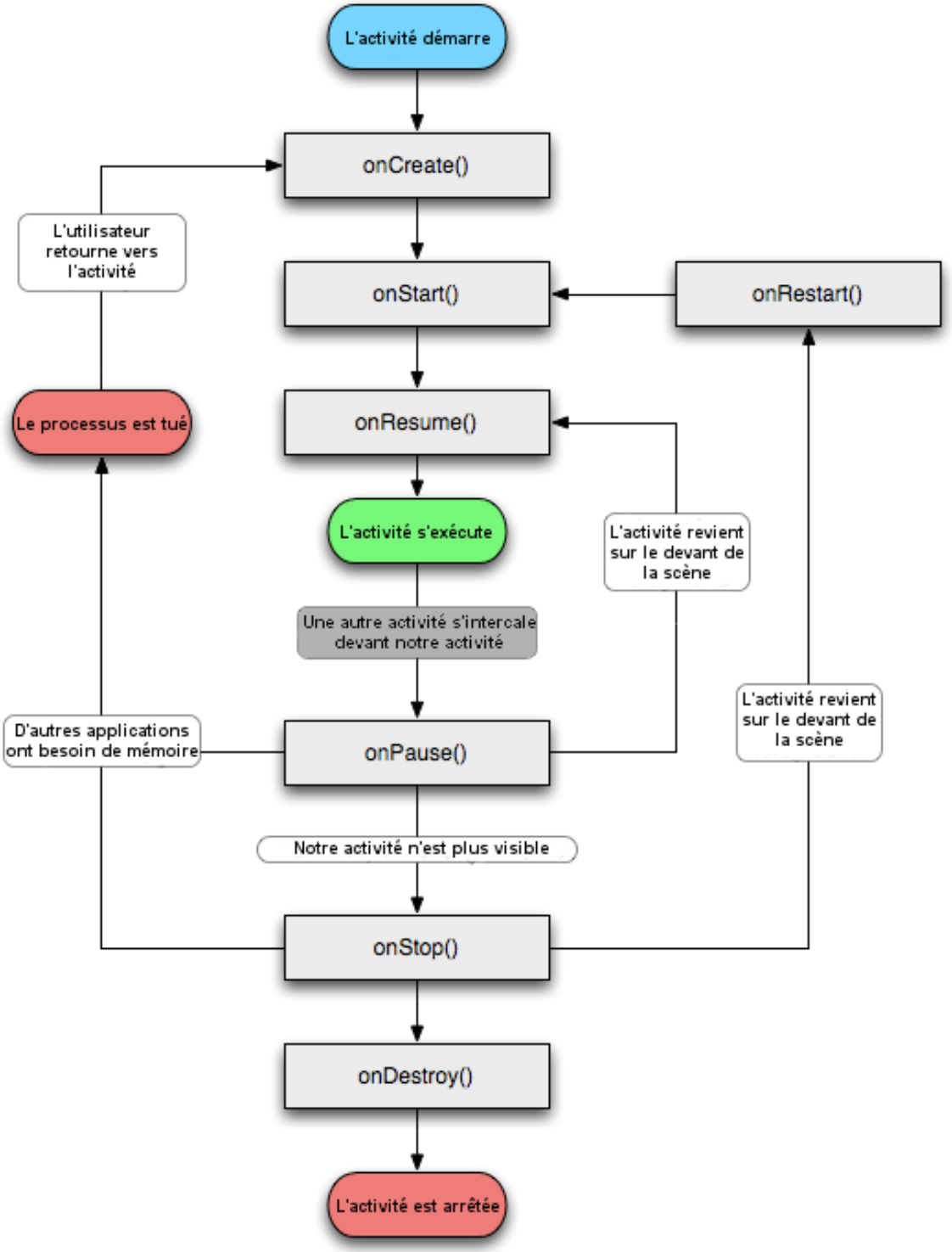
D'autres ressources sont spécifiables dans res :

- Les menus
- Les images (R.drawable)
- Des dimensions (R.dimen)
- Des couleurs (R.color)



1.6.Les activités :

Une application Android étant hébergée sur un système embarqué, le cycle de vie d'une application ressemble à celle d'une application Java ME. L'activité peut passer des états :



- **Démarrage -> actif** : détient le focus et est démarré
- **Actif -> suspendue** : ne détient plus le focus
- **Suspendue -> actif** :

- **Suspendue -> détruit** :

Le nombre de méthodes à surcharger et même plus important que ces états

```
public class Main extends Activity {
  public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.acceuil); }
  protected void onDestroy() {
    super.onDestroy(); }
  protected void onPause() {
    super.onPause(); }
  protected void onResume() {
    super.onResume(); }
  protected void onStart() {
    super.onStart(); }
  protected void onStop() {
    super.onStop(); } }
```

Cycle de vie d'une activité

onCreate() / onDestroy() : permet de gérer les opérations à faire avant l'affichage de l'activité, et lorsqu'on détruit complètement l'activité de la mémoire. On met en général peu de code dans onCreate() afin d'afficher l'activité le plus rapidement possible.

onStart() / onStop() : ces méthodes sont appelées quand l'activité devient visible/invisible pour l'utilisateur.
onPause() / onResume(): une activité peut rester visible mais être mise en pause par le fait qu'une autre activité est en train de démarrer, par exemple B. onPause() ne doit pas être trop long, car B ne sera pas créé tant que onPause() n'a pas fini son exécution.

onRestart() : cette méthode supplémentaire est appelée quand on relance une activité qui est passée par onStrop(). Puis onStart() est aussi appelée. Cela permet de différencier le premier lancement d'un relancement.
Le cycle de vie des applications est très bien décrit sur la page qui concerne les Activity.

Sauvegarde des interfaces d'activité

L'objet Bundle passé en paramètre de la méthode onCreate permet de restaurer les valeurs des interfaces d'une activité qui a été déchargée de la mémoire. En effet, lorsque l'on appuie par exemple sur la touche Home, en revenant sur le bureau, Android peut être amené à déchargé les

éléments graphiques de la mémoire pour gagner des ressources. Si l'on rebascule sur l'application (appui long sur Home), l'application peut avoir perdu les valeurs saisies dans les zones de texte.

Pour forcer Android à décharger les valeurs, il est possible d'aller dans "Development tools > Development Settings" et de cocher "Immediately destroy activities". Si une zone de texte n'a pas d'identifiant, Android ne pourra pas la sauver et elle ne pourra pas être restaurée à partir de l'objet Bundle.

Si l'application est complètement détruite (tuée), rien n'est restauré. Le code suivant permet de visualiser le déclenchement des sauvegardes :

```
protected void onSaveInstanceState(Bundle outState) {
      super.onSaveInstanceState(outState);
      Toast.makeText(this, "Sauvegarde !", Toast.LENGTH_LONG).show();
}
```

Description des ressources

- Ressource : élément statique réutilisable
- Ressources définies dans des fichiers XML (ou binaires) dans res/ (noms de fichiers [a-z0-9_]+) :

  - ✓ res/values : déclaration XML (avec i18n) de string, color, dimen, style...
    res/drawable : fichiers multimédias binaires (images, sons)
  - ✓ res/layout : définition XML d'agencements de vues graphiques (sous forme d'arbre)
  - ✓ res/anim : fichiers XML de définition d'animations

    - Animation d'interpolation (changement de transparence, échelle, angle de rotation...)
    - Animation pour séquences d'images : liste des images avec leur durée d'affichage

  - ✓ res/xml : pour des fichiers XML divers
  - ✓ res/raw : pour d'autres ressources sous la forme de fichiers binaires

Référence aux ressources

- Référencement de chaque ressource par une constante entière dans des classes internes de R.java
- Utilisation de Resources Context.getResources() (Activity hérite de Context) pour obtenir un getter de ressources
- Quelques exemples :
    - getString(R.string.hello_world)
    - getStringArray(R.stringarray.messages)
    - getColor(R.color.my_nice_color)
    - getLayout(R.layout.activity_layout)
    - getDimension(R.dimen.world_width)
    - getDrawable(R.drawable.card_picture)
    - getIntArray(R.intArray.coordinates)
    - openRawResource(R.raw.bindata) (retourne un InputStream)
- Il existe des ressources système en utilisant le paquetage android (par exemple : @android:color/blue)
- Accès direct à une vue depuis une activité : View Context.findViewById(int)
- Il n'est pas possible simplement et proprement de lister dynamiquement toutes les ressources disponibles

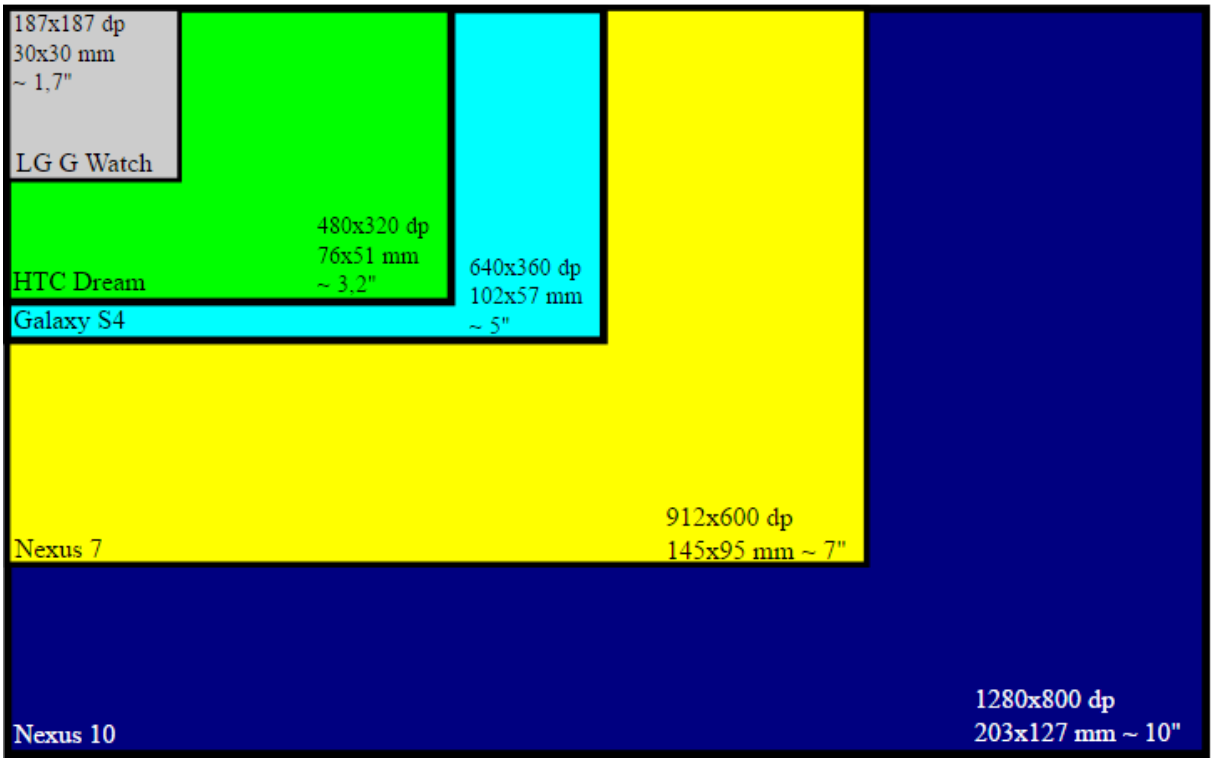
À propos des dimensions

Différents moyens d'indiquer une dimension pour les layouts :

- En nombre de pixels (100px)
- En nombre de pixels densité-indépendante (100dp). Un dp représente un pixel avec une résolution standardisée de 160dpi (160 pixels par pouce). Cela permet d'indiquer une dimension qui a toujours physiquement à peu près la même taille quel que soit l'écran (1dp ~ 159 µm).

- En nombre de pixels échelle-indépendante (100sp). Par défaut un sp correspond à un dp sauf si l'utilisateur a redéfini la taille des fontes utilisées dans les paramètres d'accessibilité. On utilisera donc les sp pour indiquer la taille d'une police.
- En millimètres (16mm)
- En pouces (0.63in). Un pouce (inch) mesure environ 25,4mm.
- En points. Un point représente 1/72 pouces, soit 353µm.

Quelques dimensions d'écrans d'appareils Android :



2. Layouts

L'API propose certains layouts permettant de créer un agencement de composants sous la forme de fichiers XML (ou programmatiquement).

Lorsque l'on place un composant graphique sur un layout, des paramètres de placements (LayoutParams) peuvent être utilisés (sous la forme d'attributs XML ou d'une instance de LayoutParams).

Paramètres d'agencement communs à tous les layouts :

- width et height pour indiquer les dimensions demandées pour le composant enfant
    - Spécifiés à l'aide de valeurs absolues
    - Ou alors avec des constantes spéciales :
        - MATCH_PARENT si la dimension doit occuper tout l'espace fourni par le parent
        - WRAP_CONTENT si la dimension doit être réduite au minimum nécessaire pour afficher le contenu du composant (par exemple le texte d'un bouton ou d'un TextView)

Approches possibles pour mettre en oeuvre un agencement complexe :

- Créer programmatiquement une nouvelle classe héritée de ViewGroup avec la redéfinition de onMeasure et onLayout
- Imbriquer différents ViewGroup déjà fournis par l'API

2.1. Les types de Layout en Android

N°	Mise en page et description
1	**LinearLayout** LinearLayout est un groupe de vues qui aligne tous les enfants dans une seule direction, verticalement ou horizontalement.
2	**RelativeLayout** RelativeLayout est un groupe de vue qui affiche des vues de l'enfant dans des positions relatives.
3	**TableLayout** TableLayout est une vue que les groupes de vues en lignes et en colonnes.

5	**FrameLayout** Le FrameLayout est un espace réservé à l'écran que vous pouvez utiliser pour afficher une vue unique.
6	**ListView** ListView est un groupe de vue qui affiche une liste d'éléments défilants.
7	**GridView** GridView est un ViewGroup qui affiche des objets dans une grille de défilement à deux dimensions.

2.2. Les Attributs Layout

Sr.No	Attribut & Description
1	**android: id** Ceci est l'ID qui identifie de manière unique la vue.
2	**android: layout_width** Ceci est la largeur de la mise en page.
3	**android: layout_height** Ceci est la hauteur de l'agencement
4	**android: layout_marginTop** C'est l'espace supplémentaire sur le côté supérieur de la mise en page.

5	**android: layout_marginBottom** C'est l'espace supplémentaire sur le côté inférieur de la mise en page.
6	**android: layout_marginLeft** C'est l'espace supplémentaire sur le côté gauche de la mise en page.
7	**android: layout_marginRight** Ceci est l'espace supplémentaire sur le côté droit de la mise en page.
8	**android: layout_gravity** Cela indique comment les enfants Vues sont positionnés.
9	**android: layout_weight** Ceci indique à quel point de l'espace supplémentaire dans la mise en page doit être alloué à la vue.
10	**android: layout_x** Ceci spécifie la coordonnée x de la mise en page.
11	**android: layout_y** Ceci spécifie la coordonnée y de la mise en page.
12	**android: layout_width** Ceci est la largeur de la mise en page.
13	**android: layout_width** Ceci est la largeur de la mise en page.

14	**android: paddingLeft** Ceci est la marge intérieur à gauche rempli pour la mise en page.
15	**android: paddingRight** Ceci est la marge intérieur à droite rempli pour la mise en page.
16	**android: paddingTop** Ceci est la marge intérieur en haut rempli pour la mise en page.
17	**android: paddingBottom** Ceci est la marge intérieur en basrempli pour la mise en page.

```
android: layout_width = wrap_content
android: layout_width = fill_parent
android: layout_width = Une taille fixe
```

a. Créer un objet :

```
android:id="@+id/moncontrole"
```

b. Récupération en Java

```
Controlemoncontrole =(Controle) findViewById(R.id.my_moncontrole);
```

2.3. Android UI Controls

Il y a nombre de contrôles d'interface utilisateur fournis par Android qui vous permettent de construire l'interface utilisateur graphique pour votre application.

N°	UI Control & description

1	**TextView** Cette commande est utilisée pour afficher le texte à l'utilisateur.
2	**Éditer le texte** EditText est une sous-classe prédéfinie de TextView qui comprend de riches fonctionnalités d'édition.
3	**AutoCompleteTextView** Le AutoCompleteTextView est une vue qui est similaire à EditText, sauf qu'il présente une liste de suggestions d'achèvement automatiquement lorsque l'utilisateur tape.
4	**Bouton** Un button qui peut être pressé ou cliqué par l'utilisateur d'effectuer une action.
5	**ImageButton** Un ImageButton est un AbsoluteLayout qui vous permet de spécifier l'emplacement exact de ses enfants. Ceci montre un bouton avec une image (au lieu du texte) qui peut être pressé ou activé par l'utilisateur.
6	**CheckBox** Un interrupteur marche / arrêt qui peut être basculée par l'utilisateur. Vous devez utiliser case à cocher lors de la présentation des utilisateurs avec un groupe d'options sélectionnables qui ne sont pas mutuellement exclusifs.
7	**ToggleButton** Un bouton marche / arrêt avec un témoin lumineux.
8	**Bouton radio** Le RadioButton a deux états: soit cochée ou décochée.
9	**RadioGroup** Un RadioGroup est utilisé pour regrouper un ou plusieurs RadioButtons.

10	**ProgressBar** La vue ProgressBar fournit une rétroaction visuelle à propos de certaines tâches en cours, comme lorsque vous effectuez une tâche en arrière-plan.
11	**Spinner** Une liste déroulante qui permet aux utilisateurs de sélectionner une valeur à partir d'un ensemble.
12	**TimePicker** La vue TimePicker permet aux utilisateurs de sélectionner un moment de la journée, soit en mode 24 heures ou AM / PM.
13	**DatePicker** La vue DatePicker permet aux utilisateurs de sélectionner une date de la journée.

2.4. Créer des contrôles UI

Les contrôles d'entrée sont les composants interactifs dans l'interface utilisateur de votre application. Android fournit une grande variété de contrôles que vous pouvez utiliser dans votre interface utilisateur, tels que les boutons, les champs de texte, rechercher des bars, cochez la case, les boutons de zoom, boutons à bascule, et beaucoup plus.

Comme expliqué dans le chapitre précédent, un objet de vue peut avoir un ID unique attribué à ce qui permettra d'identifier la vue unique dans l'arbre. La syntaxe d'une ID, à l'intérieur d'une balise XML est -

```
android:id="@+id/text_id"
```

Pour créer une interface de contrôle / View / Widget vous devrez définir une vue / widget dans le fichier de mise en page et de lui attribuer un identifiant unique comme suit: -

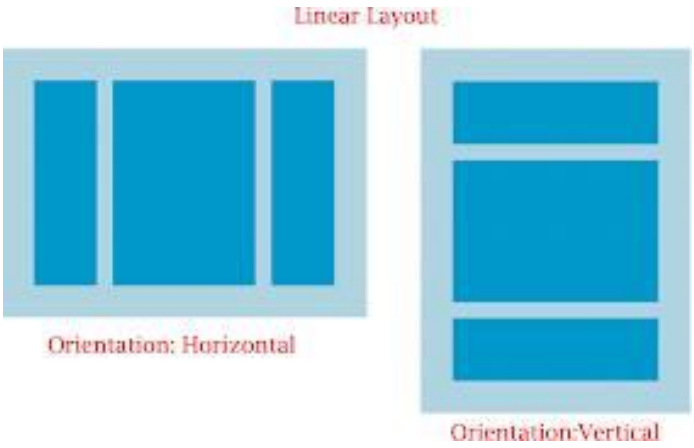
```
<?xml version="1.0" encoding="utf-8"?>
<typelayoutxmlns:android="http://schemas.android.com/apk/res/android"
```

```
android:layout_width="fill_parent"
android:layout_height="fill_parent"
>

<Controleandroid:id="@+id/text_id"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:text="Texte pour le teste "/>
</typelayout>
```

a. LinearLayout

- Conteneur dont les éléments enfants sont disposés, suivant la valeur de l'attribut orientation, soit :
- - Verticalement, les uns sous les autres, un seul élément par ligne.
- - Horizontalement, les uns après les autres, à la droite du précédent.



Exemple 1 :

Dans le fichier : **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="match_parent"
android:layout_height="match_parent"
android:orientation="horizontal" >
<Button
android:id="@+id/b1"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:text="Bouton #1" />

<Button
android:id="@+id/b2"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:text="Bouton #1" />
<Button
android:id="@+id/b3"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:text="Bouton #1"
android:layout_weight="1"/>
</LinearLayout>
```

Exemple 2 :

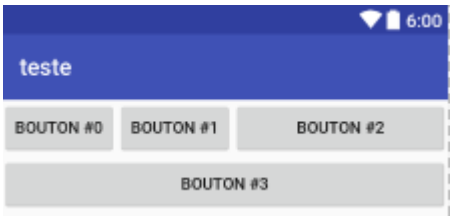
Dans le fichier : **res/layout/activity_main.xml**


```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
android:layout_width="match_parent"
android:layout_height="match_parent"
android:orientation="vertical" >
<Button
android:id="@+id/b0"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:text="Bouton #0" />
<Button
android:id="@+id/b1"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:layout_gravity="right"
android:text="Bouton #1" />
<Button
android:id="@+id/b2"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:layout_gravity="center"
android:text="Bouton #2" />
<Button
```

```
android:id="@+id/b3"
android:layout_width="match_parent"
android:layout_height="wrap_content"
android:text="Bouton #3" />
</LinearLayout>
```

Exemple 3 :

Dans le fichier : **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="match_parent"
android:layout_height="match_parent"
android:orientation="vertical" >
<LinearLayout
android:layout_width="match_parent"
android:layout_height="wrap_content">
<Button
android:id="@+id/b0"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:text="Bouton #0" />
<Button
android:id="@+id/b1"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:layout_gravity="right"
android:text="Bouton #1" />
<Button
android:id="@+id/b2"
android:layout_width="match_parent"
android:layout_height="wrap_content"
android:layout_gravity="center"
android:text="Bouton #2" />
</LinearLayout>
<Button
android:id="@+id/b3"
android:layout_width="match_parent"
android:layout_height="wrap_content"
android:text="Bouton #3" />
</LinearLayout>
```

TAF : Réaliser le layout suivant :

b. *RelativeLayout*

Conteneur dont les positions des vues enfants sont précisées par rapport à la vue parente ou par rapport aux autres vues enfants.

Le principe du **RelativeLayout** est de placer les éléments selon d' autreséléments du conteneur.

Voici les différents moyens qui sont à votre disposition pour le placement des éléments dans le cas d'un **RelativeLayout** :

Positionnement relatif au conteneur

Dans cette relation, vous pouvez lier un élément à son conteneur :

- android:layout_alignParentTop (true / false) : Cette option permet de préciser si le haut de l'élément doit être aligné avec celui de son conteneur.
- Même principe pour :
  - android:layout_alignParentBottom
  - android:layout_alignParentLeft
  - android:layout_alignParentRight.
- android:layout_centerHorizontal : Indique si l'élément doit être centré horizontalement dans son conteneur.
- Même principe pour : android:layout_centerVertical.

- android:layout_centerInParent : Vous permet d'indiquer que l'élément doit être centré horizontalement et verticalement dans le conteneur.

Position relative aux autres éléments

Afin de pouvoir référencer le positionnement d'un élément par rapport à un autre, vous disposez d'un moyen simple et efficace, il s'agit des identificateurs (ID).

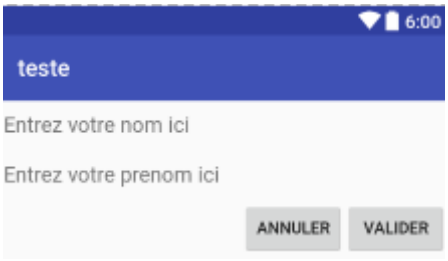
Donc voilà comment vous pouvez utiliser un ID :
A la déclaration d'un élément : android:id= "@+id/idElem"
A l'utilisation : @id/idElem

Maintenant que les bases sont posées, voici les différentes options disponibles :

- ✓ **android:layout_above** : Indique que l'élément sera placé au-dessus de celui indiqué par son id.
- ✓ **android:layout_below** : Indique que l'élément sera placé en dessous de celui indiqué par son id.
- ✓ **android:layout_toLeftOf** : Indique que l'élément sera placé à gauche de celui indiqué par son id.
- ✓ **android:layout_toRightOf** : Indique que l'élément sera placé à droite de celui indiqué par son id.
- ✓ **android:layout_alignTop** : Indique que le haut de notre élément est aligné avec le haut de l'élément indiqué.
- ✓ **android:layout_alignBottom** : Indique que le bas de notre élément est aligné avec le bas de l'élément indiqué.
- ✓ **android:layout_alignLeft** : Indique que le côté gauche de notre élément est aligné avec le côté gauche de l'élément indiqué.
- ✓ **android:layout_alignRight** : Indique que le côté droit de notre élément est aligné avec le côté droit de l'élément indiqué.
- ✓ android:layout_alignBaseLine : Indique que les lignes de base des 2 éléments sont alignés.

Exemple 1 :

Dans le fichier : **res/layout/activity_main.xml**

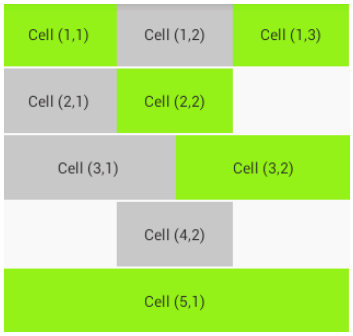


```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
android:orientation="vertical"
android:layout_width="match_parent"
android:layout_height="match_parent"
>
<EditText android:id="@+id/nomEdit"
android:hint="Entrez votre nom ici"
android:layout_width="match_parent"
android:layout_height="wrap_content"
android:layout_alignParentRight="true"
android:layout_alignParentTop="true" />
<EditText android:id="@+id/prenomEdit"
android:hint="Entrez votre prenom ici"
android:layout_width="match_parent"
android:layout_height="wrap_content"
android:layout_alignParentRight="true"
android:layout_below="@id/nomEdit" />
<Button android:id="@+id/valider"
android:text="valider"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:layout_below="@id/prenomEdit"
android:layout_alignRight="@id/prenomEdit" />
<Button android:id="@+id/annuler"
android:text="annuler"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:layout_alignBaseline="@+id/valider"
android:layout_alignBottom="@+id/valider"
android:layout_toStartOf="@+id/valider" />
</RelativeLayout>
```

TAF : Réaliser le layout suivant :

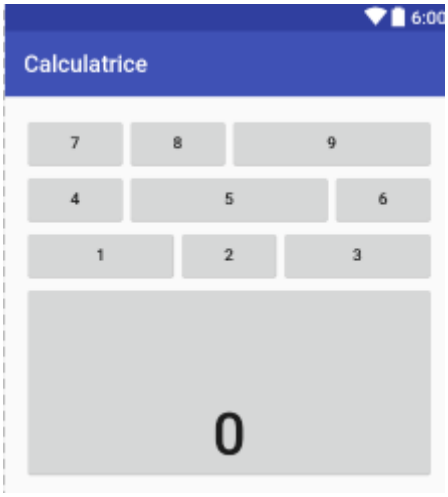
c. TableLayout

Conteneur dont les éléments enfants sont disposés sous forme de tableau. Les vues enfants sont des objets TableRow définissant chacun une ligne du tableau. Les éléments enfants des objets TableRow sont les cellules du tableau. À noter : ces tableaux ne disposent pas, à proprement parler, de colonnes, comme il en existe en HTML.



Exemple 1 :

Dans le fichier : **res/layout/activity_main.xml**




```
<?xml version="1.0" encoding="utf-8"?>
<TableLayout xmlns:android="http://schemas.android.com/apk/res/android"
xmlns:tools="http://schemas.android.com/tools"
android:layout_width="match_parent"
android:layout_height="match_parent"
android:paddingLeft="@dimen/activity_horizontal_margin"
android:paddingRight="@dimen/activity_horizontal_margin"
android:paddingTop="@dimen/activity_vertical_margin"
android:paddingBottom="@dimen/activity_vertical_margin">

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

<Button
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:text="7"
android:id="@+id/button7"
android:layout_column="0" />

<Button
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:text="8"
android:id="@+id/button8"
```

```
android:layout_column="1" />

<Button
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:text="9"
android:id="@+id/button9"
android:layout_column="2"
android:layout_weight="1" />
</TableRow>

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

<Button
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:text="4"
android:id="@+id/button4"
android:layout_column="0"
android:layout_weight="0" />

<Button
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:text="5"
android:id="@+id/button5"
android:layout_column="1"
android:layout_weight="1" />

<Button
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:text="6"
android:id="@+id/button6"
android:layout_column="2"
android:layout_weight="0" />
</TableRow>

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

<Button
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:text="1"
android:id="@+id/button1"
android:layout_column="0"
android:layout_weight="1" />

<Button
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:text="2"
android:id="@+id/button2"
android:layout_column="1"
android:layout_weight="0" />

<Button
```

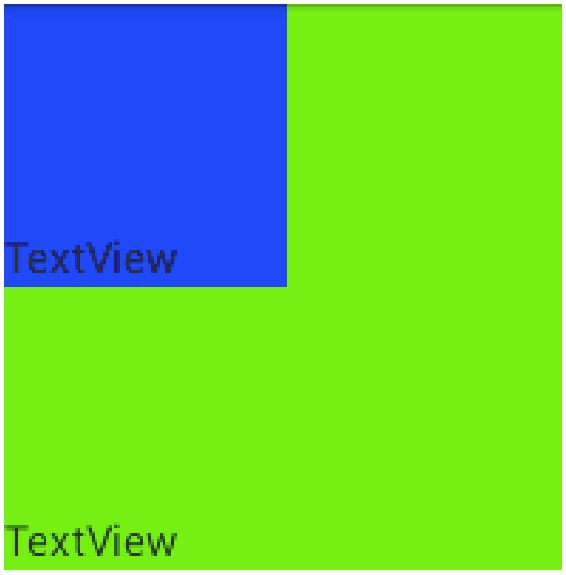
```
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:text="3"
android:id="@+id/button3"
android:layout_column="2"
android:layout_weight="1" />
</TableRow>

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

<Button
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:text="0"
android:id="@+id/button0"
android:layout_column="0"
android:layout_span="3"
android:layout_weight="1"
android:paddingTop="100dp"
android:focusableInTouchMode="false"
android:textSize="50sp" />
</TableRow>
</TableLayout>
```

d. FrameLayout

Conteneur réduit à sa plus simple expression. Tout ce qu'il contient sera dessiné à partir du coin en haut à gauche. Les derniers éléments enfants ajoutés seront dessinés par-dessus les plus anciens.



Exemple 1 :

Dans le fichier : **res/layout/activity_main.xml**


```
<?xml version="1.0" encoding="utf-8"?>
<FrameLayout xmlns:android="http://schemas.android.com/apk/res/android"
xmlns:tools="http://schemas.android.com/tools"
android:layout_width="match_parent"
android:layout_height="match_parent"
android:paddingRight="@dimen/activity_horizontal_margin"
android:paddingTop="@dimen/activity_vertical_margin"
android:paddingBottom="@dimen/activity_vertical_margin">

<ImageView
android:layout_width="match_parent"
android:layout_height="match_parent"
android:id="@+id/imageView"
android:layout_gravity="center"
android:src="@android:drawable/sym_action_call"/>
```

```
<Button
android:layout_width="wrap_content"
android:layout_height="60dp"
android:id="@+id/bt1"
android:layout_gravity="right|bottom"
android:paddingBottom="100dp" />

<Button
android:layout_width="match_parent"
android:layout_height="wrap_content"
android:id="@+id/imageView2"
android:layout_gravity="right|top" />

</FrameLayout>
```

Exemple 2 :

Dans le fichier : **res/layout/activity_main.xml**


```
<FrameLayout xmlns:android="http://schemas.android.com/apk/res/android"
android:id="@+id/frameLayout"
android:layout_width="fill_parent"
android:layout_height="fill_parent">
<ImageView
android:layout_width="match_parent"
android:layout_height="match_parent"
android:scaleType="fitXY"
android:src="@drawable/face" />
<TextView
android:layout_width="fill_parent"
android:layout_height="fill_parent"
android:gravity="center"
android:text="Formation Android"
android:textSize="30sp"
android:textColor="#000000"
android:textStyle="bold" />
</FrameLayout>
```

TAF : Réaliser le layout suivant :

e. GridView

Le GridView est un layout à deux dimensions. Pour remplir le GridView ont utilisé un adapter, qui est le lien entre la source de donnée et l'interface graphique.

Exemple 1 :

Dans le fichier : **res/layout/activity_main.xml**


```
<GridLayout xmlns:android="http://schemas.android.com/apk/res/android"
xmlns:tools="http://schemas.android.com/tools"
```

```
android:id="@+id/GridLayout1"
android:layout_width="match_parent"
android:layout_height="match_parent"
android:columnCount="3"
android:rowCount="3"
android:orientation="horizontal">
<Button
android:id="@+id/b1"
android:text="X" />
<Button
android:id="@+id/b2"
android:text="O" />
<Button
android:id="@+id/b3"
android:text="X" />
<Button
android:id="@+id/b4"
android:text="X" />
<Button
android:id="@+id/b5"
android:text="O" />
<Button
android:id="@+id/b6"
android:text="X" />
<Button
android:id="@+id/b7"
android:text="X" />
<Button
android:id="@+id/b8"
android:text="O" />
<Button
android:id="@+id/b9"
android:text="O" />

</GridLayout>
```

Exemple 2 :

Dans le fichier : **res/layout/activity_main.xml**

```
<GridLayout xmlns:android="http://schemas.android.com/apk/res/android"
xmlns:tools="http://schemas.android.com/tools"
android:id="@+id/GridLayout1"
android:layout_width="match_parent"
android:layout_height="match_parent"
android:columnCount="3"
android:rowCount="3"
android:orientation="horizontal">
<ImageButton
android:id="@+id/im1"
android:src="@drawable/face"
/>
<ImageButton
```

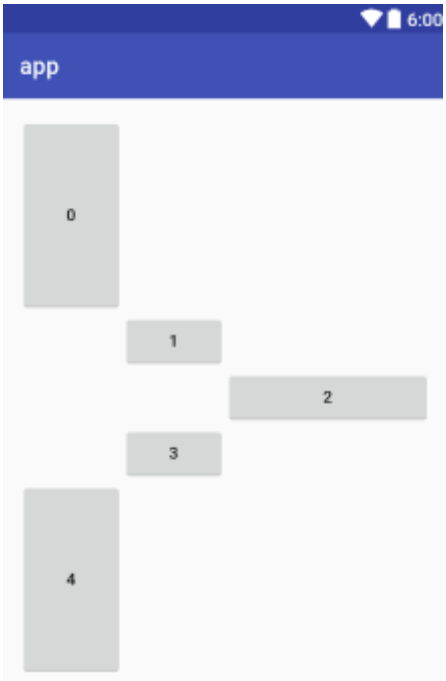
```
android:id="@+id/im2"
android:src="@drawable/linkedin"
/>
<ImageButton
android:id="@+id/im3"
android:src="@drawable/yahoo"
/>
<ImageButton
android:id="@+id/im4"
android:src="@drawable/twitter"
/>
<ImageButton
android:id="@+id/im5"
android:src="@drawable/skype"
/>

<ImageButton
android:id="@+id/im6"
android:src="@drawable/google"
/>

</GridLayout>
```

Exemple 3 :

Dans le fichier : **res/layout/activity_main.xml**



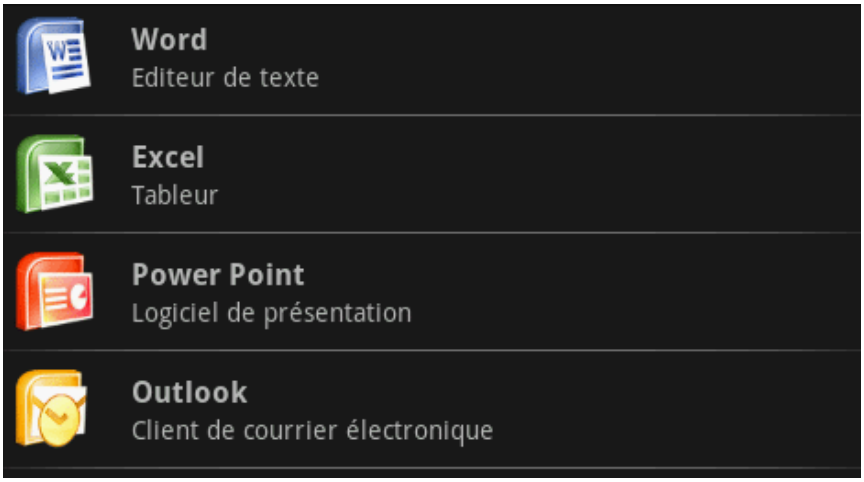

```
<?xml version="1.0" encoding="utf-8"?>
<GridLayout xmlns:android="http://schemas.android.com/apk/res/android"
xmlns:tools="http://schemas.android.com/tools"
android:layout_width="match_parent"
android:layout_height="match_parent"
android:paddingLeft="@dimen/activity_horizontal_margin"
android:paddingRight="@dimen/activity_horizontal_margin"
android:paddingTop="@dimen/activity_vertical_margin"
android:paddingBottom="@dimen/activity_vertical_margin"
android:columnCount="3"
```

```
android:rowCount="5">
<Button
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:text="0"
android:id="@+id/b0"
android:layout_row="0"
android:layout_column="0"
android:layout_rowWeight="1"/>
<Button
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:text="1"
android:id="@+id/b1"
android:layout_row="1"
android:layout_column="1" />

<Button
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:text="2"
android:id="@+id/b2"
android:layout_row="2"
android:layout_column="2"
android:layout_columnWeight="1"/>
<Button
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:text="3"
android:id="@+id/b3"
android:layout_row="3"
android:layout_column="1" />
<Button
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:text="4"
android:id="@+id/b4"
android:layout_row="4"
android:layout_column="0"
android:layout_rowWeight="1"/>
</GridLayout>
```

f. Listview

Les ListView sont très utiles pour afficher clairement un grand nombre de données se forme des linges d'une vue.



1.Dans le fichier layout, ajouter le fichier :affichageitem

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
android:orientation="horizontal"
android:layout_width="fill_parent"
android:layout_height="wrap_content"
>
<ImageView
android:id="@+id/img"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:layout_gravity="center_vertical"
android:padding="10px"
/>
<LinearLayout
android:orientation="vertical"
android:layout_width="match_parent"
android:layout_height="wrap_content"
android:layout_gravity="center_vertical"
android:paddingLeft="10px"
android:layout_weight="1"
>

<TextView android:id="@+id/titre"
android:layout_width="match_parent"
android:layout_height="match_parent"
android:textSize="16px"
android:textStyle="bold"
/>

<TextView android:id="@+id/description"
android:layout_width="match_parent"
android:layout_height="match_parent"
/>
</LinearLayout>
</LinearLayout>
```

2.Dans le fichier layout, ajouter le fichier :

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

```
android:layout_width="match_parent"
android:layout_height="match_parent"
>

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

</LinearLayout>
```

```
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;

import java.util.ArrayList;
import java.util.HashMap;

import android.view.View;
import android.widget.AdapterView;
import android.widget.ListView;
import android.widget.SimpleAdapter;
import android.widget.AdapterView.OnItemClickListener;


public class MainActivity extends AppCompatActivity {
private ListView L;
@Override
protected void onCreate(Bundle savedInstanceState) {
super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
L = (ListView) findViewById(R.id.liste);

        ArrayList<HashMap<String, String>> Element = new
ArrayList<HashMap<String, String>>();
        HashMap<String, String> map;
         map = new HashMap<String, String>();
         map.put("titre", "Word");
         map.put("description", "Editeur de texte");
         map.put("img", String.valueOf(R.drawable.word));
        Element.add(map);

        map = new HashMap<String, String>();
        map.put("titre", "Excel");
        map.put("description", "Tableur");
        map.put("img", String.valueOf(R.drawable.excel));
        Element.add(map);

        map = new HashMap<String, String>();
        map.put("titre", "Power Point");
        map.put("description", "Logiciel de présentation");
        map.put("img", String.valueOf(R.drawable.power));
        Element.add(map);

        map = new HashMap<String, String>();
        map.put("titre", "Outlook");
        map.put("description", "Client de courrier électronique");
```

```
        map.put("img", String.valueOf(R.drawable.outlook));
        Element.add(map);

//Création d'un SimpleAdapter qui se chargera de mettre les items présent
dans notre list (listItem) dans la vue affichageitem
SimpleAdapter Adp = new SimpleAdapter (this.getBaseContext(), Element,
R.layout.affichageitem,
new String[] {"img", "titre", "description"}, new int[] {R.id.img,
R.id.titre, R.id.description});

//On attribut à notre listView l'adapter que l'on vient de créer
L.setAdapter(Adp);

    }
}
```

3.Styles

3.1. Créer des fiches sources soit dans layout ou drawable

```
<?xml version="1.0" encoding="utf-8"?>
<selector xmlns:android="http://schemas.android.com/apk/res/android">
<item>
<shape android:shape="rectangle"><!-- type de bordure-->
<solid android:color="couleur "/><!-- en cas d'une seule couleur dans le fond-->
<gradient android:startColor="couleur1 " android:endColor="couleur2 " android:angle="angle"
/>
<!-- en cas d'un dégradé dans le fond==> gradient -->
<corners android:radius="valeur" /><!-- arrondir les coins -->
<stroke android:width="epaisseur" android:color="couleur" /><!-- cadre -->
</shape>
</item>
</selector>
```

3.2. Exemples

a- EditText

```
<?xml version="1.0" encoding="utf-8"?>
<selector xmlns:android="http://schemas.android.com/apk/res/android">
<item>
<shape android:shape="rectangle">
<solid android:color="#ffffff"/>
<corners android:radius="10dp" />
<stroke android:width="2dp" android:color="#3bbdfa" />
</shape>
</item>
</selector>
```

b- Button

```
<?xml version="1.0" encoding="utf-8"?>
<selector xmlns:android="http://schemas.android.com/apk/res/android">
```

```
<item android:state_pressed="true" >
<shape>
<gradient android:startColor="@color/yellow1"
android:endColor="@color/yellow2" android:angle="270" />
<stroke android:width="1dp" android:color="@color/grey05" />
<corners android:radius="20dp" />
</shape>
</item>
<item android:state_focused="true" >
<shape>
<gradient android:endColor="@color/orange4"
android:startColor="@color/orange5" android:angle="270" />
<stroke android:width="1dp" android:color="@color/grey05" />
<corners android:radius="20dp" />
</shape>
</item>
<item>
<shape>
<gradient android:endColor="@color/white1"
android:startColor="@color/white2" android:angle="270" />
<stroke android:width="1dp" android:color="@color/grey05" />
<corners android:radius="20dp" />
</shape>
</item>
</selector>
```

4.Android Toast

4.1. Toast class

Constante	Description
public static final int LENGTH_LONG	Affiche voir pour la longue durée de temps.
public static final int LENGTH_SHORT	Affiche voir pour la courte durée de temps.

4.2. Méthodes de classe Toast

Les méthodes couramment utilisées de la classe Toast sont donnés ci-dessous.

Méthode	Description
public static Toast makeText(Context context, CharSequence text, int duration)	Rend le pain contenant du texte et de la durée.
public void show()	Affiche un toast.

public void setMargin (float horizontalMargin, float verticalMargin)	Modifie la différence de marge horizontale et verticale.

La méthode getApplicationContext () renvoie l'instance de contexte.

Syntaxe :

```
Toast.makeText(this, 'Message ",Toast.LENGTH_SHORT).show();
```

5.Notification :

Exemple :

```
private void SendNotification() {
    NotificationCompat.Builder b = new NotificationCompat.Builder(this);
    b.setAutoCancel(true)
            .setDefaults(NotificationCompat.DEFAULT_ALL)
            .setWhen(System.currentTimeMillis())
            .setSmallIcon(R.drawable.web)
            .setTicker("Formation Android")
            .setContentTitle("Notification")
            .setContentText("bonjour mes collegues je suis heureux d'etre
avec vous :).")
            .setContentInfo("INFO");
    NotificationManager nm = (NotificationManager)
this.getSystemService(Context.NOTIFICATION_SERVICE);
    nm.notify(1, b.build());
}
```

6.AlertDialog

```
AlertDialog.Builder adb = new AlertDialog.Builder(this);
adb.setMessage("Message");
adb.setPositiveButton("Oui",
new DialogInterface.OnClickListener() {
@Override
public void onClick(DialogInterface arg0, int arg1) {
                        Toast.makeText(MainActivity.this,"clic sur
yes",Toast.LENGTH_LONG).show();
                    }
                });
adb.setNeutralButton("Annuler",new DialogInterface.OnClickListener() {
@Override
public void onClick(DialogInterface dialog, int which) {
        Toast.makeText(MainActivity.this,"clic sur
yes",Toast.LENGTH_LONG).show();
        finish();
    }
});

adb.setNegativeButton("Non",new DialogInterface.OnClickListener() {
```

```
@Override
public void onClick(DialogInterface dialog, int which) {
        Toast.makeText(MainActivity.this,"clic sur
yes",Toast.LENGTH_LONG).show();
        finish();
    }
});

AlertDialog alertDialog = adb.create();
alertDialog.show();
```

7.Intent

Les Intents sont des objets permettant de faire passer des messages contenant de l'information entre composants principaux(Activity, Services,…). La notion d'Intent peut être vue comme une demande de démarrage d'un autre composant, d'une action à effectuer. La raison d'être des Intents provient du modèle de sécurité d'Android. Chaque application est en effet *sandboxée*. Cela veut dire qu'une application A ne peut accéder aux données d'une application B. Grâce aux Intents, les applications ont la possibilité de fournir leurs services ou données si elles le souhaitent.

7.1. Ajouter l'activité dans le fichier Manifest

```
<activity android:name=".classe2"  android:label="Nomactive" />
```

7.2. Création d'un objet intent sans paramètres

```
Intent intent = new Intent(this, classe2.class);
```

7.3. Démarrage

```
startActivity(intent);
```

7.4. Avec paramètres

```
Intent intent = newIntent(this, classe2.class);
      intent.putExtra("variable", valeur);
      startActivity(intent);
```

7.5. Récupération

```
String ch = getIntent().getExtras().get("variable ");
```

7.6. Exemple (Voir le ficheTP intent)

7.7. Remarque :

- Autorisation (dans le fiche manifest) :

<uses-permission android:name="android.permission.CALL_PHONE" />	Appels
<uses-permission android:name="android.permission.SEND_SMS" />	Envoi des messages
<uses-permission android:name="android.permission.RECEIVE_SMS" />	Réception des messages
<uses-permission android:name="android.permission.CAMERA" />	Utilisation camera
<uses-permission android:name="android.permission.READ_CONTACTS"/>	Lire les contacts

- Envoyer un message

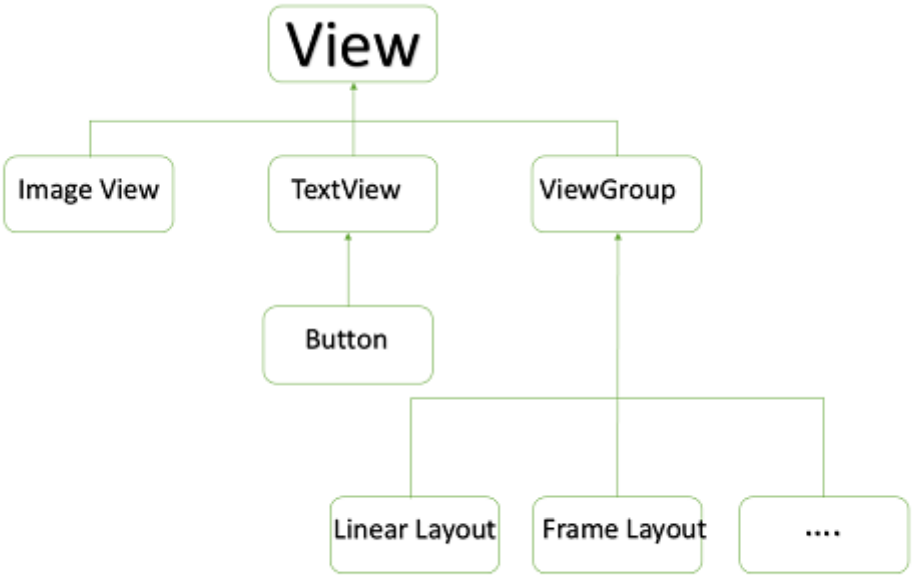
```
SmsManager sms=SmsManager.getDefault();
sms.sendTextMessage("0661848628", null, "OK",   null,null);
```

- Passer un appel :

```
Intent appel = new Intent(Intent.ACTION_CALL);
appel.setData(Uri.parse("tel:" + num));
if (ActivityCompat.checkSelfPermission(this,
    Manifest.permission.CALL_PHONE) !=
    PackageManager.PERMISSION_GRANTED) return;
    startActivity(appel);
```

- Lire la liste des contact (Voir le TP)

8.Android- Manipulation des événements



Il existe après trois concepts liés à Android Event Management -

- **Event Listeners** - Un écouteur d'événement est une interface dans la classe View qui contient une méthode de rappel unique. Ces méthodes seront appelées par le Framework Android lorsque la vue à laquelle l'auditeur a été enregistré est déclenchée par l'interaction de l'utilisateur avec l'élément dans l'interface utilisateur.
- **Event Listeners Registration** - Inscription à l'événement est le processus par lequel un gestionnaire d'événements s'inscrit un Listener événement afin que le gestionnaire est appelé lorsque le Listener événement déclenche l'événement.
- **Event Handlers**- Quand un événement se produit et nous avons enregistré un écouteur d'événement pour l'événement, l'écouteur d'événement appelle les gestionnaires d'événements, ce qui est la méthode qui gère en fait l'événement.

8.1. Les auditeurs d'événements et gestionnaires d'événements

Event Handler	Event Listener & Description

onClick()	**OnClickListener()** Ceci est appelé lorsque l'utilisateur soit des clics ou des touches ou se concentre sur aucun widget comme le bouton, texte, image, etc. Vous allez utiliser onClick () gestionnaire d'événements pour gérer cet événement.
onLongClick()	**OnLongClickListener()** Ceci est appelé lorsque l'utilisateur soit des clics ou des touches ou se concentre sur aucun widget comme le bouton, texte, image, etc. pour une ou plusieurs secondes. Vous allez utiliser onLongClick () gestionnaire d'événements pour gérer cet événement
onFocusChange()	**OnFocusChangeListener()** Ceci est appelé lorsque le widget perd son-à-dire de mise au point. L'utilisateur va loin du point de vue. Vous allez utiliser onFocusChange () gestionnaire d'événements pour gérer cet événement.
onKey()	**OnFocusChangeListener()** Ceci est appelé lorsque l'utilisateur se concentre sur l'élément et presses ou libère une clé matérielle sur le périphérique. Vous allez utiliser OnKey () gestionnaire d'événements pour gérer cet événement.
onTouch()	**OnTouchListener()** Ceci est appelé lorsque l'utilisateur appuie sur la touche, relâche la touche, ou tout geste de mouvement à l'écran. Vous allez utiliser OnTouch

	() gestionnaire d'événements pour gérer cet événement.
onMenuItemClick()	**OnMenuItemClickListener()** Ceci est appelé lorsque l'utilisateur sélectionne un élément de menu. Vous allez utiliser onMenuItemClick () gestionnaire d'événements pour gérer cet événement.
onCreateContextMenu()	**onCreateContextMenuItemListener()** Ceci est appelé lorsque le menu contextuel est en cours de construction (comme le résultat d'un soutenue "clic long)

8.2. Event Listeners Registration

Enregistrement de l'événement est le processus par lequel un gestionnaire d'événements s'inscrit un Listener événement afin que le gestionnaire est appelé lorsque le Listener événement déclenche l'événement. Bien qu'il existe plusieurs façons délicates pour enregistrer votre écouteur d'événement pour tout événement, mais je vais lister seulement top 3 façons, à partir de laquelle vous pouvez utiliser l'un d'eux en fonction de la situation.

- En utilisant une classe interne anonyme
- Classe d'activité implémente l'interface Listener.
- Utilisation de la mise en page fichier activity_main.xml pour spécifier directement gestionnaire d'événements.

Ci-dessous, la section vous fournira des exemples détaillés sur tous les trois scénarios -

- **isFocusable ()** - il retourne true ou false

- **isFocusableInTouchMode ()** - vérifie si la vue est focalisable en mode tactile.

8.3. Syntaxes & Exemples :

9. Le système de message Android

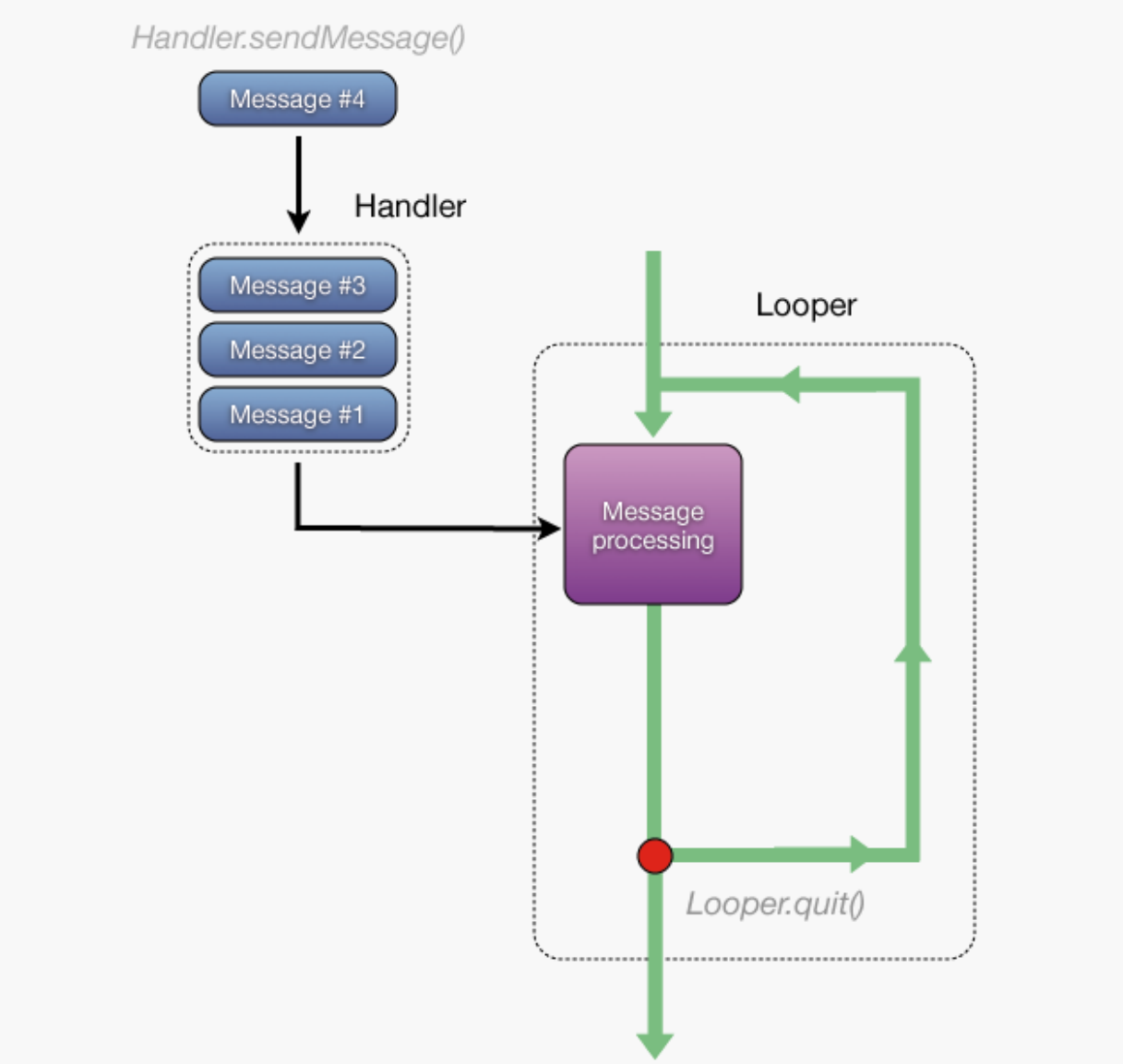
Android facilite énormément la programmation inter-*thread* grâce à un ingénieux système de messages. Etant particulièrement attaché à cette notion j'ai souhaité la présenter comme première alternative à la programmation concurrente sous Android (pour ne rien vous cacher, c'est à mon sens la meilleure technique pour résoudre ce genre de problématiques sous Android).

9.1. Définitions

- Thread : Souvent appelés processus légers, les threads peuvent être considéré comme des tâches dans lesquels s'exécutent des instructions. Dès lors qu'une application Android démarre, un thread est créé (le main thread) et sert de réceptacle pour les instructions du système. Pour donner une illusion de parallélisme, il est possible de créer plusieurs threads dans une application. Chacun des threads exécute de façon alternative (mais rapide d'où l'impression de parallélisme ou de simultanéité) des instructions. Le schéma ci-dessous montre une application disposant de 3 threads :

- Looper : La classe Looper permet de préparer un Thread à la lecture répétitive d'actions. Un tel Thread, présenté dans la figure ci-dessous, est souvent appelé looper thread. Sous Android, le main thread est en réalité un looper thread. Un Looper étant propre à un

  unique Thread.



- Message : Un Message représente une ou un ensemble de commandes à exécuter. Dans la définition précédente, le Message fait donc office d'instructions.
- Handler : Cette classe vous permet d'interagir avec les looper threads. C'est par le biais d'un Handler qu'il vous sera possible de poster des Messages ou des Runnables dans le Looper qui seront exécutés (au plus vite, après un temps donné ou à un moment donné) par le thread looper pointé par le Handler.

Exemple : (TP Messages)

```
final Toast msg=  Toast.makeText(getApplicationContext()," Message durée
limitée",Toast.LENGTH_LONG);
 msg.show();
Handler hd = new Handler();
hd.postDelayed(new Runnable() {
@Override
```

```
public void run() {
msg.cancel();
    }
} , 3000);
```