

Introduction au Développement d'Application Android

Amosse EDOUARD,
Doctorant

Séance 1

OBJECTIF DU COURS

- Comprendre l'architecture du système Android
- Comprendre l'organisation d'une application Android
- Développer et déployer des applications natives Android

Référence

- ▶ L'art du développement Android par **Mark L. Murphy**
- ▶ Professional NFC application développement for android by **Vedat Coskun**
- ▶ Android developer :
<http://developer.android.com/index.html>

Pré requis

- Programmation Orienté Objet
- JAVA
- XML

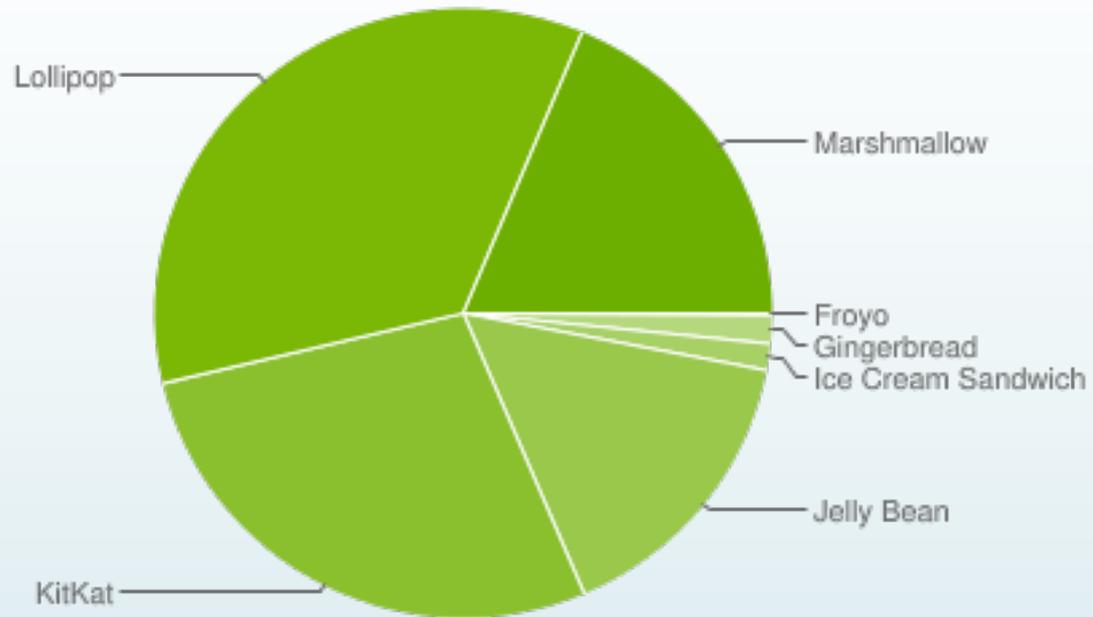
Enjeux du développement mobile

- CPU entre 500 et 600 MHz
- Mémoire allouée à une application est de l'ordre de quelques mégabytes
- Gestion du cycle de vie → une seule application peut être visible à la fois
- Densité multiples des écrans
 - Très faible résolution
 - Très haute définition
- Plusieurs orientations (Portrait, Paysage, Portrait inverse, ...)

Introduction / Définition

- ▶ Android est un système d'exploitation OPEN SOURCE pour terminaux mobiles (Smartphones, Tablet ,....
- ▶ Conçu à la base par une startup (Android) rachetée par Google en 2005
- ▶ Pour la promotion de ce système Google a fédéré autour de lui une trentaine de partenaires réunis au sein de l'Open Handset Alliance (OHA)
- ▶ C'est aujourd'hui le système d'exploitation mobile le plus utilisé à travers le monde

Versions de l'OS

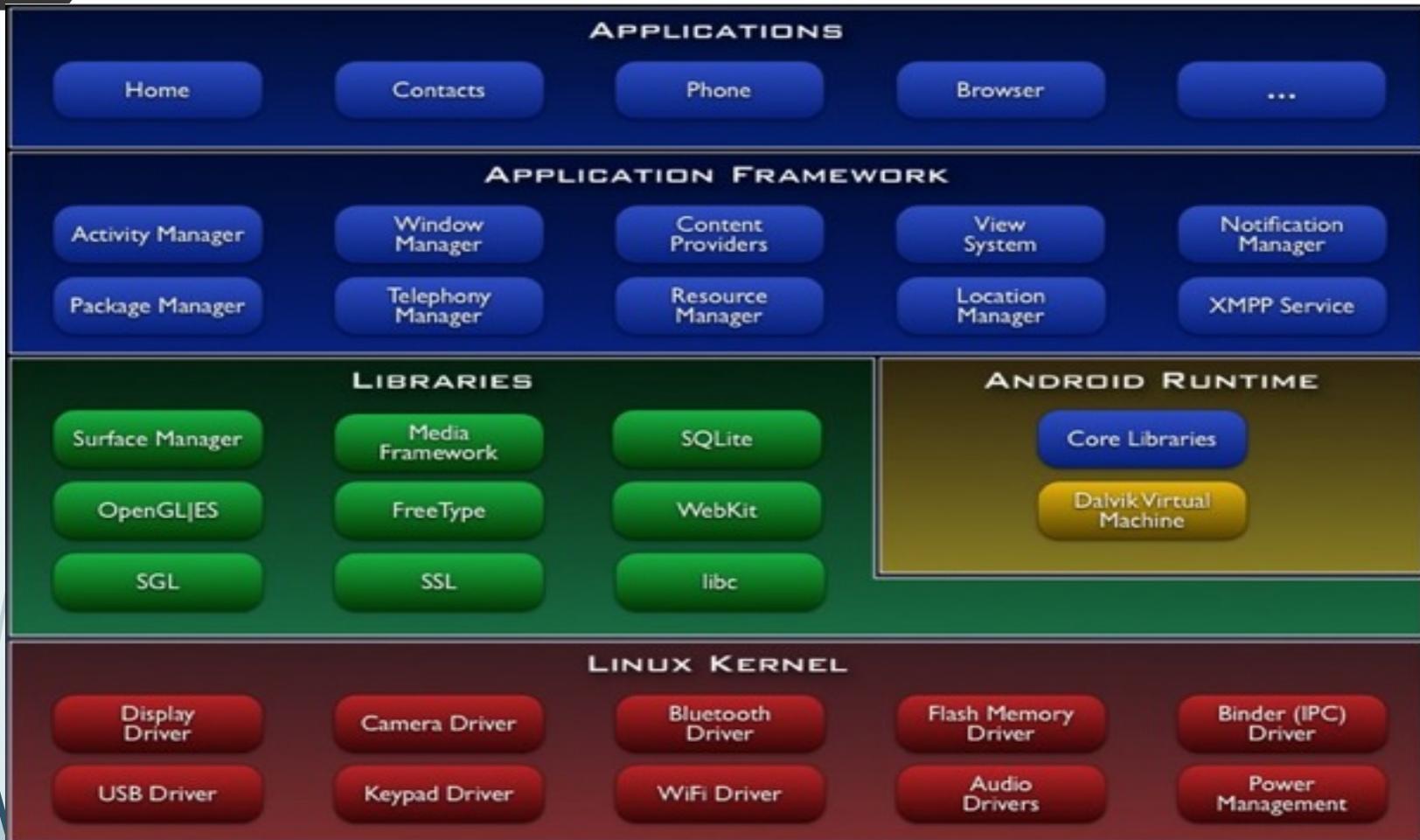


Plateforme Android

Le système d'exploitation Android est basé sur Linux. Au plus bas niveau de ce système se trouve un noyau Linux destiné à la gestion du matériel comme :

- ▶ Drivers de ces terminaux,
- ▶ La gestion de la mémoire,
- ▶ La gestion des processus
- ▶ L'accès au réseau
- ▶ ...

Plateforme / Architecture



Android pour tous

- Développeurs
 - Pas besoin de licence
 - Simple et intuitifs
 - Modulables
- Constructeurs
 - Tous les constructeurs peuvent utiliser Android
 - Un ensemble de services sont déjà disponibles dans le core
 - API disponible pour les accès de bas niveau

Android et les langages de programmation

- Android n'est pas un langage de programmation
- Pour développer sous Android, il existe deux possibilités :
 - Développement native (Java, C, C#)
 - Développement hybride (Titanium, PhoneGap)

Android & Java

- Le SDK Android est développé en Java → Permet de développer des applications avec un haut niveau d'abstraction
- Android a sa propre machine virtuelle Dalvik Virtual Machine
- Ne supporte pas toutes les fonctionnalités de la JRE
- Une application Android ne peut pas s'exécuter sur une machine virtuelle Java
- Une application Java (native) ne peut pas s'exécuter sous Android

Android & C/C++

- Il est possible d'écrire des applications Android en utilisant le langage C/C++ qui seront exécutées directement par le système d'exploitation Linux embarqué
- Android fournit le kit de développement NDK pour les développements d'application en C/C++
- Utilisé dans le développement de jeux 2D/3D se basant fortement sur la librairie OpenGL

Android vs Développement Hybride

- Android supporte le développement hybride
 - Titanium
 - Phonegap
 - Neomad

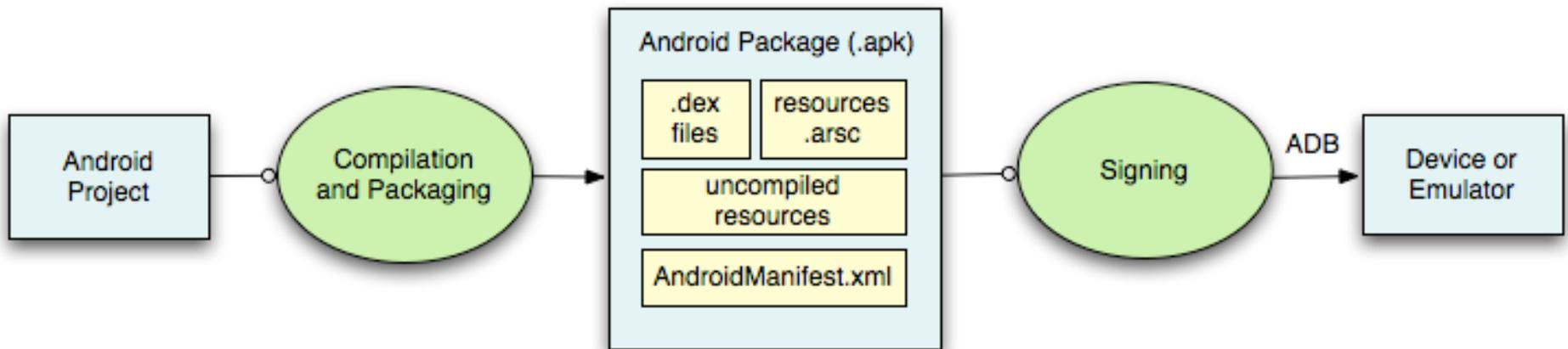
Pour ce cours....

- Framework et langage
 - Android SDK
 - Java
 - XML
- Outils
 - Android Studio

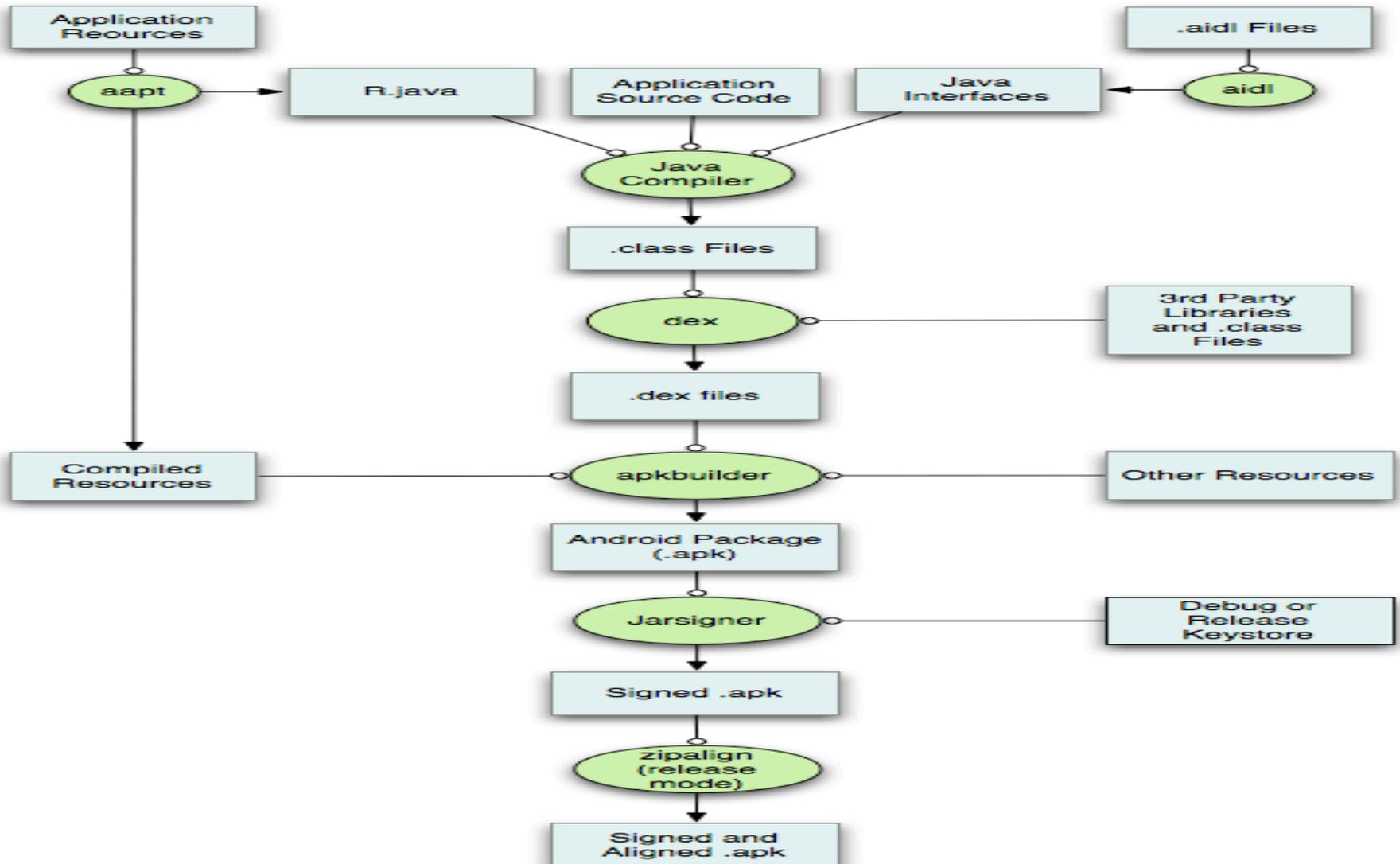
Outils du SDK

- **adb** → Accessibles à partir d'une ligne de commande (fenêtre DOS) adb permet la connexion au terminal (smartphone ou simulateur) pour :
 - Transférer des fichiers (push / pull)
 - Installer une application (install)
 - Paramétrer le réseau (forward)
- **dx** → Transforme le bytecode java en code Dalvik
- **apkbuilder** → Compiler les sources d'une application Android pour constituer une archive (.apk) directement installable sous un terminal Android
- **DDMS / Monitor** → Monitoring sur les activités du terminale

Production de logiciel



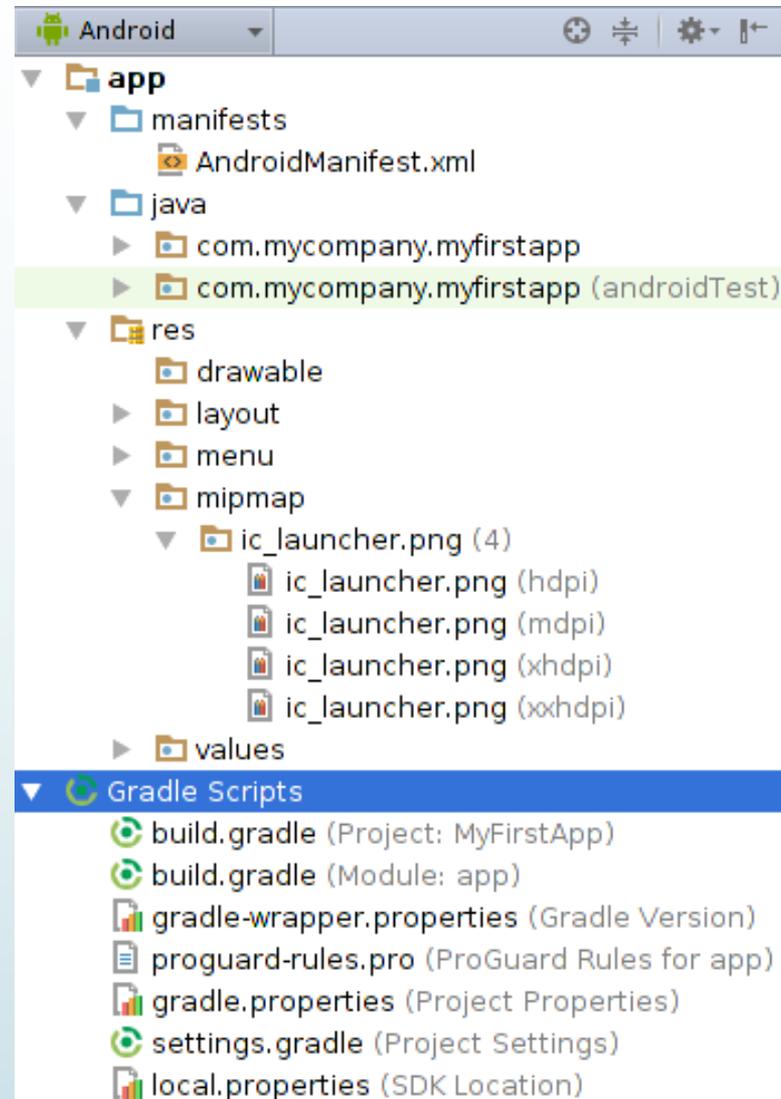
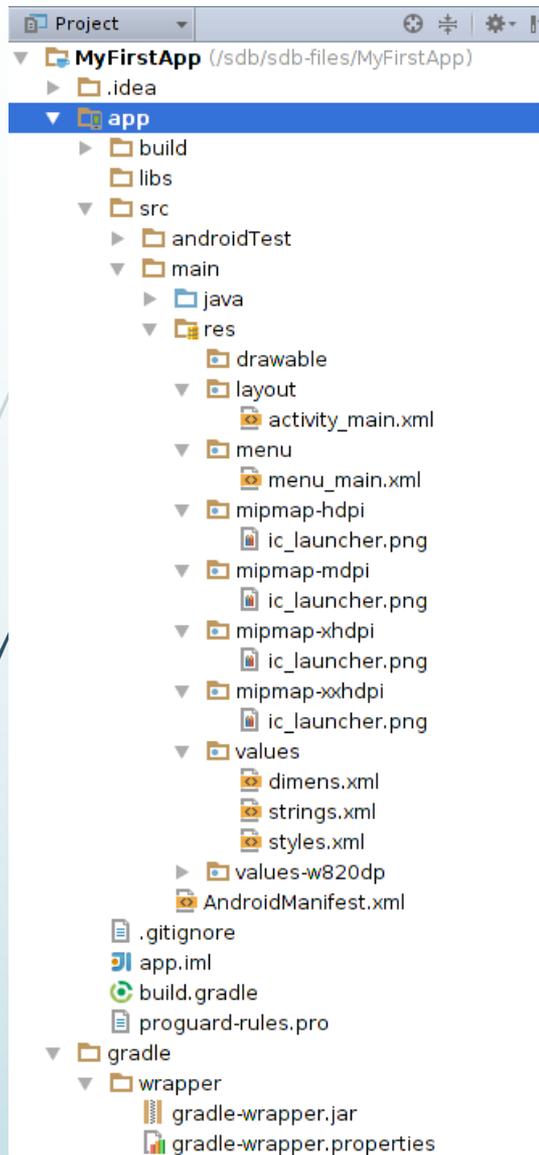
Production de logiciel



Type d'applications Android

- Application de premier plan : Application utilisable que lorsqu'elle est visible et n'effectuant aucune tâche de fond
- Services : Application s'exécutant en tâche de fonds et ne présentant pas d'interfaces utilisateurs
- Intermittente : Application exécutant à la fois des tâches des fonds et permet l'interaction avec l'utilisateur; la communication entre les tâches de fonds et l'utilisateurs se fait par des notifications
- Widgets: Application(utilitaires) pouvant être placé directement sur un écran du téléphone

Structure d'une application



Structure d'une application

- App: Racine de l'arborescence d'un projet Android; nom du module principal
- Manifests : Contient le manifest (fichier de configuration d'une application)
- Java: Dossier contenant les sources de l'application (code java)
- Res: Contient les ressources statiques de l'application
 - Drawable: contient les images
 - Layout: les interfaces graphiques
 - Menu : les menus de l'application
 - Mipmap: Icones de l'application
 - Values : Variables statiques
 - Colors: code de couleurs
 - Dimens: Dimensions (marges, ...)
 - Strings: textes statistiques

Gradle

- Outil de gestion, d'intégration et de génération d'application Android
- Déclare et exécute les tâches nécessaires à l'exécution d'application Android
- Permet de développer des applications multiplateforme et multiprojet

Gradle (Example)

```
android {
    compileSdkVersion 19
    buildToolsVersion "19.0.0"

    defaultConfig {
        minSdkVersion 8
        targetSdkVersion 19
        versionCode 1
        versionName "1.0"
    }
    buildTypes {
        release {
            minifyEnabled true
            proguardFiles getDefaultProguardFile('proguard-android.txt'), 'proguard-rules.pro'
        }
    }
}

dependencies {
    compile project(":lib")
    compile 'com.android.support:appcompat-v7:19.0.1'
    compile fileTree(dir: 'libs', include: ['*.jar'])
}
```

Composantes d'une application

- **Activité** (`android.app.Activity`) → Programme qui gère une interface graphique
- **Service** (`android.app.Service`) → Programme qui fonctionne en tâche de fond sans interface
- **Fournisseur de contenu**
(`android.content.ContentProvider`) → Partage d'informations entre applications
- **Intentions** : Communication inter et intra-application
- **Ecouteur d'intention**
(`android.content.BroadcastReceiver`) → Permet à une application de récupérer des informations générales (réception d'un SMS, batterie faible, ...)
- **Widgets** : Bibliothèque de composants visuels
- **Manifest**: Fichier de configuration de l'application

AndroidManifest

- Configuration de l'application
 - Définit le nom de package de l'application;
 - Décrit les composants de l'application: activités, services, fournisseurs de contenus...., il nomme les classes implémentant chacun de ces composants et propage leur capacité sur le système;
 - Détermine quel processus répond à une action donnée;
 - Déclare les permissions de l'application;

Manifest : Exemple

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

<manifest
xmlns:android="http://schemas.android.com/apk/res/android"
package="ht.solutions.android.whereami"
android:versionCode="1" android:versionName="1.0">

<uses-permission
android:name="android.permission.ACCESS_FINE_LOCATION"/>

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

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

<activity android:name="Home"
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>
```

Contexte (Context)

- Interface fournissant des informations sur l'environnement de l'application:
- C'est une classe abstraite implémentée par le système Android
- Il permet d'accéder aux principales ressources de l'application
- Obtenir le contexte courant d'une application
 - Contexte global de l'application
 - `Context c = getApplicationContext();`
 - Contexte d'une activité ou un service
 - `Context c = Activity.this;`
 - `Context c = getContext();`

Les Intentions (Intent)

Attributs principaux

Action : Action à exécuter

Data : Données sur lesquelles l'action va opérer

Autres attributs

Category : Précise la catégorie de l'activité demandée

Type : Préciser le type de contenus des données de l'intention

Extras : Informations additionnelles envoyées à l'action (envoi d'un email, par exemple)

Intent/Action

L'action d'un **intent** indique au système l'action à exécuter. En fonction de l'action le système déterminera l'activité à lancer.

Certaines actions sont prédéfinies:

- **ACTION_VIEW**: Afficher une information; généralement indiquée par une URI.
 - ACTION_VIEW:contacts/people/1 : Afficher le profile d'une personne
 - ACTION_VIEW:contacts/people : Affiche la liste des contacts
- **ACTION_DIAL**: Composer un numéro de téléphone.
 - ACTION_DIAL:tel:06xxxxxxx
 - ACTION_DIAL:contacts:contacts/people/1

Intent/Data

Contient les données sur lesquelles l'activité appelée peut opérer.

Par exemple, l'id d'un contact à afficher ou à supprimer.

Les « data » sont généralement exprimés via des URIs

Exemple : `contents://contacts/people/1`

Intent/Data

Contient les données sur lesquelles l'activité appelée peut opérer.

Par exemple, l'id d'un contact à afficher ou à supprimer.
Les « data » sont généralement exprimés via des URIs

Exemple : `contents://contacts/people/1`

Intent/Data

Data

Contient les données sur lesquelles l'activité appelée peut opérer.

Par exemple, l'id d'un contact à afficher ou à supprimer.

Les « data » sont généralement exprimés via des URIs

Exemple : `contents://contacts/people/1`

Category

Information supplémentaire sur l'action à exécuter

→ `ACTION_MAIN & CATEGORY_HOME`

→ `ACTION_GET_CONTENT & CATEGORY_OPENABLE`

Résolution d'intention

Il existe deux principaux types d'intentions :

- **Explicite** : Spécifie les composants qui précisent la classe exacte qui doit être exécutée (`setComponent (nom)` ou `setClass(context, class)`)
- **Implicite** : Ne spécifie pas le composant mais fournit suffisamment d'informations permettant au système de déterminer les composants nécessaires correspondant à cette action.

Filtre d'intention

- Les filtres d'intentions permettent de définir les actions supportées par une activité
- Les filtres d'intentions sont déclarés dans le manifest au moment de définir l'activité

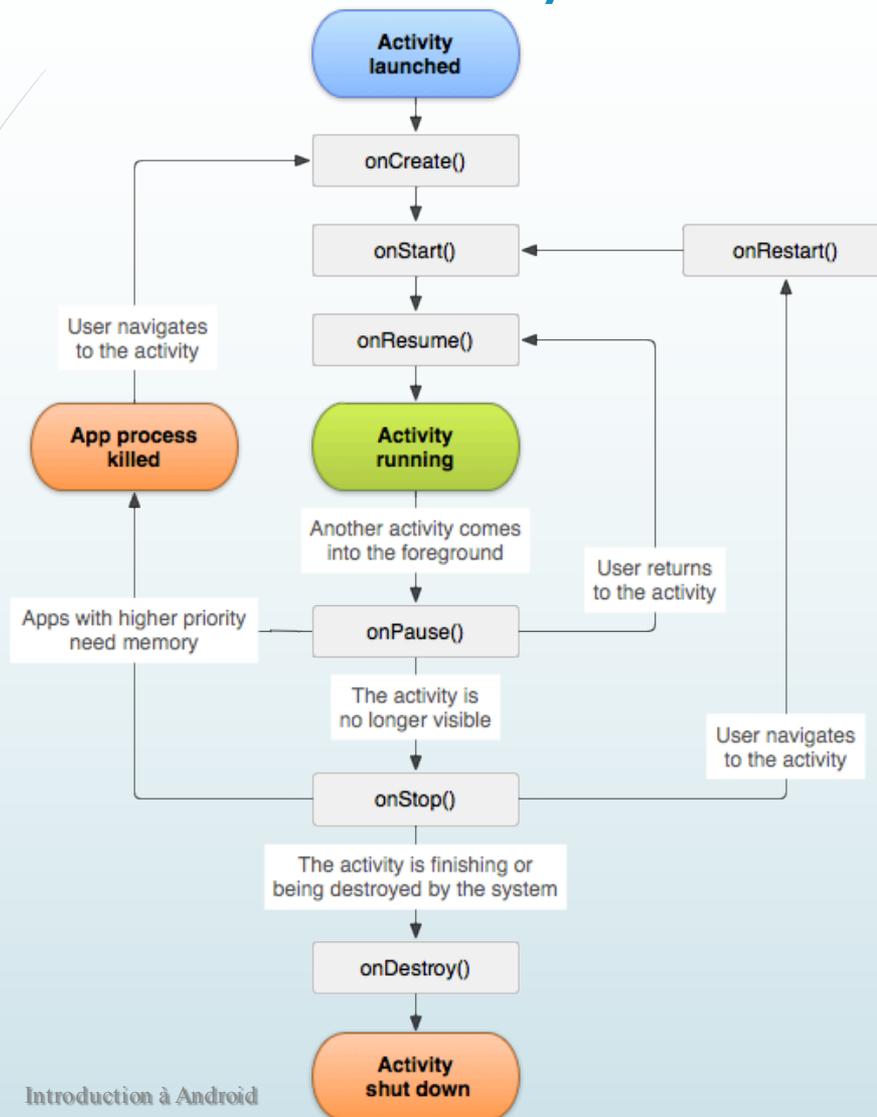
Activité (Activity)

- Modélise et gère les interfaces de l'application. Dans un modèle MVC il joue le rôle de contrôleur
- Hérite de la classe Activity d'Android ou une de ses dérivés
- Doit être déclarée dans le Manifest pour être visible par le système
- Ne peut être instanciée directement, cette tâche est effectuée par le système

```
Activity a = new Activity();
```

- Une activité est instanciée en utilisant les intentions

Activité /cycle de vie



Activité / Création

```
public class ExampleActivity extends Activity {
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        // The activity is being created.
    }
    @Override
    protected void onStart() {
        super.onStart();
        // The activity is about to become visible.
    }
    @Override
    protected void onResume() {
        super.onResume();
        // The activity has become visible (it is now "resumed").
    }
    @Override
    protected void onPause() {
        super.onPause();
        // Another activity is taking focus (this activity is about to be
        "paused").
    }
}
```

Activité/Configuration

- Une activité doit être obligatoirement déclarée dans le Manifest

```
<manifest ... >
```

```
  <application ... >
```

```
    <activity android:name=".ExampleActivity" />
```

```
    ...
```

```
  </application ... >
```

```
  ...
```

```
</manifest >
```

Activité/Configuration

- ▶ Quelques propriétés de la balise activity
 - ▶ `android:configChanges` : Abonnement à des événements particuliers
 - ▶ `touchscreen` : L'utilisateur touche l'écran
 - ▶ `orientation` : L'utilisateur réoriente le téléphone
 - ▶ `Icon` : Image qui apparaît en haut à gauche dans la barre de navigation
 - ▶ `screenOrientation` : Spécifier les types d'orientation supportés par l'interface
 - ▶ `Landscape`
 - ▶ `Portrait`
 - ▶ `uiOptions` : Définit le comportement de la barre d'action
 - ▶ `None` : Comportement par défaut
 - ▶ `splitActionBarWhenNarrow` : Ajouter une action bar au bas de l'écran quand le nombre d'item est supérieur à l'espace restant.

Activité/Configuration

```
<activity class=".NotesList" android:label="@string/title_notes_list">
  <intent-filter>
    <action android:name="android.intent.action.MAIN" />
    <category android:name="android.intent.category.LAUNCHER" />
  </intent-filter>
  <intent-filter>
    <action android:name="android.intent.action.VIEW" />
    <action android:name="android.intent.action.EDIT" />
    <action android:name="android.intent.action.PICK" />
    <category android:name="android.intent.category.DEFAULT" />
    <data android:mimeType="vnd.android.cursor.dir/vnd.google.note" />
  </intent-filter>
  <intent-filter>
    <action android:name="android.intent.action.GET_CONTENT" />
    <category android:name="android.intent.category.DEFAULT" />
    <data android:mimeType="vnd.android.cursor.item/vnd.google.note" />
  </intent-filter>
</activity>
```

La méthode startActivity

Permet de lancer une nouvelle activité.

```
Intent i = new Intent(this, MainActivity.class);  
startActivity(i);
```

```
Uri chemin = Uri.parse("http://www.google.fr");  
Intent naviguer = new Intent(Intent.ACTION_VIEW, chemin);  
startActivity(naviguer);
```

```
Uri numero = Uri.parse("tel:0123456789");  
Intent appeler = new Intent(Intent.ACTION_CALL, numero);  
startActivity(appeler);
```

startActivityResult

- La méthode `startActivityResult (intent)`, permet de démarrer une activité et attendre un résultat
- Elle permet à l'activité appelée de renvoyer des résultats à l'activité appelante
- Par exemple une activité de Login doit pouvoir renvoyer le statut de la connexion à l'activité appelante : **FacebookLogin fonctionne ainsi**
- La méthode `onActivityResult` permet de récupérer les données renvoyées par l'activité appelante
- La méthode `setResult` permet à l'activité appelée de renvoyer des résultats à l'activité appelante

startActivityForResult- Exemple

Activité A

```
Intent intent = new Intent(Intent.ACTION_PICK, Contacts.CONTENT_URI);
startActivityForResult(intent, 0);
```

Activité B

```
Intent intent = getIntent();
i.putExtra("contact", theContact);
setResult(RESULT_OK, intent);
```

Activité A

```
onActivityResult (int requestCode, int resultCode, Intent data)
```

- requestCode : Identifie l'action qui lancé l'activité
- resultCode: Indique le statut du résultat (RESULT_OK, ...)
- Data : renforme des données renvoyées par l'activité appelée

Intent - Paramètres

- La classe Intent permet de passer des paramètres à l'activité appelée et d'en récupérer les valeurs en retour
- Ajouter des paramètres (types simples ou tableaux)

Exemple:

```
intent.putExtra(String, val)
```

- Le 1er paramètre est un nom (clé)
- Le second paramètre est la valeur
 - De type simple (boolean, int, short, long, float, double, char)
 - Tableau de types simples

L'activité appelée pourra récupérer ces paramètres par leur nom

Activités / Paramètres (Bundle)

L'activité lancée récupère l'objet de classe Bundle contenant les paramètres par :

```
Bundle params = getIntent().getExtras()
```

- Les paramètres sont récupérés dans ce Bundle par ses méthodes :

- `getBoolean(String)`

- `getInt(String)`

- `getBooleanArray(String)`

- ...

Exemple :

```
String myId = getIntent().getStringExtra(« id »);
```

Ressources (Resource)

- Une application Android n'est pas seulement fait de codes mais aussi de ressources statiques (images, sons, textes statiques,...)
- Tout projet Android a un dossier de ressources (`res/`) contenant les ressources du projet (bitmap, xml,...)
 - ▶ `/res/drawable` → images (`R.drawable.nom_de_la_ressources`)
 - ▶ `/res/layout` → Design des vues (`R.layout.nom_de_la_vue`)
 - ▶ `/res/values/strings` → Chaînes de caractères, tableaux, valeurs numériques ... (`R.string.nom_chaine`, `R.array.nom`)
 - ▶ `/res/anim` → description d'animations (`R.anim.nom_animation_`)
 - ▶ `/res/menus` → Menus pour l'application (`R.menu.nom_menu`)
 - ▶ `/res/values/color` → Code de couleurs (`R.color.nom_couleur`)
 - ▶ ...

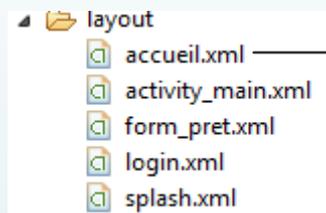


Ressources & valeurs

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
<string name="app_name">Mon application</string>
<string name="server_url">http://mon.serveur.com</string>
<integer-array name="codes_postaux">
<item>64100</item>
<item>33000</item>
</integer-array>
<string-array name="planetes">
<item>Mercure</item>
<item>Venus</item>
</string-array>
<dimen name "taille">55px</dimen>
</resources>
```

Référencer les ressources

- L'ensemble ressources sont modélisés par la classe « R.java » et les sous dossiers par des classes internes à R
- Chaque ressource est un attribut de la classe représentant le sous dossier dans lequel il est déclaré



→ R.layout.accueil

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
<string name="app_name">Mon application</string>
.....
</resources>
```

→ R.string.app_name

Référencer les ressources

Référencement d'une ressource dans une autre ressource. La forme générale est : "**@type/identificateur**"

```
<TextView
    android:id="@+id/lblUsername"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_weight="0.00"
    android:text="@string/lblUsername"
    android:textColor="@android:color/holo_blue_dark"
    android:textAppearance="?android:attr/textAppearanceMedium"
/>
```

Référencer les ressources

- Dans le code (java), on obtient une instance de cette classe par `getResources()`
- Principales méthodes de la classe `Resources` (le paramètre est un identifiant défini dans `R` de la forme `R.type.nom`) :
 - `boolean getBoolean(int)`
 - `int getInteger(int)`
 - `int[] getArray(int)`
 - `String getString(int)`
 - `String[] getStringArray(int)`
 - `int getColor(int)`
 - `float getDimension(int)`
 - `Drawable getDrawable(int)`

Exemple : `String titre = context.getResources().getString(R.string.ma_chaine);`

Mais dans une activité on peut faire plus simplement :

```
String titre = getString(R.string.ma_chaine);
```

Référencer les ressources

- Accéder aux vues :

```
getResources().getLayout(R.layout.nom_layout);
```

- Accéder aux valeurs :

```
String chaine = getResources().getString (R.string.nom_string);
```

```
String[] tableau= getResources(). getStringArray(R.string.nom_array);
```

- Accéder aux images :

```
Drawable monImage =  
getResources().getDrawable(R.drawable.nom_image)
```

Interfaces (Layout)

Les interfaces (Layout) permettent de dessiner la vue tel qu'elle doit s'afficher à l'utilisateur.

Il existe deux possibilités de développer des interfaces :

- Code java
- Fichier XML

Android recommande l'utilisation des fichiers XML pour définir les interfaces.

- La méthode `setContentView` permet d'associer un layout à une activité

Exemple :

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

Interfaces (Layout) / Vues

Les interfaces sont composées de vues :

Les vues ne peuvent être modifiées que par le thread principal « UIThread » .

Tous les autres threads créés par un utilisateur ne peuvent pas modifier les vues.

Vues - Propriétés communes

Identifiant :

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

Dimension:

`layout_height, android:layout_width : fill_parent /
match_parent / wrap_content`

fill_parent : Remplit toute la place

wrap_content : remplit la place que necessite le contenu

match_parent : remplit la place qui reste dans le parent

Fond

`android: background`

Visibilité

`android:visibility : visible / invisible/gone`

Vues Propriétés communes

Marges internes

- ▶ `android:layout_paddingBottom` ,
`android:layout_paddingLeft` ,
- ▶ `android:layout_paddingRight` ,
`android:layout_paddingTop`

Marges externes

- ▶ `android:layout_marginBottom` ,
`android:layout_marginLeft` ,
- ▶ `android:layout_marginRight` ,
`android:layout_marginTop`

Vues

Les vues déclarées dans les fichiers xml sont automatiquement instanciées par Android.

Pour récupérer l'instance d'une vue associée à une interface, Android propose la méthode `findViewById()` prenant en paramètre l'identifiant de la vue dans le fichier xml.

La méthode `findViewById` utilise le conteneur principal de l'interface de l'activité courante.

Exemple :

```
View v = findViewById(R.id.myView);
```

Connaissant le type de la vue

```
Button b = (Button)findViewById(R.id.btnCancel);
```

Les vues peuvent être créés dynamiquement :

```
Button b = new Button(this);
```

```
b.setId(1245);
```

Vues / Evènements

Les évènements permettent de gérer les actions utilisateurs sur les vues:
Pour gérer les évènements sur les vues, il suffit d'ajouter un écouteur

```
button.setOnClickListener(new View.OnClickListener()  
@Override  
public void onClick(DialogInterface dialog, int which) {  
// Du code ici  
}  
});
```

```
button.setOnClickListener(new OnClickListener() {  
@Override  
public void onClick(View v) {  
// TODO Auto-generated method stub  
}  
});
```

Les conteneurs (ViewGroups)

Permettent de positionner des widgets dans une interface :

- `FrameLayout`
- `AbsoluteLayout`
- `LinearLayout`
- `TableLayout`
- `RelativeLayout`
- `ScrollView`
- `HorizontalScrollView`

LinearLayout

Définit le positionnement linéaire (horizontal ou vertical) des vues filles

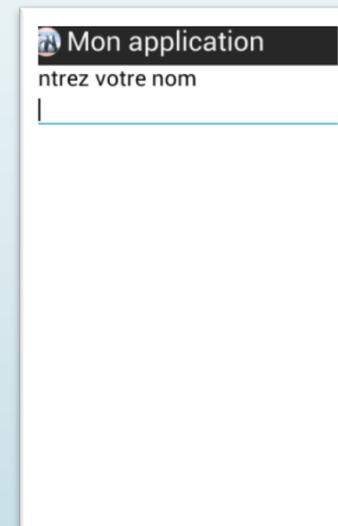
```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical" >

    <TextView
        android:id="@+id/textView1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/Label_nom"
        android:textAppearance="?android:attr/textAppearanceLarge" />

    <EditText
        android:id="@+id/editText1"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:ems="10"
        android:inputType="textPersonName" >

        <requestFocus />
    </EditText>

</LinearLayout>
```



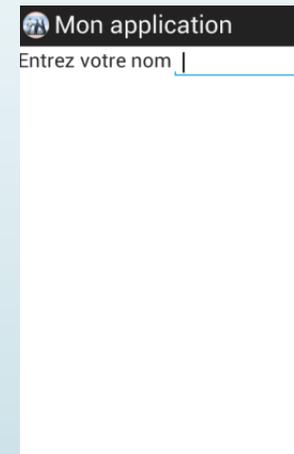
LinearLayout

```
<?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" >

    <TextView
        android:id="@+id/textView1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/label_nom"
        android:textAppearance="?android:attr/textAppearanceLarge" />

    <EditText
        android:id="@+id/editText1"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:ems="10"
        android:inputType="textPersonName" >

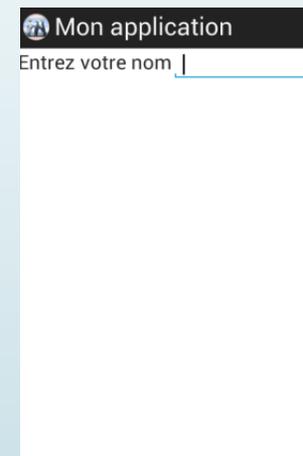
        <requestFocus />
    </EditText>
</LinearLayout>
```



RelativeLayout

Positionner les éléments de l'interface les uns par rapport aux autres

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="horizontal" >
    <TextView
        android:id="@+id/textView1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/label_nom"
        android:textAppearance="?android:attr/textAppearanceLarge" />
    <EditText
        android:id="@+id/editText1"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:ems="10"
        android:layout_toRightOf="@+id/textView1"
        android:inputType="textPersonName" >
        <requestFocus />
    </EditText>
</RelativeLayout>
```



TableLayout

- Tableau de positionnement des vues en ligne de TableRow (similaire au `<table>` `<tr>` `<td>` de HTML)
- TableRow hérite de LinearLayout avec alignement automatique des colonnes sur chaque ligne
- Propriétés de TableRow.LayoutParams
 - ▀ `layout_column`: indice de départ de la colonne (à partir de 0)
 - ▀ `layout_span`: nombre de colonnes occupées

Widgets

- Les widgets représentent les blocs élémentaires visibles dans l'interface utilisateurs. Une vie occupe une portion rectangulaire et répond aux évènements d'interaction entre l'utilisateur final et l'application
- Leur disposition dans l'interface est défini par son contenant (Viewgroup)
- On distingue :
 - TextView
 - Button
 - EditText
 - CheckBox
 - RadioButton
 - ImageView
 - ...

Widgets

```
<TextView android:id="@+id/my_text"  
android:layout_width="match_parent"  
android:layout_height="wrap_content"  
android:layout_weight="0.00"  
android:text="@string/lblUsername"  
android:textColor="@android:color/holo_blue_dark"  
android:textAppearance="?android:attr/textAppearanceMedium"  
/>
```

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

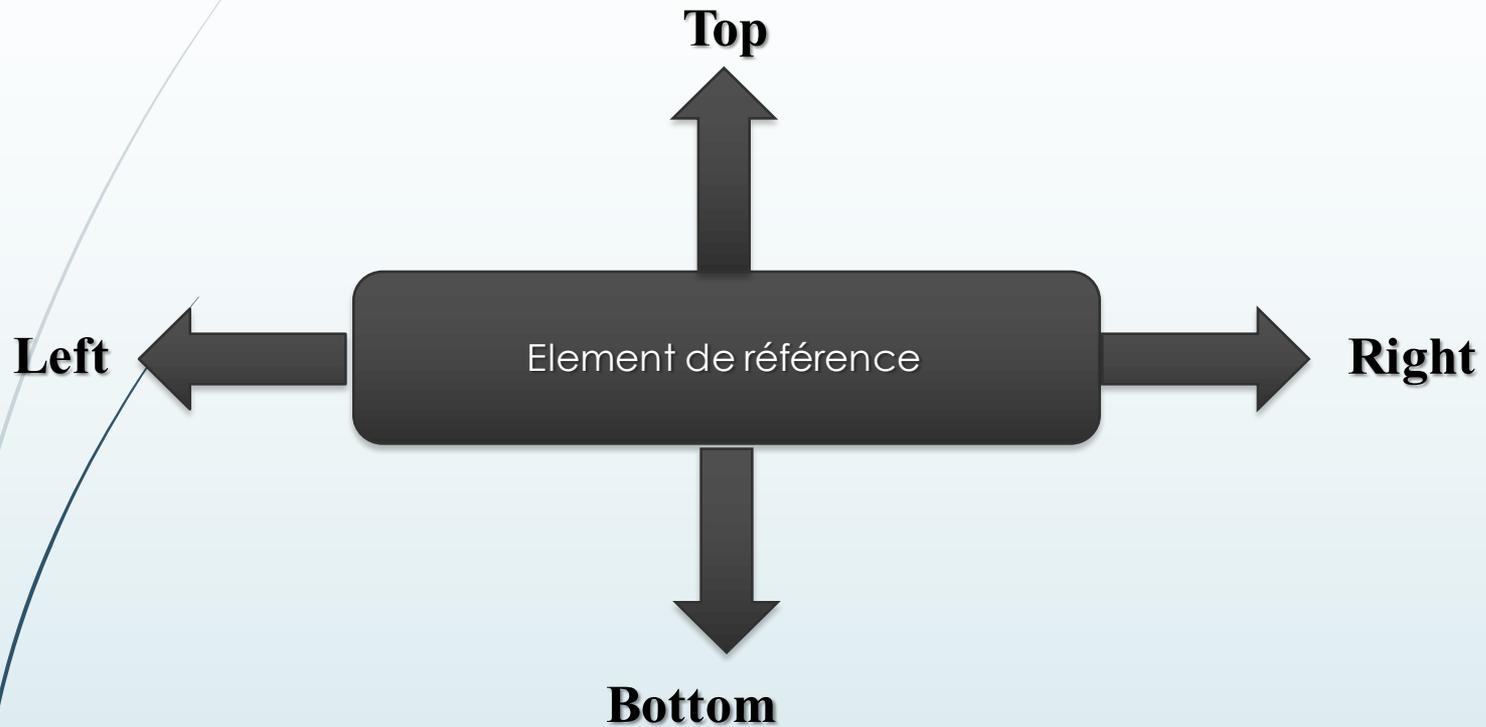
```
<ImageView android:id="@+id/imageView1"  
android:layout_width="fill_parent"  
android:layout_height="wrap_content"  
android:contentDescription="@string/image_description"  
android:src="@drawable/login" />
```

Travaux Pratiques

RelativeLayout

- ▶ Un RelativeLayout permet de positionner des éléments de manière relatives dans une vue. Ils peuvent être positionnés les uns par rapport aux autres ou par rapport à leur parent.
- ▶ Les éléments sont rendus indépendamment de leur ordre d'apparition dans le fichier XML
- ▶ Par défaut si toutes les vues d'un relativelayout sont positionnés en haut et à gauche de la vue parent

RelativeLayout



Positionnement relatif...

- Positionner un élément au dessus d'un élément de référence
android:layout_above="@id/element_ref»
- Positionner un élément en dessous d'un élément de référence
android:layout_below="@id/element_ref»
- Positionner un élément à droite d'un élément de référence
android:layout_toRightOf="@id/element_ref»
android:layout_toEndOf="@id/element_ref»
- Positionner un élément à gauche d'un élément de référence
android:layout_toLeftOf="@id/element_ref»
android:layout_toStartOf="@id/element_ref»

Autres propriétés...

- Positionner un élément par rapport à son conteneur (parent)

`android:layout_centerVertical="true"`

`android:layout_alignParentBottom="true"`

`android:layout_alignParentTop="true"`

`android:layout_alignParentLeft="true"`

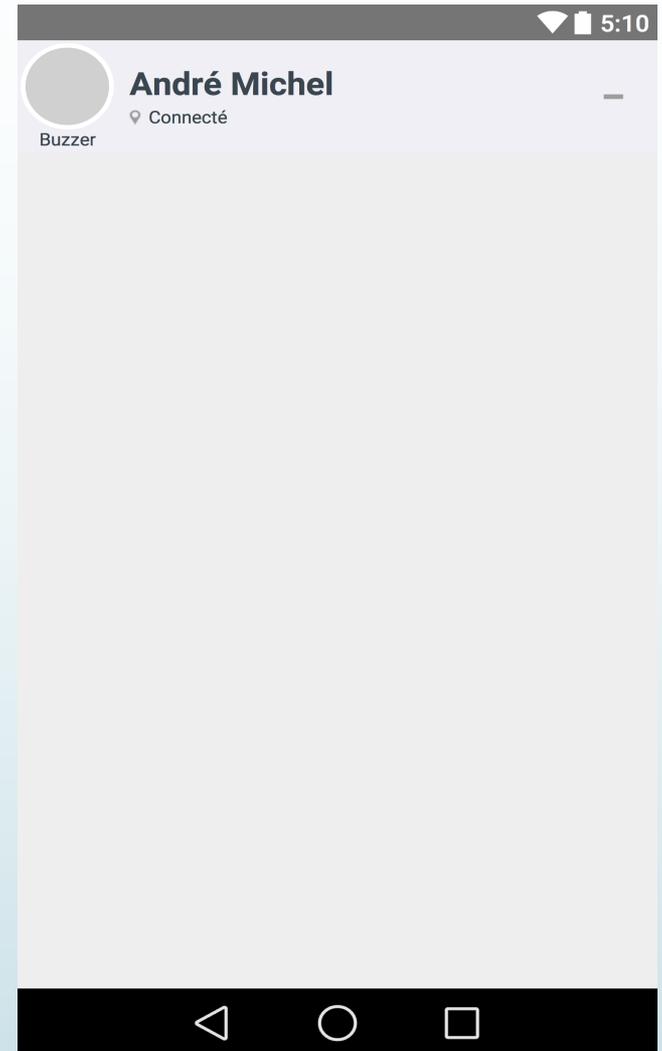
`android:layout_alignParentRight="true"`

`android:layout_centerVertical="true"`

`android:layout_centerHorizontal="true"`

Positionnement relatif...

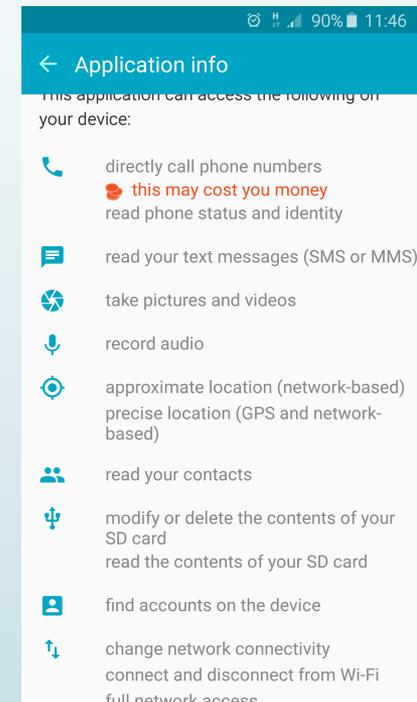
- Essayez de construire cette interface
- Si vous êtes courageux, refaites les interfaces existantes de votre application en utilisant les `RelativeLayout` au lieu de `LinearLayout`



Séance 2

Permission

- Moyen de sécurité proposé par Android pour gérer l'accès aux ressources du terminal
- Elles apparaissent dans le fichier Android Manifest et sont visibles par l'utilisateur au moment de l'installation de l'application
- Elles concernent :
 - La géolocalisation (GPS)
 - Les accès aux contacts et à l'agenda du téléphone
 - Les modifications de paramètres (orientation, fond d'écran ...)
 - Les appels téléphoniques
 - L'envoi et réception de SMS/MMS
 - L'audio
 - Le réseau (dont l'accès à Internet)
 - Le matériel (bluetooth, appareil photo, ...)



Mécanisme de Sécurité en Android

- Par défaut, une application n'a pas l'autorisation d'effectuer des actions pouvant affecter d'autres applications, le système d'Exploitation ou l'utilisateur.
- Chaque application est considérée dans une sorte de boîte noire vis-à-vis des autres applications.
- Ce mécanisme est indépendant du type de technologie utilisé pour développer l'application (native ou hybride)
- Pour exposer ses données, une application doit le déclarer explicitement
- De même, pour accéder aux données d'autres applications, il est obligatoire de demander la permission appropriée.

Signature des Applications

- Toute application est signée par un certificat dont le développeur est seul propriétaire de la clef privé.
- Pas besoin d'un 'Certificate AUTHORITY', Il peut être généré simplement par le développeur.
- Le certificat permet d'identifier les auteurs des applications
- La signature permet de garantir l'unicité des applications dans Linux.
- A l'installation Android l'OS affecte un ID unique à l'application sur le terminal

Utiliser les Permissions

La déclaration des permissions doit se faire explicitement dans le fichier manifest.

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

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

- Chaque Opération « sensible » nécessite une permission appropriée
- Il est interdit d'utiliser une ressource pour laquelle la permission n'a pas été accordée.

Gestion des permissions (1)

- ▶ API inférieure à 5.1 ou `targetSdkVersion < 22`
 - ▶ Les permissions sont accordées à l'installation de l'application.
 - ▶ Si une nouvelle permission est ajoutée dans une mise à jour, l'autorisation est redemandée à l'utilisateur
 - ▶ Le seul moyen de refuser une permission est de désinstaller l'application

Gestion des permissions (2)

- ▶ API supérieure à 5.1 ou targetSDKVersion < 23
 - ▶ Les permissions sont demandées à l'utilisateur à l'exécution de l'application
 - ▶ La permission n'est demandée qu'au moment de l'accès à la ressource
 - ▶ L'utilisateur peut évoquer une permission à n'importe quel moment

Niveaux de permissions

- Deux niveaux de permission
 - Normale
 - Dangereuse

Permission normale

- Celles qui ne présentent pas de risque ou des risques minimes pour l'utilisateur
- Automatiquement accordées par le système
- NFC
- Internet
- Vibrate
- Set_Alarm

- Liste complète →
<https://developer.android.com/guide/topics/security/normal-permissions.html>

Permission Dangereuse

- Regroupe toutes les permissions qu'une application demande d'accéder aux données d'une autre application
- Le système demande à l'utilisateur l'autorisation formelle de la permission (via une boîte de dialogue)
- Par exemple
 - [READ_CONTACTS](#)
 - [WRITE_CONTACTS](#)

Permission Dangereuse (2)

- Les permissions dangereuses sont regroupées dans des groupes « permission group »
- Si une permission spécifique est demandée, android présente à l'utilisateur la description du groupe auquel elle appartient.
 - Si l'utilisateur l'accepte toutes les autres permissions du groupe sont aussi accordées, même si elles n'étaient pas demandées
- Exemple: SMS
- [SEND_SMS](#), [RECEIVE_SMS](#), [READ_SMS](#), [RECEIVE_WAP_PUSH](#), [RECEIVE_MMS](#)

<https://developer.android.com/guide/topics/security/permissions.html>

Connectivité

- Android fournit des mécanismes permettant aux applications d'utiliser la connectivité pour accomplir certaines tâches :
 - Internet (WIFI, 3G,...)
 - Bluetooth
 - NFC
- Pour accéder à ces ressources, il est obligatoire de demander explicitement la permission associée
 - **Internet** : `<uses-permission android:name="android.permission.INTERNET"/>`
 - **NFC**: `<uses-permission android:name="android.permission.NFC"/>`
 - **Bluetooth**: `<uses-permission android:name="android.permission.BLUETOOTH"/>`

Internet

- Une application Android peut utiliser la connexion réseau du téléphone pour échanger des informations avec une application distante (web service, web application,..)
- Plusieurs protocoles internet sont supportés
 - HTTP, FTP, ...
- Permission pour utiliser internet :

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

- Android utilise la bibliothèque Apache Components pour les communications par http

```
HttpClient client =new DefaultHttpClient();  
HttpGet request =new HttpGet(url);  
HttpResponse response = client.execute(request);  
...
```

Taches Asynchrones

- La UIThread est le thread principal de gestion de l'interface utilisateur. L'exécution des processus longs dans ce thread comme l'accès au réseau doit se faire dans un thread à part.
- A partir de la version 4, Android génère une exception pour toute opération longue s'exécutant dans le thread principal.
- L'exécution des tâches asynchrones peut se faire de deux manières:
 - L'utilisateur crée explicitement un thread auquel il délègue l'exécution des processus longs
 - Utiliser la classe AsyncTask du SDK qui se chargera de la création du thread secondaire

Taches Asynchrones

```
private class DownloadFilesTask extends AsyncTask<URL, Integer, Long> {  
    protected Long doInBackground(URL... urls) {  
        long totalSize = 0;  
        return totalSize;  
    }  
  
    protected void onPreExecute() {  
        //Avant le téléchargement (ex. Afficher un loading)  
    }  
  
    protected void onProgressUpdate(Integer... progress) {  
        setProgressPercent(progress[0]);  
    }  
  
    protected void onPostExecute(Long result) {  
        showDialog("Downloaded " + result + " bytes");  
    }  
}
```

Taches Asynchrones (2)

Les paramètres

- Premier type : définit le type des de données pour la méthode `doInBackground` .
- Les informations sont envoyées sous formes de tableaux (non dimensionnés); définies à l'instanciation de l'objet
- Deuxième Type : Paramètre pour la méthode `onProgressUpdate` sous forme de tableaux non dimensionnées également
- Troisième Type: Type de retour pour la méthode `doInBackground` et paramètre pour la méthode `onPostExecute`.

Taches Asynchrones (3)

Enchaînement des appels

- ▶ La classe peut être instancié de cette manière
`new DownloadFilesTask().execute(url1, url2, url3);`
url1, url2, url3 sont les URLs à télécharger.
- ▶ En appelant la méthode exécute, ce qui se passe :
 - ▶ onPreExecute est appelée → Thread principal
 - ▶ doInBackground est appelée → thread secondaire
 - ▶ Fair appel à publishProgress
 - ▶ onProgressUpdate est appelée → dans le thread principal
 - ▶ onPostExecute est appelé → dans le thread principal

Travaux Pratiques

Les vues de groupe

Les vues de groupes permettent d'afficher une collection d'éléments:

- ListView
- GridView
- RadioGroupe
- Galery
- Spinner

Les adaptateurs

- Les adaptateurs jouent le rôle de pont entre une vue et ses données
- Un adaptateur a la responsabilité de gérer les items à afficher dans la vue de groupe.
- Chaque item de la liste de données est converti en élément de vue qui s'affichera dans la vue de groupe.

Les adaptateurs – Éléments de vues

- Les éléments d'une vue de groupe sont des constitués de vues
- Les vues sont construites en utilisant des layouts
 - Simples : définis par Android
 - Personnalisés : Définis par le développeur
- Les adaptateurs réutilisent les modèles de vues (ou templates) pour créer les éléments d'une vue de groupe

Les adaptateurs

- Les adaptateurs sont modélisés par des classes
 - ArrayAdapter
 - CursorAdapter
 - SimpleCursorAdapter
 - BaseAdapter
 - ListAdapter
 - ...

Liste complète sur <http://developer.android.com/reference/android/widget/Adapter.html>

Les adaptateurs

- Android propose des adaptateurs génériques que le développeur peut utiliser pour afficher des informations dans un élément de groupe
- Le développeur peut définir son propre adaptateur pour personnaliser l'affichage dans les groupes

ArrayAdapter

- Les ArrayAdapter permettent de créer des éléments d'une vue de groupe à partir d'un tableau d'éléments
- Les objets peuvent être de type :
 - Simples (String, Integer, ...)
 - Complexes (Ses propres de classe...)
- L'élément de base par défaut est un TextView
- L'adaptateur affecte au texte du TextView la méthode toString() de la donnée de référence
- Pour utiliser d'autres types d'élément il faut surcharger la méthode getView pour renvoyer le type de vue souhaitée

ArrayAdapter - Exemple

```
<string-array name="country_arrays">  
  <item>Malaysia</item>  
  <item>United States</item>  
  <item>Indonesia</item>  
  <item>France</item>  
  <item>Italy</item>  
  <item>Singapore</item>  
  <item>New Zealand</item>  
  <item>India</item>  
</string-array>
```

ArrayAdapter

```
//Récupérer la liste dans les ressources  
String [] countries=  
getResources().getStringArray(R.array.country_arrays);  
//Instance de la vue de groupe  
Spinner sp = (Spinner) findViewById(R.id.spinner1);  
//Créer l'adapteur  
ArrayAdapter<String> dataAdapter = new  
ArrayAdapter<>(context,  
android.R.layout.simple_spinner_item, countries);  
//Associer l'adapteur à l'instance de la vue de groupe  
sp.setAdapter(dataAdapter);
```

LayoutInflater

- Les LayoutInflater permettent de convertir un layout XML dans la vue correspondante
- Ne se créent pas directement mais à partir des méthodes `getLayoutInflater()` ou `getSystemService(class)`
- Peuvent être considérés comme des calques sur lesquels des layouts peuvent être calqués.

LayoutInflater

//Instance d'un layout inflater

```
LayoutInflater inflater = (LayoutInflater) context
```

```
.getService(Context.LAYOUT_INFLATER_SERVICE);
```

//Utiliser l'inflater pour récupérer une vue à partir d'un layout

```
View rowView = inflater.inflate(R.layout.rowlayout, parent, false);
```

ArrayAdapter - Personnalisé

```
public class MyAdapter extends ArrayAdapter<Item> {
    private final Context context;
    private final Item[] values;

    public MyAdapter(Context context, Item[] values) {
        super(context, R.layout.rowlayout, values);
        this.context = context;
        this.values = values;
    }

    @Override
    public View getView(int position, View convertView, ViewGroup parent) {
        //Instance d'un layout inflater
        LayoutInflater inflater = (LayoutInflater) context
            .getSystemService(Context.LAYOUT_INFLATER_SERVICE);
        //Utiliser l'inflater pour récupérer une vue à partir d'un layout
        View rowView = inflater.inflate(R.layout.rowlayout, parent, false);
        TextView textView = (TextView) rowView.findViewById(R.id.label);
        ImageView imageView = (ImageView) rowView.findViewById(R.id.icon);
        textView.setText(values[position]);
        // Change the icon for Windows and iPhone
        String s = values[position].text;
        imageView.setImageResource(values[position].icon);
        return rowView;
    }
}
```

BaseAdapter

- Implémentation de base des Adapters
- Peut être utilisé avec plusieurs éléments de groupes
 - ListView
 - Spinner
 - Gridview
- Les BaseAdapters peuvent gérer différents types de liste
 - List, Array etc...

BaseAdapter

```
public class ExampleAdapter extends BaseAdapter {
    public WalkItemsAdapter(BaseActivity context, List<MyData> data) {
    }

    @Override
    public int getCount() {
        return data.size();
    }

    @Override
    public MyData getItem(int position) {
        return data.get(position);
    }

    @Override
    public View getView(int position, View convertView, ViewGroup parent) {
    }
}
```

Élément de groupes - Evénements

► Un élément est sélectionné :

```
listView.setOnClickListener(new  
AdapterView.OnItemClickListener() {  
    @Override  
    public void onItemClick(AdapterView<?>  
arg0, View arg1, int arg2, long arg3)  
    {  
        //Votre logique ici  
    }  
});
```

Travaux Pratiques