Développement Android

Objectifs du cours

- En remplissant ce cours, vous serez en mesure de:

- **Expliquez** ce qui est la plate-forme Android
- **Développer** des interfaces utilisateur
- **Sauvegarder** les **données**.

Plan du cours

- Plate-forme Android
- Arborescence d’un projet Android
- Activités
- Interface utilisateur
- Intention
- Persistance
- Web Services REST

Plate-forme Android

Introduction

- Apple a transformé le marché de la téléphonie avec l'iPhone :
    - Nouvelles fonctionnalités
    - Nouveaux usages
    - Nouvelles opportunités (marché de l'application)
- Difficultés pour les concurrents ...
- fin 2007: création **Open Handset Alliance**
    - coalition pour la promotion d'un nouveau système d'exploitation: **android**
    - 34 entreprises de TI à la création
    - 79 aujourd'hui

Open Handset Alliance

Présentation

- Points clés de la plate-forme Android :
  - Innovant : intègre toutes les dernières technologies de téléphone
    - Ecran tactile, gps, appareil photo, accéléromètre ...
  - Accessible :
    - Pas besoin d'acheter logiciel ou matériel spécifique pour développer
    - Java est le langage le plus largement utilisé, pas besoin d'apprendre un autre langage peu utilisé
  - Libre :
    - licence open source: tout le monde peut voir les sources et de travailler sur elle

Présentation

- Android est conçu pour les appareils mobiles dans un sens large:
  - Téléphones
  - Tablettes
  - téléviseurs
  - Écouteurs
  - Micro-ondes
  - …

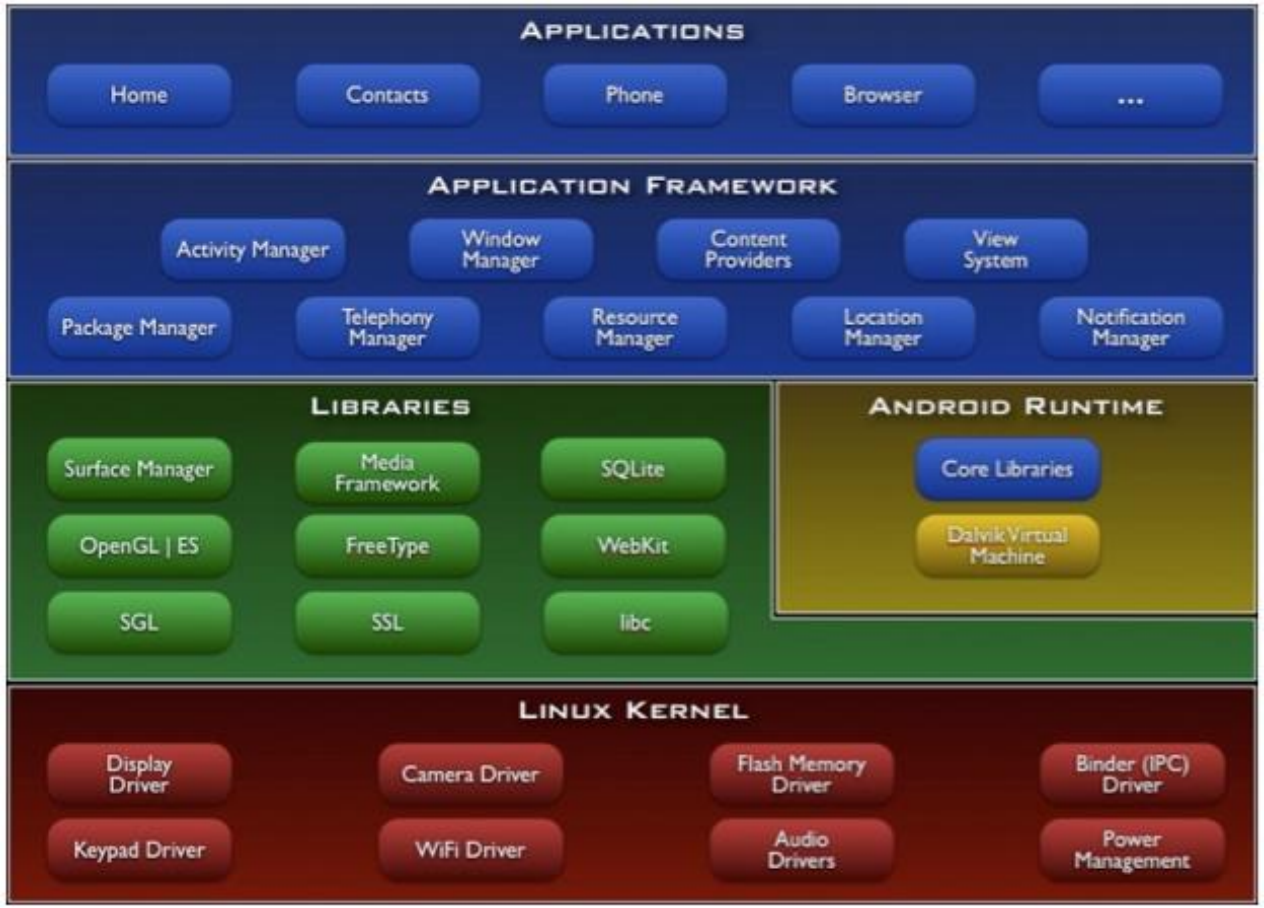
Composants de la Plate-forme

Android est composé de différentes couches:

Applications

Bibliothèques pour l'interface utilisateur, multimédia, persévérance, etc ...

Noyau Linux



Framework applicatif avec de nombreuses fonctionnalités

Machine virtuelle java spécifique appelée **dalvik**

Versions

Septembre 2008

Android 1.0

Février 2009

Android 1.1

Septembre 2009

1.6 : Donut

Mai 2010

2.2 : Froyo

2008 2009 2010 2011

Avril 2009

1.5 : Cupcake

Janvier 2010

2.1 : Eclair

Décembre 2010

2.3 : Gingerbread

Versions

Février 2011

3.0 : Honeycomb

Juillet 2011

Android 3.2

Août 2012

4.1 : Jelly Bean

2011

2012

2013

Mai 2011

Android 3.1

Novembre 2011

4.0 : Ice Cream Sandwich

Septembre 2013

4.4 : KitKat

Lollipop

5.0 : Novembre 2014

5.1 : Mai 2015

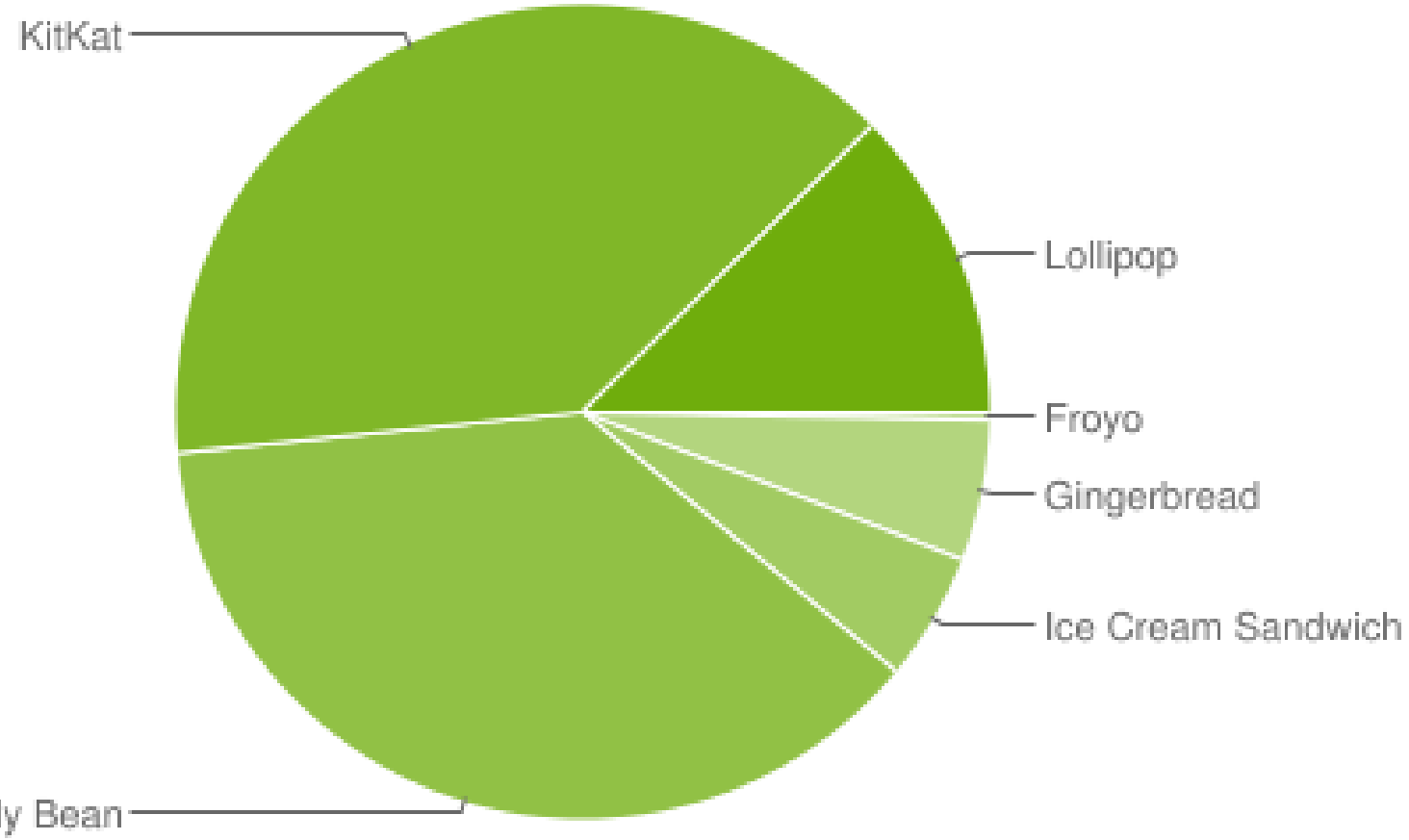
5000 nouvelles API

ART

La machine virtuelle Dalvik a été officiellement remplacée par ART (en) qui est maintenant activé par défaut

Fragmentations Android

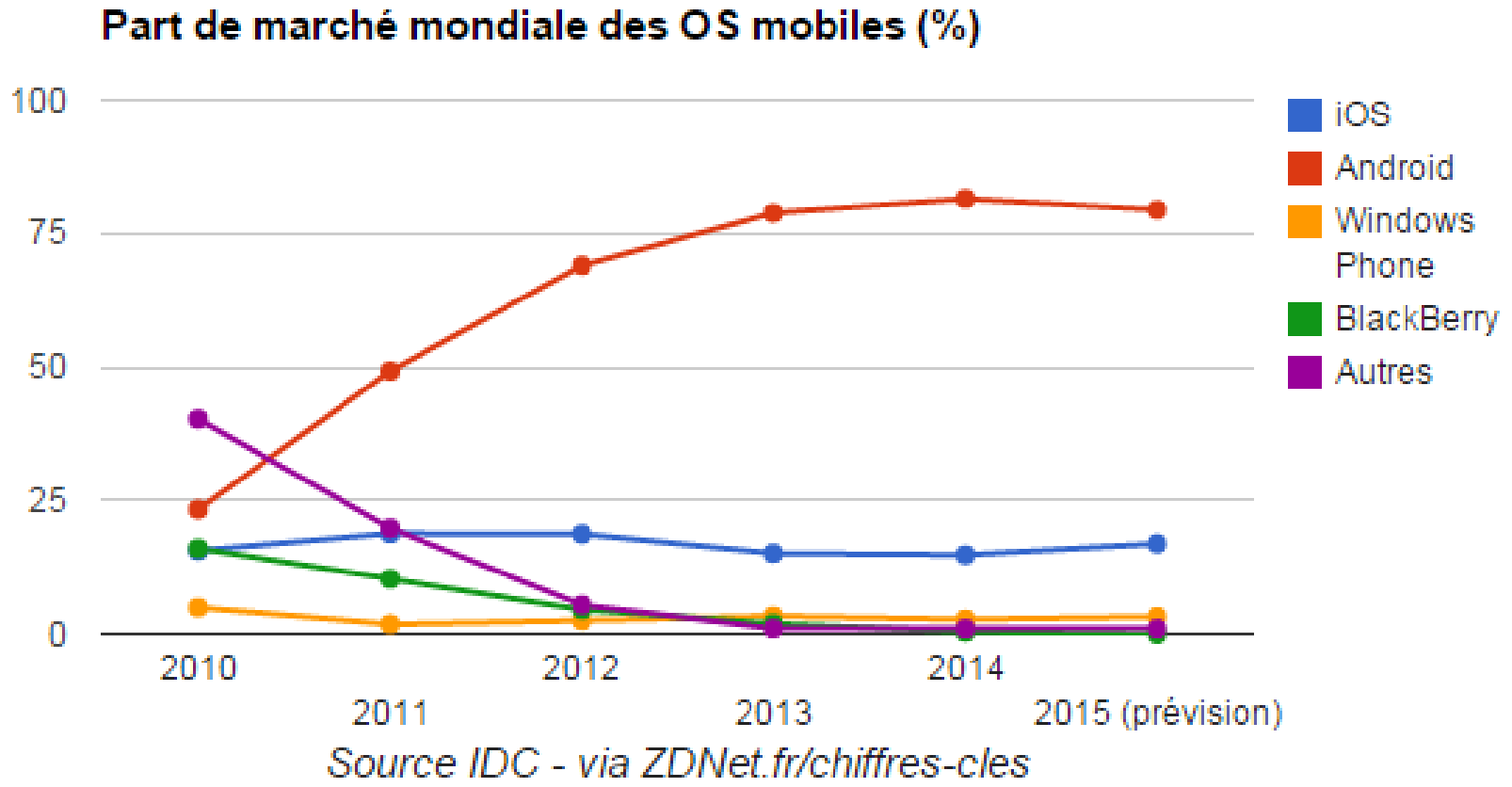
Version > = Android 4.0 ➔ 89%
Version > = Android 4.4 ➔ 51,6%



https://developer.android.com/about/dashboards/index.html?utm_source=suzunone

Android et concurrents

- Aujourd'hui, Android est le système d'exploitation n° 1 dans le monde du Mobile!

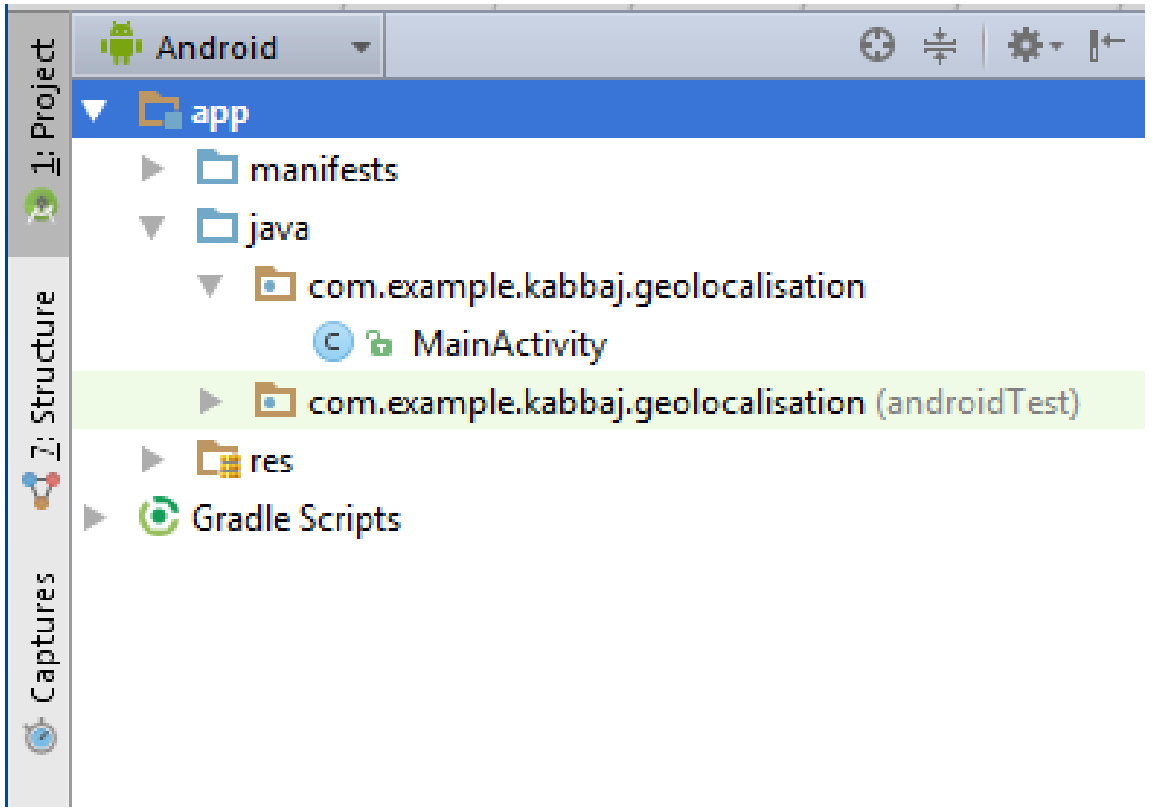


Source IDC - via ZDNet.fr/chiffres-cles

Arborescence d’un projet Android

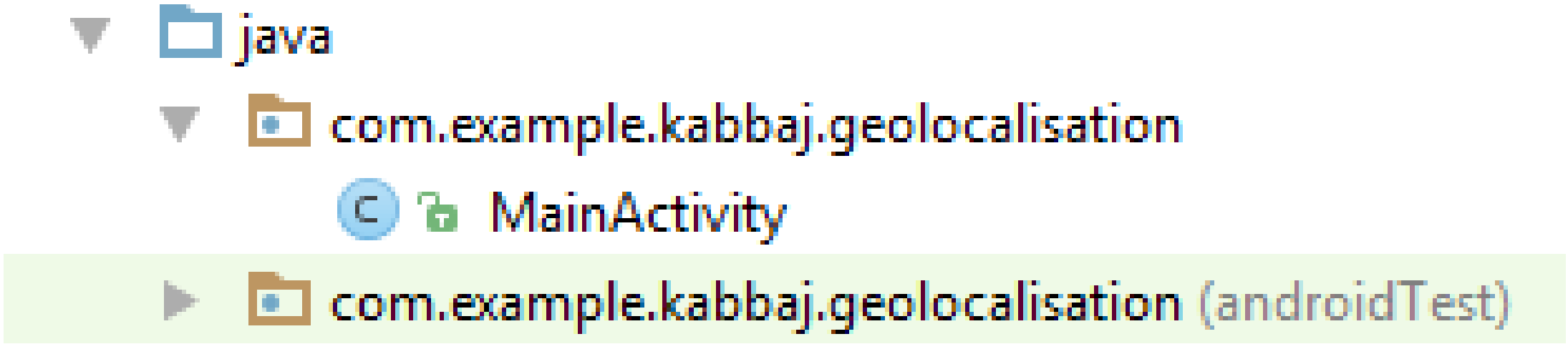
Vue d'ensemble

- Lorsque Android Studio créer un projet, il générer des nombres de dossiers et de fichiers:
  - dossier **java**
  - dossier **res**
  - dossier **manifests**
    - fichier **AndroidManifest.xml**



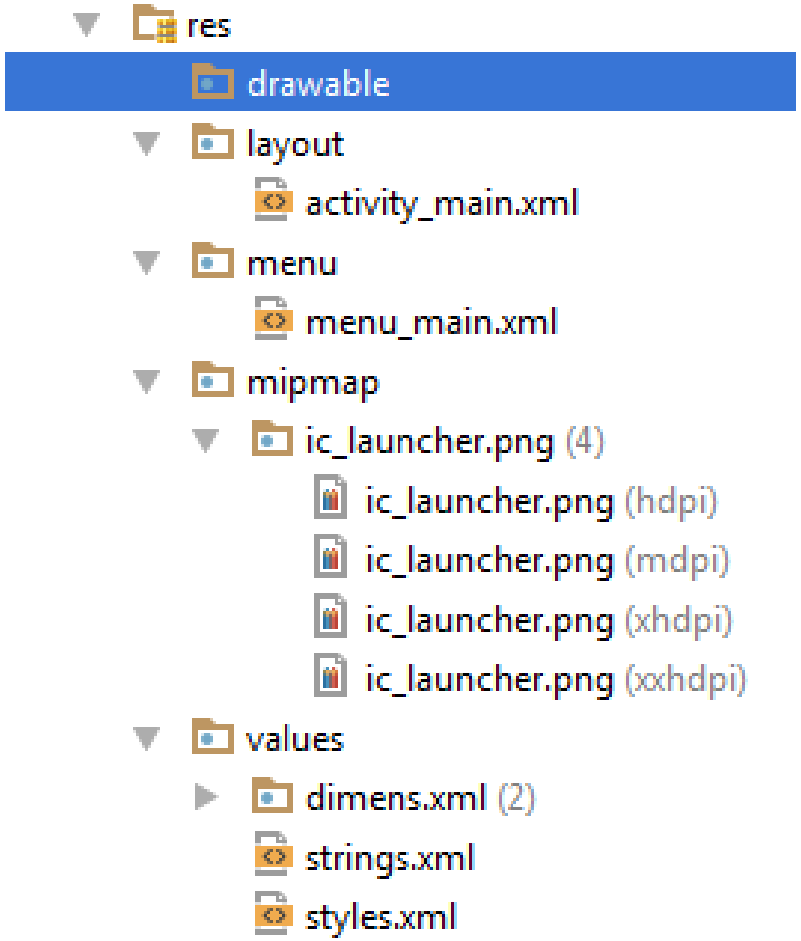
Dossier: java

- Contenir les paquets et les fichiers de code Java



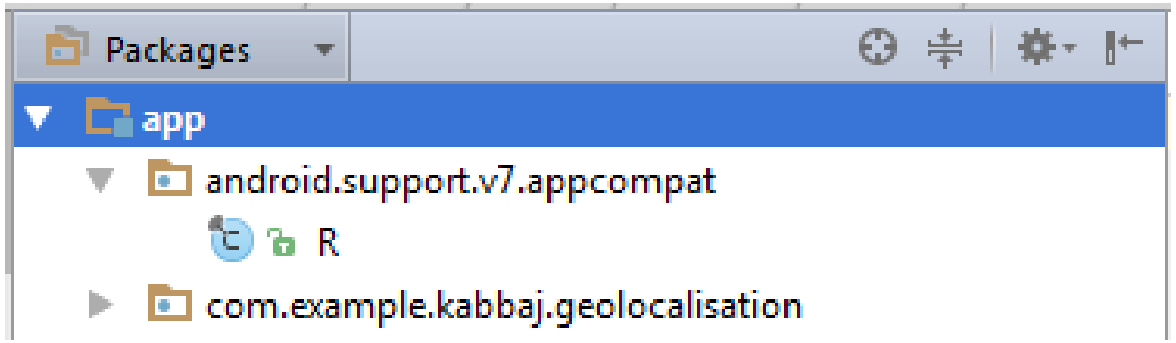
Dossier: res

- Trois types principaux:
  - **étirable** : images et animations
  - **layout:** les fichiers de format XML utilisés pour construire des interfaces utilisateur
  - **Valeurs:** les ressources de type de valeur tels que les constantes de chaîne et entier
- Contenir toutes les ressources nécessaires par l'application



Dossier: gen

- Contiennent des fichiers Java auto-généré
- Contenir la classe **R:**
  - Classe statique spécial
  - Référencer les données contenues dans les fichiers de ressources
  - Contenir statique classe interne par type de ressource




```
/.../
package android.support.v7.appcompat;

public final class R {
    public static final class anim {
        public static final int abc_fade_in = 0x7f050000;
        public static final int abc_fade_out = 0x7f050001;
        public static final int abc_grow_fade_in_from_bottom = 0x7f050002;
        public static final int abc_popup_enter = 0x7f050003;
        public static final int abc_popup_exit = 0x7f050004;
        public static final int abc_shrink_fade_out_from_bottom = 0x7f050005;
        public static final int abc_slide_in_bottom = 0x7f050006;
        public static final int abc_slide_in_top = 0x7f050007;
        public static final int abc_slide_out_bottom = 0x7f050008;
        public static final int abc_slide_out_top = 0x7f050009;
    }
    public static final class attr {
        public static final int actionBarDivider = 0x7f010064;
        public static final int actionBarItemBackground = 0x7f010065;
        public static final int actionBarPopupTheme = 0x7f01005e;
        public static final int actionBarSize = 0x7f010063;
        public static final int actionBarSplitStyle = 0x7f010060;
```

Fichier: AndroidManifest.xml

- Fichier Obligatoire dans tous les projets Android
- Contenir des informations nécessaires par Android pour exécuter l'application
    - Nom du package de l'application
    - Liste des activités, services, Récepteurs de diffusion (Broadcast Receivers), ...
    - Permissions nécessaires à l'application
    - etc ...

Fichier: AndroidManifest.xml

MainActivity.java × R.java × AndroidManifest.xml ×

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.kabbaj.geolocalisation" >

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

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

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

</manifest>
```

Outils

- Lancement manuelle de l’émulateur

  Sdk_chemain\tools\emulator.exe –avd nomAVD

- Teste de charge

  Sdk_chemain\tools\abd shell monkey – v –p nomCompletAppli cycle

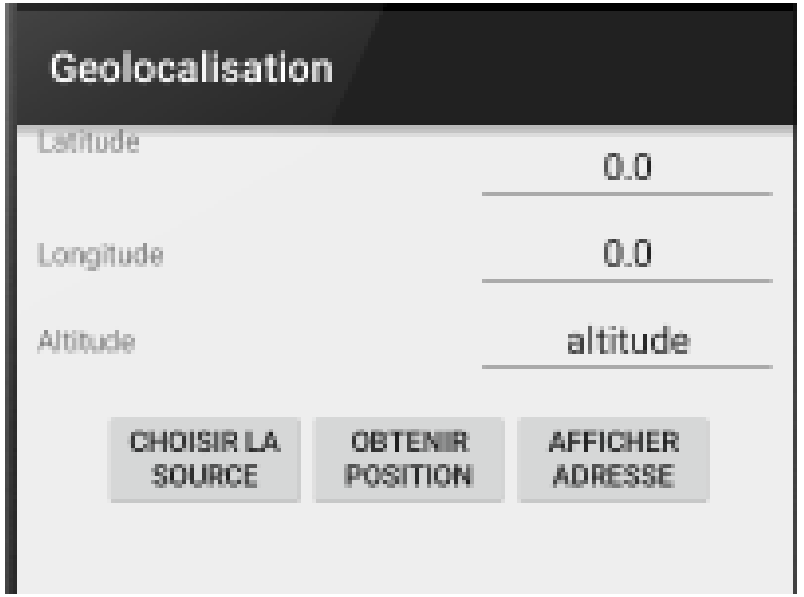
  - Exemple

    Adb shell monkey –v –p com.formation.bonjour 500

Activités

Présentation

- Une activité est une sorte d'écran composé de plusieurs vues et des contrôles:
    - Par exemple, un formulaire d’ajout de contact ou d'un Google Map personnalisé
- Autant d'activités que d’écrans des applications
- Couche présentation d'une application



Présentation

- Composé de deux parties:
  - **La logique de** l'activité:
    - Définir en Java à intérieur d'une classe étendant **android.app.Activity**
  - **L'interface** utilisateur:
    - Définir soit en Java à l'intérieur de la classe d'activité ou à l'intérieur d'un fichier XML (dans le dossier **/ res / layout** /)

Exemple

Exemple d'une simple classe Activity

MainActivity.java

```
package com.example.kabbaj.mapremiereapplication;

import ...


public class MainActivity extends Activity {

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


    @Override
    public boolean onCreateOptionsMenu(Menu menu) {
        // Inflate the menu; this adds items to the action bar if it is present.
        getMenuInflater().inflate(R.menu.menu_main, menu);
        return true;
    }

    @Override
    public boolean onOptionsItemSelected(MenuItem item) {
        // Handle action bar item clicks here. The action bar will
        // automatically handle clicks on the Home/Up button, so long
        // as you specify a parent activity in AndroidManifest.xml.
        int id = item.getItemId();

        //noinspection SimplifiableIfStatement
        if (id == R.id.action_settings) {
            return true;
        }
```

Exemple

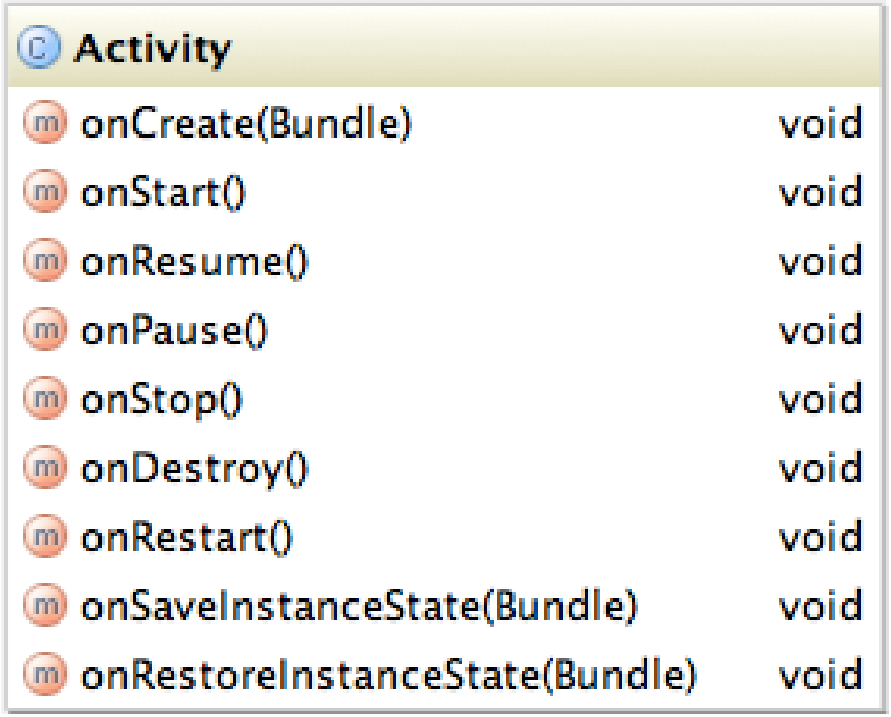
Exemple d’une simple fichier Layout

MainActivity.java | activity_main.xml

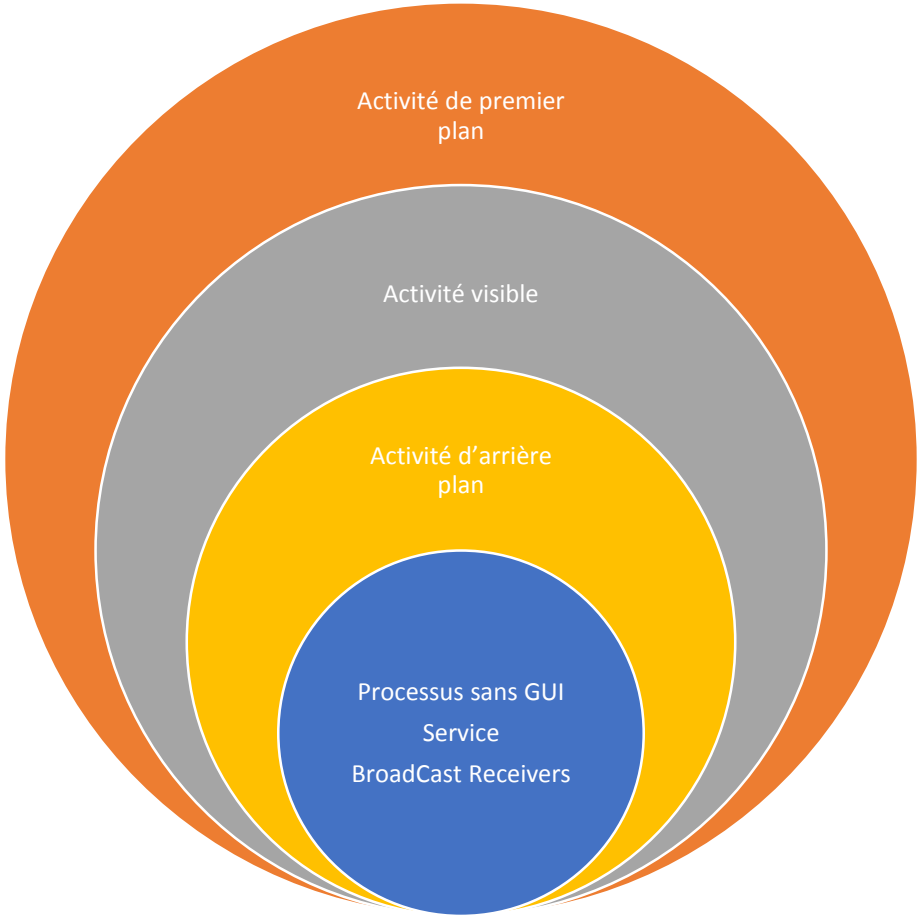
```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools" android:layout_width="match_parent"
    android:layout_height="match_parent" android:paddingLeft="16dp"
    android:paddingRight="16dp"
    android:paddingTop="16dp"
    android:paddingBottom="16dp" tools:context=".MainActivity">

    <TextView android:text="Hello world!" android:layout_width="wrap_content"
        android:layout_height="wrap_content" />

</RelativeLayout>
```

Cycle de vie d’une activité

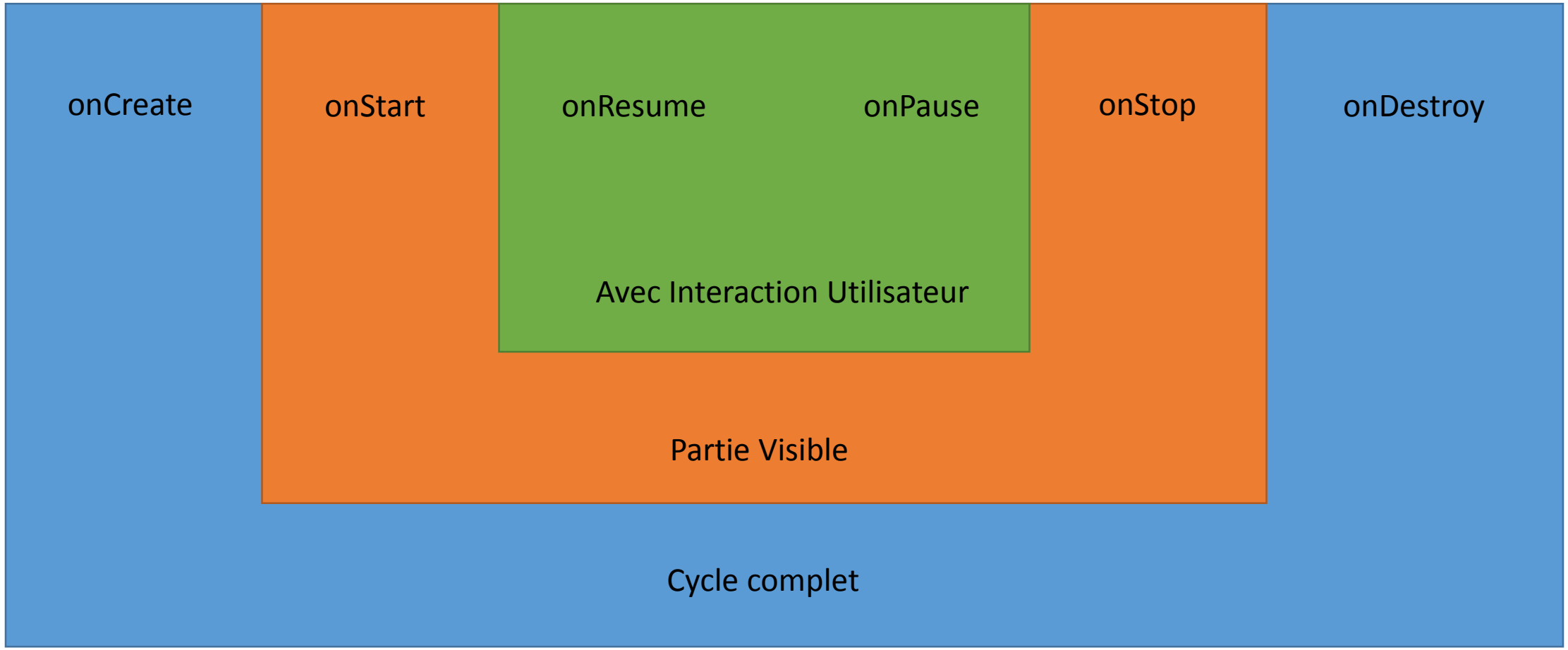
- Une activité peut avoir trois états:
  - **Actif**
    - L'activité est visible et elle a le focus de l'utilisateur (premier plan)
  - **Pause**
    - L'activité est au moins partiellement visible, mais n'a pas le focus
  - **Arrêtée**
    - L'activité n’est pas visible
- Classe Activity définit méthodes pour gérer le cycle de vie



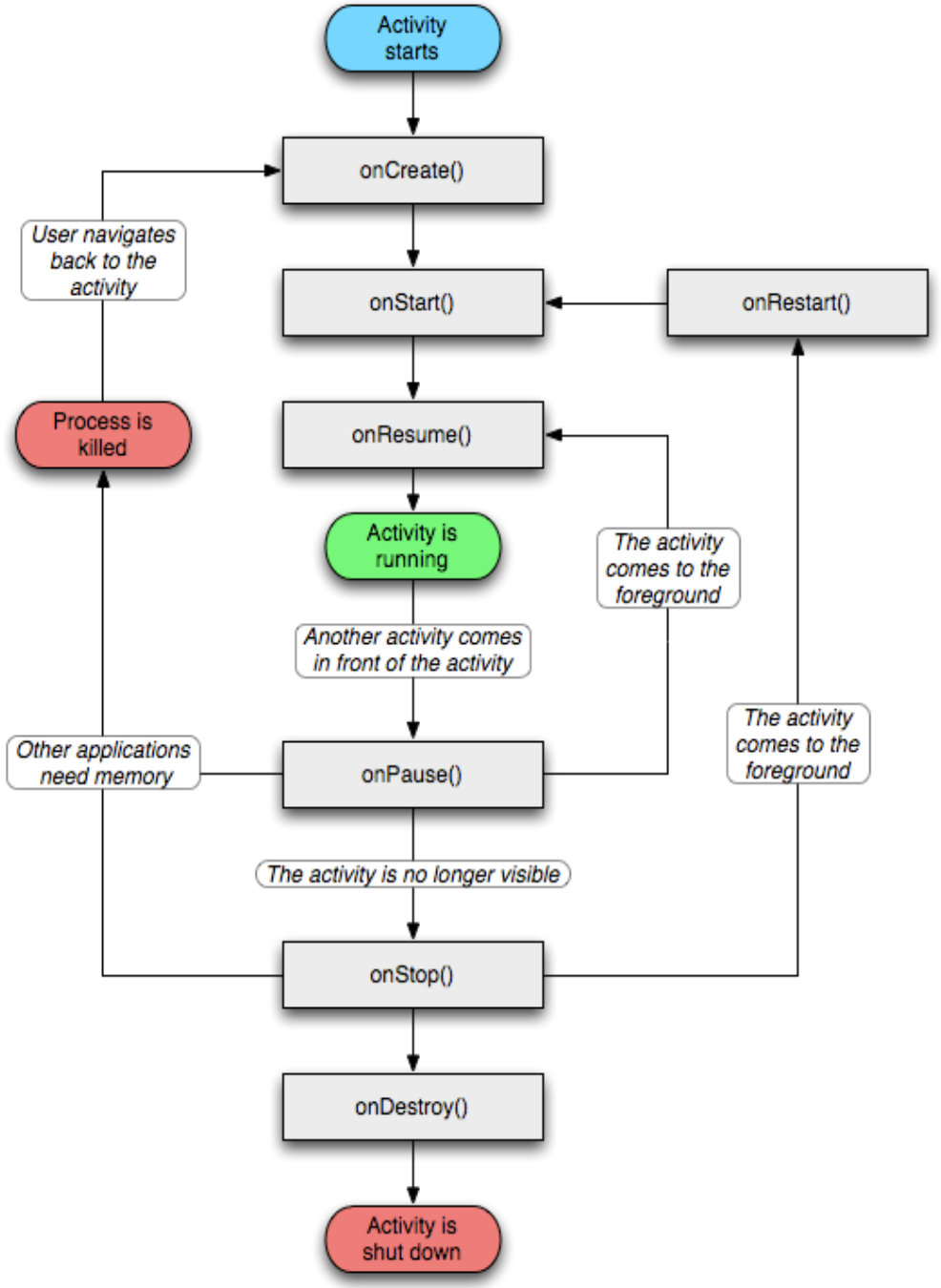
Type Activités



Type Activités



Cycle de vie


Activity starts
onCreate()
User navigates back to the activity
onStart()
onRestart()
Process is killed
onResume()
Activity is running
The activity comes to the foreground
Another activity comes in front of the activity
Other applications need memory
onPause()
The activity comes to the foreground
The activity is no longer visible
onStop()
onDestroy()
Activity is shut down

Déclarer une activité

- Pour être utilisable, une activité doit être déclarée
  - Dans le fichier **AndroidManifest.xml:**

```
<application
    android:allowBackup="true"
    android:icon="@mipmap/ic_launcher"
    android:label="PlusieursVue"
    android:theme="@style/AppTheme" >

    <activity
        android:name=".MainActivity"
        android:label="PlusieursVue" >
        <intent-filter>
            <action android:name="android.intent.action.MAIN" />

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

    <activity
        android:name=".QuestionActivity">

    </activity>

</application>
```

Déclaration d'une activité

Déclaration d'une activité principale

Exercise (1/2)

- créer un nouveau projet activitylifecycle android nommée avec:
  - nom de l'application: **"Cycle de vie d’une activité"**
  - nom du package: **"ma.fsr.lifecycle"**
  - nom de l'activité: **"MainActivity"**
- Remplacez les méthodes ci-dessous:
    - onCreate (Bundle)
    - onStart ()
    - onResume ()
    - OnPause ()
    - onStop ()
    - OnDestroy ()

Exercise (2/2)

- Dans chaque méthode, ajouter une énoncé journal comme ceci:

```
Log.d("LifeCycle", <method name>);
```

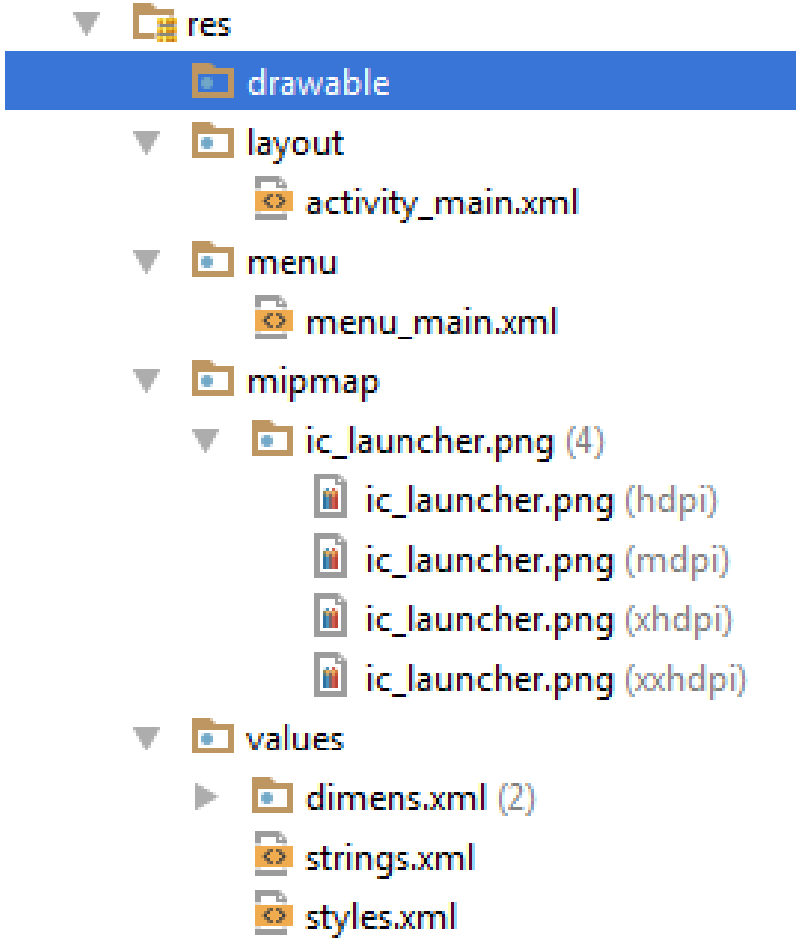
Où <nom de méthode> est le nom de la méthode dans laquelle vous êtes.

- Déployer et exécuter votre application
- Regardez la Logcat Android
- Jouez avec émulateur de voir comment le travail du cycle de l'activité Vie.
  - Afficher la Android Device Monitor
  - Essayez de simuler un appel entrant ou SMS et de regarder comment votre activité répond

Ressources

Présentation

- Android externaliser des ressources comme:
  - Images
  - Instruments à cordes
  - Description de l'interface utilisateur
  - ...
- Plus facile à gérer et à maintenir
- Figurant dans le **dossier res**



Exemple :

Bouton avec déclaration de la méthode

```xml
<Button android:id="@+id/ButtonEnvoyer"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Envoyer"
    android:onClick="envoyer"
    />
```

activity_main.xml × MainActivity.java ×

```java
package com.example.kabbaj.example1;

import ...

public class MainActivity extends ActionBarActivity {
    private EditText editText;
    private Button button;
    private String prenom;

     @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        editText = (EditText) findViewById(R.id.EditTextPrenom);
        button = (Button) findViewById(R.id.ButtonEnvoyer);
    }

    public void envoyer(View v) {
        prenom = editText.getText().toString();
        Toast.makeText(MainActivity.this, "Hello " + prenom + " !", Toast.LENGTH_LONG).show();
        ((TextView) findViewById(R.id.TextViewHello)).setText("Hello " + prenom + " !");
    }

}
```

Utilisation des ressources

- Les ressources sont accessibles dans le code grâce à la classe statique: **R**
  - Cette classe est automatiquement généré par Android Studio
  - Lorsque vous ajoutez une ressource dans le **dossier res,** Android Studio ajoute une référence à l'intérieur de la classe R

- La syntaxe pour récupérer une référence de ressource est la suivante:

```
R.resource_type.resource_name
```

Exemple

```
public final class R{
        public static final class string {
                public static final int app_name=0x7f020000;
        }
        public static final class layout {
                public static final int my_screen=0x7f030000;
        }
        ... //code omitted
}
```

```
•   // Define the layout of an activity
•   setContentView(R.layout.my_screen);

•   // Retrieve the application name
•   Resources resources = getResources();
•   String appName = resources.getString(R.string.app_name);
```

Utilisez les ressources

- Vous pouvez également utiliser les ressources à l'intérieur des ressources XML
- Très utilisé dans les présentations
- Syntaxe:

"@ [Nom_du_paquetage:] *resource_type / resource_identifier*"

- Exemple:

```
...
<TextView
android: layout_width = "fill_parent"
android: layout_height = "wrap_content"
android: text = "@ string / bonjour"
/>
...
```

Ressources système

- Android comprend déjà un certain nombre de ressources
  - Couleurs prédéfinies
  - Cordes prédéfinis
  - Images prédéfinies
- Exemples:

```
...
<TextView
android: layout_width = "fill_parent"
android: layout_height = "wrap_content"
android: textColor = "android: Couleur / darker_gray"
android: text = "@ string / bonjour"
/>
...
```

```
Chaîne cancel =
resources.getString (android.R.string.cancel);
```

Valeurs simples

- Valeurs simples sont stockés dans des fichiers XML à l'intérieur dossier **/res/values**
- Vous pouvez déclarer
  - **Instruments à cordes**
    - Vous pouvez utiliser les balises HTML **<b>, <i>** et **<u>**
  - **Couleurs**
    - Accepter le format #RGB, #ARGB, #RRGGBB et #AARRGGBB
  - **Dimensions**
    - En pixels (px), pouces (IN), millimètres (mm), points (pt), pixel indépendante de la densité (dp) ou échelle indépendant pixel (sp)
  - **Tableaux**
    - Entiers ou Chaînes de caractères.

Simple values

- Exemple :

```
<color name="Cyan">#00FFFF</color>
  <string name="first_name">Brice</string>

  <string-array name="my_array">
    <item>A string</item>
    <item>Another String</item>
  </string-array>

  <integer-array name="my_other_array">
    <item>123</item>
    <item>456</item>
  </integer-array>

  <dimen name="my_dimension">4dp</dimen>
  <dimen name="text_size">4px</dimen>
```

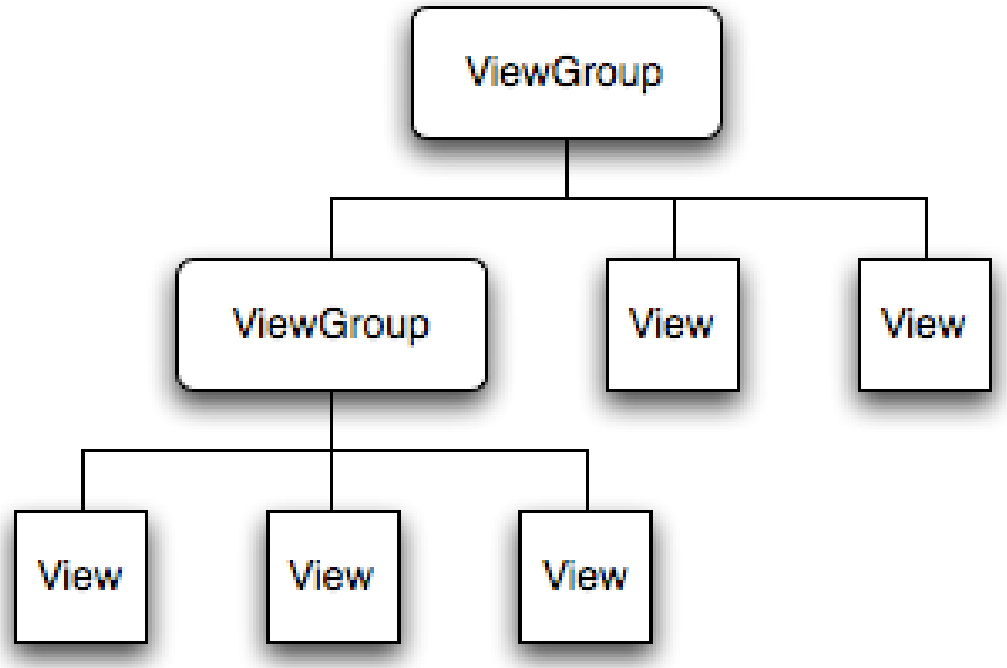
Images

- Android accepte différents formats de bitmap pour les ressources:
    - **PNG** (conseillé par la documentation)
    - **JPEG**
    - **GIF** (obsolète)
- Depuis Android 1.6, trois dossiers:
    - **étirable-hdpi:** ressources pour les écrans haute résolution
    - **étirable-MDPI:** ressources pour les écrans à résolution moyenne
    - **étirable-LDPI:** ressources pour les écrans à faible résolution

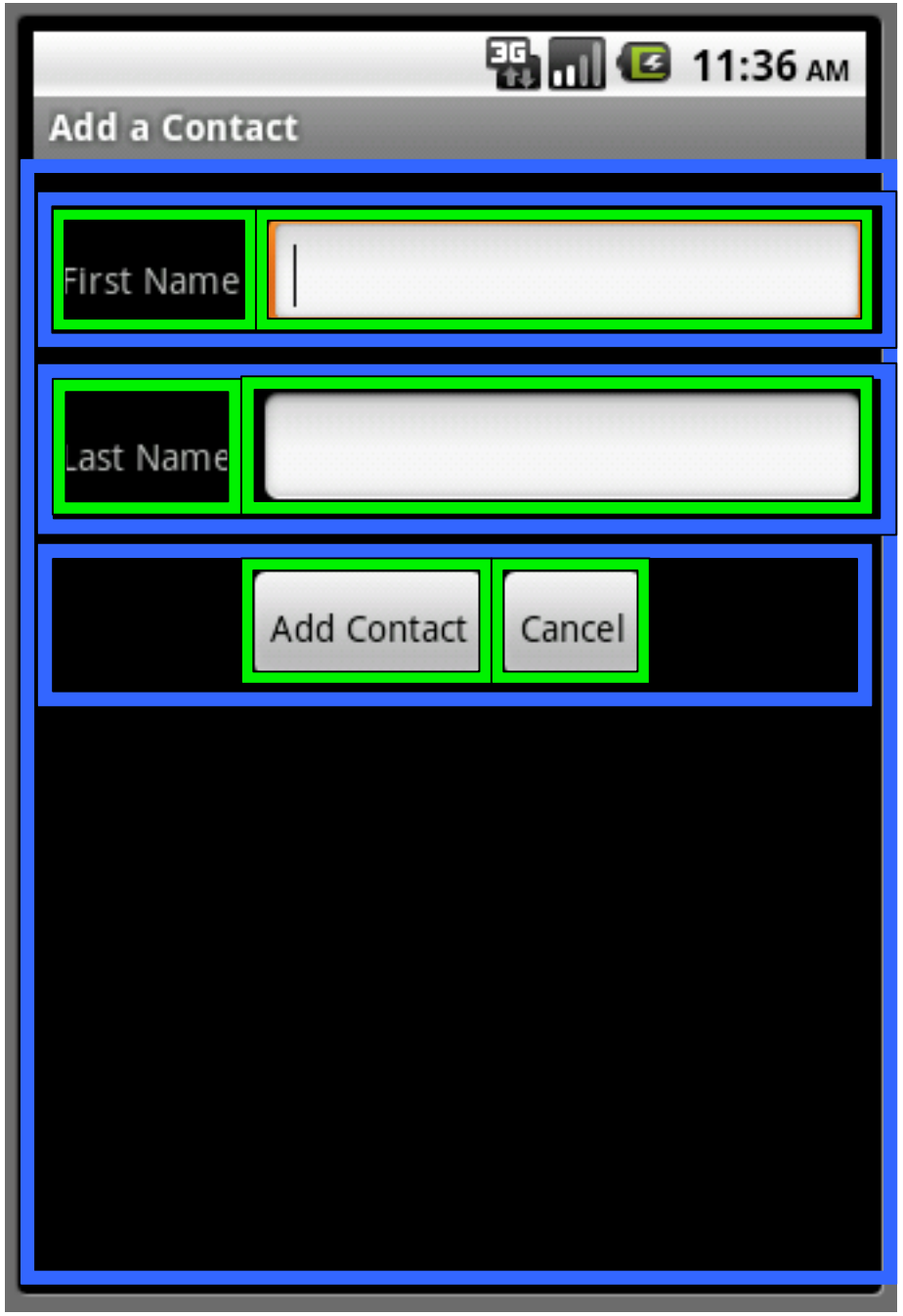
Interfaces utilisateur

Présentation

- Une interface utilisateur est un ensemble de composants graphiques comme:
    - Bouton
    - Texte
    - champ de formulaire
    - Composant composé d'autres composants ...
- Ces composants sont appelés **Views**
- Le dernier est une vue spéciale appelé **ViewGroup**



Presentation



Views

ViewGroups

Présentation

- Les interfaces utilisateur peuvent être définis:
  - En XML, à l'intérieur d'un fichier de ressources de mise en page
  - Directement dans le code d'activité

Définition XML VS Java Définition

- Utilisation du disposition XML pour définir des interfaces utilisateur est conseillé:
  - Structure d'interface séparée et la logique d'interface
  - Plus facile à entretenir
- Mais la définition de java peut également être utile:
  - Ajout de composants dynamiquement

XML Définition : Exemple

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
  android:orientation="vertical"
  android:layout_width="fill_parent"
  android:layout_height="fill_parent"
  >
  <TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="@string/first_name"
    />
  <EditText
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:id="@+id/first_name"
    />
</LinearLayout>
```

Views

GroupView

Java Définition : Exemple

```
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);

    LinearLayout layout = new LinearLayout(this);
    layout.setOrientation(LinearLayout.VERTICAL);
    layout.setLayoutParams(new LayoutParams(
              LayoutParams.FILL_PARENT,
              LayoutParams.FILL_PARENT));

    TextView textView = new TextView(this);
    textView.setText(R.string.first_name);

    EditText editText = new EditText(this);

    layout.addView(textView);
    layout.addView(editText);

    setContentView(layout);
}
```

ID Attribut

- Ids sont généralement affectés dans les fichiers Layouts XML, et sont utilisés pour récupérer des vues spécifiques à l'intérieur du code d'activité.
- Vous pouvez demander à Android Studio pour générer un avec la syntaxe particulière:

"@ + Id / *resource_identifier*"

Au lieu de:

"@ Id / *resource_identifier*"

ID Attribut

- Exemple

```
<EditText
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:id="@+id/first_name"
    />
```

```
EditText txtFirstName =
            (EditText) findViewById(R.id.first_name);
```

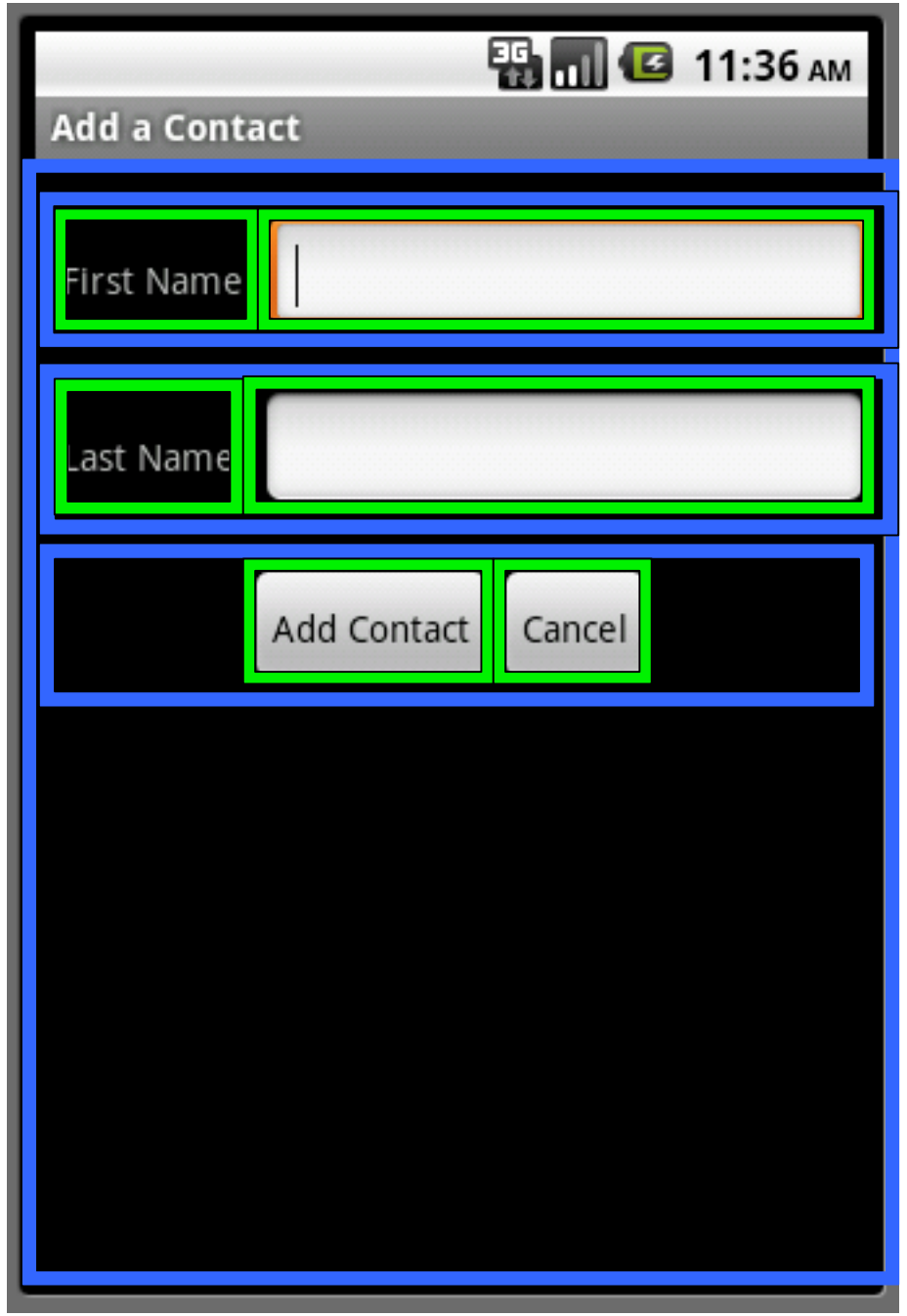
Layouts

- Layout est une ViewGroup qui nous aident à positionner nos vues
- Layout est également une vue
- Layout peut contenir d'autres dispositions
- Dispositions communes fournissent par le SDK sont:
    - **LinearLayout**
    - **RelativeLayout**
    - **FrameLayout**
    - **TableLayout**

LinearLayout

- Layout qui organise ses enfants dans une seule colonne ou une ligne unique.

- C’est le Layout le plus utilisé dans le développement Android
  - peut presque tout faire que d'autres peuvent faire



LinearLayout: Taille de composants

- La taille de ses composants peut être définir avec:
  - En XML avec **layout_width** et **layout_height** attributs

```
<TextView
    android:layout_width="wrap_content"
    android:layout_height=”10px"
    />
```

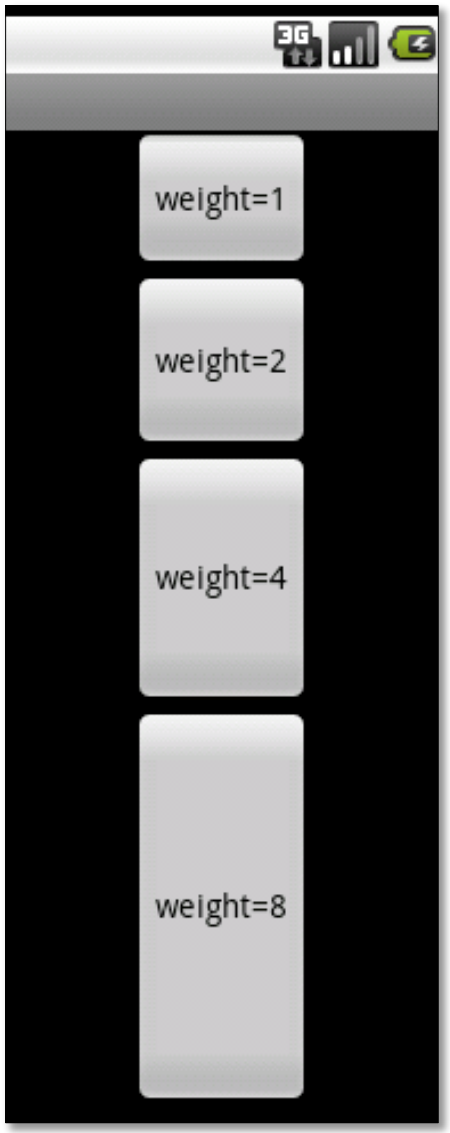
  - En Java avec un **LayoutParams** objet
  - Leurs valeurs peuvent être une dimension ou une des constantes spéciales:

FILL_PARENT, MATCH_PARENT, **WRAP_CONTENT**

LinearLayout: Poids

- Défini comment vues sur la même rangée (layout) partagent la taille de présentation
- Utile lorsque vous souhaitez plusieurs vues partagent tout l'écran
  - Exemple:

```
<Button
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_weight="2"
    android:layout_text="weight=2"
    />
```



LinearLayout: Gravity

- Spécifier comment le texte s'aligne par les axes x et/ou y de la vue lorsque le contenu est plus petite que la vue
- Doit être un ou plusieurs valeurs constantes de la classe **Gravity** (séparés par des "|") :
  - GAUCHE / DROITE
  - HAUT / BAS
  - CENTER
  - …

```
<TextView
    android:layout_width=”fill_parent"
    android:layout_height="fill_parent”
    android:gravity=”top|right"
    />
```

LinearLayout: Padding

- Par défaut, les composants sont serrés les uns des autres
- Vous pouvez définir l'espace entre elles grâce à un padding!
- Padding est définie comme l'espace entre les bords de la vue et le contenu de la vue
- Valeur en pixels
- Cinq attributs de remplissage existent:
    - **padding**
    - **paddingLeft**
    - **paddingRight**
    - **paddingTop**
    - **paddingBottom**

LinearLayout : Padding

- XML exemple :

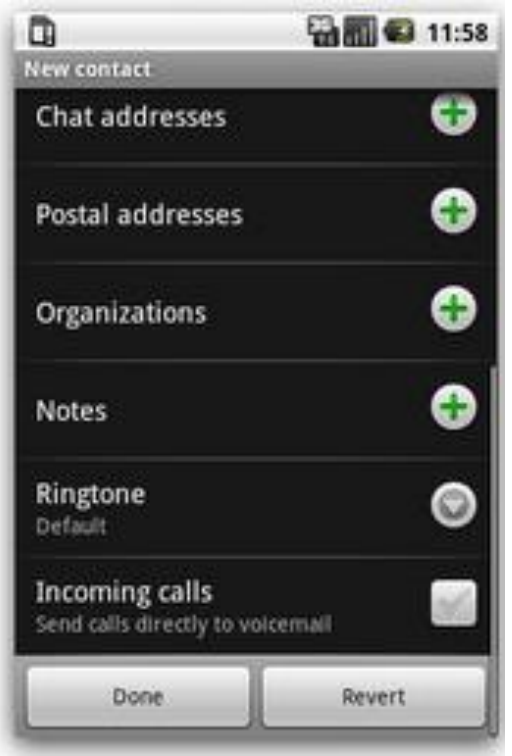
```
<TextView
    android:layout_width=”fill_parent"
    android:layout_height="fill_parent”
    android:padding=”20px"
    />
```

- Java exemple :

```
EditText txtFirstName = ... ;
// left, top, right, bottom
txtFirstName.setPadding(20, 30, 10, 20);
```

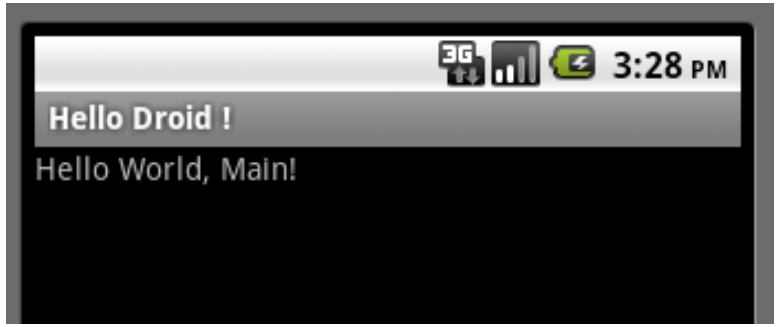
Vues

- SDK Android propose de nombreux composants communs:
  - TextView
  - EditText
  - AutoCompleteTextView
  - RadioButton
  - CheckBox
  - Spinner
  - RatingBar
  - Button
  - …



TextView

- Affiche texte à l'utilisateur
- Peut être modifiable
  - Mais désactiver par défaut




```
<TextView
    android:layout_width=”fill_parent"
    android:layout_height="fill_parent”
    android:gravity=”top|right"
    />
```

EditText

- EditText est une sous-classe de TextView
  - Modifiable par défaut!


```
<EditText
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:id="@+id/first_name"
    />
```

CheckBox

- Une case à cocher est un bouton à deux états qui peuvent être soit cochée ou décochée

```
<CheckBox
    android:id="@+id/checkbox”
    android:layout_width="wrap_content”
    android:layout_height="wrap_content”
    android:text="check it out"
    />
```

RadioButton

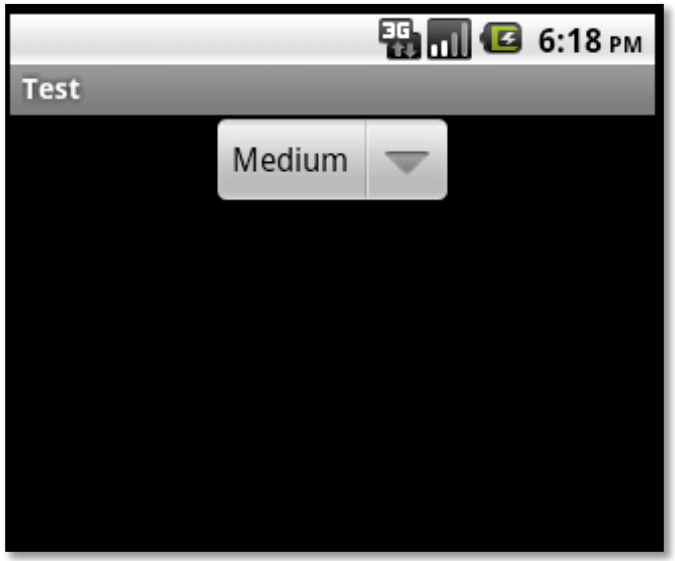
- Un bouton radio est un bouton à deux états qui peuvent être soit cochée ou décochée
  - Contrairement à case, un seul bouton par groupe de radio peut être vérifiée

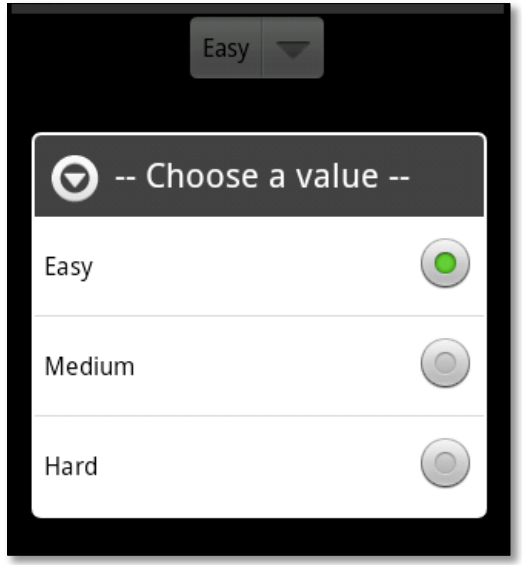
```
<RadioGroup
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:id="@+id/radio_group"
    >

    <RadioButton
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="Easy"
    />

    ...

</RadioGroup>
```

Spinner

- Un spinner est la version Android de la zone de liste déroulante

```
<Spinner
    android:id="@+id/spinner"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:prompt="@string/planet_prompt"
    />
```





Spinner : adaptateur

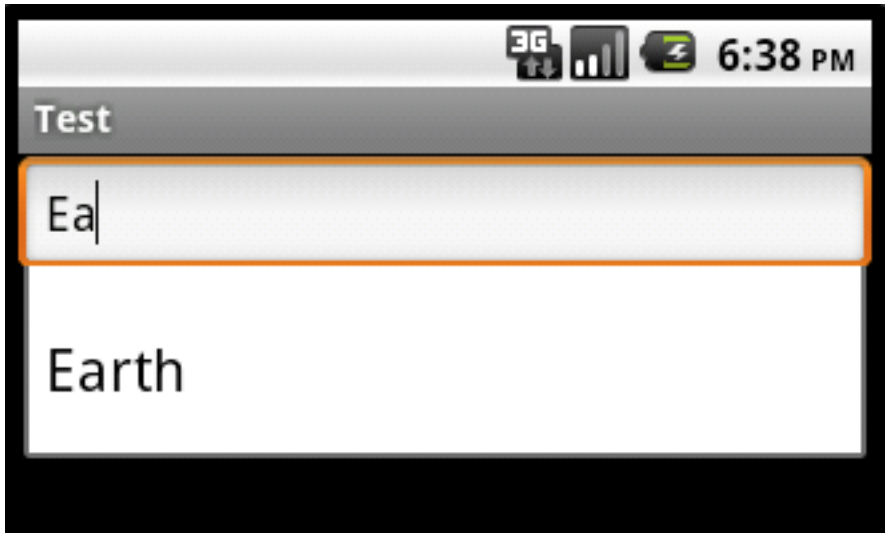
- Pour définir les options spinner, vous devez utiliser un **ListAdapter** objet

```
String[] values = { "Easy", "Medium", "Hard" };

ListAdapter adapter =
        new ArrayAdapter<String>(this,
        android.R.layout.simple_spinner_item, values);

adapter.setDropDownViewResource
        (android.R.layout.simple_spinner_dropdown_item);

Spinner spinner = (Spinner) findViewById(R.id.spinner);
spinner.setAdapter(adapter);
```

AutoCompleteTextView

- AutoCompleteTextView affiche des suggestions de complétion automatique pendant que l'utilisateur tape

```
<AutoCompleteTextView
    android:id="@+id/autocomplete_planet”
    android:layout_width=”fill_parent”
    android:layout_height="wrap_content”
    />
```



AutoCompleteTextView : Adaptateur

- Pour définir les options AutoCompleteTextView, vous devez utiliser un objet de **ListAdapter** à nouveau

```
String[] values = { "Mercury","Venus","Earth","Mars" };

ListAdapter adapter =
       new ArrayAdapter<String>(this,
       android.R.layout.simple_dropdown_item_1line, values);

AutoCompleteTextView textView = (AutoCompleteTextView)
              findViewById(R.id.autocomplete_planet);
textView.setAdapter(adapter);
```

RatingBar

- Un RatingBar est un composant qui représente une ration à l'aide des étoiles

- Deux attributs spéciaux:
  - **numStars** : Le nombre d'étoiles pour afficher
  - **stepSize** : Le nombre équivalent d'une étoile

```
<RatingBar
    android:id="@+id/rating_bar"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:numStars="5"
    android:stepSize="1.0"
    />
```

Button

- Représente un Bouton
- Boutons peuvent être pressés, ou cliqués par l'utilisateur pour effectuer une action

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

ImageButton

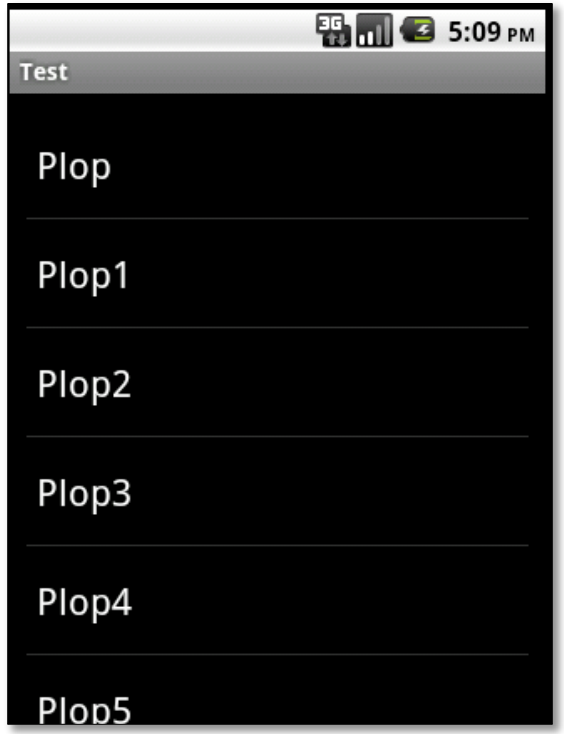
- Représente un bouton mais avec une image à l'intérieur au lieu de texte

```
<ImageButton
    android:id="@+id/my_button”
    android:layout_width=”wrap_content”
    android:layout_height="wrap_content”
    android:src="@drawable/logo_google"
    />
```

ListView

- C’est une vue qui affiche les éléments dans une liste de défilement verticale

```
<ListView
    android:id="@+id/my_list_view”
    android:layout_width=”fill_parent”
    android:layout_height=”fill_parent”
    />
```



ListView : Adaptateur

- Pour remplir la liste, vous devez utiliser un objet **ListAdapter** à nouveau

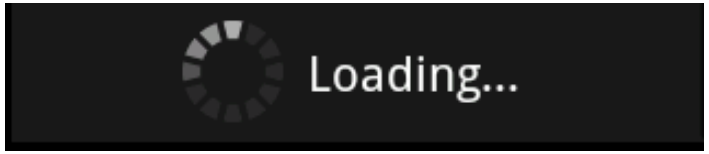
```
ListView listView =
        (ListView) findViewById(R.id.my_list_view);

Cursor cursor = new PersonDao(this).getAllPersons();

ListAdapter adapter =
       new SimpleCursorAdapter(this, android.R.layout.simple_list_item_1,
       cursor, new String[] { "name" }, new int[] { android.R.id.text1 });

listView.setAdapter(adapter);
```

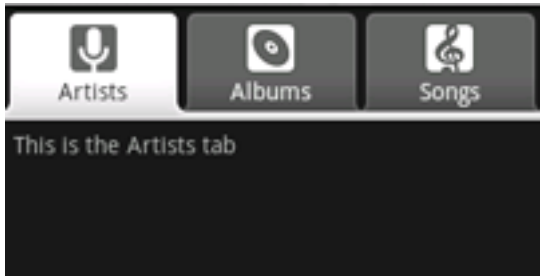
En savoir plus sur ListAdapter ...

- Pont entre un composant et les données qui sauvegarde la liste
- Sous-classes concrètes les plus utilisés sont:
  - **ArrayAdapter**
    - Adapter à lier des tableaux d'objets ou des listes d'objets à vue
  - **SimpleCursorAdapter**
    - Adapter à lier des colonnes d'un curseur sur une vue
    - Nous allons voir plus sur curseur plus tard ...
- Constructeurs de ces classes ont un identifiant de ressource:
  - Layout à appliquer à l'élément de la vue
    - Vous pouvez utiliser l'un des proposé par le SDK
    - Vous pouvez définir votre propre Layout
- Rappelez-vous: **android.R ≠ R**

Autres vues

- Maintenant, vous comprenez le principe
- Allez voir la Documentation Android pour plus d'informations
- Vous verrez beaucoup plus de vue:
  - ImageView
  - WebView
  - GridView
  - DatePicker
  - Horloge digitale
  - Barre de progression
  - ToggleButton
  - VideoView
  - ...





Événements

- Avec Android, toutes les actions de l'utilisateur sont des événements
  - Cliquez
  - Clic long
  - Key pressée
  - Élément sélectionné
  - ...
- Vous pouvez lier les comportements à cet événement
- Le mécanisme d'interception basé sur la notion d'écoute
  - Comme avec Swing!

Evénement Click

- Pour ajouter un écouteur à un événement de clic sur une vue:
    - **setOnClickListener (View.OnClickListener)**
- OnClickListener est une interface interne de la classe View
- Vous avez trois possibilités:
    - Faites votre activité une implémentation de l'interface
    - Créer une nouvelle classe implémente l'interface
    - Créer une classe anonyme

Click Event

- 1er solution

```
public class MyActivity extends Activity
        implements View.OnClickListener {

  protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);

    Button button = (Button) findViewById(R.id.my_button);
    button.setOnClickListener(this);
  }

  public void onClick(View view) {
    // Display a notification popup during 1 second.
    Toast.makeText(this, "Button clicked !", 1000).show();
  }

}
```

Click Event

- 2ème solution :

```
public class MyActivity extends Activity {

  protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);

    Button button = (Button) findViewById(R.id.my_button);
    button.setOnClickListener(new ButtonClickListener());
  }
}
```

```
public class ButtonClickListener
       implements View.OnClickListener {

  public void onClick(View view) {
    // Display a notification popup during 1 second.
    Toast.makeText(this, "Button clicked !", 1000).show();
  }
}
```

Click Event

- 3ème solution :

```
public class MyActivity extends Activity {

  protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);

    Button button = (Button) findViewById(R.id.my_button);

    button.setOnClickListener(new View.OnClickListener() {

      public void onClick(View view) {
        // Display a notification popup during 1 second.
        Toast.makeText(MyActivity.this, "Clicked!", 1000)
             .show();
      }
    });
  }
}
```

Autres événements

- Tous les événements sont basés sur le même principe
- Certains ont une valeur de retour si l'événement a été consommée
    - Si cela est vrai, l'événement ne se déclenche pas d'autres auditeurs

```
EditText editText = (EditText) findViewById(R.id.my_text);

editText.setOnTouchListener(new View.OnTouchListener() {

   public boolean onTouch(View view, MotionEvent e) {
      Toast.makeText(MyActivity.this, "Touch!", 1000)
                              .show();
      return true;
      // True means the listener has consumed the event.
   }
});
```

Intent

Présentation

- L'intention est une description abstraite d'une opération à effectuer
- Nous pouvons l'utiliser pour:
    - Lancer une activité
    - Communiquer avec des composants tels que:
        - Services
        - Récepteurs de diffusion (Broadcast Receivers)

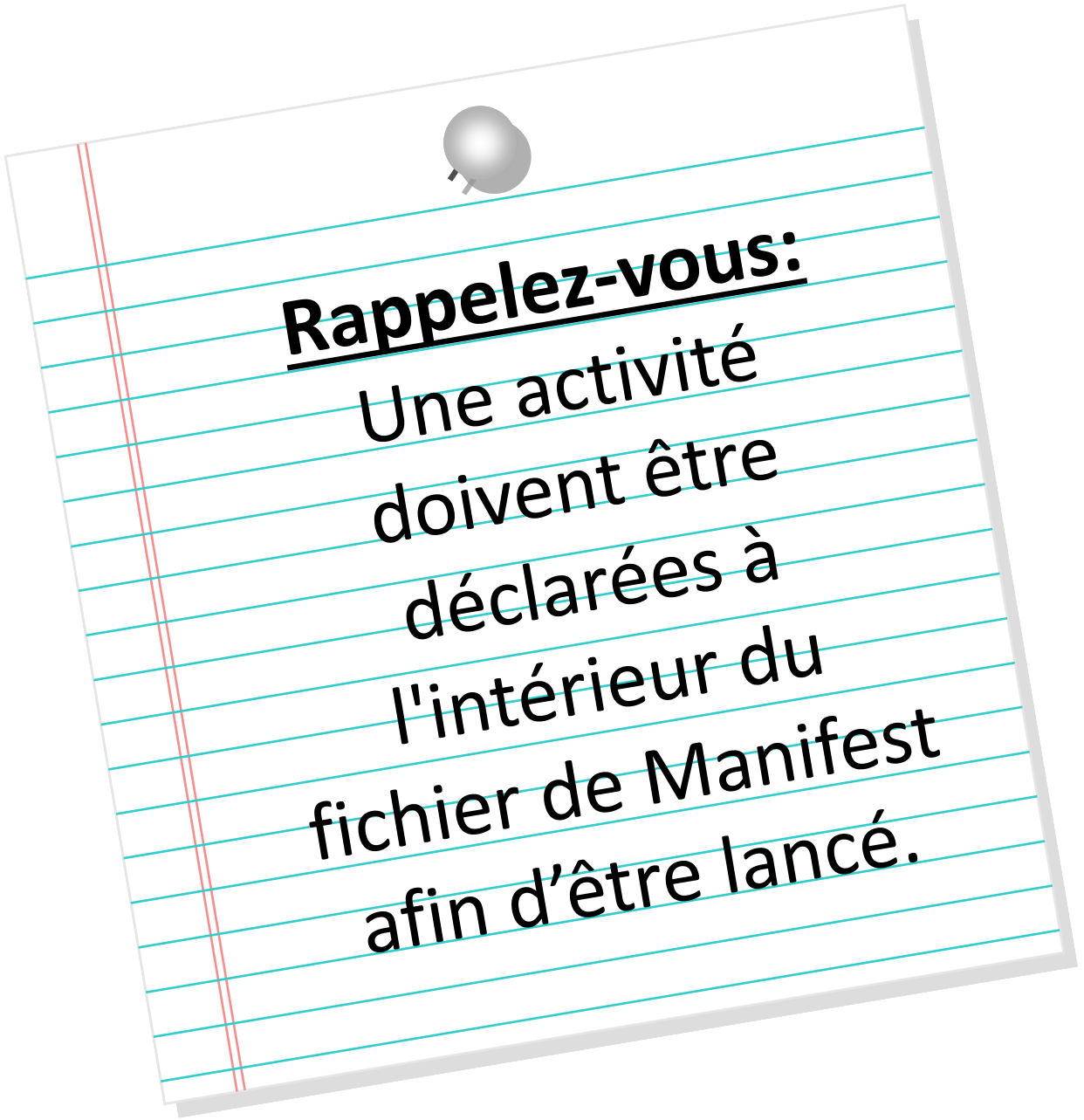
Lancer une activité

- Pour lancer simplement une activité:

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

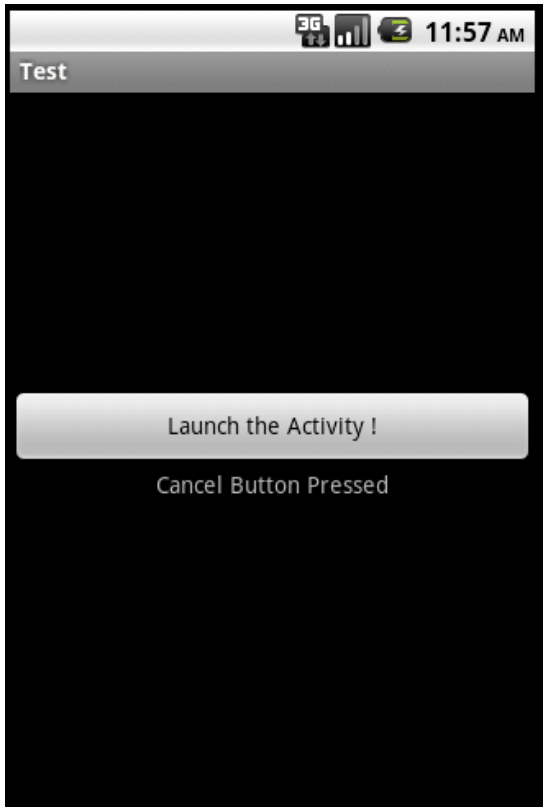
- Un des constructeurs instantanée prendre seulement ces deux paramètres:
  - Le contexte de l'intention, ici l'instance d'activité le créer
  - La classe de composant utilisé pour l'intention
- **startActivity (intention):**
  - Une méthode d'instance de la classe Activity pour démarrer une nouvelle activité avec l'intention

Note


Rappelez-vous:
Une activité
doivent être
déclarées à
l'intérieur du
fichier de Manifest
afin d’être lancé.

Lancer une activité

- Vous pouvez également démarrer une activité et d'attendre un code de résultat de celle-ci:



Lancer une activité

- Pour ce faire, il suffit d'utiliser la méthode **startActivityForResult (...)** au lieu de **startActivity**(...):

```
...

private static final int MY_ACTIVITY_CODE = 1;

...

Intent intent = new Intent(this, ActivityToLaunch.class);

// MY_ACTIVITY_CODE Constant represent an id for the
// request that will be used later identify the activity
// returning the result.
startActivityForResult(intent, MY_ACTIVITY_CODE);
```

Lancer une activité

- Dans l'activité lancée, utiliser la méthode **setResult (...)** pour retourner un code de résultat à l'activité de lancement:

```
Button submitButton = (Button) findViewById(R.id.submit);
submitButton.setOnClickListener(new View.OnClickListener() {

        public void onClick(View view) {
                setResult(RESULT_OK);
                finish();
        }

});
```

Lancer une activité

- Dans l'activité de lancement, remplacer la méthode **onActivityResult** (...):

```
protected void onActivityResult(int requestCode, int resultCode,
Intent data) {
        switch (requestCode) {
        case MY_ACTIVITY_CODE:
                TextView textView = ...
                switch (resultCode) {
                case RESULT_CANCELED :
                        textView.setText("Cancel Button Pressed");
                        break;
                case RESULT_OK :
                        textView.setText("Submit Button Pressed");
                        break;
                }
        ...
        }
}
```

Inclure des données supplémentaires

- Lorsque vous lancez une autre activité, vous avez souvent besoin de communiquer certains renseignements
- Vous pouvez utiliser les méthodes d'intention ci-dessous:
    - **void putExtra (...)**
    - **Bundle getExtras (...)**
- Les types supportés sont:
    - Primitives: byte, short, int, long, float, double, ...
    - Tableaux primitifs: int [], long [], ...
    - Chaines de Caractères
    - Objets sérialisables

Inclure des données supplémentaires

- Pour mettre des données supplémentaires:

```
Intent intent = new Intent(this, MyActivity.class);
intent.putExtra("smthg", "Hi Activity.");
startActivity(intent, MY_ACTIVITY_CODE);
```

- Pour le récupérer à l'activité lancée:

```
Bundle extras = getIntent().getExtras();
if(extras != null) {
        String message = extras.getString("smthg");
}
```

- **Intention getIntent ():**
  - Retourner l'intention qui a commencé cette activité

Inclure des données supplémentaires

- Si une activité a été lancée par la méthode **startActivityForResult (...)** :
  - Il peut envoyer des informations à l'activité appelante
    - Envoyer dans l’intention en plus du code de résultat
- Vous pouvez la récupérer à l'activité de lancement dans la méthode **onActivityResult (...)**

Inclure des données supplémentaires

- Activité appelée

```
...
setResult(RESULT_OK);
getIntent().putExtra("message", "Thank you for calling me");
finish();
...
```

- Activité appelante

```
protected void onActivityResult(int requestCode, int resultCode, Intent
data) {
        ...
        Bundle extras = data.getExtras();
        if(extras != null) {
                String message = extras.getString("message");
        }
}
```

Intent méthodes

- Appeler une deuxième activité
  - startActivityForResult(Intent activite, int requestCode)
  - startActivityForResult(intent, 1000);

  - Intent(Context, Class)
  - Intent intent = new Intent(this, Main.class);

- Passer des paramètres
  - Intent.putExtra(String cle, String valeur)
  - Intent.putExtra("madonne", "bonjour");

- Récupération des paramètres au niveau de la deuxième activité
  - Intent.getExtras();
  - this.getIntent().getExtras().getString(String cle);

- Récupération des résultat au niveau de la première activité
  - onActvityResult(int requestCode, int resultCode, Intent Data)
  - Définit resultCode par ma méthode setResult(int resultCode)

intentions implicites

- Deux formes primaires des intentions:
  - **Intentions** explicites:
    - Fournir la classe exacte pour être exécuté
  - **Intentions** implicites:
    - Composant à d'exécuter est déterminé par le système
- Nous venons de voir le premier
- Nous allons voir la seconde

Intentions implicites

- Intentions implicites sont basés sur les actions
- Android fournir de nombreuses actions indigènes
    - Mais vous pouvez créer votre propre.
- Vous avez principalement deux constructeurs pour créer une intention implicite:
    - **Intent(String action)**
    - **Intent(String action, Uri uri)**

Actions Natives

Action	Definition
ACTION_ANSWER	gérer un appel téléphonique entrant.
ACTION_CALL	effectuer un appel à quelqu'un spécifié par les données.
ACTION_DELETE	démarrer une activité pour supprimer les données indiquées de son conteneur.
ACTION_DIAL	affiche une interface utilisateur avec le numéro composé, permettant à l'utilisateur de lancer explicitement l'appel.
ACTION_EDIT	fournir un accès modifiable explicite aux données fournies.
ACTION_SEARCH	effectuez une recherche.
ACTION_SEND	livrer des données à quelqu'un d'autre par sms ou e-mail.
ACTION_SENDTO	envoyer un message à quelqu'un spécifié par les données.
ACTION_VIEW	démarrage de l'activité par défaut associé avec les données à afficher.
ACTION_WEB_SEARCH	effectuez une recherche sur le web.

Actions Natives

- Exemple :
  - Lancer Android Market :

```
Uri uri = Uri.parse("market://search?"
                +"q=pname:com.google.android.stardroid");
Intent intent = new Intent(Intent.ACTION_VIEW, uri);
startActivity(intent);
```

  - Lancer un navigateur web

```
Uri uri = Uri.parse("http://www.fsr.ac.ma");
Intent intent = new Intent(Intent.ACTION_VIEW, uri);
startActivity(intent);
```

Actions Natives

- Exemple :
  - Appeler un numéro:

```
Uri uri = Uri.parse("tel:0607080910");
Intent intent = new Intent(Intent.ACTION_CALL, uri);
startActivity(intent);
```

    - Pour faire exécuter ce code
      - Vous devez préciser que l'application a la permission d'appeler.
      - Il suffit d'ajouter un **<usage permission>** élément dans votre manifeste Android.

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

Persistence

Présentation

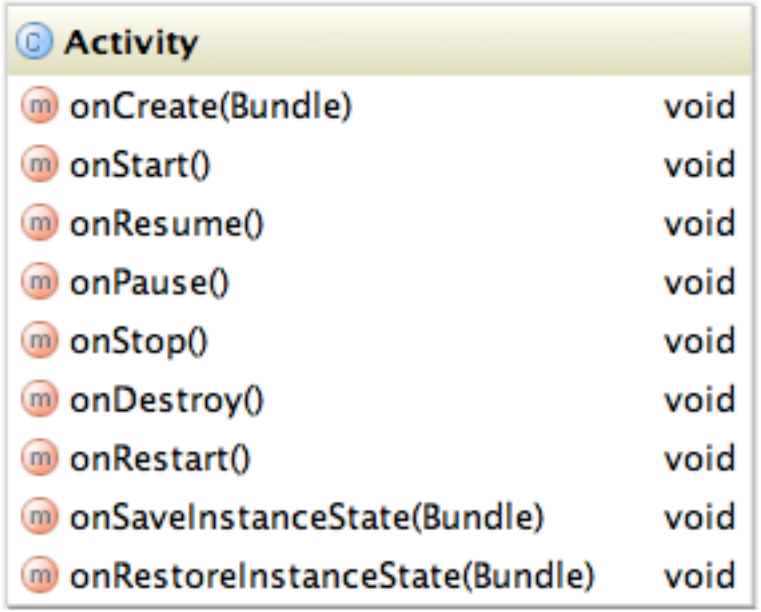
- Android fournir quatre façons de stocker les données:
    - Instance État
    - Préférences partagées
    - Bases de données SQLite
    - Fichiers
- Nous allons voir les trois premiers.

Instance État

- Vous avez vu antérieures le cycle de vie des activités
- Une activité de fond peut être déchargé si une autre à besoin de mémoire
- Comment sauvegarder l'état d'activité pour permettre à l'utilisateur de récupérer son activité comme avant?
  - Merci à instance État!
- Nous allons voir les deux méthodes d'activités pour gérer l'instance d'état

  onSaveInstanceState (...)

  onRestoreInstanceState (...)



Instance État

- **onSaveInstanceState (Bundle)**
  - Appelé pour récupérer état par exemple d'une activité avant d'être tué sorte que l'état peut être restauré en onCreate(Bundle) ou onRestoreInstanceState(Bundle) (le Bundle peuplée par cette méthode sera adoptée à la fois).

- **onRestoreInstanceState (Bundle)**
  - Cette méthode est appelée après onStart () lorsque l'activité est ré-initialisé à partir d'un état précédemment enregistré, étant donné ici à Bundle paramètre de type.

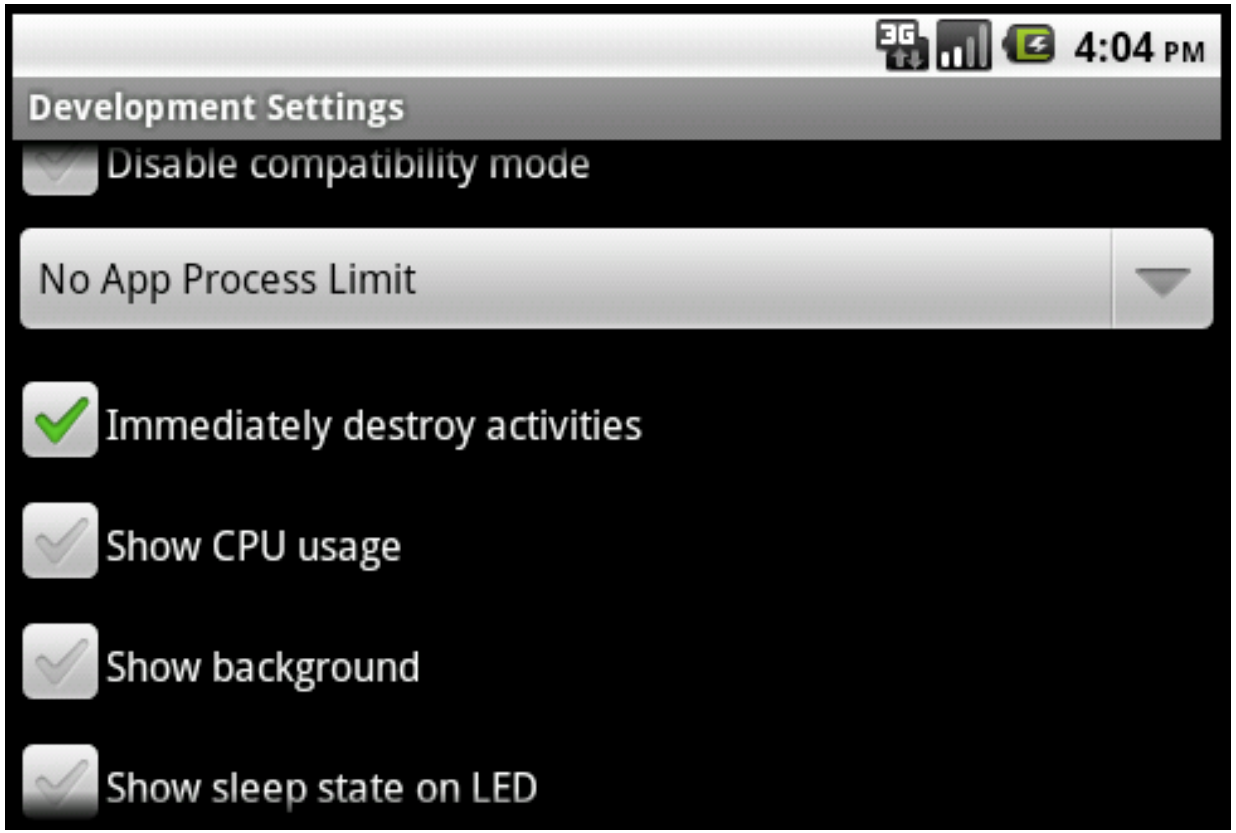
Instance État

- Par défaut, instance État sauvegarde les valeurs de tous les vues avec l'attribut **id**
- Si vous souhaitez enregistrer plus d'informations, il suffit de remplacer les deux méthodes que nous venons de voir

```
private String myInformation;
...
protected void onSaveInstanceState(Bundle outState) {
    outState.putString("anotherInformation", myInformation);
    super.onSaveInstanceState(outState);
}

protected void onRestoreInstanceState(Bundle savedInstanceState){
    super.onRestoreInstanceState(savedInstanceState);
    myInformation =
       savedInstanceState.getString("anotherInformation");
}
```

Exercice (1/2)

- Ouvrez le lanceur d'applications
    - Aller à **Dev Tools> Paramètres de** développement.



Exercice (2/2)

- Re-lancer votre application calculatrice
- Faire un calcule
- Simuler un appel entrant
- Regarder le contenu de votre liste ...

- Utiliser instance état pour corriger cela!

Préférences partagées

- Partager entre tous les composants dans une application
- Ensemble de paire clé/valeur
- Ne peut être stocker que des valeurs de type booléennes, int, long, float et String
- Permission peut être donnée:
  - **MODE_PRIVATE**
    - La valeur par défaut, le fichier créé est uniquement accessible par l'application qui l'a créé.
  - **MODE_WORD_READABLE**
    - D'autres applications peuvent lire le fichier, mais pas le modifier.
  - **MODE_WORD_WRITABLE**
    - D'autres applications peuvent modifier le fichier.

Préférences partagées

- Exemples:
  - Enregistrer les préférences partagées:

```
SharedPreferences prefs = getPreferences(Context.MODE_PRIVATE);

SharedPreferences.Editor editor = prefs.edit();
editor.putString("username", "Droid");
editor.putBoolean("isAdmin", true);
editor.commit();
```

Préférences partagées

- Exemples:
  - Récupérer les préférences partagées:

```
SharedPreferences prefs = getPreferences(Context.MODE_PRIVATE);

// If there is no value for "username", return null
String username = prefs.getString("username", null);

// If there is no value for "isAdmin", return false
boolean admin = prefs.getBoolean("isAdmin", false);

// If there is no value for "id", return zero
long id = prefs.getLong("id", 0L);
```

Bases de données SQLite

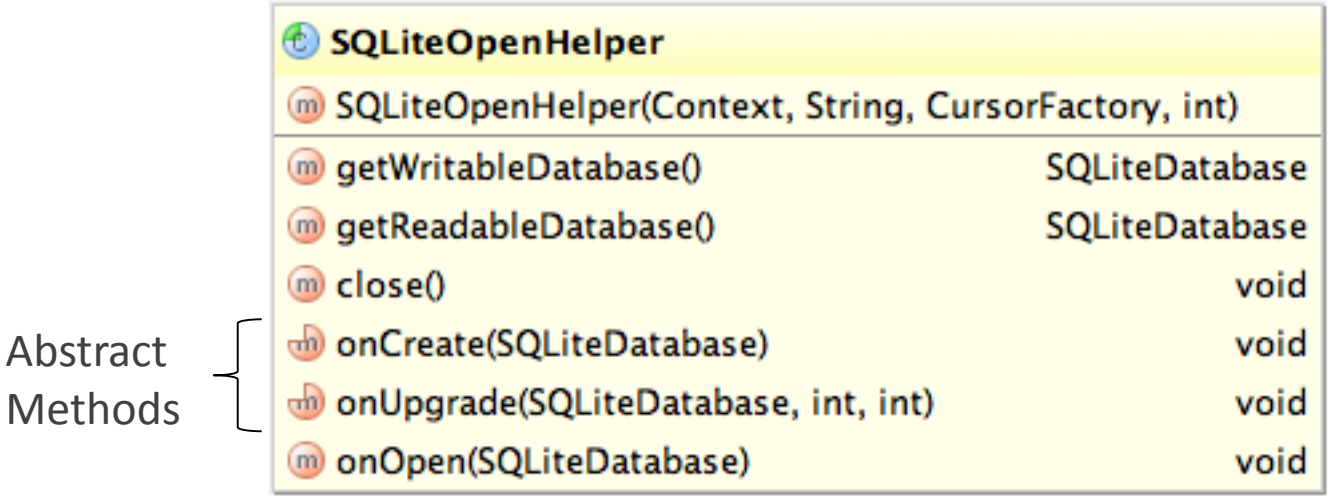
- Système de gestion de base de données relationnelle
- Utile pour stocker des données complexes
- Chaque base de données est dédiée à une seule application
- Une application peut avoir plusieurs bases de données
- Pour partager des données avec une autre application, vous pouvez utiliser un fournisseur de contenu (**Content Provider**)

Bases de données SQLite

- Ne faut pas concevoir votre base de données SQLite en tant que MySQL ou PostgreSQL.
- Les appareils mobiles ne sont pas dédiés comme étant un serveur de base de données
  - Peu d'espace de stockage
  - Peu de mémoire
- Sauvegarder seulement ce dont vous avez besoin
- Évitez les demandes fréquentes
- Conception d’une bases de données SQLite avec:
  - Une structure simple
  - Les données facilement identifiables
- Ne stocker pas les données binaires!

SQLiteOpenHelper

- Pour simplifier votre code pour créer ou mettre à jour un schéma de base de données, le SDK vous propose une classe d'aide nommée: **SQLiteOpenHelper.**
- Pour l'utiliser, créer votre classe et de l'étendre.



SQLiteOpenHelper

- Exemple :

```
public class MyOpenHelper extends SQLiteOpenHelper {

   private static final String DATABASE_NAME = "my.db";
   private static final int DATABASE_VERSION = 2;
   private static final String TABLE_NAME = "persons";
   private static final String TABLE_CREATE =
       "CREATE TABLE " + TABLE_NAME + " (" +
       "id INTEGER PRIMARY KEY AUTOINCREMENT, " +
       "name TEXT NOT NULL);";

   public MyOpenHelper(Context context) {
      super(context, DATABASE_NAME, null, DATABASE_VERSION);
   }

...
```

SQLiteOpenHelper

- Exemple :

```
...
   public void onCreate(SQLiteDatabase db) {
       db.execSQL(TABLE_CREATE);
   }

   public void onUpgrade(SQLiteDatabase db, int oldVersion,
int newVersion) {

      Log.w("Example", "Upgrading database, this will drop"
       + "tables and recreate.");
      db.execSQL("DROP TABLE IF EXISTS " + TABLE_NAME);
      onCreate(db);
   }

}
```

SQLiteOpenHelper

- Cette classe fournit deux autres méthodes très utiles:
  - **SQLiteDatabase getWritableDatabase ()**
    - Retourner une instance SQLiteDatabase pour lire ou écrire dans la base de données. Lancera une exception si la base de données ne peut pas être ouvert en écriture (mauvaise permission ou disque complet).
  - **SQLiteDatabase getReadableDatabase ()**
    - Retourner une instance SQLiteDatabase avec accès en lecture seule à la base de données.
  - Les deux vont créer la base de données si elle n'existe.

SQLiteDatabase

- Expose des méthodes pour gérer une base de données SQLite
- Classe SQLiteDatabase a des méthodes pour créer, supprimer, d'exécuter des commandes SQL, et effectuer d'autres tâches de gestion de base de données commune
- Nous allons voir quelques méthodes utiles:
    - **void execSQL(...)**
    - **long insert(...)**
    - **int update(...)**
    - **int delete(...)**
    - **Cursor query(…)**

SQLiteDatabase

- **void execSQL (String sql):**
  - Exécuter une instruction SQL unique qui ne fait pas une requête
    - Par exemple, CREATE TABLE, DELETE, INSERT, etc.
  - Exemple:

```
SQLiteDatabase db = ...

db.execSQL("DROP TABLE IF EXISTS my_table");
```

SQLiteDatabase

- **long insert (String table, String nullColumnHack, ContentValues values) :**
  - Méthode pratique pour insérer une ligne dans la base de données
  - Trois paramètres:
    - **table:** la **table** pour insérer la ligne dedans
    - **nullColumnHack:**
      - SQL ne permet pas insérer une ligne complètement vide
      - Si initialValues est vide cette colonne explicitement attribuer une valeur NULL
    - **valeurs:**
      - Map contenant les valeurs de colonne de la ligne
        - Les clés devraient être les noms de colonnes
        - Les valeurs devraient être les valeurs de colonnes

SQLiteDatabase

- **long insert (String table, String nullColumnHack, ContentValues values) :**
  - Retourne l'ID de ligne de la ligne insérée
  - Exemple:

```
SQLiteDatabase db = ...

ContentValues values = new ContentValues();
values.put("name", "Cartman");

db.insert("persons", null, values);
```

SQLiteDatabase

- **int update (String table, ContentValues values, String whereClause, String[] whereArgs) :**
  - Méthode pratique pour la mise à jour des lignes dans la base de données
  - Quatre paramètres:
    - **tableau:** la table à mettre à jour dans
    - **valeurs:** une map composée des noms de colonnes et de nouvelles valeurs de colonnes
    - **whereClause:** la clause WHERE facultative à appliquer lors de la mise à jour
    - **whereArgs:** un tableau de valeurs à appliquer à la clause WHERE
  - Retourne le nombre de lignes affectées

SQLiteDatabase

- **int update (String table, ContentValues values, String whereClause, String[] whereArgs) :**
  - Exemple :

```
SQLiteDatabase db = ...

ContentValues values = new ContentValues();
values.put("name", ”John");
String[] whereArgs = { "1" };

db.update(“persons”, values, "id=?", whereArgs);
```

SQLiteDatabase

- **int delete (String table, String whereClause, String[] whereArgs) :**
  - Méthode pratique pour la suppression de lignes dans la base de données
  - Trois paramètres:
    - **tableau:** la table à supprimer
    - **whereClause:** la clause WHERE facultative à appliquer lors de la suppression
    - **whereArgs:** un tableau de la valeur à appliquer à la clause WHERE
  - Retourne le nombre de lignes affectées

SQLiteDatabase

- **int delete (String table, String whereClause, String[] whereArgs) :**
  - Exemple :

```
SQLiteDatabase db = ...

String[] whereArgs = { "1" };
db.delete("persons", "id=?", whereArgs);
```

SQLiteDatabase

- **Cursor query(String table, String[] columns, String selection, String[] selectionArgs, String groupBy, String having, String orderBy)**
  - Interrogez la table donnée, le retour d'un curseur sur l'ensemble de résultat
  - Sept paramètres:
    - **Table:** Le nom de la table pour compiler la requête
    - **colonnes:** Une liste de colonnes à retourner
    - **sélection:** Un filtre déclarant les lignes à retourner, formatée comme une clause SQL WHERE

SQLiteDatabase

- **Cursor query(String table, String[] columns, String selection, String[] selectionArgs, String groupBy, String having, String orderBy)**
  - Sept paramètres:
    - **selectionArgs:** Vous pouvez inclure ?s dans la sélection, qui seront remplacés par les valeurs de selectionArgs
    - **groupBy:** Un filtre déclarant comment grouper des lignes, formaté comme un clause SQL GROUP BY
    - **ayant:** Un filtre déclarer quelle rangée des groupes à inclure dans le curseur, si la ligne groupement est utilisé, formaté comme une clause HAVING SQL
    - **orderBy:** Comment commander les rangées, formaté en tant que la clause SQL ORDER BY

SQLiteDatabase

- **Cursor query(String table, String[] columns, String selection, String[] selectionArgs, String groupBy, String having, String orderBy)**

```
SQLiteDatabase db = ...

String[] columns = { ID_COLUMN, NAME_COLUMN };
String[] params = { "Cartman" };
Cursor result = db.query(TABLE_NAME, columns, "name=?",
                  params, null, null, null, "1");
```

Cursor

- Fournir un accès à l'ensemble de résultats renvoyé par une requête de base de données
- Méthodes couramment utilisées sont:
    - **getCount ():** retourne le nombre de lignes
    - **moveToFirst ():** déplace le curseur sur la première rangée
    - **MoveToNext ():** déplace le curseur à la ligne suivante
    - **isAfterLast ():** retourne vrai si la position du curseur est après la dernière ligne
    - **getColumnNames ():** retourne un tableau de chaînes tenant les noms de toutes les colonnes dans le jeu de résultats
    - **getColumnIndex (String name):** retourne l'index du nom de la colonne correspondante

Cursor

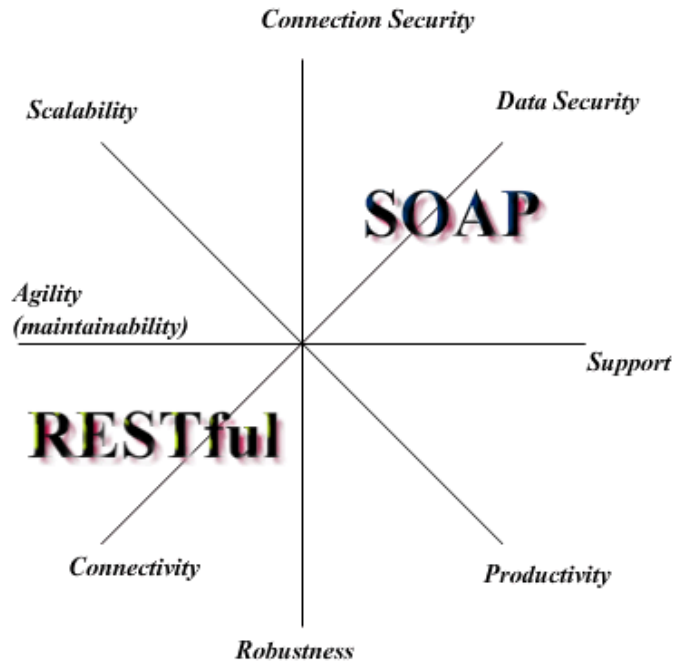
- exemple

```
String[] columns = { "id", "name"};

Cursor result = db.query("persons", columns, null,null, null, null,
null);
List<Person> persons = new ArrayList<Person>();

result.moveToFirst();
while(!result.isAfterLast()) {
   Person person = new Person();
   person.setId(result.getLong(0));
   person.setName(result.getString(1));
   persons.add(person);
   result.moveToNext();
}
result.close();

return persons;
```

REST Web Services

Présentation

- Les applications Android peuvent consommer SOAP et REST Web Services
- Les bibliothèques nécessaires pour les services Web REST sont directement disponibles dans le SDK
- Les bibliothèques pour SOAP ne le sont pas



HTTPClient

- SDK Android fournit la Bibliothèque **HttpClient** de la Fondation Apache
  - Fournir un ensemble efficace, la mise à jour, et riche en fonctionnalités mise en œuvre du côté client des normes et recommandations HTTP

HTTPClient

- Chaque requête avec HttpClient suit les étapes ci-dessus:
  - Créer une instance de la classe *HttpClient*
  - Créer une instance de la classe *HttpRequest*
  - Configurer les propriétés de la requête
  - Exécuter la demande à l'instance *de HttpClient*
  - L'analyse et le traitement de la réponse

Requête Get

```
public String sendGetRequest(String address) {
  String result = null;

  try {
    HttpClient httpClient = new DefaultHttpClient();
    HttpGet httpGet = new HttpGet();

    URI uri = new URI(address);
    httpGet.setURI(uri);

    HttpResponse response = httpClient.execute(httpGet);

    result = EntityUtils.toString(response.getEntity());
  } catch (Exception e) {
    Log.e(LOG_TAG, e.getMessage(), e);
  }

  return result;
}
```

Requête Post

```
public void sendPostRequest(String address, String entity) {
  try {
    HttpClient httpClient = new DefaultHttpClient();
    HttpGet post = new HttpPost();

    URI uri = new URI(address);
    post.setURI(uri);
    post.setEntity(new StringEntity(entity));

    httpClient.execute(post);

  } catch (Exception e) {
    Log.e(LOG_TAG, e.getMessage(), e);
  }

}
```

Entité

- Requête HTTP et réponse HTTP peuvent contenir des données (entités) dans différents formats :
    - JSON
    - XML
    - Simple Text
    - HTML
    - …
- Vous avez pour générer et / ou analyser eux!

Entité

- Pour analyser XML, SDK Android fournit des bibliothèques SAX:
    - http://www.saxproject.org/ quickstart.html
- Pour analyser JSON, SDK Android fournit des bibliothèques JSON:
    - http://json.org/java /

JSON exemple

```
public Student convertToStudent(JSONObject object) throws JSONException {
  Student student = new Student();

  student.setIdBooster(object.getLong(ID_BOOSTER_FIELD));
  student.setFirstName(object.getString(FIRST_NAME_FIELD));
  student.setLastName(object.getString(LAST_NAME_FIELD));
  String formatedDate = object.getString(BIRTH_DATE_FIELD);

  try {
    student.setBirthDate(dateFormat.parse(formatedDate));
  } catch (ParseException e) {
    Log.e(LOG_TAG, "Parse Exception!", e);
  }

  return student;
}
```

JSON exemple

```
public JSONObject convertToJson(Student student) throws JSONException {

        JSONObject jsonStudent = new JSONObject();

        jsonStudent.put(ID_BOOSTER_FIELD, student.getIdBooster());
        jsonStudent.put(FIRST_NAME_FIELD, student.getFirstName());
        jsonStudent.put(LAST_NAME_FIELD,student.getLastName());
        jsonStudent.put(BIRTH_DATE_FIELD,

        dateFormat.format(student.getBirthDate()));

        return jsonStudent;
}
```