



Développement des applications mobiles sous Android

Refs :

- Introduction au Développement d'Application Android - Amosse EDOUARD, INRIA/I3S - UNSA
- Programmation Mobile – Android – Master CCI - Bertrand Estellon Aix-Marseille Université
- Développement sous Android Jean-Francois Lalande
- www.openclassrooms.com - Frédéric Espiau
- <https://developer.android.com/guide>

Plan

2

- 1. Généralités***
- 2. Les Activités***
- 3. Les Ressources***
- 4. Les IHMs***
- 5. Les Intents***
- 6. Les SQLite***

Positionnement d'Android



L'iPhone (Apple)



Nokia (Symbian OS)



Window Mobile (Microsoft)



BlackBerry



Palm (WebOS)



Google (Android)

Généralités



Android est un système d'exploitation OPEN SOURCE pour terminaux mobiles (smartphones, PDA, 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

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 (DVM)

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 dispose de sa propre machine virtuelle

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

Outils SDK

ADB : Outil en ligne de commande permettant d'administrer un terminal (virtuel, réel) :

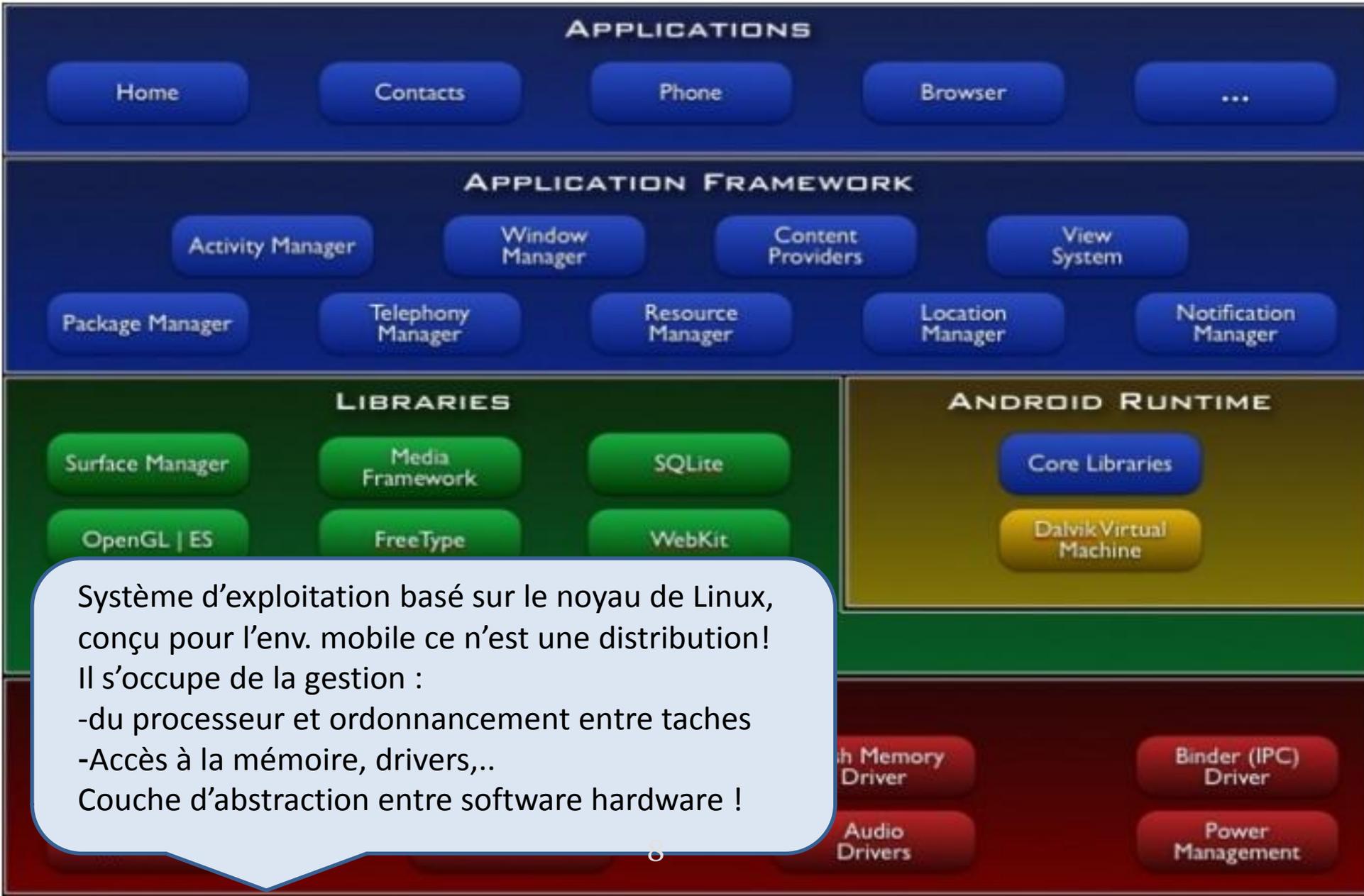
- Transférer des fichiers (push / pull)
- Installer une application (install)
- Connexion par sockets (forward)

dx : Compilateur Android qui 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

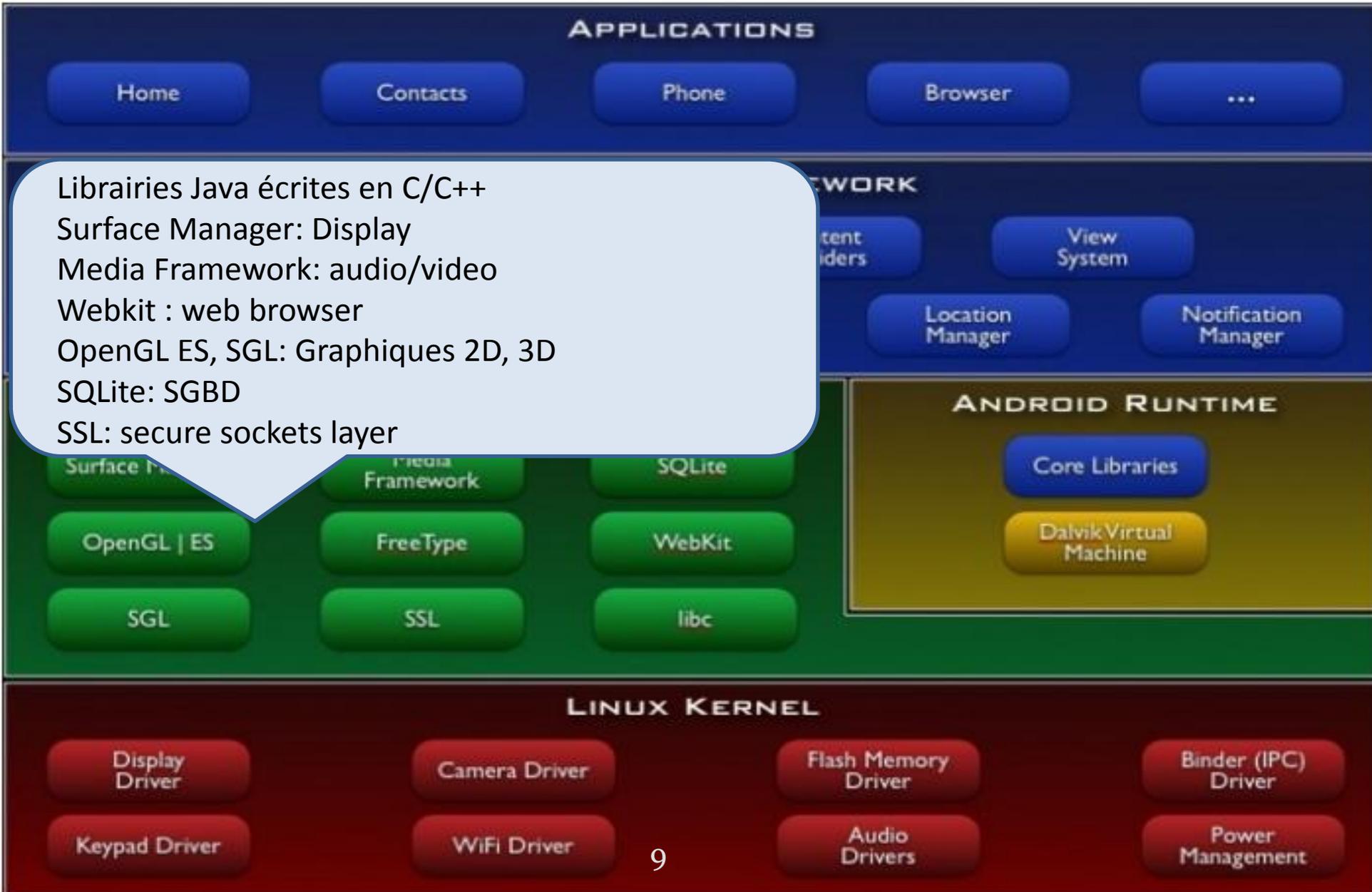
} □ DDMS / Monitor : Monitoring sur les activités du terminale

Architecture Android



Système d'exploitation basé sur le noyau de Linux, conçu pour l'env. mobile ce n'est une distribution!
Il s'occupe de la gestion :
-du processeur et ordonnancement entre taches
-Accès à la mémoire, drivers,..
Couche d'abstraction entre software hardware !

Architecture Android



Architecture Android

APPLICATIONS

Home

Contacts

APP

Activity Manager

Wi
Ma

Package Manager

Telephony
Manager

Core Libraries : des classes Java standard du sdk:
android.* , junit.* , Org.json, org.xml, ...

Mais pas Toutes les classes du sdk !

Dalvik VM : Machine virtuelle optimisé et adapté
aux contraintes des ressources système, génère
l'ex. à partir d'un bytecode spécifique

LIBRARIES

Surface Manager

Media
Framework

SQLite

OpenGL | ES

FreeType

WebKit

SGL

SSL

libc

ANDROID RUNTIME

Core Libraries

Dalvik Virtual
Machine

LINUX KERNEL

Display
Driver

Camera Driver

Flash Memory
Driver

Binder (IPC)
Driver

Keypad Driver

WiFi Driver

Audio
Drivers

Power
Management

Arc

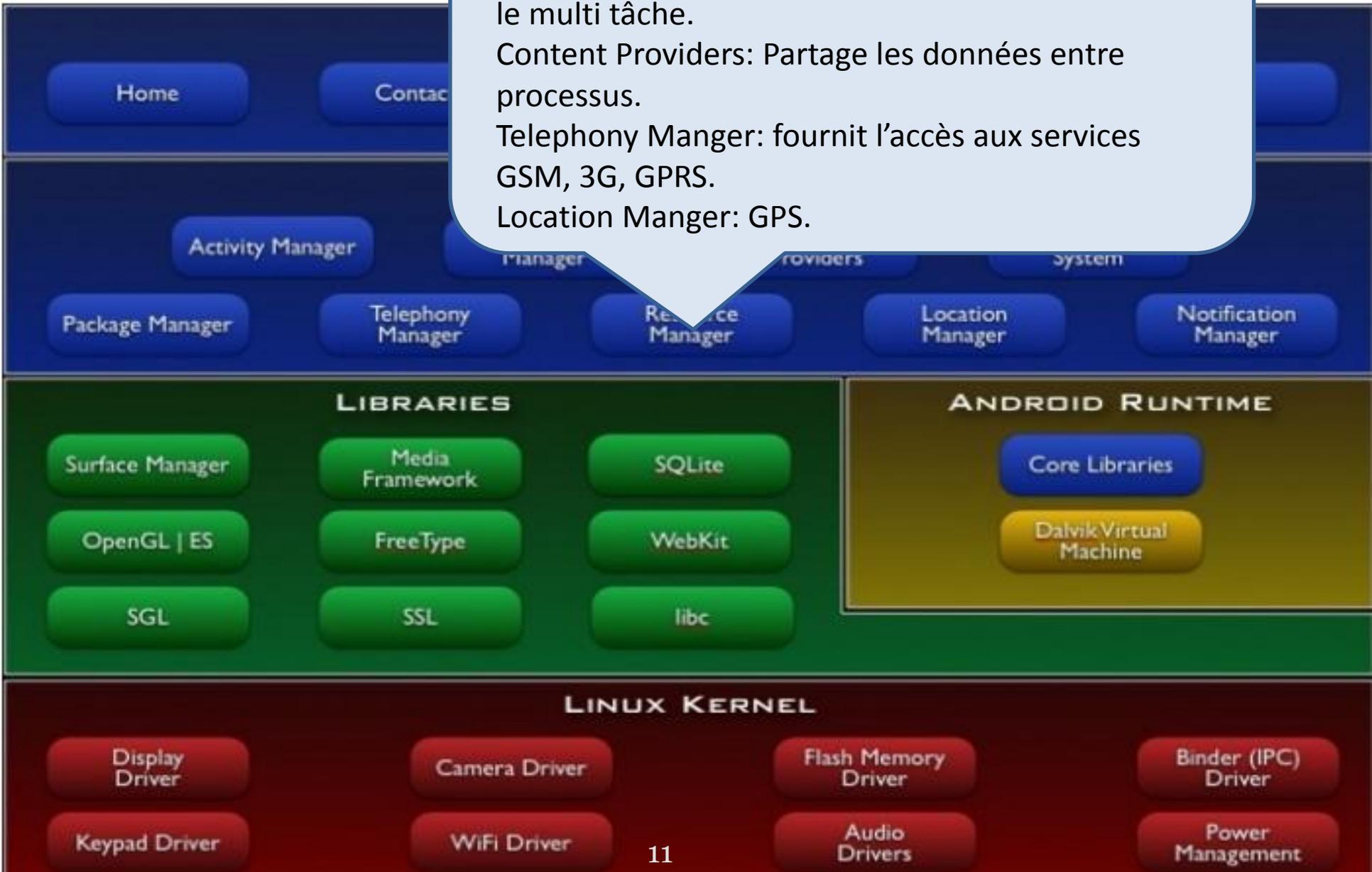
Framework Java

Activity Manger: Cycle de vie des activités. Assure le multi tâche.

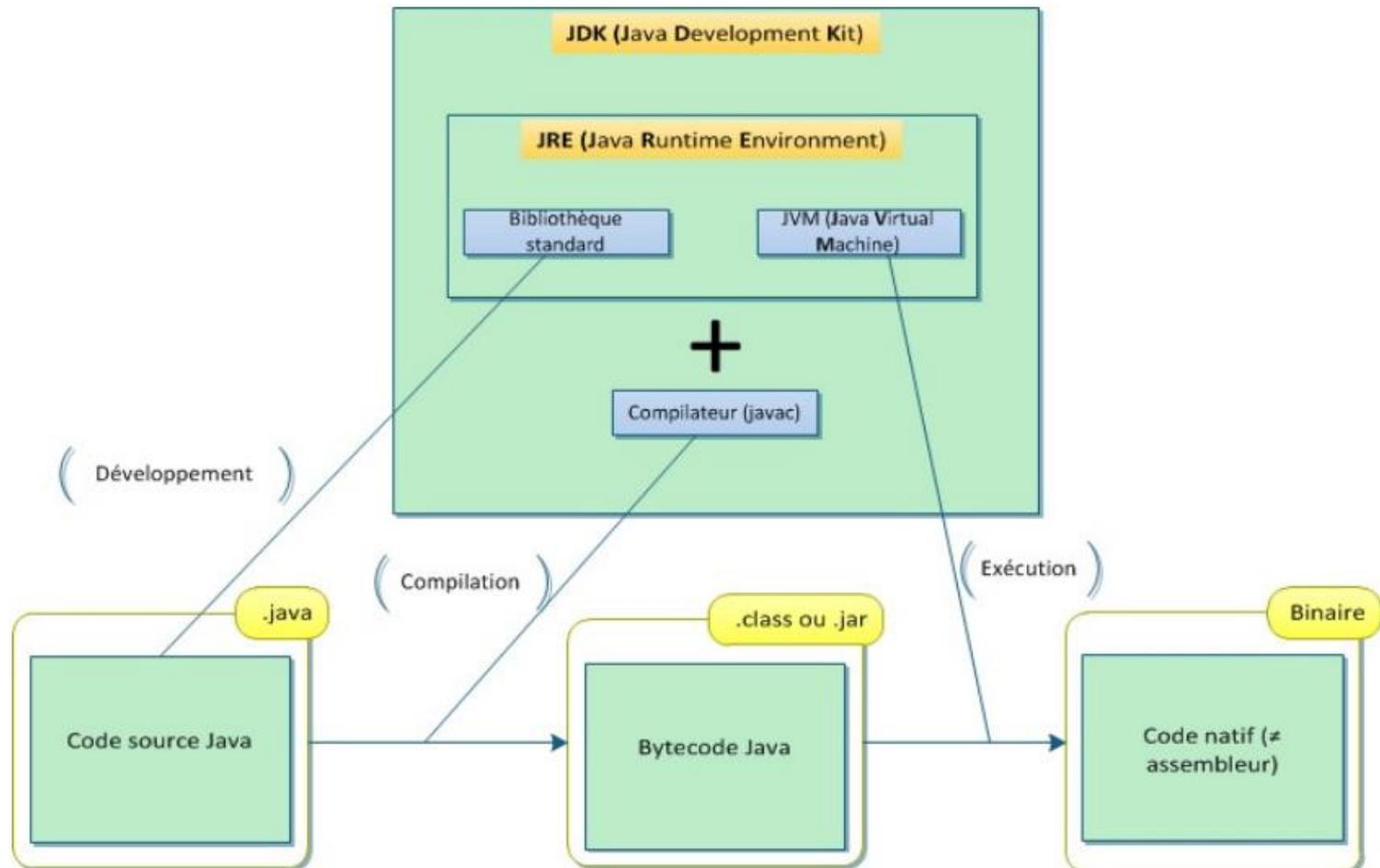
Content Providers: Partage les données entre processus.

Telephony Manger: fournit l'accès aux services GSM, 3G, GPRS.

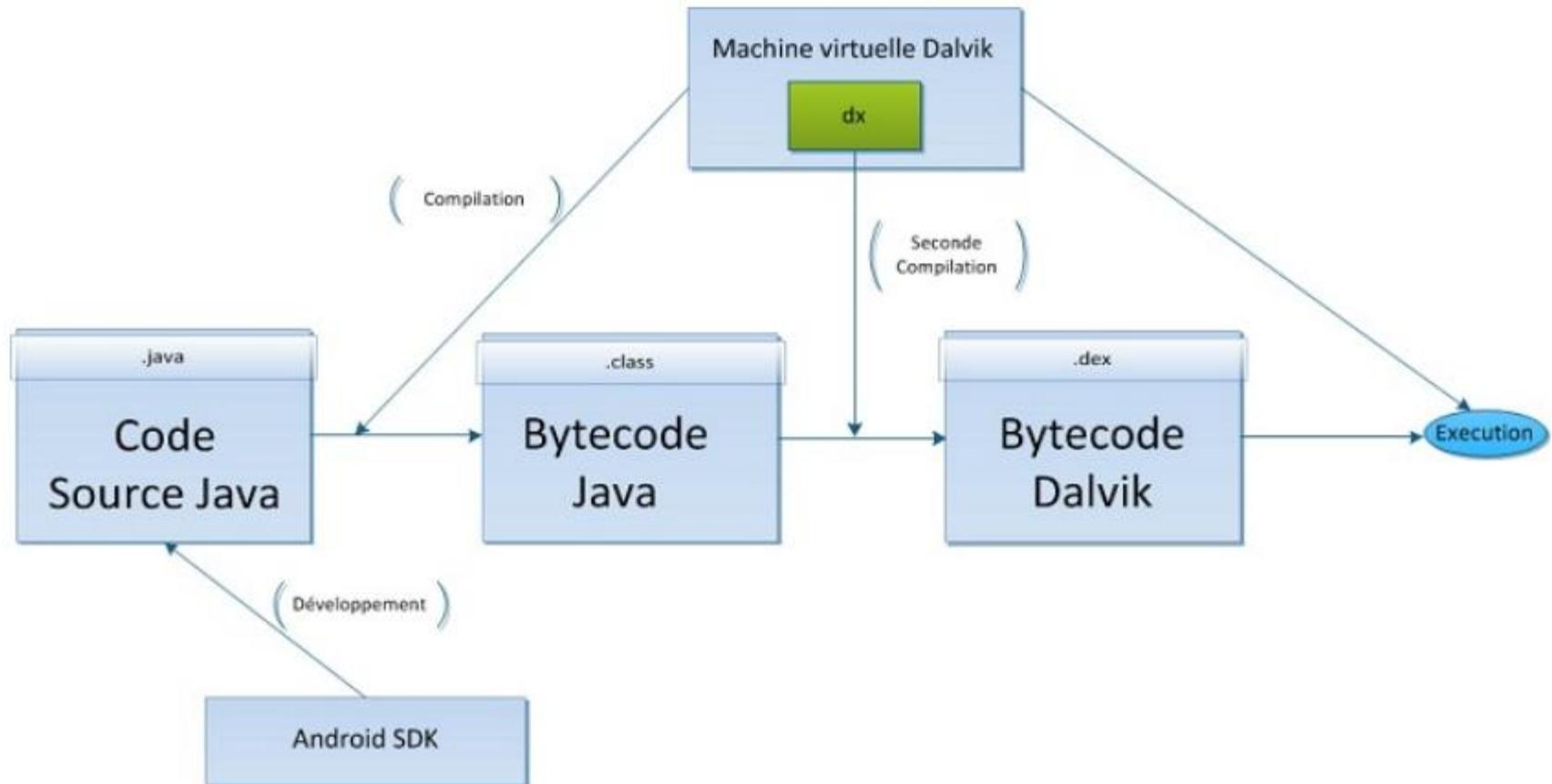
Location Manger: GPS.



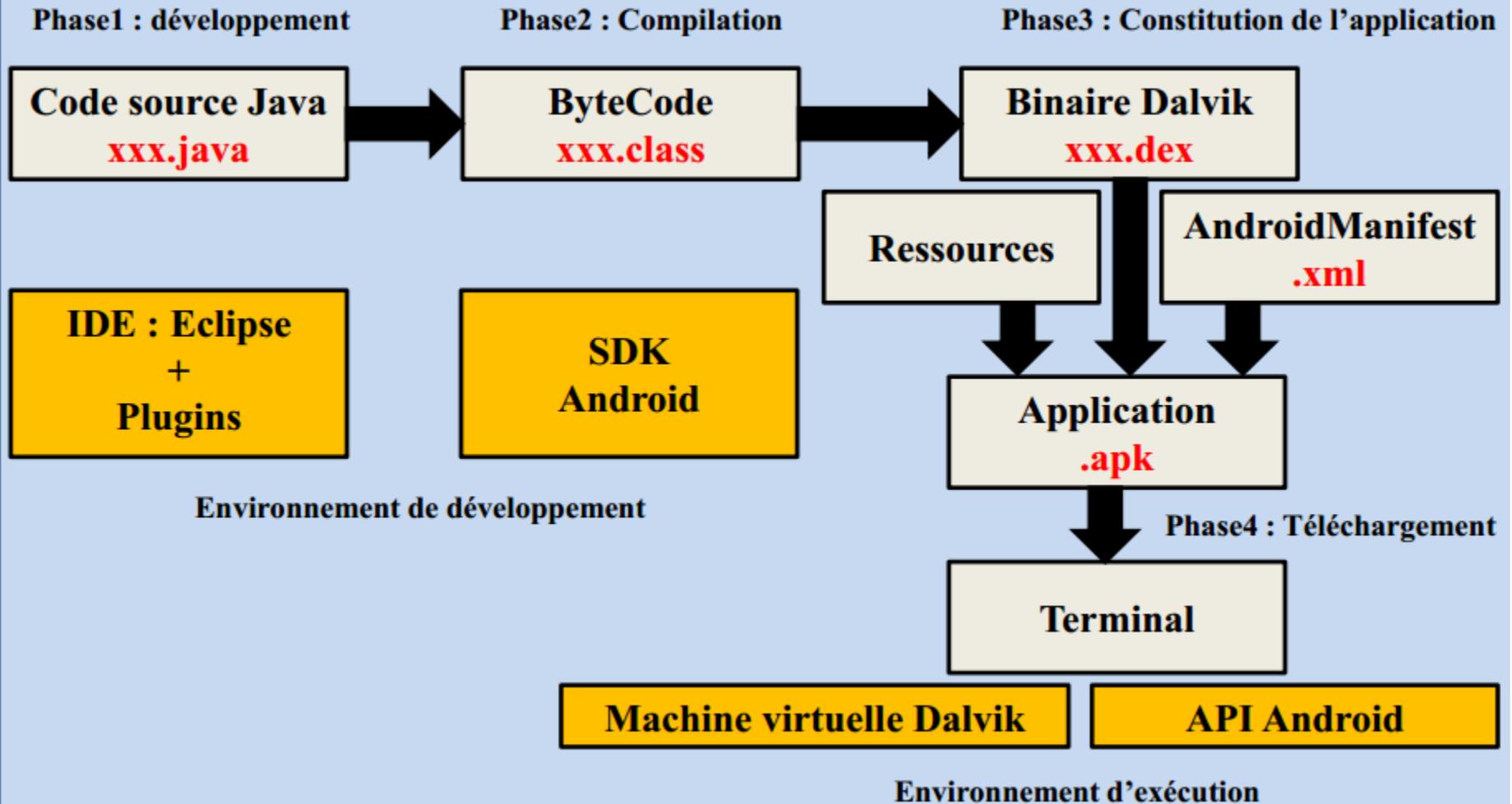
Compilation d'un programme standard en Java



Compilation d'un programme Java Android



Les Phases d'une application Android



Type de projets Android

Il existe trois catégories de projets sous Android:

- Application Android : Type primaire des applications Android destinées à être exécutées directement sur un terminal
- Test Projects : Projet de test d'une application Android
- Library project : Projet de types bibliothèques, équivalents à une API exposant certaines fonctionnalités réutilisables.

Type d'applications Android

- Application de premier plan: c'est une application qui est utilisable uniquement lorsqu'elle est visible et mise en suspens lorsqu'elle ne l'est pas.
- Application d'arrière plan (Services): N'interagit pas avec l'utilisateur, elle s'exécute en tâche de fond.

Type d'applications Android



- Intermittente: c'est une application qui présente une certaine interactivité mais effectue l'essentiel de sa tâche en arrière plan. Ces applications notifient l'utilisateur lorsque cela est nécessaire.
- Widget: ces applications représentées sous forme d'un widget de l'écran d'accueil.

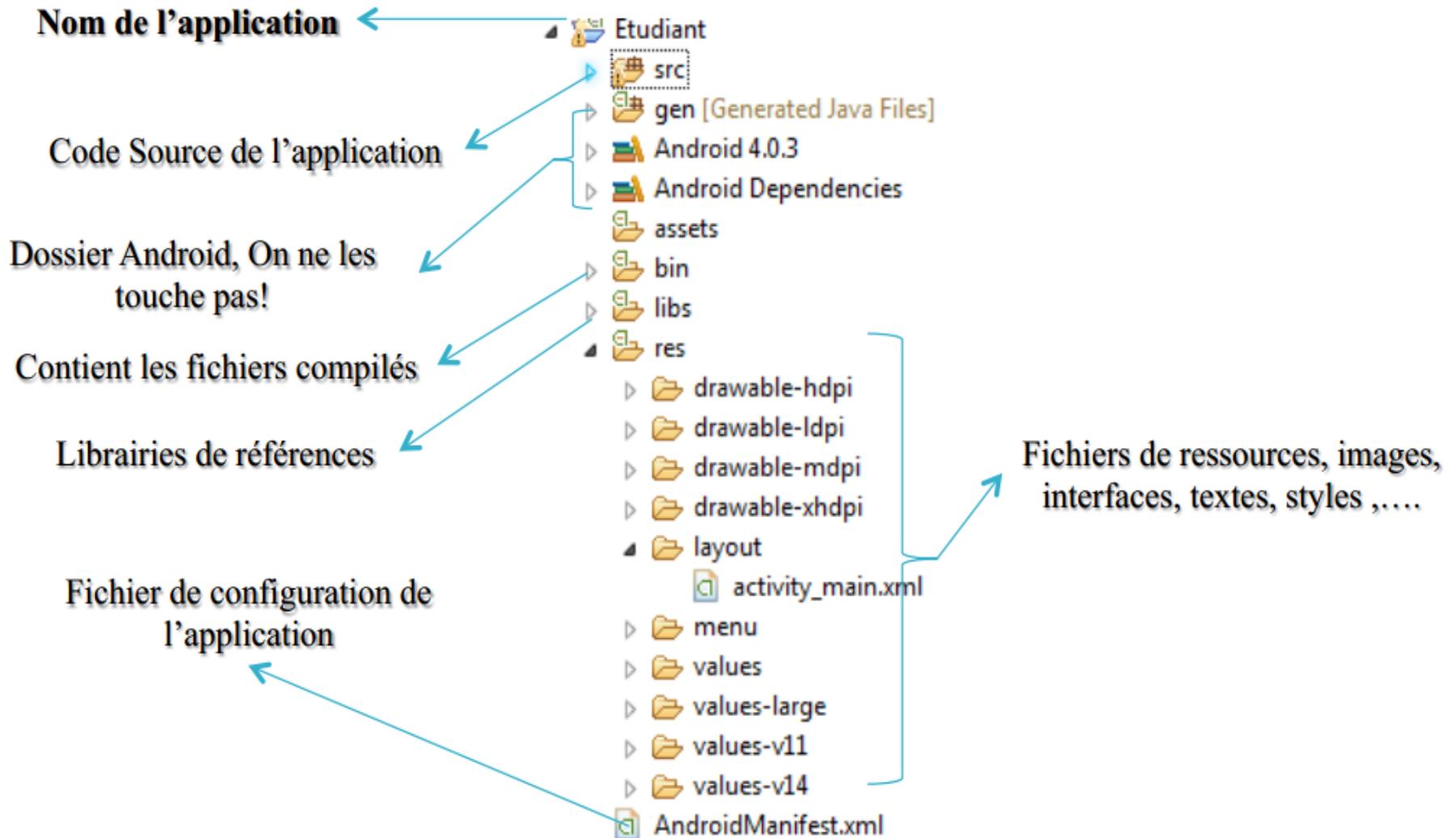
Répertoires d'un projet Android

- bin/: Répertoire qui contient l'Application compilée.
- gen/: Répertoire regroupe le code source produit par les outils de compilation Android.
- libs/: Répertoire des fichiers JAR extérieurs de l'application .
- res/: Répertoire qui contient les ressources utilisées par l'application (images, layouts,...).

Répertoires d'un projet Android

- src/: Répertoire qui regroupe la ou les activités de l'application.
- AndroidManifest.xml: c'est un fichier qui décrit l'application.
- assets/: Répertoire qui contient les autres fichiers statiques fournis avec l'Application pour son déploiement sur le terminal

Répertoires d'un projet Android



Ressources

- Une application Android n'est pas seulement fait de codes mais aussi de ressources statiques (images, sons, text statique,....)
- 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 et 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>
```

Class R

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

- ▶ bin
- ▶ libs
- ▼ res
 - ▶ drawable-hdpi
 - ▶ drawable-ldpi
 - ▼ drawable-mdpi
 - ic_launcher.png
 - reload.png
 - ▶ drawable-xhdpi
 - ▶ drawable-xxhdpi
 - ▶ layout
 - ▼ values
 - strings.xml
 - styles.xml
 - ▶ values-v11

```
1820     public static final class string {
1821         /** Content description for the action bar "home" affordance. [
1822             */
1823         public static final int abc_toolbar_collapse_description=0x7f0a0004;
1824         public static final int app_name=0x7f0a0010;
1825         public static final int hello_world=0x7f0a0011;
1826         public static final int subAct=0x7f0a0012;
1827     }
1828     public static final class style {
```

```
MainActivity.java  strings.xml ✕
1  <?xml version="1.0" encoding="utf-8"?>
2  <resources>
3
4      <string name="app_name">CycleVie</string>
5      <string name="hello_world">Hello world!</string>
6      <string name="subAct">SubActivity</string>
7
8  </resources> 23
```

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)
```

Référencer les ressources

Référencement d'une ressource dans une autre ressource. La forme générale est :

"@type/identificateur"

Exemple :

@string/string_name

@drawable/icon

@color/ma_couleur

Structure d'une app. Android (Composants Java)

- 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
- Ecouteur d'intention diffusées (`android.content.BroadcastReceiver`) :

Structure d'une app. Android (Composants Java)

Permet à une application de récupérer des informations générales (réception d'un SMS, batterie faible, ...)

- Éléments d'interaction
 - Intention (`android.content.Intent`) : permet à une application d'indiquer ce qu'elle sait faire ou de chercher un savoir-faire
 - Filtre d'intentions (`<intent-filter>`) : permet de choisir la meilleure application pour assurer un savoir-faire

AndroidManifest.xml



- Chaque application doit avoir un fichier AndroidManifest.xml dans son répertoire racine.
- Le fichier manifeste présente des informations essentielles à propos votre application pour le système Android
- Le manifeste effectue les opérations suivantes :

AndroidManifest.xml

- Il nomme le paquet Java pour l'application (son identifiant unique)
- Il décrit les composants de l'application - les activités, les services, récepteurs de radiodiffusion, et les fournisseurs de contenu qui constituent l'application.
- Il nomme les classes qui implémentent chacune des composantes et publie leurs capacités (intentions).

AndroidManifest.xml

- Il déclare les autorisations dont l'application doit avoir pour accéder à des parties protégées de l'API et d'interagir avec d'autres applications.
- Il déclare aussi les autorisations qui sont nécessaires pour d'autres avoir pour interagir avec les composants de l'application.
- Il déclare que le niveau minimum de l'API Android, la liste des bibliothèques que l'application nécessite...

AndroidManifest.xml

```
<manifest>
<uses-permission />
  <permission />
  <permission-tree />
  <permission-group />
  <instrumentation />
  <uses-sdk />
  <uses-configuration />
  <uses-feature />
  <supports-screens />
  <compatible-screens />
  <supports-gl-texture />
<application>
  <uses-library />

  <activity>
    <intent-filter>
      <action />
      <category />
      <data />
    </intent-filter>
    <meta-data />
  </activity>
```

Activité



Une activité est un composant d'application qui fournit un écran avec laquelle les utilisateurs peuvent interagir afin de faire quelque chose, comme composer le téléphone, prendre une photo, envoyer un email, ou afficher une carte. Chaque activité est donnée à une fenêtre dans laquelle dessiner son interface utilisateur.

Activité

- Une application est généralement constituée de plusieurs activités qui sont faiblement liés entre eux.
- Chaque activité peut alors démarrer une autre activité afin d'effectuer différentes actions. Chaque fois que commence une nouvelle activité, l'activité précédente est arrêté, mais le système conserve l'activité dans une pile.

Activité

- Classe héritant de la classe Activity d'Android
- Doit être déclarée dans le Manifest pour être visible par le système
- Ne peut être instanciée directement, cette tâche se fait par le système
-  Activity a = new Activity();
- L'instanciation d'une activité se fait par les intentions

Création d'une activité

```
public class ExampleActivity extends Activity {
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        // The activity is being createdddd
    }
    @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").
    }
}
```

Création d'une activité



```
public class MainActivity extends Activity {  
  
    @Override  
    protected void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        RelativeLayout layout = new RelativeLayout(this);  
        TextView text = new TextView(this);  
        text.setText("Hello World !");  
        layout.addView(text);  
        setContentView(layout);  
    }  
  
}
```

Création d'une activité



```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.cci.calculator" >
    <application
        android:allowBackup="true"
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name"
        android:theme="@style/AppTheme" >
        <activity
            android:name=".MainActivity"
            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>
```

Sauvegarder l'état d'une activité

```
public class MainActivity extends Activity {  
    private int value = 0;  
    private Button button;  
  
    @Override  
    protected void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        LinearLayout layout = new LinearLayout(this);  
        button = new Button(this);  
        button.setOnClickListener(new OnClickListener());  
        updateButtonLabel();  
        layout.addView(button);  
        setContentView(layout);  
    }  
}
```

Sauvegarder l'état d'une activité

```
public void updateButtonLabel() {  
    button.setText(""+value);  
}
```

```
private class OnClickListener  
    implements View.OnClickListener {  
    @Override  
    public void onClick(View v) {  
        value++;  
        updateButtonLabel();  
    }  
}
```

Sauvegarder l'état d'une activité

```
@Override
protected
void onRestoreInstanceState(Bundle savedInstanceState) {
    super.onRestoreInstanceState(savedInstanceState);
    value = savedInstanceState.getInt("value");
    updateButtonLabel();
}
```

```
@Override
protected void onSaveInstanceState(Bundle outState) {
    super.onSaveInstanceState(outState);
    outState.putInt("value", value);
}
}
```

Sauvegarder l'état d'une activité

Méthodes du Bundle

```
putInt(String key, int value) ;  
int getInt(String key) ;  
putDouble(String key, double value) ;  
double getDouble(String key) ;  
putString(String key, String value) ;  
String getString(String key) ;  
putStringArrayList(String key, ArrayList<String> list) ;  
ArrayList<String> getStringArrayList(String key) ;  
putCharArray(String key, char[] array) ;  
char[] getCharArray(String key) ;
```

Les IHM

Les interfaces sont définies généralement dans des fichiers XML (cela nous évite d'avoir à créer des instances explicitement)

ADT génère automatiquement une ressource correspondant à l'interface dans le fichier R.java (`accueil.xml` ; `R.layout.accueil`)

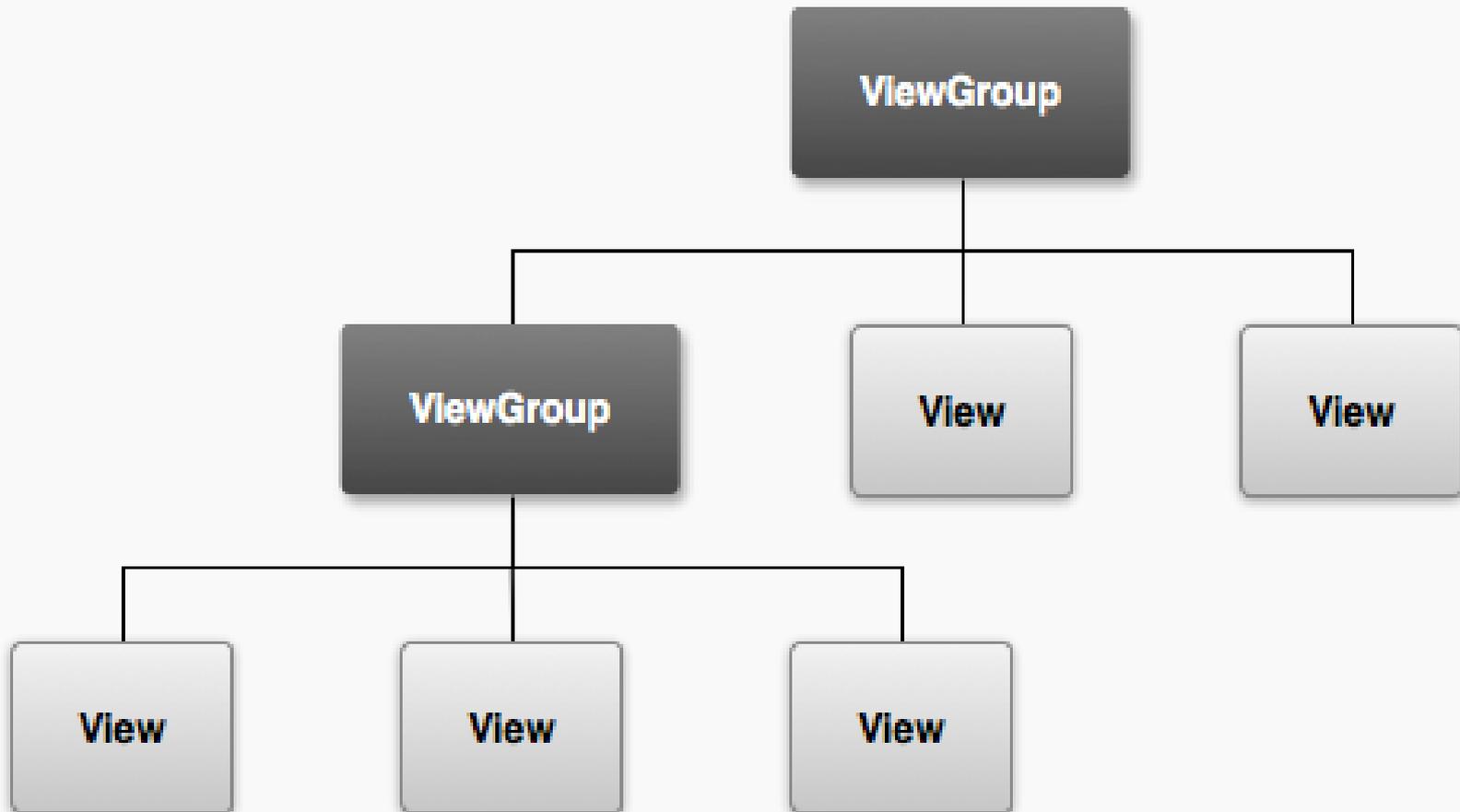
Une interface peut être associée comme vue graphique à une activité.

Les IHM

La d'une interface avec l'utilisateur :

- Les interfaces sont composées d'objets étendant View et ViewGroup ;
- Une vue se dessine et permet les interactions avec l'utilisateur ;
- Un groupe de vues contient d'autres vues ;
- Un groupe de vues organise l'affichage des vues qu'il contient ;

View/ViewGroup



View

View : Cette classe représente le bloc de construction de base pour les composants de l'interface utilisateur. A View occupe une zone rectangulaire sur l'écran et est responsable de l'élaboration et de la gestion des événements. View est la classe de base pour les widgets, qui sont utilisés pour créer des composants d'interface interactifs (boutons, champs de texte, etc.). La sous-classe ViewGroup est la classe de base pour les mises en page, qui sont des conteneurs invisibles qui détiennent d'autres vues (ou une autre vue Groupes) et définir leurs propriétés de mise en page.

ViewGroup



**Un ViewGroup est une vue spécial qui peut contenir d'autres vues (appelés enfants.)
Le groupe de vue est la classe de base pour les modèles et vues conteneurs. Cette classe définit également la classe ViewGroup.LayoutParams qui sert de classe de base pour les paramètres de mises en page.**

Exemple

- Pour déclarer votre mise en page, vous pouvez instancier Afficher les objets dans le code et de commencer à construire un arbre, mais le moyen le plus simple et le plus efficace pour définir votre mise en page est avec un fichier XML.
- Le nom d'un élément XML pour une vue est celui de la classe qu'il représente.
 - <TextView> élément crée un widget TextView*
 - <LinearLayout> crée un élément de viewgroup LinearLayout.*
- Une disposition verticale simple avec une vue de texte et un bouton ressemble à ceci:

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

Création

Une vue peut être déclarée dans un fichier XML

```
<TextView  
android:id="@+id/le_texte"  
android:layout_width="wrap_content"  
android:layout_height="wrap_content"  
android:text="@string/hello"  
android:layout_gravity="center"  
>
```

Identifiant de la vue

Texte à afficher référencé dans les ressources

Alignement de la vue dans le gabarit

Création

Une vue peut être aussi créée dynamiquement

```
public class Activity2 extends Activity {
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        LinearLayout gabarit = new LinearLayout(this);
        gabarit.setGravity(Gravity.CENTER); // centrer les éléments
        graphiques
        gabarit.setOrientation(LinearLayout.VERTICAL); // disposition
        horizontal !
        TextView texte = new TextView(this);
        texte.setText("Programming creation of interface !");
        gabarit.addView(texte);
        setContentView(gabarit);
    }
}
```

Instanciación

Les instances de vues déclarées dans les fichiers XML sont créées automatiquement par le système et peuvent être récupérées dans le code Java.



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

Sachant le type de la vue

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

Contenu

Il est possible d'accéder aux propriétés des widgets en lecture et/ou en écriture

```
EditText edit = (EditText)findViewById(R.id.nom);  
//On y met une valeur en dure  
edit.setText("Voici ta nouvelle valeur");  
//on y met une valeur dans les ressources  
edit.setText(R.string.hello_world);  
//On récupère le contenu  
edit.getText();
```

Evénements

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  
}  
});
```

```
edit.addTextChangedListener(new TextWatcher() {  
@Override  
public void onTextChanged(CharSequence s, int start,  
int before, int count) {  
// do something here  
}  
});
```

Validations

Il est possible de valider le contenu des champs d'un formulaire

```
TextUtils.isDigitsOnly(edit.getText());
```

```
TextUtils.isEmpty(edit.getText());
```

Propriétés communes aux vues



- **android:id="@+id/mon_ident"**
- **android:layout_height, android:layout_width :**
 - **fill_parent / wrap_content**
 - **fill_parent** : Remplit toute la place du parent
 - **wrap_content** : remplit la place que nécessite le contenu
 -
- **android:background: "@color/blue/#000FFFF"**
- **android:visibility : visible / invisible/gone**
- **android:padding="10.5dp"**
- ...

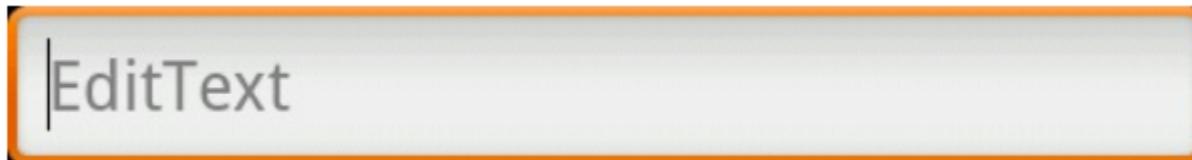
EditText (View)

Code : XML

```
<EditText  
    android:layout_width="fill_parent"  
    android:layout_height="wrap_content"  
    android:hint="@string/editText"  
    android:inputType="textMultiLine"  
    android:lines="5" />
```

Code : Java

```
EditText editText = new EditText(this);  
editText.setHint(R.string.editText);  
editText.setInputType(InputType.TYPE_TEXT_FLAG_MULTI_LINE);  
editText.setLines(5);
```



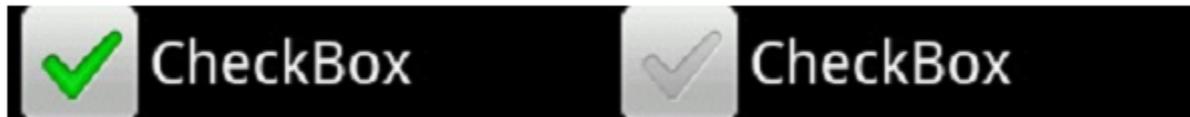
CheckBox (View)

Code : XML

```
<CheckBox
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="@string/checkBox"
    android:checked="true" />
```

Code : Java

```
CheckBox checkBox = new CheckBox(this);
checkBox.setText(R.string.checkBox);
checkBox.setChecked(true)
if(checkBox.isChecked())
    // Faire quelque chose si le bouton est coché
```



ViewGroup

Les groupes sont des vues permettant de définir un pattern d'affichage pour les collections d'objets :

Ils permettent de faire des choix sur un ensemble d'éléments :

- ListView
- GridView
- RadioGroupe
- Galery
- Spinner

ViewGroup

Les groupes utilisent des adapters pour insérer automatiquement les objets (Java) comme item dans la liste.

La source de données est une liste d'objets pouvant provenir :

- ▶ D'une base de données locale
- ▶ D'un service web
- ▶ D'une collection statique
- ▶ ...

Les collections peuvent être:

- ▶ De type simple : String, Double, Integer, ...
- ▶ De type complexe : Instance de classe java

Adapter

Les adapters peuvent être considérés comme un pont entre une source de données et une vue de groupe.

Ils fournissent les accès aux éléments de la source de données

Ils sont aussi responsables de générer des vues pour chaque élément d'un groupe

Adapter

Il existe trois type d'adapter sous Android :

- ▶ ArrayAdapter
- ▶ CursorAdapter
- ▶ SimpleCursorAdapter

Android propose des implémentations génériques pour chacun de ces types

Peuvent être étendus en fonction des besoins de l'application

RadioButton (View) et RadioGroup (ViewGroup)

Code : XML

```
<RadioGroup
  android:layout_width="wrap_content"
  android:layout_height="wrap_content"
  android:orientation="horizontal" >
  <RadioButton
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:checked="true" />
  <RadioButton
    android:layout_width="wrap_content"
    android:layout_height="wrap_content" />
  <RadioButton
    android:layout_width="wrap_content"
    android:layout_height="wrap_content" />
</RadioGroup>
```

RadioButton et RadioGroup

Code : Java

```
RadioGroup radioGroup = new RadioGroup(this);
RadioButton radioButton1 = new RadioButton(this);
RadioButton radioButton2 = new RadioButton(this);
RadioButton radioButton3 = new RadioButton(this);

// On ajoute les boutons au RadioGroup
radioGroup.addView(radioButton1, 0);
radioGroup.addView(radioButton2, 1);
radioGroup.addView(radioButton3, 2);

// On sélectionne le premier bouton
radioGroup.check(0);

// On récupère l'identifiant du bouton qui est coché
int id = radioGroup.getCheckedRadioButtonId();
```



Spinner (ViewGroup)

Déclaration d'un spinner :

```
<Spinner
    android:id="@+id/spinner"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:drawSelectorOnTop="true"
    android:spinnerMode="dropdown" />
```

```
String[] fonction = {"Formateur", "Concepteur", "Analyste", "Architecte", "Comptable",
    "Chef de projet"};|
Spinner spin=(Spinner)findViewById(R.id.spinner);
spin.setOnItemSelectedListener(this);
ArrayAdapter<String> aa=new ArrayAdapter<String>(this,android.R.layout.simple_spinner_item, fonction);
aa.setDropDownViewResource( android.R.layout.simple_spinner_dropdown_item);
spin.setAdapter(aa);
}
```

```
@Override
public void onItemSelected(AdapterView<?> parent, View view, int position,
    long id) {
    Toast.makeText(getApplicationContext(),
        "Selected item (spinner) " + position+ " : "+(String)spin.getAdapter().getItem(position),
        Toast.LENGTH_LONG)
        .show();
}
```

ListView (ViewGroup)

Au sein d'un gabarit, on peut implanter une liste que l'on pourra dérouler si le nombre d'éléments est important. Si l'on souhaite faire une liste plein écran, il suffit juste de poser un layout linéaire et d'y implanter une ListView. Le XML du gabarit est donc:

```
<LinearLayout ...>  
<ListView android:id="@+id/listView1" ...>  
</ListView></LinearLayout>
```

ListView (ViewGroup)

```
ListView list = (ListView) findViewById(R.id.maliste);
ArrayAdapter<String> tableau = new ArrayAdapter<String>(
    list.getContext(), R.layout.ligne, R.id.monTexte);
for (int i=0; i<40; i++) {
    tableau.add("coucou " + i);
}
list.setAdapter(tableau);
```

Avec le layout de liste suivant (ligne.xml):

```
<LinearLayout ...>
    <TextView ... android:id="@+id/monTexte"/>
    <LinearLayout> <ImageView /> </LinearLayout>
</LinearLayout>
```

ListView (ViewGroup)



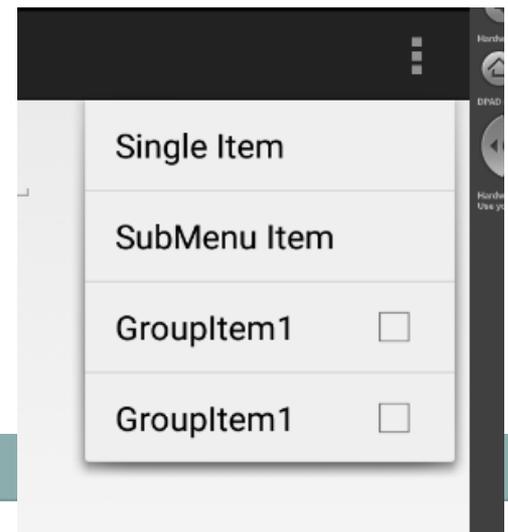
Menu

Un menu est un endroit privilégié pour placer certaines fonctions tout en économisant l'espace dans la Fenêtre.

Il existe deux sortes de menu dans Android :

-Le menu d'options : celui qu'on peut ouvrir avec le bouton Menu sur le téléphone. Si le téléphone est dépourvu de cette touche, Android fournit un bouton dans son interface graphique pour y accéder.

-Les menus contextuels : Similaires aux menus qui s'ouvrent à l'aide d'un clic droit sous Windows et Linux, dans Android ils se déroulent qu'on effectue un long clic sur un élément de l'interface graphique.



Menu

Pour qu'une activité puisse avoir un menu, elle doit redéfinir sa méthode `onOptionsItemSelected()` qui s'exécute lors du premier clic sur le menu

```
@Override
public boolean onCreateOptionsMenu(Menu menu) {

    getMenuInflater().inflate(R.menu.main, menu);

    return true;
}
```

Afin que le menu puisse évoluer avec l'application, il faut redéfinir la méthode qui s'exécute juste avant l'affichage de `onOptionsItemSelected()`

```
@Override
public boolean onPrepareOptionsMenu(Menu menu) {

    MenuItem mi = menu.add("New Item");
    mi.setIcon(R.drawable.ic_location_web_site);
    return super.onPrepareOptionsMenu(menu);
}
```

Menu

```
<menu xmlns:android="http://schemas.android.com/apk/res/android" >
  <item android:id="@+id/item1"
        android:title="Single Item">
  </item>
  <item android:id="@+id/item2"
        android:title="SubMenu Item"
        android:titleCondensed="subCond">
    <menu>
      <item android:id="@+id/item3"
            android:checkable="true"
            android:title="SubItem1"/>
      <item android:id="@+id/item4"
            android:title="SubItem1"
            android:enabled="false"/>
    </menu>
  </item>
  <group android:id="@+id/group1"
        android:checkableBehavior="all" >
    <item
          android:id="@+id/item7"
          android:title="GroupItem1">
    </item>
    <item
          android:id="@+id/item8"
          android:title="GroupItem1">
    </item>
  </group>
</menu>
```

Menu

Pour qu'une activité puisse avoir un menu, elle doit redéfinir sa méthode `onOptionsItemSelected()` qui s'exécute lors du premier clic sur le menu.

```
@Override
public boolean onCreateOptionsMenu(Menu menu) {

    getMenuInflater().inflate(R.menu.main, menu);

    return true;
}
```

Afin que le menu puisse évoluer avec l'application, il faut redéfinir la méthode qui s'exécute juste avant l'affichage du menu.

```
@Override
public boolean onPrepareOptionsMenu(Menu menu) {

    MenuItem mi = menu.add("New Item");
    mi.setIcon(R.drawable.ic_location_web_site);
    return super.onPrepareOptionsMenu(menu);
}
```

Conteneurs (ViewGroup)

Ce sont des vues permettant de définir une prédisposition pour d'autres vues qu'ils contiennent:

- `FrameLayout`
- `AbsoluteLayout`
- `LinearLayout`
- `TableLayout`
- `RelativeLayout`

Conteneurs (ViewGroup)

- LinearLayout: dispose les éléments de façon linéaire (vertical ou horizontal)
- RelativeLayout: Dispose les vues enfants les uns par rapport aux autres
- TableLayout: disposition matricielle (ref:table HTML)

Réutilisation du Layout



Réutilisation d'interfaces : Une interface peut inclure une autre interface

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

    <include
        android:id="@+id/include01"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        layout="@layout/acceuil" >
    </include>
</LinearLayout>
```

LinearLayout

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

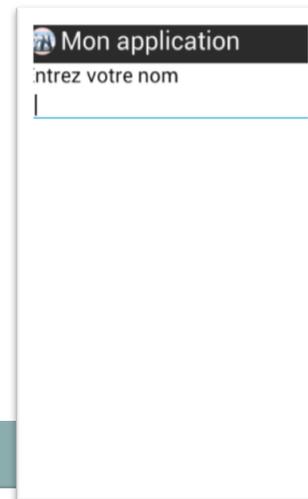
```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"

    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>
```

android:orientation="vertical"



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

- è □ Tableau de positionnement des vues en ligne de TableRow (similaire au `<table>` `<tr>` `<td>` de HTML)

- è □ TableRow hérite de LinearLayout avec alignement automatique des colonnes sur chaque ligne

- è □ Propriétés de TableRow.LayoutParams
 - `layout_column`: indice de départ de la colonne (à partir de 0)
 - `layout_span`: nombre de colonnes occupées

Communications entre composants

Android offre un mécanisme puissant qui permet de faire exécuter certaines actions et de faire circuler des messages entre applications ou à l'intérieur d'une même application

Exemple : Dans votre application de gestion de personnels, vous avez un bouton à côté des coordonnées qui permet d'appeler l'employé, vous n'avez pas à développer une interface pour passer l'appel ! Vous allez demander au système de le faire à l'aide des Intent

Intent

Les intents facilitent la communication entre différents composants. Trois cas de figure :

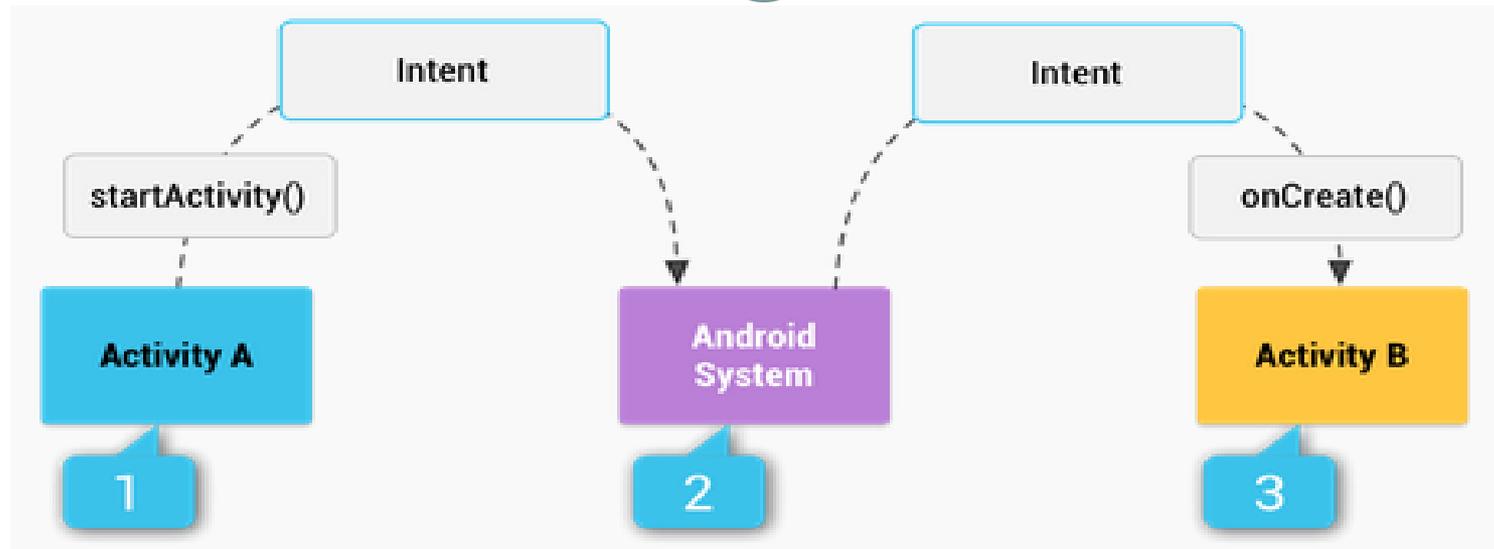
- **Pour démarrer une activité (avec/sans attente de résultat**
- **Pour démarrer un service**
- **Pour envoyer un Broadcast**

Types d'Intents

Intent Explicite : On spécifie le composant destinataire qui va traiter l'intent (*fully-qualified class name*)

Intent Implicite : On ne spécifie pas le composant, mais on offre suffisamment d'informations permettant au système de trouver le bon composant capable de traiter l'intent.

Intent Explicite

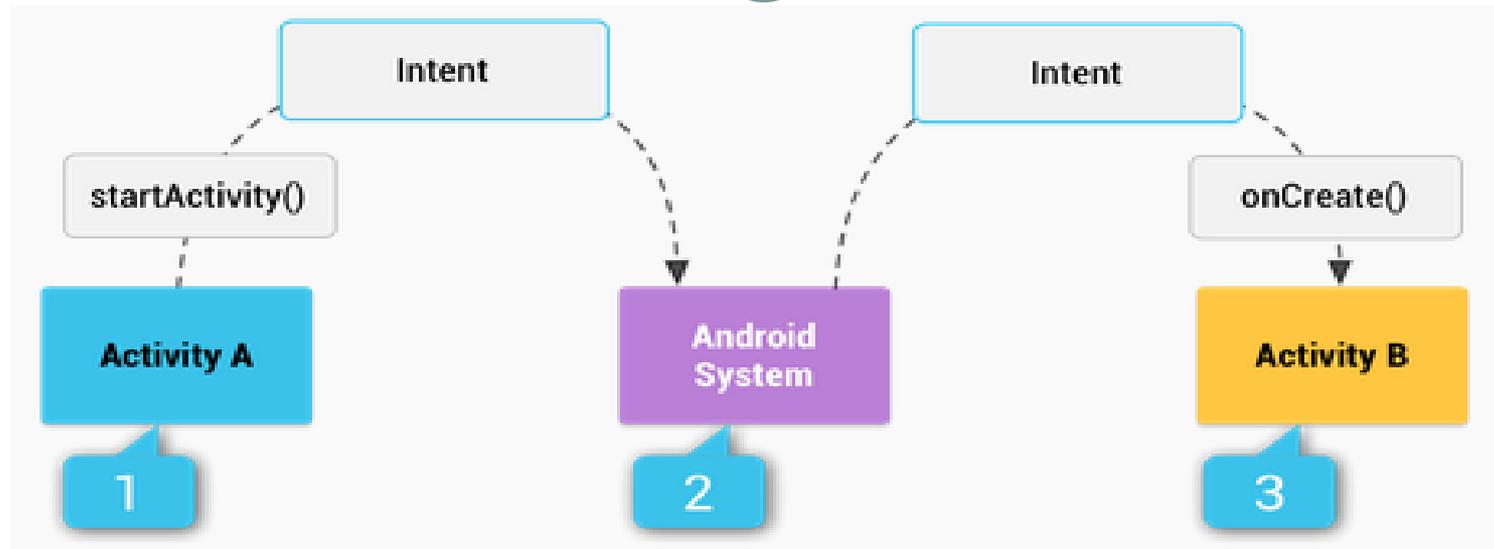


1 : Activité A crée un intent en précisant le composant destinataire et le passe à la méthode `startActivity()`.

2 : Le système trouve immédiatement le composant préciser dans l'intent. (Activity B)

3 : Le système démarre l'activité B en appelant sa méthode `onCreate()` et lui passe l'intent reçu de l'activité A

Intent Implicite



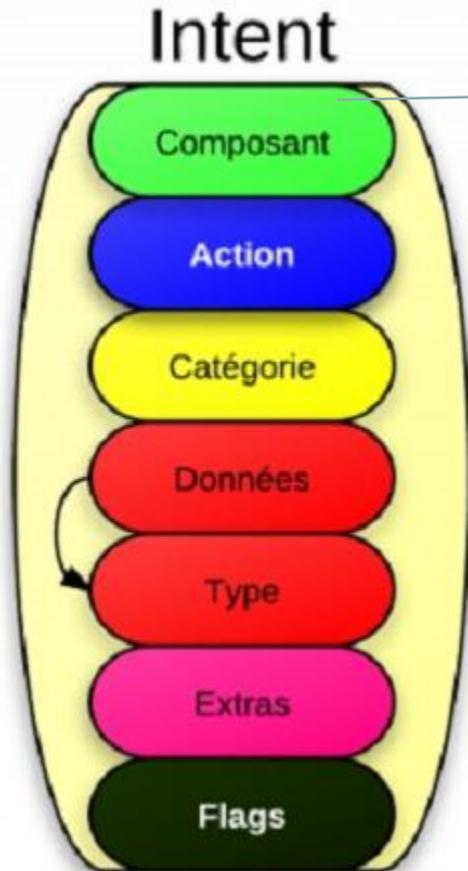
1 : Activité A crée un intent avec juste une description de l'action désirée et le passe à la méthode `startActivity()`.

2 : Le système cherche dans les `manifest.xml` des applications existantes sur le device, si le contenu de l'intent correspond à un *intent filter* d'un composant, le système le choisit. (B par exemple)

3 : Le système démarre l'activité B en appelant sa méthode `onCreate()` et lui passe l'intent reçu de l'activité A

Intent

Techniquement, un intent est un objet de la forme :



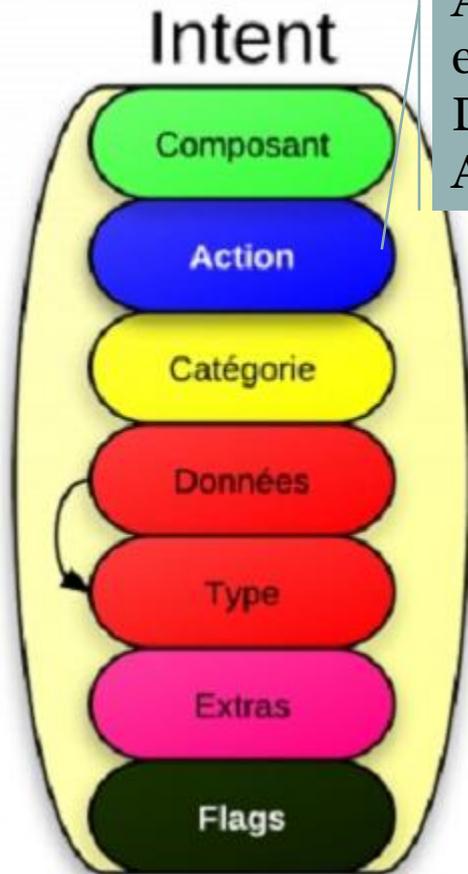
Ce champ permet de définir le destinataire de l'Intent, celui qui devra le gérer. Si ce champ est donné, l'intent est alors explicite

Intent

Techniquement un intent est un objet de la forme :

Action qu'on désire exécuter, appeler un numéro, envoyer email...

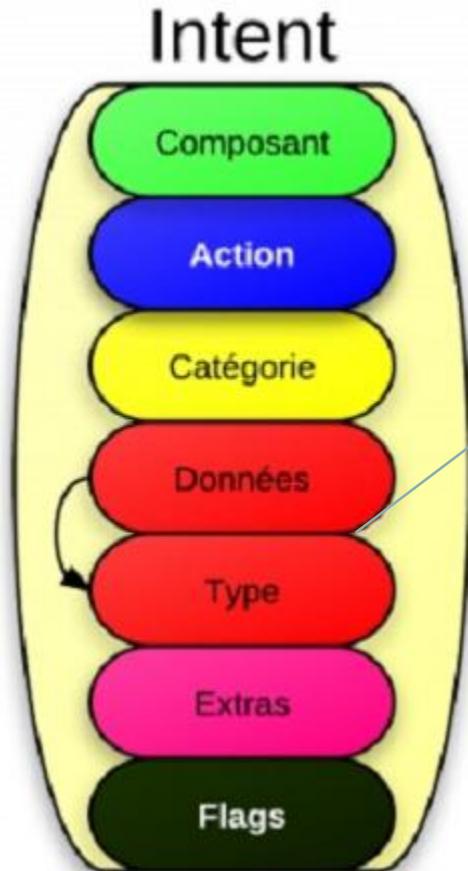
Des constantes de classe Intent de la forme ACTION_NomAction



- ACTION_MAIN
- ACTION_VIEW
- ACTION_ATTACH_DATA
- ACTION_EDIT
- ACTION_PICK
- ACTION_CHOOSER
- ACTION_GET_CONTENT
- ACTION_DIAL
- ACTION_CALL
- ACTION_SEND
- ACTION_SENDTO
- ACTION_ANSWER
- ACTION_INSERT
- ACTION_DELETE
- ACTION_RUN
- ACTION_SYNC
- ACTION_PICK_ACTIVITY
- ACTION_SEARCH
- ACTION_WEB_SEARCH
- ACTION_FACTORY_TEST

Intent

Techniquement, un intent est un objet de la forme :



Les données que va traiter l'application destinataire, et leurs types
Sous forme d'URI

```
<schéma> : <information> { ? <requête> } { # <fragment> }
```

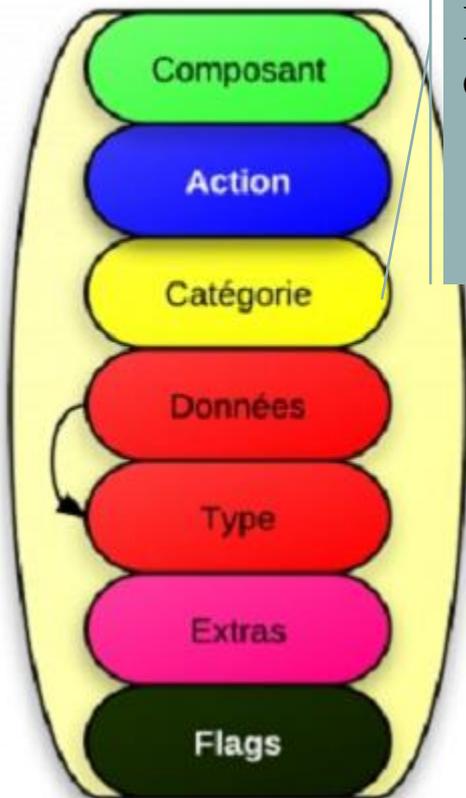
```
Uri sms = Uri.parse("sms:0606060606");
```

```
Uri sms = Uri.parse("sms:0606060606,0606060607?  
body=Salut%20les%20potes");
```

Intent

Techniquement, un intent est un objet de la forme :

Intent

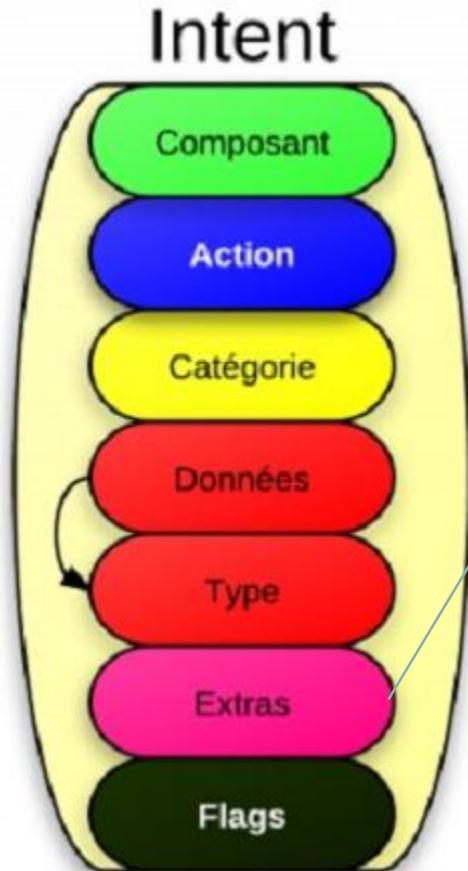


Information additionnelle à propos le type de composant capable de traiter l'intent

- CATEGORY_DEFAULT
- CATEGORY_BROWSABLE
- CATEGORY_TAB
- CATEGORY_ALTERNATIVE
- CATEGORY_SELECTED_ALTERNATIVE
- CATEGORY_LAUNCHER
- CATEGORY_INFO
- CATEGORY_HOME
- CATEGORY_PREFERENCE
- CATEGORY_TEST
- CATEGORY_CAR_DOCK
- CATEGORY_DESK_DOCK
- CATEGORY_LE_DESK_DOCK
- CATEGORY_HE_DESK_DOCK
- CATEGORY_CAR_MODE
- CATEGORY_APP_MARKET

Intent

Techniquement, un intent est un objet de la forme :



Les Extras servent à transiter d'autres données additionnelles

Source :

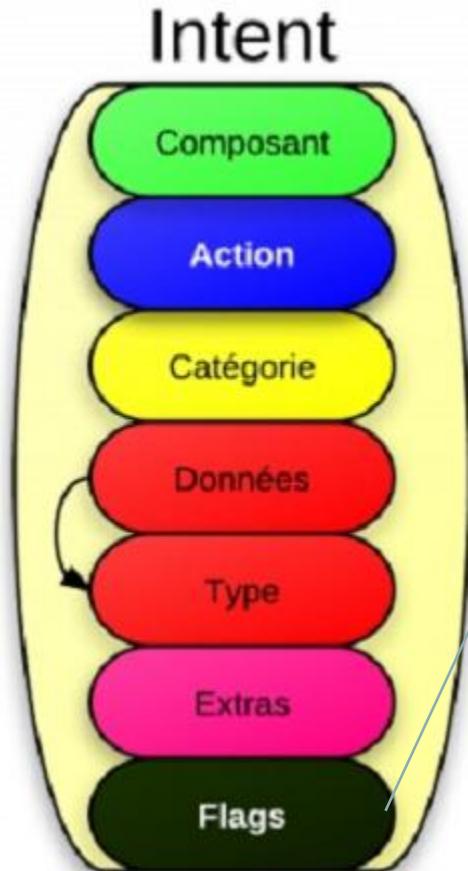
```
Intent i = new Intent();  
String[] noms = new String[] {"Dupont", "Dupond"};  
i.putExtra(FirstClass.NOMS, noms);
```

Dest. :

```
String[] noms = i.getStringArrayExtra(FirstClass.NOMS);
```

Intent

Techniquement, un intent est un objet de la forme :



Les flags servent à préciser comment le systeme traitera le composant dest.

Ex : `FLAG_ACTIVITY_NO_HISTORY`

Si ce flag est présent , l'app. Dest. Ne sera pas gardé dans l'historique si on la quitte.

Intent explicite sans retour

Si l'activité source n'attend pas de résultats de l'activité destinataire, l'appel se fait naturellement ainsi :

```
Intent destActivite = new Intent(MainActivity.this,  
IntentExample.class);
```

```
// On rajoute un extra  
secondeActivite.putExtra(AGE, 31);
```

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

Intent explicite avec retour

Si l'activité source attend des données qui proviennent de l'activité destinataire, l'appel se fait ainsi à l'aide de la méthode ***void startActivityForResult(Intent intent, int requestCode)***

requestCode : identifiant unique de l'intent **différent de 0**

Quand l'activité appelée s'arrêtera, la première méthode de callback appelée dans l'activité précédente sera ***void onActivityResult(int requestCode, int resultCode, Intent data)***

requestCode : l'identifiant, repérer l'intent appelant 2eme activité

resultCode : information sur l'état d'achèvement du 2eme activité ,
Activity.RESULT_OK si tout s'est bien déroulé

data: données attendues

Intent explicite avec retour

Dans la seconde activité, vous pouvez définir un résultat avec la méthode ***void setResult(int resultCode, Intent data)***

```
Intent result = new Intent();  
result.putExtra("akey", "1");  
setResult(RESULT_OK, result);  
finish();
```

Dans l'activité 1 :

```
data.getStringExtra("akey");
```

Exemples

Actions

- `android.intent.action.CALL` appel téléphonique
- `android.intent.action.EDIT` affichage de données pour édition par l'utilisateur
- `android.intent.action.MAIN` activité principale d'une application
- `android.intent.action.VIEW` affichage de données
- `android.intent.action.WEB_SEARCH` recherche sur le WEB

Catégories

- `android.intent.category.DEFAULT` activité pouvant être lancée explicitement
- `android.intent.category.BROWSABLE` peut afficher une information désignée par un lien
- `android.intent.category.LAUNCHER` activité proposée au lancement par Android

Lancement d'activité à l'aide d'Intent

Lancer explicitement une activité

```
Intent demarre = new Intent(context,  
nomActiviteALancer.class);  
startActivity(demarre);
```

Lancer implicitement une activité

- Exemple : lancer un navigateur sur une page :

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

- Exemple : appeler un n° de téléphone :

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

Intent

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)
`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

SQLite

- ▶ SQLite est une base de données relationnelles disponible sur tous les terminaux Android et le SDK fournit des API permettant au développeur de créer/gérer sa base de données privées.
- ▶ Une application peut disposer de plusieurs bases de données (il n'y a pas de limitation à priori) . Toutes les bases de données sont sauvegardées sous :
 - `/data/data/<package_name>/databases`

SQLite

- ▶ Les types de données supportées :
 - INTEGER : Valeur entière signée sur 1 à 8 bytes
 - REAL : Valeur flottante
 - TEXT : Chaîne de caractère formatée (UTF-8, UTF-16BE or UTF-16LE)
 - NUMERIC: données blob stockées comme elle a été entrée
 - NONE : Type non précisé

- ▶ SQLite propose un tableau d'affinité permettant de mapper un type de données dans les langages avancés à l'un des types supportés dans le framework.

- ▶ INT , INTEGER, TINYINT, SMALLINT, MEDIUMINT, BIGINT, UNSIGNED BIG INT, INT2, INT8
- INTEGER

- CHARACTER(20) , VARCHAR(255), VARYING CHARACTER(255), NCHAR(55), NATIVE CHARACTER(70), NVARCHAR(100), TEXT, CLOB
- TEXT

SQLite

- ▶ L'application embarque les commandes nécessaires pour la manipulation de la base de données : création, suppression,....
- ▶ Le développeur se doit de créer les scripts de création de la base de données et les intégrer à l'application au moment de la compilation.
- ▶
- ▶ La base de donnée est automatiquement créée par le système qui dispose d'un mécanisme permettant de vérifier l'état de la base.
- ▶ Lors du premier appel à la base, si celle-ci n'existe pas le système exécutera alors les scripts de création des tables.
- ▶ La base de données n'est pas créée tant qu'on y fait pas référence
- ▶ La base de données est générée au runtime

SQLite

- ▶ Syntax SQL

```
CREATE TABLE categorie( nom TEXT,  
description TEXT,  
id INTEGER PRIMARY KEY AUTOINCREMENT,);
```

- ▶ **insert into** categorie **values** ("Mathématique",
"Livre de mathématiques");

- ▶ **update** categorie **set** description="Description
des livres de Mathématiques" **where** nom **like**
%théma%;

SQLite

android.database : Encapsule les classes nécessaires pour travailler avec les bases de données

android.database.sqlite : Classes pour les fonctionnalités liées à SQLite

▶ Manipuler la base de données

SQLiteOpenHelper permet de manipuler les bases de données SQLite : création, évolution...

onCreate() : à la création de la base

onUpgrade() : Evolution de la base

SQLite

- ▶ SQLiteOpenHelper est appelé à chaque requête d'accès à la base, elle est chargée de la création et de la mise à jour de la BD.
- ▶ Créer une classe qui implémente SQLiteOpenHelper afin d'indiquer au système la structure de la base de données de l'application.

```
public class myDB extends SQLiteOpenHelper {
    public myDB(Context context, String name, CursorFactory factory, int version)
    { super(context, name, factory, version);
    // TODO Auto-generated constructor stub }

    @Override
    public void onCreate(SQLiteDatabase db) {
        //Script de Création de la base
    }

    @Override
    public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion)
    {
        // Script de mise à jour de la base
    }
}
```

SQLite

- ▶ SQLiteDatabase : Classe permettant de manipuler une base de données SQLite sous Android. Elle dispose d'un ensemble de méthodes permettant d'accéder à la base en lecture et en écriture:
 - Open() : Ouvrir la base de données en lecture ou en écriture
 - Query() : Pour exécuter des requêtes SQL
 - Update () : Fermer dla base
 - Close() : Fermer une connection à la base
- ▶ Plus spécifiquement elle propose des méthodes génériques pour les requêtes CRUD : Create, read, update, delete.
- ▶ En addition elle propose execSQL permettant d'exécuter des requêtes SQL natives.

SQLite

Création d'une table dans la base de données

```
@Override
public void onCreate(SQLiteDatabase db)
{
    String script = "CREATE TABLE categorie( "
    + " nom TEXT, " +
    "description TEXT" +
    "id INTEGER PRIMARY KEY AUTOINCREMENT);";
    db.execSQL(script);
}
```

SQLite

- ▶ Accéder à la base :

```
myDB helper = new myDB(HomeActivity.this, "myDB", null, 1);
```

Context

Nom de la base

CursorFactory

Version de la base

- ▶ Ouvrir la base de données en écriture

```
SQLiteDatabase db = db.getWritableDatabase();
```

- ▶ Ouvrir la base de données en lecture

```
SQLiteDatabase db = db.getReadableDatabase();
```

SQLite

- ▶ Ajouter un nouvel objet dans la base de données
 - Utiliser les requêtes natives SQL
 - Utiliser les méthodes génériques de la classe SQLiteDatabase
- ▶ Les méthodes génériques permettent d'effectuer les requêtes CRUD plus simplement.

```
ContentValues ct = new ContentValues();  
Categorie c = new Categorie();  
c.description = "Description of categorie";  
c.nom = "Mathématiques";  
ct.put("nom", c.description);  
ct.put("nom", c.nom);  
db.insert("categorie", null, ct);
```

Nom de la table

Forcer une colonne à recevoir une valeur nulle

HashMap contenant les colonnes de la table et leur valeurs

SQLite

ContentValues

- ▶ Classe de type HashMap permettant de passer les valeurs à insérer dans la base de données aux méthodes génériques
- ▶ Les clefs sont les noms des colonnes des tables telles qu'elles ont été nommées à la création de la table
- ▶ Les valeurs sont les valeurs à insérer pour chaque colonne

SQLite

Requêtes natives SQL

- ▶ `myBase.execSQL("insert into categorie values (" + c.nom + ", " + c.description + "");`
- ▶ `myBase.execSQL("insert into categorie values (?, ?", new String[] {c.nom, c.description});`
- ▶ `myBase.execSQL("delete from categorie");`
- ▶ `myBase.execSQL("delete from categorie where id=?", new Integer[] {18});`

SQLite

Curseur « SQLiteCursor »

- ▶ Un curseur est une classe permettant d'exposer les réponses d'une requête SQLite en Android.
- ▶ Les curseurs peuvent être considérés comme des ResultSet
- ▶ Utilisés dans les requêtes de lecture sur la base
- ▶ Cursor est une classe dérivée de SQLCursor exposant les des méthodes permettant de manipuler plus simplement les curseurs

SQLite

```
Cursor cursor = myBase.query("categorie",  
null, null, null, null, null, null);
```

- ▶ Renvoie toutes les lignes et toutes les colonnes de la table

- ▶ Ouverture explicite

- ▶ Parcourir le curseur :

```
Cursor cursor = myBase.query("categorie", null, null, null, null, null,  
null);  
while (cursor.moveToNext()){  
    nom = cursor.getString(cursor.getColumnIndex("nom"));  
    description = cursor.getString(cursor.getColumnIndex("description"));  
    .....  
}
```

SQLite

- ▶ Obtenir le nombre de lignes dans un curseur :

```
cursor.getCount ();
```

- ▶ Obtenir le nombre de colonnes :

```
cursor.getColumnCount ();
```

- ▶ Se déplacer dans un curseur

```
cursor.move(-1); //remonter d'un niveau
```

```
cursor.move(2); //Descendre de deux niveaux
```

- ▶ Fermer le curseur :

```
cursor.close ();
```



Il est obligatoire de fermer un curseur après utilisation!

SQLite

▶ Bonnes pratiques

- Utiliser une classe statique dans laquelle vous déclarerez le nom de la base de données, des tables ainsi que les colonnes des tables. Ce qui vous permettra de vous embrouiller dans les noms des tables ainsi que les colonnes.
- N'oubliez jamais de fermer la base de données après chaque utilisation
- Toujours fermer les curseurs
- Vérifiez toujours qu'un curseur a au moins une ligne avant de le parcourir.
- Evitez d'accéder à la base de données dans le thread principal de l'application