

# Introduction au Développement d'Application Android

Amosse EDOUARD,  
Doctorant, INRIA/I3S - UNSA

# OBJECTIF DU COURS

- Comprendre l'architecture du système Android
- Comprendre l'organisation d'une application Android
- Développer et déployer des applications natives Android
- Connaitre les bonnes pratiques du développement d'application Android

# Référence

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

# Contraintes Mobiles

- ▶ Tenir compte du matériel :
- ▶ Une puissance processeur faible;
- ▶ Une RAM limitée;
- ▶ Plusieurs type de résolutions de l'écran;
- ▶ Des coûts élevés de transferts de données;
- ▶ Des connexions réseau moins stables;
- ▶ Efficacité : optimiser votre code afin qu'il soit rapide et réactif.

# Contraintes mobiles

- La performance;
- La réactivité;
- La sécurité;
- La transparence;

# Pourquoi Android?

Plusieurs langages :



Multi OS



# Pourquoi Android?

Plusieurs outils :



Déploiement facile (multi plateforme)



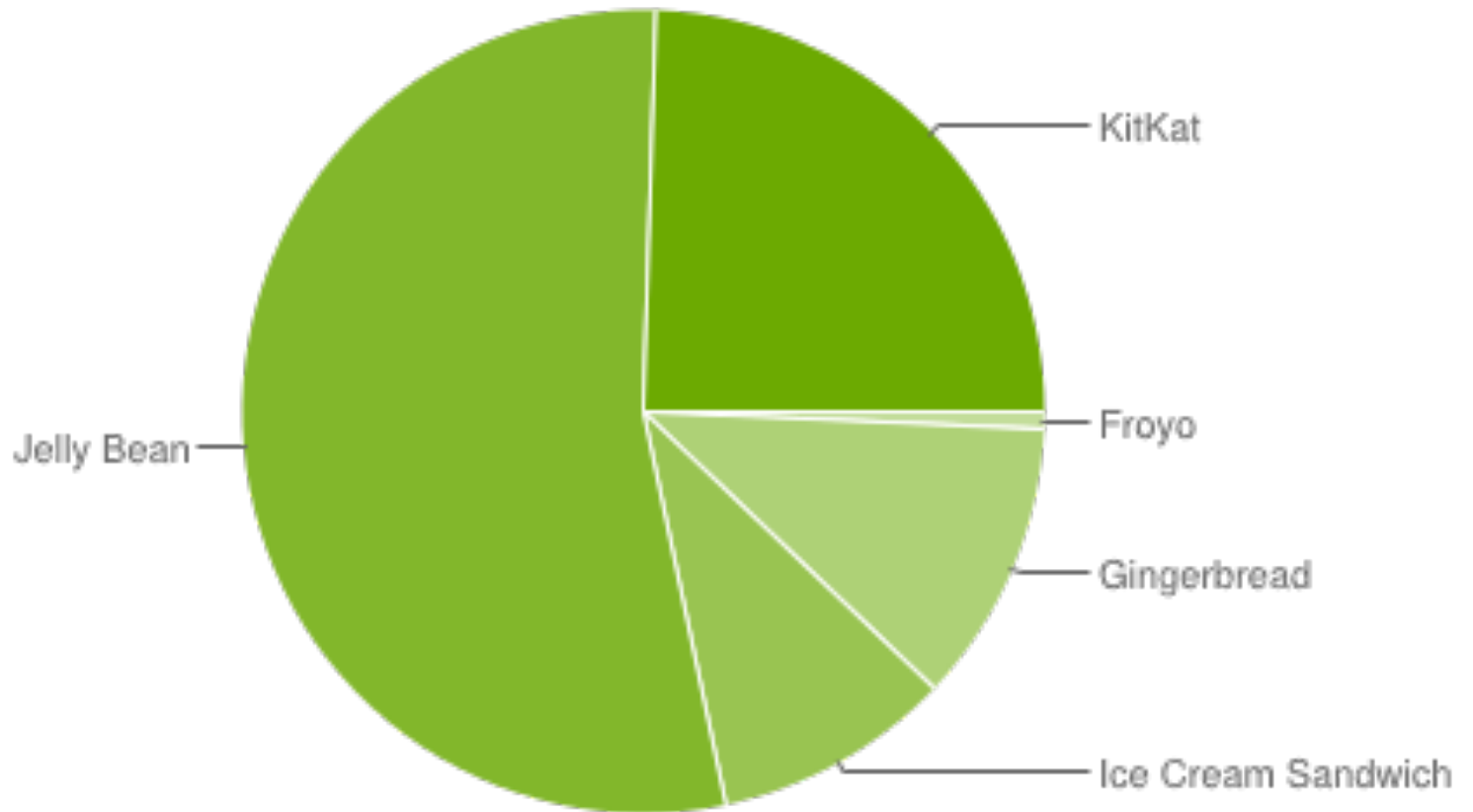
# Historique/ Définition

- ▶ 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





# Versions de l'OS

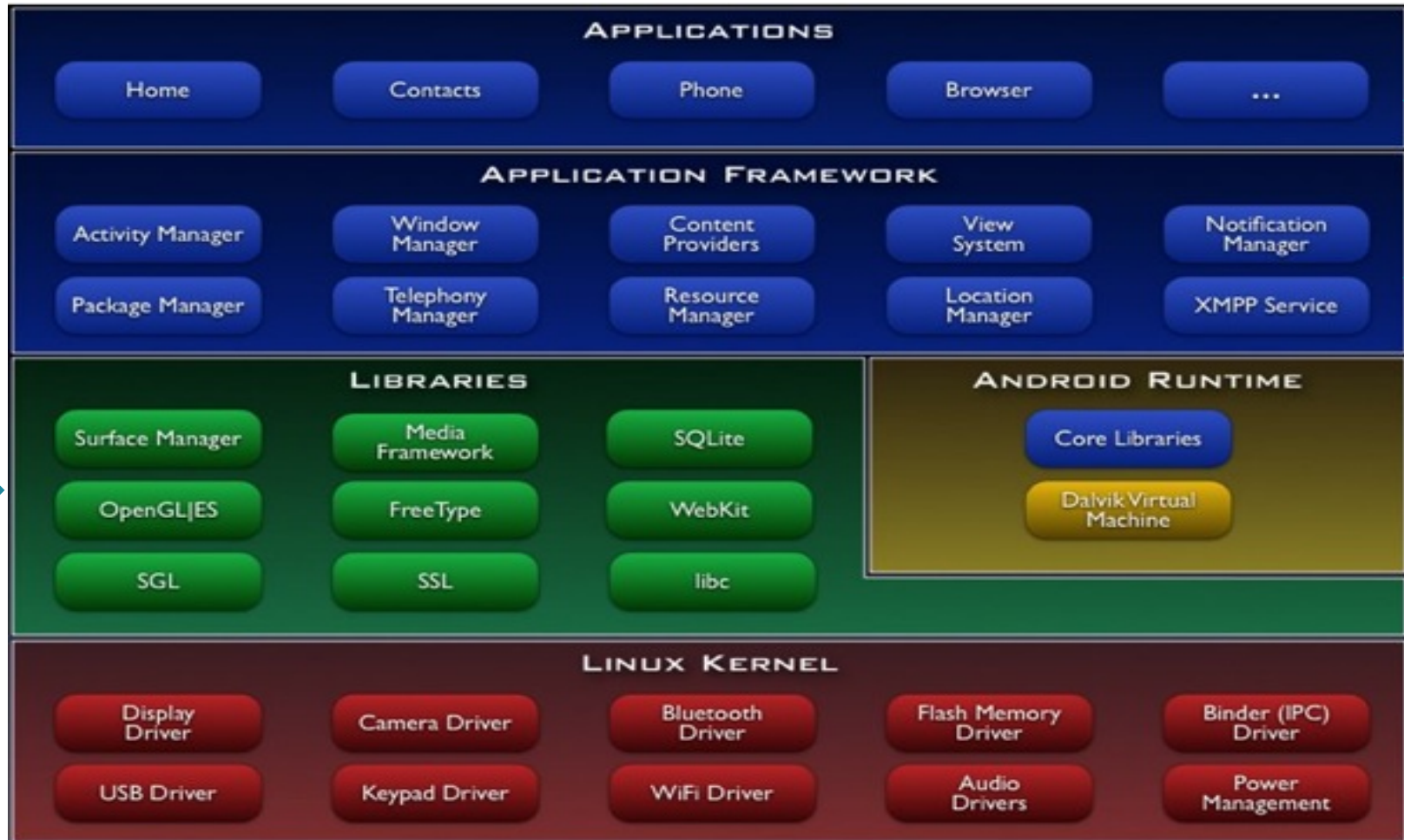


# Plateforme Android

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

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

# Plateforme / Architecture



# Android pour tous

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

# Android vs langages de programmation

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

# 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

# Android vs Développement Hybride

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



# Pour ce cours....

- ▶ Framework et langage
  - Android SDK
  - Java
  - XML
- ▶ Outils
  - Eclipse (Intellij ou Netbeans)
  - Le plugin ADT

# Android SDK

- ▶ Outil de développement d'application Android
- ▶ Fournit un environnement unifié permettant de développer « facilement » des applications Android
- ▶ Mise à jour automatique via le SDKManager
- ▶ Prise en charge de toutes les versions d'Android
- ▶ Plusieurs Outils et API
  - Android xx
  - Google API xx
  - Outils d'administration, de débogage ...

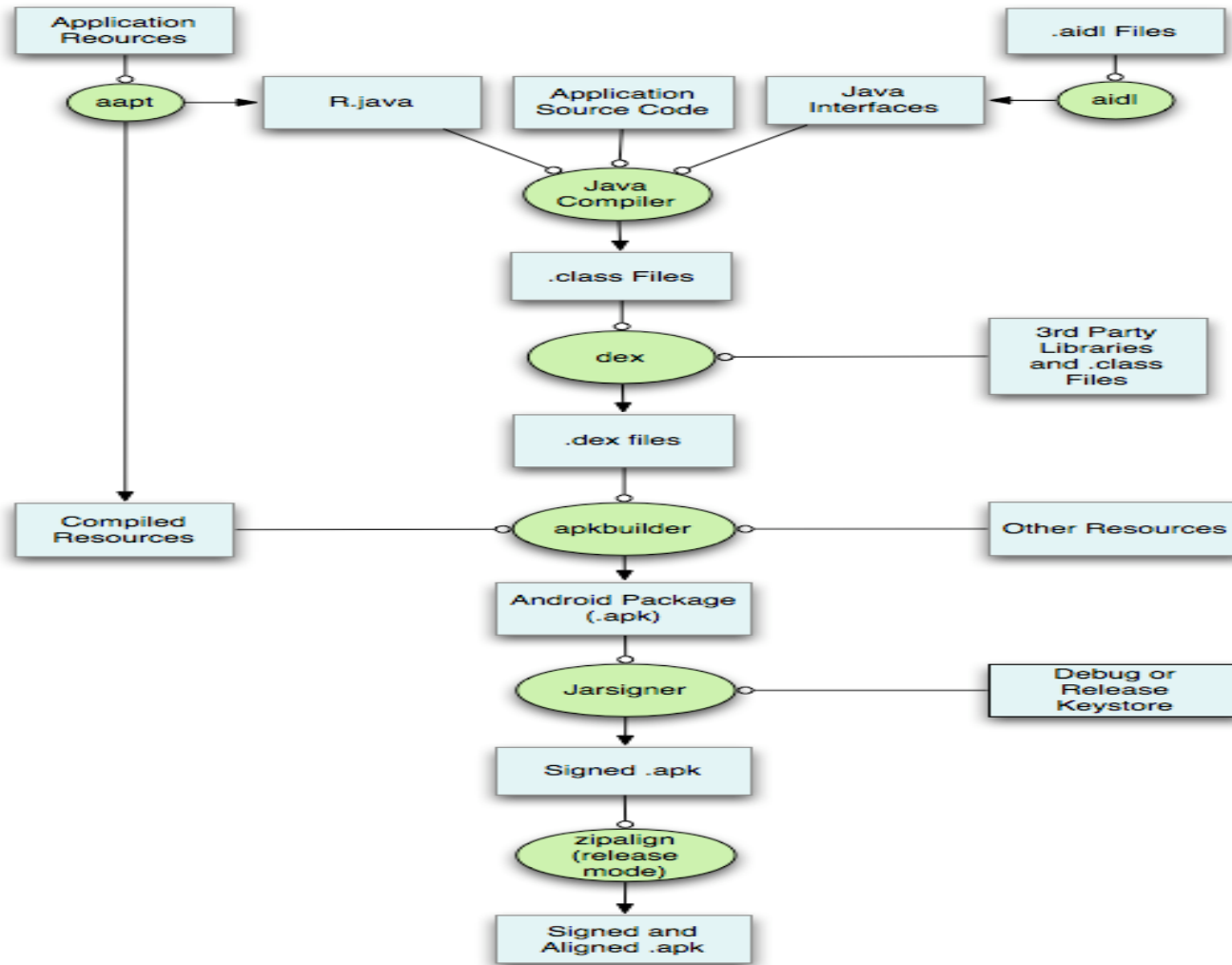
# Outils du SDK

- ▶ **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

# Compilation

- La compilation sous Android avec le plugin ADT est automatisée.
- Android SDK utilise aussi ANT pour automatiser la compilation.
- Android 2.2 Froyo intègre le compilateur JIT(Just in Time compiler) stable.

# Compilation



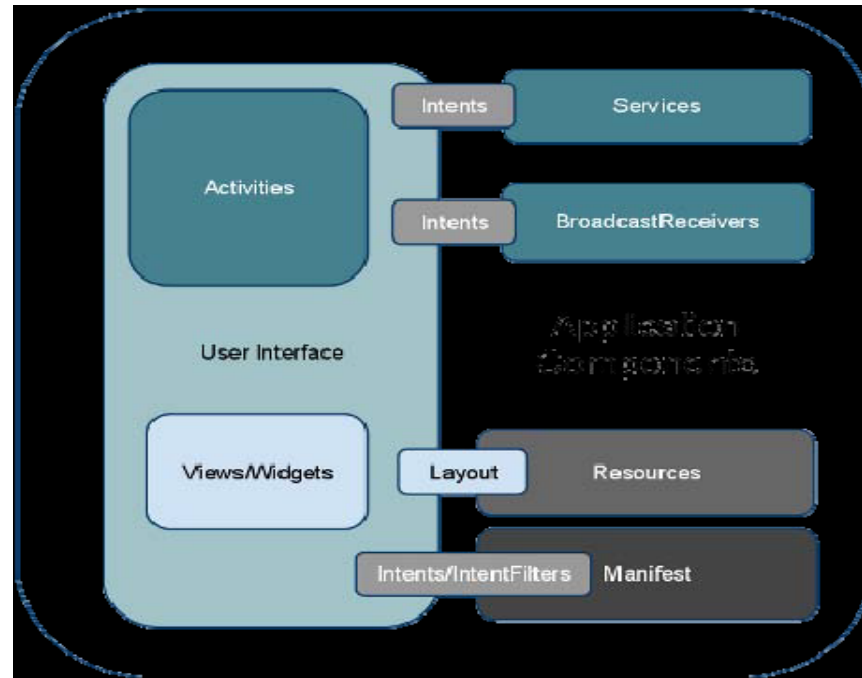
<http://developer.android.com/tools/building/index.html>

# Contraintes Mobiles

- Les contraintes de test;
- Le faible nombre d'API disponibles;
- Les limitations matériel (mémoire, espace disque,...);
- Migration;
- Performances;
- Langage propriétaires

# Composante d'une application Android

- Les applications Android sont constitués de composants à couplage
- Les composants sont liés par un Manifest qui décrit chacun d'eux ainsi que les interactions entre elles.



# Composante d'une application Android

1. **Activities** : Couche de présentation de l'application;
2. **Services** : les composants qui tournent en arrière plan;
3. **Content providers** : Partage de contenus entre applications;
4. **Intents** : Framework de communication interapplications.
5. **Broadcast receivers** : Consommateurs des messages diffusés par les intents.
6. **Widgets** : Composant d'application visuels;
7. **Notifications** : Framework de notifications aux utilisateurs;



# Type de Projet Android

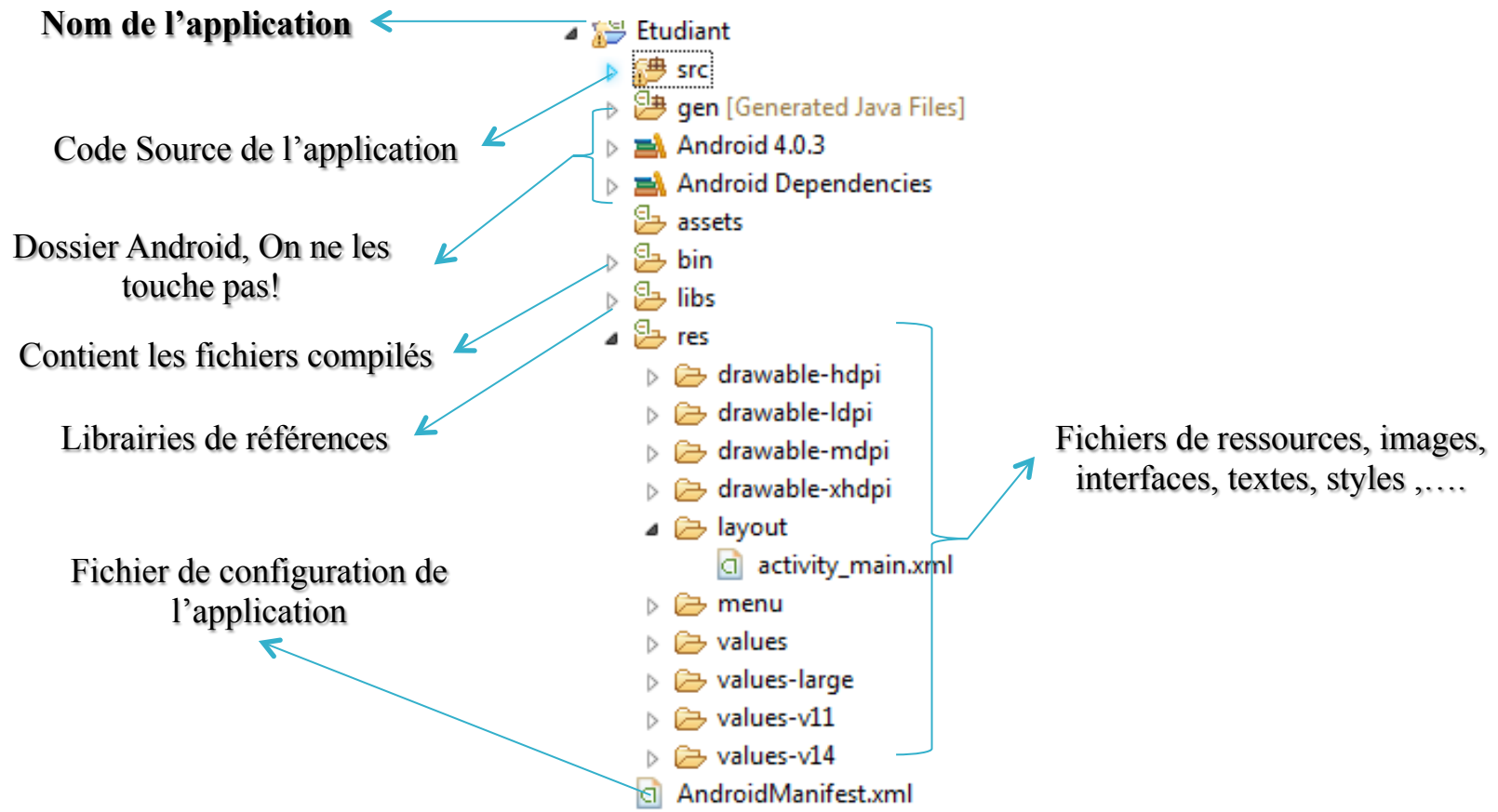
Il existe trois catégories de projets sous Android:

- ❑ **Application Android** : Type primaire des applications Android destiné à être exécuté 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 pouvant être réutilisé par d'autres applications.

# Type d'application Android

1. **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;
2. **Application d'arrière plan (Services):** N'interagit pas avec l'utilisateur, elle s'exécute en tâche de fond.
3. **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;
4. **Widget:** ces applications, représentées sous forme d'un widget de l'écran d'accueil;

# Structure d'une application Android ADT



Référence : <http://developer.android.com/tools/projects/index.html>

# Android Manifest

- Chaque projet Android contient un fichier au format XML nommé manifeste (AndroidManifest.xml).
- Le manifeste permet de définir la structure et les métadonnées d'une application, ses composants et ses prérequis.
- Il contient des balises pour chacun des composants qui constituent l'application (Activities, Services, Content Providers et Broadcast Receivers)

# Android Manifest

- Le manifeste fournit aussi des attributs permettant de spécifier les métadonnées d'une application (Exemple son icône, son thème).
- Il contient également les paramètres de sécurité, les tests unitaires et la définition des prérequis matériels et de plateforme.
- Le fichier manifeste (AndroidManifest.xml) se trouve à la racine du projet.

# Android Manifest

## Strucutre du manifeste:

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

```
<manifest>
```

```
  <uses-permission />
```

```
  <permission />
```

```
  <permission-tree />
```

```
  <permission-group />
```

```
  <instrumentation />
```

```
  <uses-sdk />
```

```
  <uses-configuration />
```

```
  <uses-feature />
```

```
  <supports-screens />
```

```
  <application>
```

```
    <activity>
```

```
      <intent-filter>
```

```
        <action />
```

```
        <category />
```

```
        <data />
```

```
      </intent-filter>
```

```
      <meta-data />
```

```
    </activity>
```

# Android Manifest

## Strucutre du fichier manifeste (suite):

```
<activity-alias>
  <intent-filter> . . . </intent-filter>
  <meta-data />
</activity-alias>
<service>
  <intent-filter> . . . </intent-filter>
  <meta-data/>
</service>
<receiver>
  <intent-filter> . . . </intent-filter>
  <meta-data />
</receiver>
<provider>
  <grant-uri-permission />
  <meta-data />
</provider>
<uses-library />
</application>
</manifest>
```

# Android Manifest

Le plugin ADT eclipse fournit un éditeur visuel permettant de manipuler le Manifest.

The screenshot displays the 'Android Manifest' editor window. The 'Manifest General Attributes' section includes fields for Package (com.edou.gmail.android.rest), Version code (1), Version name (1.0), Shared user id, Shared user label, and Install location. The 'Manifest Extras' section shows a list of permissions, currently containing 'Uses Sdk'. The 'Exporting' section provides options for exporting the application, such as 'Use the Export Wizard' and 'Export an unsigned APK'. The bottom of the window shows a tabbed interface with 'Manifest', 'Application', 'Permissions', 'Instrumentation', and 'AndroidManifest.xml' tabs. A status bar at the bottom indicates '151M of 317M'.

Visualisation graphique

Paramètres de l'application

Gestion des permissions

Visualisation en text xml



# Activité (Activity)

- ▶ Une activité décrit les actions que peuvent effectuer un utilisateur. Elles sont destinées à interagir avec l'utilisateur
- ▶ 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



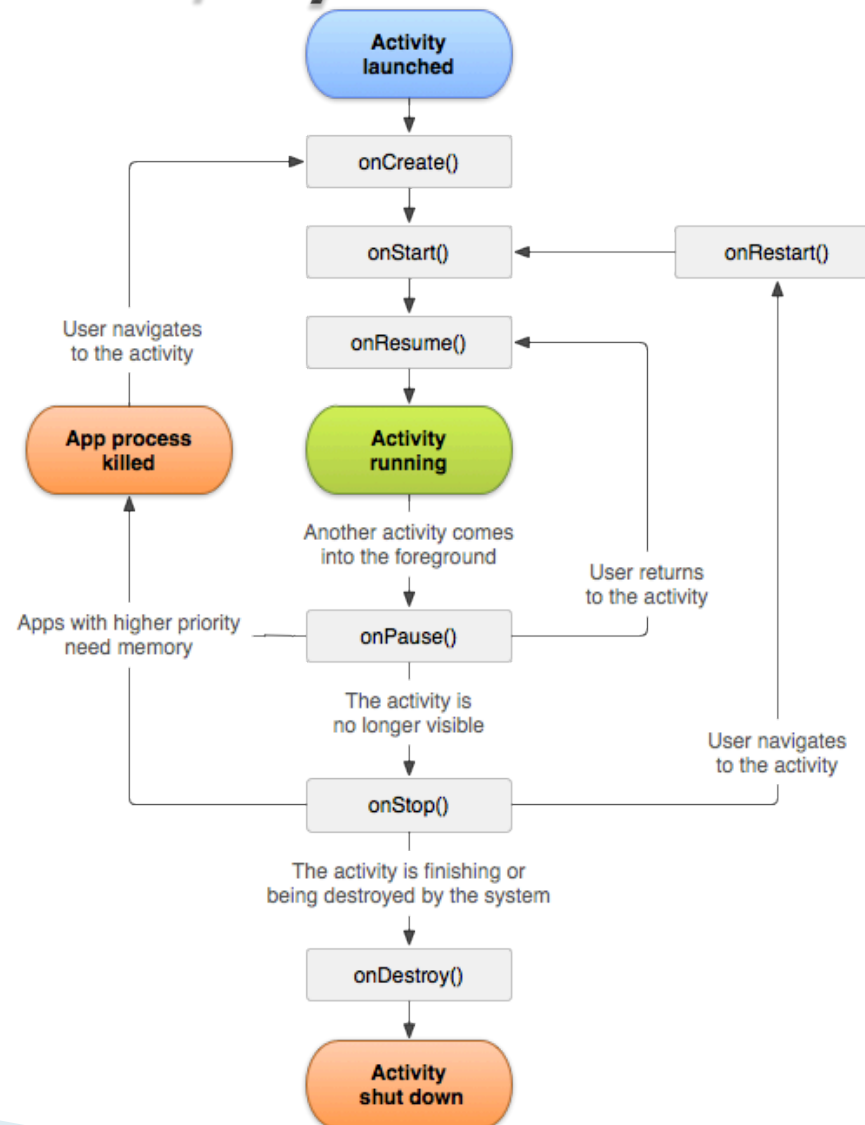
```
Activity a = new Activity();
```

- ▶ L'instanciation d'une activité se fait par les intentions

# Activity : le cycle de vie

- ↘ Une bonne compréhension du cycle de vie de l'Activity permet de garantir que votre application satisfait respecte les bonnes pratiques de développement Android.
- ↘ L'état de chaque Activity est déterminé par sa position dans la pile des Activities:
  1. Active : Quand l'Activity est au sommet de la pile;
  2. En pause : Si l'Activity est visible sans avoir le focus;
  3. Arrêtée: Lorsque l'Activity n'est plus visible;
  4. Inactive : Quand elle est tuée ou qu'elle n'est pas démarrée;

# Activité / cycle de vie



# Activité / Création

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

# Session 2

# Contexte (Context)

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

# Les Intentions (Intent)

Description abstraite d'une action à exécuter:

**Action** : Action à exécuter

**Data** : Données sur lesquelles l'action va opérer

Exemple : Action\_DIAL tel:0641xxxx

Autres attributs :

**Category** : Précise la catégorie de l'activité demandée

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

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

# Résolution d'intention

Il existe deux principaux types d'intentions :

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



# Intention → Instantiation

- ▶ `Intent i = new Intent();`
- ▶ `Intent i = new Intent(action:String)`
- ▶ `Intent i = new Intent(action:String, uri:Uri)`
- ▶ `Intent i = new Intent (context:Context, class:Class<?>)`

# Intentions → Paramètres

- ▶ Les intentions peuvent être utilisées pour transiter des données entre deux activités
- ▶ `putExtra(nom:String, valeur:Object)`
- ▶ `getxxxExtra(nom:String)`
- ▶ `xxx` dépend du type de données: `Int`, `String`, `StringArray`

# Intentions → Actions

- ▶ `addCategory(category: String)`
- ▶ `setDataAndType(uri:Uri, mime:String)`

# Quelques valeurs prédéfinies

## 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
- `android.intent.category.TAB` activité associée dans un onglet d'interface (TabHost)

# Lancer une activité

## 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);
```

# Lancer une activité

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

# Activités / Paramètres (Bundle)

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

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

- Les paramètres sont récupérés dans ce Bundle par ses méthodes :
  - `getBoolean(String)`
  - `getInt(String)`
  - `getBooleanArray(String)`
  - ...

Exemple :

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

# Ressources (Resource)

- Une application Android n'est pas seulement fait de codes mais aussi de ressources statiques (images, sons, 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 & valeurs

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

# Référencer les ressources

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

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

```
<string name="app_name">Mon application</string>
```

```
.....  
</resources>
```

→ R.string.app\_name

```
└─ layout  
  └─ accueil.xml → R.layout.accueil  
  └─ activity_main.xml  
  └─ form_pret.xml  
  └─ login.xml  
  └─ splash.xml
```

# 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

# Interfaces (Layout)

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

Il existe deux possibilités de développer des interfaces :

- Code java
- Fichier XML

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

- Une interface XML peut être associée à une activité (vue) grâce à la méthode `setContentView` (identifiant)

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

# Interfaces / Vues

- **Activity** = Ecran Android (GUI). Il représente les fenêtres ou les écrans affichés. Une Activity contient des views.
- **View** = composant graphique (Widget). C'est la classe de base, tous les contrôles d'interface sont dérivés de View.
- **View Group** : c'est une extension de la classe View. Il contient plusieurs Views enfants. Son extension permet de :
  1. Créer des contrôles composites interconnectées;
  2. Fournir les gestionnaires de layouts pour disposer les contrôles dans une Activities;

# Vues Propriétés communes

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

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

***fill\_parent*** : Remplit toute la place du parent

***wrap\_content*** : remplit la place que nécessite le contenu

***match\_parent*** : remplit la place qui reste dans le parent

`android:background: "@color/blue/#000FFFF"`

`android:visibility : visible / invisible/gone`



# Vues Propriétés communes

## Marges internes

android:layout\_paddingBottom ,  
android:layout\_paddingLeft ,  
android:layout\_paddingRight ,  
android:layout\_paddingTop

## Marges externes

android:layout\_marginBottom ,  
android:layout\_marginLeft ,  
android:layout\_marginRight ,  
android:layout\_marginTop

# Les conteneurs (ViewGroups)

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

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

# Les conteneurs (ViewGroups)

- **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)

# Interfaces graphiques

- ✓ 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.

# Interfaces graphiques

```
public class MainActivity extends Activity {
```

```
    @Override
```

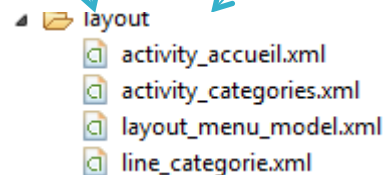
```
    protected void onCreate(Bundle  
savedInstanceState) {
```

```
        super.onCreate(savedInstanceState);
```

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

```
    }
```

```
}
```



# Interfaces graphiques

Les interfaces peuvent être récupérées et modifiées dynamiquement.

```
LinearLayout l = (LinearLayout)findViewById(R.id.id_accueil);  
l.setBackgroundColor(Color.BLACK);
```

# Interfaces graphiques

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

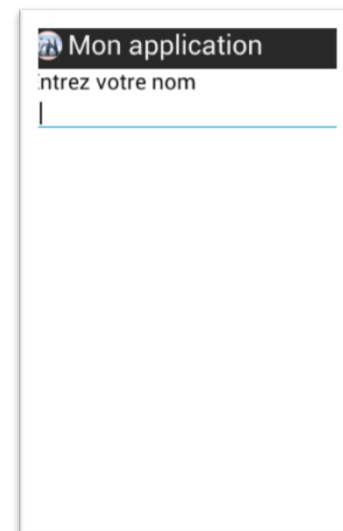
Définit le positionnement linéaire (horizontal ou vertical) des vues filles

```
<?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>
```





# LinearLayout

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

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

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

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



# RelativeLayout

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

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

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

    <EditText
        android:id="@+id/editText1"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:ems="10"
        android:layout_toRightOf="@+id/textView1"
        android:inputType="textPersonName" >

        <requestFocus />
    </EditText>
</RelativeLayout>
```



# TableLayout

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

# Vues de base (Widgets)

- TextView : pour les labels texte;
- EditText : Champ de saisie;
- ListView : Liste de vues horizontales;
- Button : bouton standard;
- CheckBox : case à cocher;
- Spinner : Liste déroulante
- ImageView : image
- RadioButton : radio (choix exclusif)
- TimePicker : Choix de date
- ProgressBar : Bar de progression
- .....

# Widgets → 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

# Widgets → 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);
    }
}
```

# Widgets → Instance

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);
```

# Widgets → Contenus

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();
```



# Widgets → Évènements

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
}
});
```

# Widgets → Validation

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

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

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

# Travaux Pratiques

# »» Session 2

# Les Groupes

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

# Les Groupes

Les groupes utilisent des adaptateurs 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

# Les Groupes

Les groupes utilisent des adaptateurs 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

# Les Adapters

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



# Les Adapters

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

# ArrayAdapter

Adapter concret s'appuyant sur une liste d'objets arbitraires.

Par défaut Android utilise un textview pour afficher le contenu de chaque Objet

Vous pouvez passer la référence à un Textview que vous avez défini

Chaque objet est représenté par sa représentation (toString())

# ArrayAdapter

L'affichage peut être personnalisé :

ImageView

EditText

ListView

...

Il faut redéfinir pour cela surcharger  
la méthode `getView` de la classe

# ArrayAdapter

## Strings.xml

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

## Layout

```
<Spinner android:id="@+id/spinner1"  
android:layout_width="match_parent"  
android:layout_height="wrap_content" android:entries="@array/  
country_arrays" android:prompt="@string/country_prompt" />
```

# ArrayAdapter

```
Spinner sp = (Spinner)
findViewById(R.id.spinner1);

ArrayAdapter<String> dataAdapter = new
ArrayAdapter<String>(this,
android.R.layout.simple_spinner_item,
R.array-string. country_arrays);

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

sp.setAdapter(dataAdapter );
```

# ArrayAdapter

```
public class CategorieAdapter extends BaseAdapter {
    private Context mContext;
    List<Categorie> categories;

    public CategorieAdapter(Context c, List<Categorie> categories) {
        mContext = c;
        this.categories = categories;
    }
    // create a new ImageView for each item referenced by the Adapter
    public View getView(int position, View convertView, ViewGroup parent) {
        LayoutInflater inflater = (LayoutInflater) mContext
            .getSystemService(Context.LAYOUT_INFLATER_SERVICE);

        View gridView;
        Categorie c = categories.get(position);

        if (convertView == null) {

            gridView = new View(mContext);

            // get layout
            gridView = inflater.inflate(R.layout.line_categorie, null);
            // set value into textview
            TextView nom = (TextView) gridView
                .findViewById(R.id.nom);
            TextView desc = (TextView) gridView
                .findViewById(R.id.desc);

            nom.setText(c.getNom());
            desc.setText(c.getDescription());

        } else {
            gridView = (View) convertView;
        }

        return gridView;
    }
}
```

# ListView

- ▶ Une listview est une vue de groupe qui peut afficher des éléments sous forme d'une liste déroulante horizontale
- ▶ Une listview utilise un Adapter pour remplir son contenu en fonction de la collection d'objets qui lui est fournie



# ListView

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:id="@+id/id_accueil"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <ListView
        android:id="@+id/listView1"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_alignParentLeft="true"
        android:layout_alignParentTop="true"
        android:layout_marginTop="16dp" >
    </ListView>

</RelativeLayout>
```



# Listview

## ▶ SimpleAdapter

```
ListView lst = (ListView) findViewById(R.id.listview1);  
ArrayAdapter<Categorie> dataAdapter = new  
ArrayAdapter<Categorie>(this, android.R.layout.simple_spinner_item,  
categories);  
lst.setAdapter(dataAdapter );
```

## ▶ Adapter personnalisé

```
CategorieAdapter myAdapter = new CategorieAdapter(context,  
categories);  
Lst.setAdapter(myAdapter )
```

# Listview

- ▶ La déclaration d'une listview peut être omise en utilisant une ListActivity

```
public class ListViewLoader extends ListActivity
{
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        // Create a progress bar to display while the list loads
        ProgressBar progressBar = new ProgressBar(this);
        progressBar.setLayoutParams(new LayoutParams(LayoutParams.WRAP_CONTENT,
            LayoutParams.WRAP_CONTENT, Gravity.CENTER));
        progressBar.setIndeterminate(true);
        getListView().setEmptyView(progressBar);

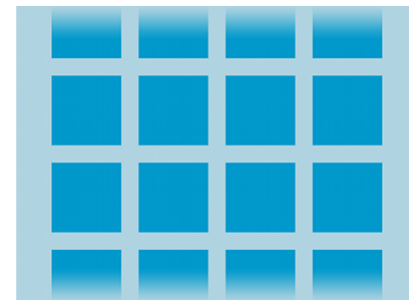
        ArrayAdapter<Categorie> dataAdapter = new ArrayAdapter<Categorie>(this,
            android.R.layout.simple_spinner_item, categories);

        // Must add the progress bar to the root of the layout
        ViewGroup root = (ViewGroup) findViewById(android.R.id.content);
        root.addView(progressBar);

        setListAdapter(dataAdapter );
    }
}
```

# GridView

- ▶ Une gridview est une vue de groupe disposant des éléments déroulantes de manière bi dimensionnel dans une interface
- ▶ Une gridview utilise des adapters pour remplir son contenu en fonction de la collection d'objets qui lui est fournie.



# GridView

Une griview peut être utilisée comme racine d'une interface

```
<?xml version="1.0" encoding="utf-8"?>  
<GridView xmlns:android="http://schemas.android.com/apk/res/android"  
    android:id="@+id/gridview"  
    android:layout_width="fill_parent"  
    android:layout_height="fill_parent"  
    android:columnWidth="90dp"  
    android:numColumns="auto_fit"  
    android:verticalSpacing="10dp"  
    android:horizontalSpacing="10dp"  
    android:stretchMode="columnWidth"  
    android:gravity="center"  
>
```

# GridView

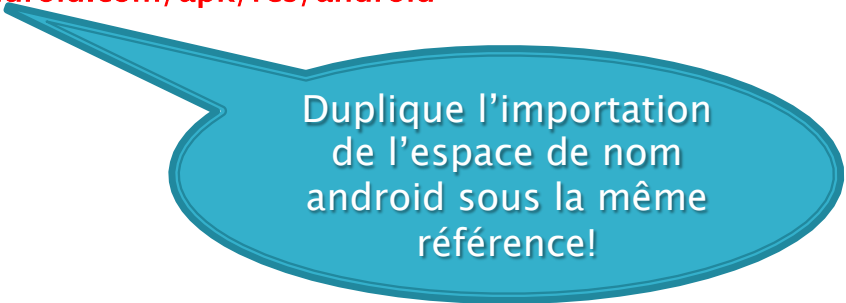
Peut être utilisée comme sous vue d'un gabarit

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

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"  
    xmlns:tools="http://schemas.android.com/tools"  
    android:id="@+id/id_accueil"  
    android:layout_width="match_parent"  
    android:layout_height="match_parent">
```

```
<GridView xmlns:android="http://schemas.android.com/apk/res/android"  
    android:id="@+id/gridview"  
    android:layout_width="fill_parent"  
    android:layout_height="fill_parent"  
    android:columnWidth="90dp"  
    android:numColumns="auto_fit"  
    android:verticalSpacing="10dp"  
    android:horizontalSpacing="10dp"  
    android:stretchMode="columnWidth"  
    android:gravity="center"  
>
```

```
</RelativeLayout>
```



Duplique l'importation  
de l'espace de nom  
android sous la même  
référence!

# GridView

## ▶ SimpleAdapter

```
GridView grd = (GridView) findViewById(R.id.gridview1);  
ArrayAdapter<Categorie> dataAdapter = new  
ArrayAdapter<Categorie>(this, android.R.layout.simple_spinner_item,  
categories);  
grd.setAdapter(dataAdapter );
```

## ▶ Adapter personnalisé

```
CategorieAdapter myAdapter = new CategorieAdapter(context,  
categories);  
grd.setAdapter(myAdapter )
```

# Groupes → Evènements

Les groupes permettent la gestion d'évènement sur les éléments qu'ils contiennent

```
gridview.setOnItemClickListener(new OnItemClickListener() {  
    public void onItemClick(AdapterView<?> parent, View v, int  
position, long id) {  
        // Ce qui se passe quand on clique sur un élément  
    }  
});
```

```
gridview.setOnLongClickListener(new OnLongClickListener() {  
@Override  
public boolean onLongClick(View v) {  
    //Ce qui se passe quand on reste appuyé sur un élément  
}  
});
```

# Permission

- ▶ Elles renseignent le système sur les besoins de l'application en terme de ressources sur le terminal;
- ▶ Mécanisme de sécurité permettant au système de garantir la transparence sur les données de l'utilisateur
- ▶ Elles apparaissent dans le fichier Android Manifest et sont visibles par l'utilisateur au moment de l'installation de l'application
- ▶ Exemple:
  - Capteurs (GPS, NFC, Bluetooth,...)
  - Les accès aux contacts et à l'agenda du téléphone
  - Les modifications de paramètres (orientation, fond d'écran ...)
  - Les appels téléphoniques
  - Le réseau (dont l'accès à Internet)
  - Le matériel (Caméra, ...)



# Permission

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

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

Si une permission a été omise et vous essayez d'utiliser la ressource correspondante, vous aurez une erreur à l'exécution!

Les permissions peuvent être ajoutées en utilisant l'outil graphique d'édition du Manifest fourni par ADT

# Connectivité

- ▶ Android fournit des mécanismes permettant aux applications d'utiliser la connectivité pour accomplir certaines tâches :
  - Internet (WIFI, 3G,...)
  - Bluetooth
  - NFC
- ▶ Pour accéder à ces ressources, il est obligatoire de demander explicitement la permission associée

# Internet

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

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

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

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

Réf : <http://hc.apache.org>

# Taches Asynchrones

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

# Tâches Asynchrones

```
protected class LongTask extends AsyncTask<String, Void, Void> {
    @Override
    protected void onPreExecute() {
        super.onPreExecute();
        //Avant l'execution du processus long (Thread principal)
    }
    @Override
    protected void onPostExecute(Void resp) {
        super.onPostExecute(resp);
        //Après l'execution du processus long (Thread principal)
    }
    @Override
    protected Void doInBackground(String... arg0) {
        //Execution du processus long (Thread secondaire)
        return null;
    }
}
```

# Session 3

# Android Localisation

- ▶ Le SDK offre au développeur la possibilité d'ajouter les fonctionnalités liées à la géolocalisation dans leurs applications.
  - Position géographique de l'utilisateur
  - Places
  - Géocodage (Geocoding, Reverse Geocoding)
- ▶ `Android.location` : Package contenant un ensemble de classes et d'interfaces supportant la géolocalisation :
- ▶ Interfaces : `GpsStatus.Listener`, `LocationListener`
- ▶ Classes: `Address`, `Criteria`, `Geocoder`, `Location`, `LocationManager`, `LocationProvider`

# Android Localisation

- ▶ Deux possibilités
  - GPS
  - NETWORK (WIFI, 3G)

Localisation  
approximative : WIFI, 3G

- ▶ Permissions :

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

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

Localisation précise : GPS



# Android Localisation

## LocationListener:

- ▶ Interface vous permettant d'abonner aux évènements de géolocalisation
  - Détection d'une nouvelle position
  - Changement de statut du fournisseur (activation, désactivation,...)
- ▶ Disposer d'une classe implémentant cette interface
  1. Vous créez une nouvelle classe
  2. Votre activité implémente cette interface

```
class MyListener implements
LocationListener{
.....
//Implémenter les méthodes
}
```

```
public class HomeActivity
extends Activity implements
LocationListener {
.....
}
```

# Android Localisation

LocationManager :

- ▶ dispose d'APIS vous permettant d'accéder au service de location du terminal
- ▶ Ne peut être instanciée directement (comme c'est le cas pour la plupart des services système Android)
- ▶ Une instance est obtenue par l'appel à la méthodes getSystemService (Context.Location\_Service) qui retourne une instance de LocationManager
- ▶ `lm = (LocationManager)  
this.getSystemService (LOCATION_SERVICE) ;`

# Android Localisation

- ▶ Demandez la position courante de l'utilisateur :

```
if (lm.isProviderEnabled(LocationManager.GPS_PROVIDER))  
    lm.requestLocationUpdates(LocationManager.GPS_PROVIDER, 10000, 0,  
        locationManager);  
else  
    lm.requestLocationUpdates(LocationManager.NETWORK_PROVIDER, 10000, 0,  
        locationManager);
```

- ▶ Quand une nouvelle position est disponible, le système appelle la méthode `onLocationChanged` du Listener

```
@Override public void onLocationChanged(Location location)  
{ // TODO Auto-generated method stub Toast.makeText(this, "My  
Location " + location.getLongitude(),  
    Toast.LENGTH_LONG).show(); }
```

# Android Localisation

## ▶ Address

Permettant de représenter une adresse (rue, ville, code postal,.....)

## ▶ Geocoder : Convertit une position GPS en adresse et réciproquement

```
Geocoder geocoder = new Geocoder(context, Locale.getDefault());  
List<Address> addresses = geocoder.getFromLocation(loc.getLatitude(),  
loc.getLongitude(), 1);
```

## ▶ Bonne pratique

- Lancer les requêtes de mises à jour de la position dans `onResume`
- Stopper les demandes de mises à jour quand l'activité n'est plus visible (`onPause`, `onStop`)
- Faites le géocodage dans un thread secondaire car cette opération est déléguée par la suite aux serveurs de Google

# Google MAPS

- ▶ Android supporte également les fonctionnalités de Google Maps et met à disposition des développeurs un ensemble d'APIs permettant d'ajouter des fonctionnalités liées à ce service à leurs applications
  - Afficher la MAPS
    - MapView
    - Markers
- ▶ Ces fonctionnalités sont disponibles via Google Play Library
- ▶ L'utilisation de Google MAPS nécessite d'ajouter une référence à cette librairie à votre projet → Ces fonctionnalités ne sont pas natives dans le SDK, ce sont des services de Google

# Google MAPS

## ▶ Permissions

```
<permission android:name=« your.package.name.permission.MAPS_RECEIVE"  
android:protectionLevel="signature" />
```

```
<uses-permission android:name=« your.package.name.permission.MAPS_RECEIVE" />  
<uses-permission android:name="android.permission.INTERNET" />  
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />  
<uses-permission  
android:name="com.google.android.providers.gsf.permission.READ_GSERVICES" />  
<uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION" />  
<uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />
```

## ▶ Capacité du terminal :

```
<uses-feature android:glEsVersion="0x00020000" android:required="true" />
```

## ▶ Metadonnées de l'activité :

```
<meta-data android:name="com.google.android.maps.vx (2,3).API_KEY"  
android:value="your_apikey" />
```

# Android Persistence

- ▶ Android offre plusieurs possibilités de persistance de données :
  - Système de fichiers (xml, .txt, doc, ...)
  - Préférences
  - SQLite

# 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`
- ▶ Le SDK dispose d'un outil permettant de manipuler les bases de données du téléphone via une fenêtre de commande
  - `tools/sql3`



# 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

- ▶ Packages

**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 :

Context

Nom de la base

CursorFactory

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

- ▶ Ouvrir la base de données en écriture

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

Version de la base

- ▶ 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

# CursorAdapter

- ▶ Vous pouvez utiliser les curseurs pour créer des adaptateurs pour les vues de groupes

```
public class CategorieAdapter extends CursorAdapter {
    public CategorieAdapter(Context context, Cursor c, boolean autoRequery)
    {
        super(context, c, autoRequery);
    }
    @Override
    public void bindView(View view, Context context, Cursor cursor)
    {
        //Les informations à ajouter dans la vue du groupe
    }

    @Override
    public View newView(Context context, Cursor cursor, ViewGroup parent)
    {
        //La vue à jouter dans le groupe
    }
}
```



# CursorAdapter

- ▶ Vous pouvez utiliser les curseurs pour créer des adaptateurs pour les vues de groupes

```
@Override
public void bindView(View view, Context context, Cursor cursor)
{
    TextView textViewPersonName = (TextView) view.findViewById(R.id.nom);
    textViewPersonName.setText(cursor.getString(cursor.getColumnIndex("nom")));
    TextView textViewPersonPIN = (TextView) view.findViewById(R.id.description);
    textViewPersonPIN.setText(cursor.getString(cursor.getColumnIndex("description")));
}
```

```
@Override
public View newView(Context context, Cursor cursor, ViewGroup parent) {
    // TODO Auto-generated method stub
    LayoutInflater inflater = LayoutInflater.from(parent.getContext());
    View retView = inflater.inflate(R.layout.line_categorie, parent, false);
    return retView;
}
```

# CursorAdapter

- ▶ Vous pouvez utiliser les curseurs pour créer des adaptateurs pour les vues de groupes

```
@Override
public void bindView(View view, Context context, Cursor cursor)
{
    TextView textViewPersonName = (TextView) view.findViewById(R.id.nom);
    textViewPersonName.setText(cursor.getString(cursor.getColumnIndex("nom")));
    TextView textViewPersonPIN = (TextView) view.findViewById(R.id.description);
    textViewPersonPIN.setText(cursor.getString(cursor.getColumnIndex("description")));
}
```

```
@Override
public View newView(Context context, Cursor cursor, ViewGroup parent) {
    // TODO Auto-generated method stub
    LayoutInflater inflater = LayoutInflater.from(parent.getContext());
    View retView = inflater.inflate(R.layout.line_categorie, parent, false);
    return retView;
}
```

# SimpleCursorAdapter

- ▶ Adaptateur générique permettant de mapper les colonnes d'un curseur à des vues puis les attacher à une vue de groupe

```
SimpleCursorAdapter adapter = new SimpleCursorAdapter(  
    HomeActivity.this, R.layout.line_categorie,  
    cursor, new String[] { "nom", "description" },  
    new int[] { R.id.nom, R.id.description});  
setListAdapter(adapter );
```

- Cette méthode est dépréciée à partir de l'API 11 car s'exécutant dans le thread principal
- Pour un projet avec une version minimum 11, utilisez :

```
SimpleCursorAdapter adapter = new SimpleCursorAdapter(  
    HomeActivity.this, R.layout.line_categorie,  
    cursor, new String[] { "nom", "description" },  
    new int[] { R.id.nom, R.id.description}, 1);
```

# Service

- ▶ Un service est une composante d'une application ne nécessitant pas d'interaction avec les utilisateurs et dans lequel une application peut effectuer des processus longs de manière transparente pour l'utilisateur final.
- ▶ Un service doit être déclaré dans le Manifest du projet en utilisant la balise `<service>`
- ▶ Un service n'est pas un processus séparé, il s'exécute dans le contexte de l'application
- ▶ Un service n'est pas un thread

# Services

- ▶ Il existe deux façons de démarrer un service
  - Le client appelle la méthode `startService` auquel cas le système récupère le service en question et appelle sa méthode `onCreate` si nécessaire sinon il appelle la méthode `onStartCommand`. Le service s'exécute tant qu'il ne s'arrête lui-même (`stopSelf()`) ou le client ne l'arrête (`Context.stopService()`).
  - On peut aussi obtenir une connection persistante à un service en appelant sa méthode `Context.bindService()`, auquel cas le système récupère le service et renvoie un objet `IBinder` au client. Par cette approche, le service est démarré mais le système n'appelle pas la méthode `onStartCommand()`. Le service tourne tant qu'il existe au moins une connexion.

# Services

```
public class MyService extends Service
{
public MyService()
{
}
private final IBinder mBinder = new LocalBinder();
public class LocalBinder extends Binder { MyService getService() {
return MyService.this; }
}
@Override public IBinder onBind(Intent intent)
{
return mBinder;
}
@Override
public void onCreate() {
//Nous faisons notre travail ici }
@Override public int onStartCommand(Intent intent, int flags, int startId) {
return START_STICKY; }
}
```

# Services

- ▶ Démarrer le service en mode interactif :

```
private MyService mBoundService;
    private ServiceConnection mConnection = new ServiceConnection()
    {
    public void onServiceConnected(ComponentName className, IBinder service) {
        }
    public void onServiceDisconnected(ComponentName className) {
    } };
void doBindService() { }
void doUnbindService() { }
@Override
protected void onDestroy() { }
}
```

# Services

- ▶ Démarrer le service sans maintenir une connexion :

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

Dans ce cas, le système démarre le service s'il n'a pas encore été démarré



# Notification

- ▶ Les notifications sont des mécanismes permettant d'alerter l'utilisateur sur des événements s'effectuant en tâche de fond.
- ▶ Elles apparaissent dans la barre de taches et peuvent être de trois types:
  - Une icône persistante dans la barre de statut et accessible via le lanceur (the launcher)! Quand l'utilisateur le sélectionne une intention peut être déclenchée
  - Clignote les LEDs de notifications
  - Alerter l'utilisateur en allumant son écran, jouer une sonnerie ou vibrer le terminal

# Notification

## Créer une notification

### ► Méthode dépréciée à partir de l'API 11

```
private void showNotification() {  
    // In this sample, we'll use the same text for the ticker and the expanded notification  
    CharSequence text = getText(R.string.notif_message);  
  
    // Set the icon, scrolling text and timestamp  
    Notification notification = new Notification(R.drawable.my_icon, text,  
        System.currentTimeMillis());  
  
    // The PendingIntent to launch our activity if the user selects this notification  
    PendingIntent contentIntent = PendingIntent.getActivity(this, 0,  
        new Intent(this, LocalServiceActivities.Controller.class), 0);  
  
    // Set the info for the views that show in the notification panel.  
    notification.setLatestEventInfo(this, getText(R.string.notif_mesage),  
        text, contentIntent);  
  
    // Send the notification.  
    manager.notify(NOTIFICATION, notification);  
}
```

# Notification

## Créer une notification

- ▶ Nouvelle approche

```
Notification notif = new  
Notification.Builder(this) .setContentTitle(getString  
(R.string.app_name)) .setContentText(getString(R.stri  
ng.app_name)) .setSmallIcon(R.drawable.ic_launcher)  
.build();
```

# NotificationManager

Les notificationManagers sont utilisés pour gérer les notifications :

- ▶ Afficher une notification
- ▶ Enlever une notification dans la barre des statuts

```
NotificationManager manager =  
(NotificationManager) getSystemService (NOTIFICATION_SE  
RVICE) ;  
    manager.notify (CODE, notif) ;  
    manager.cancel (0) ;
```