

Introduction à la programmation sous Android

Christophe Renaud

M2 Informatique

Version 2.4 du 20/11/2016

Objectifs du cours

- Connaître les bases de la programmation sous Android
 - Environnement de développement (Android Studio)
 - Architecture d'une application
 - Modèle d'exécution

Plan du cours

- Introduction
- Architecture d'une application Android
- Les activités
- Définir une interface graphique
- Les intentions
- Les menus
- Les listes
- Les content providers

Android (1)

- Système d'exploitation à destination des dispositifs mobiles
 - Téléphones, tablettes, téléviseurs, montres, lunettes, voitures
- Caractéristiques :
 - Opensource (licence Apache), gratuit, flexible
 - Basé sur un noyau linux
 - Inclut les applications de base (téléphone, sms, carnet d'adresse, navigateur, etc.)
 - Un ensemble important d'API (OpenGL, media, etc ...)
 - Un SDK basé sur un sous-ensemble de JAVA (autres langages disponibles : C, C++, ...)
 - Une machine virtuelle (Dalvik) qui exécute la majorité des applications
 - Remplacée par ART depuis la version 5.0 d'Android

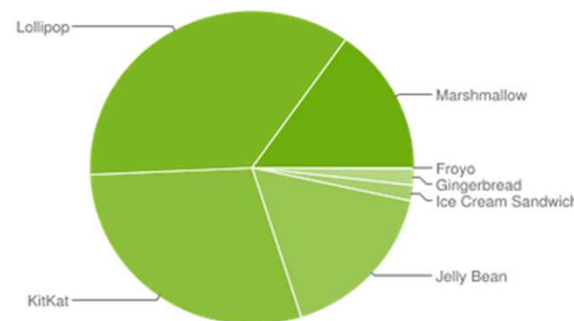
Android (2)

■ Historique :

- Créé en 2005 par la société Android
- Rachat en 2007 par Google
- Plus de 20 versions depuis la 1.0 (Apple Pie) en 2008 jusqu'à la 7.0 (Nougat) en 08/2016.

Version	Codename	API	Distribution
2.2	Froyo	8	0.1%
2.3.3 - 2.3.7	Gingerbread	10	1.7%
4.0.3 - 4.0.4	Ice Cream Sandwich	15	1.6%
4.1.x	Jelly Bean	16	6.0%
4.2.x		17	8.3%
4.3		18	2.4%
4.4	KitKat	19	29.2%
5.0	Lollipop	21	14.1%
5.1		22	21.4%
6.0	Marshmallow	23	15.2%

Data collected during a 7-day period ending on August 1, 2016.
Any versions with less than 0.1% distribution are not shown.



Source : developer.android.com

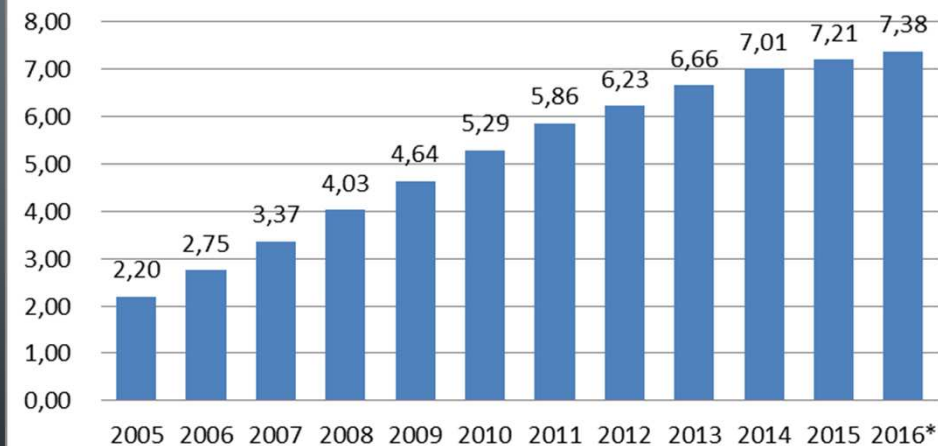
Version	Codename	API	Distribution
2.2	Froyo	8	0.2%
2.3.3 - 2.3.7	Gingerbread	10	4.1%
4.0.3 - 4.0.4	Ice Cream Sandwich	15	3.7%
4.1.x	Jelly Bean	16	12.1%
4.2.x		17	15.2%
4.3		18	4.5%
4.4	KitKat	19	39.2%
5.0	Lollipop	21	15.9%
5.1		22	5.1%

Data collected during a 7-day period ending on September 7, 2015.
Any versions with less than 0.1% distribution are not shown.

Android (3)

■ Pourquoi développer des applications mobiles ?

Nombre d'abonnés au mobile dans le monde
(en milliards)



Source : http://www.itu.int/en/ITU-D/Statistics/Documents/statistics/2016/ITU_Key_2005-2016_ICT_data.xls

Device type	2014	2015	2016	2017
Traditionnels PCs (desk based and notebooks)	277,118	252,881	243,628	236,341
Ultramobile (Premium)	36,699	53,452	74,134	90,945
PC Market	313,817	306,333	317,762	327,285
Ultramobiles (Tablets and Clamshells)	227,080	236,778	257,985	276,026
Computing Devices Market	540,897	543,111	575,747	603,311
Mobile Phones	1,878,968	1,943,952	2,017,861	2,055,954
Total Devices Market	2,419,864	2,487,062	2,593,608	2,659,265

Ventes en centaines de milliers d'unités - source : (mars 2015) <http://www.gartner.com/newsroom/id/3010017>

Android (4)

- Pourquoi développer sous Android ?

Worldwide Smartphone Sales to End Users by Operating System in 1Q16 (Thousands of Units)

Operating System	1Q16	1Q16	1Q15	1Q15
	Units	Market Share (%)	Units	Market Share (%)
Android	293,771.2	84.1	264,941.9	78.8
iOS	51,629.5	14.8	60,177.2	17.9
Windows	2,399.7	0.7	8,270.8	2.5
Blackberry	659.9	0.2	1,325.4	0.4
Others	791.1	0.2	1,582.5	0.5
Total	349,251.4	100.0	336,297.8	100.0

source (mai 2016) : <http://www.gartner.com/newsroom/id/3323017>

Android (5)

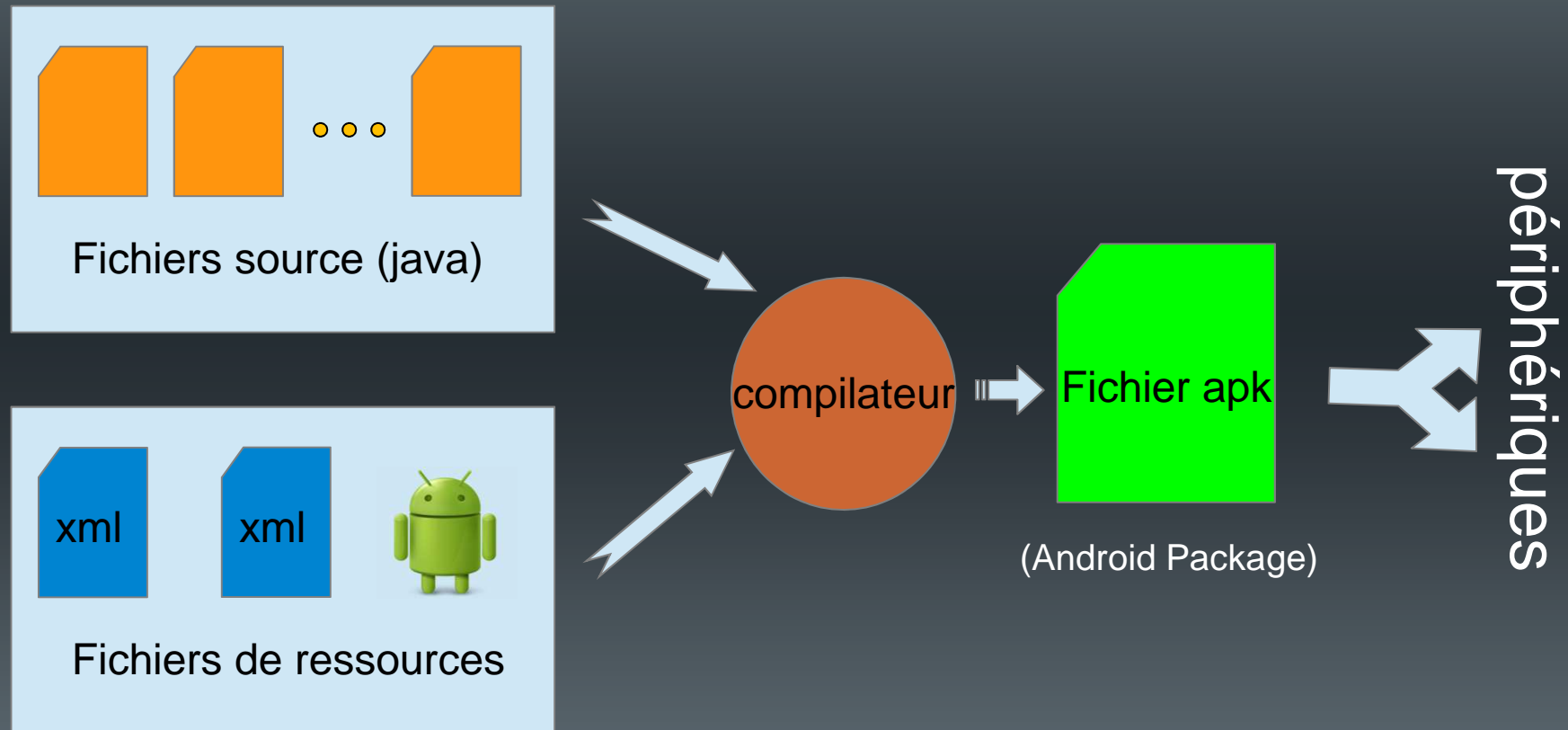
■ Les contraintes

- Hétérogénéité du matériel
 - Processeurs, mémoire
 - Écrans
 - Dispositifs spécialisés
- Puissance et mémoire limitées
- Interface tactile
- Connectivité à internet (disponibilité, rapidité, ...)
- Développement extérieur au périphérique

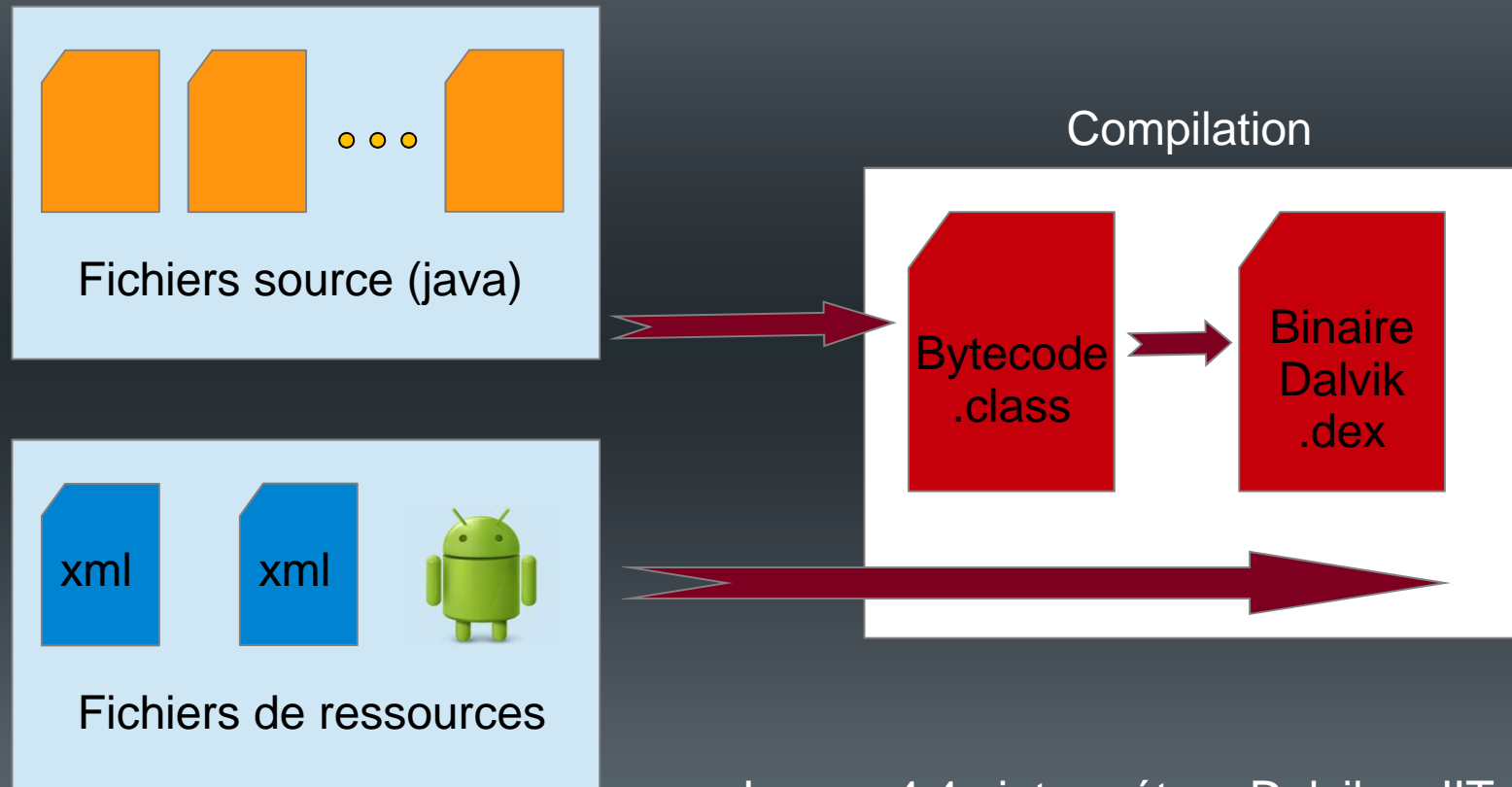
Plan du cours

- Introduction
- Architecture d'une application Android
- Les activités
- Définir une interface graphique
- Les intentions
- Les menus
- Les listes
- Les content providers

Schéma de développement



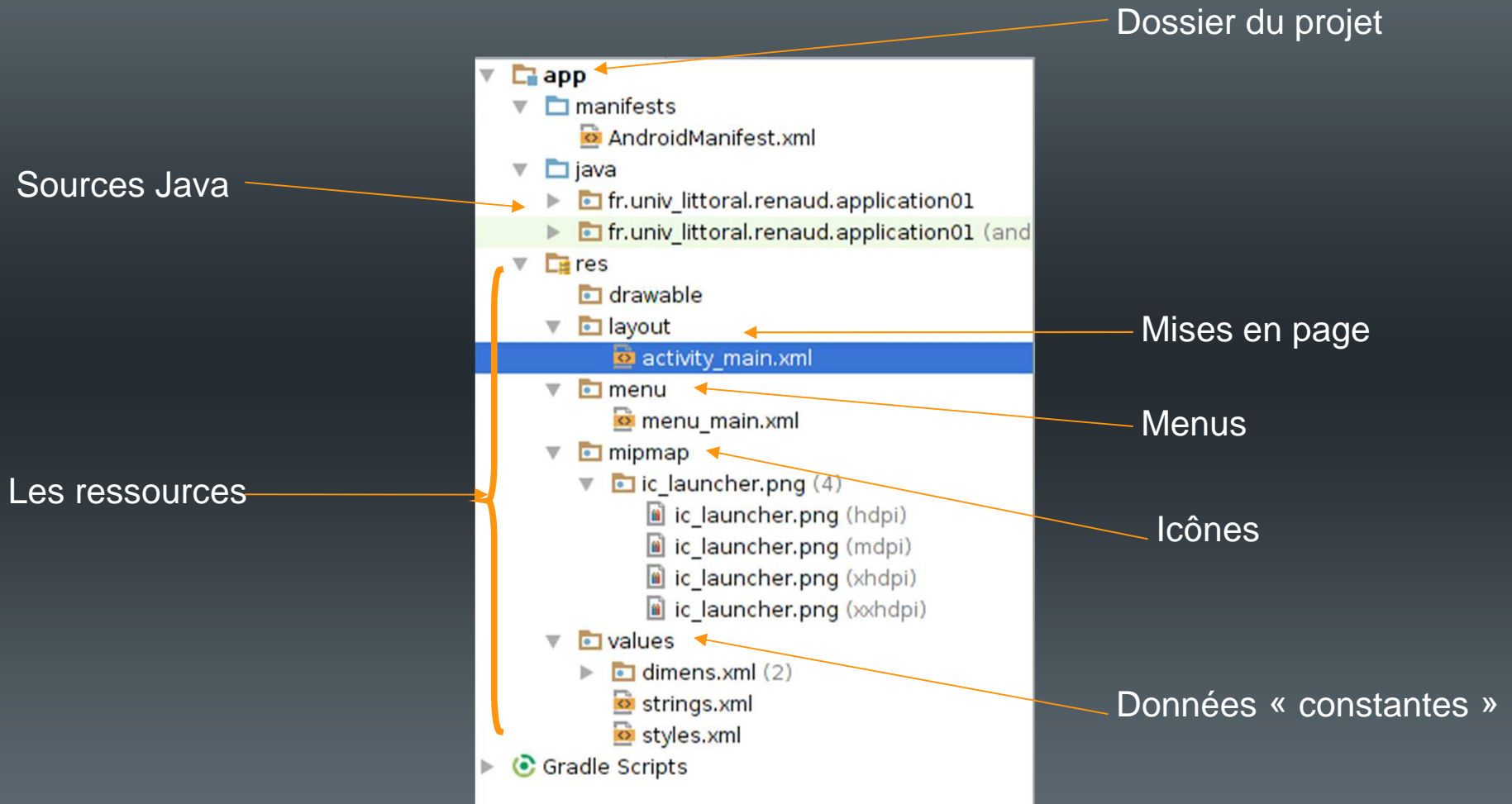
La compilation



Jusque 4.4 : interpréteur Dalvik + JIT compilation de parties « critiques »

À partir de 5.0 : ART (compilation en code natif sur le support)

Architecture d'un projet

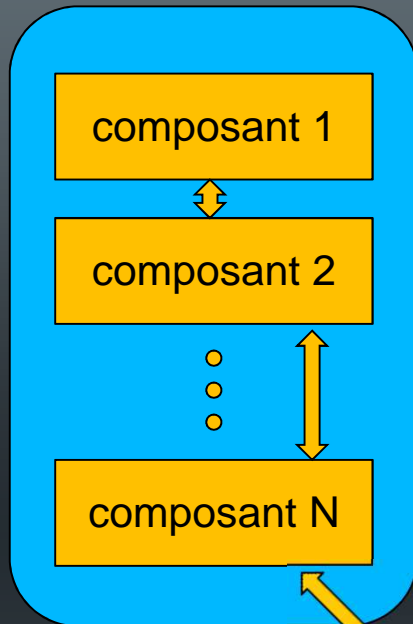


(sous Android Studio)

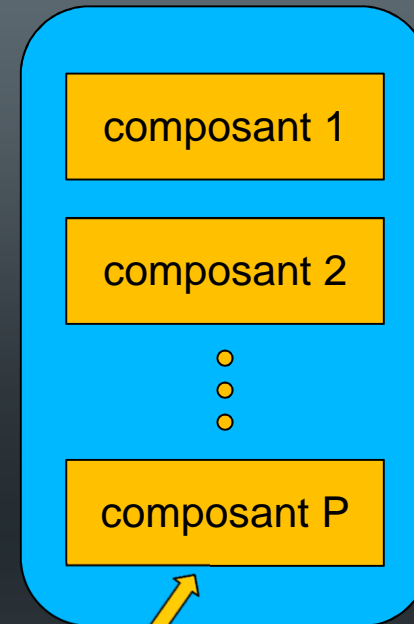
Les éléments d'une application

- Une application = {composants}
- Les composants :
 - Existent de manière indépendante
 - Vus comme autant de points d'entrée par le système
 - Pas de « main » dans une application
- Liés au design d'Android :
 - Toute application doit pouvoir démarrer un composant d'une autre application (sous réserve de droits) et récupérer ses « résultats »

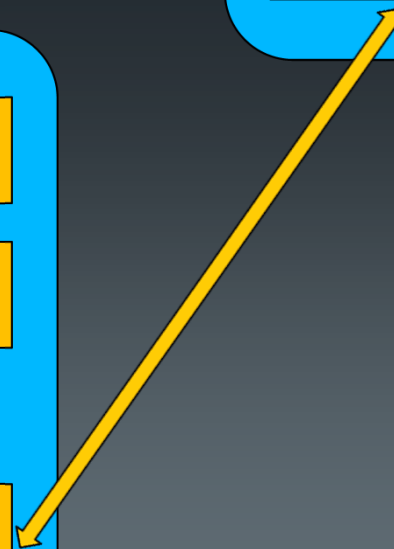
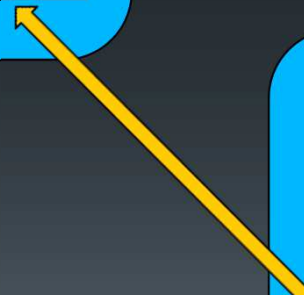
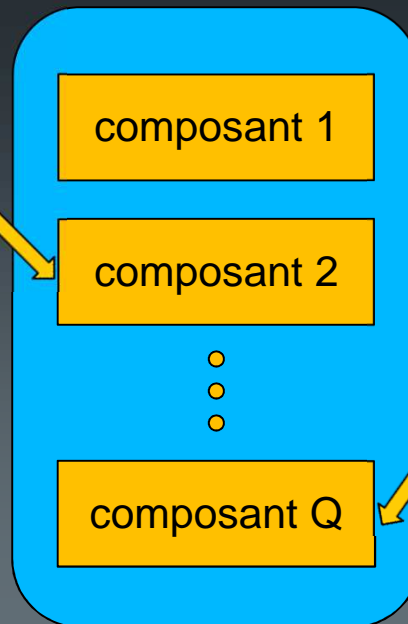
MonApplication



Application Y

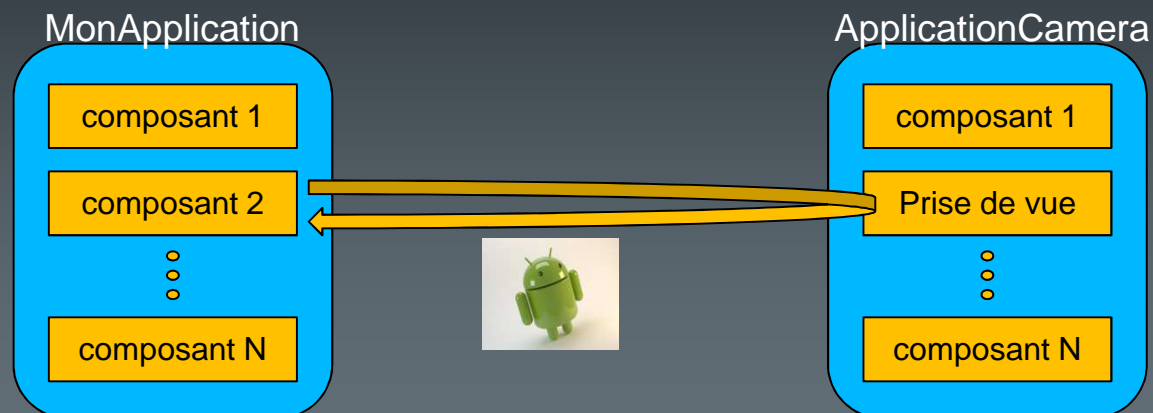


Application X



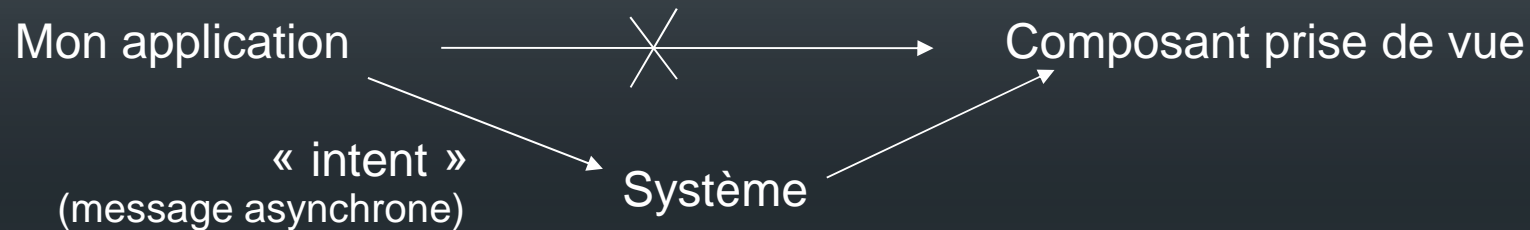
Exemple

- Mon application = application d'effets sur un portrait de l'utilisateur
 - Difficulté : écrire le code de gestion de l'appareil photo embarqué
 - Android :
 - démarrage d'un composant existant permettant la prise de vue
 - Récupération de l'image



Remarques

- Problèmes de droits :



- Problèmes d'information :

- Le système doit connaître le rôle particulier de certains composants
 - Ce sont les applications qui enregistrent ces informations auprès du système

Les composants

- Les activités (Activity)
 - Un écran avec une interface utilisateur et un contexte
- Les services (Service)
 - Composant sans écran, qui tourne en fond de tâche (lecteur de musique, téléchargement, ...)
- Les fournisseurs de contenu (ContentProvider)
 - I/O sur des données gérées par le système ou par une autre application
- Des récepteurs d'intentions (BroadcastReceiver)
 - Récupération d'informations générales
 - arrivée d'un sms, batterie faible, ...

Les interactions

- Les intentions (*Intent*)
 - Permet d'échanger des informations entre composants
 - Démarrage d'un composant en lui envoyant des données
 - Récupération de résultats depuis un composant
 - Recherche d'un composant en fonction d'un type d'action à réaliser
- Les filtres d'intentions (<intent-filter>)
 - Permet à un composant d'indiquer ce qu'il sait faire
 - Permet au système de sélectionner les composants susceptibles de répondre à une demande de savoir-faire d'une application

AndroidManifest.xml

- Description de l'application
 - Liste des composants
 - Niveau minimum de l'API requise
 - Liste des caractéristiques physiques nécessaires
 - Évite d'installer l'application sur du matériel non compatible (gestion de la visibilité sur Google Play)
 - Liste des permissions dont l'application a besoin
 - Liste des autres API nécessaires
 - ex. Google Map
 - Etc.
- Généré automatiquement par Android Studio

Example

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="fr.univ_littoral.renaud.bidon" >

    <application
        android:allowBackup="true"
        android:icon="@mipmap/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>
```

Les ressources

- Ressources = toutes les données (autres que le code) utilisées par l'application
- Rangées dans le dossier **res**, puis incluses dans l'apk
 - res/drawable et res/mipmap (images en différentes résolutions)
 - Layout (description en XML des interfaces)
 - Menus (description en XML des menus)
 - Values (définitions en XML des constantes utilisées par l'application : chaînes, tableaux, valeurs numériques, etc.)

strings.xml

- Fichier ressources, contenant toutes les chaînes constantes
 - Principalement utilisées pour l'interface

Type de la
constante

```
<resources>
  <string name="app_name">MyApplication</string>
  <string name="hello_world">Hello world!</string>
  <string name="action_settings">Settings</string>
</resources>
```

Nom de la constante
(permet l'appel depuis
l'application ou un autre
fichier XML)

Valeur de la constante

Internationalisation

■ Objectif :

- Disposer de plusieurs versions des textes, libellés, etc utilisés par l'application
- Choix automatique des textes en fonction de la configuration du périphérique

■ Principe

- Dupliquer le fichier strings.xml : 1 version par langue supportée
- Stocker chaque version dans un dossier spécifique
 - values-xx (ex. values-en, values-fr, ...)
- Géré via Android Studio

```
app/  
  res/  
    values/  
      strings.xml  
    values-en/  
      strings.xml  
    values-fr/  
      strings.xml
```

La classe R

- Classe générée par l'IDE

- Permet l'accès aux ressources
- Créée à partir de l'arborescence présente dans le dossier **res**
- Elle contient des classes internes dont les noms correspondent aux différents types de ressources (drawable, layout, ...)
- Elle contient des propriétés permettant de représenter l'ensemble des ressources de l'application

- Utilisation en Java :

- R.type.identificateur

```
<resources>  
  <string name="app_name">MyApplication</string>  
  
  <string name="hello_world">Hello world!</string>  
  <string name="action_settings">Settings</string>  
</resources>
```

R.string.app_name

R.string.hello_world

R.string.action_settings

Référencement des ressources en XML

- Forme générale : @type/identificateur

```
<resources>  
  <string name="app_name">MyApplication</string>  
  <string name="hello_world">Hello world!</string>  
  <string name="action_settings">Settings</string>  
</resources>
```

@string/app_name

@string/hello_world

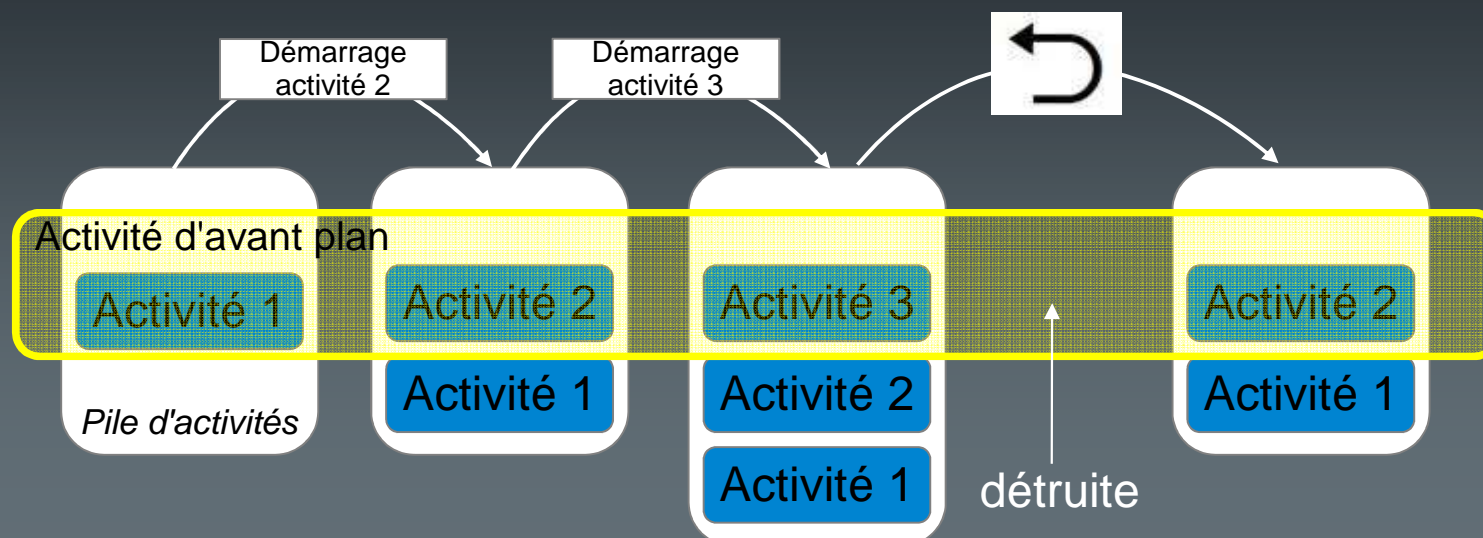
@string/action_settings

Plan du cours

- Introduction
- Architecture d'une application Android
- Les activités
- Définir une interface graphique
- Les intentions
- Les menus
- Les listes
- Les content providers

Les activités (1)

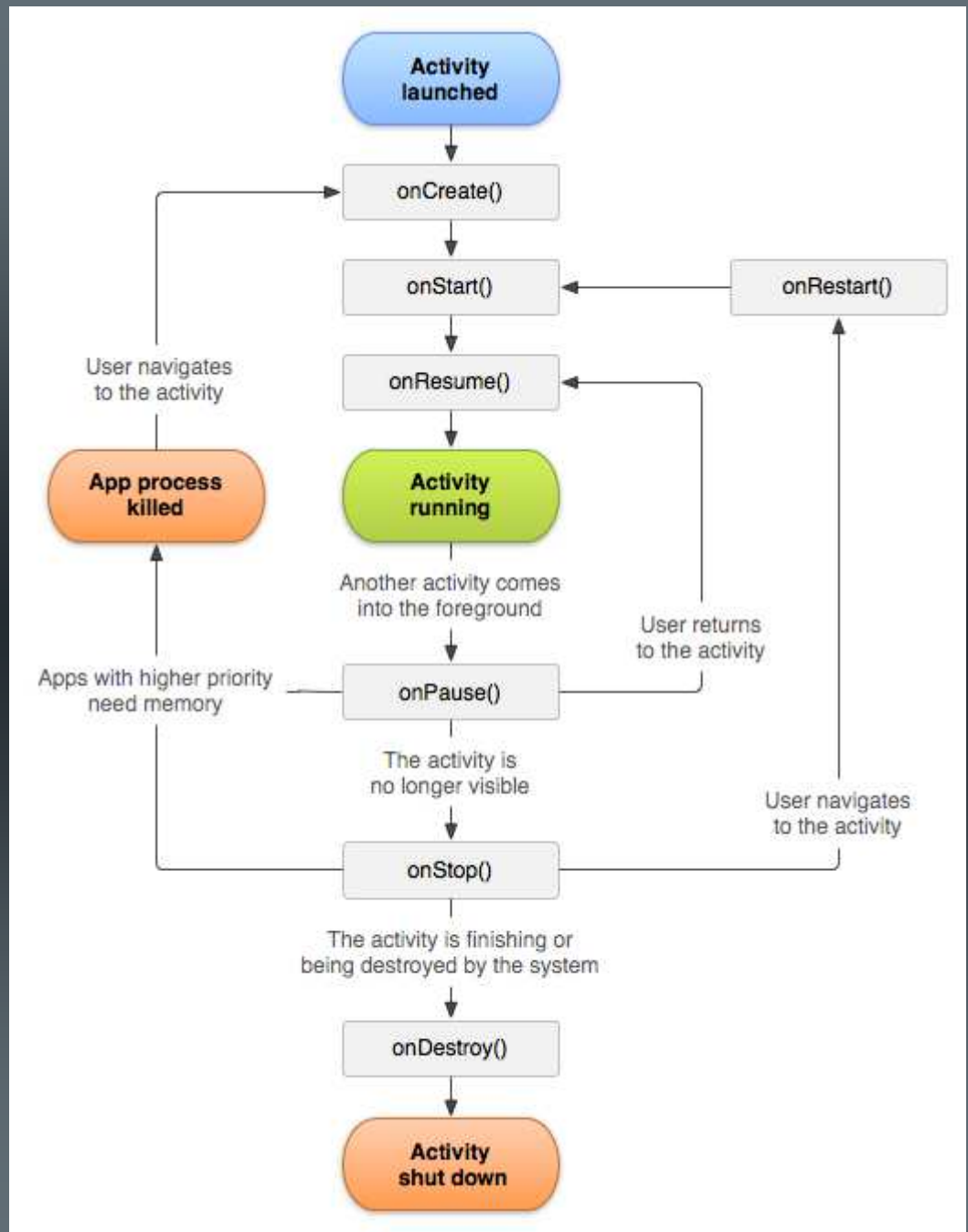
- Un composant d'une application, doté d'une interface graphique (IHM) et d'un contexte
- Une activité à la fois visible de l'utilisateur
 - Pour une même application
 - Pour des applications différentes
- Empilement des activités



Les activités (2)

- Cycle de vie

- Une activité peut se trouver dans différents états en fonction des actions du système et/ou de l'utilisateur :
 - Active : après un appel à `onResume()`
 - Suspendue : après un appel à `onPause()`
 - Arrêtée : après un appel à `onStop()`
 - Terminée ; après un appel à `onDestroy()`



Les activités (3)

■ Développement

- Une classe java par activité ;
- Les ressources associées (layout, menu, etc.) ;
- La classe hérite de la classe Activity ;
- Génération d'un code minimum par défaut sous Android Studio.

```
...  
  
public class Bidon extends Activity {  
  
    @Override  
    protected void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        setContentView(R.layout.Bidon);  
    }  
  
    ...  
}
```

Les activités (4)

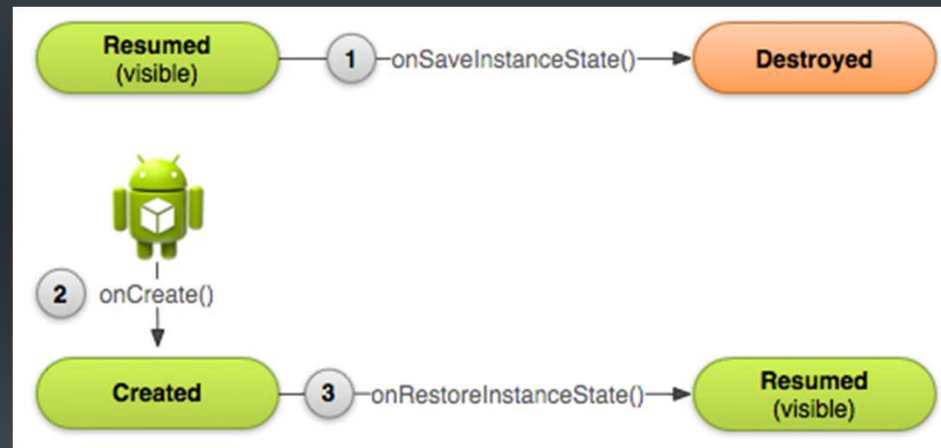
- D'autres méthodes peuvent être surchargées, en précisant ce qui doit être fait quand :
 - `protected void onDestroy()` : l'activité se termine
 - `protected void onStart()` : l'activité démarre (ou redémarre)
 - `protected void onPause()` : l'activité n'est plus au premier plan
 - `protected void onResume()` : l'activité revient au premier plan
 - `protected void onStop()` : l'activité n'est plus visible
 - `protected void onRestart()` : l'activité redevient visible

Les activités (5)

- Destruction de l'application par le système
 - Cas normal : l'activité est terminée. Le système récupère les ressources, en particulier la mémoire
 - Cas spéciaux : suppression d'une activité non active pour des raisons :
 - de limites des ressources ;
 - de changement d'orientation de l'écran.
 - Le système doit sauvegarder l'état de l'activité, pour pouvoir la redémarrer dans son état courant
 - Sauvegarde dans un objet **Bundle** : couples (nom_donnée, valeur)
 - Contient les données utilisées par l'interface par défaut
 - Systématique dès que l'activité n'est plus visible
 - Surcharge des méthodes de sauvegarde et restauration si d'autres données doivent être sauvées

Les activités (6)

`void onSaveInstanceState(Bundle outState)`



Source : develop.android.com

`void onCreate(Bundle savedInstanceState)`

`void onRestoreInstanceState(Bundle savedInstanceState)`

Plan du cours

- Introduction
- Architecture d'une application Android
- Les activités
- Définir une interface graphique
- Les intentions
- Les menus
- Les listes
- Les content providers

Quelques règles de base

- Interface = seul contact de l'utilisateur
 - Faire attirant
 - Faire simple
 - L'application doit être intuitive
 - Éviter les trop longs messages
- Faire ergonomique
 - L'enchaînement des activités doit être rapide
 - L'utilisateur doit toujours connaître l'état courant de l'activité
- Conseils et « matériels » :
 - <http://developer.android.com/design/index.html>

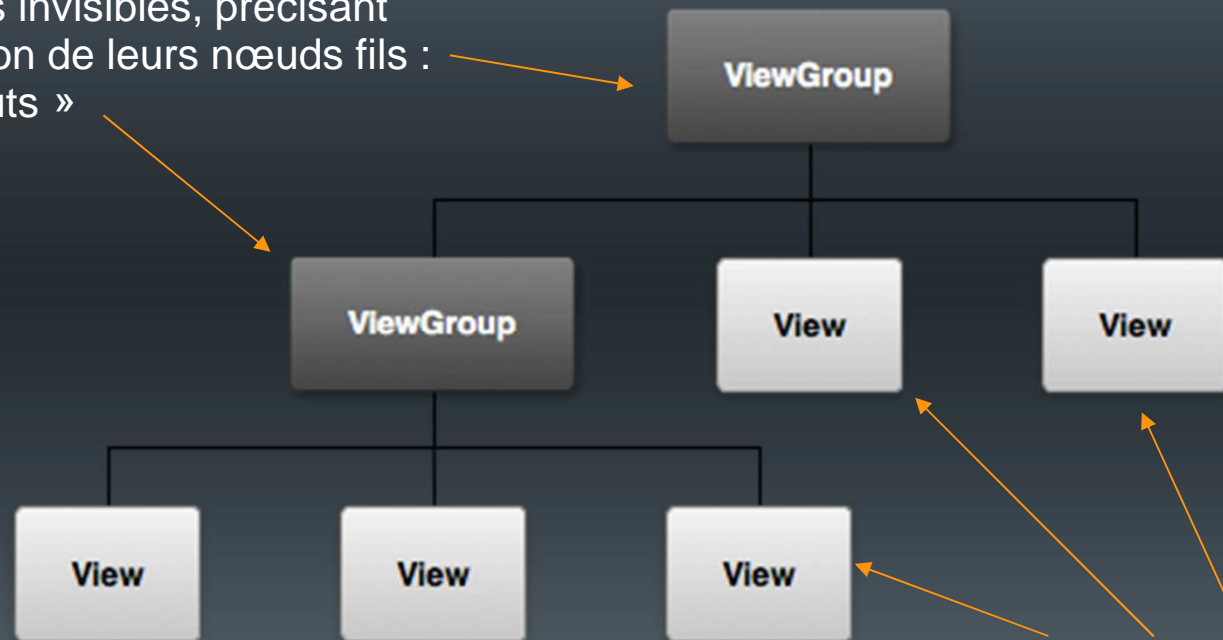
Définir une interface graphique

- Définir les « interacteurs »
 - Objets graphiques visibles par l'utilisateur pour :
 - L'affichage (texte, images, etc.)
 - L'interaction (boutons, cases, champs de saisie, etc.)
- Définir leur mise en page
 - Positions dans l'interface (fixes ou relatives)
- XML ou Java (sauf traitement de l'interaction : Java seul)
 - Privilégier XML
 - Souplesse de mise à jour
 - Permet la prise en compte simplifiée de différents types d'écran

Représentation d'une interface

- Représentation arborescente

Conteneurs invisibles, précisant l'organisation de leurs nœuds fils : les « Layouts »



Source : developer.android.com

Objets graphiques permettant l'interaction (boutons, zones de texte, etc.) : les Widgets

Les Layouts (1)

- Zone invisible assurant l'organisation automatique des composants graphiques
 - Peuvent être déclarées en XML ou Java
 - Privilégier XML
 - Séparation du code et de la mise en page
 - Souplesse d'adaptation à différents périphériques
- Possèdent des propriétés « intuitives » permettant l'organisation des composants
- Nombreux layouts différents
 - Peuvent être imbriqués (cf arborescence)
- Un layout doit être chargé dans onCreate()
 - setContentView(R.layout.nom_du_layout)

Les Layouts (2)

- Gestion multi-écrans

- Différentes tailles

- small, normal, large, xlarge

- Différentes densités de pixels

- low (ldpi), medium (mdpi), high (hdpi), extra high (xhdpi)

- Prévoir un layout par taille (et orientation) de l'écran si nécessaire

- effets de positionnements relatifs pouvant être gênants

- Prévoir des images en différentes résolutions

Les Layouts (3)

- Fonctionnement similaire à l'internationalisation
 - Un sous-dossier spécifique à chaque layout et/ou à chaque image

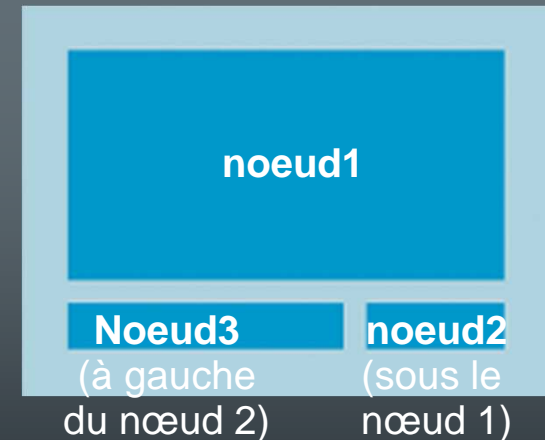
```
MyProject/  
  res/  
    layout/  
      main.xml           # default (portrait)  
    layout-land/  
      main.xml           # landscape  
    layout-large/  
      main.xml           # large (portrait)  
    layout-large-land/  
      main.xml           # large landscape
```

```
MyProject/  
  res/  
    drawable-xhdpi/  
      awesomeimage.png  
    drawable-hdpi/  
      awesomeimage.png  
    drawable-mdpi/  
      awesomeimage.png  
    drawable-ldpi/  
      awesomeimage.png
```

Source : developer.android.com

RelativeLayout (1)

- Layout par défaut pour un nouveau projet
 - Positionnement des noeuds par rapport au parent ou les uns par rapport aux autres



```
MainActivity.java x activity_main.xml x strings.xml x
<RelativeLayout
  xmlns:android="http://schemas.android.com/apk/res/android"
  xmlns:tools="http://schemas.android.com/tools"
  android:layout_width="match_parent"
  android:layout_height="match_parent"
  android:paddingLeft="@dimen/activity_horizontal_margin"
  android:paddingRight="@dimen/activity_horizontal_margin"
  android:paddingTop="@dimen/activity_vertical_margin"
  android:paddingBottom="@dimen/activity_vertical_margin" tools:context=".MainActivity">

  <TextView
    android:text="@string/hello_world_01"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content" />

  <TextView
    android:text="@string/hello_world_02"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    />

</RelativeLayout>
```

- **match_parent** : S'adapte à la taille du conteneur parent (ici l'écran)
- **wrap_content** : S'adapte à la taille de ce qu'il contient (ici deux zones de texte)
- *dimension fixe*

RelativeLayout (2)

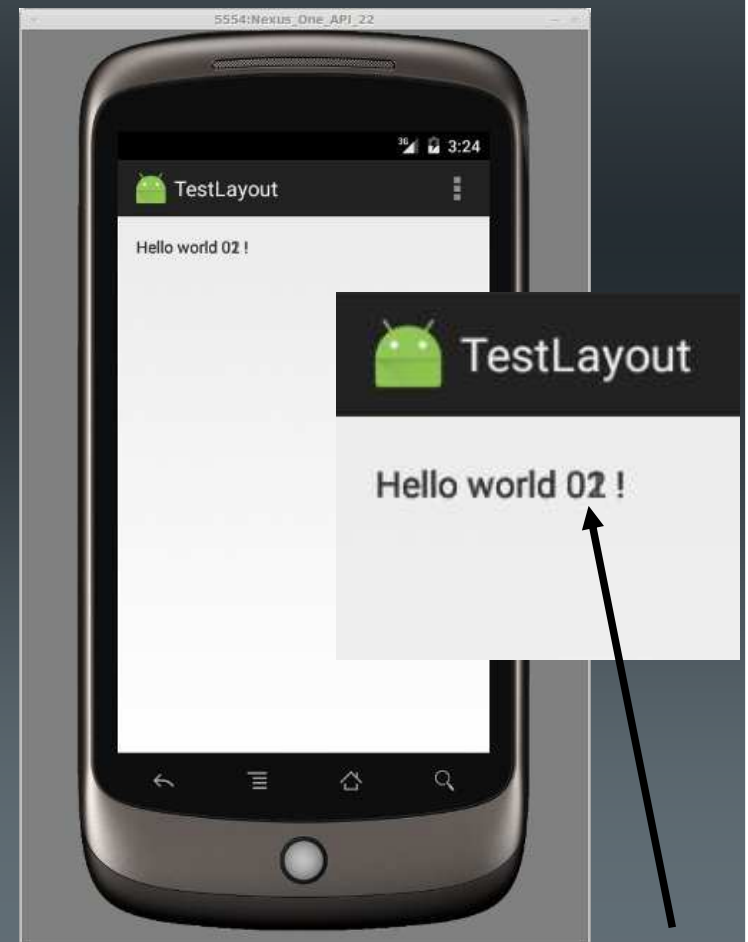
- Comportement par défaut :
 - Tous les noeuds sont positionnés à partir du coin supérieur gauche
 - Superposition !!

```
MainActivity.java x activity_main.xml x strings.xml x
<RelativeLayout
  xmlns:android="http://schemas.android.com/apk/res/android"
  xmlns:tools="http://schemas.android.com/tools"
  android:layout_width="match_parent"
  android:layout_height="match_parent"
  android:paddingLeft="@dimen/activity_horizontal_margin"
  android:paddingRight="@dimen/activity_horizontal_margin"
  android:paddingTop="@dimen/activity_vertical_margin"
  android:paddingBottom="@dimen/activity_vertical_margin" tools:context=".MainActivity">

  <TextView
    android:text="@string/hello_world_01"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content" />

  <TextView
    android:text="@string/hello_world_02"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    />

</RelativeLayout>
```



RelativeLayout (3)

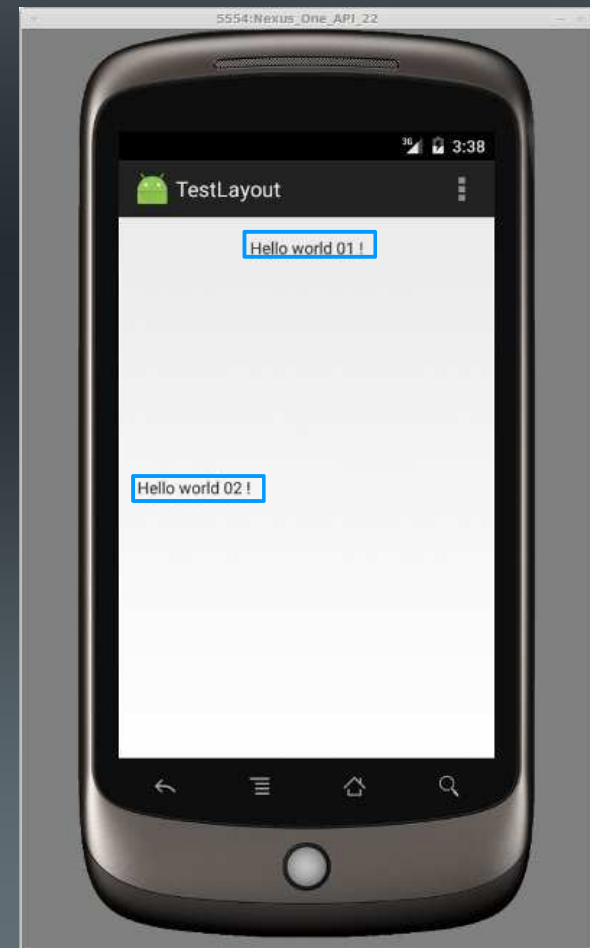
- Attributs de positionnement par rapport au parent :
 - android:layout_centerHorizontal
 - android:layout_centerVertical
 - android:centerInParent
 - ... (cf RelativeLayout.LayoutParams)

```
MainActivity.java x activity_main.xml x strings.xml x
<RelativeLayout
  xmlns:android="http://schemas.android.com/apk/res/android"
  xmlns:tools="http://schemas.android.com/tools"
  android:layout_width="match_parent"
  android:layout_height="match_parent"
  android:paddingLeft="@dimen/activity_horizontal_margin"
  android:paddingRight="@dimen/activity_horizontal_margin"
  android:paddingTop="@dimen/activity_vertical_margin"
  android:paddingBottom="@dimen/activity_vertical_margin" tools:context=".MainActivity">

  <TextView
    android:text="@string/hello_world_01"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_centerHorizontal="true"
  />

  <TextView
    android:text="@string/hello_world_02"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_centerVertical="true"
  />

</RelativeLayout>
```



RelativeLayout (4)

- Attributs de positionnement par rapport aux autres nœuds :
 - android:layout_below
 - android:layout_above
 - android:layout_toLeftOf
 - android:layout_toRightOf
 - ... (cf RelativeLayout.LayoutParams)
- Nécessité de nommer les nœuds
 - Permet de préciser le nœud à partir duquel on se positionne

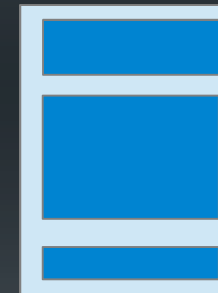
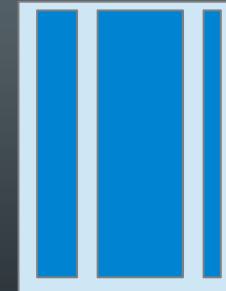
```
<TextView  
  android:id="@+id/hw01"  
  android:text="@string/hello_world_01"  
  android:layout_width="wrap_content"  
  android:layout_height="wrap_content"  
  android:layout_centerHorizontal="true"  
>
```

```
<TextView  
  android:text="@string/hello_world_02"  
  android:layout_width="wrap_content"  
  android:layout_height="wrap_content"  
  android:layout_toLeftOf="@id/hw01"  
>
```



LinearLayout (1)

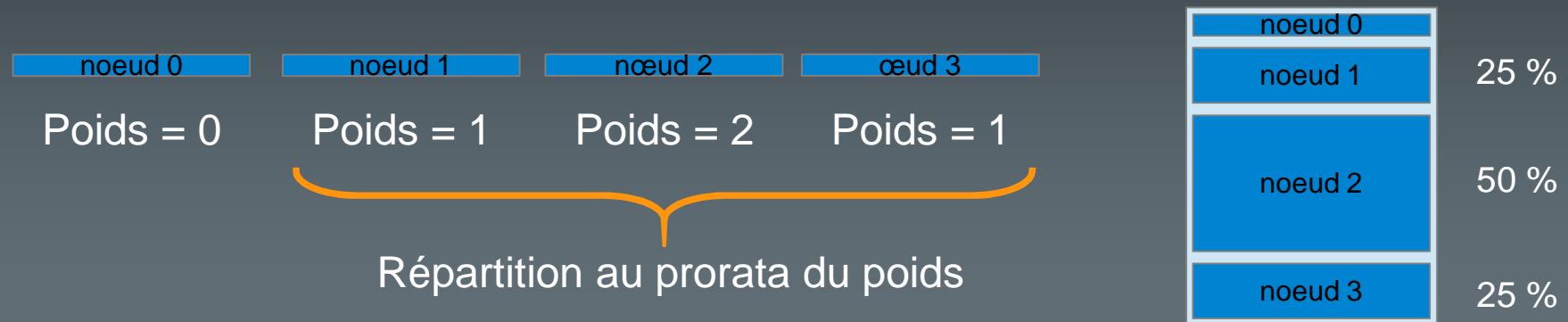
- Aligne les nœuds dans une seule direction
 - horizontale (par défaut)
 - verticale



```
MainActivity.java x activity_main.xml x strings.xml x
<LinearLayout
  xmlns:android="http://schemas.android.com/apk/res/android"
  xmlns:tools="http://schemas.android.com/tools"
  android:layout_width="match_parent"
  android:layout_height="match_parent"
  android:paddingLeft="@dimen/activity_horizontal_margin"
  android:paddingRight="@dimen/activity_horizontal_margin"
  android:paddingTop="@dimen/activity_vertical_margin"
  android:paddingBottom="@dimen/activity_vertical_margin"
  android:orientation="vertical"
  tools:context=".MainActivity"
>
```

LinearLayout (2)

- Modification du « poids » de chaque nœud
 - Permet de changer la taille de la zone occupée par chaque nœud dans l'écran
 - Ajout d'un attribut `android:layout_weight` à chaque nœud
 - 0 (par défaut) : n'utilise que la zone nécessaire au nœud
 - $n > 0$: poids du nœud par rapport aux autres nœuds



LinearLayout (3)

■ Example

```
<TextView
    android:text="@string/hello_world_01"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
/>

<TextView
    android:text="@string/hello_world_02"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_weight="2"
/>

<TextView
    android:text="@string/hello_world_03"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_weight="1"
/>

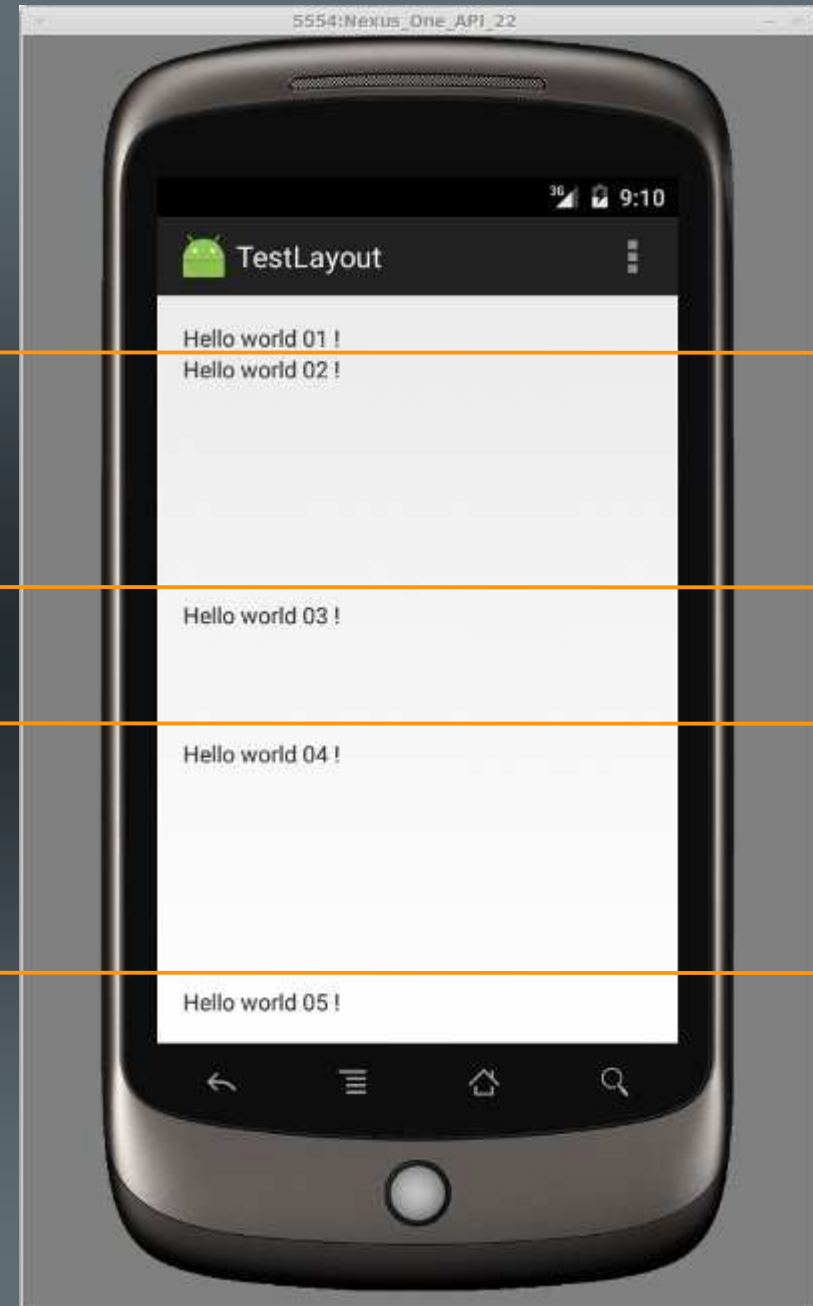
<TextView
    android:text="@string/hello_world_04"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_weight="2"
/>

<TextView
    android:text="@string/hello_world_05"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
/>
```

40 %

20 %

40 %



LinearLayout (4)

- Alignement de chaque noeud dans sa zone
 - Ajout d'un attribut `android:layout_gravity`
 - Nombreuses valeurs possibles :
 - `center`, `center_vertical`, `center_horizontal`
 - `left`, `right`, `top`, `bottom`
 - Etc. (cf `LinearLayout.LayoutParams`)

LinearLayout (5)

■ Example

```
<TextView
    android:text="@string/hello_world_01"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
/>

<TextView
    android:text="@string/hello_world_02"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_gravity="center_horizontal"
    android:layout_weight="2"
/>

<TextView
    android:text="@string/hello_world_03"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_gravity="right"
    android:layout_weight="1"
/>

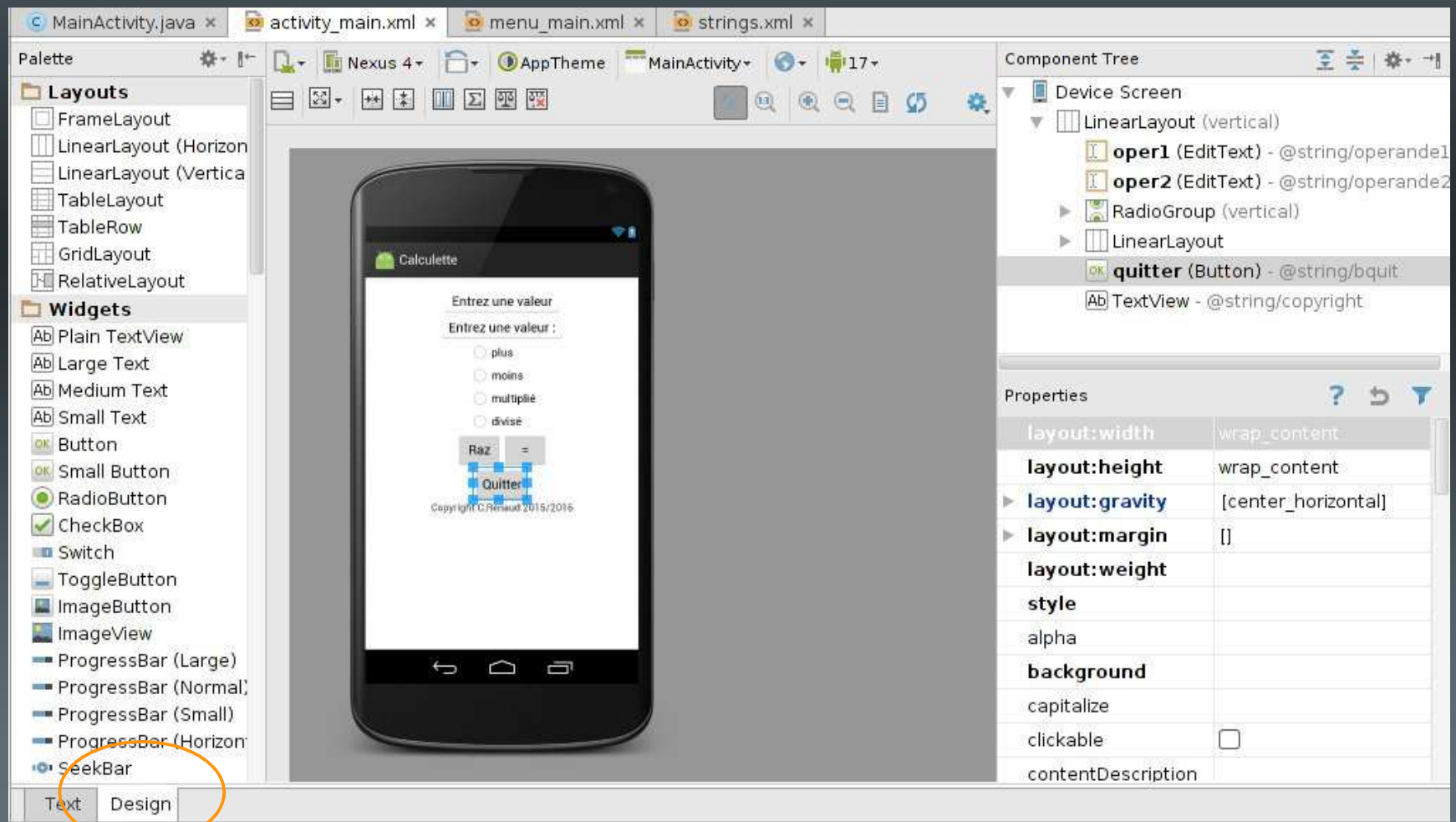
<TextView
    android:text="@string/hello_world_04"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_gravity="center"
    android:gravity="center"
    android:layout_weight="2"
/>

<TextView
    android:text="@string/hello_world_05"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_gravity="right"
/>
```



Remarque

- Possibilité d'organiser visuellement les layouts sous Android Studio



Les Widgets

- Composants graphiques visibles par l'utilisateur
 - Widgets simples : zones de texte, boutons, listes, etc.
 - Widgets plus complexes : horloges, barres de progression, etc.
- Héritent de la classe View
- Utilisation :
 - Définition en XML (type, taille, centrage, position, etc.)
 - Comportement en Java
 - Peuvent également être créés dynamiquement en Java

Les TextView

- Widget permettant l'affichage d'un texte
 - Normalement non éditable

- Exemple :

```
<TextView
  android:id="@+id/text"
  android:layout_width="wrap_content"
  android:layout_height="wrap_content"
  android:text="@string/letexte"
  android:hint="texte initial"
  android:layout_gravity="center"
  android:gravity="center"
/>
```

- Nombreux autres attributs
 - Cf classe TextView

Les EditText

- Widget permettant la saisie d'un texte (TextFields)
 - Accès : ouverture d'un clavier pour la saisie
 - nombreux attributs permettant l'aide à la saisie

```
<EditText  
    android:id="@+id/email_address"  
    android:layout_width="fill_parent"  
    android:layout_height="wrap_content"  
    android:hint="@string/email_hint"  
    android:inputType="textEmailAddress" />
```

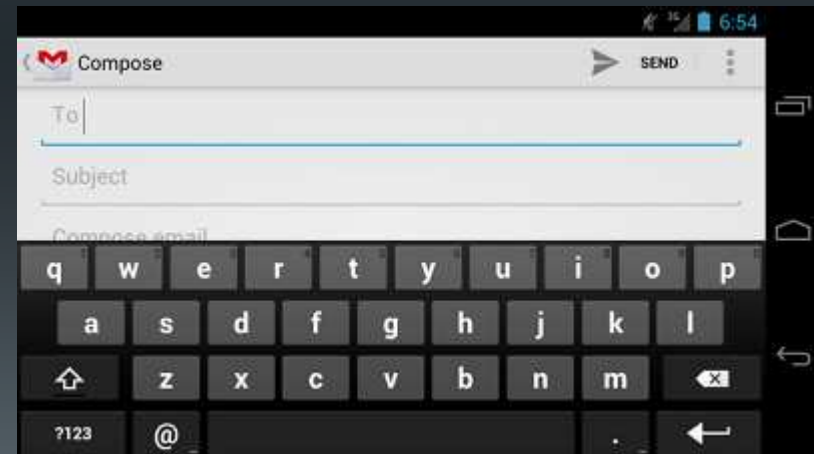
"text" : Normal text keyboard.

"textEmailAddress" : Normal text keyboard with the @ character.

"textUri" : Normal text keyboard with the / character.

"number" : Basic number keypad.

"phone" : Phone-style keypad.



Source : developer.android.com

Les Button

- Widget représentant un bouton d'action
 - Renvoie un événement lors de l'appui
 - Peut contenir un texte, une image ou les deux
- Exemples :

```
<Button  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:text="@string/button_text"  
    ... />
```

```
<Button  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:text="@string/button_text"  
    android:drawableLeft="@drawable/button_icon"  
    ... />
```



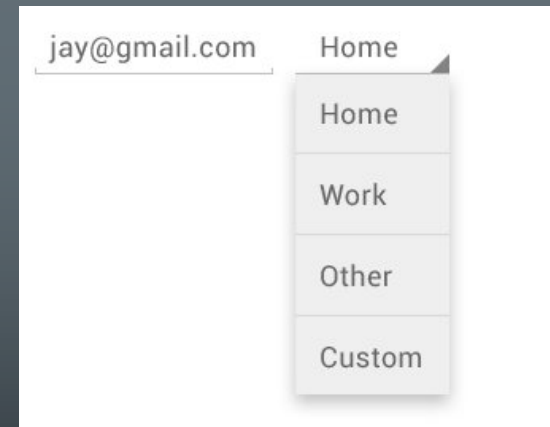
```
<ImageButton  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:src="@drawable/button_icon"  
    ... />
```

Source : developer.android.com

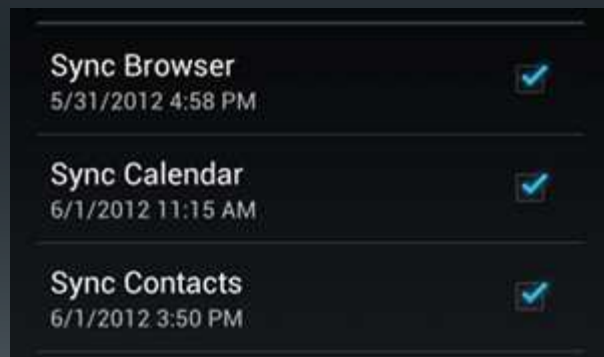
En vrac ...

- Quelques autres widgets

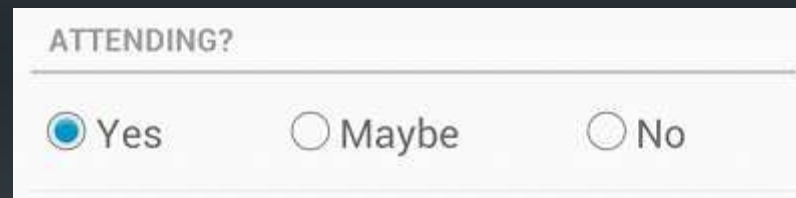
- Source developer.android.org



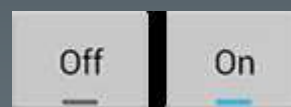
Spinner



CheckBox



RadioButton



ToggleButton



Switch
(android 4.0+)

Implantation du comportement (1)

- Les fichiers XML ne permettent que de :
 - positionner les composants ;
 - définir leurs caractéristiques.
- Nécessité de :
 - définir leur comportement
 - type d'interaction (clic court, clic long, etc.)
 - code de prise en compte (Java)
 - lier composant et code
 - XML : attribut `android:onClick`
 - Java : instancier un *event listener*

Implantation du comportement (2)

- Attribut android:onClick
 - Doit être suivi du nom de la méthode à appeler en cas de déclenchement
 - Prototype :
 - `public void nomDeLaMethode(View maVue)`

```
<Button  
  android:id="@+id/monBouton"  
  android:layout_width="wrap_content"  
  android:layout_height="wrap_content"  
  android:text="@string/monTexte"  
  android:onClick="onBoutonClique"  
>
```

```
public void onBoutonClique(View maVue) {  
    System.out.println("le bouton a été cliqué");  
}
```

Permet de récupérer des informations sur le composant graphique qui a généré l'événement

`maVue.getId()` ➡ `R.id.monBouton`

Implantation du comportement (3)

- Les *event listener*

- interfaces de la classe View
- ne disposent que d'une seule méthode à implanter
- méthode appelée quand le composant associé est déclenché par l'utilisateur

- Exemples :

Interface	Méthode
View.OnClickListener	abstract void onClick(View v)
View.OnLongClickListener	abstract boolean onLongClick(View v)
View.OnFocusChangeListener	abstract void onFocusChange(View v, boolean hasFocus)

Implantation du comportement (3)

- Exemple : l'interface View.OnClickListener
 - public void onClick(View v)

```
...  
Button button = (Button) findViewById(R.id.button_name);  
  
button.setOnClickListener(new View.OnClickListener() {  
  
    public void onClick(View v) {  
        // Do something in response to button click  
    }  
  
});  
...
```

Source : developer.android.com

Plan du cours

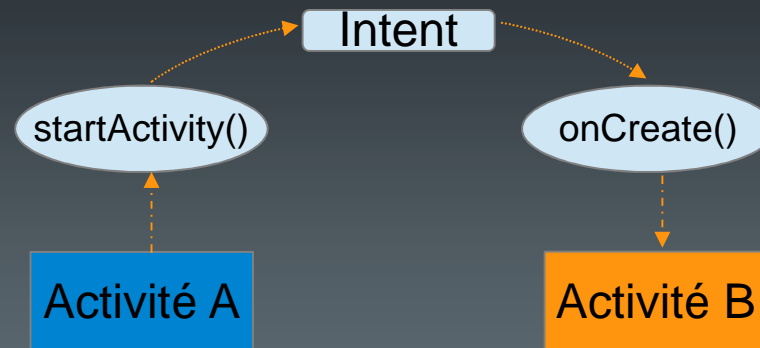
- Introduction
- Architecture d'une application Android
- Les activités
- Définir une interface graphique
- Les intentions
- Les menus
- Les listes
- Les content providers

Les Intentions

- Classe représentant un message échangé entre une activité et un composant présent sur le système
 - Une autre activité
 - Un service
 - Un diffuseur d'événements
- Deux types de messages
 - Explicite : on nomme le composant à démarrer
 - Implicite : on demande au système de trouver un composant adéquat, en fonction d'une action à effectuer

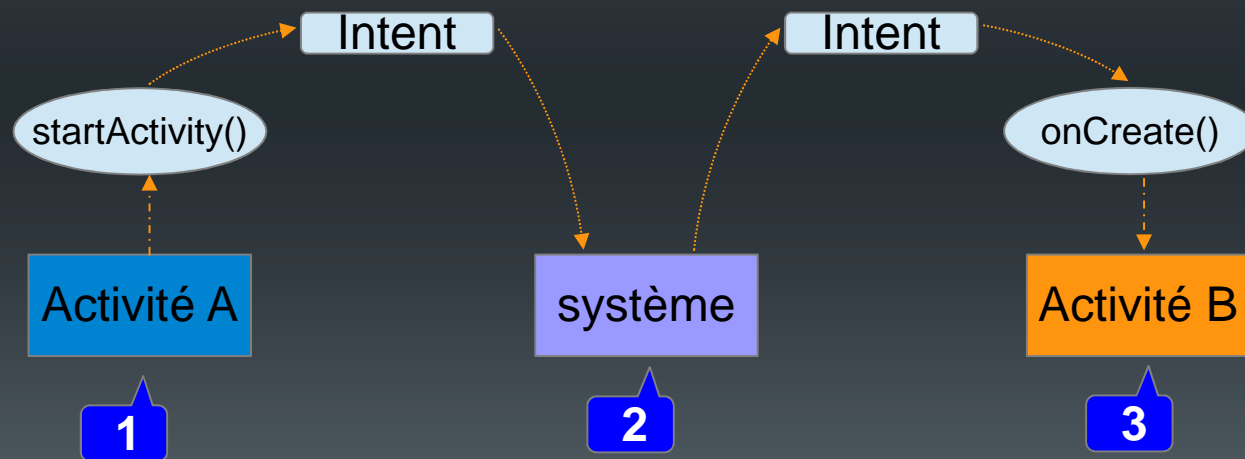
Les Intentions explicites

- Message adressé à un composant connu
 - On donne le nom de la classe correspondante
 - Généralement réservé aux composants appartenant à la même application ...
- Le composant est démarré immédiatement



Les Intentions implicites

- Message à destination d'un composant inconnu
 - Le système se charge de trouver le composant adéquat et le démarre



- En cas de possibilités multiples, affichage des choix à l'utilisateur

Création et lancement

Création d'une intention

```
Intent intention = new Intent (...);  
...  
startActivity(intention);
```

Démarrage d'une activité

```
Intent intention = new Intent (...);  
...  
startService(intention);
```

Démarrage d'un service

```
void maFonction(...) {  
    ...  
    Intent intention = new Intent (...);  
    ...  
    startActivityForResult(intention, requestCode);  
    ...  
}
```

*Démarrage d'une activité
avec attente d'un résultat,
récupéré dans la surcharge
de cette méthode*

```
protected void onActivityResult (int requestCode, int resultCode, Intent data){  
    ...  
}
```

Récupération

Code côté récepteur

```
public void onCreate (Bundle savedInstanceState) {  
    ...  
    Intent intention = getIntent () ;  
    ...  
    // extraction des informations reçues  
    ...  
    // traitement des informations reçues  
    ...  
}
```


Intentions avec résultats (émetteur)

```
public static final int CODE = 4 ;
```

Code créé par le développeur pour identifier de manière unique son intention (>0)

```
void maFonction(...) {  
    ...  
    Intent intention = new Intent (...) ;  
    ...  
    startActivityForResult(Intent intention, int requestCode) ;  
    ...  
}
```

```
protected void onActivityResult (int requestCode, int resultCode, Intent data){  
    ...  
}
```

Code de retour d'exécution de l'intent
Activity.RESULT_OK
Activity.RESULT_CANCELED

Données transmises en retour
à l'activité appelante

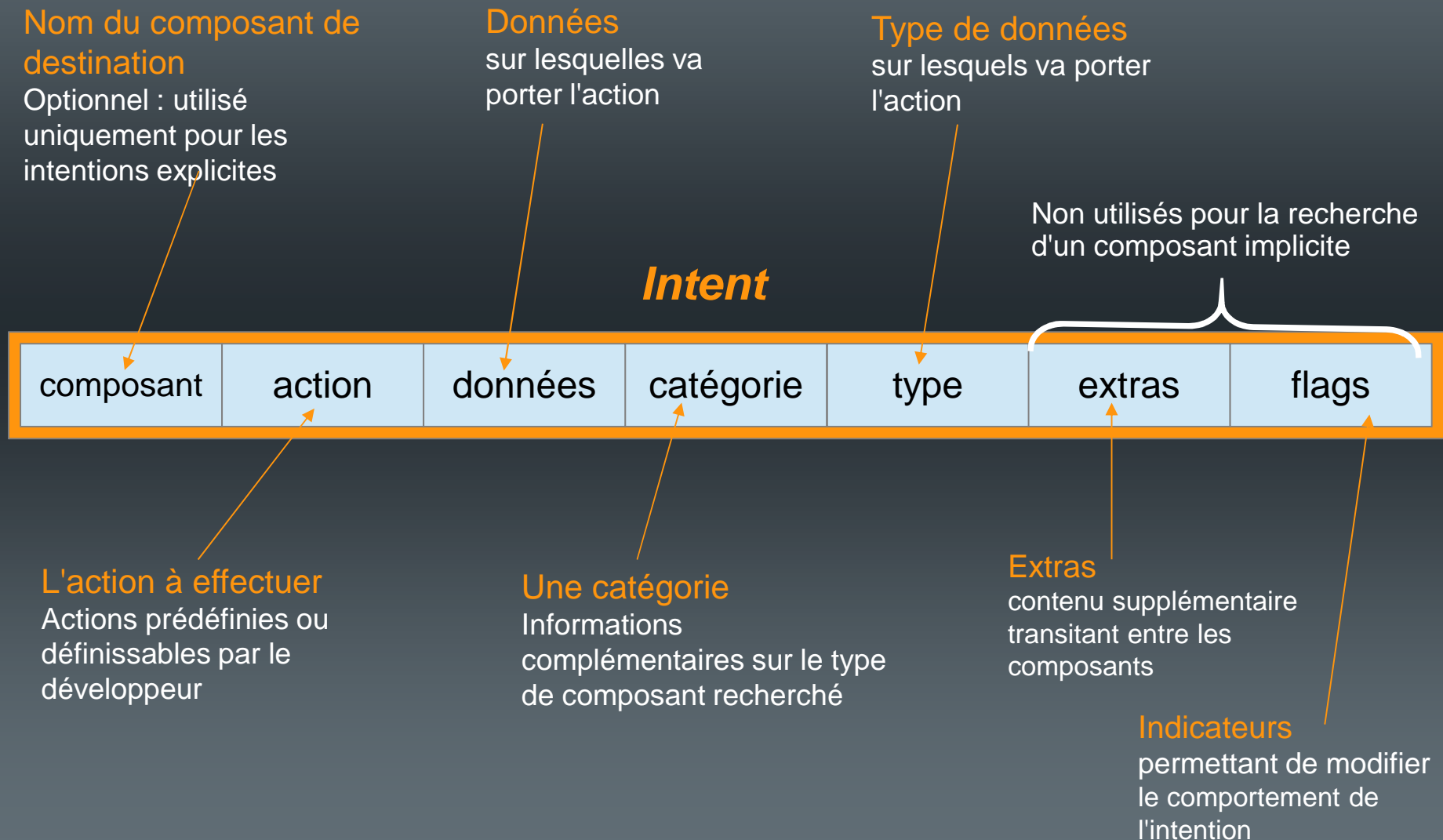
Intentions avec résultats (récepteur)

```
void maFonction(...) {  
    ...  
    Intent intention = new Intent (...);  
    ...  
    startActivityForResult(Intent intention,  
                           int requestCode);  
    ...  
}  
  
protected void onActivityResult (int requestCode,  
                                  int resultCode,  
                                  Intent data){  
    ...  
}
```

```
public void onCreate (Bundle savedInstanceState)  
{  
    // récupérer l'intent  
    Intent intention = getIntent ();  
  
    // extraire les données et les traiter  
    ...  
    // créer l'intent résultat  
    Intent resultat = new Intent();  
    // ajout des résultats  
    ...  
    // prépare le retour des résultats  
    setResult(RESULT_OK, resultat);  
    finish();  
}
```

callback

Structure générale des Intentions



Nom du composant

■ Différentes méthodes utilisables :

- Intent(...) : le constructeur
- SetComponent(...), setClassName(...), setClass(...), etc.

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

Le contexte

```
...
Intent = new Intent() ;
Intent.setClassName(this, "fr.univ-littoral.renaud.bidon.SecondActivity") ;
...
startActivity(intent) ;
...
```

Le package

```
...
Intent = new Intent() ;
Intent.setComponent(new ComponentName("fr.univ-littoral.renaud.bidon",
"fr.univ-littoral.renaud.bidon.SecondActivity") ;
...
startActivity(intent) ;
...
```

La classe

Action à effectuer

- Nombreuses constantes prédéfinies
 - Dans la classe Intent :
 - ACTION_MAIN : démarre l'activité comme point d'entrée principal
 - ACTION_VIEW : affiche les données fournies à l'utilisateur
 - ACTION_EDIT : permet l'édition des données fournies par l'utilisateur
 - ACTION_SEND : permet l'envoi des données (mail, réseau social, ...)
 - etc
- Possibilité de définir ses propres actions
 - Comme constante de classe

```
static final String ACTION_BIDON = "fr.univ-littoral.renaud.bidon.BIDON";
```



type Nom Valeur, incluant le nom du package

Données

- Formatées à l'aide des URI (*Uniform Resource Identifier*)
 - Chaîne de caractères représentant un endroit ou une ressource
 - Syntaxe normalisée :

<schéma> : <information> { ? <requêtes> } { # <fragment> }

Précision quant à l'information

Nature de l'information :

http, https
content
geo
file
tel , voicemail
sms,smsto, mms, mmsto

L'information :

- Dépend du schéma
- Exemples :

tel:060000007
geo:123.456,12.3456
http://www.google.fr

Accès à une sous-partie
de l'information

```
Intent intention = new Intent() ;  
Uri sms = Uri.parse("sms:060000007") ;  
Intention = intention.setData(sms) ;
```

Le type

- Normalement implicite dans les données
- Absence de données/recherche plus précise :
 - peut être précisé en utilisant les codes MIME
 - text, audio, video
 - ...
- Et des sous-types
 - text/plain, text/xml, text/html, ...
 - audio/mp3, audio/ogg, audio/wav, ...
 - video/mpeg, video/mp4, ...

```
Intent intention = new Intent() ;  
...  
Intention = intention.setType("audio/mp3") ;  
...  
String mime = intention.getType() ;
```

Élimine les données ...
(setData élimine le type)

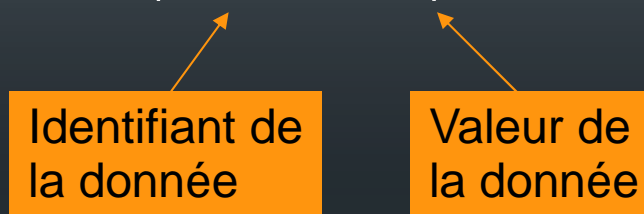


Catégorie

- Précise le type de composant qui doit prendre en charge l'intention
- Exemples :
 - CATEGORY_LAUNCHER : le composant doit être visible dans le lanceur d'applications
 - CATEGORY_CAR_MODE : le composant doit être utilisable en mode conduite de véhicule
- Remarques
 - Nombre quelconque de catégories dans une intention
 - La plupart des intentions ne nécessitent pas de catégories

Les extras (1)

- Permet d'insérer des données dans l'intention
 - Communications inter-composants
 - Structure : (clé, valeur)



insertion

*type = n'importe quel type de base
(boolean, int, String, float[], ...)*

récupération

```
Intent.putExtra(String cle, type valeur) ;
```

```
type get{type}Extra(String cle, type vdefault) ;
```

- Clé définie par l'utilisateur

```
public final static String MACLE = "fr.ulco.renaud.appli.MACLE" ;
```

- Quelques clés prédéfinies

- vdefault : valeur par défaut renvoyée si la clé n'est pas trouvée dans l'intention
- pas de valeur par défaut pour les tableaux, ni les String

```
type[] get{Type}ArrayExtra(String cle) ;  
String getStringExtra(String cle) ;
```

Les extras (2)

```
public class MainActivity extends Activity {  
  
    public final static String MARQUE = "fr.ulco.renaud.appli.MARQUE" ;  
    public final static String PUISS = "fr.ulco.renaud.appli.PUISS" ;  
  
    ...  
    Intent intention = new Intent(this, otherActivity.class) ;  
    String marque = "Citroen" ;  
    int puissance = 6 ;  
    intention.putExtra(MainActivity.MARQUE, marque) ;  
    intention.putExtra(MainActivity.PUISS, puissance) ;  
    startActivity(intention) ;  
    ...  
}
```

```
public class otherActivity extends Activity{  
  
    ...  
    Intent intention = getIntent() ;  
    int p = intention.getIntExtra(MainActivity.PUISSANCE, 0) ;  
    String m = intention.getStringExtra(MainActivity.MARQUE) ;  
    ...  
}
```

Les extras (3)

- Possibilité d'ajouter un Bundle :
 - Intent putExtras(String cle, Bundle valeur) ;
 - Bundle getBundleExtras(String cle) ;
- Possibilité d'ajouter une intention :
 - Intent putExtras(Intent valeur) ;
 - Recopie tous les extras de valeur dans l'intent appelant
- Possibilité d'ajouter des objets complexes :
 - Doivent être sérialisables
 - La classe d'origine doit implémenter l'interface Parcelable
 - Intent putExtra(String cle, Classe valeur) ;
 - Classe getParcelableExtra(String cle) ;

Recherche d'un composant

- Utilisation d'intents implicites :
 - Le système recherche le(s) composant(s) possible(s)
 - Le système doit connaître les types d'intents que peut recevoir chaque composant
 - Nécessité de préciser pour chaque composant les intents réceptionnables

 *Intent filters*

Les intent filters

- Présents dans le fichier *manifest.xml* pour chaque composant d'une application
- Syntaxe :

activity,
service,
...

Filtre d'intentions

```
<composant ...>  
  <intent-filter>  
    <action android:name="..." />  
    <category android:name="..." />  
    <data android:mimeType="..." />  
  </intent-filter>  
</composant>
```

Peuvent être en nombre quelconque

```
<intent-filter>  
  <action android:name="android.intent.action.MAIN" />  
  <category android:name="android.intent.category.LAUNCHER" />  
</intent-filter>
```

Composant principal
d'une application

Doit apparaître dans la liste
des applications pouvant être lancées

Remarques

- Intents et services :
 - Démarrage d'un service :
 - N'utiliser que des intents explicites
 - Intents implicites : l'utilisateur ne sait pas forcément quel service est lancé et il n'a pas de retour visuel
 - Pas d'intent filter pour un service

Intentions implicites

- Possibilité de ne pas trouver de destinataire
 - Vérifier qu'une application externe existe avant de lancer l'intention (sinon crash de l'application)

```
Intent intent = new Intent() ;  
...  
if (intent.resolveActivity(getPackageManager()) != null) {  
    startActivity(intent);  
}else{  
    ...  
}
```

- Récupération des données retournées
 - On récupère un « lien » vers le données :

Uri uri = data.getData()

Les permissions (1)

- Restreindre ce qu'une application a le droit de faire
- Protéger les données présentes sur le périphérique
- Par défaut, impossibilité d'impacter :
 - Une autre application
 - Le système
 - Les données utilisateur
- Chaque application est lancée dans son propre espace (*sandbox*)
 - Nécessité de préciser les permissions
 - Validation par l'utilisateur

Les permissions (2)

- Deux groupes de permissions
 - Normales :
 - Sans risque pour l'utilisateur
 - ex. allumer le flash
 - Sont attribuées automatiquement sur demande de l'application
 - Dangereuses
 - Possibilité de compromettre les données utilisateur ou d'autres applications
 - ex. lire les contacts
 - Nécessitent une validation de l'utilisateur

Les permissions (3)

- Définition

- Où ?

- Dans le fichier manifest.xml

- Comment ?

- Balise `<use-permission>`

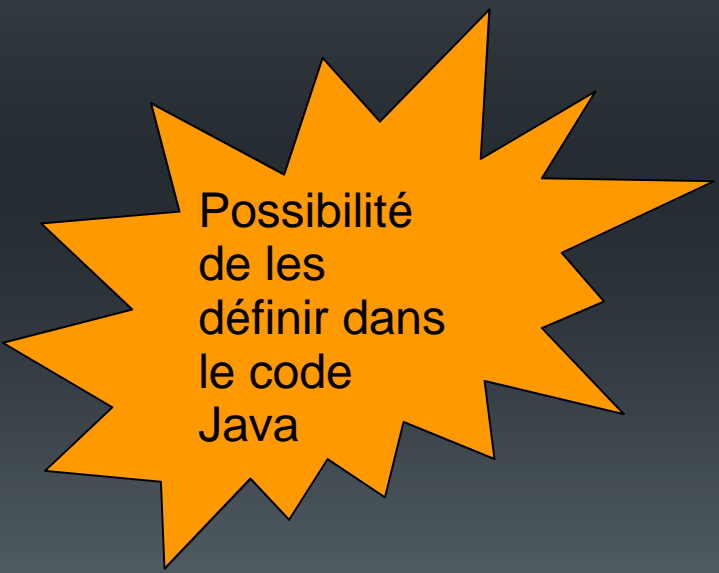
```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="fr.ulco.renaud.myapp" >
    ...
    <uses-permission android:name="android.permission.RECEIVE_SMS" />
    ...
</manifest>
```

Plan du cours

- Introduction
- Architecture d'une application Android
- Les activités
- Définir une interface graphique
- Les intentions
- Les menus
- Les listes
- Les content providers

Les menus

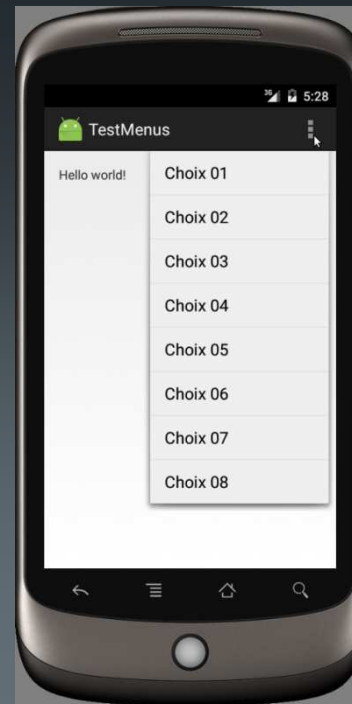
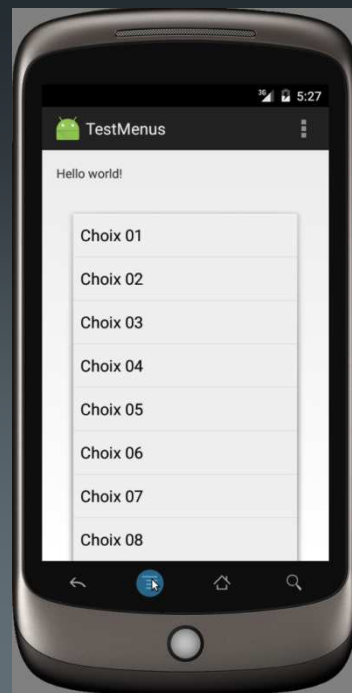
- Disponibles pour chaque activité
- Trois types de menus :
 - Le menu d'options
 - Les menus contextuels
 - Les menus pop-ups
- Considérés comme des ressources
 - À définir en XML (dossier res/menus)
 - Facilite la maintenance des menus
 - séparation de la définition du code de gestion
 - Possibilité de créer des menus multi-plateforme
 - Facilite la visualisation de la structure des menus



Possibilité
de les
définir dans
le code
Java

Le menu d'options (1)

- Accessibles via :
 - le bouton de « menu »
 - N'est plus forcément disponible sur les périphériques récents
 - l'icône « menu » de la barre d'application



Le menu d'options (2)

■ Définition XML

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

```
<item android:id="@+id/choix01"
      android:title="@string/texte_choix01" />
```

...

```
<item android:id="@+id/choixNN"
      android:title="@string/texte_choixNN" />
```

```
</menu>
```

Identifiant de
chaque entrée

Libellé du choix

Le menu d'options (3)

■ Attributs des items

```
<item android:id="@+id/choix01"  
      android:title="@string/texte_choix01"  
      android:icon="..."  
      android:onClick="..."  
      android:showAsAction=" "  
/>
```

← Icône associée au choix

← Callback en cas de choix

Précise quand et comment l'item peut apparaître dans la barre d'application :


- "ifRoom" : s'il y a de la place
- "never" : jamais
- "always" : toujours (à éviter sauf item important pour des raisons de place)
- ...

Le menu d'options (4)

- Chargement du menu
 - Appel de la méthode *onCreateOptionsMenu()* au lancement de l'application

```
public boolean onCreateOptionsMenu(Menu menu) {  
    // Inflate the menu; this adds items to the action bar if it is present.  
    MenuInflater inflater = getMenuInflater();  
    inflater.inflate(R.menu.monMenu, menu);  
  
    return true;  
}
```

Menu de l'application
devant être initialisé



Objet permettant de gérer
l'association XML/menu



Association XML/menu



Le menu d'options (5)

- Gestion des choix

- Appel de la méthode *onOptionsItemSelected()* à chaque sélection (par défaut)

```
public boolean onOptionsItemSelected(MenuItem item) {
```

```
    int id = item.getItemId();
```

```
    switch(id){  
        case R.id.choix01 : ...
```

```
        ...  
        case R.id.choixNN : ..  
    }
```

```
}
```

L'item choisi



Récupération de l'identifiant
de l'item dans la classe R



- Remplaçable par une méthode *nom(MenuItem item)*

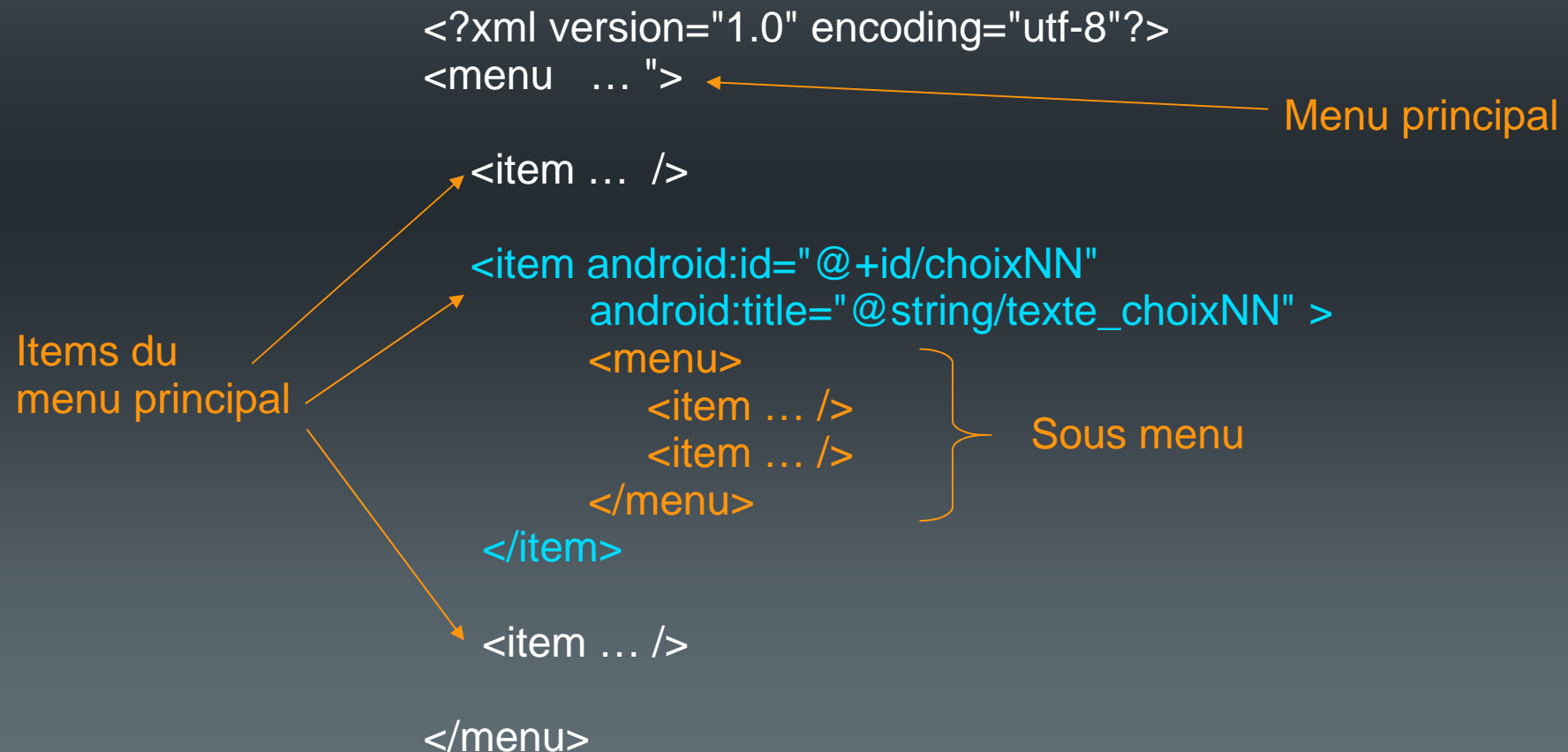
```
public void nom(MenuItem item){ ... }
```

```
<item ...
```

```
    android:onClick="nom" />
```

Le menu d'options (6)

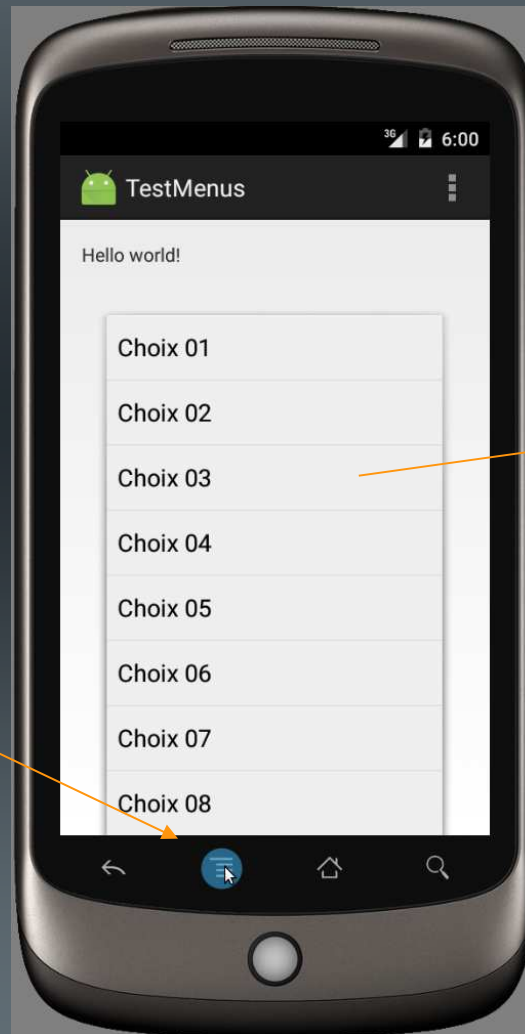
- Extensions aux sous-menus
 - tout item peut contenir un sous-menu



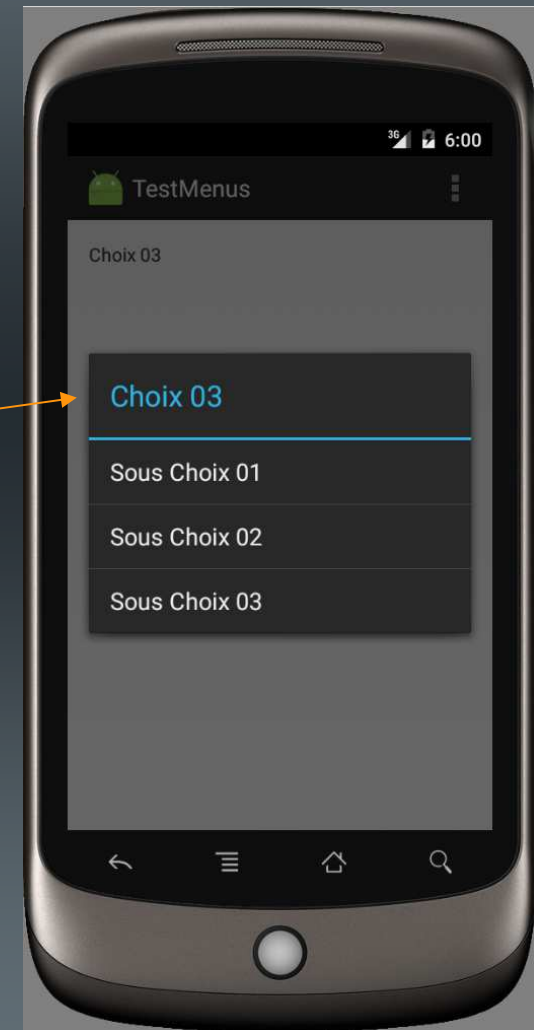
Le menu d'options (7)

■ Exemples

Sélection du menu principal

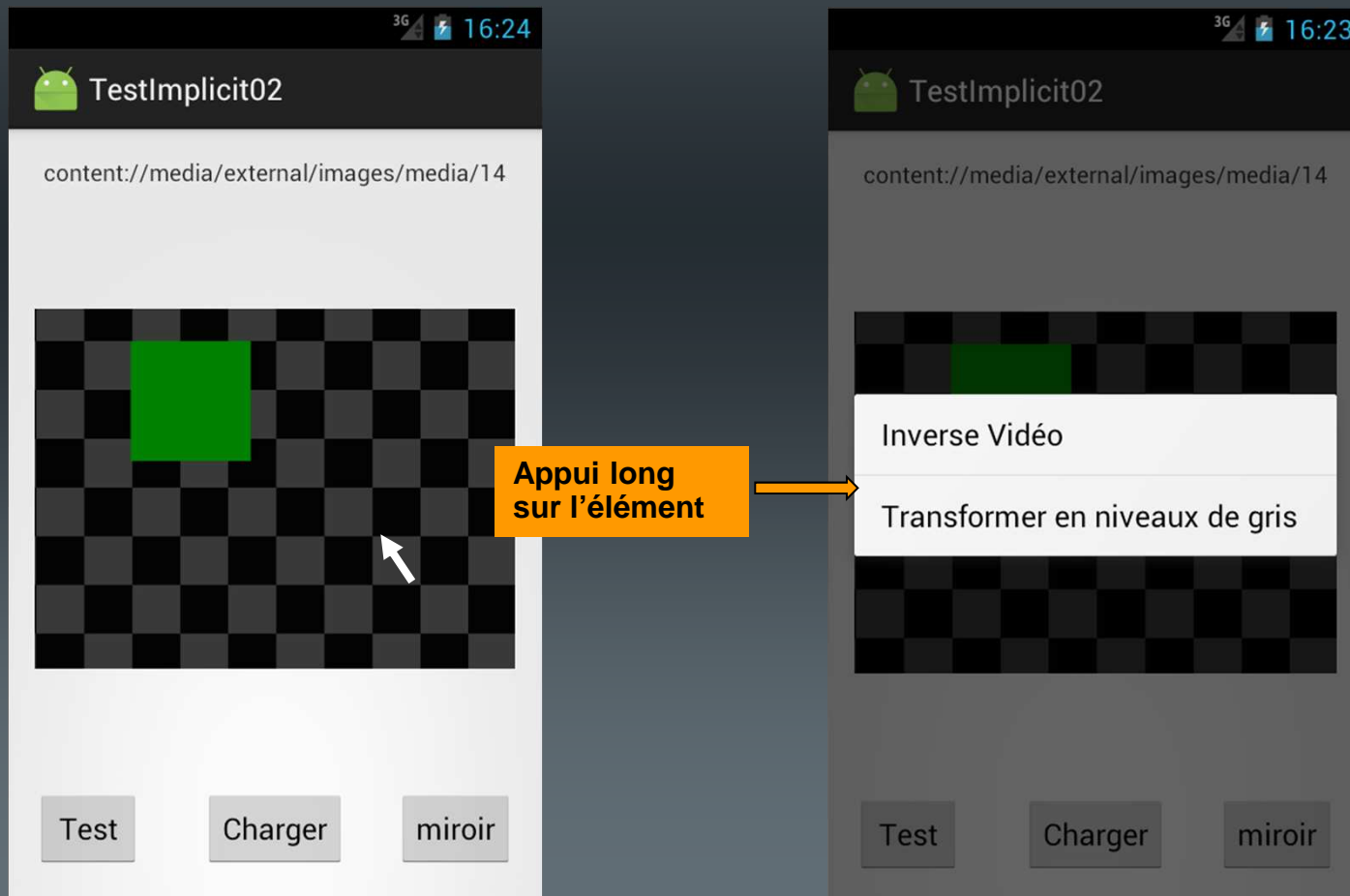


Sélection d'un sous-menu



Les menus contextuels (1)

- Associables à tout élément de l'interface graphique



Les menus contextuels (2)

- Utilisent un fichier XML décrivant un menu

1. Enregistrer l'élément pour lequel un menu contextuel doit être créé

```
registerForContextMenu(View view)
```

2. Surcharger la méthode onCreateContextMenu()

```
@Override
public void onCreateContextMenu(ContextMenu menu, View v,
                                ContextMenuInfo menuInfo) {

    super.onCreateContextMenu(menu, v, menuInfo);

    MenuInflater inflater = getMenuInflater();
    inflater.inflate(R.menu.context_menu, menu);
}
```

Les menus contextuels (3)

3. Surcharger la méthode onOptionsItemSelected()

```
@Override
public boolean onOptionsItemSelected(MenuItem item) {

    switch (item.getItemId()) {
        case R.id.xxx:
            ...
            return true;
        case R.id.yyy:
            ...
            return true;
        default:
            return super.onOptionsItemSelected(item);
    }
}
```

Plan du cours

- Introduction
- Architecture d'une application Android
- Les activités
- Définir une interface graphique
- Les intentions
- Les menus
- Les listes
- Les content providers

Les listes

- Éléments très utilisés sous android
 - Facilitent l'affichage et la manipulation d'ensembles de données importants sur des écrans de taille réduite
 - Possibilité de faire défiler/sélectionner les items
 - Paramétrage possible de la présentation des items via un *layout*
 - Remplissage « automatique » à partir de données extraites de tableaux ou de bases de données
 - Nécessitent l'utilisation d'un adaptateur

Examples

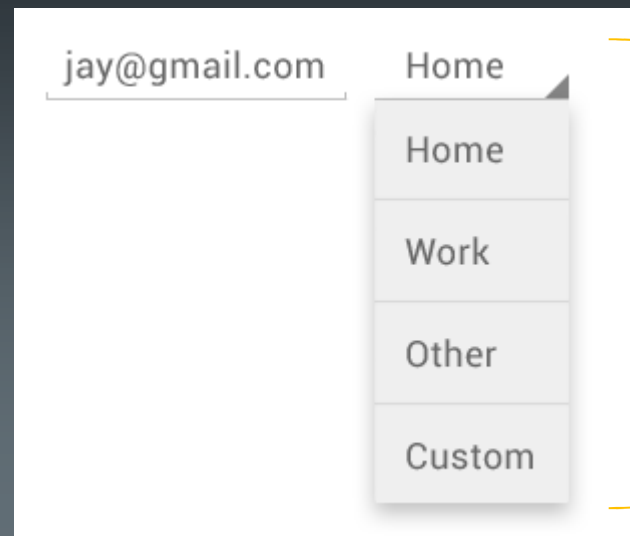


ListView

GridView



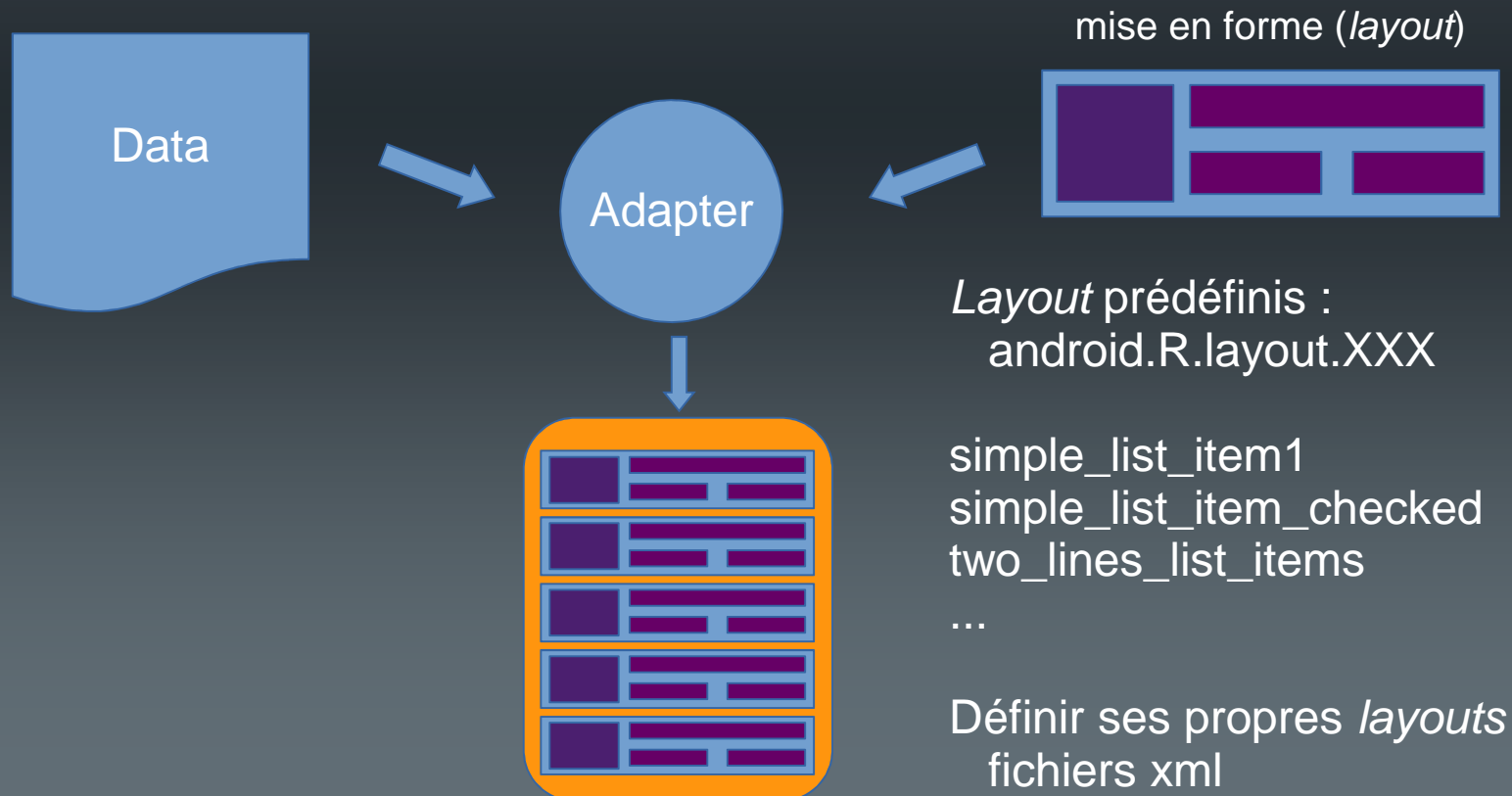
Spinner



Les adaptateurs (1)

■ Rôle

- Adapter les données au format d'affichage
- Nécessité de préciser le layout utilisé pour les items



Les adaptateurs (2)

- Différentes classes prédéfinies
 - *ArrayAdapter* : utilise un tableau d'objets quelconques ou une liste java
 - *SimpleCursorAdapter* : utilise des ensembles de données issus de requêtes sur des bases de données
 - *SimpleAdapter* : utilise des maps {(clé, valeur),...}

Le contexte de l'activité (this, getContext())

`ArrayAdapter(Context c, int ressource) ;`
`ArrayAdapter(Context c, int ressource, T[] objets) ;`

L'identifiant du fichier ressource
décrivant le layout à utiliser
(R.id....)

Les objets à insérer dans
une liste via l'adaptateur

Les adaptateurs (3)

- Quelques méthodes utiles (ArrayAdapter)
 - `int getCount()` : nb d'objets gérés par l'adaptateur
 - `add(T objet)` : ajoute l'objet à la fin de la liste
 - `insert(T objet, int position)` : insertion de l'objet à une position donnée
 - `remove(T objet)` : suppression de l'objet de la liste
 - `clear()` : suppression de tous les objets de la liste
 - `getItem(int position)`

Les ListView (1)

- Composant graphique qui gère l'affichage d'une liste
 - Nécessite un adaptateur
- Configuration XML

```
<ListView
```

```
    android:layout_width="wrap_content"
```

```
    android:layout_height="wrap_content"
```

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

```
    android:choiceMode="none" />
```

none : éléments non sélectionnables

singleChoice : 1 seul choix possible

multipleChoice : choix multiples

- Déclaration Java



L'adaptateur doit utiliser
le layout correspondant ...

```
ListView lv = (ListView)findViewById(R.id.maListe);
```

- Remplissage :

- Statique : depuis un tableau de données xml
- Dynamique : via un adaptateur

Les ListView (2)

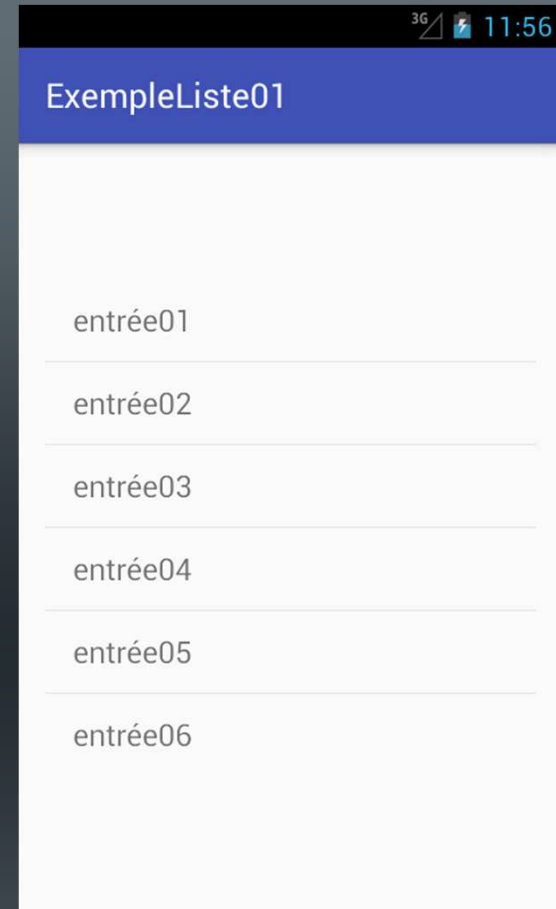
- Remplissage statique (xml)

res/values/strings.xml

```
<resources>
...
  <string-array name="maListe">
    <item>entrée01</item>
    <item>entrée02</item>
    ....
    <item>entrée06</item>
  </string-array>
...
</resources>
```

res/layout/content.xml

```
<RelativeLayout ... >
...
  <ListView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:id="@+id/maListe"
    android:entries="@array/maListe" />
...
</RelativeLayout>
```



Les ListView (3)

- Remplissage dynamique (java)

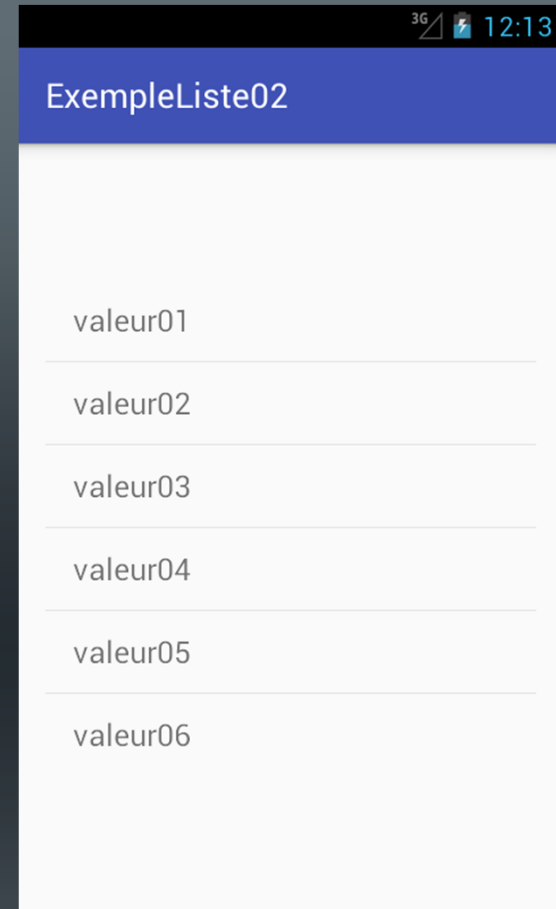
res/layout/content.xml

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

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

```
String entrees[]={"valeur01", "valeur02", "valeur03",  
                  "valeur04", "valeur05", "valeur06"};
```

```
ArrayAdapter<String> adapter =  
    new ArrayAdapter<String>(this,  
                             android.R.layout.simple_list_item_1,  
                             entrees);  
lv.setAdapter(adapter);
```



Plan du cours

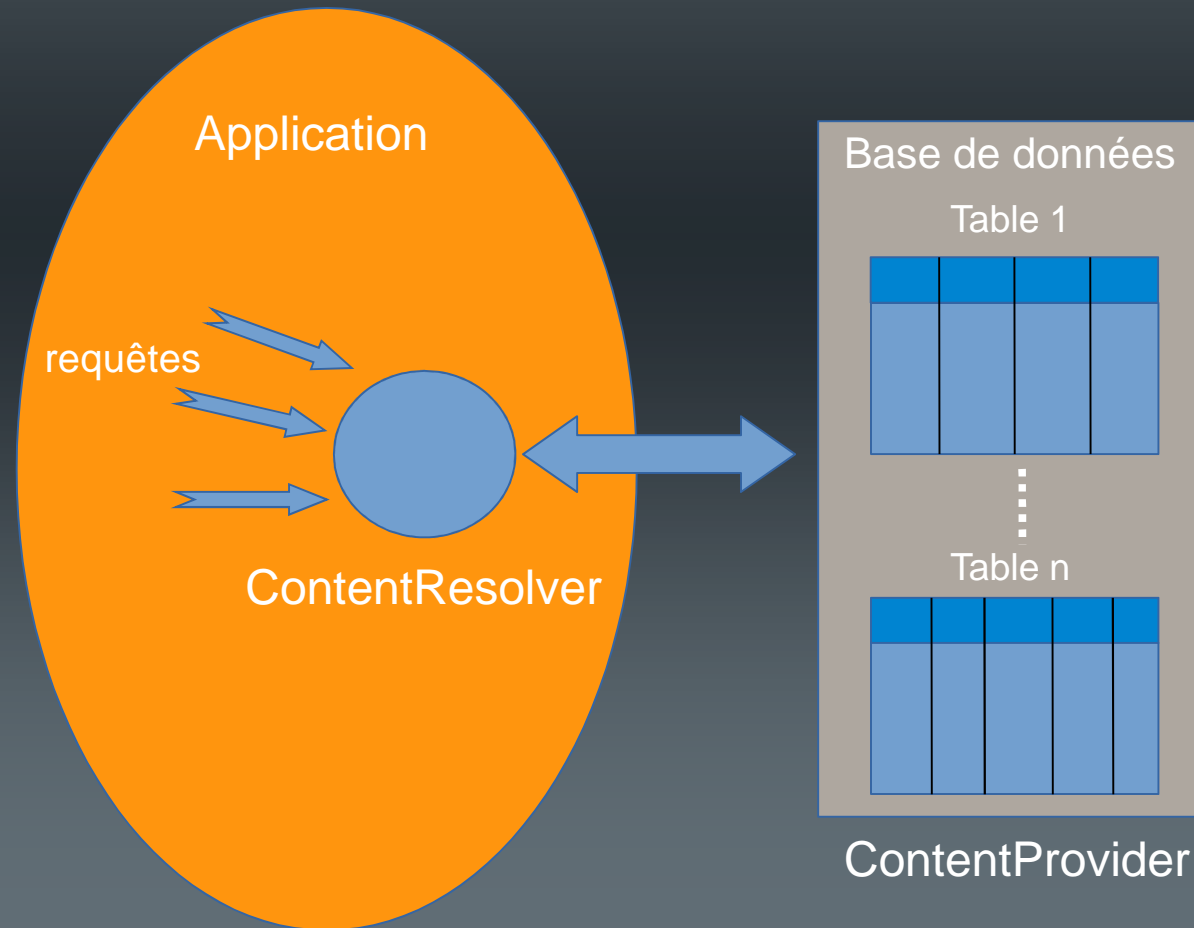
- Introduction
- Architecture d'une application Android
- Les activités
- Définir une interface graphique
- Les intentions
- Les menus
- Les listes
- Les content providers

Introduction

- Gestion des données par une application
 - Données locales/privées à une application
 - Sauvegardes dans des fichiers classiques
 - Sauvegardes dans des bases de données
 - SQL lite sous Android
 - Données extérieures à l'application (ex. contacts)
 - **Accès** via des fournisseurs de contenu
 - Nécessite de disposer des droits adéquats
 - Autoriser l'accès extérieur à des données de l'application
 - **Créer** un fournisseur de contenu

Les fournisseurs de contenu (1)

- Classe ContentProvider
 - Interface entre une application et des données



Les fournisseurs de contenu (2)

- Ils disposent

- d'une URI permettant de les identifier : content://authority/path
 - authority : le nom du fournisseur (UserDictionary)
 - path : le nom des structures/sous-structures gérées
- de données internes
 - Présentées à l'extérieur sous forme de tables

word	app id	frequency	locale	_ID
mapreduce	user1	100	en_US	1
precompiler	user14	200	fr_FR	2
applet	user2	225	fr_CA	3
const	user1	255	pt_BR	4
int	user5	100	en_UK	5

content://user_dictionary/words



Nom de la table

Nom du fournisseur de contenu

Les fournisseurs de contenu (3)

- Ils disposent

- D'une classe de contrat, qui définit les constantes facilitant la manipulation du fournisseur
 - URI
 - Nom des colonnes des tables
 - Etc.
- Exemple : classe `UserDictionary.Words`
 - `UserDictionary.Words.CONTENT_URI` (`content://user_dictionary/words`)
 - `UserDictionary.Words.WORD`
 - `UserDictionary.Words.FREQUENCY`

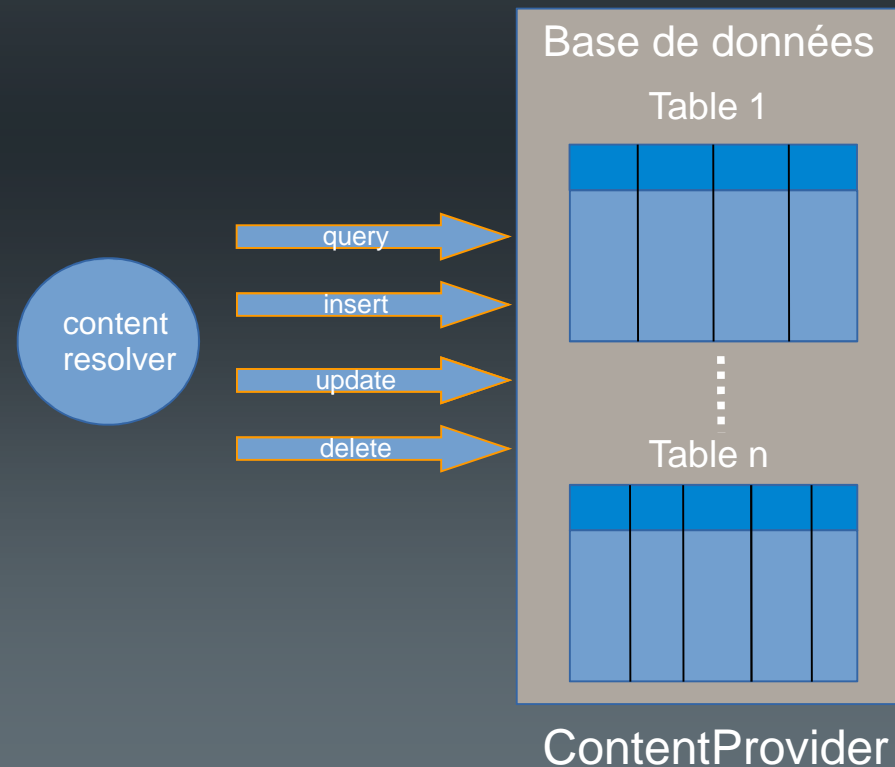
word	app id	frequency	locale	_ID
mapreduce	user1	100	en_US	1
precompiler	user14	200	fr_FR	2
applet	user2	225	fr_CA	3
const	user1	255	pt_BR	4
int	user5	100	en_UK	5

Les fournisseurs de contenu (4)

- Ils disposent
 - De permissions
 - Définies par le concepteur du fournisseur
 - Rendues publiques dans la « documentation »
 - Exemple :
 - `android.permission.READ_USER_DICTIONARY`
 - `android.permission.WRITE_USER_DICTIONARY`
 - Doivent être demandées dans le manifeste de l'application

Les content resolvers (1)

- Seul mécanisme d'accès au content providers
- Disposent des méthodes d'accès aux fournisseurs
 - query()
 - insert()
 - update()
 - delete()



Les content resolvers (2)

- Création :

```
ContentResolver resolver = getContentResolver();
```

Objet pouvant interagir avec n'importe quel fournisseur de contenu

➡ On fournit l'URI du fournisseur quand on souhaite l'utiliser ...

Exemples :

```
delete(UserDictionary.Words.CONTENT_URI, ...) ;  
insert(Calendars.CONTENT_URI, ...) ;  
update(Contacts.Phones.CONTENT_URI, ...) ;
```

Les content resolvers (3)

- La requête query()

L'URI du fournisseur de contenu.

La liste des noms de colonnes du fournisseur à récupérer.
(null = toutes les colonnes)

```
final Cursor query( Uri uri,  
                    String[] projection,  
                    String selection,  
                    String[] selectionArgs,  
                    String sortOrder)
```

Filtre de sélection : clause WHERE
de SQL sans le mot WHERE
(null = toutes les lignes)

Ordre de tri : clause ORDER BY
de SQL sans les mots ORDER BY
(null = ordre par défaut)

Arguments de sélection (les « ? » qui
apparaissent dans la sélection sont
remplacés par ces arguments)

Les content resolvers (4)

- Le paramètre **selection**

- Une chaîne contenant l'intégralité de la clause WHERE de la requête SQL
 - Inclus les critères liés (AND, OR, LIKE, ...)
 - Pas de mot-clé WHERE

- Problèmes de sécurité

- Injection de code SQL

```
String mySelection = " var = " + userInput;  
avec userInput valant " nothing; DROP TABLE *;"
```

- Utilisation du paramètre **selectionArgs**

- Contient les valeurs à sélectionner
 - Interprétées comme des String, pas comme des requêtes SQL

```
String mySelection = " var1 = ? AND var2 = ?" ;
```

```
String mySelectionArgs[] = {"toto" ,"titi" };
```



Les content resolvers (5)

- La requête query() : exemples

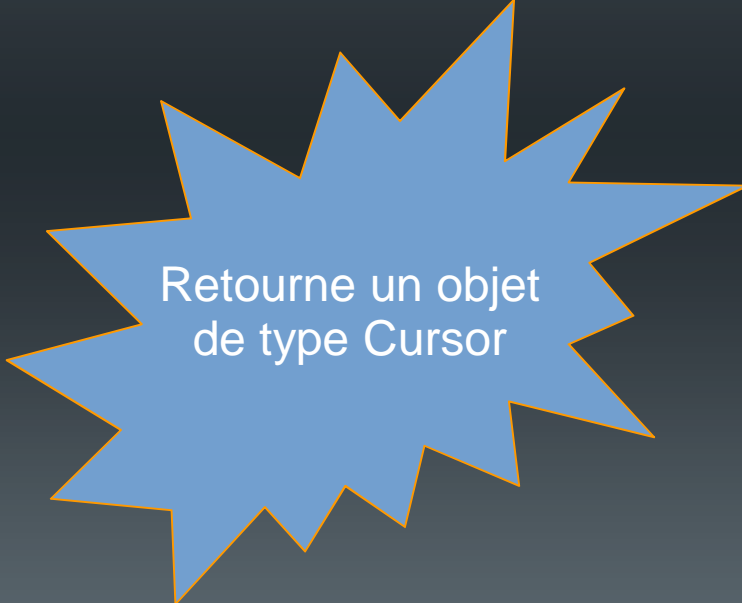
```
String[] projection = {  
    Calendars.CALENDAR_LOCATION,  
    Calendars.CALENDAR_TIME_ZONE  
}
```

```
String selection = {  
    Calendars.CALENDAR_LOCATION + " = ?"  
}
```

```
String[] selectionArgs = { " France" }
```

```
Resolver.query(Calendars.CONTENT_URI,  
    projection,  
    selection,  
    selectionArgs,  
    Calendars.CALENDAR_LOCATION+" ASC" );
```

```
Resolver.query(Calendars.CONTENT_URI,  
    projection,  
    null, null,null) ;
```

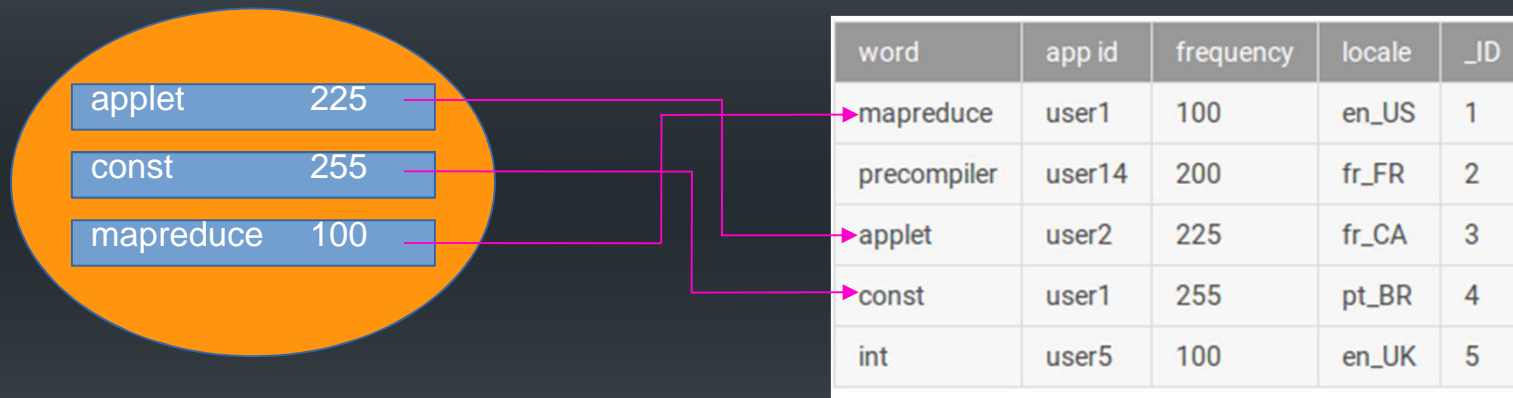


Retourne un objet
de type Cursor

```
SELECT CALENDAR_LOCATION, CALENDAR_TIMEZONE FROM Calendars  
WHERE CALENDAR_LOCATION='France' ORDER BY CALENDAR_LOCATION ASC
```

Les curseurs

- Objet représentant le résultat d'une requête
 - Liste des enregistrements (et colonnes) sélectionnés



- Diverses classes utilisables
 - Cursor
 - SimpleCursorAdapter

La classe Cursor

- Classe de base pour la récupération

```
ContentResolver resolver = getContentResolver();  
Cursor mcursor = resolver.query(...) ;
```

- Nombreuses méthodes

- getCount()
- getColumnIndex(String nomColonne)
- get**Type**(int index)
- moveToFirst(), moveToLast, moveToNext(), ...



Index pour les
données du curseur



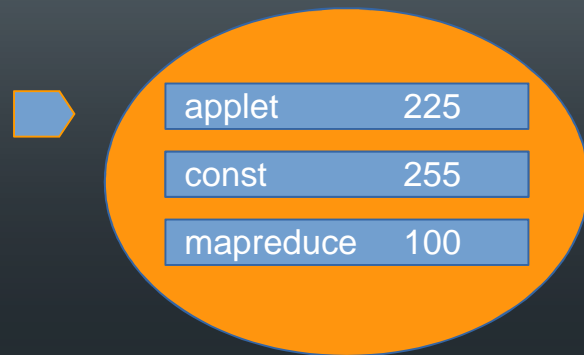
Gestion de la position du curseur dans la liste

La classe Cursor

■ Exemple

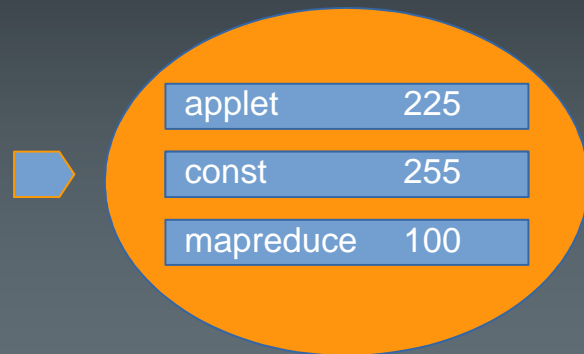
UserDictionary.Words

word	app id	frequency	locale	_ID
mapreduce	user1	100	en_US	1
precompiler	user14	200	fr_FR	2
applet	user2	225	fr_CA	3
const	user1	255	pt_BR	4
int	user5	100	en_UK	5



getCount() 3
getColumnIndex(UserDictionary.Words.FREQUENCY) 1
getString(0) applet
getInt(1) 225

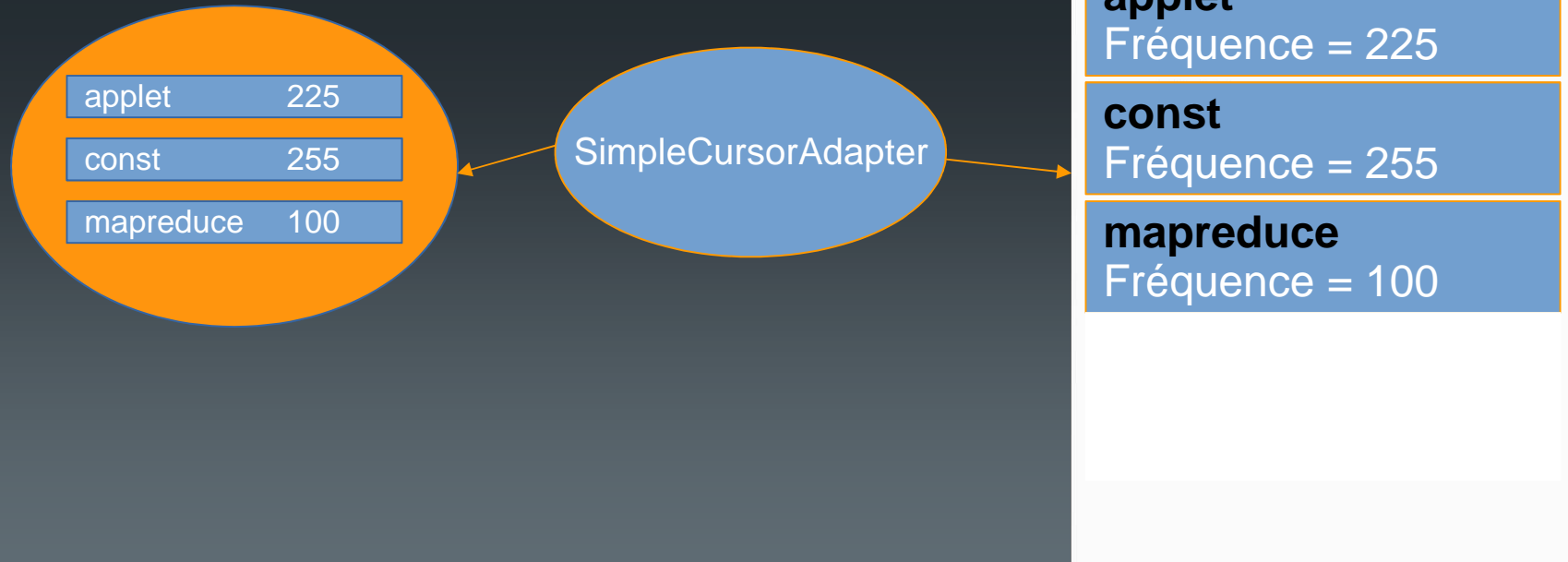
moveToNext()



getString(0) const
getInt(1) 255

La classe SimpleCursorAdapter

- Facilite le mapping entre :
 - Les données récupérées
 - Les vues qui doivent les afficher



La classe SimpleCursorAdapter

- Constructeur

SimpleCursorAdapter (Context context,

int layout,

Cursor c,

String[] from,

int[] to,

int flags)

← La mise en page de
chaque
ligne de données extraite
Les données extraites

← Le nom des colonnes du
fournisseur de contenu à
utiliser

← Le nom des vues vers
lesquelles envoyer les
données

← Flag gérant le comportement
de l'adaptateur

La classe SimpleCursorAdapter

- Exemple

android.R.layout.two_line_list_item

android.R.id.text1
android.R.id.text2

```
Cursor monCurseur = xxx.query(...);
```

```
SimpleCursorAdapter sca =  
    new SimpleCursorAdapter(this,  
        android.R.layout.two_line_list_item,  
        monCurseur,  
        new String[] {  
            Userdictionary.Words.WORD,  
            Userdictionary.Words.FREQUENCY  
        },  
        new int[] {  
            android.R.id.text1,  
            android.R.id.text2  
        }, 0);
```




Source : ark.media.mit.edu/~mike/Vanguard2012/i/androidall.jpg