Développement mobile sous Android

Fabien Teytaud

Université du Littoral Cote d'Opale

1er septembre 2014

Sommaire

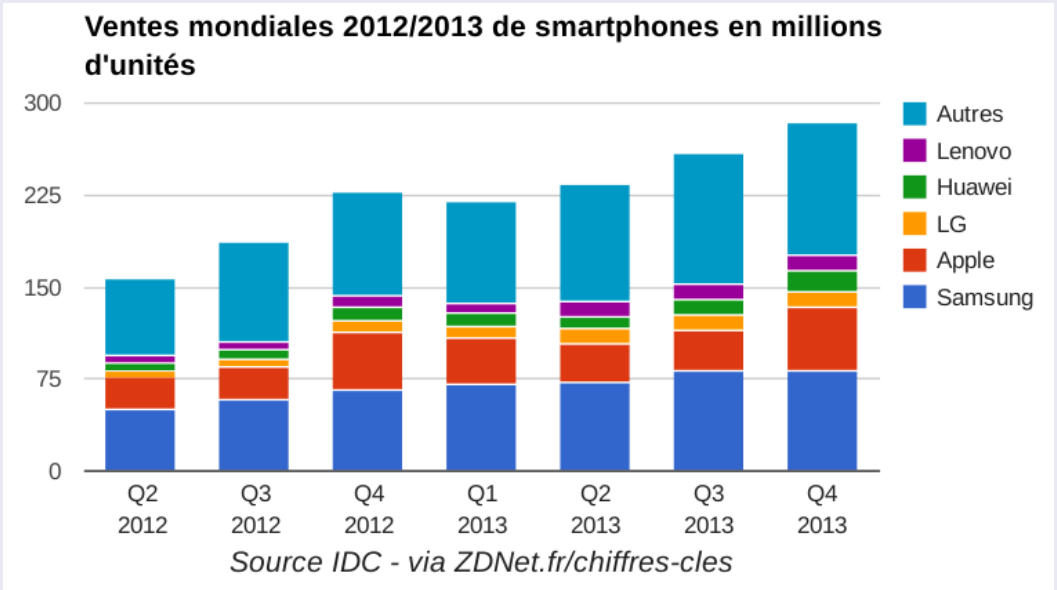
Pourquoi développer des applications mobiles ?

Explosion des smartphones

- Nombres d'abonnements mobiles :
  - En 2000 : < 1 milliard
  - En 2014 : presque 7 milliards

Pourquoi développer des applications mobiles ?

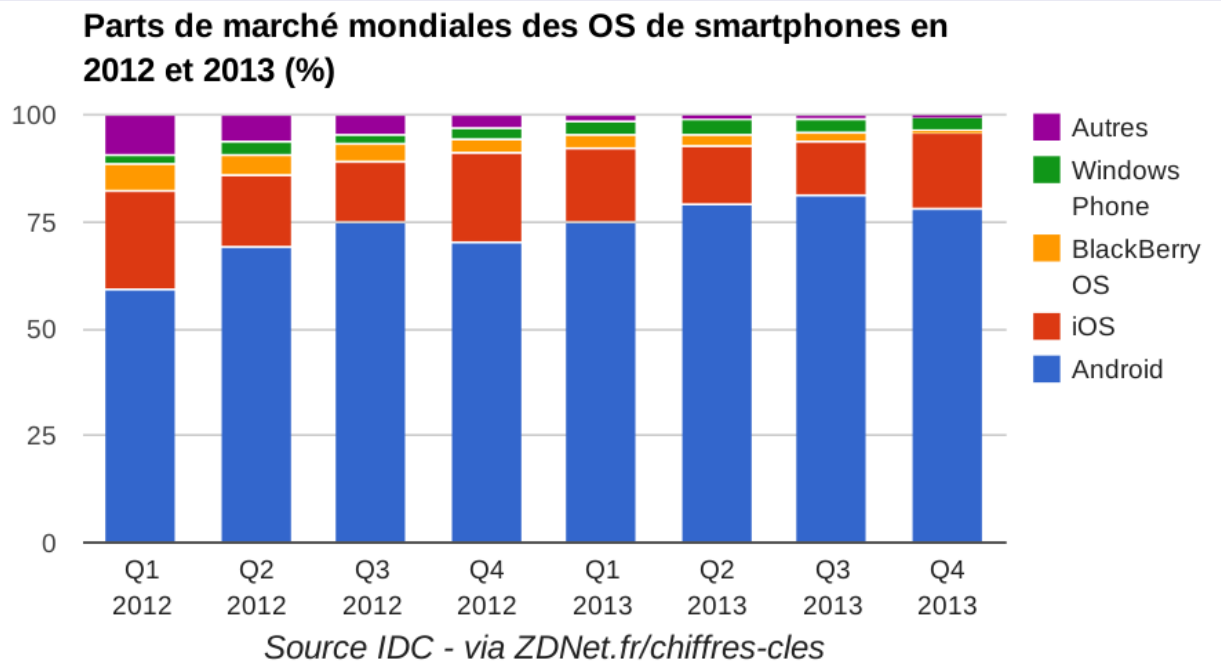
Explosion des smartphones



Prévision : x3 d'ici 2018 !

Pourquoi développer des applications Android ?

Répartition du marché



Principe

Qu'est ce qu'Android

- Plateforme pour les devices mobiles
- Gratuit
- Open source
- Flexible

Android inclue :

- Un système d'exploitation basé sur Linux.
- Des applications basiques (téléphones, contacts, ...).
- Un ensemble d'API avancées
- SDK basé sur un sous-ensemble de JAVA (existe aussi dans d'autres langages).

Les difficultés

Distribution des applications

- Dépôt centralisé ou décentralisé
- Depuis un PC

Hardware

- Hétérogénéité du matériel.
- Puissance et mémoire limitées.
- Connectivité à internet (disponibilité, rapidité, ...).
- Dispositifs d'affichage nombreux et variés.
- Interface tactile.
- Interaction avec le système sans l'interrompre.

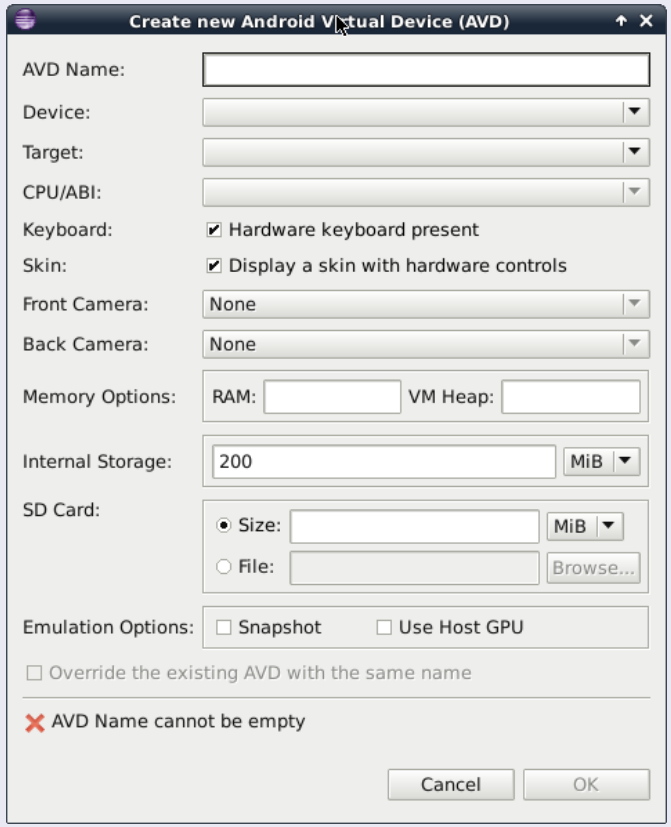
Installation

Installation sous linux

- Installer eclipse.
- Installer le SDK d'Android :
  - `http://developer.android.com/sdk/index.html`
- Dans Eclipse, installer ADT.
  - "Install new software".
  - `http://dl-ssl.google.com/android/eclipse`

Installation

Émulateur de téléphone : Android Virtual Device (AVD)



Sommaire

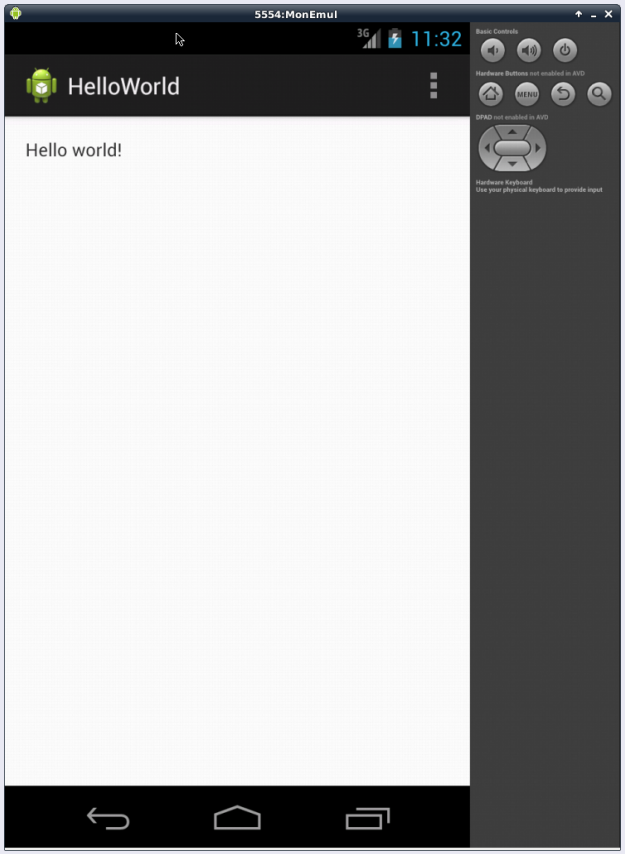
Une première application

Hello World !

- Créer un nouveau "Android Application Project".
- Donner comme nom "HelloWorld".
- Laisser tous les autres paramètres par défaut.

Une première application

Hello World !



Notion d'activité

Définition

- Une activité est un composant d'une application qui fournit un écran avec lequel l'utilisateur peut interagir.
- Une application contient (généralement) plusieurs activités.
- Activité = contexte + interface graphique (unique).

Organisation

- Que se passe t-il si on reçoit un message alors que l'utilisateur utilise le navigateur ?
- Il existe une notion de priorité.
- Une activité peut être dans différents états.
- Les activités sont empilées :
  - Une seule activité est visible à la fois.
  - Une seule activité est active à la fois.

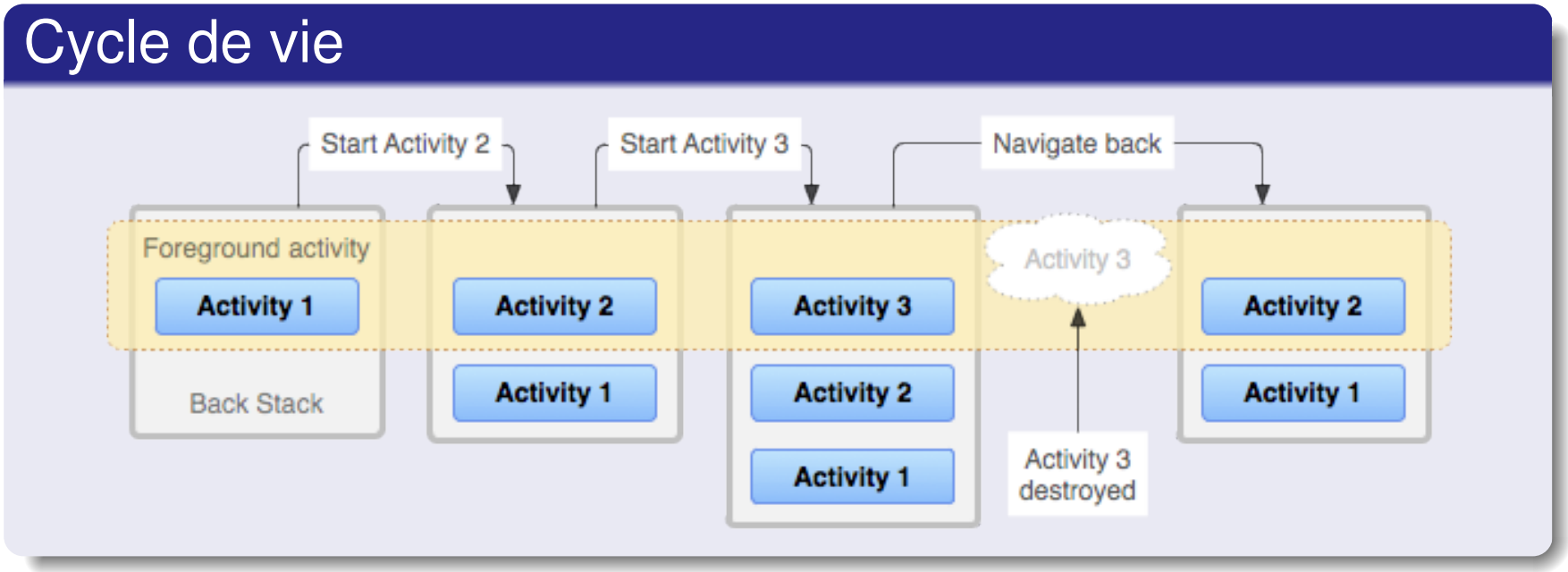
Activité

Différents états

- Active : activité visible en totalité.
- Suspendue : activité partiellement visible.
- Arrêtée : activité non visible.

Activité

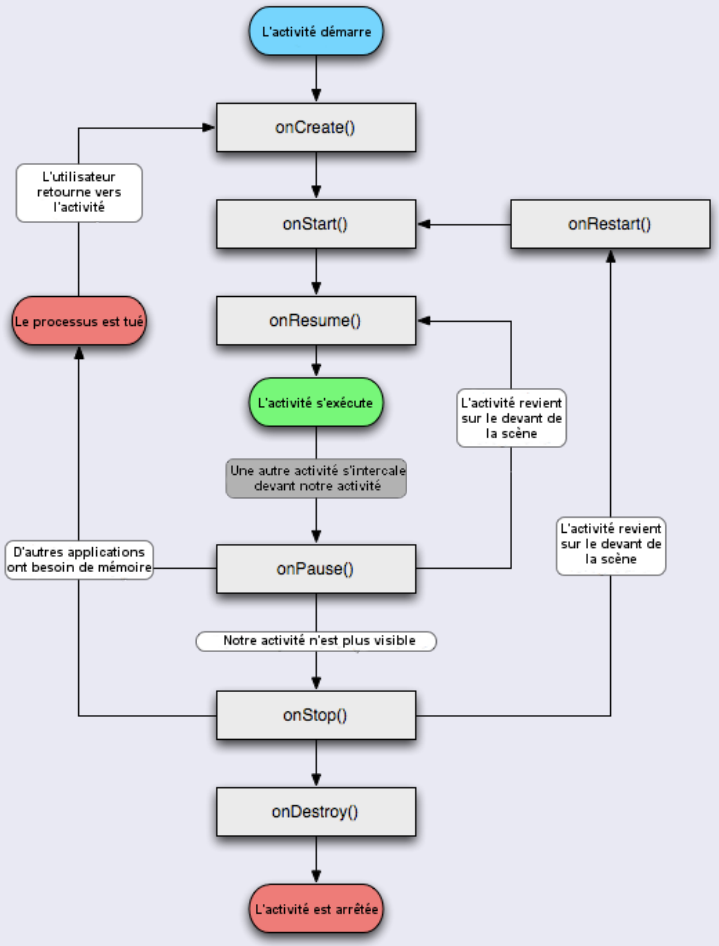
Cycle de vie



Source : `http://www.edureka.in/`

Activité

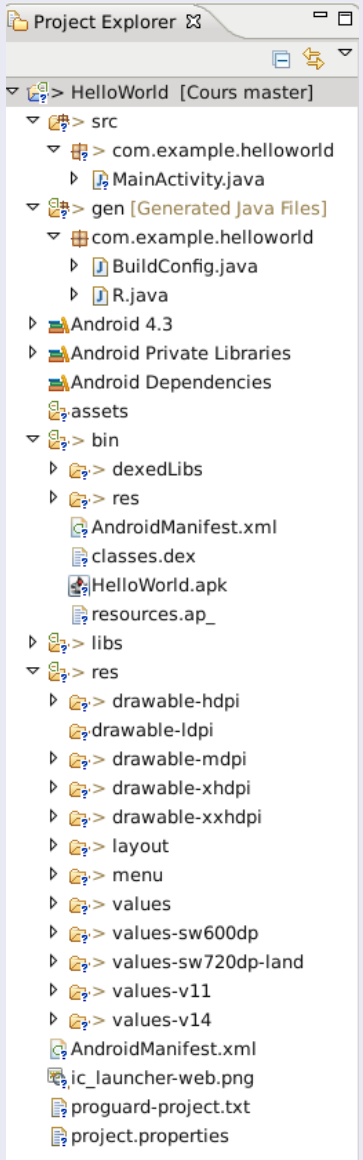
Cycle de vie



Source : `http://fr.openclassrooms.com`

Sommaire

Explorer


Project Explorer
> HelloWorld [Cours master]
> src
> com.example.helloworld
MainActivity.java
> gen [Generated Java Files]
com.example.helloworld
BuildConfig.java
R.java
Android 4.3
Android Private Libraries
Android Dependencies
assets
> bin
> dexedLibs
> res
AndroidManifest.xml
classes.dex
HelloWorld.apk
resources.ap_
> libs
> res
> drawable-hdpi
drawable-ldpi
> drawable-mdpi
> drawable-xhdpi
> drawable-xxhdpi
> layout
> menu
> values
> values-sw600dp
> values-sw720dp-land
> values-v11
> values-v14
AndroidManifest.xml
ic_launcher-web.png
proguard-project.txt
project.properties

Explorer

Répertoire src/

- Contient les fichiers sources.

Répertoire gen/

- Contient les fichiers générés automatiquement.
- Contient le fichier R.java.

Répertoire res/

- res/drawable/ : Dessin et image.
- res/layout/ : Mise en page ou interface graphique.
- res/values/ : Variable (en particulier les chaînes de caractères).

Le fichier de configuration : AndroidManifest.xml

<uses-sdk>

- Version minimum d'Android gérée par l'application.
- Version visée d'Android.

<application>

Un fichier manifest par application. Les options principales de l'application sont définies ici.

<activity>

Déclare une activité.

Exercice

Exercice

Modifier (proprement) votre application HelloWorld pour afficher le message "Voici ma première application".

Sommaire

Introduction

Préceptes

- Beauté : l'élégance de votre application est la première chose que verra l'utilisateur.
- Simplicité :
  - Votre application doit être intuitive.
  - Ne pas faire de longs messages.
  - La navigation entre les différentes activités doit être le plus rapide possible.
  - L'utilisateur doit toujours être informé de l'état courant de l'application.
- Différents conseils : `http://developer.android.com/design/index.html`.

Vues et widgets

Vue

- Chaque activité en possède une.
- Deux constructions possibles :
  - **Définie à l'aide d'un fichier XML.**
  - Définie par instanciation des widgets totalement en JAVA.

Widget

- Éléments graphiques constituant la vue.
- Widgets simples : boutons, champs de saisie, radios, listes, ...
- Widgets complexes : datepicker, horloge, barre de progression ...
- Définition en XML et implémentation du comportement en JAVA.

Fichier XML basique : activity_main.xml

- xmlns :android="http ://schemas.android.com/apk/res/android"
- xmlns :tools="http ://schemas.android.com/tools"

Ces deux lignes permettent d'utiliser les attributs spécifiques à Android.

Fichier XML basique : activity_main.xml

Layout XML

- Un layout XML est une représentation de l'organisation des widgets (mais aussi d'autres Layout).
- Il existe différents modèles : Linear, Relative, Grid ...

Attributs classiques

- *layout_width* : largeur de la vue.
- *layout_height* : hauteur de la vue.
  - *match_parent* : taille du parent sur l'axe concerné.
  - *wrap_content* : taille la plus petite pour coller aux données.
  - Une valeur numérique.

@string/firstApp

Affiche la ressource qui se trouve dans la classe *string* de *R.java* et qui s'appelle *firstApp*.

TextView

Un textview permet d'afficher du texte.

Exemple

```
<TextView
   android:id="@+id/text"
   android:layout_width="wrap_content"
   android:layout_height="match_parent"
   android:text="@string/letexte" />
```

Le champ "@+id/..." permet de spécifier à Android un identifiant à la vue.

EditText

Un edittext permet à l'utilisateur d'écrire du texte.

Exemple

```
<TextView
   android:id="@+id/text"
   android:layout_width="wrap_content"
   android:layout_height="match_parent"
   android:inputType="textMultiLine"
   android:lines="3"
   android:hint="@string/letexte" />
```

- android :autoText : correction automatique.
- android :capitalize : mets automatiquement une majuscule à la première lettre.
- android :digit : n'accepte que des chiffres.
- android :singleLine : permet de passer au champ suivant avec la touche entrée.
- android :autoCompleteTextView : utilise l'autocompletion Android.

Button

La classe button permet de créer un bouton (un textview cliquable).
Les attributs sont identiques à ceux de TextView.

Exemple

```
<Button
   android:id="@+id/b1"
   android:layout_width="wrap_content"
   android:layout_height="wrap_content"
   android:text="@string/boutonTexte" />
```

- Possibilité de rajouter une image avec android :background.
- Bouton à deux états (ON/OFF) : classe ToggleButton.
- Possibilité de définir une image par état avec la classe StateListDrawable.

Switchs

Variante de case à cocher sous la forme d'un switch à deux états (interrupteur).

Exemple

```
<Switch
   android:id="@+id/s1"
   android:layout_width="wrap_content"
   android:layout_height="wrap_content"
   android:text="@string/switchTexte" />
```

- getTextOn() : renvoie le texte associé à l'état 1 du switch.
- getTextOff() : renvoie le texte associé à l'état 2 du switch.
- setChecked() : met le switch à l'état 1.

CheckBox

La classe checkbox permet de créer une boite à cocher (composant à deux états).

Exemple

```
<CheckBox
   android:id="@+id/c1"
   android:layout_width="wrap_content"
   android:layout_height="wrap_content"
   android:text="@string/texte"
   android:checked="true" />
```

android :checked permet de spécifier que la case est cochée par défaut.

- isChecked() permet d'accéder à l'état.
- setChecked() permet de modifier l'état.
- toggle() permet d'inverser l'état.

RadioButton

Similaire à checkbox mais une seule case cochée à la fois. Il est recommandé de les regrouper dans un RadioGroup.

Exemple

```
<RadioGroup
   android:id="@+id/idGroupe"
   android:layout_width="wrap_content"
   android:layout_height="wrap_content"
   android:orientation="vertical"
   android:checkedButton="@+id/radioAdd">
   <RadioButton
      android:id="@+id/radioAdd"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text="@string/plus" />
   <RadioButton
      android:id="@+id/radioDiv"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text="@string/div" />
</RadioGroup>
```

Spinner

Menu déroulant.

Exemple

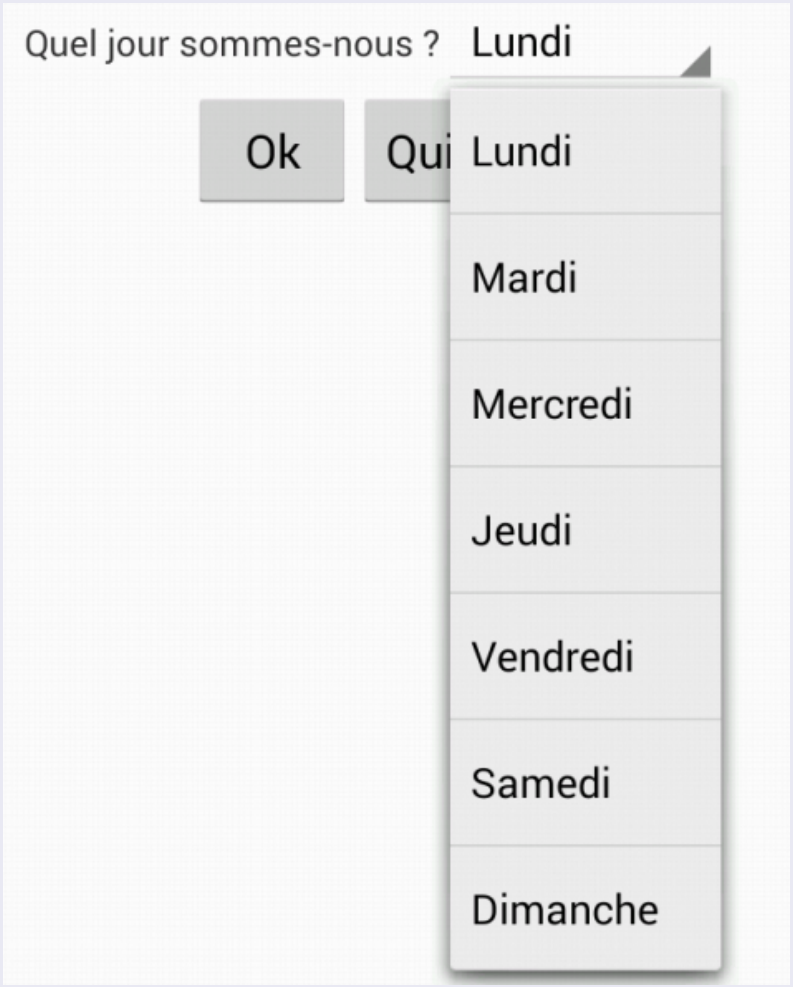
```
<Spinner
   android:id="@+id/lesjours"
   android:layout_width="wrap_content"
   android:layout_height="wrap_content"
   android:entries="@array/lesjours_array"
/>
```

Dans strings.xml

```
<string-array name="lesjours_array">
<item>Lundi</item>
<item>Mardi</item>
<item>Mercredi</item>
<item>Jeudi</item>
<item>Vendredi</item>
<item>Samedi</item>
<item>Dimanche</item>
</string-array>
```

Spinner

Exemple



Widgets

Documentation Android

Il existe de nombreux autres widgets :

`http://developer.android.com/`

Exercices

1. TP1.

Positionner et organiser les vues

Les layouts

- **LinearLayout** : alignement linéaire (gauche-droite ou haut-bas)
- **RelativeLayout** : positionnement les uns en fonction des autres. Le premier sert de référence.
- **FrameLayout** : Positionnement en haut à gauche avec priorité aux enfants.
- **GridLayout/TableLayout** : Positionnement comme un table.
- **ScrollView** : Fournit un défilement à son contenu.

Gestion des événements

Principe

- Gestion des événements à l'aide de listener.
- Différents types d'interactions (clic court, clic long, effleurement).
- 2 possibilités :
  - Implémentation d'un objet JAVA (OnClickListener, OnLongListener, ...) lié ensuite au widget.
  - En implémentant directement l'objet xml.

Exemple 1

Vue

```
<Button
    android :id="@+id/boutonQuitter"
    android :layout_width=“wrap_content”
    android :layout_height=“wrap_content”
    android :text=“@string/Quitter”
    android :onClick=“onQuitte” />
```

Code Java

```
public void onQuitte(View view) {
        finish() ;
}
```

Exemple 2

Vue

```
<Button
    android :id="@+id/boutonQuitter"
    android :layout_width=“wrap_content”
    android :layout_height=“wrap_content”
    android :text=“@string/Quitter”
```

Code Java

```
quitter = (Button)findViewById(R.id.boutonQuitter) ;
quitter.setOnClickListener(quitterListener) ;
```

```
public OnClickListener quitterListener = new OnClickListener() {
    @Override
    public void onClick(View v) {
        finish() ;
    }
} ;
```

Application simple

Une seule activité !

1. TP2.

Les listes

Une première liste, en xml

```
<ListView
        android :id="@+id/list"
        android :layout_width="match_parent"
        android :layout_height="match_parent"
        android :drawSelectorOnTop="false"
/>
```

Les listes

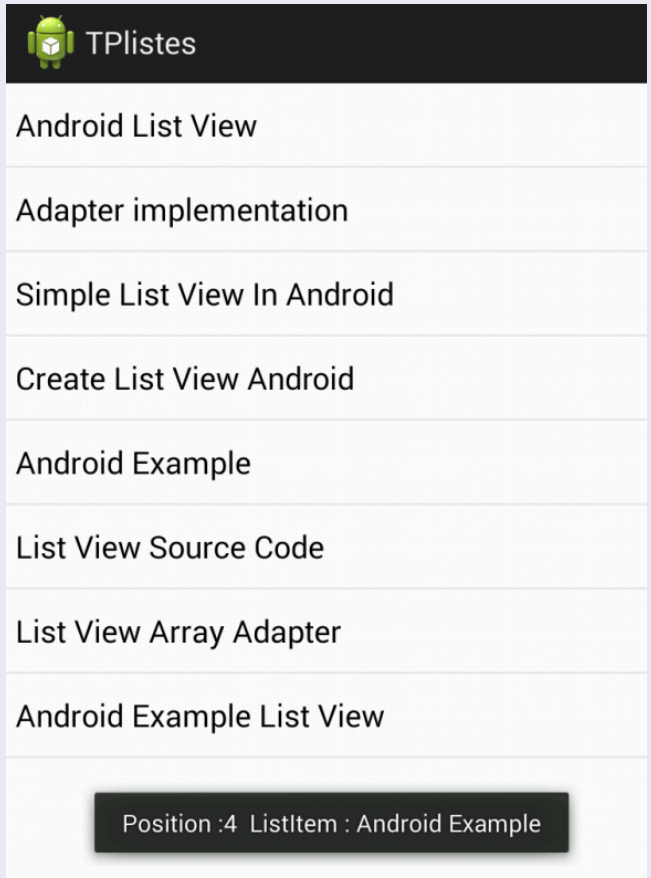
... et en java

```
ListView listView ;
listView = (ListView) findViewById(R.id.list) ;
String[] values = new String[]{ ... } ;

ArrayAdapter<String> adapter = new ArrayAdapter<String>(contexte,
idLayout, values) ;

listView.setAdapter(adapter) ;
```

- Le idLayout correspond à la mise en page de l'élément. Vous pouvez créer la votre, mais un certain nombre sont à disposition (par exemple android.R.layout.simple_list_item_1).
- Clic : public void onItemClick(AdapterView< ?> parent, View view, int position, long id) { }

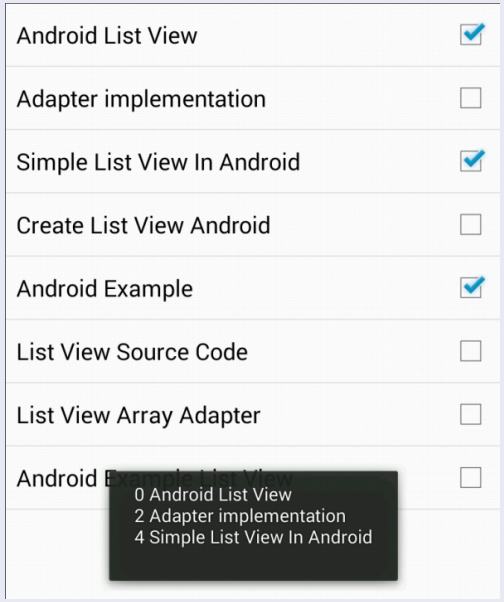
Les listes

Et ça donne :



Plusieurs choix

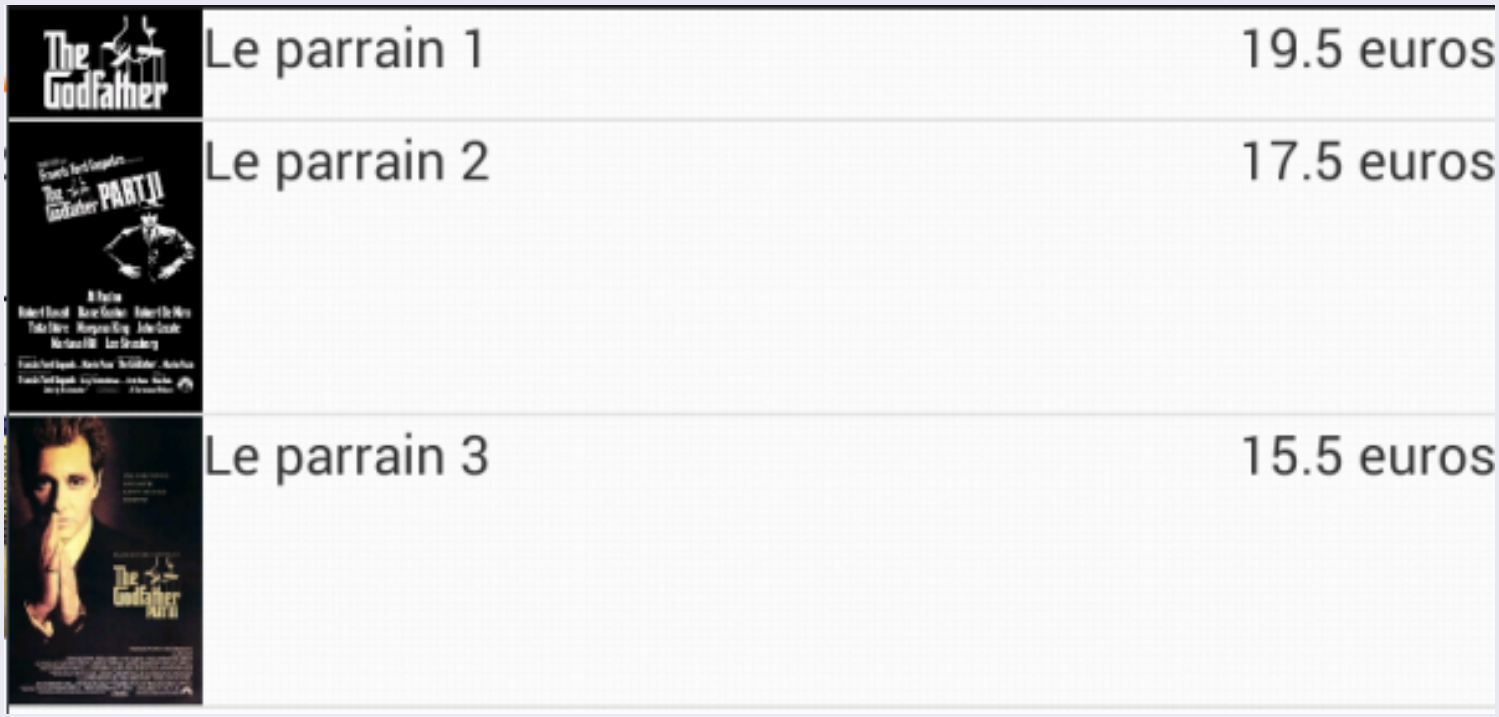
Liste multiple



- android :choiceMode="multipleChoice"
- android.R.layout.simple_list_item_multiple_choice
- listView.getCheckedItemPositions()

Les listes

Adapter sur mesure



- Définir un xml pour le layout,
- et créer une classe héritant de BaseAdapter.
- Utilisation d'un support (ViewHolder) pour une utilisation plus rapide.

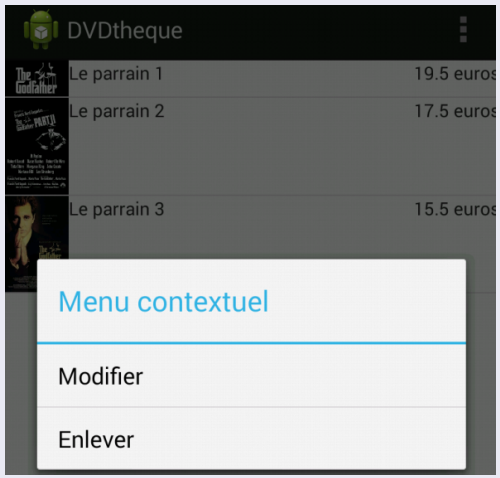
Le menu de base

Menu d'option basique

- Création : public boolean onCreateOptionsMenu(Menu menu) { menu.add(...) }.
- Utilisation : public boolean onOptionsItemSelected(MenuItem item) { ... }.

Menu contextuel

Menu sur un widget, un bouton, une ListeViev ...



- Création : public void onCreateContextMenu(ContextMenu menu, View v, ContextMenu.ContextMenuInfo menuInfo) { ... }
- Utilisation : public boolean onContextItemSelected(MenuItem item) { ... } //clic long

TP3

TP3

TP sur les listes, ici.

Sommaire

Une application : plusieurs activités

Plusieurs activités

- On a vu : l'activité principale (celle ouverte directement par l'utilisateur).
- Souvent on a besoin d'activités indépendantes (options, aide, menu ...).
- Communication entre les différentes activités à l'aide des "intentions".

Les intentions

- Une catégorie : LAUNCHER, DEFAULT ou ALTERNATIVE.
- Un type MIME : type de ressource sur laquelle on travaille.
- Un composant : l'activité qui doit recevoir l'intention.
- Des 'extras" : un Bundle d'autres informations destiné à l'activité réceptrice.

Lancement de sous-activités (dans une même application).

Déclarer une intention

- new Intent(this, SecondActivity.class)
- Indique que l'activité SecondActivity doit être lancée.
- Récupérer l'activité fille :
  - startActivity()
  - startActivityForResult()
  - Réponse : onActivityResult()
    - Identifiant de startActivityForResult correspondant.
    - Un code de réponse (généralement, RESULT_OK ou RESULT_CANCELLED).
    - Un objet String facultatif (résultat éventuel).
    - Un objet Bundle facultatif (contentant des informations supplémentaires).

Lancement de sous-activités (dans une même application).

Lancement

- StartActivity(Intent) : lance l'activité (passée à l'intent).
- StartActivityForResults(Intent, requestCode) : lance l'activité (passée à l'intent) et attend un résultat.
- setResult(RESULT_OK) : pour spécifier un resultCode.
- onActivityResult(int requestCode, int resultCode, Intent i) : pour traiter le résultat.

Communications

Exercices

1. TP4.