

# Formation ANDROID

15 au 17 octobre 2014

# Qui suis-je ?

- Anthony Monteiro
- Application sur le store :
  - UrbanPulse
  - Acadomia (iOS & Android)
- Site internet : [www.amonteiro.fr](http://www.amonteiro.fr)
- Contact : [contact@amonteiro.fr](mailto:contact@amonteiro.fr)

# Plan

- Premiers pas
  - Présentation / Installation de l'IDE
- Rappels sur Java
- L'univers d'Android
- Architecture d'un projet
  - Les différents types de fichiers et classes
  - Les Activités
- IHM
  - Layouts, composants graphiques
  - Gestion des événements
  - ListView
  - Communication entre activités

# Plan

- Les fragments
- (Opt) AlertDialog, Toast et Menu
- BroadcastReceiver
- Services
  - [Librairie Otto](#)
- Conseils d'architecture
- Handler
- AsyncTask
- Ergonomie et User Expériences
- SQLite
  - [Avec et sans GreenDAO](#)

# Plan

- Découverte de librairies
  - GraphView
  - Zbar
  - Facebook
  - Scribe
- (Opt) Notification et GCM

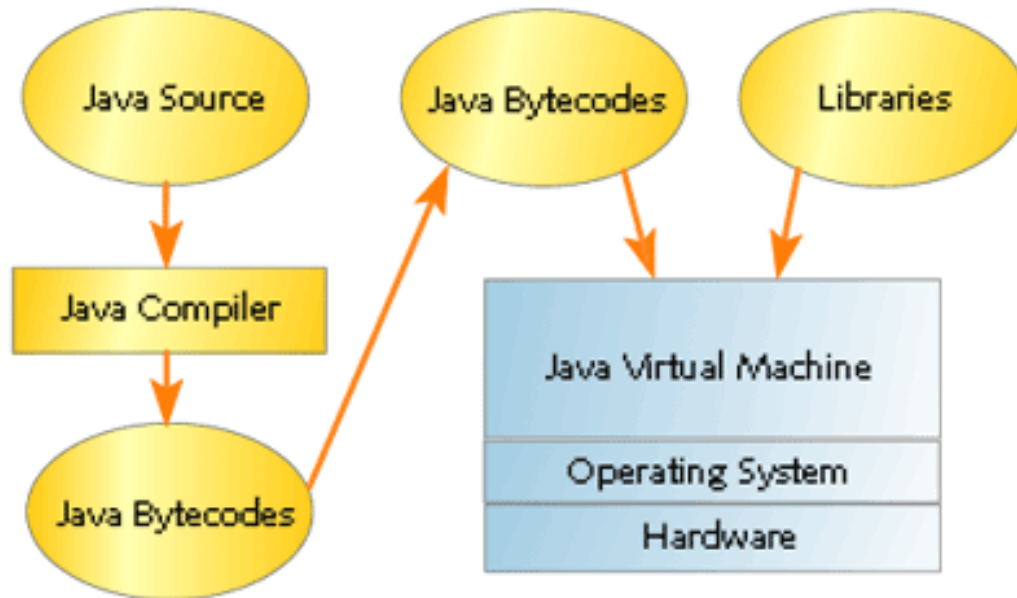
# Charger les projets

- Créer un compte GitHub
  - Anth06ny
- Chargement du projet de formation
  - <https://github.com/Anth06ny/formationExo/tree/master>

# Présentation et tour de table.

# Rappels Java

- La JVM





# Rappels Java

- Les collections

	Utilisation générale
List	ArrayList LinkedList
Set	HashSet TreeSet LinkedHashSet
Map	HashMap TreeMap LinkedHashMap

- `SparseArray HashMap<Integer, ?>`

# Rappels Java

- Java.util
  - Arrays, Calendar, Date, Random, Timer...
- Notion d'objet et de pointeur.
- Commons.lang (StringUtils, NumberUtils...)
  - `If(StringUtils.isNotBlank(var)) {`  
...  
`}`

# Rappels Java

- Interface

//Déclaration

```
public interface OnClickListListener {  
    void onClickOnList(Eleve eleve);  
}
```

//Assignment

```
public void setOnClickListListener(final OnClickListListener onClickOnList) {  
    OnClickOnList = onClickOnList;  
}
```

//Utilisation

```
if (OnClickOnList != null) {  
    OnClickOnList.onClickOnList(eleve);  
}
```

# Rappels Java

- Classe abstraite

//Déclaration

```
public abstract class Vehicule{  
    private int vitesseMax;  
    protected int getNbrRoue();  
    protected int getVitesseMax() { return vitesseMax; }  
}
```

//Utilisation

```
public class Moto extends Véhicule {  
    public Moto(){  
        super();  
        vitesse = 300;  
    }  
    @Override  
    protected int getNbrRoue() { return 2; }  
}
```

Présentation

# ANDROID

# Introduction

- Android est un système d'exploitation pour téléphone portable de nouvelle génération développé par Google. Celui ci met à disposition un kit de développement (SDK) basé sur le langage Java.
- OS complètement ouvert au développeur:
  - Lancer des appels, sms, emails
  - Accès au hardware (gps, appareil photo, wifi)
  - Accès à toutes les fonctionnalités du téléphone
  - => Applications plus riches

# L'Open Handset Alliance

- L'Open Handset Alliance (abrégé OHA) est un consortium de plusieurs entreprises dont le but est de développer des normes ouvertes pour les appareils de téléphonie mobile.
- Le consortium a été créé le 5 novembre 2007 à l'initiative de Google qui a su fédérer autour de lui 34 compagnies

# L'Open Handset Alliance

ASUS®

Bouygues  
Telecom

Sprint®

htc  
quietly brilliant

MOTOROLA



Google™

ARM®

• T • Mobile •

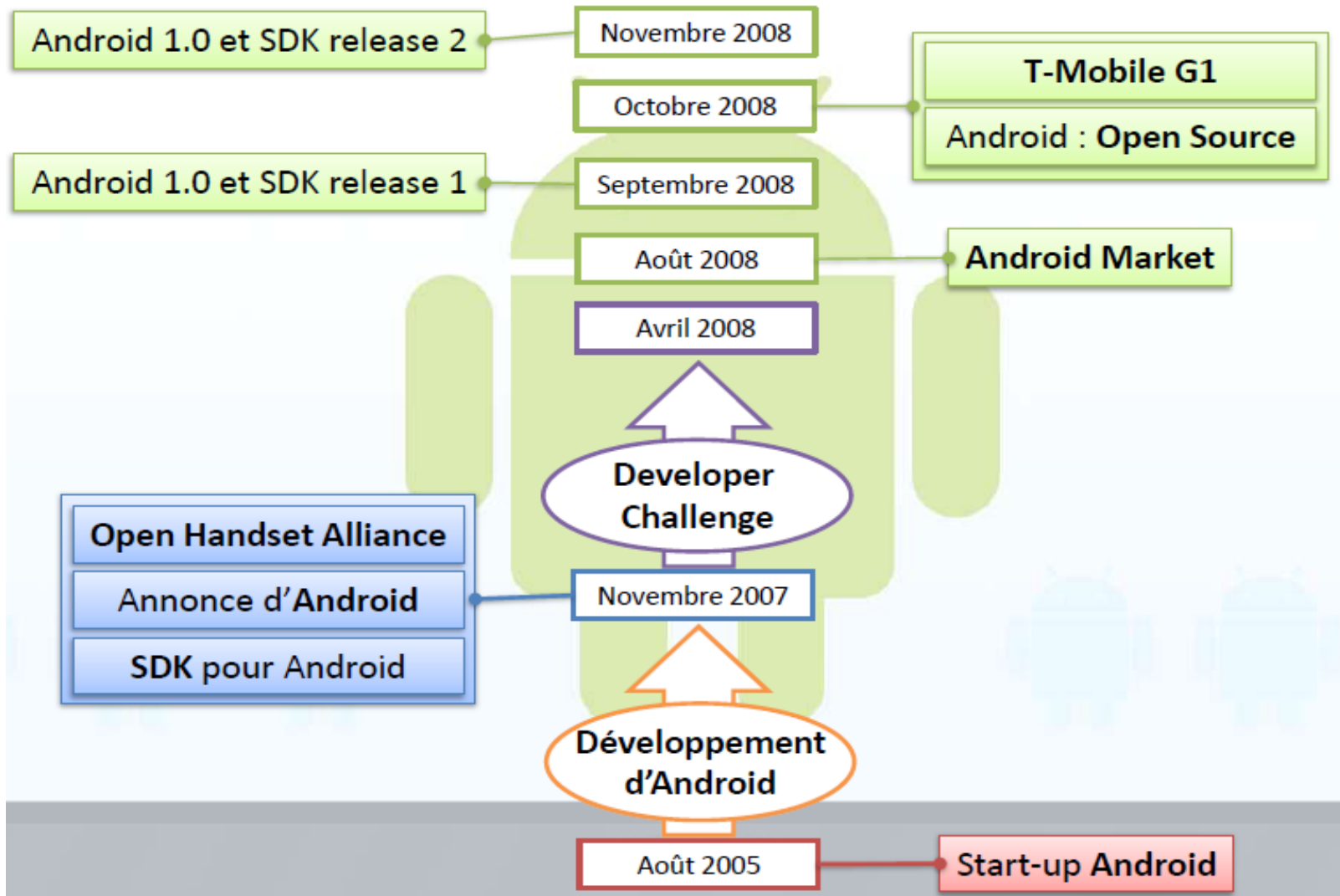
ebay

nvidia

vodafone™



# Evolution



# Evolution



# En chiffre



- 486M de smartphone vendu en 2011
- AngryBird 1M\$ / mois
- 2% des applications génèrent 90% des revenus

# Android c'est:

- un noyau Linux qui lui confère notamment des caractéristiques multitâches
- des bibliothèques graphiques, multimédias
- une machine virtuelle Java adaptée : la Dalvik Virtual Machine
- un framework applicatif proposant des fonctionnalités de gestion de fenêtres, de téléphonie, de gestion de contenu...
- des applications dont un navigateur web, une gestion des contacts, un calendrier...

# En image...



# Développer pour Android

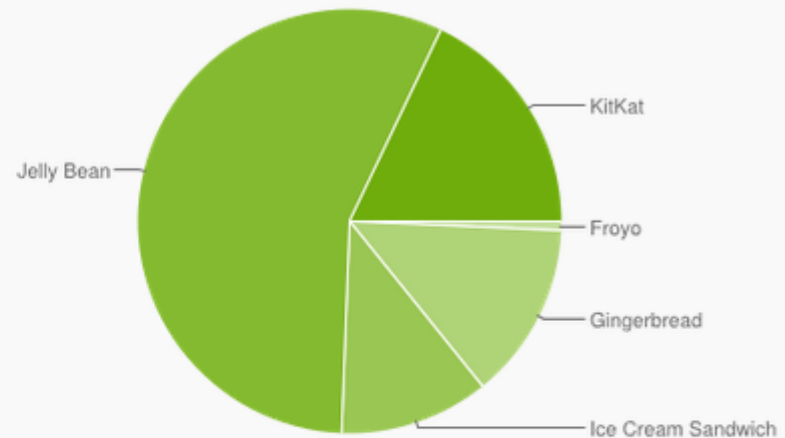
- Gratuit
- Application distribuable rapidement
  - Par mail
  - Par son propre «store»
  - Via Google Play (payant)
- Programmation en Java
  - Orientée Objet
  - Possibilité de réutiliser du code métier existant
- API Android / Google Service
  - Pas besoin d'écrire du code bas niveau

# Version de l'API android

- Les API (classes et méthodes) android évoluent avec chaque nouvelle version de l'OS.
- Mai 2013:
  - Les devices avec une version 2.2 ou supérieure représentent 98% du marché
  - Les devices avec une version 4.0 ou supérieure représentent 55% du marché.

# Répartition des OS par version

Version	Codename	API	Distribution
2.2	Froyo	8	0.7%
2.3.3 - 2.3.7	Gingerbread	10	13.5%
4.0.3 - 4.0.4	Ice Cream Sandwich	15	11.4%
4.1.x	Jelly Bean	16	27.8%
4.2.x		17	19.7%
4.3		18	9.0%
4.4	KitKat	19	17.9%



Data collected during a 7-day period ending on July 7, 2014.  
Any versions with less than 0.1% distribution are not shown.

Source : <http://developer.android.com/about/dashboards/index.html>



## Lien divers

- [www.openhandsetalliance.com](http://www.openhandsetalliance.com) : alliance contribuant au développement de Android
- [source.android.com](http://source.android.com) : source pour le développement Android
- [www.android.com](http://www.android.com) : site officiel Android
- <http://developer.android.com> : documentation pour le développeur
- <http://android-france.fr> : news d'Android sur un site français
- [www.androlib.com](http://www.androlib.com) : les applications du Market sur un site en version française.

Outils de développement

# ANDROID

# Le kit de développement

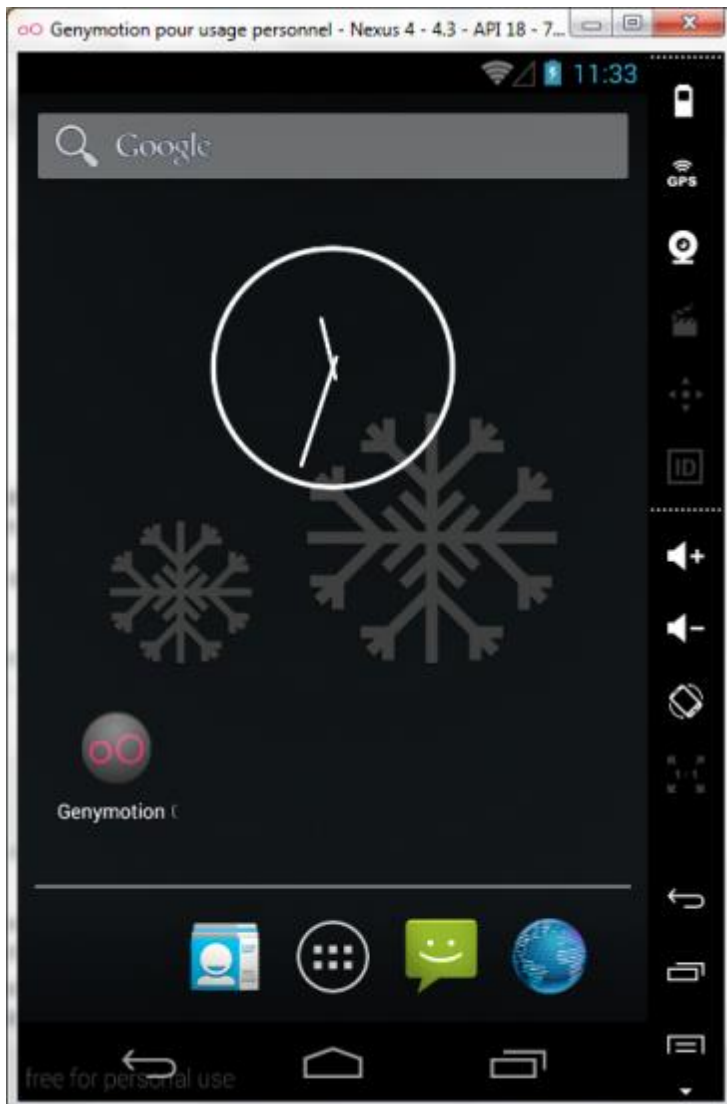
- De quoi avons-nous besoin?
  - Un Environnement de développement
    - Android Studio (basé sur IntelliJ)
    - Eclipse (avec plugin Android)
  - Le framework Android
    - Téléchargeable via le SDK Manager
  - Un Device
    - Réel de test (préférable)
    - Simulateur créé depuis le AVD Manager ou depuis Genymotion

# Android Studio

- IDE développé par Google (2013)
- Basé sur IntelliJ Community Edition
- Multiplateforme (MacOS, Windows, Linux)
- Installe automatiquement le SDK Manager et le AVD Manager
- NECESSITE d'avoir le JDK installé !!!
- <http://developer.android.com/sdk/installing/studio.html>

# Genymotion

© Matelli sarl 2013



- L'émulateur plus rapide que le device !!!
- Les Google services ne sont plus installés.
  - [http://wiki.rootzwiki.com/Google\\_Apps#Universal\\_Packages\\_2](http://wiki.rootzwiki.com/Google_Apps#Universal_Packages_2)
- La compatibilité ARM non plus.
  - Genymotion-ARM-Translation

# Dalvik Debug Monitor Server (DDMS)

© Matelli sarl 2013

- Fonctionnalité de l'IDE que l'on ouvre via une perspective
- Liste les devices avec la possibilité d'avoir des informations sur les processus créés (thread, mémoire), fichiers systèmes
- Possibilité d'interagir avec l'émulateur, en simulant des appels ou un envoi de sms, changer de position de GPS
- Contient aussi la journalisation de toutes les activités de l'émulateur : le logCat.

# Tester son application

- Impossible sur chaque type de téléphone.
- Minimum :
  - Device réel
  - 3 tailles d'écran (petite, normale (nexus 4), tablette).
  - 3 densités (mdpi, hdpi, xhdpi).
  - Avec réseau faible
  - Sur un Samsung

# Une fois sur le PlayStore

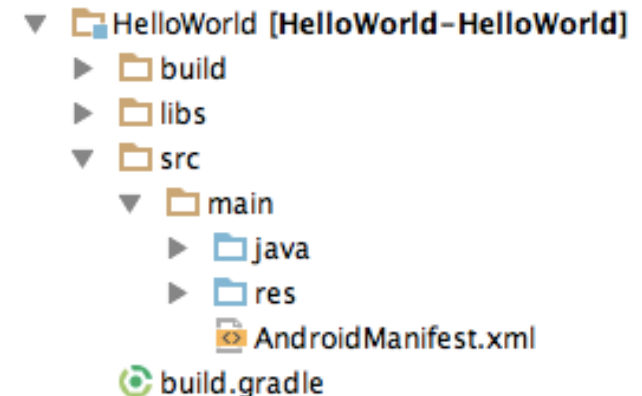
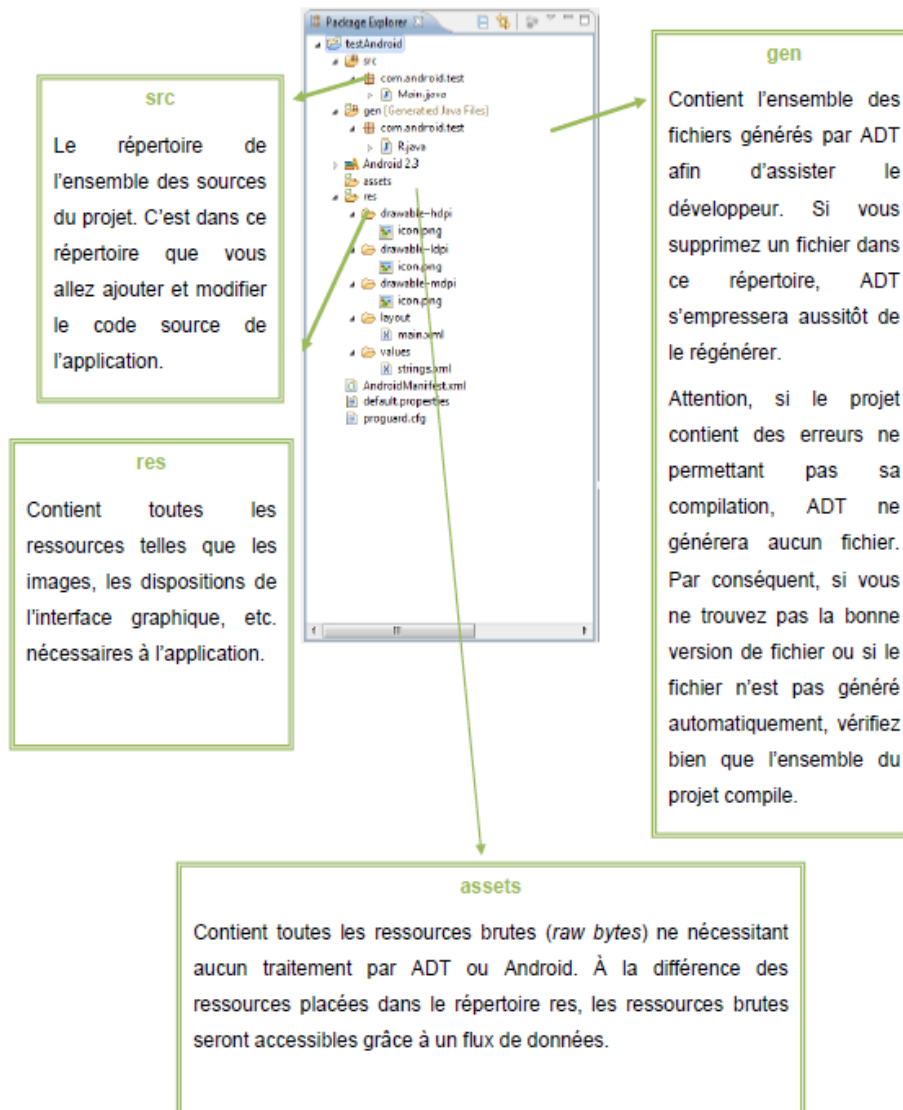
- Librairie d'Analytics d'application
  - Google Analytics
  - Capptain
  - CrashLitycs
  - Accra
- Serveur Push
  - java-apns
  - Capptain
- Ils existent des centaines de librairies :
  - <http://android-arsenal.com/free>



Architecture d'une application

# ANDROID

# Que contient mon projet ?



# Le répertoire gen

- La classe R (générée)

```
public final class R{  
    public static final class drawable {  
        public static final int icon=0x7f020000;  
    }  
    public static final class layout {  
        public static final int main=0x7f030000;  
    }  
}
```

# Le répertoire java

© Matelli sarl 2013

- Contient votre code source
  - Classes java
  - Différents packages

- Contient les ressources externes de l'application
  - values : Valeurs simples (chaines, couleurs, dimensions)
  - drawable : ressources images (conseil : utiliser le png)
  - layout : fenêtre correspondant à l'interface graphique; entièrement paramétrable en XML
  - animations : associées aux composants graphiques (translation, rotation)
- Possibilité de fournir des ressources pour différentes langues et tailles d'écran

# Le répertoire res

© Matelli sarl 2013

```
//string.xml  
<string name="today">Aujourd'hui</string>
```

```
//dimen.xml  
<dimen name="standart_height">48dp</dimen>
```

```
// Retrieve the text
```

```
Resources resources = getResources();
```






















```
String today= resources.getString(R.string.today);
```

```
//the number
```

```
Int standart_height = getResources().getDimensionPixelSize(R.dimen.standart_height);
```

# Le répertoire res

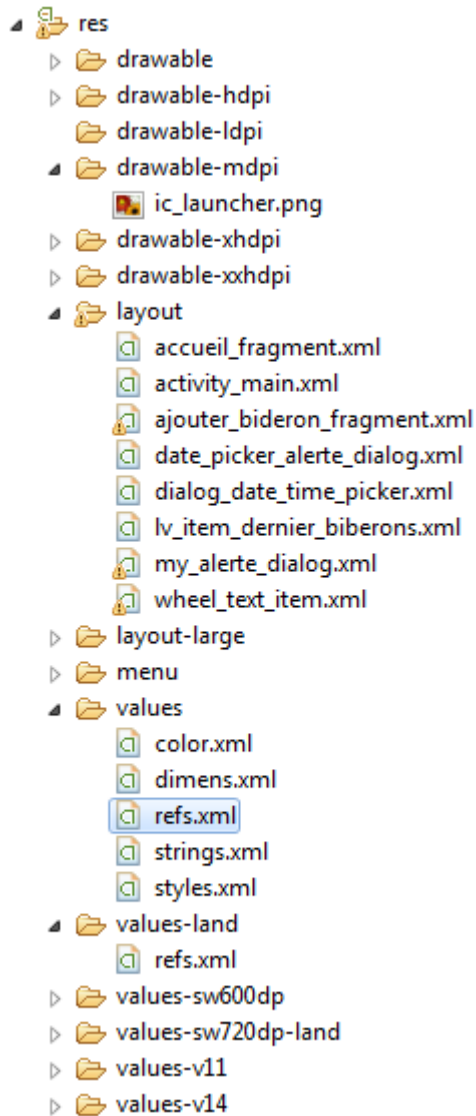
© Matelli sarl 2013

Available Qualifiers	
	Country Code
	Network Code
	Language
	Region
	Bidi Layout Direction
	Smallest Screen Width
	Screen Width
	Screen Height
	Size
	Ratio
	Orientation
	UI Mode
	Night Mode
	Density
	Touch Screen
	Keyboard
	Text Input
	Navigation State
	Navigation Method
	Dimension
	Version

- On ne peut pas tout prendre tous en charge.
- Recommandation :
  - La langue
  - 3 tailles d'écrans (medium, large, xlarge)
  - Orientation
  - Density (mdpi, ldpi, hdpi)
  - Version Android

# Le répertoire res

© Matelli sarl 2013



- Création de sous répertoire impossible.



- Les valeurs de bases sont dans les répertoires /res/values et peuvent contenir.
  - String : Vous pouvez utiliser les HTML tags <b>, <i> and <u>.
  - Colors : #RGB, #ARGB, #RRGGBB and #AARRGGBB
  - Dimensions: In pixels (px), inches (in), millimeters (mm), points (pt) , density-independent pixel (dp) or scale-independent pixel (sp)

- Fichier permettant de configurer votre application.
- Plusieurs sortes de configuration:
  - Informations générales (version, packages)
  - Informations concernant l'application : activity, attributs de l'application
  - Permission : pour autoriser l'application à avoir accès à certaines ressources (géolocalisation, internet)
  - Instrumentation : correspond aux classes de test associées.

# AndroidManifest.xml

© Matelli sarl 2013

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.project"
    android:versionCode="1"
    android:versionName="1.0" >

    <uses-sdk
        android:minSdkVersion="11"
        android:targetSdkVersion="19" />

    <application
        android:name="MyApplication"
        android:allowBackup="true"
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name"
        android:theme="@style/AppTheme" >
        <activity
            android:name="com.project.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>
```

# La classe application

- Créée à l'initialisation de l'application, elle est accessible partout et est détruite avec l'application.
- Idéal pour initialiser les variables communes à toutes l'application.
- Initialisation de nombreuses librairies. (Capptain, Google analytics...)
- Parfait pour stocker la liste des services en cours ou appel WebService.

# La classe application

```
public class MyApplication extends Application {

    private static MyApplication instance;

    public static MyApplication getInstance() {
        return instance;
    }

    @Override
    public void onCreate() {
        super.onCreate();
        //initialisation des variables static
        new Constante();
        instance = this;
    }

    /**
     * Detect si on est en simple ou double affichage
     * @return
     */
    public boolean isTwoPane() {
        return getResources().getBoolean(R.bool.twoPane);
    }
}
```

# TP

- Créez un nouveau projet
- Le lancer dans Genymotion
- Changer la langue du helloworld en fonction de la langue du téléphone.

# Android - Les composants

- Principaux composants d'une application Android:
  - Les activités (activities)
  - Les services (services)
  - Les broadcast receivers
  - Contents provider

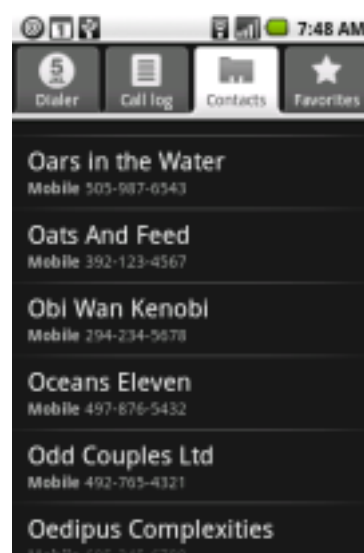
# Les composants - Les activités

© Matelli sarl 2013

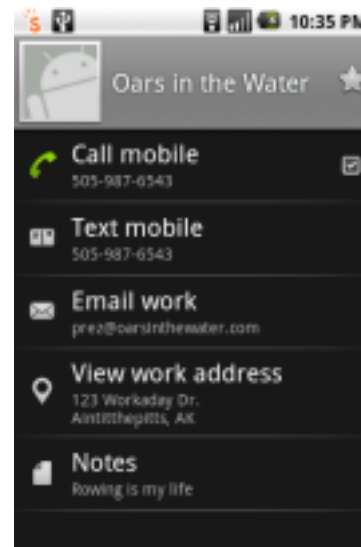
- Une interface visuelle correspond à une activité
  - Sorte de « page » ou d' « écran » dans une application
- Chaque activité est indépendante des autres
- Exemple : l'application Dialer



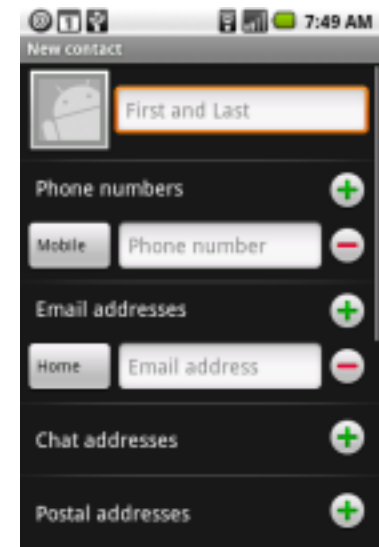
Dialer



Contacts



View Contact



New Contact



# Les composants - Les services

© Matelli sarl 2013

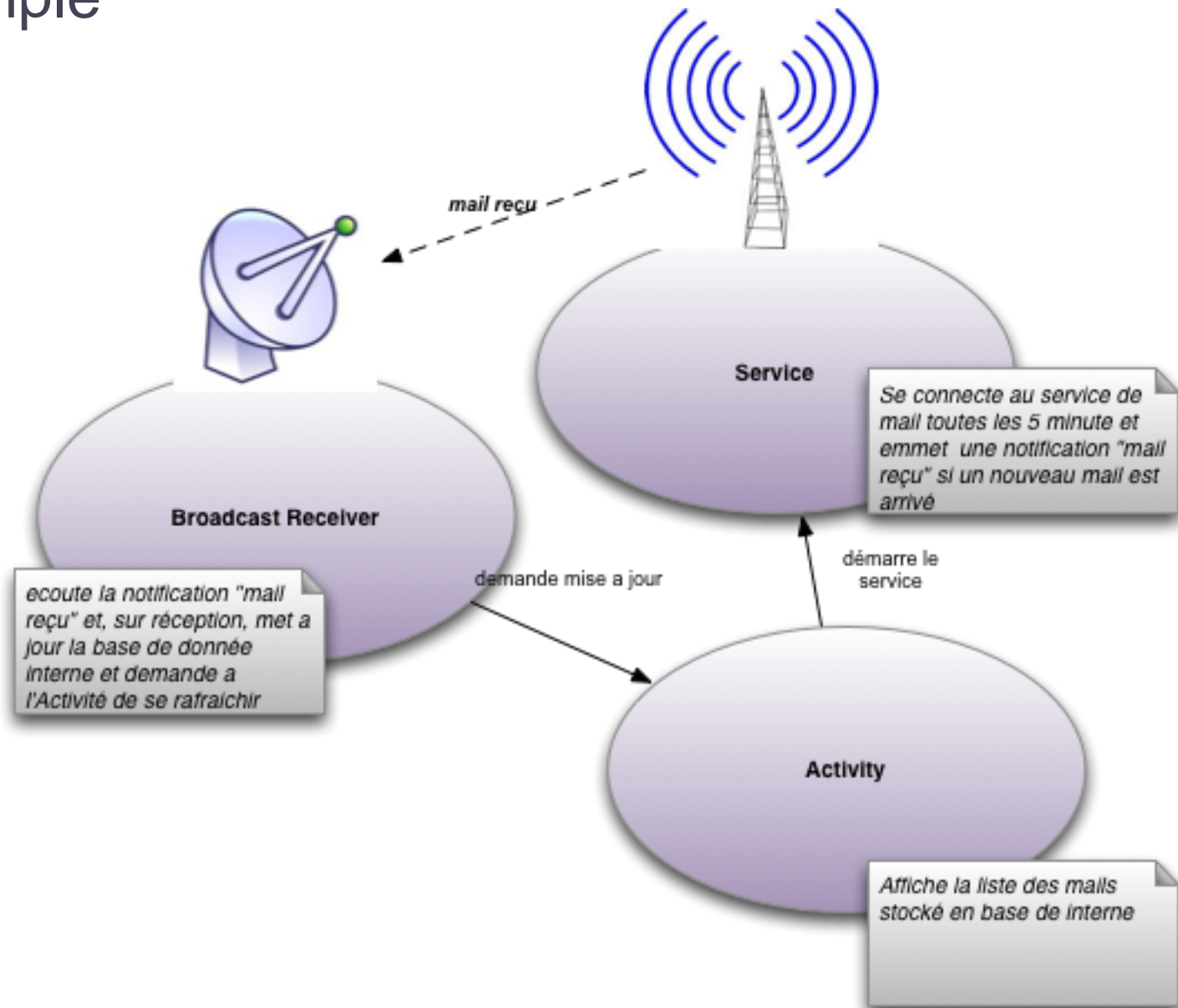
- Effectuent une tâche en arrière-plan
  - Téléchargement de données, jouer de la musique
- Ne présentent pas d'interface graphique
  - Se contrôlent via le code
- Emettent une / des notification(s) quand elles ont des informations à communiquer
  - «téléchargement terminé»
- Tourne sur le thread de l'application

# Les composants – Broadcast receiver

© Matelli sarl 2013

- «Ecoutent» une (et une seule) notification donnée
  - Notification Système
    - batterie faible, sms reçu
  - Notification Interne
    - notification émise par un service
  - Notification Google
    - émise par le Google Cloud Messaging
- Exécute un code à la réception de cette notification
  - Affichage d'une notification à l'écran, code métier, arrêt d'un service...
- Possibilité d'avoir plusieurs BR par application

# Exemple



# Les composants – Content Providers

- Rend disponible les données de l'application
- Moyen de partage de données entre applications
- Stockage des données :
  - Fichier, base de données SQLite, réseau, etc...
- Utilisation - implémentation de méthodes du Content Resolver :
  - query(), insert(), update(), delete(), getType() et onCreate()

# Bonnes pratiques

- Respecter la charte d'Android.
- Respecter les bonnes pratiques du système.
- Android est différent d'iOS.
- Respecter l'utilisateur
  - Ses données
  - Sa confidentialité
- Respecter ses ressources
  - CPU
  - Batterie
  - Mémoire
- Prévenir l'utilisateur

# Bonnes pratiques

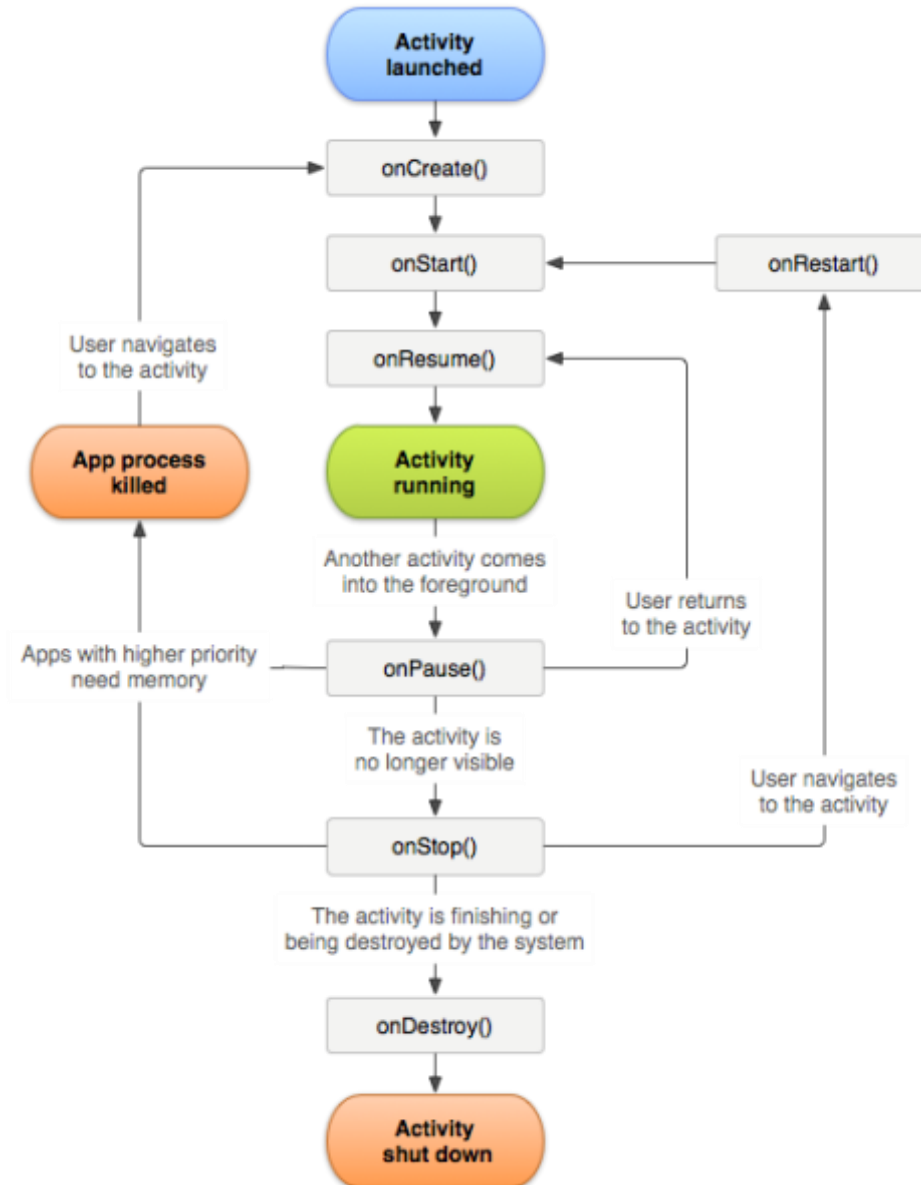
- Gestion des erreurs
  - Log
  - Messages techniques et logiques.
  - Donner la possibilité de recommencer
- Gérer les attentes augmente la fluidité
  - Un spinner ou message d'attente
  - Un préaffichage de l'écran
  - Chargement partiel des informations

# Les Activity

- Composant de base d'une application
- Représente un écran (une tâche) de l'application
- DOIT être déclarée dans le fichier manifest
- Programmation événementielle
  - Pas de main()
  - Répondre à des événements
    - Activité démarrée, click sur le bouton menu...
  - possède un cycle de vie géré par le systeme
    - reçoit des événements à différents moments de sa vie

# Les activity

© Matelli sarl 2013

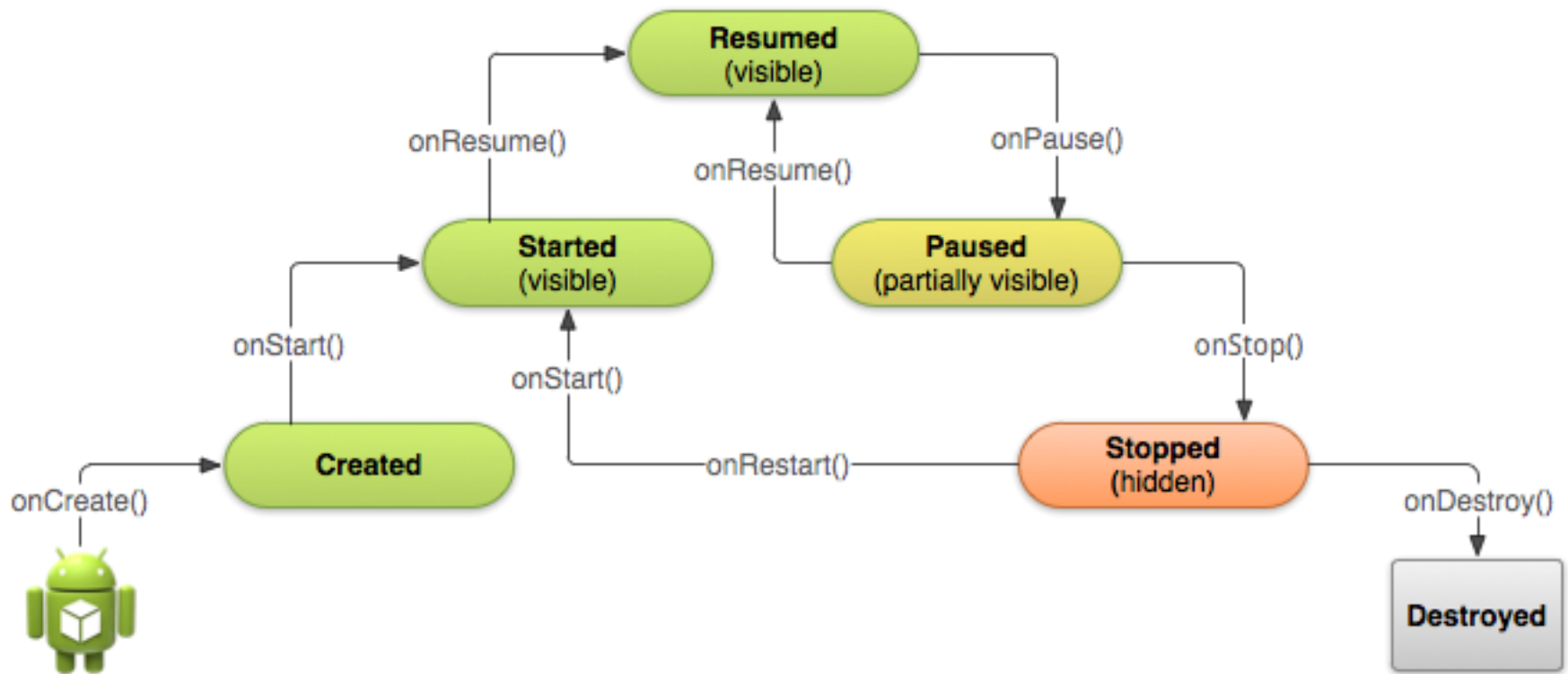


- 4 états :

- **Active** : Au 1<sup>er</sup> plan
- **En pause** : Visible mais elle n'a plus la main, une notification ou une autre activité est active.
- **Stoppée** : Existe, mais n'est plus visible. L'activité ne peut interagir avec l'utilisateur que par notification.
- **Morte** : L'activité n'est pas lancée.



# Cycle de vie des activités



# Les composants - Les activités

© Matelli sarl 2013

```
public class MainActivity extends Activity {  
  
    private TextView tv_heure;  
  
    @Override  
    protected void onCreate(final Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        setContentView(R.layout.activity_main);  
  
        tv_heure = (TextView) findViewById(R.id.tv_heure);  
    }  
  
    @Override  
    protected void onPause() {  
        super.onPause();  
    }  
  
    @Override  
    protected void onResume() {  
        super.onResume();  
        tv_heure.setText("Heure : " + new Date().getTime());  
    }  
  
    @Override  
    protected void onDestroy() {  
        // TODO Auto-generated method stub  
        super.onDestroy();  
    }  
}
```

# Cycle de vie

- **onCreate** initialise l'activité (création d'un objet à partir du layout xml, affectation des variables, chargement des données sur les composants)
- **onDestroy** est appelée à la destruction et se charge de fermer toutes les connexions
- **onStop** aura la charge de stopper les threads, animations, et plus généralement les process qui vont agir sur l'interface
- Utilisation de **onStart** pour lancer/relancer ces process
- **onPause/Resume** doit être léger de façon à ne pas ralentir la machine lors du redémarrage (on peut s'en servir pour s'abonner à des broadcasts)

# Lancer une activity

```
@Override
public void onClick(final View v) {

    final Intent intent = new Intent(this, SecondActivity.class);
    intent.putExtra(Constants.EXTRA_SECOND_ACTIVITY_MSG, "Hello from
MainActivity");
    startActivity(intent);
}
```

# Afficher des messages en console

- Plusieurs niveaux de
  - Verbose
  - Debug
  - Information
  - Warning
  - Error
- Utilisation des méthodes statiques de la classe Log( respectivement les fonctions v(),d(),i(),w(), et e())
- Chaque message est associée a un tag facilitant le filtre des messages dans la console

# Exemple

- Bonne Pratique :
  - Définir dans le fichier de constante les tags pour être sûr d'avoir toujours le même dans la console.
  - Définir une variable permettant de savoir si on est en prod.

```
protected void onResume() {  
    //Log.e("tag", "text");  
    if (Constante.LOG_DEV_MODE) {  
        Log.v(LogUtils.SET_FRONT_LOGTAG, "activity : " + activity);  
    }  
}
```

# TP

- Créez un nouveau projet
- Ajouter une seconde Activity
- Créer un bouton sur la 1<sup>er</sup> qui redirige sur la seconde
- Surchargez chacun des callback correspondant au changement d'état dans le cycle de vie de l'activity.
- Pour chacun de ces événements, écrire un message en console.
- «Jouer» avec le simulateur pour comprendre à quel moment ces messages sont reçus
- Même chose en ajoutant un Thread.sleep dans chacun des états.

IHM



# Android

- Construire une interface graphique
- Pas de traitement lourd sur l'UIThread. (Service, AsyncTask...)
- Les modifications d'IHM uniquement sur l'UIThread

```
//On peut afficher les info  
runOnUiThread(new Runnable() {  
  
    @Override  
    public void run() {  
        //traitement IHM  
    }  
});
```

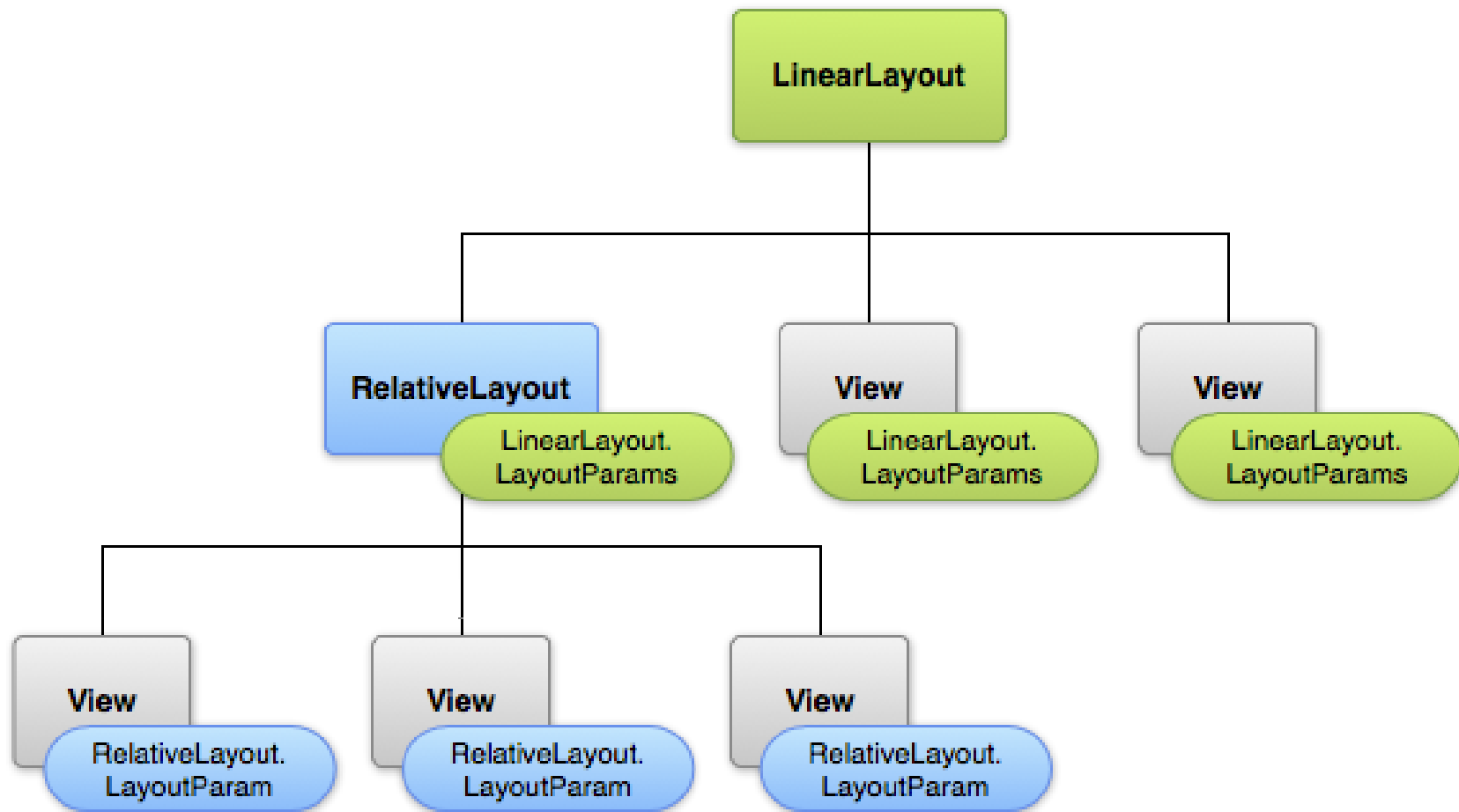
# Construire son interface graphique

- 2 possibilités
  - En code
  - En XML
- Privilégier autant que possible le XML
  - Séparation comportement et visuel (MVC)
  - Outil de rendu graphique en XML
- Utilisation du code pour des changements dynamiques
- Hauteur et largeur minimum des composants cliquables de 40dp

# Layouts

- Les layouts sont des composants permettant de positionner les différents composants graphiques.
- Ce sont des conteneurs d'éléments visuels (ils peuvent contenir d'autre vues et même d'autres layouts) représentés sous forme de fichier xml ou créés directement dans le code.
- Plusieurs types de layouts :
  - Linear
  - Relatif
  - Table
  - ...

# Schéma global



# Les layouts et le XML

© Matelli sarl 2013

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
    // Attributs du layout
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical" >

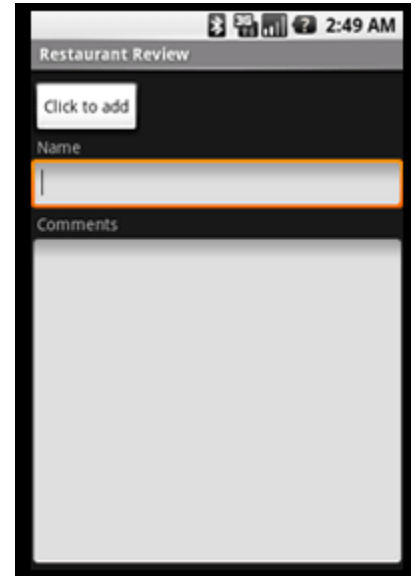
    <!-- Contenu du layout -->

</LinearLayout>
```

# Linear layout

© Matelli sarl 2013

- Layout le plus utilisé
- Container permettant de placer les éléments en ligne
- Constructeur :
  - `LinearLayout(context,[object arg])`
- Méthodes :
  - `setOrientation(LinearLayout.VERTICAL)` : changer l'orientation des lignes du container
  - `setGravity(Gravity.RIGHT)` : place les éléments selon un côté.
  - `setPadding(left,top,right,bottom)` : marge des composants
  - `addView(View)` : ajouter des composants graphiques
  - `setBackgroundDrawable(BitmapDrawable)`: definit un arrière plan



# Attributs du LinearLayout

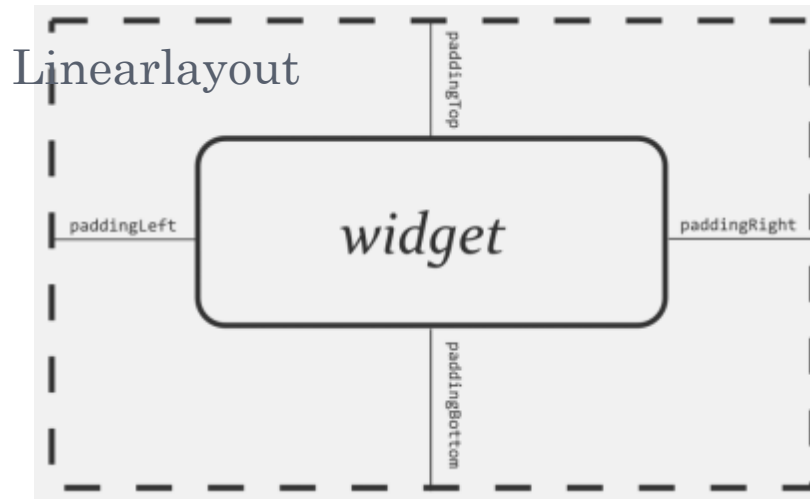
© Matelli sarl 2013

- Orientation : indique si le layout ajoute les composants par ligne ou par colonne
  - `android:orientation`
- Width et height : 2 propriétés
  - « `wrap-content` » et « `fill_parent` » (`match_parent`)
- Gravity : par défaut, l'alignement se fait de haut en bas
- Weight : définit l'espace que prendra la vue associée

# Linear layout

© Matelli sarl 2013

- Padding : spécifie l'espace entre le composant et son wrapper





- Positionner les composants de façon relative entre eux (leurs positions dépendent d'eux-mêmes)
- Position relative à un container
  - `android:layout alignParentTop`: le composant est aligné par rapport au début du container
  - `android:layout alignParentBottom`: le composant est aligné par rapport à la fin du container
  - `android:layout alignParentLeft`: le composant est aligné par rapport à la partie gauche du container
  - `android:layout alignParentRight`: le composant est aligné par rapport à la partie droite du container

# Relative layout

© Matelli sarl 2013

- `android:layout centerHorizontal`: le composant est positionné au centre horizontal
- `android:layout centerVertical`: le composant est positionné au centre vertical
- `android:layout centerInParent` : les deux
- Ces propriétés possèdent comme valeur vrai ou faux
- Propriétés utilisées entre composants

# Relative layout

© Matelli sarl 2013

- `android:layout above` : indique que cet élément sera placé au dessus du composant référencé dans les propriétés
- `android:layout below` : indique que cet élément sera placé au dessous du composant référencé dans les propriétés
- `android:layout toLeftOf` : indique que cet élément sera placé à gauche du composant référencé dans les propriétés
- `android:layout toRightOf` : indique que cet élément sera placé à droite du composant référencé dans les propriétés

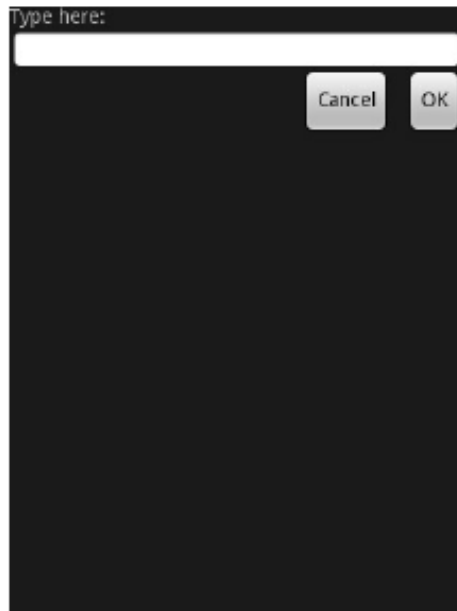
# Exemple

© Matelli sarl 2013

Type here:

Cancel OK

# exemple



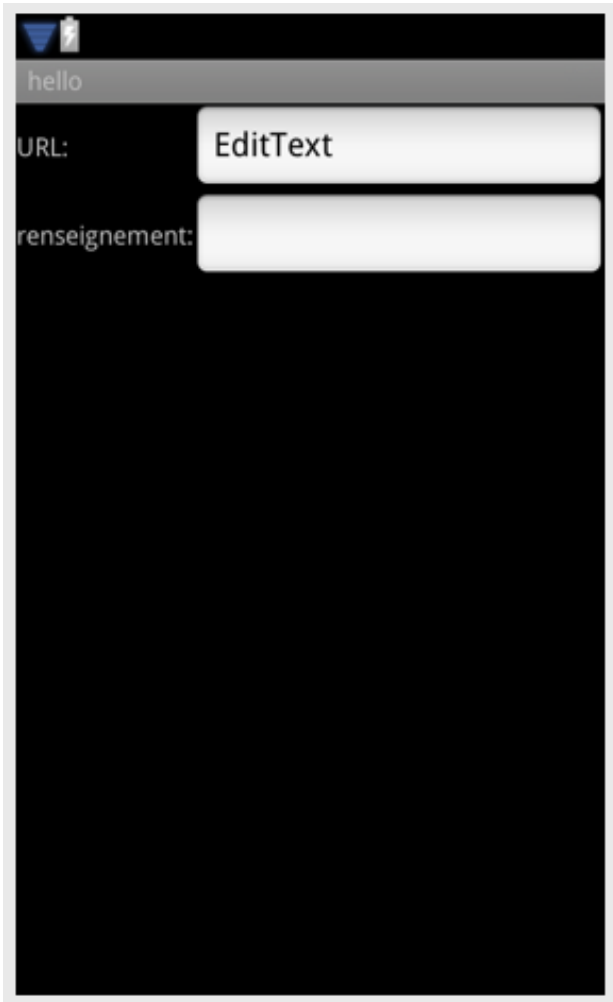
```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent">
    <TextView
        android:id="@+id/label"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="Type here:"/>
    <EditText
        android:id="@+id/entry"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:background="@android:drawable/editbox_background"
        android:layout_below="@id/label"/>
    <Button
        android:id="@+id/ok"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_below="@id/entry"
        android:layout_alignParentRight="true"
        android:layout_marginLeft="10dip"
        android:text="OK" />
    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_toLeftOf="@id/ok"
        android:layout_alignTop="@id/ok"
        android:text="Cancel" />
</RelativeLayout>
```

# Tablelayout

© Matelli sarl 2013

- Le fonctionnement du `tableLayout` ressemble beaucoup à celle des tables en HTML
- Les éléments enfants sont des `tableRow` où on ajoute nos composants
- On les utilise pour aligner les formulaires
- Pour placer les composants de manière idéale, nous pouvons utiliser le concept de `span` sur les composants
- `android:layout_span="2"`
- `android:layout_column="2"`

# Example



```

<TableLayout
xmlns:android="http://schemas.android.com/apk/res/android"
android:stretchColumns="1"
android:layout_height="fill_parent"
android:layout_width="fill_parent">
<TableRow android:id="@+id/ligne1">
<TextView
    android:text="URL:"
    android:id="@+id/textView1"></TextView>
<EditText android:layout_height="wrap_content"
    android:text="EditText"
    android:layout_width="wrap_content"
    android:id="@+id/EditText01"></EditText>
</TableRow>
<TableRow android:id="@+id/ligne2">
<TextView android:text="renseignement:"/>
<EditText android:id="@+id/entry"
    android:layout_height="wrap_content"
    android:layout_width="wrap_content"/>
</TableRow>
</TableLayout>

```

# Les composants graphiques

© Matelli sarl 2013

- On appelle composant graphique (vues), les composants qui constituent votre fenêtre.
- Des bases Android vous en fournissent de multiples (éditeur, bouton, label...)



- Attention aux nombres de layout utilisés.
- Si la support library est utilisée au moins une fois, alors elle doit l'être partout. Si version min < IceCream.
- Utiliser les fichiers de config (dimen) pour définir les tailles des composants en fonction du device.
- Un galaxy S2 et un nexus 10 n'ont pas la même taille d'écran ni de densité mais ils lanceront la même application, mais pas les mêmes fichiers de value.

- Champ texte modifiable par l'utilisateur
- En code :
  - new EditText(Context context)
  - utilisation de setText() et getText() pour gérer le contenu

```
<EditText  
    android:layout_width="fill_parent"  
    android:layout_height="wrap_content"/>
```

# TextView

© Matelli sarl 2013

- Champ texte non modifiable par l'utilisateur
- En code :
  - `new TextView(Context context)`
  - utilisation de `setText()` et `getText()` pour gérer le contenu

```
<TextView  
    android:layout_width="fill_parent"  
    android:layout_height="wrap_content"/>
```

# Button

© Matelli sarl 2013

- Un simple bouton à appuyer.
  - `setText(var)` : ajouter un texte.

```
<Button  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"/>
```

# Checkbox

- Sélectionne des informations

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

- Boutons à choix exclusifs à mettre dans des radio group

```
<RadioGroup
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:orientation="vertical">

    <RadioButton android:id="@+id/radio_red"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Red" />

    <RadioButton android:id="@+id/radio_blue"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Blue" />

</RadioGroup>
```

# ImageView

© Matelli sarl 2013

- Affiche une image simple
  - [setImageResource\(int\)](#)

```
<ImageView  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:src="@drawable/icon"/>
```

# TP : construction d'une vue

Réaliser et afficher le layout suivant

The screenshot shows an Android application interface with a dark theme. At the top, the status bar displays various icons including a warning triangle, a USB icon, a 3G signal icon, and a battery icon, along with the time 3:27. Below the status bar, the app's title bar reads "mes musiques". The main content area has a header "Ajout/Edit musique". Below this header, there is a text input field labeled "titre" which is highlighted with a red border. Underneath the input field, there are two radio buttons: the first is labeled "j'aime" and is currently selected (indicated by a green dot), and the second is labeled "je n'aime pas" and is unselected. Below the radio buttons, there is a label "Categories" followed by a text input field containing the placeholder text "une categorie". At the bottom of the screen, there is a large white rectangular area labeled "description". At the very bottom, there are two buttons: "ok" and "annuler".



# Du XML au code

- Les composants sont déclarés dans l'interface graphique. Comment :
  - Mettre du texte dans le label?
  - Etre informé du click sur un bouton?
- Solution:
  - Récupérer les vues qui nous intéressent sous forme de variables d'instance de l'activity
    - Appeler des méthodes
      - afficher un text dans une textView
  - Ajouter des « écouteurs d'événements » sur ces variables
    - Ecouter le clic sur un bouton

# Du XML au code

- Dans le XML
  - On ajoute un identifiant au composant

```
<Button  
android:id="@+id/Button01"  
android:layout_width="wrap_content"  
android:layout_height="wrap_content"/>
```

- Dans le code de l'activity (au moment du onCreate)
  - On récupère le composant via son identifiant

```
Button b= (Button)findViewById(R.id.Button01) ;
```

# Gestion des événements

- Sur une interface graphique, chaque action de la part de l'utilisateur va être perçue comme un événement (click, effleurement, saisie...)
- On associe un traitement sur événement au niveau d'un composant grâce à un gestionnaire d'événements.
- C'est concrètement une classe possédant une ou plusieurs méthodes imposées à implémenter.
- Ces méthodes vont correspondre aux traitements à effectuer lorsque l'utilisateur interagit avec l'application

# Gestion des événements

- Un écouteur d'événements est appelé en java un « Listener ».
- Certains composants peuvent réagir à des événements
  - Boutons réagissent au clic
- Ces composants proposent une API pour leur ajouter un écouteur de l'événement X
  - `public void setOnXListener(OnXListener listener)`

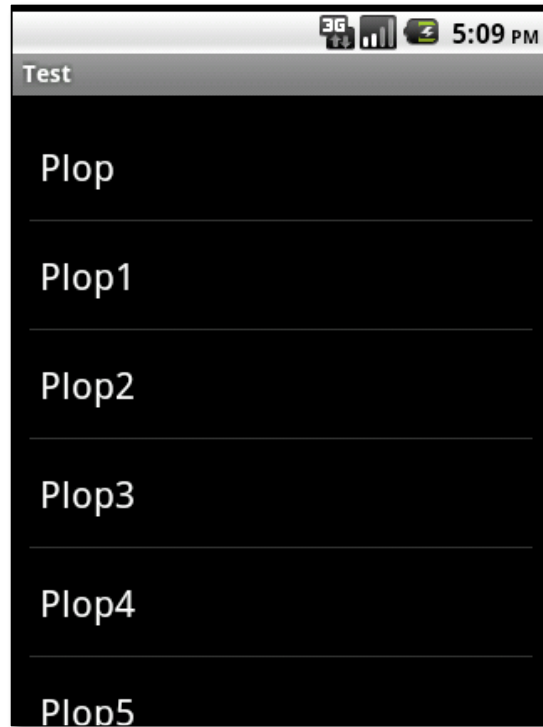
```
Button myButton = ...;
myButton.setOnClickListener(new View.OnClickListener() {
    public void onClick(View view) {
        // Exécuter quand on click sur myButton
    }
});
```

# Quelques APIs

- Button
  - `setOnClickListener(...)`
  - Appelé lors du clic sur le bouton
- EditText
  - `setKeyListener(...)`
  - Appelé à chaque touche du clavier appuyée
- CheckBox / RadioButton
  - `setOnClickListener(...)`

# Android

- Afficher les données sous forme de liste



# ListView

- Widget pour afficher les données sous forme de liste
- Utilise un «adapter» pour savoir quoi afficher
  - **ArrayAdapter**
    - Construit simplement avec une ArrayList
    - Affiche le toString() de chaque objet
  - **BaseAdapter (90% du temps)**
    - Plus complexe
    - Permet un design plus évolué
      - design d'une cellule en XML
- Recycler les cellules, pour le bien de votre device.

# ListView

- Possibilité d'ajouter un `OnItemClickListener`
- Possibilité de faire apparaître un menu contextuel lors de l'appui long sur un item



# Adapter

- Différentes sous-classes
  - **ArrayAdapter**
    - Constructeur à 3 paramètres
      - context: l'activity «propriétaire» de la liste
      - ressourceId : un design de cellule
      - une liste d'objet
    - Affiche le toString() de chaque objet
  - **Base Adapter**
    - Permet d'afficher un xml par cellule.
    - Permet des designs plus complexes (image, etc...)

# Exemple : ArrayAdapter

```
// Récupération de la listview
ListView listView = ...;

// Récupération de la liste d'objet à afficher
ArrayList<Object> items = ...;

// Création et affectation de l'ArrayAdapter
ArrayAdapter<Object> adapter = new
ArrayAdapter<Object>(this, R.layout.simple_list_item_1,
items);

listView.setAdapter(adapter);
```

# Ecouter le clic sur un élément du tableau

```
OnItemClickListener listener = new.OnItemClickListener()
{
    @Override
    public void onItemClick(AdapterView<?> parent, View view,
int position, long id) {
        // Do something here
    }
}

listView.setOnItemClickListener(listener);
```

# Associer un menu contextuel aux items

```
OnCreateContextMenuListener listener = new ...()
{
    @Override
    public void onCreateContextMenu(ContextMenu menu,
    View v, ContextMenuInfo menuInfo)
    {
        menu.add(0, 1000, 0, "Edit");
        menu.add(0, 1001, 0, "Delete");
    }
}
listView.setOnCreateContextMenuListener(listener);
```

# Menu contextuel

Nous récupérons les choix dans la fonction `onContextItemSelected` de notre activité :

```
public boolean onContextItemSelected(MenuItem item) {  
    AdapterView.AdapterContextMenuInfo info =  
    (AdapterView.AdapterContextMenuInfo)item.getMenuInfo();  
    switch (item.getItemId()){  
        //la position de l'item se recupere avec info.position  
        case 1000:  
            ...  
            break;  
            //delete  
        case 1001:  
            ...  
            break;  
        default:break;  
    }  
}
```

# Exemple : BaseAdapter

```
public class EleveAdapter extends BaseAdapter {  
    private final LayoutInflater mInflater;  
    private final List<Eleve> eleveList;  
  
    public EleveAdapter(final Context context, final List<Eleve> eleveList) {  
        mInflater = (LayoutInflater)  
            context.getSystemService(Context.LAYOUT_INFLATER_SERVICE);  
        this.eleveList = eleveList;  
    }  
  
    @Override  
    public int getCount() {  
        return eleveList.size();  
    }  
  
    @Override  
    public Eleve getItem(final int position) {  
        return eleveList.get(position);  
    }  
  
    @Override  
    public long getItemId(final int position) {  
        return position;  
    }  
}
```

# Exemple : BaseAdapter

- Le View Holder represente les composants à modifier entre chaque cellule.
- Le but est de reutiliser une cellule existante et de ne faire que modifier les valeurs.

```
//-----  
// View Holder  
//-----  
public static class ViewHolder {  
    public TextView ec_tv_nom, ec_tv_prenom;  
    public ImageView ec_iv;  
    public Eleve eleve;  
}
```

# Exemple : BaseAdapter

@Override

```
public View getView(final int position, final View convertView, final ViewGroup parent) {  
    View rowView = convertView;  
  
    //-----  
    // inflate  
    //-----  
    final ViewHolder viewHolder;  
    if (rowView == null) {  
        //création  
        rowView = mInflater.inflate(R.layout.eleve_cellule, null);  
  
        viewHolder = new ViewHolder();  
        viewHolder.ec_tv_nom = (TextView) rowView.findViewById(R.id.ec_tv_nom);  
        viewHolder.ec_tv_prenom = (TextView) rowView.findViewById(R.id.ec_tv_prenom);  
        viewHolder.ec_iv = (ImageView) rowView.findViewById(R.id.ec_iv);  
  
        rowView.setTag(viewHolder);  
    }  
    else {  
        //recyclage  
        viewHolder = (ViewHolder) rowView.getTag();  
    }  
}
```



# Exemple : BaseAdapter

```
//on remplit avec l'objet voulu
final Eleve eleve = eleveList.get(position);

viewHolder.ec_tv_nom.setText(eleve.getNom());
viewHolder.ec_tv_prenom.setText(eleve.getPrenom());

return rowView;
}
```

# Exemple : BaseAdapter

**//Création**

```
private ListView lv;  
private List<Eleve> eleveList;  
private EleveAdapter eleveAdapter;
```

```
eleveList = new ArrayList<Eleve>();  
eleveAdapter = new EleveAdapter(this, eleveList);  
lv = (ListView) findViewById(R.id.lv);  
lv.setAdapter(eleveAdapter);
```

**//Mise à jour**

```
//on vide la liste  
eleveList.clear();  
//on la remplit, attention à ne pas casser le pointeur.  
eleveList.addAll(getEleves());
```

```
//on previent la liste que les données ont changés  
eleveAdapter.notifyDataSetChanged();
```

# XML to Code

- Un site permettant de générer le code à partir du xml valable aussi pour les listView
  - <https://www.buzzingandroid.com/tools/android-layout-finder/>

## TP - ListView

- Ajouter une ListView d'élève sur la seconde activité.
  - Une cellule sera défini par une image, prénom et nom
- Pour les filles le nom sera en rose.
- Mettre un bouton qui ajoute 10 élèves à la liste.
- Tester avec + de 100 élèves.
- Passer le device en mode paysage, qu'est ce qu'il se passe?

# Gestion dynamique des images

- Librairie Picasso
  - [square.github.io/picasso/](https://square.github.io/picasso/)
- Chargement en arrière plan des images à partir d'une url
- Gestion d'un cache mémoire et disque.

# Android

- Communication entre 2 activities

# Les activités et leurs interactions

© Matelli sarl 2013

- Android offre un système de communication très ingénieux permettant de faire passer l'information entre Activity ou plus généralement entre composants applicatifs
- Ce système est connu sous le nom d'Intent
- Il est possible de passer des informations entre activités grâce aux intents

# Utilisation des Intents

- Nous mettons directement le nom de notre activité sur l'intent
- Nous l'utilisons pour lancer nos propres activités
  - `new Intent(this, HelpActivity.class);`
- Une fois l'intent créé, nous le lançons dans notre activité
  - Ces activités seront considérées comme sous-activités
- 2 méthodes existent :
  - `startActivity(Intent i)`
  - `startActivityForResult(Intent i, int req)`



# Exemple

```
Intent t = new Intent(this,Edit.class);  
  
t.putExtra("musicUpdate", 10);  
startActivity(t);
```

- Si on ajoute l'instruction finish() cela détruit l'activité courrante.

# Utilisation des Intents

- Possibilité d'ajouter une information de l'activité parent aux sous-activités
  - `intent.putExtra("clé", valeur);`
- Il est donc possible de transmettre tout type primitif, même des objets "serializable".
- L'activité enfant le récupère grâce aux méthodes fournies par son intent.
  - `this getIntent().getExtras().getString("maonnee");`

# Intent utilisation

## Activité 2

```
public void onCreate(Bundle savedInstanceState) {  
    super.onCreate(savedInstanceState);  
    setContentView(R.layout.main);  
    if(getIntent().getExtras()!=null)  
        int musicUpdate = getIntent().getExtras().getInt("musicUpdate");};
```

# Utilisation des Intents

- Nous pouvons aussi récupérer des informations de l'enfant pour son parent.
- Utilisation de la méthode `setResult(int res, Intent i)` pour envoyer un résultat de l'enfant au parent.
  - 1er paramètre est un code de retour (int id)
  - 2eme paramètre est un intent qui contient des informations
- Sur le parent, implémentation de la fonction `onActivityResult(int requestCode, int resultCode, Intent data);`

# Exemple

## Activité 1

```
Intent t = new Intent(this, Edit.class);
```

```
t.putExtra("musicUpdate", 10);  
startActivityForResult(t, 110);
```

# Exemple

- Ici startActivityForResult contient 2 paramètres :
  - Un intent identique startActivity
  - Un request code nous permettant de tracer l'activité appelée

# Exemple

## Activité 2 envoie des informations et quitte

```
int m = 30;  
  
Intent t = new Intent();  
  
t.putExtra("maMusic",m);  
  
setResult(50, t);  
  
Edit.this.finish();
```

# Exemple

- Avec son setResult, activity 2 :
  - envoie un resultCode en 1er paramètre, afin de notifier l'endroit exact où l'application a quitté,
  - puis un intent contenant des informations destinées à l'activity 1



# Exemple

Activité1 reçoit les informations

```
@Override
protected void onActivityResult(int requestCode, int resultCode, Intent data) {
    super.onActivityResult(requestCode, resultCode, data);
    if (requestCode == 110){
        if (resultCode==50){
            int m= data.getExtras().getInt("maMusic");
        }
    }
}
```

# Serializable et Parcelable

- En implementant serializable ou Parcelable, l'objet peut être transmit par intent.
- Parcelable est plus optimisé que serialisable mais nécessite de le remplir à la main.
  - Heureusement <http://www.parcelabler.com/> le fait pour VOUS
- Parcelable permet aussi de transmettre une liste d'objet récupérable
  - `List<Eleve> eleves =  
intent.getParcelableArrayListExtra(cle);`

# Les intents implicites

- En utilisant un Intent implicite, nous laissons Android décider de la meilleure application à lancer.
- Nous appelons une action au lieu du nom de la classe dans l'intent
- L'action est tout simplement une chaîne de caractères.
- Cette action est souvent associée à une URI pour apporter plus de précision sur le type d'application à lancer.

# Tableau des principaux intents

Action	Uri	Description
Intent.ACTION_VIEW	geo:latitude,longitude	ouvre google maps en cherchant les lat/long
Intent.ACTION_VIEW	geo:0,0?q=street+address	ouvre google maps en cherchant l'adresse
Intent.ACTION_CALL	tel:phone_number	ouvre l'appli tel pour faire un appel
Intent.ACTION_DIAL	tel:phone_number	ouvre l'appli tel directement avec le no composé
Intent.ACTION_VIEW	http://web_address	ouvre un browser
Intent.ACTION_WEB_SEARCH	un texte	ouvre un browser avec une recherche google

# Exemple d'utilisation de l'intent implicite

// Call

```
final String callUrl = "tel:" + friend.getPhoneNumber();  
startActivity(new Intent(Intent.ACTION_DIAL, Uri.parse(callUrl)));
```

// Sms

```
final String smsUrl = "sms:" + friend.getPhoneNumber();  
startActivity(new Intent(Intent.ACTION_VIEW, Uri.parse(smsUrl)));
```

# TP – Extra

- Reprendre le TP sur la ListView, et faire en sorte de sauvegarder la liste lors de la rotation.
  - Rendre les élèves Parcelable  
<http://www.parcelabler.com/>
  - Sauvegarder les éléments de l'activité

```
@Override
protected void onSaveInstanceState(final Bundle outState) {
    super.onSaveInstanceState(outState);
    outState.putParcelableArrayList(Constants.EXTRA_LIST_ELEVE, eleveList);
}
```

- Charger les éléments en mémoire.

```
if (savedInstanceState != null) {
    eleveList =
        savedInstanceState.getParcelableArrayList(Constants.EXTRA_LIST_ELEVE);
}
```

S'adapter à l'orientation et à la plateforme

# **AFFICHAGE DYNAMIQUE**

# Problématique

- Différentes tailles d'écran
  - Tablette vs telephone
- Différentes orientation
  - Landscape vs paysage
- Comment adapter l'interface graphique de son application tout en unifiant notre code?



# Exemple sur l'appli music

- Sur téléphone
  - Un écran présentant la liste des musique
  - Sur sélection d'une musique, affichage d'un autre écran présentant le détail de la musique
- Sur tablette
  - Un écran présentant la liste d'un coté et le détail de la musique sélectionnée de l'autre

# Problème

- La réflexion avec des activities nous obligerait à créer 3 activities (`listeActivity`, `detailActivity` et `combinedActivity`) et à dupliquer beaucoup de code
- Impossible de charger une Activity spécifique en fonction du device!!!!

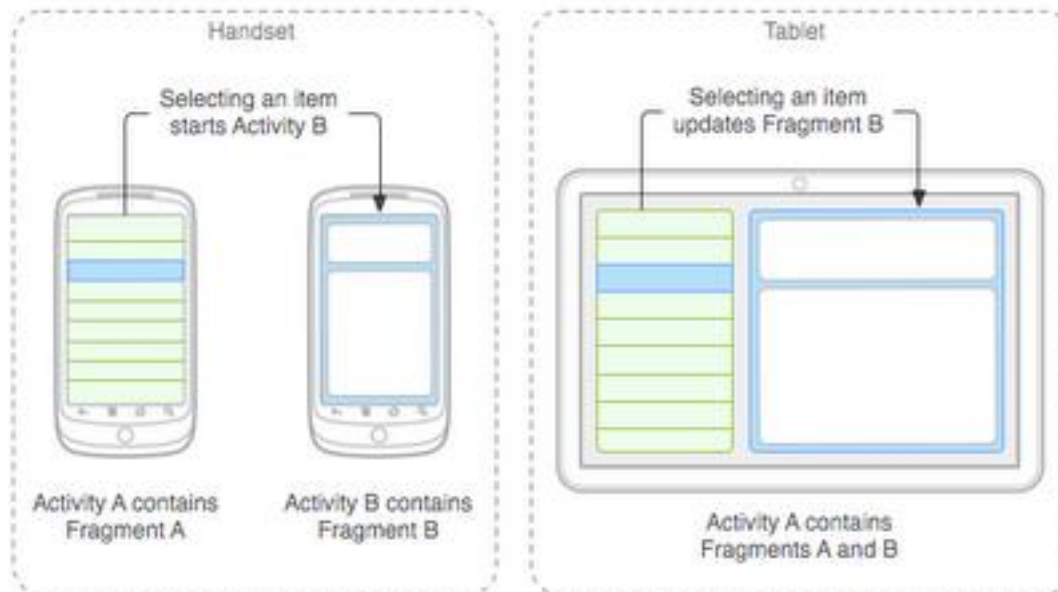
# Solution

- Dissocier la notion d'écran (activity) de la notion de fonctionnalité.
  - L'application contient 1 écran (1 activity)
  - L'application à 2 fonctionnalités (2 fragments)
    - 1 fragment MusicList
    - 1 fragment MusicDetail

# Application

- Sur téléphone
  - MainActivity présente MusicListFragment
  - Sur sélection d'une musique, MainActivity présente MusicEditFragment avec la musique sélectionnée
- Sur tablette
  - MainActivity présente un LinearLayout horizontal comprenant un MusicListFragment et un MusicDetailFragment

# Exemple



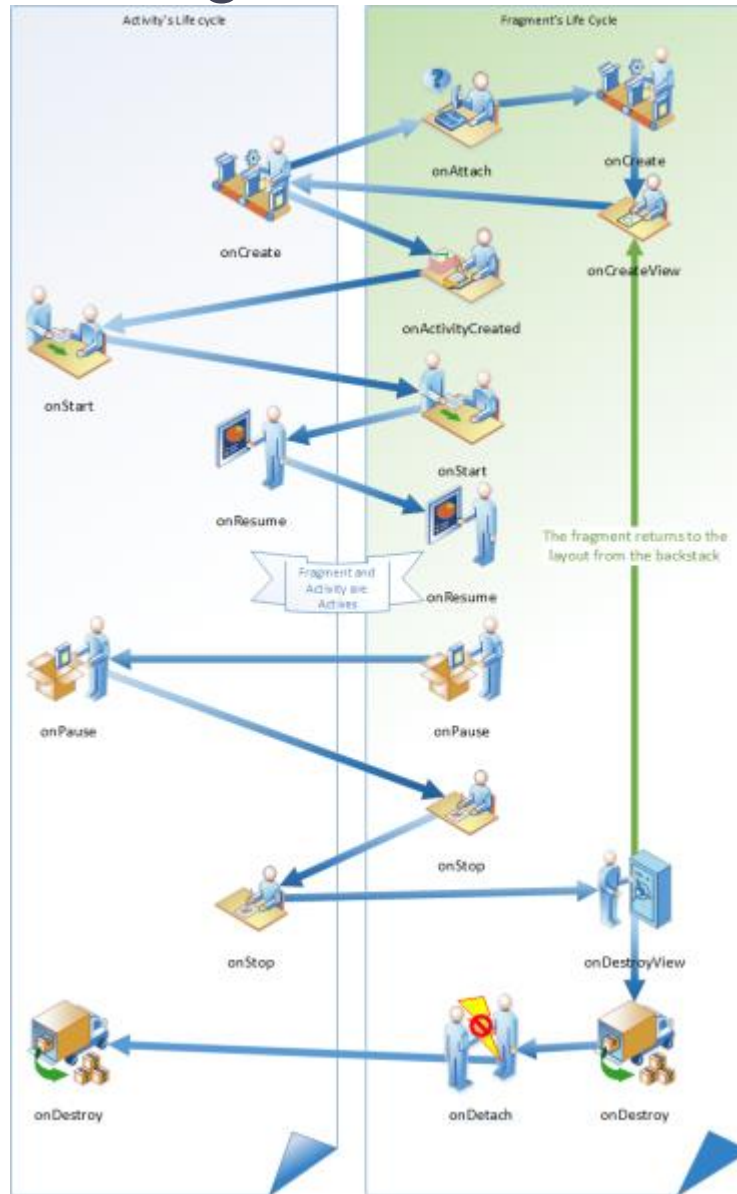
# Fragment

- Un fragment est un composant indépendant ayant son propre fonctionnement interne.
- Un fragment a sa propre vue et son propre model (MVC)
  - Un fragment encapsule une partie de l'interface de l'application le comportement qui lui est associé
- Un fragment doit être géré par une activity
- Une activity peut gérer plusieurs fragments
  - L'activity gère la navigation / communication

# Creation d'un Fragment

- Créer une classe héritant de Fragment
- Cycle de vie
  - `onCreateView(Inflater, ViewGroup, Bundle)`
    - Permet de créer une hiérarchie de vue
    - Charger un XML grâce à l'Inflater
    - Définir les different Listener
  - `onAttach(Activity), onDetach, onDestroy()...`

# Creation d'un Fragment





# onCreateView()

```
import android.os.Bundle;
import android.support.v4.app.Fragment;
import android.view.LayoutInflater;
import android.view.ViewGroup;

public class ArticleFragment extends Fragment {
    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
        Bundle savedInstanceState) {
        // Inflate the layout for this fragment
        return inflater.inflate(R.layout.article_view, container, false);
    }
}
```

# Layout conditionnel

- Une activity est capable de charger un Layout
  - `setContentView(R.layout.content_frame);`
- On peut fournir plusieurs versions du layout
  - `res/layout/content_frame.xml`
    - Layout pour téléphone
  - `Res/layout-xlarge/content_frame.xml`
    - Layout spécifique tablette
- Ressource appropriée chargée automatiquement

# Layout/content\_frame

```
<?xml version="1.0" encoding="utf-8"?>  
  
<Fragment xmlns:android="http://schemas.android.com/apk/res/android"  
    android:name="com.florianBurel.itms.fragment.MusicListFragment"  
    android:layout_width="match_parent"  
    android:layout_height="match_parent"  
    android:id="@+id/list_fragment">  
  
</Fragment>
```

# Layout-xlarge/content\_frame

```
activity_item_twopane.xml x  MusicDetailFragment.java x  MusicListFragment.java 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"
    >

    <fragment
        android:id="@+id/item_list"
        android:name="fr.florianBurel.itunesmusicstore.MusicListFragment"
        android:layout_width="0dp"
        android:layout_height="match_parent"
        android:layout_weight="1"
        tools:layout="@android:layout/list_content" />

    <fragment
        android:id="@+id/item_detail_container"
        android:name="fr.florianBurel.itunesmusicstore.MusicDetailFragment"
        android:layout_width="0dp"
        android:layout_height="match_parent"
        android:layout_weight="3" />

</LinearLayout>
```

# Vers une singleActivity app

- La plupart des applications ne contiennent qu'une seule activity.
  - Sur téléphone :
    - Switch dynamique d'un fragment à l'autre
  - Sur tablette :
    - Affichage simultanée de plusieurs fragments
- La classe `FragmentManager` offre les fonctionnalités pour la gestion d'affichage de fragments

# Single Activity Application

- Ne pas définir statiquement le fragment dans le layout
- Utilisation d'un `FrameLayout`
  - Container de layout / fragment
- L'activity décide dynamiquement du/des fragments à afficher en fonction du context
- Les fragments :
  - déclare leur propre API pour se mettre à jour
    - Getter/setter, `reloadData()` ...
  - Utilise des interfaces type `onXListener` pour remonter les informations à l'activity

# Exemple

- Layout de l'activité sur telephone

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

  <FrameLayout
    android:id="@+id/content"
    android:layout_width="match_parent"
    android:layout_height="match_parent" >
</FrameLayout>

</LinearLayout>
```

# Remonter les évènements

- Best Practice: l'Interface
- Le fragment définit une interface pour tous les callback qu'il peut appeler.
- L'activity peut ensuite implémenter cette interface et se déclarer listener du fragment.
- A chaque réception de callback, l'activity n'a plus qu'à déterminer si elle doit faire une transition (Intent) ou une simple mise à jour.



# Conseil d'architecture

- Utiliser les activity juste pour la communication entre plusieurs fragment
  - Les Fragments n'ont pas d'IntentFilter
  - Laisser l'activity répondre aux callback des fragment et aux intents
  - L'activity sait si elle doit mettre à jour un Fragment qu'elle contient ou si elle doit appeler une autre Activity

# TP

- Créer un nouveau projet avec
  - en mode portrait une liste d'élève (prenom et nom) cliqueable menant sur une fiche detail reprenant le nom et prenom.
  - en mode paysage la liste sur la gauche et le detail sur la droite.
- Guide
  - Créer une variable boolean differente en fonction du mode portrait ou paysage
  - Créer les 3 fichiers xml
  - Créer le FragmentActivity et les 2 fragments

# Exemple: Fragments

- A mettre dans values et values-land

```
<resources>  
    <bool name="twoPane">false</bool>  
</resources>
```

- ListFragment.xml et DetailFragment.xml sont des fichiers xml basiques.

# Exemple: MainActivity

```
<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:orientation="horizontal"
    tools:context=".MainActivity" >
```

```
    <FrameLayout
        android:id="@+id/fl_fragment1"
        android:layout_width="0dp"
        android:layout_height="match_parent"
        android:layout_weight="1" >
</FrameLayout>
```

```
    <FrameLayout
        android:id="@+id/fl_fragment2"
        android:layout_width="0dp"
        android:layout_height="match_parent"
        android:layout_weight="1" >
</FrameLayout>
</LinearLayout>
```

# Exemple: ListFragment

```
@Override
    public View onCreateView(final LayoutInflater inflater, final ViewGroup container, final
Bundle savedInstanceState) {
    final View rootView = inflater.inflate(R.layout.list_fragment, container, false);

    lv = (ListView) rootView.findViewById(R.id.lv);
    eleveList = new ArrayList<Eleve>();
    eleveAdapter = new EleveAdapter(getActivity(), eleveList);
    lv.setAdapter(eleveAdapter);

    return rootView;
}
```

# Exemple: DetailFragment

```
public View onCreateView(final LayoutInflater inflater, final ViewGroup container, final
Bundle savedInstanceState) {
    final View rootView = inflater.inflate(R.layout.detail_fragment, container, false);
    tv = (TextView) rootView.findViewById(R.id.tv);
    eleve = null;
}
@Override
public void onResume() {
    super.onResume();
    refreshText();
}

public void setEleve(final Eleve eleve) {
    this.eleve = eleve;
    refreshText();
}
private void refreshText() {
    if (eleve == null) {
        tv.setText("Aucun élève");
    }
    else {
        tv.setText(eleve.getPrenom() + " " + eleve.getNom());
    }
}
```

# Exemple: MainActivity

```
public class MainActivity extends FragmentActivity implements OnClickListListener {

    private FrameLayout fl_fragment2;
    private ListFragment listFragment;
    private DetailFragment detailFragment = null;

    @Override
    protected void onCreate(final Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        fl_fragment2 = (FrameLayout) findViewById(R.id.fl_fragment2);
        //on lance le 1er fragment
        listFragment = new ListFragment();
        getSupportFragmentManager().beginTransaction().replace(R.id.fl_fragment1, listFragment).commit();

        //Si on en a un 2eme
        if (MyApplication.getInstance().isTwoPane()) {
            //on lance l'ajout
            detailFragment = new DetailFragment();
            getSupportFragmentManager().beginTransaction().replace(R.id.fl_fragment2, detailFragment).commit();
            fl_fragment2.setVisibility(View.VISIBLE);
        }
        else {
            fl_fragment2.setVisibility(View.GONE);
        }
    }
}
```

# Exemple: MainActivity

```
public class MainActivity extends FragmentActivity implements OnClickListListener {

    private FrameLayout fl_fragment2;
    private ListFragment listFragment;
    private DetailFragment detailFragment = null;

    @Override
    protected void onCreate(final Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        fl_fragment2 = (FrameLayout) findViewById(R.id.fl_fragment2);
        //on lance le 1er fragment
        listFragment = new ListFragment();
        getSupportFragmentManager().beginTransaction().replace(R.id.fl_fragment1, listFragment).commit();

        //Si on en a un 2eme
        if (MyApplication.getInstance().isTwoPane()) {
            //on lance l'ajout
            detailFragment = new DetailFragment();
            getSupportFragmentManager().beginTransaction().replace(R.id.fl_fragment2, detailFragment).commit();
            fl_fragment2.setVisibility(View.VISIBLE);
        }
        else {
            fl_fragment2.setVisibility(View.GONE);
        }
    }
}
```



# TP

- Implementer la communication entre les fragments
  - Callback vers l'activité du click sur la liste.

# Exemple: ListFragment

- Ajouter une interface

```
private OnClickListener onClickOnList = null;

public void setOnClickListener(final OnClickListener onClickOnList) {
    onClickOnList = onClickOnList;
}

public interface OnClickListener {
    void onClickOnList(Eleve eleve);
}
```

- Ajouter un listener sur la liste.

```
lv.setOnItemClickListener(this);

@Override
public void onItemClick(final AdapterView<?> parent, final View view, final int position, final long id) {

    final Eleve eleve = eleveList.get(position);

    if (onClickOnList != null) {
        onClickOnList.onClickOnList(eleve);
    }
}
```

# Exemple: MainActivity

- S'inscrire au callback

```
listFragment = new ListFragment();  
listFragment.setOnClickListener(this);
```

- Gestion du callback

```
@Override  
public void onClickOnList(final Eleve eleve) {  
    if (MyApplication.getInstance().isTwoPane()) {  
        detailFragment.setEleve(eleve);  
    }  
    else {  
        //on remplace le fragment visible par celui de l'ajout  
        final FragmentTransaction ft = getSupportFragmentManager().beginTransaction();  
        final DetailFragment df = new DetailFragment();  
  
        final Bundle bundle = new Bundle();  
        bundle.putParcelable(Constante.EXTRA_ELEVE, eleve);  
        df.setArguments(bundle);  
  
        ft.replace(R.id.fl_fragment1, df);  
        ft.addToBackStack(null); // permet de revenir à l'ecran d'avant avec un back bouton  
        ft.commit();  
    }  
}
```

# Exemple: DetailFragment

- Récupérer le bundle

```
final Bundle bundle = getArguments();  
if (bundle != null) {  
    eleve = bundle.getParcelable(Constante.EXTRA_ELEVE);  
}
```

# MENUS ET BOÎTES DE DIALOGUE

# Les boîtes de dialogues

- Les boîtes de dialogue sont des éléments visuels flottants
- Il existe plusieurs sortes de dialogue :
  - `alertDialog`
  - `DatePicker`
  - `TimePicker..`
- Héritent de la classe “Dialog”
- Permet à l'utilisateur de faire des actions rapides comme répondre à des questions, voir les messages...

- L'alertDialog est le dialogue le plus flexible disponible
  - Présenter un message suivi de boutons yes/no
  - Offrir une liste de choix
  - Offrir une saisie
- L'alertDialog se crée spécialement en appelant sa classe builder
  - `AlertDialog.Builder ad = new AlertDialog.Builder(this);`

# Exemple simple

```
AlertDialog.Builder ad = new AlertDialog.Builder(this);
ad.setTitle("essai dialog");
ad.setMessage("Vous avez ouvert cette boite.");
ad.setPositiveButton("Go Back", new OnClickListener() {
    public void onClick(DialogInterface dialog, int arg1) {
        //traitement
    }
});
ad.setNegativeButton("Move Forward", new OnClickListener() {
    public void onClick(DialogInterface dialog, int arg1) {
        // Do nothing
    }
});
ad.setCancelable(true);
ad.setOnCancelListener(new OnCancelListener() {
    public void onCancel(DialogInterface dialog) {
        // do nothing
    }
});
ad.show();
```



# AlertDialog

- Possibilité d'importer ses propres layout
- Si chargé depuis un fichier XML, on utilisera un `LayoutInflater`. `LayoutInflater` instancie un layout à partir du document XML. Récupérer de préférence un objet existant (mémoire) :
  - `ad.setView(LayoutInflater.from(this).inflate(R.layout.main,null));`

# Toast

- Moyen simple et rapide pour afficher une information à l'utilisateur
- Affichage limité à quelques secondes
- Disparition automatique

```
Toast monToast;  
monToast = Toast.makeText(this, "Ceci est un Toast",  
    Toast.LENGTH_LONG);  
monToast.show();
```

# Toast

```
//Afficher un toast
public static Toast toast;
public static void showToastOnUiThread(final Activity activity, final
String message, final int length) {
    activity.runOnUiThread(new Runnable() {

        @Override
        public void run() {
            //on efface l'ancien pour eviter d'en avoir 50 en attente
            if (toast != null) {
                toast.cancel();
            }
            toast = Toast.makeText(activity, message, length);
            toast.show();
        }
    });
}
```

# TP : Dialog

- Préparer une classe avec 3 méthodes
  - Une pour lancer une alerte dialog d'affichage
  - Une pour lancer une alerte dialog ou on demande une valeur à l'utilisateur

```
final EditText input = new EditText(activity);  
new AlertDialog.Builder(activity).setView(input)...
```
  - Une pour afficher un toast

# OptionsMenu

- Créé lors de l'appui sur le bouton Menu du téléphone
- Événement reçu par l'activité en cours
  - `public boolean onCreateOptionsMenu(Menu menu)`
- Possibilité d'ajouter des MenuItem à ce menu
  - Les 6 premiers seront affichés
  - Si +, un bouton «more» est ajouté automatiquement
- Callback reçu sur sélection d'un élément
  - `public boolean onOptionsItemSelected(MenuItem i)`

# Exemple

```
/* Crée le menu */
public boolean onCreateOptionsMenu(Menu menu)
{
    menu.add(0, MENU_NEW_GAME, 0, "New Game");
    menu.add(0, MENU_QUIT, 0, "Quit");
    return super.onCreateOptionsMenu(Menu menu);
}

/* Handles item selections */
public boolean onOptionsItemSelected(MenuItem item)
{
    switch (item.getItemId())
    {
        case MENU_NEW_GAME:
            newGame();
            break;
        case MENU_QUIT:
            quit();
            break;
    }
    return super.onOptionsItemSelected(MenuItem item);
}
```

# MenuItem avec icônes

- Possibilité d'ajouter une icône à un menuItem
  - `public void setIcon(Drawable icon)`
- Possibilité d'utiliser nos propres drawables ou ceux d'Android
  - accessible via `android.R.drawable.ic_*`

```
menu.add(...).setIcon(R.drawable.myIcon);  
menu.add(...).setIcon(android.R.drawable.ic_menu_agenda)
```

# Sous Menu

- Possibilité d'ajouter un sous menu à chaque élément d'un menu
- Méthode de la classe Menu
  - `public SubMenu addSubMenu(String menuTitle)`
- Possibilité d'ajouter des items au sous menu retourné

```
final SubMenu fileMenu = menu.addSubMenu("File");  
fileMenu.add("new");  
fileMenu.add("open");  
fileMenu.add("save");
```

- Pour un Menu dynamique : `onPrepareOptionsMenu`



# Exemple

```
public boolean onCreateOptionsMenu(Menu menu) {  
    boolean result =  
super.onCreateOptionsMenu(menu);  
    SubMenu fileMenu = menu.addSubMenu("File");  
    fileMenu.add("new");  
    fileMenu.add("open");  
    fileMenu.add("save");  
    return result;  
}
```

# Action bar

- À partir de la version 3.0 d'Android, l'affichage du menu d'une activité s'appelle à l'intérieur d'une ActionBar
- C'est une barre de menu se situant au plus haut niveau de notre activité



# Utilisation

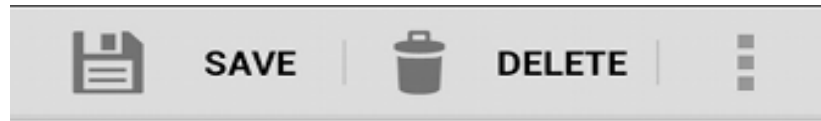
- Pour l'utiliser, il suffit donc de compiler avec au minimum la version 11 du sdk (Android 3.0) avec un target-sdk suffisamment haut

```
<manifest ... >  
  <uses-sdk android:minSdkVersion="4"  
            android:targetSdkVersion="11" />  
  ...  
</manifest>
```

# Ajouter des icones sur l'ActionBar

© Matelli sarl 2013

- Nous ajoutons un menu de manière classique puis nous utilisons la fonction “setShowAsAction” avec plusieurs constantes :



```
public boolean onCreateOptionsMenu(Menu menu) {  
  
    MenuItem mnu1 = menu.add(0, 0, 0, "save");  
    mnu1.setIcon(R.drawable.save);  
    mnu1.setShowAsAction(MenuItem.SHOW_AS_ACTION_ALWAYS |  
        MenuItem.SHOW_AS_ACTION_WITH_TEXT);  
  
    MenuItem mnu2 = menu.add(0, 1, 0, "delete");  
    mnu2.setIcon(R.drawable.delete);  
    mnu2.setShowAsAction(MenuItem.SHOW_AS_ACTION_IF_ROOM |  
        MenuItem.SHOW_AS_ACTION_WITH_TEXT);  
  
    return true;  
}
```

# Détails

- **SHOW\_AS\_ACTION\_ALWAYS :**
  - Nous permet d'afficher l'icone même s'il y a peu de place
- **SHOW\_AS\_ACTION\_IF\_ROOM :**
  - Affiche l'icone seulement s'il y a de la place dans la barre
- **MenuItem.SHOW\_AS\_ACTION\_WITH\_TEXT :**
  - Affiche le texte à côté de l'icone

# Menu Contextuel

- S'affiche sur une boîte de dialogue
  - longClick sur un élément
  - L'élément doit d'abord s'enregistrer
    - `registerForContextMenu(view);`
- Callback reçu lors de l'affichage du menu
  - `void onCreateContextMenu(Menu m, View v)`
- Callback reçu lors de la sélection d'un élément
  - `boolean onContextMenuItemSelected(MenuItem i)`

# TP: Menu

- Ajouter un menu sur les 2 activités (liste et détail)
  - Liste
    - possibilité de rafraîchir la liste
  - Détail
    - Ajouter un menu proposant de changer le nom de l'élève
    - Propose de rechercher le texte sur Google
      - Utilisation d'un Intent Implicite

```
final Uri uri = Uri.parse("http://www.google.com/#q=Alfred");  
final Intent intent = new Intent(Intent.ACTION_VIEW, uri);  
startActivity(intent);
```

# BROADCAST



# Android

- Répondre à des événements extérieurs

# Broadcast receiver

- Le concept des broadcast receiver est simple:
  - ❑ Réceptionner un intent qui broadcaste sur toutes les applications
  - ❑ Chaque classe broadcast receiver doit être déclarée dans le manifest et associée à des broadcasts
  - ❑ Une seule méthode **onReceive()**
  - ❑ Un **BroadcastReceiver** ne vit que le temps de traiter votre **onReceive()**.

```
public class MyIntentReceiver extends BroadcastReceiver {  
    @Override  
    public void onReceive(Context context, Intent intent) {  
        ...  
    }  
}
```

# Broadcast receiver

- Lecture et réception des sms:

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.android2ee.formation.service.smslisteners.smslistenertuto"
    android:versionCode="1"
    android:versionName="1.0" >

    <uses-sdk
        android:minSdkVersion="8"
        android:targetSdkVersion="16" />

    <uses-permission android:name="android.permission.RECEIVE_SMS" />
    <uses-permission android:name="android.permission.VIBRATE" />

    <application
        android:name=".SmsListenerApplication"
        android:label="@string/app_name"
        android:theme="@style/AppTheme" >
        <receiver android:name=".MySmsReceiver" >
            <intent-filter>
                <action android:name="android.provider.Telephony.SMS_RECEIVED" />
            </intent-filter>
        </receiver>

        <service android:name=".MySmsService" >
        </service>
    </application>
</manifest>
```

# Envoyer un broadcast

- Pour envoyer un broadcast, il nous suffit tout simplement de définir un message dans un intent et d'envoyer cet intent avec la fonction :  
    sendBroadcast(intent)

```
sendBroadcast(new Intent("com.formation.futon"));
```

## TP : broadcast

- Créer une application qui envoie un toast à chaque réception de sms.
  - Doc :  
<https://developer.android.com/reference/android/provider/Telephony.Sms.Intents.html>

# SERVICES

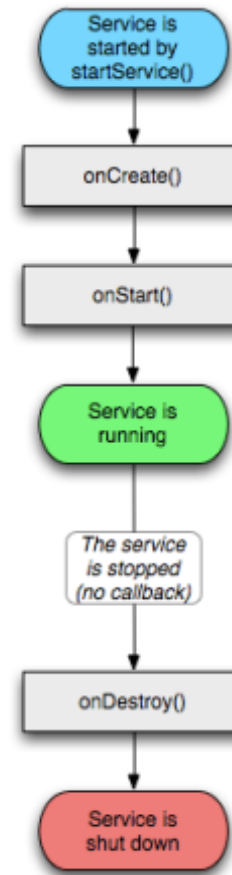
# Services

- Les services ont pour but de réaliser des tâches de fond sans aucune interaction avec l'utilisateur pour une durée indéfinie.

- Les services doivent être déclarés dans `AndroidManifest.xml`:

```
<service android:name=".subpackagename.ServiceName"/>
```

- 2 types de service
  - Local : Même processus que l'application
  - Remote : En dehors du processus de l'application



# Services

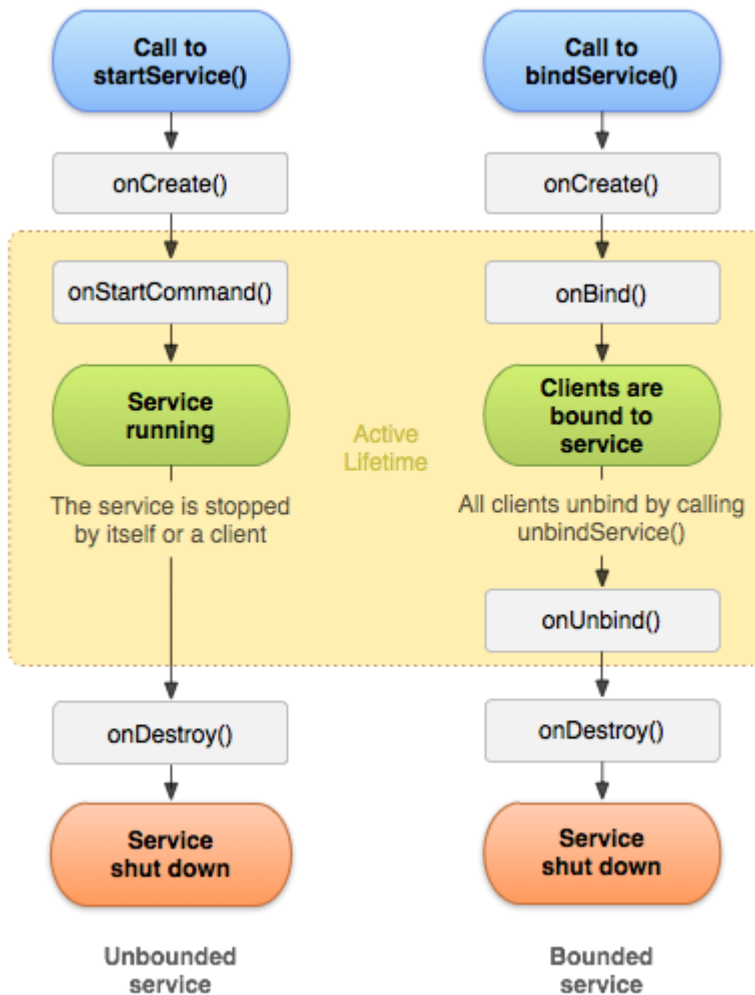
- Ils doivent étendre la classe Service dont vous devrez surcharger les méthodes suivantes en fonction de vos besoins

```
void onCreate(); // initialisation des ressources
void onStartCommand(Intent intent, int flags, int startId); // SDK>2.0 la tâche de fond démarre
void onDestroy(); // libération des ressources

IBinder onBind(Intent intent); // connexion client distant
boolean onUnbind(Intent intent); // déconnexion d'un client
void onRebind(Intent intent)
```

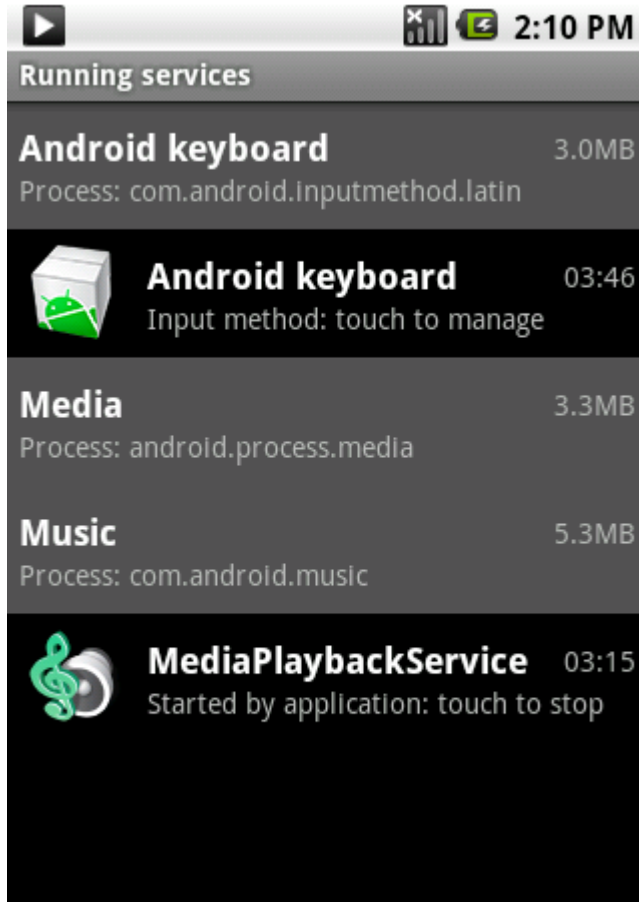


# Services



- Cycle de vie
- `startService()` invoque la méthode `onCreate()`
- `onStartCommand()` [service running]
- `stopService()` invoque la méthode `onDestroy()`
- `bindService()` qui appelle uniquement la méthode `onCreate()`

# Services



- Il est possible de voir la liste des services exécutés en allant dans :  
*Menu > Settings > Applications > Running Services > du téléphone:*

# Démarrer/arrêter un service

- Il est possible d'utiliser un appel explicite ou implicite  
`this.stopSelf()` : dans le service pour se stopper lui-même.

```
// implicite (à définir dans le manifest)  
startService(new Intent(MyService.MY_ACTION));  
  
// Explicite  
startService(new Intent(this, MYService.class));  
  
// stop service en appelant la methode stopService  
et en lui passant le service a stopper  
stopService(new Intent(this, MYService.class));
```

# Exemple

```
public class BackgroundService extends Service {

    private Timer timer;

    @Override
    public void onCreate() {
        super.onCreate();
        timer = new Timer();
    }

    @Override
    public int onStartCommand(final Intent intent, final int flags, final int startId) {
        timer.scheduleAtFixedRate(new TimerTask() {
            @Override
            public void run() {
                // Executer votre tâche
            }
        }, 0, 60000);
        return START_NOT_STICKY;
    }

    @Override
    public void onDestroy() {
        this.timer.cancel();
    }

    @Override
    public IBinder onBind(final Intent intent) {
        return null;
    }

}
```

# Service Binder

- Grâce à l'Intent pour démarrer le service, on peut transmettre des valeur au service en utilisant `intent.putExtra`
- En utilisant le binder pour lancer le service, on récupère une instance de celui-ci.

# onStartCommand

- `public int onStartCommand (Intent intent, int flags, int startId)`
- Valeur de retour: elle correspond au comportement que doit adopter le service par rapport au processus qui l'a créé. Plusieurs constantes sont disponibles dans la classe `Service`.
  - `START_STICKY`: par défaut, redémarre le service et donc le processus, après un arrêt inattendu de celui-ci (ex: kill du process dans le DDMS).
  - `START_NOT_STICKY`: le service n'est pas redémarré lors d'un arrêt inattendu de notre processus
  - `START_REDELIVER_INTENT`: pareil que `START_STICKY` mais redélivre l'intent en paramètre.

# RemoteService

- Exécution dans un autre processus
- AIDL (Android Interface Définition Language) afin de communiquer entre 2 processus
  - Décrire les méthodes publiques et données échangées
  - Fichier .aidl connu des 2 processus
  - Echange de données primitifs (bool, int...), List, Set, Map et parcelable

# Fichier AIDL

- Il sert d'interface entre le service et le client

```
interface IRemoteBackgroundService {  
    int getPid(); // Renvoie le PID du processus du service  
    Data getData(); // Renvoie notre objet mis à jour  
    //void updateData(in Data data);  
}
```



## TP : services

- Implémenter un service qui sera démarré grâce à une interface possédant des boutons start/stop.
- L'intérêt de cet exercice est de constater que le service ne s'arrête pas lorsque l'activité est fermée.
- Créer un service qui affiche un toast avec les coordonnées GPS du téléphone.

```
locationMgr = (LocationManager) getSystemService(Context.LOCATION_SERVICE);  
locationMgr.requestLocationUpdates(LocationManager.GPS_PROVIDER, 10000, 0,  
this);
```

```
@Override  
public void onLocationChanged(final Location location) {  
    final Double latitude = location.getLatitude();  
    final Double longitude = location.getLongitude();  
    //toast it
```

# Service distant

- Lorsque nous déclarons un service, celui ci peut communiquer avec tout le téléphone via un broadcast.
- Le service distant est une solution permettant de binder un service avec une activité.

# aidl

- Android Interface Definition Language est un langage de description permettant de définir des interfaces de communication entre le service et l'appelant.
- Une configuration se crée via un fichier d'extention .aidl que l'on place sur les 2 parties

# Communication InterProcessus

- Librairie otto
  - <http://square.github.io/otto/>

- Application

```
Bus eventBus = new Bus();  
public static Bus getEventBus() {  
    return eventBus;  
}
```

- Classe de réception

```
MyApplication.getEventBus().register(this);  
@Subscribe  
public void answerAvailable(LoginSuccesEvent event) {  
    //traitements  
}
```

- Classe émettrice

```
MyApplication.getEventBus().post(new LoginSuccesEvent(null, true, true));
```

# TP

- Grâce à Otto, communiquer avec le service précédent pour récupérer les coordonnées.
- Ajouter un bouton stop service qui arrête le service.
- Vérifier ce qu'il se passe si on tourne l'écran et que le service est lancé.
- Faire en sorte que quand l'activité est détruite, le service le soit aussi.

# COMMON ACTIVITY

# Common Activity

- Une activity pour les surveiller toutes.
- Un xml avec un `FrameLayout` pour laisser la place au activité.

```
<FrameLayout
    android:id="@+id/container"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:layout_below="@+global/toolbar_top_layout"
    android:layout_margin="1dp" />
```

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

# Common Activity

- Les Activités filles pourront définir leur XML grâce à ces methodes.

```
@Override
public void setContentView(final int layoutResID) {
    final View v = getLayoutInflater().inflate(layoutResID, null);
    setContentView(v);
}
```

```
@Override
public void setContentView(final View v) {
    final FrameLayout container = (FrameLayout) findViewById(R.id.container);
    container.removeAllViews();
    container.addView(v);
}
```



# Common Activity

- Exemple d'Activity Fille

```
public class MainActivity extends CommonActivity {  
  
    private Button bt;  
    @Override  
    protected void onCreate(final Bundle savedInstanceState) {  
  
        // Cette méthode est à appeler APRES la récupération des extras  
        super.onCreate(savedInstanceState);  
        super setContentView(R.layout.activity_main);  
  
        bt = (Button) findViewById(R.id.bt);  
        bt.setOnClickListener(this);  
  
    }  
}
```

# Common Activity

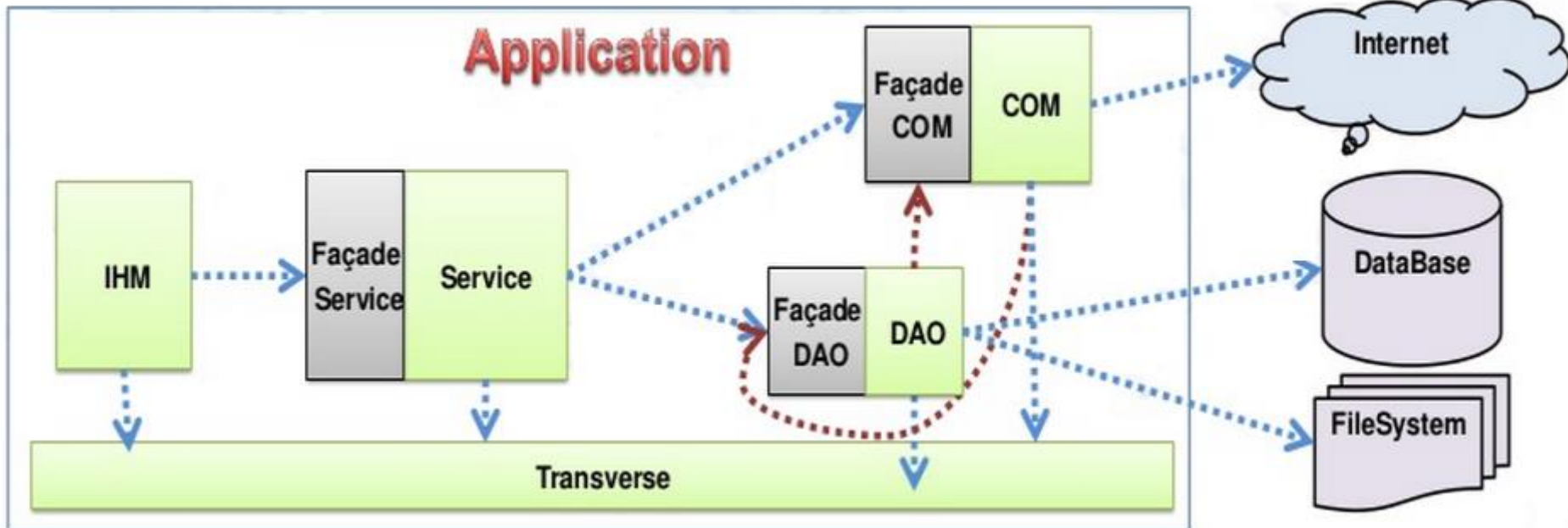
- Permet de centraliser les informations pour chaque activité
  - Mettre un log sur l'activité courante
  - Mettre à dispo une fenêtre d'attente
  - Mettre en commun une base Graphique

# Common Activity

- Permet de centraliser les informations pour chaque activité
  - Mettre un log sur l'activité courante
  - Mettre à dispo une fenêtre d'attente
  - Mettre en commun une base Graphique

# Architecture Application

- Suggestion d'architecture



- Couche transverse
  - Beans, ExceptionManager, StringUtils, LogUtils,

# TP

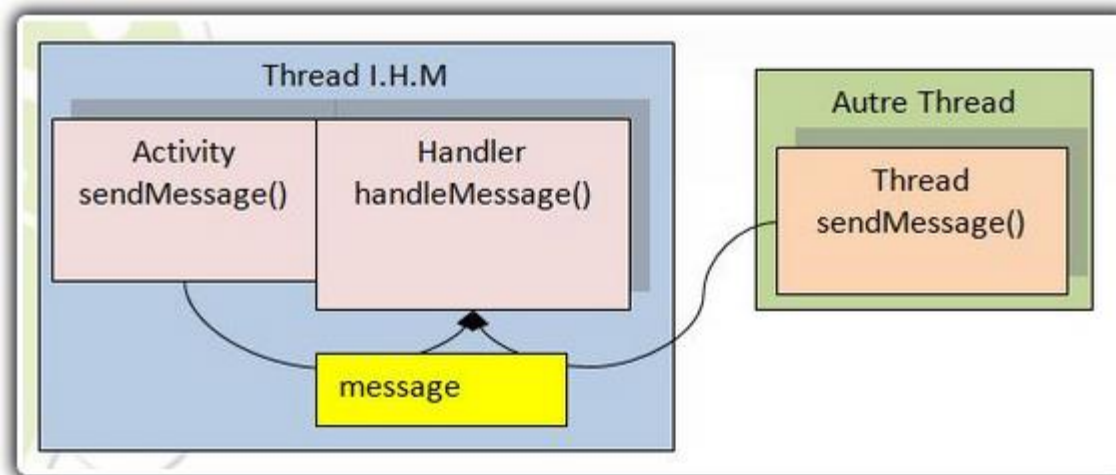
- Créer une classe `CommonActivity` et son layout permettant de mettre un titre commun à toutes les activités.
- Faire en sorte que `mainActivity` étende de `CommonActivity` et puisse changer son titre.

# HANDLER ET ASYNCTASK

# Handler

- Permet une communication depuis n'importe quel thread vers le ThreadUi.
- Uniquement pour de la mise à jour de l'IHM
- La Thread envoie le message au Handler en utilisant l'une des méthodes suivantes :
  - **sendMessage** (envoie le message et le place à la fin de la queue)
  - **sendMessageToFrontOfQueue** (envoie le message et le place au début de la queue)
  - **sendMessageAtTime** (envoie le message au moment donné en paramètre et le place à la fin de la queue)
  - **sendMessageDelayed** (envoie le message après un temps d'attente passé en paramètre et le place à la fin de la queue)

# Handler





# Handler

- Définition du handler

```
private final Handler handler = new Handler() {
    @Override
    public void handleMessage(final Message msg) {
        runOnUiThread(new Runnable() {
            @Override
            public void run() {
                switch (msg.what) {
                    case MSG_START_PROGRESS:
                        if (!progressDialog.isShowing()) {
                            progressDialog.show();
                        }
                        break;
                    case MSG_STOP_PROGRESS:
                        if (progressDialog.isShowing()) {
                            progressDialog.cancel();
                        }
                        break;
                }
            }
        });
    }
};
```

# Handler

- Utilisation

```
private static final int MSG_START_PROGRESS = 0;  
handler.sendMessage(MSG_START_PROGRESS);
```

# TP

- Reprendre le TP sur CommonActivity et ajouter la possibilité de faire apparaître et disparaître une fenêtre d'attente pour toutes les activités qui en héritent.

```
public void startProgress() {  
    // à remplir  
}  
  
public void stopProgress() {  
    // à remplir  
}
```

# AsyncTask

- AsyncTask ou asynchrone task, comme un Thread.
- Avantage
  - Le thread se crée automatiquement
  - Et la communication entre les différents thread est simplifiée.
- A utiliser pour les traitements lourds
  - Gros calculs
  - Requête WS
- 5 AsyncTask maximum simultanément, après ça se complique.
- Une tache ne peut être executée qu'une seule fois.

# AsyncTask

```
public class ChargementEleveAT extends AsyncTask<Params, Progress, Result> {

    public ChargementEleveAT() {
    }

    @Override // peut être sur l'UIThread
    protected void onPreExecute() {
        super.onPreExecute();
    }

    // Garanti en dehors de l'UIThread
    protected Result doInBackground(final Params... params) {
        // traitement
        return null;
    };

    // peut être sur l'UIThread
    protected void onProgressUpdate(final Progress... values) {
        // mise à jour progress bar ou UI
    };

    // peut être sur l'UIThread
    protected void onPostExecute(final Result result) {
    };
}
```

# AsyncTask

- Utilisation

```
private ChargementEleveAT chargementEleveAT = null;

if (chargementEleveAT == null || chargementEleveAT.getStatus() == Status.FINISHED) {
    chargementEleveAT = new ChargementEleveAT(this);
}

if (chargementEleveAT.getStatus() == Status.PENDING) {
    chargementEleveAT.execute();
};
```

- Si l'activité qui a lancé l'AT est détruite, elle l'est aussi.

# AsyncTask

- Arrêter une AsyncTask

```
chargementEleveAT.cancel(true);
```

- Si le « doInBackground » a commencé il finira son exécution et on ne passera pas par le « onPostExecute »

- Faire stopper le doInBackground

```
if(isCancelled()) {  
    return null;  
}
```

# TP

- Reprendre le TP de l'affichage des élèves sur une ListView
- Faire en sorte que le clic sur le bouton lance une AsyncTask qui chargera les données.
- L'AsyncTask devra rendre son résultat à l'aide de l'interface suivante :

```
public interface LoadEleveListener {  
    //succes  
    void eleveLoad(List<Eleve> eleve);  
  
    //fail  
    void loadFail(String message);  
  
    void updateChargement(int max, int current);  
}
```



# TP

```
• protected String doInBackground(final Void... arg0) {  
    //Appel WS avec attente  
    if (random.nextInt(2) > 0) {  
        for (int i = 0; i < nbrEleve; i++) {  
            eleveList.add(new Eleve("Jean" + i, "Pierre" + i, i % 2 == 0));  
            publishProgress(i);  
            SystemClock.sleep(1000);  
        }  
        return null;  
    }  
    else {  
        //Simulation de l'echec  
        SystemClock.sleep(2000);  
        return "Pas de chance cela à échoué";  
    }  
};
```

# USER EXPERIENCE

# User Experience

- Qu'est ce que l'User Experience?
- Official Design Guidelines
  - <http://developer.android.com/design/index.html>
- Un peu d'aide
  - <https://android-arsenal.com/>

# Le règles

- Donner un retour à l'utilisateur de son action.
  - Le bouton Bootstrap



# Le règles

- 48 dp la taille d'un doigt
- 8dp l'espace minimum entre 2 UI élément
- Reprendre ce que l'utilisateur connaît.
  - (Facebook, Twiter, youtube...)

# Le règles

- Le faire patienter (Si possible lui indiquer combien de temps)
  - `ProgressBar`, `animation...`
- Gestion d'un cache mémoire
- Affichage de données obsolètes avec chargement en arrière plan.
- Donner de la vie avec les animations.

# Inscription

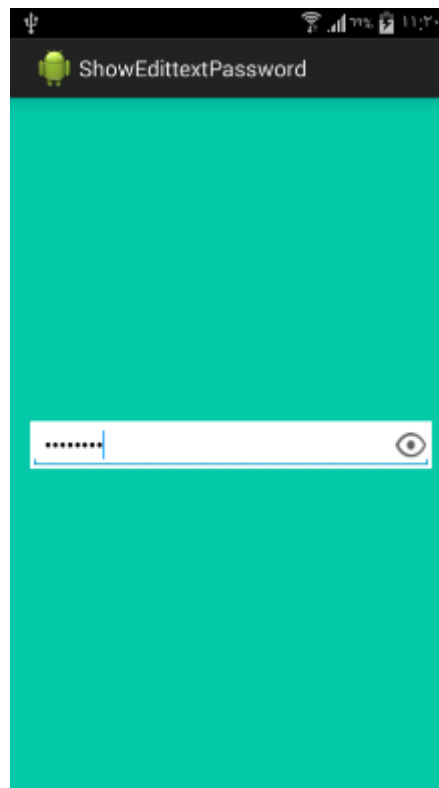
- Inscription le plus tard possible
- Utiliser la connexion par les réseau sociaux, facebook ou google+



The image shows a dark-themed login and registration interface for 'FANCY'. At the top, the word 'FANCY' is displayed in large, white, sans-serif capital letters, with the text 'Se connecter' centered below it. On the left side, there are three social media login buttons: a red button with the Google+ icon and text 'Sign in with Google', a blue button with the Facebook icon and text 'Se connecter avec Facebook', and a light blue button with the Twitter icon and text 'Se connecter avec Twitter'. On the right side, there are two white input fields: the top one is labeled 'Email' and the bottom one is labeled 'Mot de passe'. Below the password field is a blue button labeled 'Se connecter' and a link labeled 'Mot de passe oublié ?'. At the bottom center, there is a link that reads 'Besoin d'un compte ? S'inscrire'.

# Connexion

- Librairie **ShowEditTextPassword** pour le mot de passe.



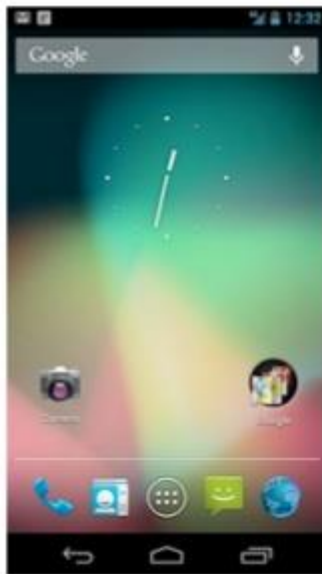


# Différence iPhone Android (Andy Fitzgerald)

© Matelli sarl 2013

## HOME SCREEN

■ ■ Android



■ ■ iPhone

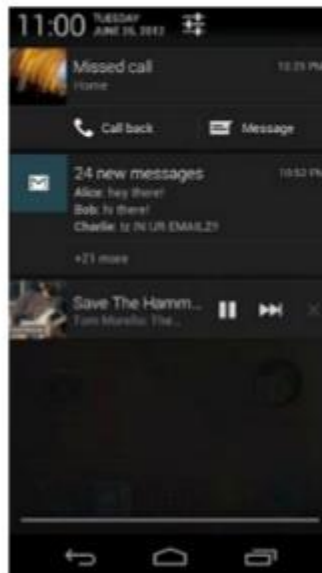


# Différence iPhone Android (Andy Fitzgerald)

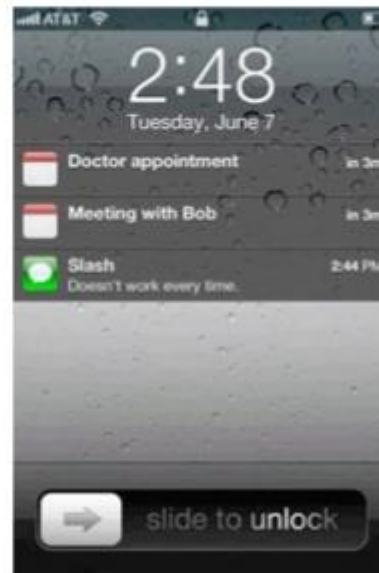
© Matelli sarl 2013

## NOTIFICATIONS

### ■ ■ Android



### ■ ■ iPhone



# Différence iPhone Android (Andy Fitzgerald)

© Matelli sarl 2013

## SETTINGS

### ■ ■ Android



### ■ ■ iPhone

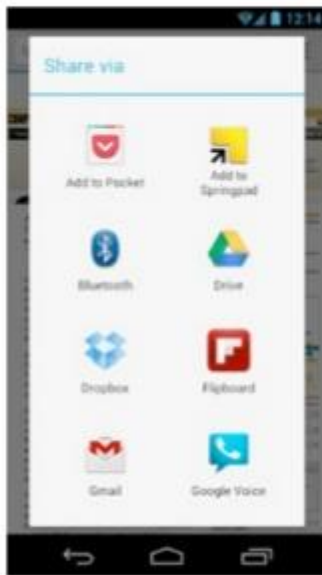


# Différence iPhone Android (Andy Fitzgerald)

© Matelli sarl 2013

## SHARING

■ ■ Android



■ ■ iPhone



- Réaliser un bouton bootstrap

- Le fond

```
<?xml version="1.0" encoding="utf-8"?>
<shape xmlns:android="http://schemas.android.com/apk/res/android"
    android:padding="10dp" android:shape="rectangle" >

    <solid android:color="@color/main" />

    <corners
        android:bottomLeftRadius="5dp"
        android:bottomRightRadius="5dp"
        android:topLeftRadius="5dp"
        android:topRightRadius="5dp" />
</shape>
```

- Le bouton

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

    <item android:drawable="@drawable/background_button_main_on" android:state_pressed="true"/>
    <item android:drawable="@drawable/background_button_main"/>

</selector>
```

- Réaliser un bouton 9Patch
  - Pixel gauche et haut pour la duplication
  - Pixel droite et bas, pour l'espace du texte.



# Android

- Persistance des données

# Préférences partagées

- Stockage des informations sous forme clef / valeurs
- Mise en place rapide
- Idéales pour de petites quantités d'informations
  - nom de l'utilisateur
  - email
  - préférence de tri d'une liste...
- Classe de base : SharedPreferences



# Préférences partagées

- Il est possible de récupérer cet objet de plusieurs façons avec des droits différents :
  - De manière générale:
  - `getSharedPreferences(String nom, int droit)`
- Informations disponibles en lecture seule par des méthodes associées :
  - `getString()`, `getInt()`...

# Préférences partagées

- Permissions
- Il existe 3 type de paramètres:
  - `MODE_PRIVATE`: par défaut, créé et accessible uniquement dans l'application courante.
  - `MODE_WORLD_PRIVATE` : les autres applications y ont accès en lecture seule.
  - `MODE_WORLD_WRITABLE` : lecture/écriture partout.

# Préférences partagées

L'enregistrement de préférences se fait avec un objet de type Editor, récupéré par la fonction `SharedPreferences.edit()`;

```
SharedPreferences mPrefs =  
getSharedPreferences("SAVEDATA", MODE_PRIVATE);  
Editor editor = mPrefs.edit();  
editor.putBoolean("hold", false);  
editor.putInt("place", 0);  
editor.commit();
```

# Introduction aux bases de données

© Matelli sarl 2013

- Sqlite est beaucoup utilisé dans les systèmes embarqués car il allie simplicité et mémoire légère.
- Pour Android, la base de données Sqlite est native et directement connectée à la machine virtuelle. De ce fait, toutes les applications peuvent l'utiliser.
- Cependant son API n'est pas jdbc mais une API plus légère

# Exemple

```
db.execSQL("create table produits ( _id integer primary key  
autoincrement, "  
+ "codebarre text not null, titre text not null, "  
+ "description text not null"  
+ ");");
```

# Introduction aux BDD

- Il y a certaines fonctionnalités non disponibles sur sqlite:
  - Les jointures externes
  - Les foreign key

# Créer une BDD

- Aucune base de données ne vous est fournie automatiquement par Android.
- Il sera nécessaire de créer votre propre base et de la peupler.
- Pour cela, il faut utiliser une redéfinition de la classe SQLiteOpenHelper.

# Créer une BDD

- Trois méthodes à ré-implémenter :
  - Le constructeur : a besoin en paramètres du context(Activity), d'un nom et d'un numéro de version.
  - Oncreate(): qui nous donnera un objet SQLiteDatabase à peupler
  - OnUpgrade(): comportement à adopter si la version de la base change. Il possède comme argument la nouvelle version, l'ancienne ainsi qu'un objet SQLiteDatabase. De manière générale, nous dropons les anciennes tables pour créer les nouvelles



# Ouvrir une connexion

- Selon le contexte, appel des 2 méthodes
  - `getReadableDatabase()`: ouvre la base en lecture seule
  - `getWritableDatabase()` : ouvre une connexion et accepte l'écriture.
- Ces 2 méthodes nous fournissent un objet `SQLiteDatabase` qui va nous permettre d'effectuer des requêtes.

# Création des tables

- Nous utiliserons la méthode `db.execSQL(String)` afin de lancer une requête de creation.
- `db.execSQL("CREATE TABLE constants ( id INTEGER PRIMARY KEY AUTOINCREMENT, title TEXT, value REAL);");`
- S'écrit dans le `onCreate(SQLiteDatabase db)` de la classe `SQLiteOpenHelper`

# Exemple : Création de la table

```
public class MaBaseSQLite extends SQLiteOpenHelper {

    private static final String NOM_BDD = "mabase.db";
    private static final int VERSION_BDD = 1;

    public MaBaseSQLite(Context context, SQLiteDatabase.CursorFactory factory) {
        super(context, NOM_BDD, factory, VERSION_BDD);
    }

    @Override
    public void onCreate(SQLiteDatabase sqLiteDatabase) {
        //on créé la table à partir de la requête écrite dans la variable CREATE_ELEVE_TABLE
        sqLiteDatabase.execSQL(EleveBDD.CREATE_ELEVE_TABLE);
    }

    @Override
    public void onUpgrade(SQLiteDatabase sqLiteDatabase, int oldVersion, int newVersion) {
        //On peut fait ce qu'on veut ici moi j'ai décidé de supprimer la table et de la recréer
        //comme ça lorsque je change la version les id repartent de 0
        sqLiteDatabase.execSQL("DROP TABLE " + EleveBDD.TABLE_ELEVE + ");
        onCreate(sqLiteDatabase);
    }
}
```

# Exemple : Création de la table

```

public class EleveBDDManager {

    public static final String TABLE_ELEVE = "Eleve";

    private static final String COL_ID = "ID";
    private static final String COL_PRENOM = "Prenom";
    private static final String COL_NOM = "Nom";

    public static final String CREATE_ELEVE_TABLE = "CREATE TABLE " + TABLE_ELEVE + " (" + COL_ID + "
    INTEGER PRIMARY KEY AUTOINCREMENT, "
        + COL_PRENOM + " TEXT NOT NULL, " + COL_NOM + " TEXT NOT NULL);";

    private SQLiteDatabase bdd;
    private MaBaseSQLite maBaseSQLite;

    public EleveBDD(Context context) {
        //On créer la BDD et sa table
        maBaseSQLite = new MaBaseSQLite(context, null);
    }

    /* *****
     * Gestion session ***
     * ***** */
    private void open() {
        bdd = maBaseSQLite.getWritableDatabase();
    }

    private void close() {
        bdd.close();
    }

    ...
}

```

# Exemple : Insert - Update

```

public void insertEleve(Eleve eleve) {
    open();
    //Création d'un ContentValues (fonctionne comme une HashMap)
    ContentValues values = new ContentValues();
    //on lui ajoute une valeur associée à une clé (qui est le nom de la colonne dans laquelle on
    veut mettre la
    // valeur)
    values.put(COL_PRENOM, eleve.getPrenom());
    values.put(COL_NOM, eleve.getNom());
    //on insère l'objet dans la BDD via le ContentValues
    eleve.setId(bdd.insert(TABLE_ELEVE, null, values););
    if (eleve.getId() == -1) {
        //gestion erreur
    }
    close();
}

public int updateEleve(Eleve eleve) {
    open();
    //La mise à jour d'un élève dans la BDD fonctionne plus ou moins comme une insertion
    //il faut simplement préciser quel élève on doit mettre à jour grâce à l'ID
    ContentValues values = new ContentValues();
    values.put(COL_PRENOM, eleve.getPrenom());
    values.put(COL_NOM, eleve.getNom());
    int result = bdd.update(TABLE_ELEVE, values, COL_ID + " = " + eleve.getId(), null);
    close();
    return result;
}

```

# Exemple : Remove - Get

```
public int removeEleveWithID(int id) {  
    open();  
    //Suppression d'un élève de la BDD grâce à l'ID  
    int result = bdd.delete(TABLE_ELEVE, COL_ID + " = " + id, null);  
    close();  
    return result;  
}
```

```
public List<Eleve> getAllEleves() {  
    open();  
    //Récupère dans un Cursor tous les élèves correspondant au prénom  
    Cursor c = bdd.query(TABLE_ELEVE, new String[] { COL_ID, COL_PRENOM, COL_NOM }, null, null,  
null, null, null);  
    List<Eleve> result = cursorToEleves(c);  
    close();  
    return result;  
}
```

# Utilisation d'un cursor

- Retourner le nombre d'enregistrements : `getCount()`
- Itération du cursor avec les méthodes “`moveToFirst()`”, “`moveToNext()`” et “`isAfterLast()`”
- Retourner les noms des colonnes avec `getColumnNames()`, pour les afficher
- `getColumnIndex()`, nous renvoie l'index du nom de la colonne donné en paramètre
- Récupération des informations de la colonne avec `getString()`, `getInt()`, etc.
- Libérer le curseur avec le méthode `close()`

# Exemple : Cursor

```
//Cette méthode permet de convertir un cursor en list d'Eleve
private List<Eleve> cursorToElevs(Cursor c) {
    ArrayList<Eleve> eleveListe = new ArrayList<Eleve>();

    if (c != null) {
        //Sinon on se place sur le premier élément
        if (c.moveToFirst()) {
            do {
                Eleve eleveBean = new Eleve(c.getString(c.getColumnIndex(COL_NOM)),
                c.getString(c.getColumnIndex(COL_PRENOM)), false);
                eleveListe.add(eleveBean);
            } while (c.moveToNext());
        }
    }

    //On ferme le cursor
    c.close();

    //On retourne la liste
    return eleveListe;
}
```



# TP : Sqlite

© Matelli sarl 2013

- Rendre persistant la liste d'élève de la listView

# Les cursors loaders

- Un cursor loader est une classe permettant de charger des curseurs de base de données
  - Chaque requête est exécuté de façon asynchrone
  - Le curseur se met à jour automatiquement
- Depuis les versions 3.xx charger un cursor dans l'UI thread devient déprécié (`cursor.requery`, `activity.startManagingCursor`), car cela ralentit l'application.

# Les cursors loaders

- Le principe est donc d'implémenter des callbacks sur notre activité pour charger le cursor.
  - `Loader<Cursor> onCreateLoader(int arg0, Bundle arg1)`
  - `onLoadFinished(Loader<Cursor> arg0, Cursor arg1)`
  - `onLoaderReset(Loader<Cursor> arg0)`
- Le cursor loader ne peut être chargé seulement avec une URI pointant sur un content Provider.
- Il devient donc obligatoire de préparer un content provider.

# LoaderManager.LoaderCallbacks<Cursor>

© Matelli sarl 2013

© Matelli sarl 2012

- Cette interface est à rajouter sur chaque activité ou à chaque fragment.
- Elle permettra d'appeler le chargement du curseur au moyen de la fonction
- `getLoaderManager().initLoader(id, bundle, callback)`
  - Le 1er paramètre est un Id unique qui permet de retrouver le loader
  - Le 2eme paramètre est un bundle pour transmettre des infos
  - Le 3eme paramètre est le callback lui même

# Méthodes utilisées

- Une fois la fonction appelée, 3 méthodes seront appelées dans le callback . Il faut donc les redéfinir:
  - onCreateLoader(): c'est ici que l'on crée et retourne le curseur dans un cursor loader. Nous sommes obligés de passer par un content provider
  - onLoadFinished(): appelé une fois que le cursor à été chargé. Nous mettons à jour les éléments des UI.
  - onLoaderReset(): appelé lorsque le cursor est fermé

# Exemple

© Matelli sarl 2013

```
public class Main extends FragmentActivity implements LoaderManager.LoaderCallbacks<Cursor> {
    SimpleCursorAdapter aa;

    @Override
    public Loader<Cursor> onCreateLoader(int arg0, Bundle arg1) {
        String[] projection = { "", "" };
        CursorLoader cursorLoader = new CursorLoader(this,
            ProviderMusic.CONTENT_URI, projection, null, null, null);
        Log.e("main activity", "onCreateLoader");
        return cursorLoader;
    }

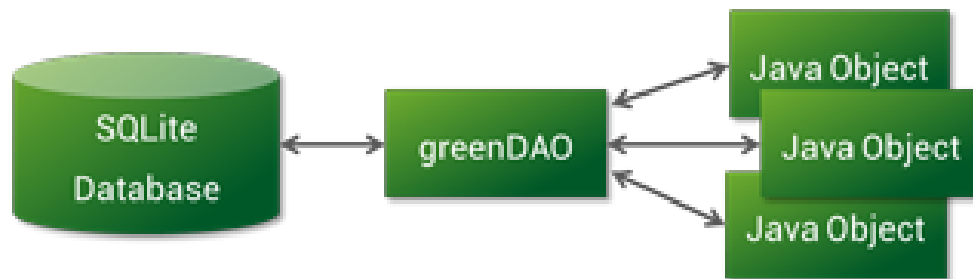
    @Override
    public void onLoadFinished(Loader<Cursor> arg0, Cursor arg1) {
        aa.swapCursor(arg1);
        arg1.SetNotificationURI(ProviderMusic.CONTENT_URI);
        Log.e("main activity", "onLoadFinished");
    }

    @Override
    public void onLoaderReset(Loader<Cursor> arg0) {
        Log.e("main activity", "onLoaderReset");
    }

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        [...]
        aa= new SimpleCursorAdapter(this, R.layout.listeitem, null, new String[]{MusicDB.MUSIC_titre}, new int[]{R.id.titre}, 0 );
        getSupportLoaderManager().initLoader(0, null, this);
    }
}
```

# GreenDAO

- Un projet open source s'occupant du mapping (ORM).
- <http://greendao-orm.com/>



# GreenDAO DAOGenerator

- Exemple facile de mise en place
  - <https://github.com/SureCase/GreenDaoForAndroidStudio>
- Importer le module MyDaoGenerator
- Définir sa table dans MyDAOGenerator

```
public class MyDaoGenerator {  
  
    public static void main(String args[]) throws Exception {  
        Schema schema = new Schema(versionNumber, „javapackage“);  
  
        //Table Eleve  
        Entity eleve = schema.addEntity("Eleve");  
        eleve.addIdProperty();  
        eleve.addStringProperty("Nom");  
        eleve.addStringProperty("Prenom");  
        new DaoGenerator().generateAll(schema, args[0]);  
    }  
}
```

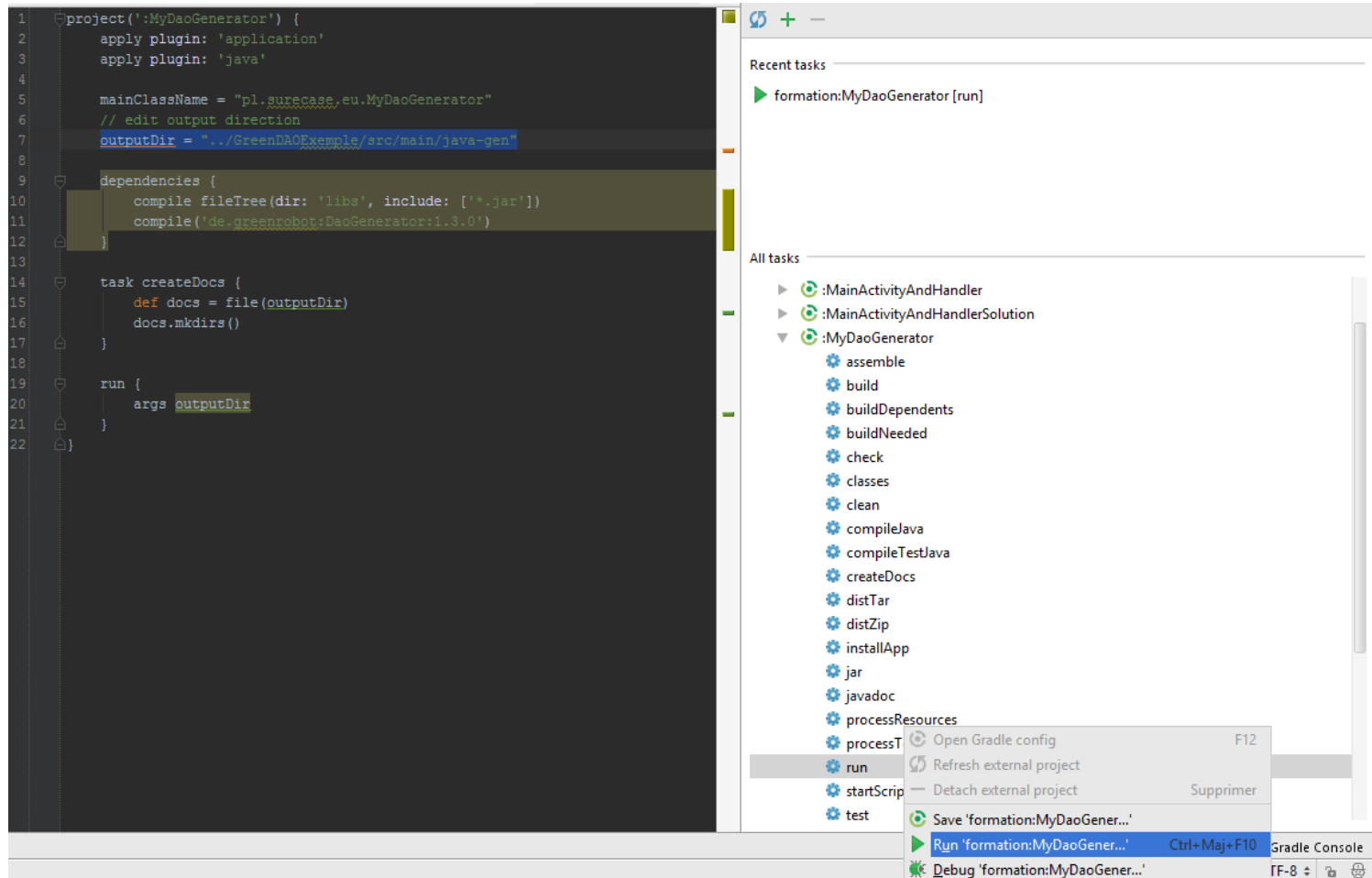


# GreenDAO DAOGenerator

- Modifier le répertoire des fichiers générées dans le gradle
  - `outputDir = "../GreenDAOExemple/src/main/java-gen"`

# GreenDAO DAOGenerator

- Lancer la génération dans la fenêtre de Gradle



# GreenDAO Utilisation

- Les classes du mapping apparaissent dans java-gen
- Dans notre projet ajouter GreenDAO à Gradle

```
dependencies {  
    compile 'de.greenrobot:greendao:1.3.7'  
}
```

- Ajouter le répertoire générer aux sources compilées

```
android {  
    sourceSets {  
        main {  
            java.srcDirs = ['src/main/java', 'src/main/java-gen']  
        }  
    }  
}
```

# GreenDAO Utilisation

```
public class EleveBDDManager {  
  
    public static void insertOrUpdate(Context context, Eleve eleve) {  
        getEleveDao(context).insertOrReplace(eleve);  
    }  
  
    public static void clearEleve(Context context) {  
        getEleveDao(context).deleteAll();  
    }  
  
    public static void deleteEleveWithId(Context context, long id) {  
        getEleveDao(context).delete(getEleveForId(context, id));  
    }  
  
    public static Eleve getEleveForId(Context context, long id) {  
        return getEleveDao(context).load(id);  
    }  
  
    public static ArrayList<Eleve> getAllEleve(Context context) {  
        return (ArrayList<Eleve>) getEleveDao(context).loadAll();  
    }  
  
    private static EleveDao getEleveDao(Context c) {  
        return MyApplication.getInstance().getDaoSession().getEleveDao();  
    }  
}
```

# GreenDAO Ne pas oublier

```
public class MyApplication extends Application {

    private DaoSession daoSession;

    private static MyApplication instance;

    public static MyApplication getInstance() {
        return instance;
    }

    @Override
    public void onCreate() {
        super.onCreate();
        instance = this;
        setupDatabase();
    }

    private void setupDatabase() {
        final DaoMaster.DevOpenHelper helper = new DaoMaster.DevOpenHelper(this, "mytable-db", null);
        final SQLiteDatabase db = helper.getWritableDatabase();
        final DaoMaster daoMaster = new DaoMaster(db);
        daoSession = daoMaster.newSession();
    }

    public DaoSession getDaoSession() {
        return daoSession;
    }

}
```

# TP : GreenDAO

© Matelli sarl 2013

- Rendre persistant la liste d'élève de la listView avec GreenDAO

- Exemple

```
Entity customer = schema.addEntity("Customer");
customer.addIdProperty();
customer.addStringProperty("name").notNull();
```

```
Entity order = schema.addEntity("Order");
order.setTableName("ORDERS"); // "ORDER" is a reserved keyword
order.addIdProperty();
Property orderDate = order.addDateProperty("date").getProperty();
Property customerId =
    order.addLongProperty("customerId").notNull().getProperty();
order.addToOne(customer, customerId);
```

```
ToMany customerToOrders = customer.addToMany(order, customerId);
customerToOrders.setName("orders");
customerToOrders.orderAsc(orderDate);
```

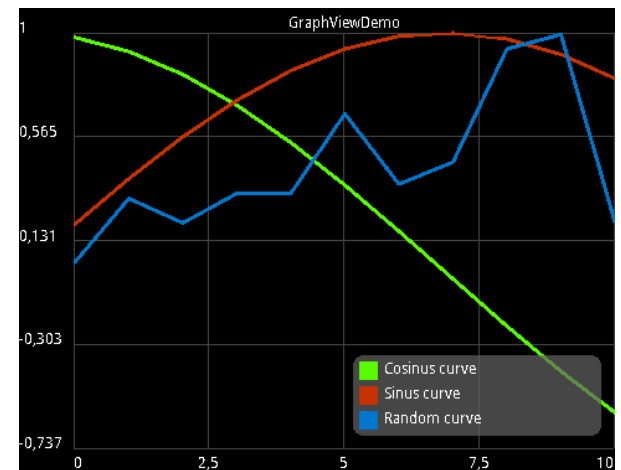
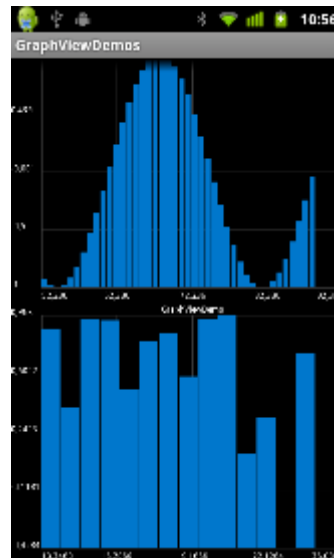
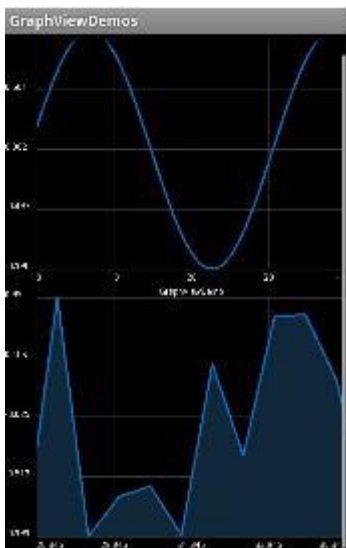
# Librairies Android

- Otto
- GreenDAO
- Picasso
- GraphView
- Zbar
- Simple-Facebook



# GraphView

- Réalisation de graphiques, courbes, diagrammes...
  - <http://android-graphview.org/>
- Utilisation avec Gradle
  - compile 'com.jjoe64:graphview:3.1.3'



# GraphView

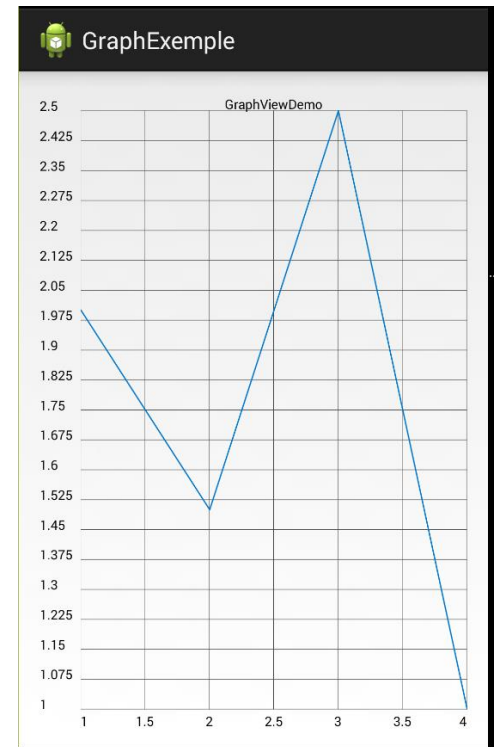
- Utilisation : Création d'un objet de donnée

```
public class GraphViewData implements GraphViewDataInterface {  
  
    private double x, y;  
  
    public GraphViewData(double x, double y) {  
        this.x = x;  
        this.y = y;  
    }  
  
    @Override  
    public double getX() {  
        return x;  
    }  
  
    @Override  
    public double getY() {  
        return y;  
    }  
}
```

# GraphView

- Utilisation : Création du graph

```
private void createGraph1() {  
    // init example series data  
    GraphViewSeries exampleSeries = new GraphViewSeries(new  
    GraphViewData[] {  
        new GraphViewData(1, 2.0d), new GraphViewData(2, 1.5d),  
        new GraphViewData(3, 2.5d), new GraphViewData(4, 1.0d) });  
  
    GraphView graphView = new LineGraphView(this // context  
        , "GraphViewDemo" // heading  
    );  
    graphView.addSeries(exampleSeries); // data  
  
    ((LinearLayout) findViewById(R.id.ll_graph1)).addView(graphView);  
}
```



# TP - GraphView

- Créer un nouveau projet et afficher un 1<sup>er</sup> graph.
- Se servir de la doc sur le site pour créer d'autre version

# Zbar or ZXing

- <http://stackoverflow.com/questions/13268250/android-zxing-vs-zbar>
- ZXing is NOT an open-source library, or at the very best it is only "semi"-open source. You are meant to implement ZXing in tandem with its partner app, which you have to download from the PlayStore separately. This app is the one that actually does the QR Reading; all ZXing does is trigger this particular app. This means that if you are trying to integrate a QR Reader INTO your app and not call a separate one, ZXing is not what you want.
- ZXing is hackable to some extent, however as the versions have gone by the developers have purposely made it harder and harder to hack, because they don't want you to use it as a stand-alone application and bypass theirs. I tinkered with v2.1 for a day, gave up and switched to ZBar, which got me what I wanted in 10 minutes. I could have kicked myself in frustration.
- ZBar is also incredibly fast and accurate, and the tutorial is very extensive; the demo app provided even provides a "Scan Again" button if the scan turns out wrong (which I have yet to see happen). ZBar is highly customizable and doesn't have the tons of red-tape that you have to hack your way through in ZXing; it shows very simply how to get what you want out of your QR/bar code, and sending the result somewhere else is simply just a call to a different Activity.
- The final word: ZBar.

# ZBar

- Lecteur de code barre.
  - <http://zbar.sourceforge.net/>
  - <https://github.com/dm77/barcodescanner>
- Utilisation avec Gradle
  - compile 'me.dm7.barcodescanner:zbar:1.5'

# Zbar - Déclaration

```
public class MainActivity extends Activity implements ZBarScannerView.ResultHandler {

    private FrameLayout cameraPreview;
    private ZBarScannerView mScannerView;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        cameraPreview = (FrameLayout) findViewById(R.id.cameraPreview);
        mScannerView = new ZBarScannerView(this);
        cameraPreview.addView(mScannerView);
    }

    @Override
    protected void onResume() {
        super.onResume();
        mScannerView.setResultHandler(this); // Register ourselves as a handler for scan results.
        mScannerView.startCamera(); // Start camera on resume
    }

    @Override
    public void onPause() {
        super.onPause();
        mScannerView.setResultHandler(null);
        mScannerView.stopCamera(); // Stop camera on pause
    }
}
```

# Zbar - Utilisation

```
/* *****  
****          ZBAR  
* *****/  
  
@Override  
public void handleResult(Result result) {  
    // Do something with the result here  
    tv_resultat.setText("Scan : " + result.getContents() + "\nScan format : " +  
result.getBarcodeFormat());  
}
```



# TP - ZBar

- Créer un lecteur de code barre.

# Simple Facebook

- Version complète
  - <https://developers.facebook.com/docs/android>
- Version allégée du SDK de Facebook
  - <https://github.com/sromku/android-simple-facebook>
- Version très allégée (juste le login)
  - <https://github.com/greenhalolabs/facebooklogin>

# Simple Facebook

- Version complète
  - <https://developers.facebook.com/docs/android>
- Version allégée du SDK de Facebook
  - <https://github.com/sromku/android-simple-facebook>
- Version très allégée (juste le login)
  - <https://github.com/greenhalolabs/facebooklogin>

# Installation

- Importer en module le projet « facebook » du sdk facebook
- Importer en module le projet « simple facebook » de la lib simple-facebook

```
dependencies {  
    compile project(':facebook')  
}
```

- Dans notre projet
  - `compile project(':Simple Facebook')`

# Utilisation

- Générer un facebook\_app\_id sur le site du sdk
- Le mettre dans le manifest

```
<meta-data
    android:name="com.facebook.sdk.ApplicationId"
    android:value="@string/facebook_app_id"/>
```

```
private SimpleFacebook mSimpleFacebook;
//OnCreate
mSimpleFacebook = SimpleFacebook.getInstance(this);
//OnResume
mSimpleFacebook = SimpleFacebook.getInstance(this);
mSimpleFacebook.eventAppLaunched();
//OnActivityResult
mSimpleFacebook.onActivityResult(this, requestCode, resultCode, data);
```

# Utilisation

- Login

```
if (mSimpleFacebook.isLogin()) {  
    mSimpleFacebook.logout(null);  
}  
mSimpleFacebook.login(this);
```

- Logout

```
private void logout() {  
  
    mSimpleFacebook.logout(new OnLogoutListener() {  
        public void onFail(final String reason) {}  
        public void onException(final Throwable throwable) {}  
        public void onThinking() {}  
        public void onLogout() {}  
    });  
}
```

# Utilisation

- PostOnWall

```
final Feed feed = new Feed.Builder().setDescription(message).setName(getString(R.string.app_name))
    .setPicture(« http://url_icone »).setLink(getString(R.string.app_url)).build();
mSimpleFacebook.publish(feed, true, new OnPublishListener() {
    @Override
    public void onFail(final String reason) {}

    @Override
    public void onException(final Throwable throwable) {}

    @Override
    public void onThinking() {}

    @Override
    public void onComplete(final String postId) {}
});
```

# TP - Facebook

- Se loguer avec Facebook
- Poster un message sur son mur.

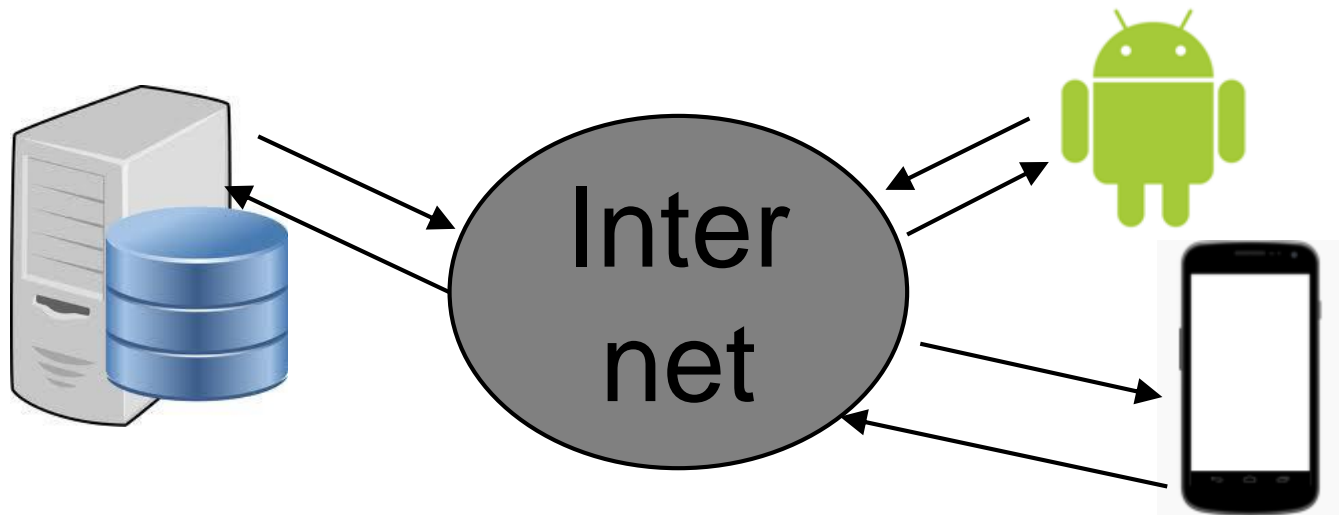


# NOTIFICATION

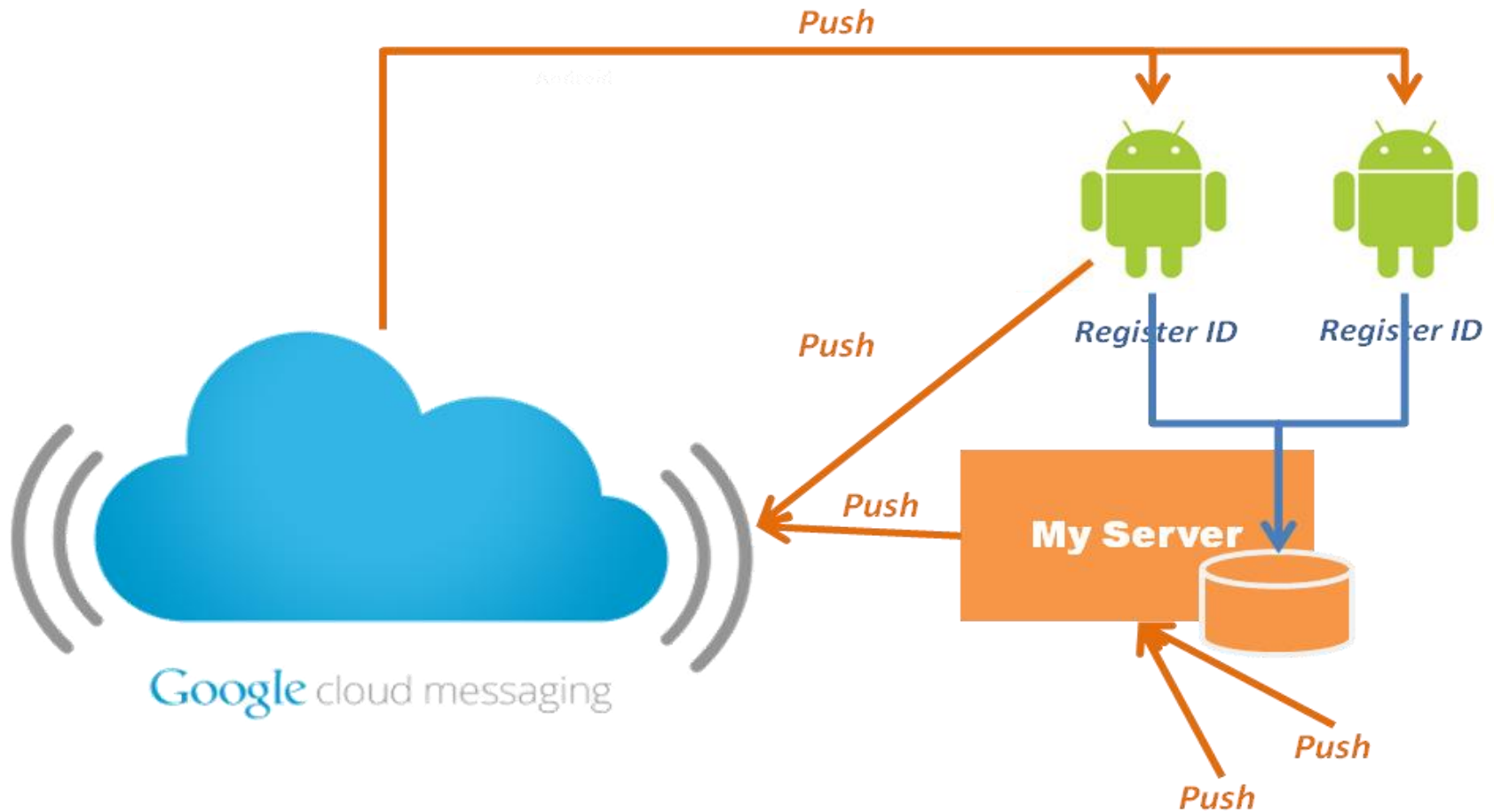
# Notifications

- Les notifications permettent d'alerter l'utilisateur et de garder un historique des dernières notifications dont l'utilisateur n'a pas encore pris connaissance.
- Les notifications sont gérées par un gestionnaire central de notifications. Les applications utilisent la barre de statut du système pour afficher les notifications.
- 2 types de notifications
  - Google cloud messaging
  - NotificationManager

# Google cloud messaging



# Google cloud messaging



# NotificationManager

- Les notifications font partie des services proposés par Android. Nous récupérons ce service en utilisant la méthode `getSystemService` avec le paramètre `Context.NOTIFICATION_SERVICE`.
- La création d'une notification se fait tout d'abord en créant une instance de type `Notification`. Une notification possède trois caractéristiques : une icône, un texte défilant et une heure (qui déterminera quand la notification sera affichée)

# Notifications

- Lorsque l'utilisateur visualisera la notification, il doit pouvoir revenir sur l'activité émettrice de la notification. Pour cela, vous devez utiliser un type spécifique d'Intent, le PendingIntent.

```
PendingIntent pendingIntent = PendingIntent.getActivity(this, 0,
    new Intent(this, Activite.class), 0);
    // Le titre de la notification
String titreNotification = "Titre de ma notification";
    // Le détail de la notification
String texteNotification = "Ma notification personnalisée";
    notification.setLatestEventInfo(this,
titreNotification,
    texteNotification, pendingIntent);
```

# Notifications

```
public class NotificationHelper {

    private static final int NOTIFICATION_REQUEST_CODE = 13; //au hasard pour l'activity
    private static final int NOTIFICATION_ID = 14; //au hasard pour retrouver la notif

    //UNiquemeent pour JellyBean et +
    public static void createNotification(final Context context) {
        final NotificationManager mNotification = (NotificationManager)
            context.getSystemService(Context.NOTIFICATION_SERVICE);
        final Intent launchNotifiactionIntent = new Intent(context, MainActivity.class);
        final PendingIntent pendingIntent = PendingIntent.getActivity(context,
            NOTIFICATION_REQUEST_CODE, launchNotifiactionIntent,
            PendingIntent.FLAG_ONE_SHOT);

        final Notification.Builder builder = new Notification.Builder(context)
            .setWhen(System.currentTimeMillis()).setTicker("Ticker").setSmallIcon(R.drawable.ic_launcher)
            .setContentTitle("ContentTitle").setContentText("ContentText")
            .setContentIntent(pendingIntent);

        mNotification.notify(NOTIFICATION_ID, builder.build());
    }
}
```

# TP : notifications

- Reprendre l'exercice sur le listener de sms et y ajouter une notification en plus du Toast.
- Un clic sur la notification relance l'application.