

Développement mobile :

Android



Présentation

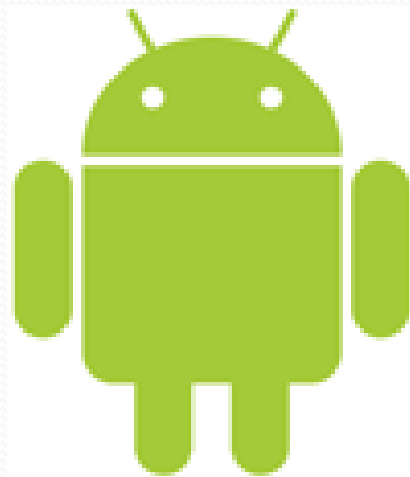
- OS open source pour téléphones mobiles
- Développé par Open Handset Alliance sous l'autorité de Google
- Basé sur un noyau linux
- Séparation couche matérielle et logicielle
- Android également un Framework:
 - Accès au SDK
 - Accès au code source

Versions

- Cupcake (Petit gateau) 1.5 (API 3) : avril 2009
- Donut (Beignet) 1.6 (API 4) : sept. 2009
- Eclair (Eclair) 2.0/2.1 (API 7) : oct. 2009
- Froyo (Yaourt glacé) 2.2.x (API 8) : mai 2010
- Gingerbread (Pain d'épice) 2.3.x (API 10) : déc. 2010
- Honeycomb (Ruche d'abeille) 3.x (API 11 - 12 - 13) : février 2011
- Ice Cream Sandwich (Sandwich à la crème glacée) 4.0.x (API 14 - 15) : octobre 2011
- Jelly Bean (Dragibus) 4. x (API 16 et +) : juin 2012.
- Kit Kat 4.X
- Lollipop : 5.0
- Marshmallow:6.0

Le personnage

- Son nom : Bugdroid
- Issu d'un jeu des années 90 sur atari



Google Play

- Store d'application
- Anciennement Android Market
- Rechercher et télécharger des applications, livres, musiques et films.
- Noter, commenter, désinstaller et mettre à jour les applications déjà installées sur un appareil.
- Publier votre application





Environnement de développement



Environnement développement

- JDK
- SDK Android qui contient:
 - **aapt - Android Asset Packaging Tool** : gestion des *.apk
 - **adb - Android Debug Bridge** : connexion au shell du device ou de l'émulateur
 - **dx - Delvik Cross-Assembler** : conversion de *.class en *.dex pour fonctionner dans la VM DALVIK
 - **ddms - Dalvik Debug Monitor Service** : pour le debug de l'appli

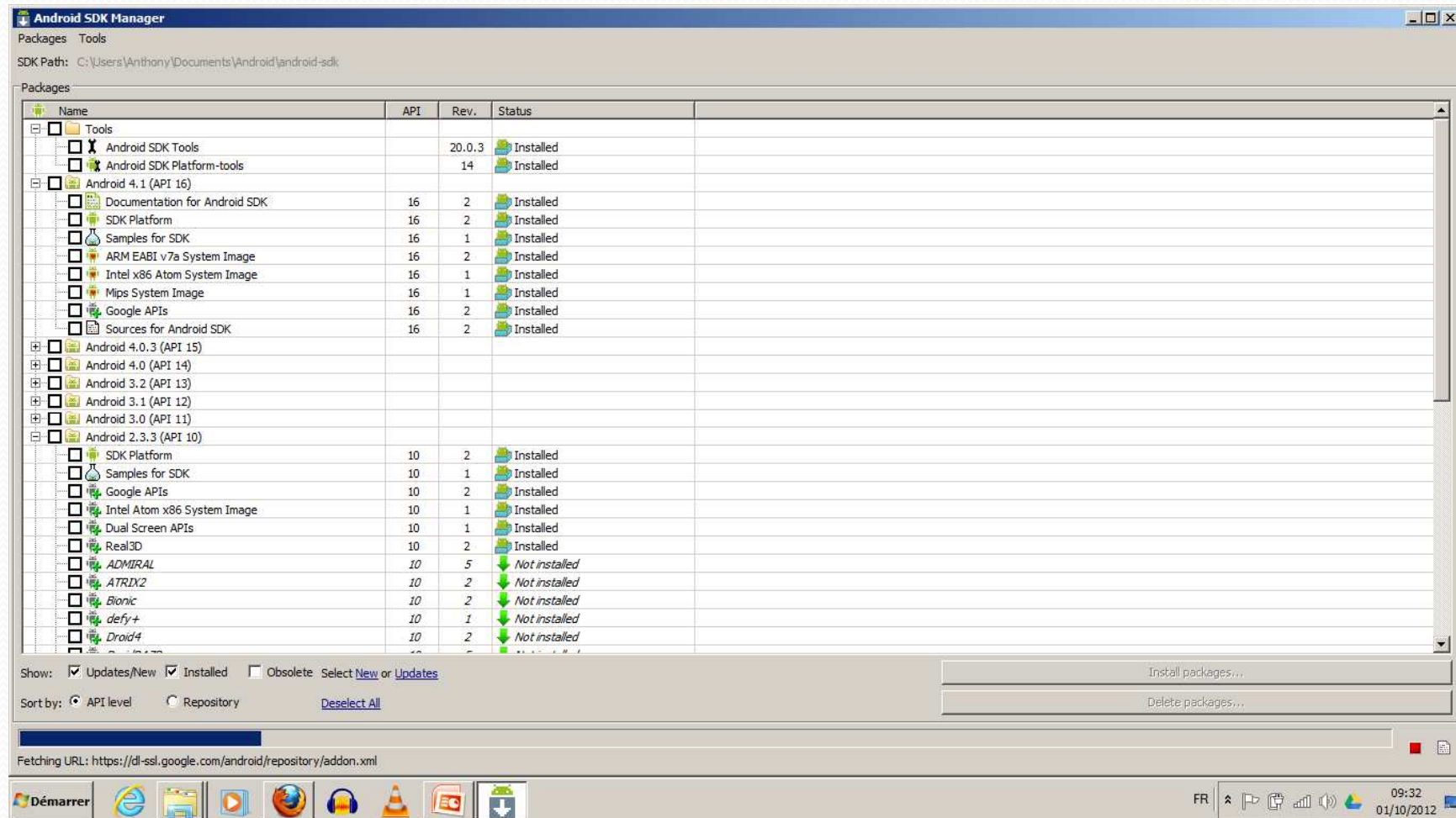
LE DDMS

- Faire des captures d'écrans.
- Voir les threads en cours d'exécution.
- Voir le Logcat.
- Connaître la liste des processus en cours d'exécution sur l'appareil.
- Simuler l'envoi de messages et d'appels.
- Simuler une localisation, etc.

IDE : SDK Manager

- Gestion du SDK:
 - Téléchargement :
<http://developer.android.com/sdk/index.html>
 - Avec **SDK Manager**
 - Permet la gestion des API installées
 - Offre du code sample
 - Les documentations...

SDK Manager



IDE : Eclipse

- S'intègre facilement au SDK
- Fonctionne par rapport à un workspace
- Intégration possible avec plugin ADT



Android studio

- Basé sur IntelliJ IDEA
- Depuis 2013/14
- Fin 2014 -> version stable



Je n'ai pas de smartphone Android!



- Emulateur permet de simuler un téléphone Android
- Machine virtuelle paramétrable:
 - Version Android
 - SD Card
 - Résolution ...
- GenyMotion



Principes Généraux

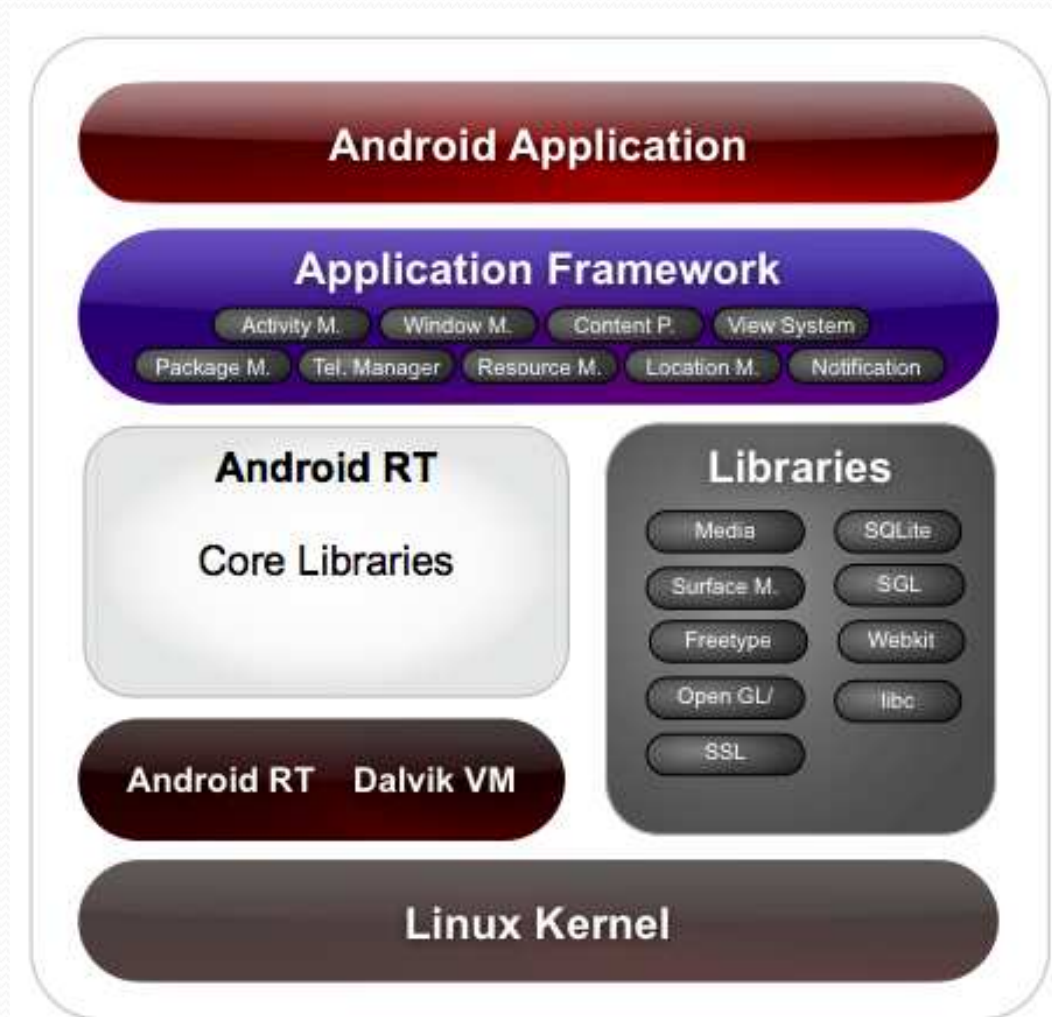
Principes généraux

- Développement sur un système contraint:
 - Vitesse proc, RAM, consommation énergétique
- Comme un site internet, une seule vue
- Succession des pages dans une pile Back Stack
- Retour sur la page précédente avec le bouton retour (comme Web)
- Le NDK permet du développement natif
- JNI permet l'appel de code natif par Java

L'APK

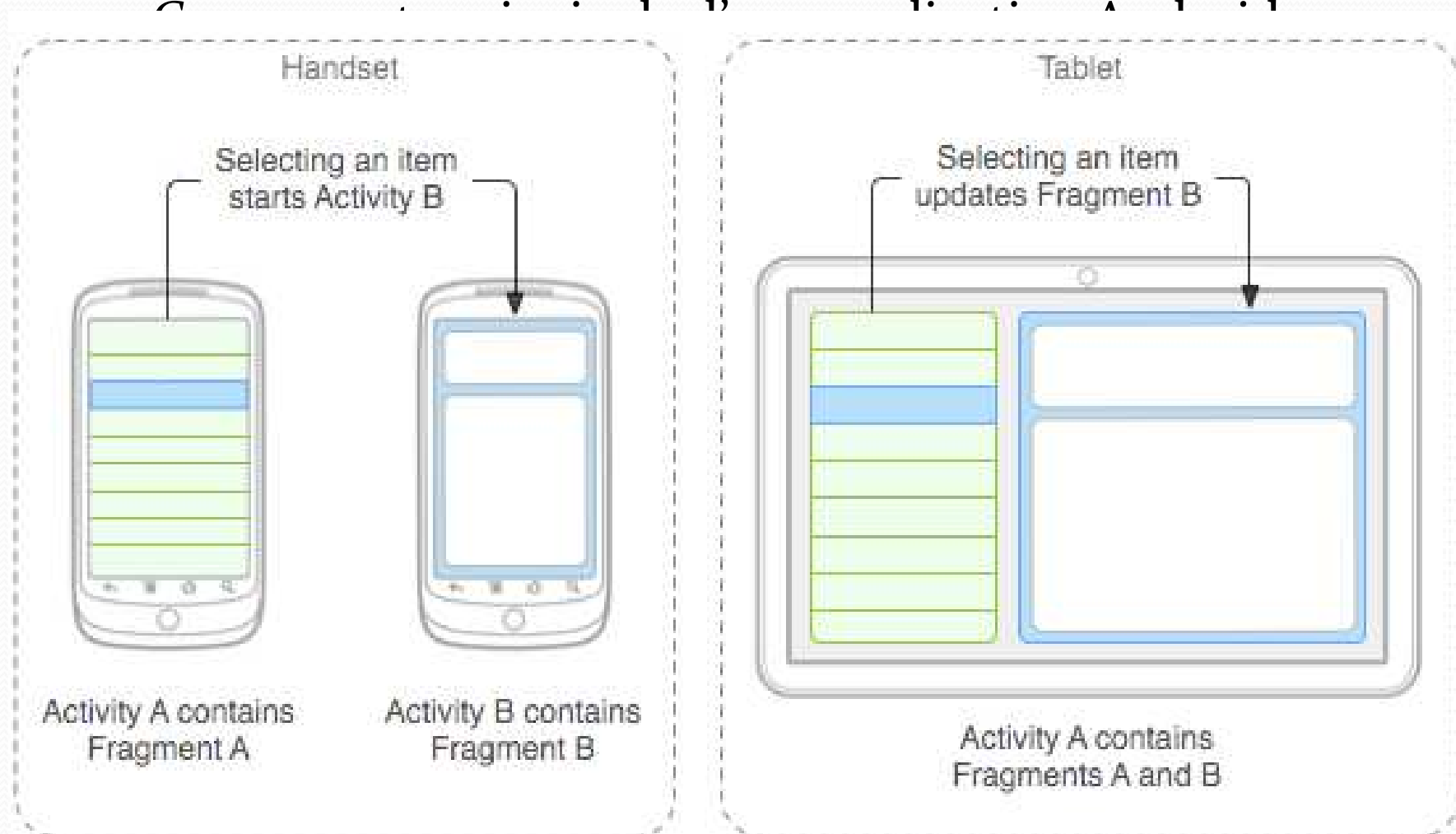
- Binaire représentant l'application
- Il contient:
 - Le code de l'application compilée (.dex).
 - Les ressources.
 - Les assets.
 - Les certificats.
 - Le fichier manifeste.
- On utilise Android Tools pour le générer (cf Eclipse)
- Keystore pour signer les application

Architecture Android



Composants Android 1/6

- Activity:



Composants Android 2/6

- Service
 - Activity sans interface
 - Traitement en tâche de fond
 - Ex: lire musique en tâche de fond
- Broadcast Receiver:
 - Composant sans interface qui réagit à un événement système
 - Ex: permet de savoir quand
 - le tél reçoit un sms
 - Le tél se verrouille...
 - Doit effectuer une tâche courte (sinon service)
- Ne peut pas agir sur l'interface, seulement sur notification

Composants Android 3/6

- Content Provider
 - Permet de partager des données:
 - SQLite
 - Fichiers
 - Web
 - But: d'autres applis accèdent aux données
 - Par défaut:
 - Contacts
 - Agenda
 - Médias, etc

Composants Android 4/6

- Intent
 - Permet aux composants Android de communiquer ensemble
 - Permet l'enchaînement de vues
- Intent-filter
 - Sert à indiquer à une activité, service ou broadcast receiver quels intents ils peuvent implicitement traiter.
 - Utilisation de la balise intent-filter dans `AndroidManifest.xml`
 - Ou création à partir de la classe `IntentFilter`

Composants Android 5/6

```
<activity android:name=".HelloAndroidActivity"
          android:label="@string/app_name" >
  <intent-filter>
    <action android:name="android.intent.action.MAIN" />
    <category android:name="android.intent.category.LAUNCHER" />
  </intent-filter>
</activity>
```

- Signifie:
 - (MAIN) Est la principale de l'application.
 - (LAUNCHER) Appartient à la catégorie Launcher (disponible en raccourci depuis le application d'Android)

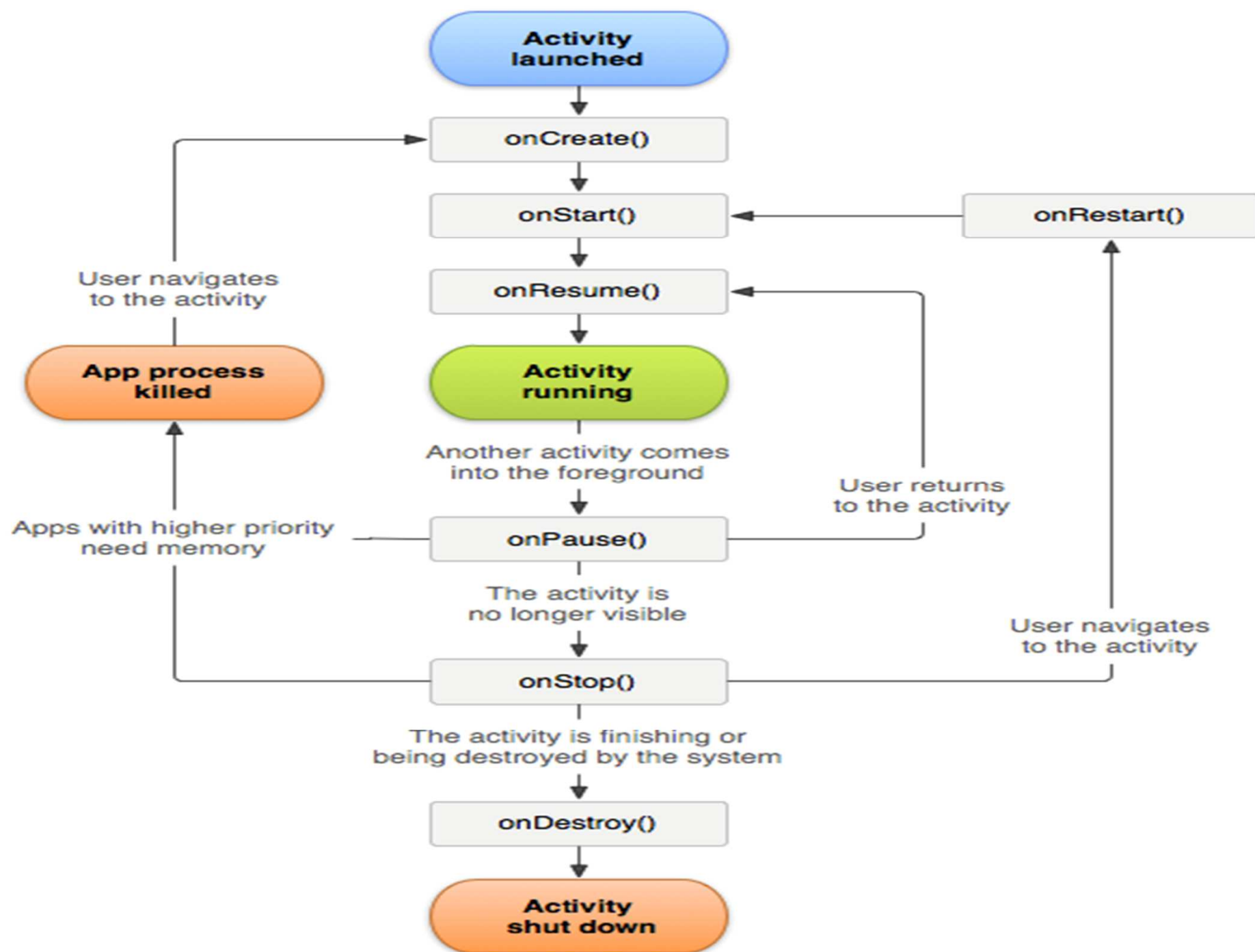
Composants Android 6/6

- PendingIntent
 - Permet à un mécanisme externe (NotificationManager) de déclencher un intent de votre appli
 - Créé à l'aide de:
 - **getActivity(Context, int, Intent, int)**
 - Génère un pendingIntent pointant vers une activité.
 - **getBroadcast(Context, int, Intent, int)**
 - Génère un pendingIntent déclenchant un Broadcast Receiver.
 - **getService(Context, int, Intent, int)**
 - Génère un pendingIntent qui exécutera un service.
 - Ex: Notification à la réception d'un message

La classe Application

- Chaque appli possède une classe Application
- A déclarer dans le manifest
- Instanciée au lancement de l'appli
- Reste instanciée tout au long de la vie de l'appli

```
<application android:icon="@drawable/ic_launcher"  
             android:label="@string/app_name"  
             android:name=".MyApplication">
```



Contexte d'une application

- État courant d'une application et informations sur son environnement.
- Sert à récupérer des objets transmis par d'autres parties de votre application.
- Récupération du contexte:
 - **getApplicationContext()** : Contexte de l'application.
 - **getContext()** : Contexte de la vue courante
 - **getBaseContext()** : Contexte défini avec **setBaseContext()**.
 - **this** : Quand on est dans une classe héritant de **Context**
 - Activity étend Context

Manifeste

- **AndroidManifest.xml**
- À la racine de tout projet Android
- Description complète de l'application:
 - Composantes
 - Activités
 - Services
 - Permissions
 - Fournisseurs de contenu, etc.

Attributs balise manifest

```
<manifest  
  xmlns:android="http://schemas.android.com/apk/res/android"  
    package="com.eni.android.hello"  
    android:versionCode="1"  
    android:versionName="1.0" >  
  ...  
</manifest>
```

- La déclaration xmlns (obligatoire).
- Le package de votre application
- Le code de la version de votre application. (+1)
- Le nom de la version (visible sur le Market).

Nœuds balise manifest

- **permissions**
- **instrumentations**: framework de test
- **uses-sdk** : minSDKVersion, maxSDKVersion , targetSDKVersion
- **uses-configuration**: spécificités de navigation supportées (trackball, stylet....)
- **uses-feature** : spécificités matérielles (audio, bluetooth)
- **supports-screens** : dimensions d'écran supportées
- **application** : cœur et contenu de l'application (activité, service, etc.).

Permissions

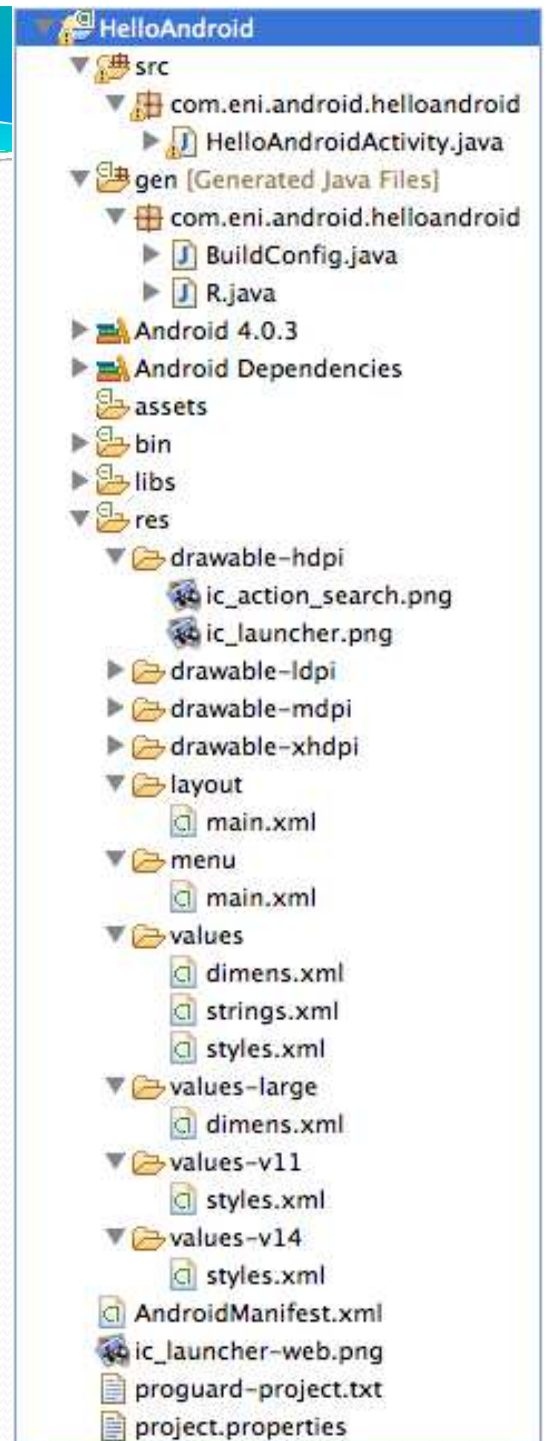
- Correspond au droit d'accès à une fonctionnalité ou une donnée
- Par défaut : Aucune permission
- But : connaître permissions nécessaire avant install
- Demander l'accès internet pour l'appli:
 - Ajout dans le manifeste :

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

- Erreurs de permissions à l'exécution : SecurityException
- Possibilité de déclarer une permission personnalisée

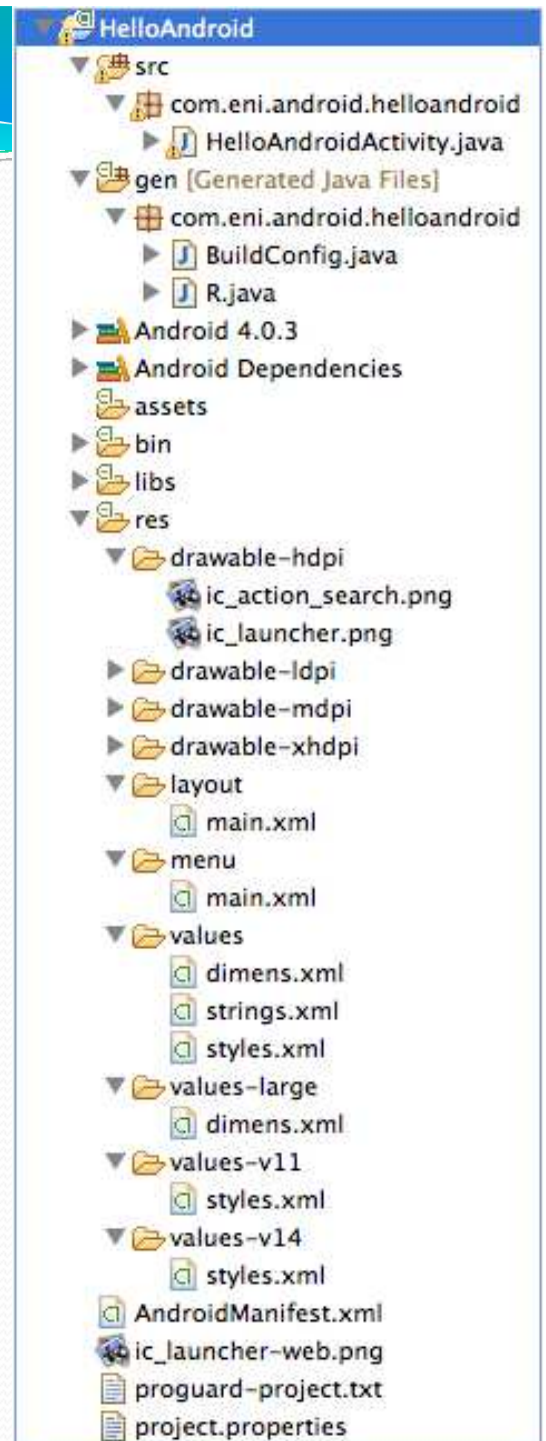
Architecture d'un projet 1/3

- **src** : sources de votre application (Activité, Service, Code métier, etc.).
- **gen** : Résultat de la génération des fichiers du dossier **res** . + **BuildConfig** permettant de gérer l'affichage du log en mode debug et développement
- **assets** : données ou fichiers à utiliser dans l'application (en lecture seule)
- **bin** : binaires et *.class générés.
- **libs** : contient les différentes bibliothèques utilisées par l'application.



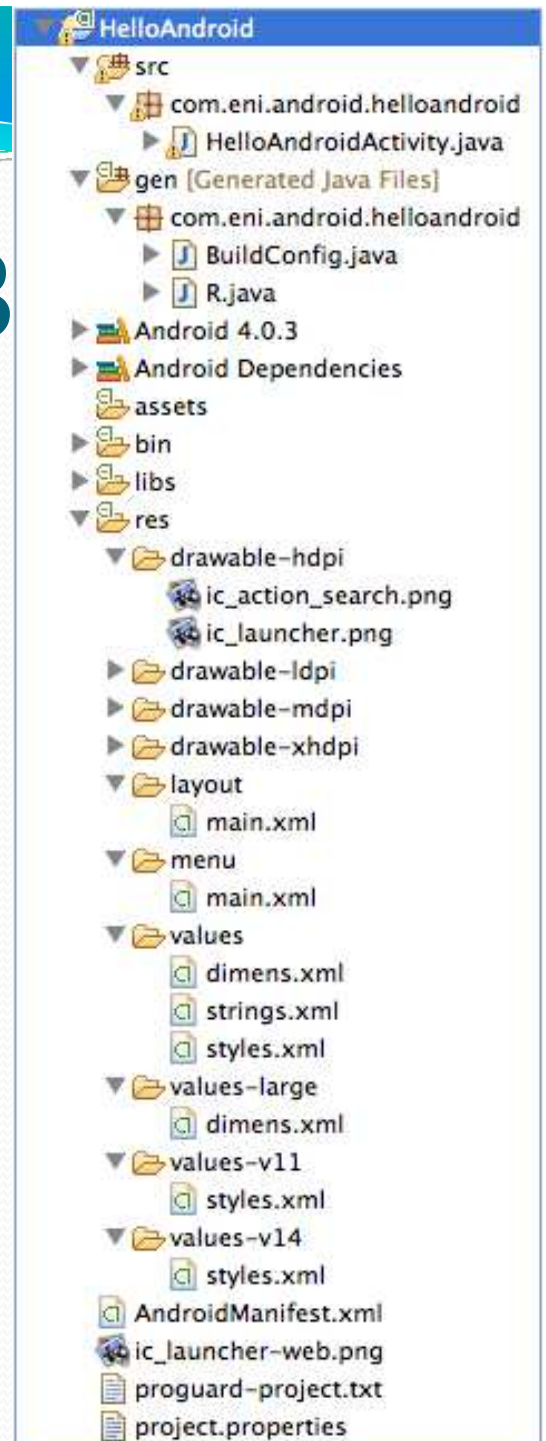
Architecture d'un projet 2/3

- **res** : Ressources utilisées dans l'application, précompilées et ajoutées à **R.java**
- **drawable-xxx** : images
- **layout-xxx** : Les vues
- **values-xxx** : données ou valeurs (strings, arrays)
- **menus** : Barres d'actions et menus
- **raw-xxx** : Fichiers non compilables (audio, vidéo)



Architecture d'un projet 3

- **AndroidManifest.xml**
- **proguard.cfg** : offuscation et d'optimisation du code
- **project.properties** : Informations sur le projet (version, bibliothèques référencées).



Exemple manifeste

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.velibs.parisiens.andoid"
    android:versionCode="1"
    android:versionName="1.0" >
    <uses-sdk
        android:minSdkVersion="8"
        android:targetSdkVersion="15" />
    <uses-permission android:name="android.permission.INTERNET" />
    <uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION"/>
    <uses-permission android:name="android.permission.ACCESS_FINE_LOCATION"/>
    <uses-permission android:name="android.permission.ACCESS_LOCATION_EXTRA_COMMANDS"/>
    <application
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name"
        android:theme="@style/AppTheme" >
        <uses-library android:name="com.google.android.maps" />
        <activity
            android:name=".ArrondissementsActivity"
            android:label="@string/title_activity_arrondissements" >
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
        <activity
            android:name=".StationsActivity"
            android:label="@string/title_activity_stations" >
        </activity>
        <activity
            android:name=".MapActivity"
            android:label="@string/title_activity_map" >
        </activity>
    </application>
</manifest>
```

Debug : Logcat

- Indispensable pour le développement
- Permet de visualiser tous les messages des différentes applications de l'appareil
- Dans le code, envoyer message vers le Logcat:
 - **d (Debug)**
 - **e (Erreur)**
 - **i (Information)**
 - **v (Verbose)**
 - **w (Warning)**
 - **wtf (What a Terrible Failure)**

```
Log.e(String tag, String message);
```


Lint

- Pour spécifier des règles à appliquer aux projets Android
- Important pendant le développement :
 - Vérifie les bonnes pratiques
- Comme tout bon langage, Android permet les tests unitaires

Création d'UI

- Statique : directement en Xml
- Dynamique : en Java
- Combinaison des 2
- Une UI se compose :
 - 1 ou n fichiers Xml
 - La classe Java (Activity) associée
- View est la classe mère de tous les objets UI

Identifiants de composants UI

- Chaque composant a un identifiant unique dans l'application

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

- Accessible depuis Java:

```
R.id.nom_identifiant
```

- Une fois le XML créé, travailler sur le *.java

La classe associée

- Hérite de **Activity**
- Surcharge au minimum **onCreate**
- Est liée à son xml par **setContentView**
- Pour un home.xml :

```
@Override  
public void onCreate(Bundle savedInstanceState) {  
    super.onCreate(savedInstanceState);  
    setContentView(R.layout.home);  
}
```

- Chaque activity doit apparaître dans le manifeste

Dimension des éléments UI

- Pour un layout ou composant
- Déclaration de la taille:
 - **match_parent** (égale à celle de l'élément parent)
 - **wrap_content** (égale à celle de son contenu+espaces internes)
 - **en spécifiant une valeur fixe**
- Les tailles doivent être en dp (density-independent pixels)



Les Layouts

- Servent de conteneur aux composants d'une vue.
- Héritent de ViewGroup qui hérite de View

FrameLayout

- Le plus simple
- Espace que l'on remplit avec des éléments
- Par défaut, l'élément se place en haut à gauche
- Possibilité de changer Gravity
- Utilisé pour superposition d'éléments



LinearLayout

- Permet d'aligner des éléments (vert. ou horiz.)
- Les attributs:
 - Orientation
 - La gravité des éléments.
 - Le poids des éléments.

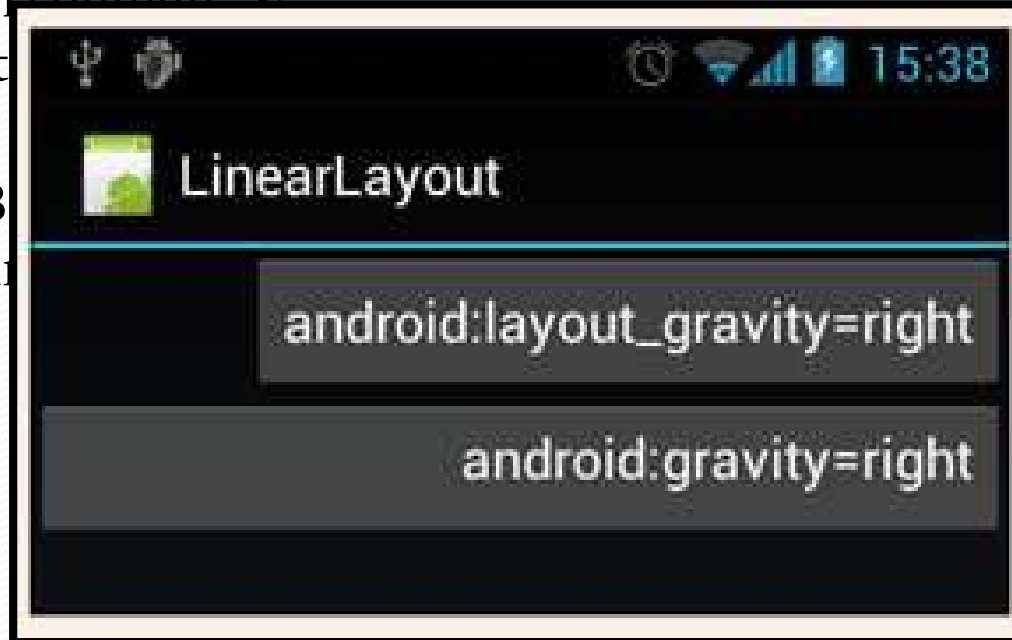


LinearLayout : Orientation

- Par défaut Horizontale
- Attribut **android:orientation**

Positionnement dans LinearLayout

- Layout vertical
- Button A : largeur égale son contenu et à droite layout
- Button B placé à droite bouton



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

```
    <Button
        android:text="A"
        android:layout_gravity="right"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:gravity="center" />
    <Button
        android:text="B"
        android:layout_gravity="right"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:gravity="center" />
</LinearLayout>
```

```
    <Button
        android:text="A"
        android:layout_gravity="right"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:gravity="center" />
    <Button
        android:text="B"
        android:layout_gravity="right"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:gravity="center" />
</LinearLayout>
```

```
    <Button
        android:text="A"
        android:layout_gravity="right"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:gravity="center" />
    <Button
        android:text="B"
        android:layout_gravity="right"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:gravity="center" />
</LinearLayout>
```

```
    <Button
        android:text="@string/gravity"
        android:gravity="right" />
</LinearLayout>
```

```
</LinearLayout>
```



Le poids d'un élément

- Sert à indiquer à un élément l'espace qu'il peut occuper.
- Plus le poids est important, plus :
 - Il peut s'allonger
 - Donc plus il occupe l'espace disponible.

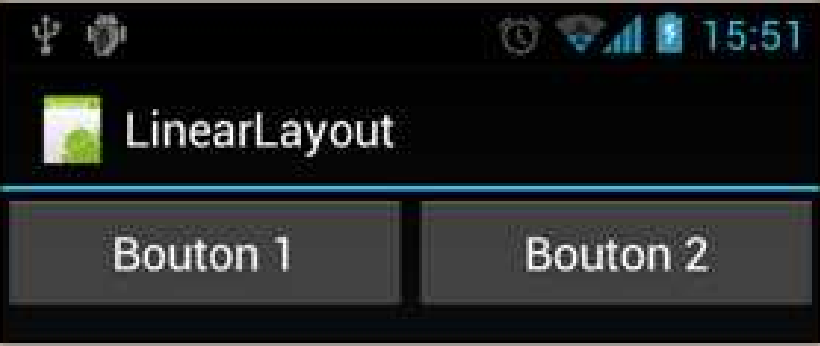
Exemple de poids (=)

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

  <Button
    android
    android
    android
    android:layout_weight="1" />

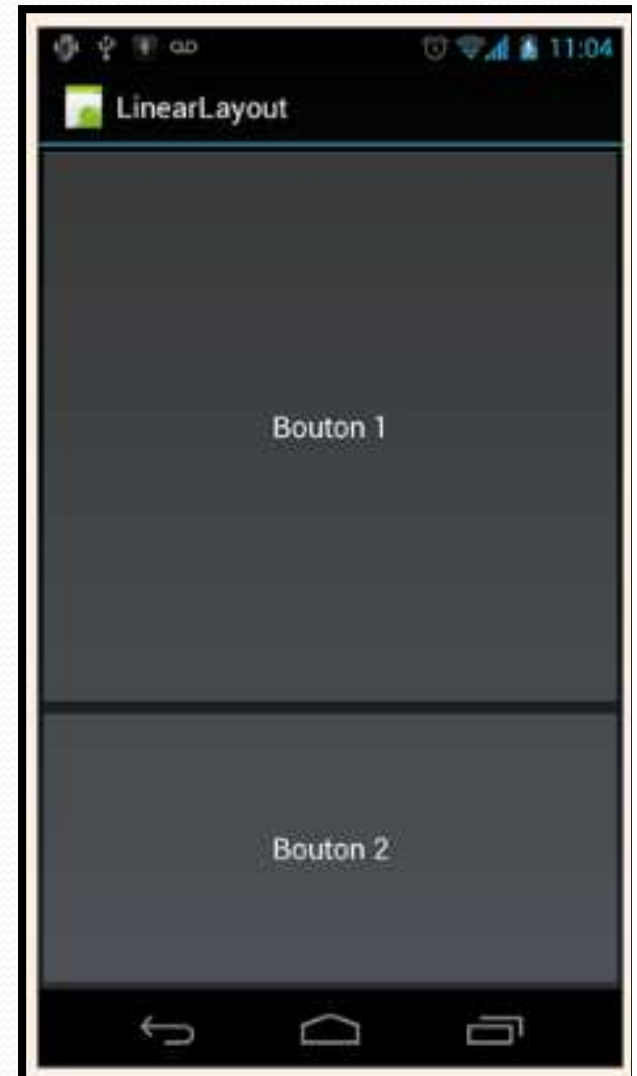
    <Button
      android:layout_width="0px"
      android:layout_height="wrap_content"
      android:text="@string/btn2"
      android:layout_weight="1" />

</LinearLayout>
```

A screenshot of an Android emulator interface. At the top, there's a status bar with icons for signal, Wi-Fi, battery, and the time 15:51. Below the status bar, the title bar of the application window says "LinearLayout" with a small green icon on the left. The main content area of the emulator shows two buttons side-by-side, labeled "Bouton 1" and "Bouton 2". The buttons are dark gray with white text. The text "Bouton 1" is slightly larger than "Bouton 2", indicating they share the available space equally due to their equal weight values.

Exemple de poids (<>)

- Layout vertical
- 2 boutons
- Bouton1 à un `layout_weight` 2 fois supérieur à Bouton2



TableLayout

- Positionne tous ses fils en ligne et colonne
- Comparable à un tableau
- 1 Cellule peut être vide
- 1 cellule ne peut pas prendre la place de plrs cellules
- TableRow est une ligne du tableau

Exemple TableRow

```
<?xml version="1.0" encoding="utf-8"?>
<TableLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent" >
    <TableRow>
        <TextView
            android:padding="3dp"
            android:text="@string/line1_column1" />
        <TextView
            android:padding="3dp"
            android:text="@string/line1_column2" />
    </TableRow>
    <TableRow>
        <TextView
            android:padding="3dp"
            android:text="@string/line2_column1" />
        <TextView
            android:padding="3dp"
            android:text="@string/line2_column2" />
    </TableRow>
</TableLayout>
```

A screenshot of an Android emulator interface. At the top, there is a status bar with icons for signal, Wi-Fi, battery, and the time 16:01. Below the status bar is a title bar with a green icon and the text "TableLayout". The main content area displays a table with two rows. The first row contains the text "Ligne 1 - Colonne 1" and "Ligne 1 - Colonne 2". The second row contains the text "Ligne 2 - Colonne 1" and "Ligne 2 - Colonne 2". The text is white on a dark background.



Padding et margin

- Padding spécifie l'espacement interne d'un élément
- Padding impose l'espace entre les bordures d'un élément et son contenu
- **Margin** spécifie l'espacement externe d'un élément.



RelativeLayout

- Permet de placer les éléments les uns en fonction des autres:
 - Soit en fonction de son conteneur
 - Soit en fonction d'un autre élément



Relatif au bord du conteneur

- **android:layout_alignParentTop** : aligner avec le haut du conteneur.
- **android:layout_alignParentBottom** : aligner avec le bas du conteneur.
- **android:layout_alignParentLeft** : aligner avec le côté gauche du conteneur.
- **android:layout_alignParentRight** : aligner avec le côté droit du conteneur.
- Combinaisons possibles

Relatif aux autres éléments

- **android:layout_above** : au dessus.
- **android:layout_below** : en dessous
- **android:layout_toLeftOf** : à gauche
- **android:layout_toRightOf** à droite
- **android:layout_alignTop** : les parties hautes alignées
- **android:layout_alignBottom** : les parties basses alignées
- **android:layout_alignLeft** : les parties gauches alignées
- **android:layout_alignRight** : les parties droites alignées
- **android:layout_alignBaseline** : les lignes de bases align.
- Déclaration d'un élt possible en même temps:
 - `android:layout_below="@+id/connect" />`

Exemple RelativeLayout



```
<?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"
    android:orientation="vertical" >

    <EditText
        android:id="@+id/nomEdit"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:layout_alignParentRight="true"
        android:layout_alignParentTop="true"
        android:hint="@string/email" />

    <EditText
        android:id="@+id/prenomEdit"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:layout_alignParentRight="true"
        android:layout_below="@id/nomEdit"
        android:hint="@string/pass" />

    <Button
        android:id="@+id/valider"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignRight="@id/prenomEdit"
        android:layout_below="@id/prenomEdit"
        android:text="@string/ok" />

    <Button
        android:id="@+id/annuler"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignTop="@id/valider"
        android:layout_toLeftOf="@id/valider"
        android:text="@string/cancel" />

</RelativeLayout>
```

GridLayout

- Dispo depuis Ice Cream Sandwich (API 14)
- Comme TableLayout avec **span** en plus





Les ressources

- Important : externalisation des ressources
- Facilite maintenance et màj
- Nom ressource ne peut contenir que:
 - Des minuscules
 - Des chiffres
 - Un point
 - Un underscore

Spécificités de ressources

- Indiqués par un tiret:
 - Nom de code d'un pays, Langue,
 - Largeur minimum de l'écran **sw (<dimension>dp)**
 - Largeur de l'écran (**w<dimension>dp**)
 - Hauteur de l'écran (**h<dimension>dp**)
 - Taille de l'écran (**small, medium, large, xlarge**)
 - Orientation de l'écran
 - Version d'Android (v14, v11, v8...)...
- **getResources** permet d'accéder aux ressources depuis une activité



Drawable

- Gestion des images
- Répertoire selon résolution

Values

- Dédié au stockage des différentes valeurs (constantes)
- Les différentes values:
 - Chaînes de caractères
 - Accessible depuis Java avec : `R.string.nom_string`
 - Gestion des pluriels (plurals)
 - Balises html utilisables sur les strings (`En gras`)
 - Tableaux dans `arrays.xml`
 - Dimensions dans `dimens.xml`
 - Couleur

Les éléments indispensables

- TextView (Label)
- EditText (Zone de saisie)
- Button (Bouton)
- Checkbox (Case à cocher)
- ImageView (pour placer une image)
- ...

Gestion du clic

- 2 façons de faire:
 - Soit l'Activity reçoit l'événement
 - Soit les éléments prennent l'événement

Clic reçu par bouton

```
Button btn1 = (Button) findViewById(R.id.btn1);  
btn1.setOnClickListener(new OnClickListener() {  
    @Override  
    public void onClick(View v) {  
        Log.v("ClickListener", "Interaction avec le bouton 1");  
    }  
});
```

1. On récupère l'instance de Button de la vue avec *findViewById*
2. On override la méthode *onClick* de l'interface *OnClickListener*.
3. On affecte notre classe anonyme au Listener de clic.

Clic reçu par l'Activity

- L'Activity implémente l'interface OnClickListener
- D'abord

```
Button btn1 = (Button) findViewById(R.id.btn1);  
btn1.setOnClickListener(this);
```

- Reste à implémenter le onClick dans notre Activity

```
@Override  
public void onClick(View v) {  
    switch (v.getId()) {  
        case R.id.btn1:  
            Log.v("ClickListener", "Interaction avec le bouton 1");  
            break;  
        default:  
            break;  
    }  
}
```

Communication entre Activity

- Navigation d'une Activity à une autre par les intents
- Application découpée en Activity
- Pour lancer une Activity:

```
public void startActivity (Intent intent)
```

- Besoin d'un intent pour l'appel de startActivity

```
Intent (Context packageContext, Class<?> activityToLaunch)
```


Exemple Intent

- Pour passer d'une Activity A à une Activity B:

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

Passage de données : Les extras

- A l'aide des extras portées par les intents
- Couples Clé/valeur : Bundle
- Méthode uniquement pour les types primitifs Java
- Pour déposer un couple dans le Bundle:
 - `monIntent.putExtra(uneClé , laDonnées) ;`
 - `uneClé` est de type `String`
 - `laDonnées` de type primitif
- Pour récupérer le Bundle dans l'activité destination:
 - `LaDonnées = getIntent().getXXXXXXExtra(uneClé);`

Activity qui retourne

- But récupération d'une donnée lorsque la 2nd activity se ferme.
- **startActivityForResult(Intent, int) :**
 - Intent : l'intent à démarrer
 - Int : code de résultat:
 - si <0 -> aucun résultat donc idem startActivity
- Surcharger **onActivityResult(int1, int2, intent)**
 - Int1 : identifie provenance du résultat (cf. préc)
 - Int2 : Spécifie le succès du résultat
 - renseigné par la cible avec setResult [RESULT_OK, RESULT_CANCELED + personnalisées]
 - Intent contient les données du résultat

Passage d'objets : Parcelable 1/4

```
public class User {  
    private String login;  
    private String password;  
  
    public User(String login, String password) {  
        super();  
        this.login = login;  
        this.password = password;  
    }  
    public String getLogin() {  
        return login;  
    }  
  
    public void setLogin(String login) {  
        this.login = login;  
    }  
  
    public String getPassword() {  
        return password;  
    }  
  
    public void setPassword(String password) {  
        this.password = password;  
    }  
}
```

Passage d'objets : Parcelable 2/4

- Objet Parcelable implements Parcelable
 - Override de **describeContents**
 - décrit le nb d'objets non primitifs
 - Override de **writeToParcel**
 - Écrit le contenu de l'objet dans le parcel (avec writeXXX)

```
@Override
public int describeContents() {
    return 0;
}

@Override
public void writeToParcel(Parcel dest, int flags) {
    dest.writeString(login);
    dest.writeString(password);
}
```

Passage d'objets : Parcelable 3/4

- Création d'un Objet CREATOR afin de pouvoir recréer une instance à partir du Parcel

```
public static final Parcelable.Creator<User> CREATOR = new
Parcelable.Creator<User>() {

    @Override
    public User createFromParcel(Parcel source) {
        return new User(source);
    }

    @Override
    public User[] newArray(int size) {
        return new User[size];
    }
};
```


Passage d'objets : Parcelable 4/4

- Constructeur de User qui prend un Parcel en paramètre

```
public User(Parcel source) {  
    this.login = source.readString();  
    this.password = source.readString();  
}
```

- Exemple de l'Activity A à B:

- Dans A

```
Intent intent = new Intent(A.this, B.class);  
User user = new User(login.getText().toString(), password.getText().toString());  
intent.putExtra(EXTRA_USER, user);  
startActivity(intent);
```

- Dans B

```
Intent intent = getIntent();  
User user = intent.getParcelableExtra(EXTRA_USER);  
  
login = (TextView) findViewById(R.id.userLogin);  
login.setText(user.getLogin());  
password = (TextView) findViewById(R.id.userPassword);  
password.setText(user.getPassword());
```

The
End