Technologies mobiles

Olivier Levitt

23 janvier 2013

Sommaire

Contents

Planning

- 30 janvier : 3h de cours, 3h de TP
- 6 février : 3h de cours
- 13 février : 6h de TP
- Validation des sujets de projet avant le 20 février
- 20 février : 6h de TP dédiées au projet
- ? mars : Soutenance du projet

Evaluation

- Projet : création d'une application
- Groupe de 2
- Sujet "libre"
- 6h de TP dédiées au projet + travail personnel
- Soutenance / Présentation de l'application

Contents

Contexte et objectifs

- Smartphones, tablettes et assimilés (TV, montre, autoradio, consoles de jeu . . .)
- Dev d'application, pas de dev de la plateforme
- 1ère partie : le dev mobile en général
- 2ème partie : application sous android

Contents

Des appareils suréquipés

- Téléphonie (SMS, MMS, appels)
- Internet (GPRS, EDGE, 3G, 4G, WIFI)
- Réseaux locaux (Bluetooth, réseaux adhoc, NFC)
- Capteurs (Luminosité, proximité)
- Localisation (GPS, triangulation, SSID wifi)
- Notifications (Vibreur, haut-parleurs, LED)
- Photo / vidéo
- Stockage de données (Mémoire flash, SD externe, SQLite)
- Interactions (Ecran tactile, gestures, boutons physique)
- Et encore d'autres ...

Et des API pour utiliser tout ça !

Des contraintes techniques importantes

- Processeur
- Mémoire RAM
- Stockage de données
- Gestion de la batterie
- Stabilité et débit de la connexion internet
- Cycle de vie de l'application
- Taille d'écran
- Inputs atypiques (clavier virtuel, gestures, peu de boutons . . .)

Contraintes à garder en tête en permanence.

La fragmentation

Une application publiée sur le google playstore cible plus de 2400 appareils différents !

- “Write once, run everywhere” ?
- Comment tester / débugger pour tous ces appareils ?
- Eviter de géner l’utilisateur (versions HD, appareils non compatibles)
- S’adapter quand une fonctionnalité n’est pas disponible

La fragmentation, taille d'écran

Comment gérer toutes les tailles d'écran ?

- Montres connectées : de 1 à 2 pouces
- Smartphones lowcost : 3 pouces (Galaxy pocket, galaxy Y)
- Smartphones high-end : 4 à 5 pouces (IPhone 5, HTC 8X, nexus 4)
- Phablets : 5 à 6 pouces (Galaxy note, HTC butterfly)
- Tablettes : 7 pouces (Nexus 7, IPad mini), 8 pouces (Archos 80g9), 10 pouces (Nexus 10, IPad)

De nombreuses autres sources de fragmentation

- Versions de l'OS
- Résolutions d'écran
- Elements hardware présents
- Puissance
- Modifications constructeur / "rom custom"
- ...

Des Ecosystèmes forts

- Obligation d'utiliser le SDK fourni
- Suivre les guidelines
- Restrictions liées à la plateforme
- Utilisation des services de la plateforme
- Processus de déploiement des applications
- Règles des "store" (validation, monétisation . . .)

Contents

iOS

- Soutenu par Apple
- Présenté le 9 janvier 2007
- Dédié aux produits apple (iPhone, iPad, iPod)
- 400 millions d'appareils (Septembre 2012)
- Programmation en objective-C, sur mac OS X uniquement
- Appstore : validation + 100$ / an

Android

- Soutenu par Google
- 1.0 en septembre 2008, 1.5 en avril 2009
- Plus de 2400 appareils officiellement supportés, + de 50 constructeurs
- 480 millions d'appareils activés (Septembre 2012)
- Programmation en JAVA, sur windows / OS X / linux
- Open-source
- Google playstore : pas de validation + 25$

Windows phone 8

- Soutenu par Microsoft
- Présentation au public le 29 octobre 2012
- Successeur de windows phone 7 (logique)
- Plusieurs constructeurs dont Nokia, HTC et Samsung
- Programmation en C# sur windows
- Windows marketplace : validation + 100$ / an

Blackberry 10

- Soutenu par RIM (Research in motion)
- Présentation au public le 30 janvier 2012 (!)
- Appareils produits par RIM
- C / C++, HTML5, Adobe AIR, Portage android
- Blackberry appworld : validation + gratuit

Ubuntu for phones

- Soutenu par Canonical
- Teaser le 2 janvier 2012, testable sur galaxy nexus fin février
- Premiers ubuntu phones promis pour début 2014
- Facilement utilisable sur les téléphones android ?
- HTML5, C/C++ + QML
- Open-source
- Peu d'infos sur le store

Contents

Les marque-page

- www.frandroid.com (actu FR)
- www.androidpolice.com (actu EN)
- www.androidcentral.com (actu EN)
- www.d.android.com (la bible EN)
- www.stackoverflow (Q/A EN)
- #android et #android-dev sur freenode (chat irc EN)
- www.breizhjug.org et www.paug.fr (communautés FR)
- www.google.fr (réservoir à tutoriels)

Bien commencer

La programmation android fait partie des plus accessibles :

- Des (bonnes) bases de programmation en JAVA
- Un ordinateur (Windows, Linux, Mac OS X)
- Un appareil android (conseillé, l'émulateur étant . . . moyen)

C'est tout !

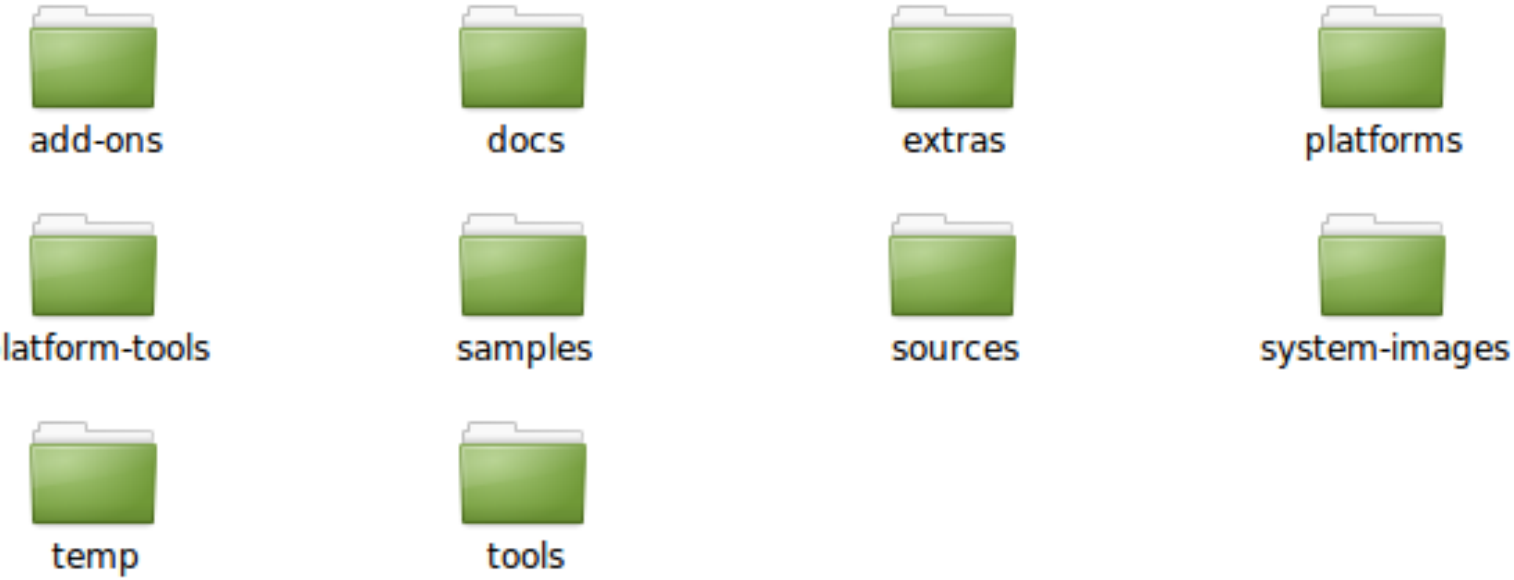
Les niveaux d'API

Version	Nom	API level	Distribution	Cumulé
1.5	Cupcake	3	0%	0%
1.6	Donut	4	0.2%	0.2%
2.1	Eclair	7	2.4%	2.6%
2.2	Froyo	8	9%	11.6%
2.3	Gingerbread	9/10	47.6%	59.2%
3.X	Honeycomb	12/13	1.5%	60.7%
4.0.X	Ice cream sandwich	15	29.1%	89.8%
4.1	Jelly bean	16	9%	98.8%
4.2	Jelly bean	17	1.2%	100%

TABLE: Répartition des versions pour les accès au google play sur la dernière quinzaine de 2012

Présentation du SDK android

Téléchargement gratuit : www.d.android.com/sdk

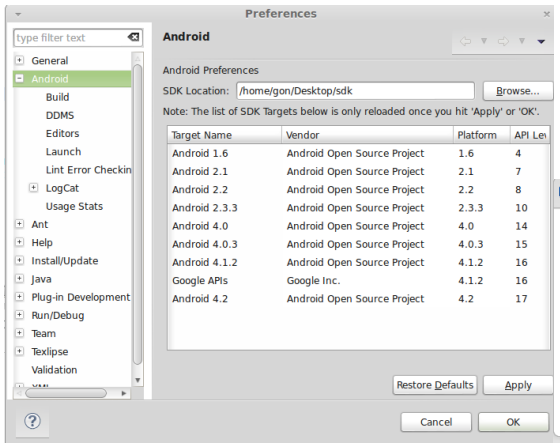


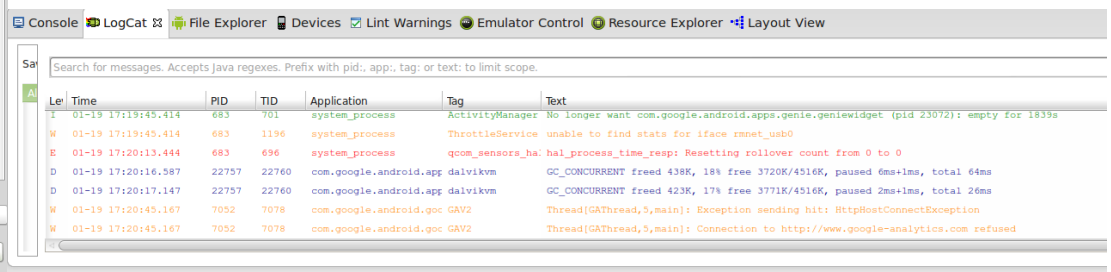
Présentation du SDK android

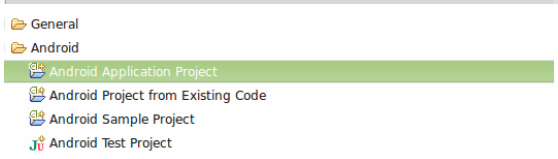
- add-ons : Google APIs
- docs : Copie de la documentation disponible sur d.android.com
- extras : Lib de compatibilité, lib pour les achats in-app . . .
- platform-tools : Binaires de communication avec les appareils android (adb, fastboot . . .)
- platforms : 1 dossier par niveau d'API téléchargé
- samples : Exemples de projets
- sources : Sources de chaque niveau d'API
- system-images : Images pour l'émulateur
- temp
- tools : Outils pour le dev (ddms, apkbuilder, lint . . .)

Plugin android pour eclipse : ADT

Installation comme un plugin eclipse classique
https ://dl-ssl.google.com/android/eclipse/
ADT fait le lien entre eclipse et le SDK android

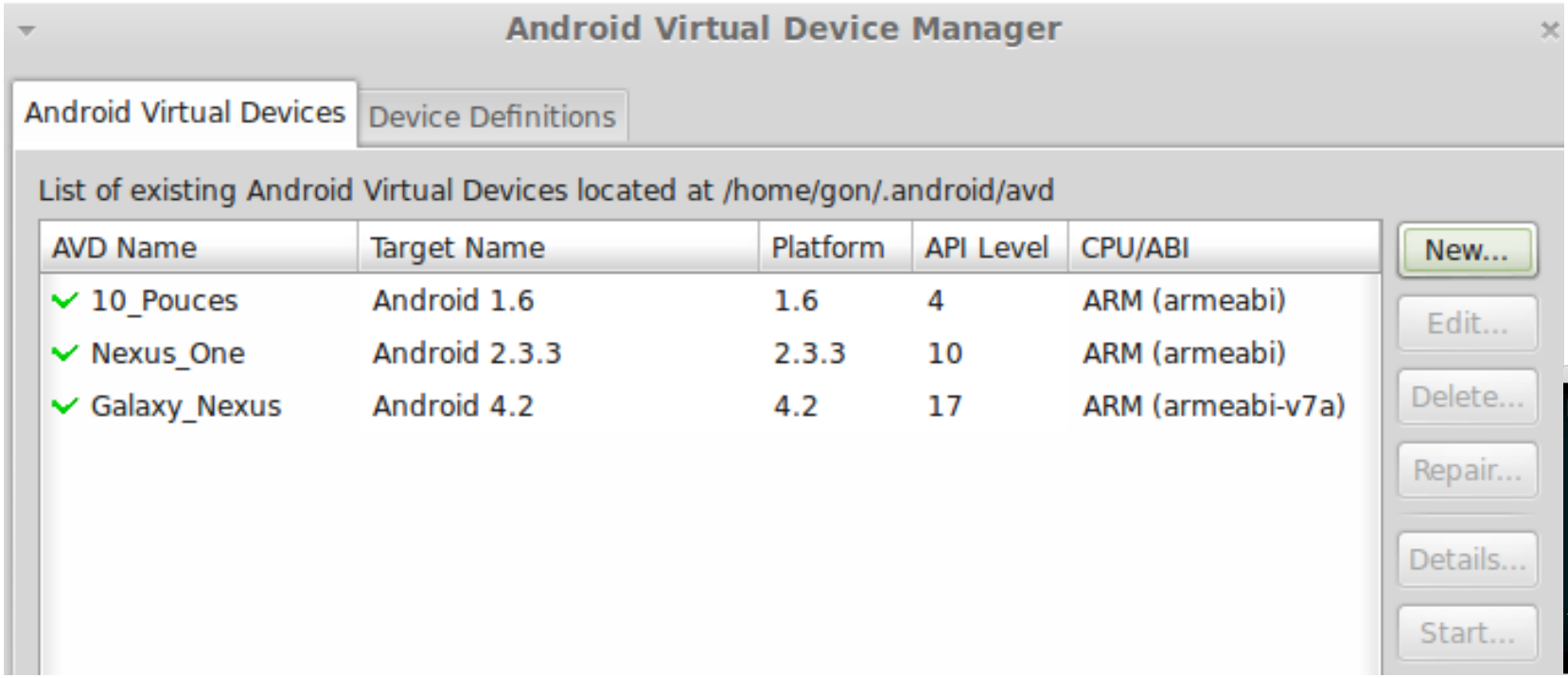






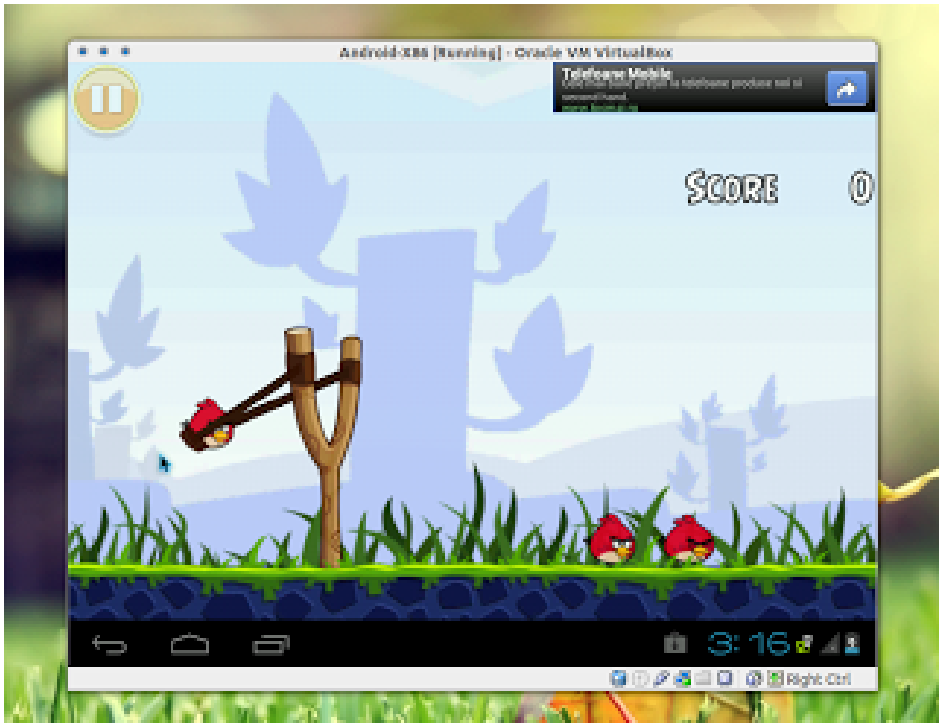
L'émulateur

- Utile pour tester certaines configurations
- ((très) très) lent
- Utiliser un appareil android à la place quand c'est possible



Alternative à l'émulateur

- Problème : émuler de l'ARM sur nos machines x86
- Résultat : émulateur ((très) très) lent
- Solution proposée : porter android sur x86
- http ://www.android-x86.org/

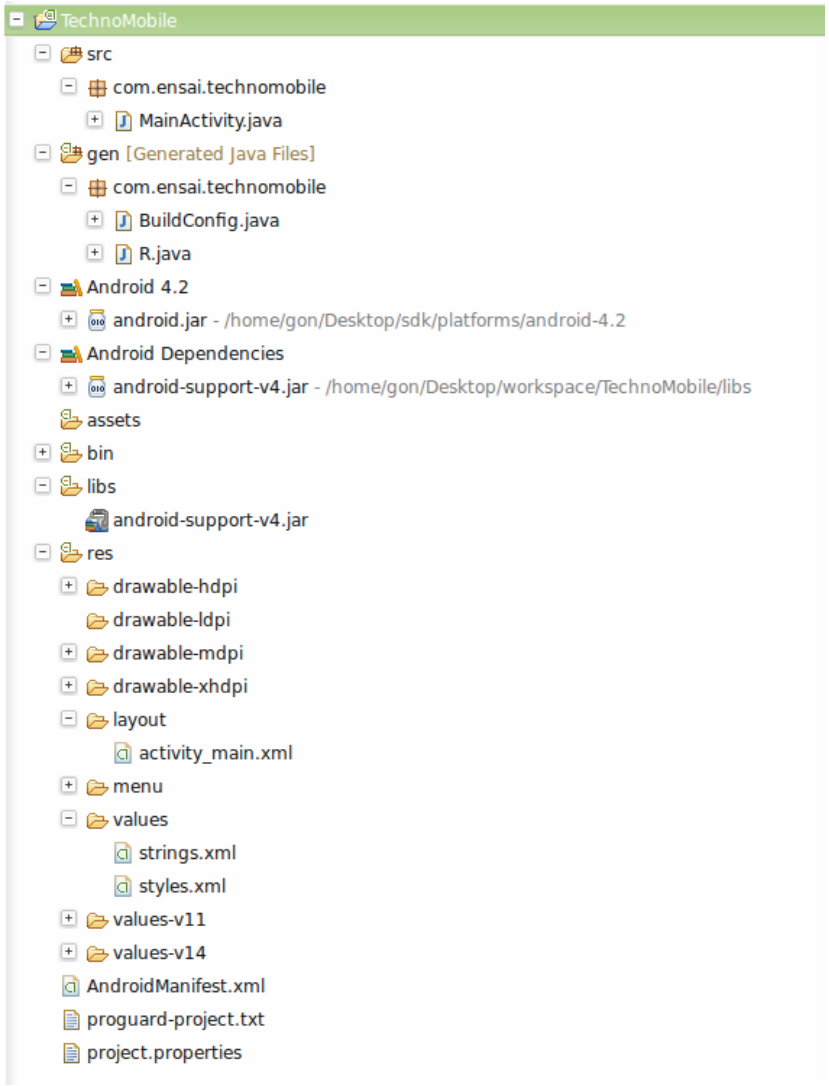


Distribuer l'application

- Une application android = un APK (+/- équivalent d'un jar)
- Création et signature de l'APK simple sous eclipse
- Distribution directe de l'APK (ex : pour tester, béta fermée)
- Publication sur le playstore, 25$ à l'inscription
- Application gratuite ou payante (30% pour google)

Contents

Organisation d'un projet android



Détail de l'organisation

- src : code source java
- gen : identifiants des ressources (généré par le sdk)
- Android 4.2 : jar correspondant à l'API cible
- Android Dependencies : jar rajoutés, correspond à libs
- assets : fichiers fournis avec l'app
- bin : résultat de la compilation (dont l'apk)
- libs : jar rajoutés
- res : ressources (layouts, strings, images . . .)
- AndroidManifest.xml : métadonnées sur l'application, composants, permissions . . .
- proguard-project.txt : configuration de proguard
- project.properties : généré par le sdk

AndroidManifest.xml : le coeur de l'application

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

    <uses-sdk
        android:minSdkVersion="8"
        android:targetSdkVersion="17" />

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

- Déclaration des composants
- Déclaration des permissions
- Analysé par l'OS à l'installation

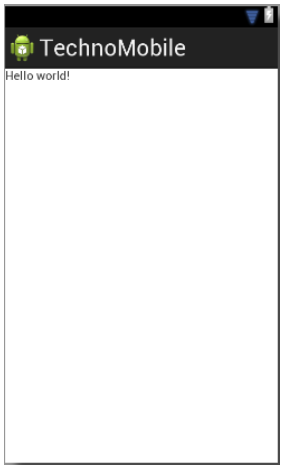
Le système de ressources

```
public final class R {
    public static final class attr {
    }
    public static final class drawable {
        public static final int ic_launcher=0x7f020000;
    }
    public static final class id {
        public static final int menu_settings=0x7f070000;
    }
    public static final class layout {
        public static final int activity_main=0x7f030000;
    }
    public static final class menu {
        public static final int activity_main=0x7f060000;
    }
    public static final class string {
        public static final int app_name=0x7f040000;
        public static final int hello_world=0x7f040001;
        public static final int menu_settings=0x7f040002;
    }
```

- Un identifiant est généré pour chaque ressource (drawable, layout, menu, values, style . . .)
- Nom de l'identifiant = nom de la ressource sans l'extension

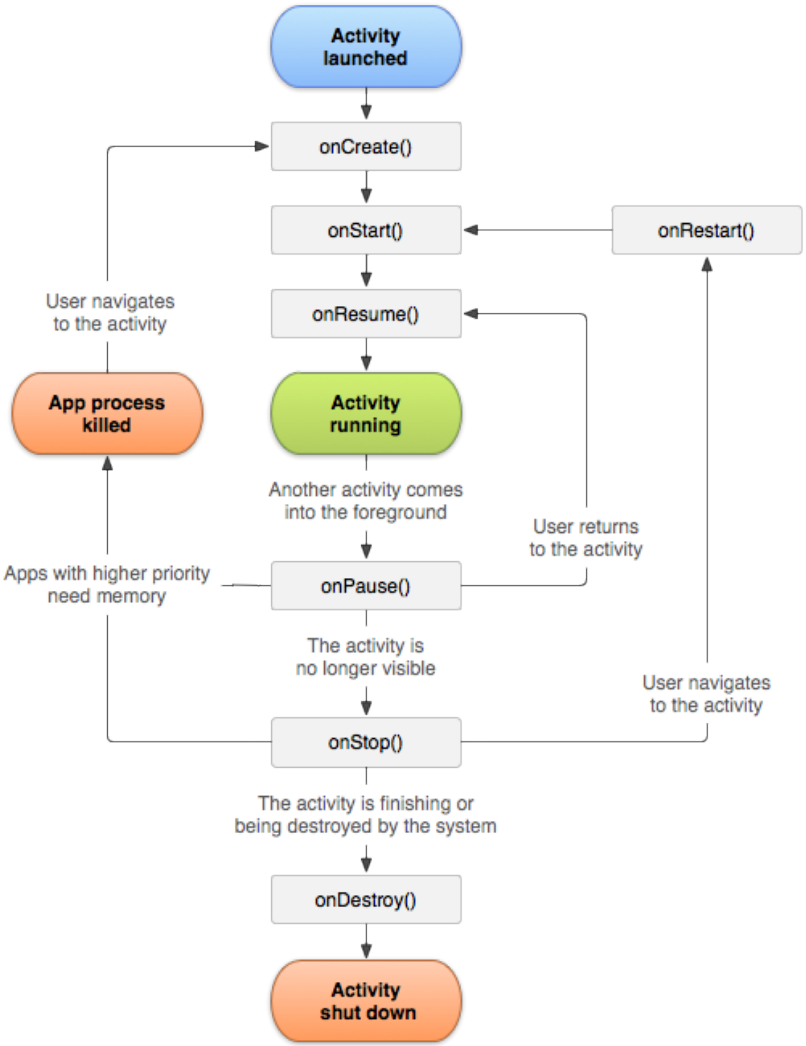
Contents

Activity, le composant de base



- 1 activity ~ un écran
- Une application peut avoir 0-n activities
- A ajouter dans le manifest
- Créer une classe java héritant de Activity

Cycle de vie d'une activity



Créer une activity : étendre Activity

```
1  public Class MyActivity extends Activity {
2
3  @Override
4  protected void onCreate(Bundle savedInstanceState) {
5  super.onCreate(savedInstanceState);
6  setContentView(R.layout.activity_main);
7  }
8  }
```

- onCreate est appellé à la création de l'activity (cf cycle de vie)
- appel obligatoire à super.onCreate
- le bundle savedInstanceState contient les informations en cas de relancement de l'activity
- savedInstanceState est null s'il s'agit du premier lancement

L'organisation d'une activity : les layouts

```
1  <LinearLayout
2  xmlns:android="http://schemas.android.com/apk/res/android"
3      android:layout_width="match_parent"
4      android:layout_height="match_parent">
5
6      <TextView
7          android:layout_width="wrap_content"
8          android:layout_height="wrap_content"
9          android:text="@string/hello_world" />
10
11 </LinearLayout>
```

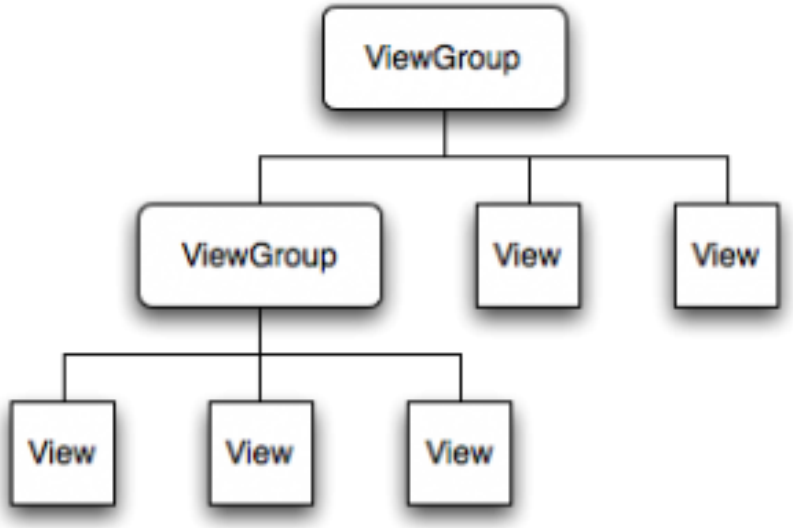
- Ils sont définis en XML dans le dossier res/layout
- Ils définissent l'organisation des vues
- Eviter au maximum de modifier / créer les layouts au runtime

Les Views

Une vue = un élement à l'écran

- TextView = Un texte
- EditText = Un champ de texte remplissable
- ImageView = Une image
- Button
- CheckBox
- Plein d'autres views de base dans android
- Possibilité de créer ses propres views en étendant View ou SurfaceView

Les ViewGroups



- LinearLayout
- RelativeLayout
- ListView
- Plein d'autres
- Les vôtres :)

Manipuler les éléments de l'UI en java

Etape 1 : donner un identifiant à la vue

```
1  <LinearLayout
2  xmlns:android="http://schemas.android.com/apk/res/android"
3      android:layout_width="match_parent"
4      android:layout_height="match_parent"
5      android:id="@+id/monlayout">
6
7      <Button
8          android:layout_width="wrap_content"
9          android:layout_height="wrap_content"
10         android:id="@+id/monbouton"
11         android:text="@string/hello_world" />
12
13 </LinearLayout>
```

Manipuler les éléments de l'UI en java

Etape 2 : récupérer les réferences vers les views

```
1  public Class MyActivity extends Activity {
2
3  ViewGroup layout = null;
4  Button bouton = null;
5
6  @Override
7  protected void onCreate(Bundle savedInstanceState) {
8  super.onCreate(savedInstanceState);
9  setContentView(R.layout.activity_main);
10 layout = (ViewGroup) findViewById(R.id.monlayout);
11 bouton = (Button) findViewById(R.id.monbouton);
12 }
13 }
```

Manipuler les éléments de l'UI en java

```
1  public Class MyActivity extends Activity {
2
3  ViewGroup layout = null;
4  Button bouton = null;
5
6  @Override
7  protected void onCreate(Bundle savedInstanceState) {
8  super.onCreate(savedInstanceState);
9  setContentView(R.layout.activity_main);
10 layout = (ViewGroup) findViewById(R.id.monlayout);
11 bouton = (Button) findViewById(R.id.monbouton);
12 }
13
14 public void changerTexte(String texte) {
15 bouton.setText(texte);
16 }
17
18 public void cacherTout() {
19 layout.setVisibility(View.INVISIBLE);
20 }
21 }
```

Ecouter les évenements

- Système de listeners (cf swing)
- Il se passe quelque chose sur la vue (touch, focus . . .) : le listener est prévenu
- Pour simplifier, sur android on a en général qu'un listener par évenement et par view (setXListener au lieu de addXListener sous swing)

Ecouter les évenements, guide du bon listener

Etape 1 : Les interfaces XListener

```
1 public Interface OnClickListener {
2 void onClick(View v);
3 }
```

Etape 2 : Implémenter l'interface

```
1 public MaClasse implements OnClickListener {
2 public void onClick(View v) {
3 //Un click a ete fait sur la vue v
4 }
5 }
```

Ecouter les évenements, guide du bon listener

Etape 3 : S'enregistrer comme listener

```
1  public Class MyActivity extends Activity implements
       OnClickListener {
2
3  Button bouton = null;
4
5      @Override
6      protected void onCreate(Bundle savedInstanceState) {
7          super.onCreate(savedInstanceState);
8          setContentView(R.layout.activity_main);
9          bouton = (Button) findViewById(R.id.monbouton);
10         bouton.setOnClickListener(this);
11     }
12
13     public void onClick(View v) {
14         //Un Click a ete fait sur la vue v
15     }
16 }
```

Ecouter les évenements, quelques feintes

Feinte 1 : Utiliser des listeners anonymes

```
1  public Class MyActivity extends Activity implements
       OnClickListener {
2
3      Button bouton = null;
4
5      @Override
6      protected void onCreate(Bundle savedInstanceState) {
7          super.onCreate(savedInstanceState);
8          setContentView(R.layout.activity_main);
9          bouton = (Button) findViewById(R.id.monbouton);
10         bouton.setOnClickListener(new OnClickListener() {
11                 public void onClick(View v) {
12                 //Un Click a ete fait sur la vue v
13                 }
14             });
15     }
16 }
```

Ecouter les évenements, quelques feintes

Feinte 2 : Définir le listener directement dans le layout

```
1  <LinearLayout
2  xmlns:android="http://schemas.android.com/apk/res/android"
3      android:layout_width="match_parent"
4      android:layout_height="match_parent"
5      android:id="@+id/monlayout">
6
7      <Button
8          android:layout_width="wrap_content"
9          android:layout_height="wrap_content"
10         android:id="@+id/monbouton"
11         android:text="@string/hello_world"
12         android:onClick="clickSurLeBouton" />
13
14 </LinearLayout>
```

```
1  public void clickSurLeBouton(View v) //dans MyActivity
```

Les intents

- On déclare son intention, android réagit en conséquence
- Intents implicites “Je veux ouvrir la page web https ://twitter.com/Ensai35”
- “Je veux envoyer un mail à jlegouic@ensai.fr avec le titre URGENT : FOOT”
- Intents explicites “Je veux lancer l’activity MyActivity”

Lancer un intent implicite

```
1  public MyActivity extends Activity {
2
3      public void envoyerMail() {
4          Intent i = new Intent(Intent.ACTION_SEND);
5          i.setType("message/rfc822");
6          i.putExtra(Intent.EXTRA_EMAIL   , "annee2@ensai.fr");
7          i.putExtra(Intent.EXTRA_SUBJECT,"URGENT : FOOT");
8          i.putExtra(Intent.EXTRA_TEXT    , "...");
9          try {
10             startActivity(i);
11         }
12         catch (ActivityNotFoundException ex) {
13             //Pas de client mail installe
14         }
15     }
16 }
```

Contents