

Technologies mobiles

Olivier Levitt

Source et pdf de ce cours et des TP :
<https://goo.gl/eIPv5B>



iOS



Windows 10



Firefox OS

Sommaire

- 1 Présentation et objectifs du cours
 - Organisation
 - Contexte et objectifs
- 2 Le développement mobile
 - Spécificités du développement mobile
 - Présentation des différents OS mobile
- 3 Le développement sur Android
 - Mise en place
 - Architecture
 - IHM
 - Les données
 - Bonus

Sommaire

1 Présentation et objectifs du cours

- Organisation
- Contexte et objectifs

2 Le développement mobile

- Spécificités du développement mobile
- Présentation des différents OS mobile

3 Le développement sur Android

- Mise en place
- Architecture
- IHM
- Les données
- Bonus

Planning

- 13 avril : 3h de cours
- 25 avril : 3h de TP, 3h de cours
- 2 mai : 6h de TP
- Validation des sujets de projet avant le 9 mai
- 9 mai : 3h de TP, 3h de TP dédiées au projet
- 31 mai : Soutenance du projet

Evaluation

- Projet : création d'une application
- Groupe de 2
- Sujet "libre"
- 3h de TP dédiées au projet + **travail personnel**
- Soutenance / Présentation de l'application

Evaluation, exemples de sujets

- Gestion d'une bibliothèque
- Client twitter
- Répondeur SMS
- Quiz
- Transports de rennes
- Open data (ex : <http://data.gouv.fr>)

Sommaire

1 Présentation et objectifs du cours

- Organisation
- Contexte et objectifs

2 Le développement mobile

- Spécificités du développement mobile
- Présentation des différents OS mobile

3 Le développement sur Android

- Mise en place
- Architecture
- IHM
- Les données
- Bonus

Contexte et objectifs

- Smartphones, tablettes et assimilés (TV, montre, autoradio, consoles de jeu ...)
- Développement natif d'application, pas de dev système
- 1ère partie : le développement mobile en général
- 2ème partie : application sous Android

Sommaire

1 Présentation et objectifs du cours

- Organisation
- Contexte et objectifs

2 Le développement mobile

- **Spécificités du développement mobile**
- Présentation des différents OS mobile

3 Le développement sur Android

- Mise en place
- Architecture
- IHM
- Les données
- Bonus

Des appareils suréquipés

- Téléphonie (SMS, MMS, appels)
- Internet (GPRS, EDGE, 3G, 4G, WiFi)
- Réseaux locaux (Bluetooth, WiFi direct, tethering)
- Capteurs (Luminosité, proximité, gyroscope, NFC)
- Localisation (GPS, triangulation, SSID WiFi)
- Notifications (Vibreur, haut-parleurs, LED)
- Photo / vidéo / visio
- Stockage de données (Mémoire flash, SD externe, SQLite)
- Interactions (Ecran tactile, gestes, boutons physique, reconnaissance vocale, lecteur d'empreintes)
- Et bien d'autres ...

Et des API pour utiliser tout ça !

Des contraintes techniques importantes

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

Contraintes à garder en tête en permanence.

La fragmentation

Une application publiée sur le google playstore cible plus de 6000 appareils différents !

- “Write once, run everywhere” ?
- Comment tester / débbuger pour tous ces appareils ?
- Eviter de gêner 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 5)
- Phablets : 5 à 6 pouces (Galaxy note, Oneplus two, iPhone 6)
- 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
- Éléments hardware présents
- Puissance
- Modifications constructeur / “rom custom”
- ...

Un monde qui évolue très vite

- Evolution technologique permanente (nouveaux appareils, nouvelles versions des OS)
- Evolution du marché (parts de marché des OS)
- Nouveaux OS
- Concurrence sur les stores
- Développement des réseaux et nouveaux usages (4G)

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 ...)
- Alternative cross-platform ?

Sommaire

1 Présentation et objectifs du cours

- Organisation
- Contexte et objectifs

2 Le développement mobile

- Spécificités du développement mobile
- **Présentation des différents OS mobile**

3 Le développement sur Android

- Mise en place
- Architecture
- IHM
- Les données
- Bonus

iOS



- Soutenu par Apple
- Présenté le 9 janvier 2007
- Réservé aux produits Apple (iPhone, iPad, iPod, iWatch)
- Programmation en objective-C (+ swift), sur mac OS X uniquement
- Appstore : validation + 100\$ / an

Android



- Soutenu par Google
- Version 1.0 en septembre 2008, commerciale 1.5 en avril 2009
- Plus de 6000 appareils officiellement supportés, plus de 50 constructeurs
- Programmation en Java (+kotlin), sur Windows / OS X / Linux
- Noyau open-source (Apache 2.0)
- Google playstore : “pas de validation” + 25\$

Windows 10 mobile

Windows 10

- Soutenu par Microsoft
- Première version en février 2015
- Téléphones Lumia (ex Nokia)
- Programmation en C# sur Windows
- Windows Store : validation + gratuit

Blackberry 10



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

Firefox OS



Firefox OS

- Soutenu par Mozilla
- Premiers téléphones présentés le 22 janvier 2012 (geekophone)
- HTML5
- Open-source
- Firefox marketplace : validation + gratuit

RIP

Ubuntu on phones

- Soutenu par Canonical
- Teaser le 2 janvier 2012
- Constructeur BQ + installable sur Android
- HTML5, C/C++ + QML
- Open-source
- Convergence PC
- Ubuntu store : validation automatique + gratuit

RIP

Sommaire

1 Présentation et objectifs du cours

- Organisation
- Contexte et objectifs

2 Le développement mobile

- Spécificités du développement mobile
- Présentation des différents OS mobile

3 Le développement sur Android

- **Mise en place**
- Architecture
- IHM
- Les données
- Bonus

Les marque-pages

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

Avant de commencer, la checklist

Obligatoire :

- Des bases de programmation en Java
- Un ordinateur (Windows, Linux, Mac OS X)

Conseillé :

- Un appareil Android
- Parler anglais
- Suivre l'actualité

Outils de développement

Téléchargement :

<https://developer.android.com/studio/index.html>

- SDK
- IDE

Le SDK



add-ons



build-tools



docs



emulator



extras



licenses



patcher



platforms



platform-tools



skins



sources



system-images



temp



tools



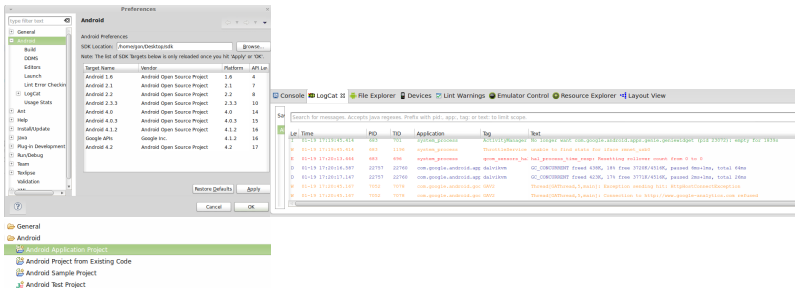
SDK Readme.txt

ADB

- Android Debugging Bridge
- Outil du SDK
- Communication ordinateur \Leftrightarrow appareil / émulateur
- Installation d'application, lancement, log ...
- Intégré dans l'IDE ou ligne de commande

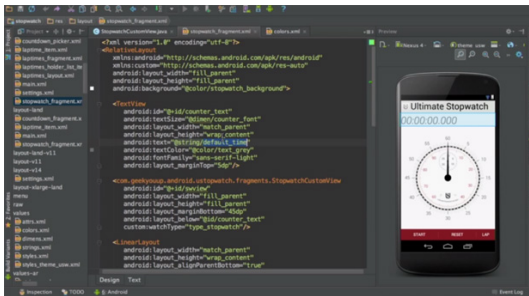
IDE — Plugin android pour eclipse : ADT

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



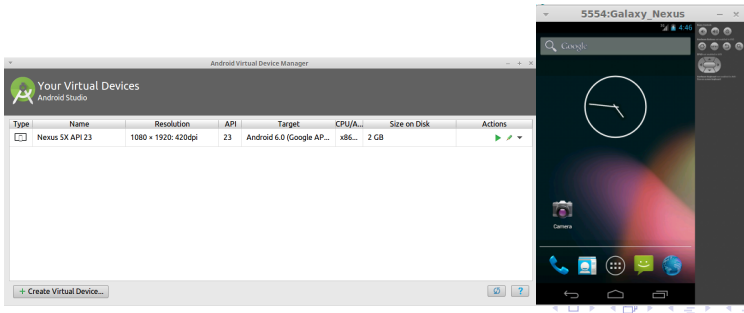
IDE — Android studio

- Basé sur IntelliJ IDEA
- IDE retravaillé pour android
- IDE officiel depuis décembre 2014
- Support natif de gradle



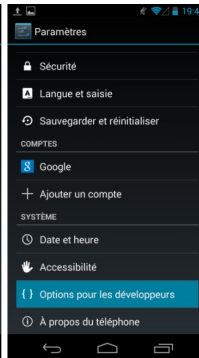
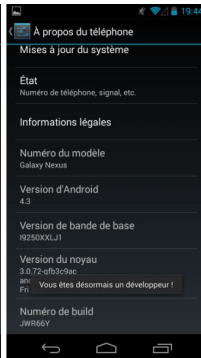
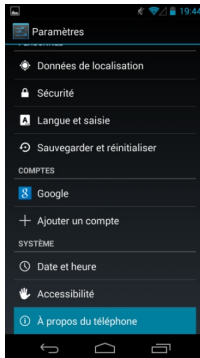
L'émulateur

- Utile pour tester certaines configurations
- Lent
- Se comporte comme un appareil Android classique (ADB)
- Utiliser un appareil android à la place quand c'est possible



Utiliser un appareil android

- Activer les “options pour les développeurs” sur l'appareil
- Paramètres / A propos du téléphone / Toucher 4 fois “Numéro de build”
- Activer le débogage USB



Les niveaux d'API

| Version | Nom | API level | Distribution | Rétrocumul |
|---------|--------------------|-----------|--------------|------------|
| 2.3 | Gingerbread | 10 | 0.9% | 100% |
| 4.0 | Ice cream sandwich | 15 | 0.9% | 99.1% |
| 4.1 | Jelly bean | 16 | 3.5% | 98.2% |
| 4.2 | Jelly bean | 17 | 5.1% | 94.7% |
| 4.3 | Jelly bean | 18 | 1.5% | 89.6% |
| 4.4 | Kitkat | 19 | 20.0% | 88.1% |
| 5.0 | Lollipop | 21 | 9.0% | 68.1% |
| 5.1 | Lollipop | 22 | 23.0% | 59.1% |
| 6.0 | Marshmallow | 23 | 31.2% | 36.1% |
| 7.0 | Nougat | 24 | 4.5% | 4.9% |
| 7.1 | Nougat | 25 | 0.4% | 0.4% |

TABLE : Répartition des versions pour les accès au google play sur la première semaine d'avril 2017

Sommaire

1 Présentation et objectifs du cours

- Organisation
- Contexte et objectifs

2 Le développement mobile

- Spécificités du développement mobile
- Présentation des différents OS mobile

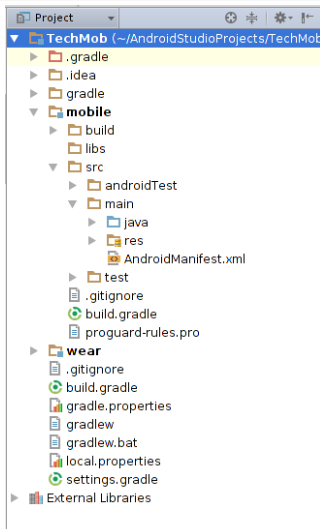
3 Le développement sur Android

- Mise en place
- **Architecture**
- IHM
- Les données
- Bonus

Gradle

- Gestion des dépendances et du build
- “Successeur” de maven
- Optionnel mais fortement recommandé
- Conf dans build.gradle

Organisation d'un projet Android gradléisé



Détail de l'organisation

- .gradle / gradle : dossiers interne gradle
- .idea : dossier interne android studio
- mobile : module téléphone / tablette
- wear : module android wear (montre)
- build : résultats des compilations dont APK
- libs : dépendances manuelles
- src : code source
- main / java : code source Java
- main / res : Ressources (strings, drawables ...)
- AndroidManifest.xml : métadonnées sur l'application, composants, permissions ...
- tests / androidTest : tests unitaires / tests fonctionnels
- build.gradle : configuration gradle
- proguard-rules.pro : configuration proguard

AndroidManifest.xml : la carte d'identité de l'application

```
<?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
- Déclaration d'autres métadonnées de l'application
- Analysé par Android à l'installation

Les permissions

- Obligatoires pour certaines fonctions (internet, géolocalisation, hardware ...)
- Les applications peuvent définir leurs propres permissions
- L'utilisateur est prévenu à l'installation / mise à jour

```
1 <uses-permission android:name="android.permission.INTERNET" /  
  >  
2 <uses-permission android:name="android.permission.READ_SMS" /  
  >  
3 <uses-permission android:name="android.permission.CAMERA" />
```

- Depuis Android 6.0 (API 23), permissions au runtime

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 ...)
- Les fonctions qui utilisent des ressources sont surchargées pour aussi accepter l'identifiant de la ressource correspondante
- Utiliser des ressources différentes en fonction de la configuration (values et values-fr, drawable et drawable-hdpi)

Exemple de ressources : les strings

- Eviter au maximum les chaînes de caractères en dur
- Mettre toutes les chaînes dans res/values/strings.xml
- Très facile de traduire ensuite : res/values-en, res/values-fr ... (values et values-fr, drawable et drawable-hdpi)

```
1 <resources>
2   <string name="app_name">Technomobile</string>
3   <string name="hello_world">Hello world!</string>
4   <string name="menu_settings">Settings</string>
5   <string-array name="statuts">
6       <item>Fonctionnaire</item>
7       <item>Ingenieur</item>
8   </string-array>
9 </resources>
```

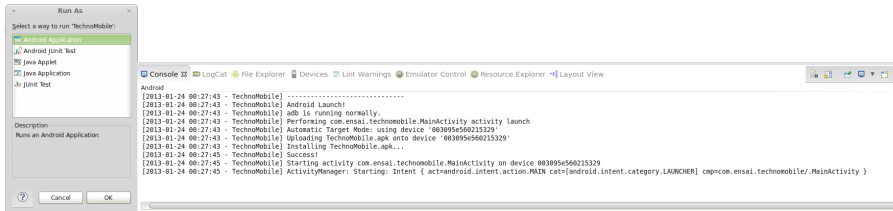
Déployer l'application

- Une application android = un APK (équivalent d'un jar)
- Une application android doit être signée
- Attention à ne pas perdre la clé !
- Création et signature de l'APK simplifié par l'IDE

Distribuer l'application

- Distribution directe de l'APK (ex : pour tester, bêta fermée)
- Publication sur le playstore, 25\$ à l'inscription
- Application gratuite ou payante (30% pour google)

Processus de déploiement en développement



- Android Studio demande au SDK de builder l'APK
- AS signe l'APK avec la clé debug
- AS demande à ADB (SDK) d'installer l'application
- Soit sur un appareil android connecté soit sur un émulateur

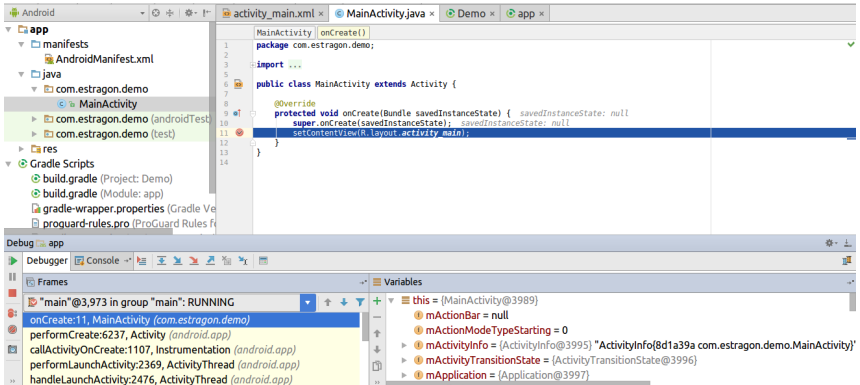
Déboguer l'application

Unfortunately, TechnoMobile
has stopped.

OK

- Si une exception n'est pas rattrapée, android tue l'application
- On parle de "force close" (FC)
- Comment déboguer une application qui tourne sur un appareil ou un émulateur ?

Le mode debug



- Le debugger Java fonctionne aussi sur Android
- Intégré directement à Android Studio

Logcat



- Le SDK fournit un outil très pratique : logcat
- On appelle logcat par adb : `adb logcat` ou on utilise la vue LogCat d'Android Studio
- Logcat affiche l'ensemble des logs, système et application

Logcat, exemple

```
Android Monitor
Emulator Nexus_5X_API 23 Android 6.0, API 23 fr.ensai.technomobile.myapplication (3355)
Monitors logcat Verbose [ 04-11 15:28:33.506 3355: 3355 D/ ]
HostConnection:getId() New Host Connection established 0x7f116e7b57a0, tid 3355

[ 04-11 15:28:33.560 3355: 3420 D/ ]
HostConnection:getId() New Host Connection established 0x7f11723d1360, tid 3420
04-11 15:28:33.567 3355-3420/fr.ensai.technomobile.myapplication I/OpenGLRenderer: Initialized EGL, version 1.4
04-11 15:28:33.667 3355-3420/fr.ensai.technomobile.myapplication W/OpenGLRenderer: Failed to choose config with EGL_SWAP_BEHAVIOR_PRESERVED, retrying without...
04-11 15:28:35.529 3355-3361/fr.ensai.technomobile.myapplication E/Surface: getSlotFromBufferLocked: unknown buffer: 0x7f116e907230
04-11 15:28:57.529 3355-3361/fr.ensai.technomobile.myapplication I/art: Debugger is no longer active

gon@gon-Macmini ~ $ adb logcat -s "AndroidRuntime"
----- beginning of /dev/log/system
----- beginning of /dev/log/main
D/AndroidRuntime(22495): >>>>> AndroidRuntime START com.android.internal.os.RuntimeInit <<<<<
D/AndroidRuntime(22495): CheckJNI is OFF
D/AndroidRuntime(22495): Calling main entry com.android.commands.pm.Pm
D/AndroidRuntime(22495): Shutting down VM
D/AndroidRuntime(22599): >>>>> AndroidRuntime START com.android.internal.os.RuntimeInit <<<<<
D/AndroidRuntime(22599): CheckJNI is OFF
D/AndroidRuntime(22599): Calling main entry com.android.commands.am.Am
D/AndroidRuntime(22599): Shutting down VM
D/AndroidRuntime(22639): FATAL EXCEPTION: main
E/AndroidRuntime(22639): java.lang.RuntimeException: Unable to start activity ComponentInfo{com.ensai.technomobile/com.ensai.technomobile.MainActivity}: java.lang.NullPointerException
E/AndroidRuntime(22639): at android.app.ActivityThread.performLaunchActivity(ActivityThread.java:2384)
E/AndroidRuntime(22639): at android.app.ActivityThread.handleLaunchActivity(ActivityThread.java:2354)
E/AndroidRuntime(22639): at android.app.ActivityThread.access$600(ActivityThread.java:150)
E/AndroidRuntime(22639): at android.app.ActivityThread$H.handleMessage(ActivityThread.java:1244)
E/AndroidRuntime(22639): at android.os.Handler.dispatchMessage(Handler.java:99)
E/AndroidRuntime(22639): at android.os.Looper.loop(Looper.java:137)
E/AndroidRuntime(22639): at android.app.ActivityThread.main(ActivityThread.java:5191)
E/AndroidRuntime(22639): at java.lang.reflect.Method.invoke(Native Method)
E/AndroidRuntime(22639): at java.lang.reflect.Method.invoke(Method.java:511)
E/AndroidRuntime(22639): at com.android.internal.os.ZygoteInit$MethodAndArgsCaller.run(ZygoteInit.java:795)
E/AndroidRuntime(22639): at com.android.internal.os.ZygoteInit.main(ZygoteInit.java:562)
E/AndroidRuntime(22639): at dalvik.system.NativeStart.main(Native Method)
E/AndroidRuntime(22639): Caused by: java.lang.NullPointerException
E/AndroidRuntime(22639): at com.ensai.technomobile.MainActivity.onCreate(MainActivity.java:13)
E/AndroidRuntime(22639): at android.app.Activity.performCreate(Activity.java:5184)
E/AndroidRuntime(22639): at android.app.Instrumentation.callActivityOnCreate(Instrumentation.java:1080)
E/AndroidRuntime(22639): at android.app.ActivityThread.performLaunchActivity(ActivityThread.java:2258)
```

Stacktrace or GTFO



Logcat, utilisation

- Oublier `System.out.println()` !
- 5 niveaux de gravité : Error, Warn, Info, Debug, Verbose
- Possibilité dans `adb logcat` de filtrer par gravité et/ou TAG
- Pratique recommandée : un TAG par application

```
1 public void sauvegarderScore(int score) {  
2     Log.i(TAG, "Sauvegarde du score "+score);  
3     if (score < 0) {  
4         Log.w(TAG, "Le score est negatif");  
5     }  
6     try {  
7         //traitement  
8     }  
9     catch (Exception e) {  
10        Log.e(TAG, "Erreur de score", e);  
11    }  
12 }
```

Démo

- Démotime !

Sommaire

1 Présentation et objectifs du cours

- Organisation
- Contexte et objectifs

2 Le développement mobile

- Spécificités du développement mobile
- Présentation des différents OS mobile

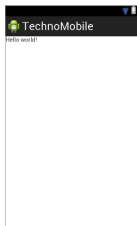
3 Le développement sur Android

- Mise en place
- Architecture
- **IHM**
- Les données
- Bonus

Une exécution moins linéaire qu'en Java classique

- Une application Android est un ensemble de composants
- Pas de méthode main
- Des points d'entrée multiples possibles
- Exemple : appli SMS

Activity, le composant de base



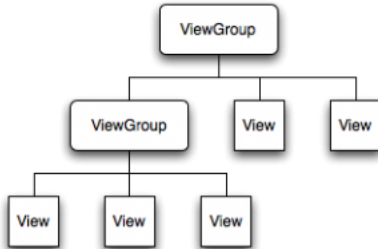
- 1 activity ~ un écran
- Une application peut avoir $0 - n$ activities
- A déclarer dans le manifest
- Créer une classe Java héritant de Activity

Les Views

Une vue = un élément à l'écran

- TextView = Un texte
- EditText = Un champ de texte remplissable
- ImageView
- Button
- CheckBox
- Plein d'autres views de base dans android
- Possibilité de créer ses propres views en héritant (directement ou indirectement) de View

Les ViewGroups



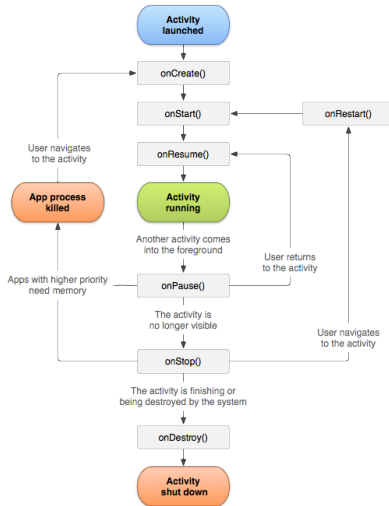
- LinearLayout
- RelativeLayout
- ListView
- Plein d'autres
- Les vôtres

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 définissent l'organisation des vues
- Ils sont définis en XML dans le dossier res/layout
- @string fait référence à une ressource de strings.xml
- Eviter au maximum de modifier / créer les layouts au runtime

Cycle de vie d'une activity



Créer une activity : étendre Activity

```
1 public Class MainActivity 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 appelé à la création de l'activity (cf cycle de vie)
- appel obligatoire à super.onCreate
- le bundle savedInstanceState contient les informations en cas de relance de l'activity
- savedInstanceState est null s'il s'agit du premier lancement

Créer une activity : la déclarer dans le manifest

```
1 <activity android:name="com.ensai.technomobile.MainActivity"
2         android:label="Accueil" >
3     <intent-filter>
4         <action android:name="android.intent.action.MAIN" />
5         <category android:name="android.intent.category.
6             LAUNCHER" />
7     </intent-filter>
</activity>
```

- Balise Activity à l'intérieur de la balise Application
- On précise la classe Java correspondante

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:text="@string/hello_world"
11      android:id="@+id/monbouton" />
12
13 </LinearLayout>
```

- @+id crée l'identifiant s'il n'existe pas déjà
- L'identifiant doit être unique dans la hiérarchie (sinon conflit)

Manipuler les éléments de l'UI en Java

Etape 2 : récupérer les références 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 }
```

- findViewById renvoie un objet de type View : on cast
- si aucune vue n'a l'identifiant demandé, findViewById renvoie null

Manipuler les éléments de l'UI en Java

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


Ecouter les événements

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

Ecouter les événements, guide du bon listener

Etape 1 : Les interfaces XListener (elles existent déjà, ne pas les réécrire!)

```
1 public Interface OnClickListener {  
2     public 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 événements, 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 événements, 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 événements, quelques feintes

Feinte 2 : Définir le listener directement dans le layout

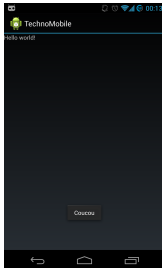
```
1 <LinearLayout
2   xmlns:android="http://schemas.android.com/apk/res/android"
3     android:layout_width="match_parent"
4     android:layout_height="match_parent"
5     android:id="@+id/monlayout">
6
7     <Button
8       android:layout_width="wrap_content"
9       android:layout_height="wrap_content"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

La classe abstraite context

- La plupart des fonctions d'android (accéder à une ressource, lancer une activité ...) nécessitent une instance de Context
- Un Context regroupe des informations globales sur l'environnement de l'application
- Android se charge de créer les contextes, n'en instanciez pas !
- Activity hérite (indirectement) de Context
- Les views ont toutes une référence vers un context

Affichage d'un court message : le toast



```
1 public MyActivity extends Activity {  
2  
3     public void faireCoucou() {  
4         Toast.makeText(this, "Coucou", Toast.LENGTH_LONG).show  
5             ();  
6         Toast.makeText(this, R.string.coucou, Toast.  
7             LENGTH_SHORT).show();  
8     }  
9 }
```

Previously on Android

- Une appli = un ensemble de composants
- Plusieurs points d'entrée possibles
- Tous les composants doivent être définis dans le manifest
- Activity, composant ~ un écran
- Créer une activity = hériter de Activity
- Méthodes du cycle de vie (onCreate, onPause, onResume ...)

Previously on Android, les layouts

- Les composants graphiques sont définis dans des layouts XML
- Layout = hiérarchie de ViewGroup et de View
- Exemple de ViewGroup : LinearLayout
- Exemples de View : Button, TextView, EditText
- Tous les éléments doivent définir `layout_width` et `layout_height`
- `Wrap_content` : seulement la place nécessaire
- `Match_parent` : toute la place du père
- Plaquer (inflate) un layout à une activity :
`setContentView(R.layout.xxx)`

Previously on Android, manipuler les éléments UI

- Pour manipuler les éléments UI en Java : leur donner un ID
- `@+id/identifiant` : crée un int IDENTIFIANT dans R.id
- R : fichier autogénéré des ressources
- Dans l'activity : utiliser `findViewById(R.id.identifiant)`
- Caster la valeur trouvée
- Ensuite, usage classique : `bouton.setText(...)`

Previously on Android, écouter les événements

- Implémenter OnXListener (ex : OnClickListener)
- S'enregistrer comme listener auprès de la View (ex : `bouton.setOnClickListener(...)`)
- Chaque événement correspond à un appel de la méthode implémentée
- Feinte 1 : enregistrer des listeners anonymes (créés ad-hoc)
- Feinte 2 : déclarer le listener dans le layout XML (View `onClick`)

Les autres composants d'une application : les services

- Tâche en arrière plan
- Cycle de vie différent de celui d'une activity
- Pas d'interface graphique (sauf si une activity communique avec le service)
- Exemples : lecteur MP3, client torrent, système de mise à jour, taskkiller ...

Les autres composants d'une application : les broadcast receivers

- Composant recevant les annonces système et les annonces des autres applications
- Permet de réagir à certains événements
- Possibilité de lancer des activités ou des services depuis le broadcast receiver
- Exemples : indicateur de batterie, antivirus, lancement au démarrage, intercepter un appel ou un SMS reçu ...

Les autres composants d'une application : les content-providers

- Composant servant à distribuer les données
- Peu importe comment les données sont stockées (sqlite, préférences, fichier . . .)
- Respecte une interface d'utilisation des données
- Peut permettre la récupération, la modification et/ou la suppression de données
- Principalement destiné à une utilisation entre applications
- Gestion de la sécurité et des droits (permissions)
- Exemple : accès aux données système (SMS, contacts . . .)

Les intents

- Messages asynchrones échangés entre applications ou entre composants
- On déclare son intention, android réagit en conséquence

Un intent est constitué de plusieurs informations principales :

- action : l'action à effectuer
- data : les données concernées, sous la forme d'une URI
- Ex : ACTION_VIEW content ://contacts/people/1
- ACTION_DIAL tel :123

Et d'informations secondaires :

- category : pour préciser l'action
- type : pour forcer le type de données
- extras : des données libres supplémentaires
- component : dans le cas des intents explicites

Lancer un intent implicite

“Je veux envoyer un mail à annee2@ensai.fr avec le titre URGENT : Android”

```
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 : Android");
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 }
```


Lancer un intent explicite

“Je veux lancer l'activity ProfilActivity en transmettant l'id 42”

```
1 public MyActivity extends Activity {  
2     public void visualiserProfil() {  
3         Intent intent = new Intent(this, ProfilActivity.  
4             class);  
5         //this est la en tant que context  
6         intent.putExtra("id",42);  
7         startActivity(intent);  
8     }  
}
```

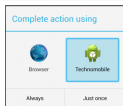
```
1 public ProfilActivity extends Activity {  
2     protected void onCreate(Bundle savedInstanceState) {  
3         super.onCreate(savedInstanceState);  
4         Intent intent = getIntent();  
5         int id = intent.getIntExtra("id",-1);  
6     }  
7 }
```

Filtrer les intents, le principe

- startActivity(), startService() et sendBroadcast() déclenchent des intents
- android filtre les composants susceptibles de recevoir l'intent en fonction des intent-filters de chaque composant

```
1 <activity android:name="com.ensai.technomobile.MainActivity"
2     android:label="@string/app_name" >
3     <intent-filter>
4         <action android:name="android.intent.action.
5             MAIN" />
6         <category android:name="android.intent.
7             category.LAUNCHER" />
8     </intent-filter>
9 </activity>
```

Filtrer les intents, exemple



```
1 <activity
2     android:name="com.ensai.technomobile.
      TwitterActivity"
3     android:label="@string/app_name" >
4     <intent-filter>
5         <data
6             android:host="twitter.com"
7             android:scheme="http" />
8         <category android:name="android.intent.
      category.DEFAULT" />
9         <category android:name="android.intent.
      category.BROWSABLE" />
10        <action android:name="android.intent.action.
      VIEW" />
11    </intent-filter>
12 </activity>
```

Sommaire

1 Présentation et objectifs du cours

- Organisation
- Contexte et objectifs

2 Le développement mobile

- Spécificités du développement mobile
- Présentation des différents OS mobile

3 Le développement sur Android

- Mise en place
- Architecture
- IHM
- **Les données**
- Bonus

Il existe plusieurs façons de stocker les données sur un appareil android

- **SharedPreferences** : stockage de données primitives
- **Fichiers** : contenus sur la mémoire interne ou externe
- **Bases de données** : SQLite
- **Stockage distant** : Webservice / cloud

Documentation android officielle sur le stockage de données :
<http://developer.android.com/guide/topics/data/data-storage.html>

SharedPreferences

- Equivalent des Preferences de Java
- Stockage de données primitives (int, boolean, String ...)
- Mapping clé / valeur
- Très facile de créer un écran de paramètres (PreferenceActivity)
- Système de listeners pour être prévenu lors d'un changement
- Léger à implémenter

SharedPreferences, accéder aux préférences

Accéder simplement aux préférences :

```
1 public void sauvegarderScore() {  
2     SharedPreferences preferences = PreferenceManager.  
        getDefaultSharedPreferences(context);  
3 }
```

Accéder en précisant le nom et le mode :

```
1 public void sauvegarderScore() {  
2     SharedPreferences preferences = context.  
        getSharedPreferences("toto", Context.MODE_PRIVATE);  
3 }
```

L'utilisation d'autres modes que `MODE_PRIVATE` est découragé depuis l'API 17 (4.2) pour des raisons de sécurité.

SharedPreferences, utilisation des préférences

Lire les préférences :

```
1 public void afficherMeilleurScore() {  
2     SharedPreferences preferences = PreferenceManager.  
        getDefaultSharedPreferences(context);  
3     int record = preferences.getInt("meilleurScore");  
4     Toast.makeText(context, "Record : "+record, Toast.  
        LENGTH_LONG).show();  
5 }
```

Modifier les préférences

```
1 public void sauvegarderScore() {  
2     SharedPreferences preferences = PreferenceManager.  
        getDefaultSharedPreferences(context);  
3     Editor editor = preferences.edit();  
4     editor.putInt("meilleurScore",42);  
5     editor.commit();  
6 }
```


Les fichiers

- API classiques de Java (File, InputStream, OutputStream ...)
- Ne jamais utiliser de chemin absolu, utiliser les fonctions android pour déterminer les dossiers
- Garder en tête les contraintes du mobile : espace libre, SD accessible
- Choisir entre stockage interne et externe

Stockage interne

```
1 FileOutputStream fos = context.openFileOutput("monFichier.  
   ext", Context.MODE_PRIVATE);  
2 fos.write("texte".getBytes());  
3 fos.close();
```

- Comme pour les préférences, ne pas utiliser les modes WORLD_READABLE et WORLD_WRITEABLE
- Les fichiers créés sur le stockage interne sont liés à l'application (en particulier, ils sont supprimés lors de la désinstallation de l'application)
- getCacheDir() : répertoire pour le cache
- getFilesDir() : dossier des fichiers de l'application
- deleteFile(String nom)
- fileList() : String[] des fichiers créés par l'application

Stockage externe

- La jungle : l'utilisateur et toutes les applications peuvent lire / écrire tout le contenu
- La présence d'un stockage externe n'est pas garanti
- Même s'il est présent, le stockage externe peut ne pas être disponible (appareil connecté au PC)
- 2 permissions : WRITE_EXTERNAL_STORAGE et READ_EXTERNAL_STORAGE

```
1 String state = Environment.getExternalStorageState();
2 if (Environment.MEDIA_MOUNTED.equals(state)) {
3     //Lecture et ecriture possibles
4 }
5 else if (Environment.MEDIA_MOUNTED_READ_ONLY.equals(state))
6     {
7     //Lecture possible , ecriture impossible
8 }
9 else { //Lecture et ecriture impossibles }
```

Stockage externe, utilisation

Pour le contenu utilisé uniquement par l'application :

```
1 File file = new File(getExternalFilesDir( null ), " DemoFile .  
    jpg" );
```

Ces fichiers seront supprimés lors de la désinstallation de l'application

Pour le contenu destiné à être partagé (donc persistant même après une désinstallation) :

```
1 File file = new File(getExternalStoragePublicDirectory(  
    Environment.DIRECTORY_PICTURES ), " DemoFile .jpg" );
```

L'argument utilisé dans les 2 cas correspond au type de données et est une variable static de Environment

Exemples : Environment.DIRECTORY_PICTURES,
DIRECTORY_MUSIC, DIRECTORY_DOWNLOADS,
DIRECTORY_DCIM ...

SQLite, le moteur de base de données portable

- SQLite est un moteur de base de données spécialement conçu pour le mobile et l'embarqué
- On retrouve à peu près l'ensemble des fonctions de base d'un moteur de BDD
- Android propose en plus une API facilitant les tâches courantes : SQLiteOpenHelper
- Pour les requêtes, android offre une API proche de JDBC

Mise en place d'une base de données SQLite

Etape 1 : étendre SQLiteOpenHelper

```
1 public class MyOpenHelper extends SQLiteOpenHelper {  
2  
3     private static final int DATABASE_VERSION = 1;  
4     private static final String DATABASE_NAME = "mabase";  
5  
6     public MyOpenHelper(Context context) {  
7         super(context, DATABASE_NAME, null, DATABASE_VERSION  
8     )  
9 }  
}
```

Etape 2 : implémenter onCreate

```
1 @Override  
2 public void onCreate(SQLiteDatabase db) {  
3     db.execSQL("CREATE TABLE exemple (nom TEXT PRIMARY  
4         KEY, score INTEGER)");  
}
```

Mise en place d'une base de données SQLite

Etape 3 : Organiser les mises à jour de la base

```
1 public class MyOpenHelper extends SQLiteOpenHelper {  
2     @Override  
3     public void onUpgrade(SQLiteDatabase db, int oldVersion,  
4         int newVersion) {  
5         //Mise a jour de la base si besoin  
6     }  
}
```

- L'entier DATABASE_VERSION permet de savoir si une mise à jour de la base est nécessaire
- Android appelle directement onUpgrade si le numéro de version est supérieur au numéro de version actuel de la base
- onUpgrade doit alors faire les mises à jour qui s'imposent en fonction des entiers oldVersion / newVersion

Démo

Démo

Utilisation de SQLite

Récupérer une instance de SQLiteDatabase

```
1 SQLiteOpenHelper helper = new SQLiteOpenHelper(context);  
2 SQLiteDatabase writableDB = helper.getWritableDatabase();  
3 SQLiteDatabase readableDB = helper.getReadableDatabase();
```

- Android propose un certain nombre de fonctions utilitaires pour le requêtage, l'insertion, la mise à jour et la suppression de données
- Penser à fermer la base avec `.close()` pour libérer des ressources

Utilisation de SQLite, les requêtes

- 2 méthodes au choix en fonction du besoin
- Ces méthodes sont déclinées en de multiples méthodes (surcharge)
- `rawQuery(String sql, String[] selectionArgs)` pour du sql dur (~ PreparedStatement)
- `query(String table, String[] columns, String selection, String[] selectionArgs, String groupBy, String having, String orderBy)` pour que android génère le sql

```
1 SQLiteDatabase writableDB = helper.getWritableDatabase();
2 Cursor cursor1 = writableDB.rawQuery("SELECT nom, score FROM
    table WHERE id =?", new String[] {"1"});
3 Cursor cursor2 = writableDB.query("table", new String[] {"nom",
    "score"}, "id =?", new String[] {"1"}, null, null, null
    );
```

Utilisation de SQLite, les cursors

- Wrapper autour d'un ResultSet
- Utilisation très proche des resultSets
- Penser à le fermer (close()) pour libérer des ressources

```
1 int nbRows = cursor.getCount();  
2 while (cursor.moveToNext()) {  
3     String nom = cursor.getString(0);  
4     int score = cursor.getInt(1);  
5 }  
6 cursor.close();
```

Utilisation de SQLite : update, insert et delete

long insert (String table, String nullColumnHack, ContentValues values)

```
1 ContentValues values = new ContentValues();
2 values.put("nom", "Bob");
3 values.put("score", 42);
4 long rowID = bdd.insert(table, null, values);
```

int update(String table, ContentValues values, String whereClause, String[] whereArgs)

```
1 ContentValues values = new ContentValues();
2 values.put("score", 43);
3 int nbRowsAffected = bdd.update(table, values, "nom = ?", new
    String[] { "Bob" });
```

int delete (String table, String whereClause, String[] whereArgs)

```
1 int nbRowsAffected = bdd.delete(table, "nom = ?", new String
    [] { "Bob" });
```

Démo

Démo

Sommaire

1 Présentation et objectifs du cours

- Organisation
- Contexte et objectifs

2 Le développement mobile

- Spécificités du développement mobile
- Présentation des différents OS mobile

3 Le développement sur Android

- Mise en place
- Architecture
- IHM
- Les données
- **Bonus**

La propriété `Layout_weight` de `LinearLayout`

- Possibilité d'assigner des “importances” aux views d'un `LinearLayout`
- Attribut `layout_weight` défini dans les fils
- Les views occupent l'espace restant au prorata de leur importance
- Si `layout_weight` n'est pas défini, il vaut 0

La propriété `Layout_weight` de `LinearLayout`, exemple

```
1 <LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
2     android:layout_width="match_parent"
3     android:layout_height="match_parent" >
4     <Button
5         android:layout_width="wrap_content"
6         android:layout_height="wrap_content"
7         android:text="1"
8         android:layout_weight="1" />
9     <Button
10        android:layout_width="wrap_content"
11        android:layout_height="wrap_content"
12        android:text="2"
13        android:layout_weight="2" />
14    <Button
15        android:layout_width="wrap_content"
16        android:layout_height="wrap_content"
17        android:text="3" />
18 </LinearLayout>
```


La propriété `Layout_weight` de `LinearLayout`, résultat



- Toutes les vues prennent la place dont elles ont besoin
- On calcule la place restante
- On répartit au prorata des poids
- Exemple notable : si une seule vue a un poids, elle prend toute la place restante

Exemples de ViewGroups : RelativeLayout

Plus évolué que LinearLayout, permet de placer les composants les uns par rapport aux autres

```
1 <RelativeLayout xmlns:android="http://schemas.android.com/
  apk/res/android"
2     android:layout_width="match_parent"
3     android:layout_height="match_parent" >
4     <EditText
5         android:id="@+id/text1"
6         android:layout_width="wrap_content"
7         android:layout_height="wrap_content"
8         android:layout_alignParentLeft="true"
9         android:layout_alignParentTop="true" />
10    <Button
11        android:layout_width="wrap_content"
12        android:layout_height="wrap_content"
13        android:layout_alignParentLeft="true"
14        android:layout_below="@+id/text1" />
15 </RelativeLayout>
```

Exemples de ViewGroups : ScrollView

Englobe une view et affiche une scroll bar si la view est trop grande

```
1 <ScrollView xmlns:android="http://schemas.android.com/apk/res/android"
2     android:layout_width="match_parent"
3     android:layout_height="match_parent"
4     android:orientation="vertical" >
5     <TextView
6         android:id="@+id/TextView01"
7         android:layout_width="wrap_content"
8         android:layout_height="wrap_content"
9         android:text="@string/longlonglongtexte" >
10     </TextView>
11 </ScrollView>
```

Exemples de ViewGroups : ListView

- Pattern ultra classique des applications mobile
- Affichage d'un nombre variable d'éléments avec, si besoin, une scroll bar
- Récupération des events (press, longpress ...) sur les éléments de la liste
- Chaque élément est affiché dans une view différente à l'intérieur de la listview



ListView : les adapters

- Les adapters font le lien entre les données et l'affichage (view)
- Un adapter hérite de BaseAdapter
- 2 implémentations classiques :
- ArrayAdapter pour les données sous forme d'array / list
- (Simple)CursorAdapter pour les données sous forme de Cursor

Extrait de la classe abstraite BaseAdapter :

```
1 public abstract View getView(int position, View convertView,
   ViewGroup parent);
2 public abstract long getItemId(int position);
3 public abstract Object getItem(int position);
4 public abstract int getCount();
```

ListView : exemple d'utilisation d'ArrayAdapter

```
1  ListView listView = (ListView) findViewById(R.id.mylister);
2  String[] values = { "ligne1", "ligne2", "ligne3" };
3
4  // 1) Context
5  // 2) Layout à utiliser pour chaque view
6  // 3) ID de la textview à l'intérieur du layout
7  //(optionnel, par défaut : android.R.id.text1)
8  // 4) Liste / tableau des données
9  ArrayAdapter<String> adapter = new ArrayAdapter<String>(this
10      ,
11      android.R.layout.simple_list_item_1, android.R.id.text1,
12      values);
13
14  listView.setAdapter(adapter);
15  //Manipulation des données
16  adapter.add("ligne 4");
17  adapter.clear();
```

ListView : écouter les événements

Les événements comme les clicks sont gérés par la listview (et non pas par l'adapter)

```
1 listView.setOnItemClickListener(new OnItemClickListener() {  
2     public void onItemClick(AdapterView<?> parent, View view,  
3         int position, long id) {  
4         //position, id et view concernes  
5         //parent == listview  
6     }  
7 });
```

Démo

Démo

Créer sa propre view

- Dans certains cas, les views de base d'android ne suffisent pas
- Exemple : création d'un jeu, création d'un color picker ...
- On peut alors chercher si d'autres développeurs n'ont pas déjà créé des views semblables
- ex : <http://androidviews.net>, <http://google.fr>
- ou créer ses propres views en héritant de View ou SurfaceView
- View : view classique, rafraichissement au besoin
- SurfaceView : view "temps réel" (pour un jeu par exemple)

Créer sa propre view

Etape 1 : hériter de View

```
1 public class CustomView extends View {  
2  
3     public CustomView(Context context) {  
4         super(context);  
5     }  
6  
7     public CustomView(Context context, AttributeSet attrs) {  
8         super(context, attrs);  
9     }  
10  
11     public CustomView(Context context, AttributeSet attrs,  
12         int defStyle) {  
13         super(context, attrs, defStyle);  
14     }  
15 }
```

Créer sa propre view

Etape 2 : implémenter onDraw

```
1 public class CustomView extends View {  
2  
3     @Override  
4     protected void onDraw(Canvas canvas) {  
5         super.onDraw(canvas);  
6         canvas.drawColor(Color.BLUE);  
7         int largeur = canvas.getWidth();  
8         int hauteur = canvas.getHeight();  
9         //canvas.drawString  
10        //canvas.drawCircle  
11        //canvas.drawBitmap  
12    }  
13 }
```

Previously on Android, S01E02

- Une application = un ensemble de composants définis dans le manifest
- Composants = Activity, Service, BroadcastReceiver, ContentProvider
- L'organisation des views pour l'interface graphique se fait en XML dans les fichiers layout
- Manipuler des views = leur donner un identifiant et appeler findViewById

Previously on Android, S01E02

- Intents = messages inter-composants / inter-applications
- Intent implicite = une action à faire (ex : envoyer un mail, afficher les contacts), android cherche les composants capables de répondre
- Android filtre en fonction des intent-filters définis dans le manifest
- Intent explicite = on invoque un composant particulier
- Les intents peuvent avoir des données “Extra”

Previously on Android, S01E02

- ListView : View affichant les données sous forme de liste scrollable
- Les adapters sont chargés de faire le lien entre les données et la ListView
- Créer un adapter = hériter de BaseAdapter
- BaseAdapter = 4 méthodes (getCount, getItem, getItemId, getView)
- Pour getView, définir un layout pour les items et l'inflater

Le réseau

- La qualité de la connexion est TRES variable
- L'utilisation du réseau est TRES consommateur de batterie
- Api : `java.net.HttpURLConnection`
- De nombreuses bibliothèques sont disponibles (robospice, android-async-http ...)
- Pour les autres protocoles : utiliser la classe `Socket` de `Java.net`

Le réseau, zoom sur HTTP

/!\ Ne pas oublier de déclarer l'utilisation de la permission
INTERNET dans le manifest /!\

- HTTP : protocole web (exemple : navigateurs)
- Fonctionnement simple : requête du client => réponse du serveur
- GET pour visualisation, POST pour modification des données

```
1 URL url = new URL("http://www.ensai.com/");
2 HttpURLConnection urlConnection = (HttpURLConnection) url.
  openConnection();
3 try {
4     BufferedInputStream in = new BufferedInputStream(
5         urlConnection.getInputStream());
6     readStream(in);
7 } finally {
8     urlConnection.disconnect();
9 }
```


Les webservices

- Une pratique courante est de créer des webservices pour échanger les informations entre client (mobile) et serveur
- Un webservice ne renvoie pas de HTML (données + présentation) mais des données brutes formatées (en XML, JSON ...)
- L'application mobile se charge de la mise en forme et de l'affichage des données
- 1 webservice pour toutes les applications, peu importe l'OS
- Webservices publics (météo, transports ...)
- Créer son propre webservice en Java (jersey), en PHP ...

Les webservices, format des données

JSON :

```
1  {"events": [  
2  {  
3    "debut": 1360053900000,  
4    "nom": "Introduction a la surete de fonctionnement –  
5          Perrine BROY",  
6    "salle": "110*",  
7    "uid": "1129-3463-52825-52848",  
8    "fin": 1360080900000  
9  },  
10 {  
11   "debut": 1360242000000,  
12   "nom": "Filtrage lineaire et non lineaire M263 – Francois  
13         LE GLAND",  
14   "salle": "209*",  
15   "uid": "263-370-53424-53435",  
16   "fin": 1360253700000  
17 }  
18 ]  
19 }
```

Les webservices, format des données

XML :

```
1 <events>
2 <event debut="1360053900000"
3   nom="Introduction a la surete de fonctionnement – Perrine
   BROY"
4   salle="110*"
5   uid="1129-3463-52825-52848"
6   fin="1360080900000"/>
7
8 <event debut="1360242000000"
9   nom="Filtrage lineaire et non lineaire M263 – Francois LE
   GLAND"
10  salle="209*"
11  uid="263-370-53424-53435"
12  fin="1360253700000"/>
13 </events>
```

Les webservices, parser les données

JSON :

```
1 JSONObject json = new JSONObject("{exemple:\"42\"}");  
2 json.getInt("exemple"); //42
```

XML :

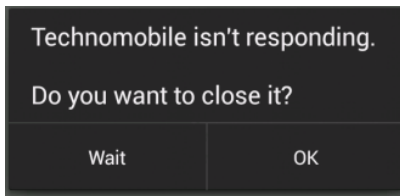
```
1 XmlPullParser parser = XmlPullParserFactory.newInstance();  
2 parser.setInput(new StringReader("<exemple>42</exemple>"));  
3 int event = parser.getEventType();  
4 while (event != XmlPullParser.END_DOCUMENT) {  
5     parser.next();  
6     event = parser.getEventType();  
7 }
```

Le multithreading

- Problématique : Les tâches à exécuter peuvent parfois être (très) longues
- Exemples : longs calculs
- Flux réseau (ping de 10-20 secondes sur réseau mobile?)
- Insertions massives en base de données
- “Solution” : l’asynchrone
- Nouveau problème : comment mettre en place l’asynchrone?

Le multithreading, constater le problème

```
1 public void calculLong() {  
2     //simule un calcul long en ne faisant rien pendant 30  
3     secondes  
4     Thread.sleep(30*1000);  
5 }
```



- L'interface graphique ne répond plus
- Si le blocage dure plus de x secondes, android affiche ce popup
- Il s'agit d'une ANR (application non responding)
- Equivalent du "ne répond pas" sous windows

Le multithreading, notion de thread

- Thread = fil
- Fil d'exécution du code
- Un thread n'exécute qu'une instruction à la fois
- Si du code est long, il bloque le thread
- Java s'occupe de répartir les threads sur les coeurs du CPU

Le multithreading, le thread UI

- Toutes les instructions de l'IHM s'exécutent dans le même thread
- **Le thread UI (main thread) doit être préservé de tout calcul long**
- **Inversement, tout le code touchant à l'UI doit être exécuté dans le thread UI**
- Le police du fun : StrictMode

Le multithreading, créer des threads

Etape 1 : créer un runnable avec le code à exécuter

```
1 Runnable code = new Runnable() {  
2     public void run() {  
3         //code ici  
4     }  
5 };
```

Etape 2 : lancer un thread exécutant ce runnable

```
1 new Thread(code).start();
```

/!\ Ne pas confondre start et run /!\

Le multithreading, astuces sous android

Astuce 1 : Les AsyncTasks

- Gère le cycle de vie d'une tâche et exécute chaque méthode dans le bon thread
- `onPreExecute` : thread UI
- `doInBackground` : thread dédié
- `onPostExecute` : thread UI

Le multithreading, astuces sous android

Astuce 2 : Exécuter du code dans le thread UI depuis un autre thread

- Pour communiquer entre threads on utilise la classe Handler et on envoie des “messages”
- Méthode pratique : `runOnUiThread(Runnable)` de Activity
- Méthode pratique : `post(Runnable)` de View

Validation des groupes / idées de projet

Pensez à faire valider vos idées de projet et les compositions de groupe avant le 9 mai

Email : ungawa14@gmail.com