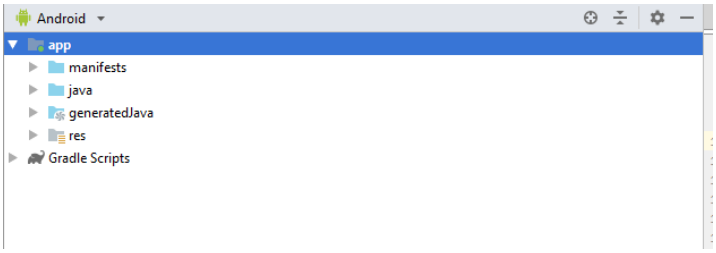
Android

Introduction

Android est une couche d'interface graphique en Java, assez similaire à Swing dans sa logique. On ne reviendra donc pas sur ce qu'est une interface graphique.

Comme on le sait, Android tourne sur des mobiles ou des tablettes. Comme la mémoire et « l'espace disque » sont limités, on utilise une JVM optimisée pour tenir compte de cela (Dalvik). Il est donc vivement conseillé de ne pas concevoir des applications trop énormes et/ou trop gourmandes en mémoire.

La principale différence entre Swing et Android, c'est que les interfaces sont décrites dans des fichiers XML tandis que le traitement est lui assuré par des Class Java classiques.



Ici, on voit qu'il y a 3 dossiers principaux :

- java : contient le code Java
- res : contient les fichiers XML
- manifests : contient le fichier de description de l'application.

Avant de continuer, voyons le cycle de vie d'une application Android :
http://supertos.free.fr/supertos.php?page=1076

Premier exemple

On va utiliser Android Studio et non plus Eclipse pour développer nos applications Android. En effet, le plugin Eclipse n'est plus vraiment maintenu tandis que Android Studio (AS) est, lui, proposé par Google et est devenu un standard de fait. L'ennui, c'est que AS est basé sur IntelliJ, ce qui va réclamer un petit travail d'adaptation.

Lançons donc AS. La première fois, il va télécharger la moitié de la Terre, c'est très long, patientez, et tout ira bien en cliquant sur « Suivant ».

Comme type de Projet, choisissez « Basic Activity ». Attendez qu'il ait fini de mouliner. Il peut arriver que des erreurs apparaissent : il y a encore des trucs à télécharger : resynchroniser avec Gradle et priez très fort.

Au bout du compte, on va lancer ce programme. Il faut savoir une chose : il y a 2 moyens pour tester un programme Android :

- Avec un émulateur. Ca peut être très lent si votre machine n'est pas performante.
- Avec un vrai mobile Android. C'est la solution la plus rapide et en plus on peut tester « en vrai ». Branchez donc votre mobile via un câble USB sur votre PC, paramétrez ce qu'il faut sur le mobile (sur un Samsung : Options de développement > Débogage USB).

Mais avant voyons un peu à quoi ça ressemble.

- Dans *java*, on a en fait une seule Class (MainActivity). Cette Class est une Activity, ce qui est le point central en Android. En gros Activity = Fenêtre. Donc pour chaque fenêtre de votre Application, on a une Activity.
- Dans *res*, on voit :
  - Le dossier *layout* : contient le XML de l'Activity (c.a.d la définition de la fenêtre). Ici on a deux Layout, parce qu'on a un Layout commun à toutes les Activity (*activity_main.xml*) qui contient la Toolbar et le FloatButton et le layout de la fenêtre elle-même (*content_main.xml*)
  - Le dossier *values* qui contient diverses données, dont entre autres :
    - *strings* : les chaines externalisées de l'application
  - Divers autres dossiers dont *menu.xml* qui contient les menus apparaissant dans l'application (ici, on a que le menu système)
- Dans *manifests*, on trouve le manifest, c.a.d le fichier décrivant l'application. Le point important ici est que MainActivity y est déclarée. TOUTES les Activity doivent être déclarées dans le manifest.

Lançons le programme : merveilleux, il apparaît « Hello world ».

Regardons un peu le code de l'Activity :

```
Bon, comme vu dans le cycle de vie, c'est la fonction qui est appelée lorsque l'Activity est crée.
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    Génial : on associe l'Activity au Layout dans le XML. R.layout.activity_main est l'id du layout en
question.
    setContentView(R.layout.activity_main);
    Ici on récupère la Toolbar par son id et on l'active.
    Toolbar toolbar = findViewById(R.id.toolbar);
    setSupportActionBar(toolbar);
    On ne s'en occupe pas trop, parce qu'on va l'enlever.
    FloatingActionButton fab = findViewById(R.id.fab);
    fab.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View view) {
            Snackbar.make(view, "Replace with your own action", Snackbar.LENGTH_LONG)
                    .setAction("Action", null).show();
        }
    });
}
```

Un peu plus loin, on voit le code qui permet d'afficher le menu système

Le point le plus important, c'est que TOUS les objets qui se trouvent dans les fichiers XML sont accessibles via R.type_de_ressource.id. C'est un point fondamental et une des bases d'Android. A noter que certaines ressources internes à Android sont aussi accessibles sur ce modèle. Les chaines aussi sont accessibles comme ceci, mais elles sont généralement accessibles dans les fichiers XML eux-mêmes via une autre syntaxe (@string/id_string). Je vous laisse découvrir tout ça.

On a n'a vu ici que le surchargement de onCreate(). C'est ce qu'on fait le plus souvent. Mais si on se souvient du cycle de vie, on peut surcharger d'autres méthodes (onDestroy(), par exemple). Un bon exemple de surchagement, est de surcharger onBackPressed() qui est appelée lorsqu'on appuie sur le bouton Back. En effet dans l'Activity principale, cela fait sortir de l'application. Il est possible de positionner une Dialog Yes/No pour confirmer cette sortie.

Maintenant qu'on a compris tout ça, on fait l'exercice 1. Et l'exercice 2 dans la foulée.

Déploiement

Normalement, les applications Android sont déployées sur le PlayStore de Google. Dans notre cas, comme il s'agit d'un test, on ne peut évidemment pas employer cette méthode.

Déjà, il faut comprendre que les applications Android sont des applications Java sur le fond, et, donc, sont générées au final sous forme d'un jar. Sauf qu'en Android, ce n'est pas un jar mais un apk, qui est grosso-modo la même chose (au final c'est une sorte de zip).

Quand on déploie une application Android, on copie donc ce fichier apk sur un serveur (le PlayStore), et les smartphones n'ont plus qu'à le télécharger. Pour déployer en test, on va donc suivre les étapes suivantes :

- Désinstaller l'application en question sur votre smartphone (AS l'installe par défaut)
- Générer un apk avec AS (par défaut, l'appli s'appelle *app-debug.apk*)
- Recopier ce fichier sur un serveur web. Une solution facile est d'employer *dl.free.fr*. Quand le fichier est déposé sur le serveur de free, un mail vous est envoyé avec le lien sur le fichier.
- Sur le smartphone, cocher *Sources Inconnues* (par défaut seul PlayStore est accepté) pour autoriser le téléchargement d'apk depuis n'importe où.
- Cliquer sur le lien de dl.free.fr : l'application s'installe. Il ne reste plus qu'à l'ouvrir et vérifier que tout est ok.

Un peu d'interface

En fait l'interface est composée de deux parties :

- Les différents objets graphiques (boutons, listes, images, etc)
- Le layout, c.a.d la façon dont les objets graphiques s'agencent à l'écran. On avait déjà cette notion en Swing.

Pour ce qui est des Layouts, il en existe un certain nombre (comme dans Swing), mais comme AS propose *ConstraintLayout* par défaut, c'est lui qu'on va utiliser. Il s'agit en gros d'un Layout qui dans lequel les objets graphiques se positionnent vis-à-vis des bords de l'écran. Il est assez laborieux à utiliser, mais c'est aussi le plus courant.

Les autres Layout sont plutôt utilisés comme Layout à l'intérieur de composants Android (comme les LinearLayout).

Les composants sont comme leur nom l'indique des composants graphiques. En Android, ce sont des *View*, ce qui signifie qu'ils dérivent tous de la Class View.

On a :

- TextView : Texte statique (label)
- EditText : Champ d'édition (avec une série de dérivés spécialisés : Password, email, etc).
- Button
- ListView : une liste
- Plein d'autres

Tous ces éléments sont donc définis dans du XML et au sein d'un Layout donné (on peut évidemment mettre des Layout dans des Layout). Pour attribuer un id à une View, on fait @+id/nom_id dans le XML.

Le plus simple pour bien comprendre est d'utiliser le designer de AS et de regarder le code XML généré.

A noter que les composants sont repérés par des unités de mesure particulières dans les fichiers XML :

- dp/dip (Density Independant Pixel) : Echelle indépendante de l'écran. Le plus utilisé.
- sp (Scale-independant Pixel) : Similaire au dp mais mise à l'échelle aussi par rapport à la taille de la police définie dans les préférences systèmes. Conseillé pour les tailles de police.
- pt : 1/72 d'un pouce basé sur la taille de l'écran physique.

Eviter d'utiliser des mesures physiques, comme le px (pixel).

Les évènements

Maintenant, qu'on a des View (composants graphiques), il faut maintenant qu'il y ait interaction. Par exemple que l'on soit tenu au courant si on clique sur un bouton, on qu'on sélectionne un élément dans une ListView.

Et, comme en Swing, on fait ça avec des Listeners.

Il existe énormément de Listeners différents, mais on va s'occuper de 2 listeners assez fréquents, le fonctionnement est globalement le même pour tous les Listener.

Listener de click

Typiquement pour un Button mais pas nécessairement.

L'idée est de récupérer la View dans du code Java (généralement dans une Activity), et de lui attacher un Listener.

```
    On récupère le Button sous forme de View depuis le XML grâce à la fonction magique findViewById()
    Button b = findViewById(R.id.button);
    On lui positionne un OnClickListener
    b.setOnClickListener(new View.OnClickListener() {
        Il faut définir la fonction onClick dans un OnClickListener
        @Override
        public void onClick(View v) {
            Le Toast permet d'afficher un message pendant 2 secondes. A remarquer : quand une fonction
a un Context comme paramètre, il s'agit en fait de l'Activity
            Toast.makeText(MainActivity.this,R.string.hello,Toast.LENGTH_LONG).show();
        }
    });
```

Listener de click dans une liste

Typiquement dans une ListView

On fait la même chose avec un OnItemClickListener.

```
ListView list = findViewById(R.id.liste);
list.setOnItemClickListener(new AdapterView.OnItemClickListener() {
    @Override
    On verra plus tard comment ça marche.
    public void onItemClick(AdapterView<?> parent, View view, int position, long id) {

    }
});
```

Et on fait l'exercice 3

Les Popup

En plus du Toast, on peut afficher des Popup classiques avec du texte, des images, des boutons, etc …

Il existe différentes Popup dérivant de *Dialog* ; on va voir le cas de *AlertDialog*.

```
On crée le Builder
AlertDialog.Builder builder = new AlertDialog.Builder(this);
On positionne le titre, le message et d'autres trucs éventuels
builder.setTitle(R.string.dlgTitle).setMessage(R.string.dlgMsg);
On positionne le bouton positif (Yes)
builder.setPositiveButton(R.string.yes, new DialogInterface.OnClickListener() {

   @Override
    public void onClick(DialogInterface dialog, int which) {
        dialog.dismiss();
        …
    }
});
On positionne le bouton négatif (No)
builder.setNegativeButton(R.string.no, new DialogInterface.OnClickListener() {
    @Override
    public void onClick(DialogInterface dialog, int which) {
        dialog.dismiss();
    }
});
On crée la Popup
AlertDialog dlg = builder.create();
On l'affiche
dlg.show();
```

Et on fait l'exercice 4

Où l'on reparle des ListView

Les ListView sont donc une des View qui présente une liste de données. On voit bien que chaque ligne de la View est elle-même une View.

Tout cela ressemble fort au Swing. Rappelez-vous : en Swing, on avait la List et on lui associait un Model (généralement un ArrayList), ce Model consistant en les données à afficher.

Avec la ListView, c'est un peu pareil, sauf qu'on associe pas le Model directement, mais via un Adapter, qui est en plus un Design Pattern.

Pour une ListView il y a 3 types d'Adapter possibles :

- ArrayAdapter (travaille à partir de array ou de List)
- SimpleCursorAdapter pour les BDD (qu'on ne verra pas ici)
- On peut aussi implémenter son propre Adapter en implémentant ListAdapter (Toujours comme dans Swing)

Résumons-nous : pour afficher quelque chose dans une ListView, il faut

- Des données (sous forme de array ou de List, disons)
- Créer un Adapter (disons un ArrayAdapter)
- Associer cet Adapter à la liste

Prenons un exemple :

```
On charge une liste avec quelque chose
List list = …
On crée l'adapter
ArrayAdapter adapter = new ArrayAdapter<?>( this, android.R.layout.simple_list_item_1, list);
On l'associe à la ListView
listView.setAdapter(adapter);
```

Et si on en revient au Listener de la ListView, on récupère l'objet lorsqu'on sélectionne sur une ligne particulière de la façon suivante :

```
public void onItemClick(AdapterView<?> parent, View view, int position, long id) {
    MonObjet o = (MonObjet)parent.getAdapter().getItem(position);
```

A noter que si on a un ArrayAdapter généricisé (<MonObjet>), ce qui s'affiche dans la ListView, c'est MonObject. toString().

Et on fait l'exercice 5

Navigation entre écrans

Jusqu'à présent, on n'affiche qu'une seule page. Ce qui est un peu limité comme application.

Voyons donc comment appeler une Activity depuis une Activity et donc passer de fenêtre en fenêtre.

La première chose dont il faut se souvenir, c'est que si vous créez une autre activité, il faut la déclarer dans le manifest.

Pour cela, il faut créer un Intent de la façon suivante :

```
On crée l'Intent. Attention : appelant est l'Activity courante et appelé est la classe de l'Activity qu'on appelle.
Intent intent = new Intent().setClass(appelant,appelé);
Eventuellement, on passe des données à la nouvelle Activity
intent.putExtra("MY_DATAS", datas);
On démarre la nouvelle activité
startActivity(intent);
```

C'est donc vraiment facile !

Dans la nouvelle Activity, on peut éventuellement récupérer les données passées de la façon suivante :

```
MonObjet o = (MonObjet) getIntent().getSerializableExtra("MY_DATAS");
```

Mais ce n'est pas tout : on peut aussi utiliser les Intent pour, par exemple, ouvrir le browser sur une url donnée.

Ça donnerait quelque chose comme ça :

```
Intent i = new Intent(Intent.ACTION_VIEW);
i.setData(Uri.parse(url));
startActivity(i);
```

On peut aussi lancer une autre Activity en récupérant un résultat dans l'Activity appelante. Si Activity1 appelle Activity2, L'opération est la suivante :

- Dans Activity1, on fait :

```
Intent it = new Intent().setClass(MainActivity.this,Detail.class);
…
startActivityForResult(it,DETAIL_CODE); (ici DETAIL_CODE est le code de requête > 0)
```

- Dans Activity2, une fois le travail effectué, on fait :

```
Intent returnIntent = new Intent();
…
setResult(Activity.RESULT_OK,returnIntent); (RESULT_OK est une valeur interne à Android, on peut aussi
utiliser RESULT_CANCELLED)
```

- Dans Activity1, on capte le retour de Activity2, en surchargeant la fonction *onActivityResult*

```
@Override
protected void onActivityResult(int requestCode, int resultCode, @Nullable Intent data) {
    super.onActivityResult(requestCode, resultCode, data);
    if (requestCode == DETAIL_CODE) {
        if (resultCode == RESULT_OK) {
…
```

Utiliser un WebService JSON

Il est plus qu'intéressant de se connecter à un WebService pour ses données. Pour cela, et avant toute chose, il faut dire à l'application qu'elle a l'autorisation d'accéder à internet. Il faut donc ajouter dans le manifest :

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

Ceci étant, comme on l'a déjà vu, un client WS a besoin de deux library :

- Une pour gérer le HTTP. Ici on prendra HttpClient. Elle est deprecated, mais pour les vieux smartphones, comme le mien, OkHttp ne fonctionne pas.
- Pour le JSON, on peut utiliser JsonObject et JsonArray (fourni avec Android). Ou Jackson, si on se sent pour ça.

Pour HttpClient, il faut déclarer dans gradle :

```
android {
    compileSdkVersion 28
    useLibrary 'org.apache.http.legacy'
```

Sinon typiquement ça marche comme ça :

```
DefaultHttpClient httpClient = new DefaultHttpClient();
HttpGet getRequest = new HttpGet(url);
HttpResponse response = httpClient.execute(getRequest);
if (response.getStatusLine().getStatusCode() != 200) {
  // erreur
}
BufferedReader br = new BufferedReader(
        new InputStreamReader((response.getEntity().getContent())));
```

Pour Json, imaginons qu'on ait une chaine JSON dans la variable *ret*. On la décoderait comme ça :

```
JSONArray jsonArray = new JSONArray(ret);
for (int i = 0; i < jsonArray.length(); i++) {
    JSONObject jsonObj = jsonArray.getJSONObject(i);
    JSONObject jsonSubObj = jsonObj.getJSONObject("subObject");
    String truc = jsonObj.getString("champ1")
```

Mais les ennuis ne sont pas terminés.
En effet, en Android, il est interdit de lancer des requêtes réseau dans le même thread que l'Activity (pour éviter que des traitements trop longs ne viennent perturber l'affichage).
Il faut donc faire la requête HTTP dans un thread à part.
Oui, mais …. Puisque la requête devient asynchrone, comment sait-on qu'elle est terminée (pour par exemple afficher les données récupérées) ?
Eh bien …. Il faut que le thread HTTP envoie un message au thread principal.
Ca marche comme ça :

- D'abord on crée un handler. Un Handler est qchose qui reçoit des messages. Créons donc le Handler :

```
private Handler handler = new Handler() {
    @Override
    public void handleMessage(Message msg) {
        if (msg.obj != null) {
           // on traite le message
```

- On a vu la partie réception du message. Pour l'envoyer, on fait quelque chose comme ça :

```
String value = …;
Message m = new Message();
m.obj = value;
handler.sendMessage(m);
```

A NOTER : On peut aussi employer *AsyncTask* au lieu d'un Thread et d'un handler.

Et on fait l'exercice 7.

Préférences

Les préférences en Android sont l'équivalent des fichiers de configuration en Java : elles contiennent les préférences qui ont été stockées via une Activity de préférences.

On a donc deux choses : le fichier de préférences (XML) et son Activity associée et d'autre part, le système permettant de récupérer ces préférences.

En général, l'Activity de préférence est accessible via le menu système et ça tombe bien parce qu'il est généré par AS par défaut. Reportez-vous au code pour voir comment ça marche.

On commence par créer une ressource de type xml. On ajoute dedans différents types de préférences. Au final, ça donne un truc comme ça :

```
<PreferenceScreen xmlns:android="http://schemas.android.com/apk/res/android">
    Cette preference est donc gérée par un champ d'édition
    <EditTextPreference
        La clé permet de récupérer la valeur de la préférence
        android:key="cle de cette preference"
        android:selectAllOnFocus="true"
        android:singleLine="true"
        Le label de la préférence
        android:title="@string/ws_url"
        android:visibility="visible" />
```

Pour créer une Activity de préférence, je vous donne le code parce que ça ne s'invente pas.

```
public class MyPrefFragActivity extends PreferenceActivity {

   @Override
   protected void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
        getFragmentManager().beginTransaction().replace(android.R.id.content, new
MyPreferenceFragment()).commit();
   }
   public static class MyPreferenceFragment extends PreferenceFragment
    {
        @Override
        public void onCreate(final Bundle savedInstanceState)
        {
            super.onCreate(savedInstanceState);
            addPreferencesFromResource(R.xml.prefs);
        }
    }
}
```

Au final, l'Activity de préférences affiche une fenêtre où l'on voit les labels des différentes préférences définies dans le fichier xml. S'il y a 5 préférences, on voit 5 lignes. En cliquant sur une ligne, on voit apparaitre le type de composant défini. Si c'est un EditTextPreference, par exemple, on voit donc un champ d'édition.

Et pour récupérer les données stockées dans les préférences, on fait :

```
PreferenceManager.getDefaultSharedPreferences ()
```

On passe une clé à cette fonction. Cette clé est celle qui est dans le fichier XML.

Et on fait l'exercice 8.

Les Sensors

Les capteurs (sensors) sont des capteurs présents sur les smartphones. Tous ne sont pas présents, ça dépend du smartphone. Le gyroscope, l'accéleromètre, le détecteur de nord magnétique sont en général présents.

`On utilise les capteurs via le SensorManager.` Lequel est récupéré via *getSystemService(**SENSOR_SERVICE**)*

On récupère le capteur via *sensorManager.getDefaultSensor(type de capteur)*

Ensuite on fait un *registerListener* sur sensorManager
Les paramètres sont :

- Un sensorEventListerner
- Le capteur
- Le délai de rafraichissement (en pratique *SensorManager.SENSOR_DELAY_UI*)

Et on fait l'exercice 9