Stefan Schwark

Android

apprendre à programmer des applis

Environnement de développement **Eclipse**
Programmation orientée objet en **JAVA**

www.elektor.fr/android

elektor

www.elektor.fr/android

Android
Apprendre à programmer des applis

Environnement de développement Eclipse
Programmation orientée objet en JAVA

Auteur : Stephan Schwark

Éditeur : Elektor
ISBN : 978-2-86661-187-3
Format : 17 × 23,5 cm
Nbre de pages : 208
Prix : 33,50 €

SOMMAIRE

Préface 7

1. Android

1.1 Histoire et développement 9
1.2 Android et Linux 11
1.3 Projets Open Source 13
1.4 Portages d'Android 14

2. Eclipse

2.1 Logiciels nécessaires 15
2.2 Appareils virtuels Android 17

3. Concepts de base de la programmation Android

3.1 Composants du système Android . . 19
3.2 Fichier manifeste d'Android 21
3.3 Niveaux d'API d'Android 22
3.4 Activités Android 24

4. Interface utilisateur d'Android

4.1 Views et ViewGroups 27
4.2 Fichiers de description XML 27
4.3 Gabarits 29
4.3.1 LinearLayout 31
4.3.2 AbsoluteLayout 32
4.3.3 TableLayout 33

4.3.4 RelativeLayout 34
4.3.5 FrameLayout. 35
4.3.6 ScrollView 36

5. Android et Java

5.1 Introduction à Java 39
5.2 Paquetages Android 42
5.3 Types de données 43
5.4 Visibilité des variables et des méthodes 44

6. Applications Android simples

6.1 Votre première application 45
6.2 Programme de calcul simple 49
6.3 Listes de choix 58
6.4 Chronomètre 71
6.5 Minuteur 77
6.6 Notifications 83
6.7 Lecture et écriture de fichiers 84
6.8 Envoi et réception de SMS 89

7. Consultation et représentation des géodonnées

7.1 Géodonnées. 97
7.2 Dessiner sur l'écran. 105
7.3 Outil GPS 110
7.4 Enregistrer une route GPS. 118
7.5 Interroger les capteurs. 126
7.5.1 Accéléromètre. 126
7.5.2 Magnétomètre. 130
7.5.3 Capteur de luminosité. 134
7.6 App Widgets 135

8. Lecture multimédia

8.1 Restitution des fichiers audio 145
8.2 Générateur de fréquences 147
8.3 Enregistrer le son 156

9. Applications pour le Web

9.1 Afficher du contenu Web 163
9.1.1 Caméra 163
9.1.2 Google Maps 165
9.1.3 Barre de progression. 166
9.2 Restitution des vidéos Internet . . .168
9.3 Applications Web en JavaScript et HTML 170
9.4 Application pour les wikis 171
9.5 Interroger une base de données via HTTP 174
9.6 Communiquer par socket 182

10. Android et Linux

10.1 Droits et utilisateurs 189
10.2 Lancer des commandes Linux . . .190

Postface 193

11. Appendices

11.1 Application HTML 195
11.2 Références. 202
11.2 Crédits des images 202
11.2 Crédits des codes source . .202
11.2 Ressources Web 202
Index 203

4. Interface utilisateur d'Android

4.1 Views et ViewGroups

Bien comprendre l'interface utilisateur d'Android est indispensable pour afficher à l'écran du texte, de l'image, ou plus généralement des informations. L'interface utilisateur comprend plusieurs classes. Les objets de ces classes représentent les éléments visuels dont se sert le système pour construire les interfaces graphiques.

La vue (*View*) est le composant graphique élémentaire d'Android. Un objet `View` représente une surface rectangulaire qui peut servir de conteneur pour d'autres objets. Un objet `View` gère lui-même les éléments de sa propre représentation ainsi que les évènements qui surviennent dans sa zone d'écran.

Les objets `View` sont placés dans l'arbre qui représente la hiérarchie des vues. Il est possible d'insérer des éléments supplémentaires dans une vue, soit en écrivant directement dans le code d'une activité, soit en utilisant un fichier de description XML externe.

`ViewGroup` est une sous-classe de la classe `View`. Les objets de cette classe peuvent contenir d'autres objets `View` ainsi que des gabarits (ou *layouts*, des conteneurs qui permettent de positionner des composants graphiques), et ainsi étendre l'arbre de la hiérarchie des vues.

L'appel d'une activité déclenche l'affichage des gabarits. L'activité invoque la méthode `setContentView()` en passant une référence au nœud supérieur de l'arbre. Android démarre depuis ce nœud et parcourt l'arbre de haut en bas en deux passes. Dans la première, Android demande les dimensions des objets contenus dans l'arborescence. La seconde passe sert au calcul de leurs positions à l'écran. Les objets qui n'appartiennent pas à une zone à afficher ne sont pas dessinés. Chaque objet enfant est ainsi responsable de sa propre représentation à l'écran.

4.2 Fichiers de description XML

La façon la plus simple et aussi la plus fréquente de créer une interface est de la déclarer dans un fichier XML. Ce fichier XML définira la structure hiérarchique des éléments graphiques qui composent l'interface. Créé en 1996, le XML (*Extended Markup Language*) est un langage qui permet de représenter le contenu des données à l'aide de

ISBN 978-2-86661-187-3

Figure 7 - La CalculOhmlette.

Créons d'abord un nouveau projet. Reprenez la procédure suivie lors de la création du projet HelloWorld précédent, mais cette fois-ci appelez le projet *Calculator*, et entrez *com.example.Calculator* dans le champ *Package Name*. Le projet apparaît dans l'explorateur de paquetages une fois terminée la configuration. L'étape suivante est la construction d'une interface pour cette CalculOhmlette.

Nous aurons besoin de champs de saisie pour entrer les valeurs nécessaires au calcul, soit trois champs, respectivement pour la tension, la résistance et l'intensité. Comme nous l'avons vu dans la précédente section 6.1, ces champs de saisie peuvent être définis dans le fichier *main.xml*. Eclipse offre également la possibilité de construire les interfaces par glisser-déposer. Ouvrez le fichier *main.xml* depuis l'arborescence du projet, puis cliquez sur l'onglet *Graphical layout* de la fenêtre d'édition si ce n'est pas la vue par défaut.

À gauche sont regroupés différents types de composants. L'interface affichée est pour le moment celle par défaut. Nous retrouvons le contenu de la chaîne *hello* de l'exemple précédent. Insérons par glisser-déposer les différents éléments qui composeront notre interface. Puisque notre CalculOhmlette attend des valeurs décimales, les trois champs de saisie dont nous avons besoin doivent être du type *Decimal*, un type qui se trouve dans la liste des éléments *Text Fields*. Positionnez-en trois dans la vue graphique. Nous avons également besoin de trois boutons, et nous avons trois textes de description (éléments *TextView*) à placer au-dessus des champs de saisie.

Si vous avez réussi à arranger les éléments dans le bon ordre, le nouveau fichier *main.xml* devrait être :

ISBN 978-2-86661-187-3

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
   android:orientation="vertical"
   android:layout_width="fill_parent"
   android:layout_height="fill_parent"
   >
   <TextView
      android:layout_width="fill_parent"
      android:layout_height="wrap_content"
      android:text="@string/resist"
   />
   <EditText
      android:id="@+id/editText1"
      android:layout_height="wrap_content"
      android:layout_width="fill_parent"
      android:inputType="numberDecimal">
      <requestFocus></requestFocus>
   </EditText>
   <TextView
      android:layout_width="fill_parent"
      android:layout_height="wrap_content"
      android:text="@string/voltage"
   />
   <EditText
      android:id="@+id/editText2"
      android:layout_height="wrap_content"
      android:layout_width="fill_parent"
      android:inputType="numberDecimal">
   </EditText>
   <TextView
      android:layout_width="fill_parent"
      android:layout_height="wrap_content"
      android:text="@string/current"
   />
   <EditText
      android:id="@+id/editText3"
      android:layout_height="wrap_content"
      android:layout_width="fill_parent"
      android:inputType="numberDecimal">
   </EditText>
   <Button
      android:text="Calculer la résistance"
      android:id="@+id/button1"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content">
   </Button>
   <Button
      android:text="Calculer la tension"
      android:id="@+id/button2"
      android:layout_width="wrap_content"
```

ISBN 978-2-86661-187-3

6.6 Notifications

L'application précédente affiche un message d'alarme et déclenche le vibreur au bout d'un temps déterminé, mais que se passe-t-il si le téléphone n'est pas à proximité de l'utilisateur à ce moment-là ? Le vibreur risque de ne pas être entendu. Voilà pourquoi nous devons également afficher un message indiquant que le temps est écoulé.

Nous nous servirons pour cela de la classe `Notification` fournie par le système Android. Cette classe permet d'afficher des messages dans la barre d'état située en haut de l'écran ou encore d'allumer la LED de notification. Complétons donc le code de l'application précédente afin qu'elle dispose de ces deux notifications :

```
package alarm.example.com;
import android.app.Notification;
import android.app.NotificationManager;
import android.app.PendingIntent;
import android.content.BroadcastReceiver;
import android.content.Context;
import android.content.Intent;
import android.os.Vibrator;
import android.widget.Toast;

public class AlarmReceiver extends BroadcastReceiver {
   @Override
   public void onReceive(Context context, Intent intent) {
      // Le gestionnaire de notification
      NotificationManager notMan = (NotificationManager) context
         .getSystemService(Context.NOTIFICATION_SERVICE);

      // Affichage d'un message d'alarme
      Toast.makeText(context, "Le temps est écoulé.", Toast.LENGTH_LONG).show();

      // Vibration du téléphone pendant 2 s
      Vibrator vibrator = (Vibrator) context
         .getSystemService(Context.VIBRATOR_SERVICE);
      vibrator.vibrate(2000);

      // Paramétrage de la notification
      String text = "Fin de la durée programmée";
      Notification notification = new Notification(
         android.R.drawable.stat_notify_sync, text, System.currentTimeMillis());
      notification.flags = Notification.FLAG_AUTO_CANCEL;
      PendingIntent contentIntent = PendingIntent.getActivity(context
         .getApplicationContext(), 0, new Intent(context
         .getApplicationContext(), AlarmActivity.class),
         PendingIntent.FLAG_UPDATE_CURRENT);
```

ISBN 978-2-86661-187-3

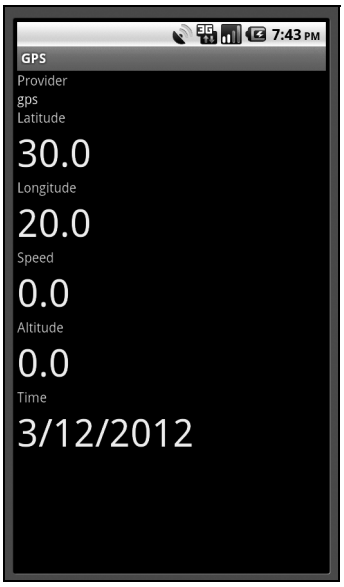

Figure 15 - L'activité GPS avec des données fictives.

Essayons de tester notre programme avec *geo fix*. Cette commande attend deux coordonnées de longitude et de latitude. Entrons par exemple :

```
geo fix 20 30
```

Les champs latitude et longitude de notre petite application GPS affichent bien 30.0 et 20.0. Comme il s'agit de la première modification de position détectée, la méthode `onLocationChanged` de l'écouteur `LocationListener` est appelée pour la première fois. Ce n'est donc qu'à cet instant que les autres champs sont remplis. Date exceptée, nous n'y trouvons cependant que des 0.0 puisqu'aucune donnée n'a été transmise. L'application évoluerait bien sûr différemment avec un vrai téléphone puisque la position serait actualisée toutes les minutes.

Nous pouvons de même transmettre une « phrase » GPS conforme à la norme NMEA 0183 utilisée par les interfaces des récepteurs GPS. Pour cela nous lançons la commande `geo nmea` suivie de la trame à transmettre. Nous ne pouvons toutefois pas nous contenter d'une trame contenant uniquement les données de latitude et de longitude ; la saisie est quelque peu fastidieuse, à moins de recourir au copier-coller d'une trame complète. Voici un exemple de commande `geo nmea` :

```
geo nmea $GPRMC,081836,A,3751.65,S,14507.36,E,000.0,360.0,130998,011.3,E*62
```

ISBN 978-2-86661-187-3

```
<wpt lat='50.81899881362767' lon='6.136915683746159'>
<ele>'243.0'</ele>
</wpt>
<wpt lat='50.81899881362767' lon='6.1369371414182785'>
<ele>'243.0'</ele>
</wpt>
<wpt lat='50.81900954246373' lon='6.1369371414182785'>
<ele>'243.0'</ele>
</wpt>
</gpx>
```

Chaque point de passage est enregistré dans un élément XML `<wpt>`. Longitude et latitude correspondent aux valeurs des attributs `lon` et `lat`. L'altitude est quant à elle contenue dans la balise `<ele>` de l'élément `<wpt>`. L'ensemble des coordonnées de cette route GPS est inséré dans la balise racine `<gpx>`. Un tel format de fichier pourra être lu par d'autres programmes, ou encore partagé sur une page Internet.

Nous pourrions de la même façon enregistrer l'un ou l'autre des nombreux éléments décrits par la spécification du format ouvert GPX. Notre exemple n'en utilise qu'un petit nombre, le minimum nécessaire à la représentation d'un tracé sous forme d'une suite de points de passage. Nous aurions même pu nous passer de l'altitude.

7.5 Interroger les capteurs

Les smartphones Android sont tous équipés de capteurs. Citons parmi ceux-ci : les capteurs photosensibles pour la commande de la caméra, les accéléromètres, les thermomètres, ou encore les magnétomètres pour l'enregistrement du champ magnétique terrestre. Bon nombre d'applications n'existeraient pas sans ces capteurs. La rotation automatique de l'écran, qui nécessite la connaissance de la position spatiale du téléphone, repose par exemple sur une mesure de l'accélération due à la pesanteur.

L'API Android permet bien évidemment d'interroger ces capteurs. Nous allons voir qu'écouter leur changement d'état est très simple, et que le *modus operandi* est peu ou prou le même pour chaque type de capteur. Les exemples suivants n'en utilisent qu'un seul à la fois, car tous les téléphones tactiles ne sont pas forcément équipés du même nombre de capteurs.

7.5.1 Accéléromètre

Commençons par l'accéléromètre, présent sur la majorité des téléphones puisque, nous l'avons dit, il est indispensable à la rotation automatique de l'affichage. L'accéléromètre mesure l'accélération selon trois axes. Ce capteur renvoie donc les

ISBN 978-2-86661-187-3

Comme toujours les textes de description des éléments `TextView` sont référencés dans le fichier de ressources *strings.xml*. Créons-le et remplissons les balises `string` des champs de texte précédents :

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
   <string name="level"></string>
   <string name="form">Forme d\'onde :</string>
   <string name="sine">Sinusoïde</string>
   <string name="square">Rectangulaire</string>
   <string name="textlevel">Niveau de sortie</string>
   <string name="textfrequency">Fréquence</string>
   <string name="frequency">Fréquence (Hz)</string>
   <string name="app_name">Générateur audio</string>
</resources>
```

La figure 16 montre l'interface produite par ces deux fichiers : deux champs de saisie pour la fréquence et le niveau de sortie, deux champs pour l'affichage des données en cours et un champ de texte pour indiquer la forme d'onde sélectionnée. Nous nous contenterons ici de deux formes d'onde, à savoir une sinusoïde et une onde rectangulaire. Dans *main.xml,* les attributs `android:text` des champs *Fréquence* et *Niveau de sortie* ont pour valeurs « -- ». Ces champs contiendront donc par défaut deux tirets lorsqu'aucune valeur n'aura encore été saisie.

Figure 16 - L'interface du générateur.

ISBN 978-2-86661-187-3

```
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>

        </activity>

    </application>

</manifest>
```

Nous avons utilisé la permission :

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

L'élément `uses-permission` possède un seul attribut, `android:name`. Sa valeur commence toujours par `android.permission` pour les permissions système, suivie du nom de la permission, ici `INTERNET`, une permission prédéfinie d'Android. Sans ce droit d'accès, le navigateur refuserait de charger le contenu demandé et le système émettrait un message d'erreur. L'utilisateur devra confirmer cette permission au moment de l'installation de l'application

Selon le débit de votre connexion, l'image prise par la caméra apparaîtra plus ou moins rapidement après le lancement de l'activité. La caméra fournit ses prises de vue au format image, l'application n'a rien d'autre à faire que de charger puis afficher le contenu de l'adresse de l'appareil. La barre d'adresse ne contient ici qu'une adresse IP car la webcam appartient à un réseau interne.

9.1.2 Google Maps

Un autre exemple intéressant exploite l'accès à l'internet : la visualisation de cartes. Pour ce prochain exemple, nous allons nous servir d'un `WebView` pour afficher sur le téléphone une des pages de démonstration de l'API *Google Maps*. Le code source de l'activité est ici encore très court :

```
package com.example.webview;
import android.app.Activity;
import android.os.Bundle;
import android.webkit.WebChromeClient;
import android.webkit.WebView;
import android.webkit.WebViewClient;
import android.widget.Toast;

public class WebviewActivity extends Activity {
    /** onCreate() est appelé au démarrage de l'activité. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
```

ISBN 978-2-86661-187-3