Épisode 4 : Premiers pas de programmation

© Alain Menu – édition 2.1, septembre 2013

Objectifs : Programmation Android (interface graphique, ressources, interception d'événements, ...)

Table des matières

Créer un nouveau projet comme précédemment par File | New | Android Application Project.
Choisir un nom, par exemple MyDemo1, et un *package name* personnalisé, comme par exemple ici com.alainmenu.android.mydemo1...
L'activité principale de l'application sera de type *blank* et nommée MyDemo1.

4.1. Icône personnalisé

La première touche personnelle consiste souvent à changer l'icône par défaut de l'application...
La troisième page de l'assistant de création de projet permet de choisir une image personnelle de base. Elle doit avoir une résolution au moins égale à 144 x 144 pixels et sera automatiquement retaillée pour les autres résolutions.

*Le look-and-feel des icônes doit respecter certaines règles qui sont exposées dans la documentation du SDK aux rubriques « **Icon Design Guidelines** » et « **Launcher Icons** »...*

L'exemple choisi ici est une image nommée llf_icon.png →

Une fois le projet créé, vérifier la présence des fichiers PNG (tous nommés ic_launcher) dans les divers sous-répertoires drawable-xxx du répertoire res.
L'assistant a aussi automatiquement généré une version 512 x 512 pixels de l'icône dans le répertoire racine...

4.2. Ajout de ressources

4.2.1. Chaines de caractères

Pour le moment, l'application se contente toujours d'afficher le traditionnel texte de bienvenue. Comme évoqué précédemment, les objets (ici de classe TextView) peuvent être paramétrés au travers du code Java ou par l'intermédiaire de ressources sous forme XML.
Les exemples qui suivent utilisent la deuxième méthode pour modifier la couleur, la taille et le style d'un texte... Les modifications peuvent être faites directement en XML ou au moyen des éditeurs graphiques.

Ajout de nouvelles chaines

Éditer le fichier strings.xml et ajouter deux chaines nommées copyright et comment. La première peut contenir la signature du développeur (du style « (c)2013 by Me »), et la deuxième un commentaire volontairement plus long tel que : « Ma première application pour Android développée avec Eclipse et le plugin ADT ! ».

La séquence "(c)" peut être inscrite dans la chaine sous sa forme HTML "©"

Projection des nouvelles chaines

Modifier activity_my_demo1.xml de manière à afficher maintenant les uns en dessous des autres le nom de l'application suivi des ressources copyright et comment. Ajouter un attribut permettant de centrer app_name.
Lorsque les modifications sont faites graphiquement, l'assistant attribue automatiquement des *id* aux objets. Les contraintes de placement d'un objet sont ensuite fixées relativement à un autre objet spécifié par son *id*.
Par exemple, l'objet copyright est contraint par l'attribut android:layout_below="@+id/textView1" à être placé directement sous l'objet d'*id* textView1 qui est ici le TextView app_name.

Attention à ne pas confondre les contraintes de placement (layout) du TextView sur l'UI avec les contraintes de placement du texte à l'intérieur de son conteneur...

Un objet peut être centré sur la fenêtre d'application par l'attribut android:layout_centerHorizontal.
Les attributs layout_width et layout_height permettent de spécifier l'adaptation d'un élément à son contenu :

- match_parent demande à l'élément de remplir totalement son parent ;
- wrap_content demande à l'élément de s'adapter à son contenu ;
- Une valeur numérique en points (*dp*) permet de forcer une dimension absolue.

Forcer la hauteur du comment à exactement 100 dp grâce à son attribut android:layout_height.

Centrer les textes dans leur conteneur au moyen de l'attribut android:gravity.

Résultat attendu →

4.2.2. Couleurs et styles

Pour créer un nouveau fichier de ressources destiné à contenir des valeurs de couleurs, sélectionner File | New | Android XML File ...

Choisir Resource Type : Values

Sélectionner le projet concerné (ici MyDemo1), entrer le nom du fichier colors.xml puis cliquer sur Next. Vérifier que le répertoire de destination est bien /res/values et cliquer sur Finish, la nouvelle ressource doit apparaître dans l'explorateur de packages...

Sur le même principe, le développeur peut choisir de regrouper les différentes ressources utiles à une application dans autant de fichiers XML que nécessaire...

On ajoute ici au fichier les trois couleurs du logotype LLF et des couleurs de fond. Ces couleurs sont spécifiées sous la forme de chaines avec les trois proportions RGB en hexadécimal "#RRGGBB" ou avec une information de transparence (format "#AARRGGBB") :

← colors.xml

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
        <color name="llf_vert">#CCCC00</color>
        <color name="llf_bleu">#0066CC</color>
        <color name="llf_pourpre">#660066</color>
        <color name="translucent_white">#FFFFFF</color>
        <color name="black">#000</color>
        <color name="light_yellow">#FFFFCC</color>
</resources>
```

Ces valeurs peuvent être affectées aux textes au moyen des attributs android:textColor et android:background.

Exemple :

```
android:textColor="@colors/llf_vert"
```

Le style d'un texte (gras, italique,...) est contrôlé par l'attribut textStyle, la taille d'un texte par textSize. Consulter la documentation et modifier activity_my_demo1.xml de manière à obtenir le résultat ci-contre : fond en noir, nom de l'application en gras et de taille 32 pixels, copyright en gras et italique, colorisation... →

4.2.3. Images

Pour faire apparaître une image, il faut d'abord l'intégrer en tant que ressource. À partir de l'explorateur de fichiers, créer dans l'arborescence du projet un nouveau répertoire res/drawable.

Copier le fichier image souhaité (par exemple ici le logotype LLF au format PNG) dans le nouveau répertoire.

Mettre à jour le Package Explorer du projet Eclipse par Refresh (F5).

Insérer ensuite les lignes suivantes dans mainactivity_my_demo1.xml, entre le nom de l'application et le copyright. La hauteur de l'élément est fixée à 200 points pour laisser une marge avec les éléments adjacents (remarques : il faut modifier la contrainte de placement du copyright et créer une nouvelle chaine dans strings.xml) :

```
    <ImageView
        android:id="@+id/image1"
        android:layout_width="wrap_content"
        android:layout_height="200dp"
        android:layout_below="@+id/textView1"
        android:contentDescription="@string/image1"
        android:src="@drawable/llf_logo"
        />
```

Le résultat obtenu est montré ci-contre →

4.3. Notions de placements (layout)

Les éléments actuels de l'unique page de l'application sont rangés suivant une stratégie définie par un RelativeLayout racine. Android offre plusieurs possibilités de stratégie de placement des composants, parmi lesquels :

- RelativeLayout qui laisse les éléments enfants se positionner en relatif par rapport au parent ;
- AbsoluteLayout qui permet aux éléments enfants de se placer de façon exacte par rapport au parent ;
- FrameLayout qui oblige l'ensemble des enfants à se placer au coin supérieur gauche ;
- LinearLayout qui permet d'aligner les enfants dans une direction unique, horizontale ou verticale...

Ces éléments peuvent être imbriqués entre eux : un LinearLayout vertical peut par exemple contenir comme élément un LinearLayout horizontal...
Créer trois nouvelles ressources *string* : « Clear », « Reset, et « Next... » (id. btnClear, btnReset et btnNext).

Modifier activity_my_demo1.xml de manière à ajouter en bas de la page d'application une ligne contenant trois boutons (de classe Button) ; mettre en place pour cela un LinearLayout horizontal avec trois composants dont le premier est défini comme ci-dessous :

```
<Button
    android:id="@+id/btnClear"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:textColor="@color/llf_vert"
    android:text="@string/btnClear" />
```

Fixer l'attribut layout_width du dernier bouton de manière à ce qu'il complète horizontalement le remplissage de son parent.

La page d'accueil de My Demo1 à obtenir est montrée ci-contre →

4.4. Interception des actions utilisateur

4.4.1. Boutons

Un bouton n'est intéressant que s'il provoque une action lorsqu'on appuie dessus ! Ce qui suit montre comment modifier le code de l'application de manière à ce que l'appui sur le bouton « Clear » efface le texte du commentaire, et que l'appui sur le bouton « Reset » le restitue.
L'intégralité de la définition de la page d'accueil a été jusqu'ici réalisée en XML, il va maintenant être nécessaire d'intervenir au niveau de code Java...

Comme évoqué précédemment, les éléments de l'interface utilisateur peuvent être retrouvés dans le code par l'intermédiaire de leur identifiant.
En réalité, toutes les ressources définies en XML sont recensées automatiquement dans une classe de l'application nommée R.java (elle aussi générée automatiquement et localisée dans le sous-répertoire gen du projet).

Ajouter, si ce n'est déjà fait, un attribut id aux objets TextView et Button concernés (le champ commentaire, le bouton « Clear » et le bouton « Reset »).

Exemple pour le bouton « Clear » (le signe + signifie qu'il s'agit d'une nouvelle ressource à créer et à ajouter dans la classe R.java ; pour référencer une ressource existante, le signe + doit être remplacé par android:) →

```
<Button
    android:id="@+id/btnClear"
    …
/>
```

Modifier MyDemo1.java comme dans l'exemple ci-dessous qui montre la récupération des id du texte et du bouton « Clear », puis la mise en place d'une routine d'écoute de l'événement *onClick* du bouton... Cet extrait de code est à placer à l'intérieur de la méthode onCreate(), à la suite de l'appel à setContentView() :

```
final TextView comment  = (TextView)findViewById(R.id.textView3);
final Button   btnClear = (Button)findViewById(R.id.btnClear);

btnClear.setOnClickListener(
          new View.OnClickListener() {
                    public void onClick(View v) {
                              comment.setText("");
                    }
          }
);
```

Avec Eclipse, le raccourci Ctrl-Shift-O ajoute automatiquement les import de classes manquants...

Bien sûr, Le traitement est identique pour l'autre bouton, sauf que le texte est restitué par R.string.comment...

La méthode setOnClickListener() est invoquée pour l'objet btnClear. Cette méthode reçoit en argument une interface de *callback* OnclickListener() de classe View, implémentée par sa méthode onClick().

Le texte de commentaire doit maintenant disparaître lorsque l'on clique sur le bouton « Clear » et réapparaitre avec le bouton « Reset » !

4.4.2. Image réactive

Dans le même ordre d'idée, il est possible de détecter un clic effectué sur l'image pour, par exemple, accéder à un site Web au travers du navigateur par défaut de l'appareil...

Modifier MyDemo1.java de manière à récupérer l'élément dans une instance img de classe ImageView, puis invoquer la méthode setOnClickListener() de cette instance de la même manière que précédemment ; sauf qu'ici, la méthode onClick() contient le code suivant :

```
Intent intent = new Intent(Intent.ACTION_VIEW, Uri.parse("http://www.lyceelafayette.fr"));
startActivity(intent);
```

U n *Intent* est un objet permettant la transmission entre composants de messages contenant de l'information : demande de démarrage d'une autre activité, action à effectuer, ...

Un clic sur l'image entraine la création d'un *Intent* de type action (premier argument). Ce type d'action est défini par le *framework* et consiste ici à « voir » l'URI (*Uniform Resource Identifier*) spécifiée par le deuxième argument.

Le lancement effectif de l'intention est réalisé par startActivity(). L'action ne définissant pas une application en particulier, on parle de démarrage d'une activité implicite ; Android va rechercher une application susceptible de répondre à la demande : le navigateur Web par défaut utilisé par l'appareil !

4.5. Lancement explicite d'une activité

En pratique, une application est souvent constituée de plusieurs pages parmi lesquelles l'utilisateur peut naviguer selon son gré, en général en cliquant sur des boutons ou autres objets de la vue en cours.

Le bouton « Next... », comme son nom semble l'indiquer, est destiné à provoquer l'affichage d'une page faisant suite à la jolie page d'accueil de MyDemo1.

Sous Android, une nouvelle page est... une nouvelle activité ! Dans ce qui suit seront donc ajoutés au projet un second *layout*, une classe java associée et un nouvel enregistrement d'activité dans le manifeste. Cela de manière à faire en sorte que l'appui sur le bouton « Next... » permette à l'utilisateur de lancer une recherche sur la toile...

Créer un nouveau fichier ressource nommé search.xml, de type *layout* avec comme élément racine un LinearLayout.

Remplir le fichier de manière à obtenir approximativement le résultat ci-contre →

L'*id* de l'objet EditText est edtSearch. Les trois boutons de taille fixe 100 x 100 dp sont de classe ImageButton, leur *id* sont btnGoogle, btnWikipedia et btnAndroid. Les images référencées ont été ajoutées dans le répertoire des *drawable*...

Créer une nouvelle classe Java publique nommée WebSearch (par le biais de File | New | Class), en spécifiant android.app.Activity comme superclasse. Le nouveau fichier doit apparaître dans le répertoire des sources Java du projet.

Éditer le fichier et ajouter le code nécessaire pour associer la vue search à l'activité →

```
package com.alainmenu.android.mydemo1;

import android.app.Activity;
import android.os.Bundle;

public class WebSearch extends Activity {

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

Déclarer la nouvelle activité dans le manifeste de l'application →

```
        <activity android:name=".WebSearch"
                  android:label="@string/app_name">
            <intent-filter>
                <action android:name="android.intent.action.DEFAULT" />
            </intent-filter>
        </activity>
```

Faire en sorte que le bouton « Next... » (*id* = btnNext) de la page d'accueil déclenche la nouvelle activité :

L'*Intent* est ici construit en spécifiant le contexte de départ (activité courante) et l'activité de destination →

Tester l'application !

```
btnNext.setOnClickListener(
    new View.OnClickListener() {
        public void onClick(View v) {
            Intent intent = new Intent(MyDemo1.this, WebSearch.class );
            startActivity(intent);
        }
    }
);
```

La deuxième page doit surgir après appui sur le bouton « Next... », la ligne d'édition de recherche doit être opérationnelle, mais les trois boutons vers les moteurs de recherche sont encore inopérants.
La touche retour de l'appareil permet de revenir sur la page d'accueil...

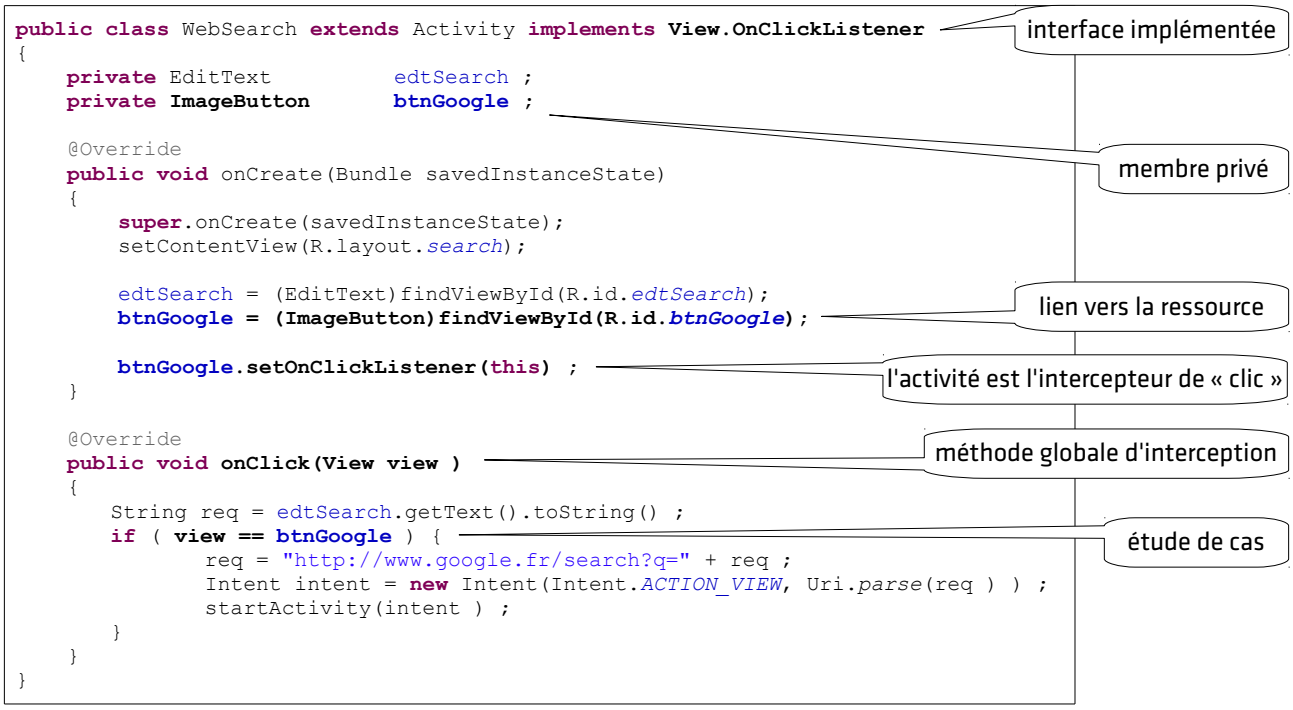
4.5.1. Intention explicite (vers navigateur Web)

Dans la première activité, les interceptions ont été codées une à une dans la méthode onCreate() en invoquant setOnClickListerner() pour chaque objet concerné.

Une autre solution consiste à spécifier que la classe d'activité implémente l'interface View.OnClickListener. L'activité est ensuite enregistrée auprès des éléments concernés afin qu'elle reçoive les événements lorsque l'utilisateur clique sur ceux-ci. Les événements sont pour finir traités globalement dans une unique surcharge de la méthode onClick() de la classe.

Exemple pour le bouton btnGoogle (cf. extrait de code page suivante) :

Ici, l'appui sur le bouton « Google » provoque le lancement d'une intention implicite vers le navigateur Web par défaut (comme précédemment lors du clic sur le logo de la page d'accueil), avec comme URI une demande de recherche sur www.google.fr.



Compléter le code de la classe WebSearch. Les URI à utiliser pour les boutons « Wikipédia » et « Android » sont respectivement (avec ??? représentant le texte saisi par l'utilisateur) :

- http://fr.wikipedia.org/wiki/???
- http://developer.android.com/search.html#q=???

4.5.2. Signalisation (toast)

Et pour finir, portons un toast !

La classe android.widget.Toast fournit un moyen très simple pour signaler à l'utilisateur de manière discrète, rapide et efficace, un événement non bloquant, sous la forme d'un affichage discret et non modal.

Ajouter le code suivant à la fin de la méthode onClick(). Ceci entraine l'affichage d'un message furtif (durant environ 2 secondes) lorsque l'utilisateur clique sur un des trois boutons de recherche, sans perturber le chargement de la page demandée :

```
Toast toast = Toast.makeText(getApplicationContext(), req, Toast.LENGTH_SHORT );
toast.show();
```

La méthode makeText() crée le composant de classe Toast qui est ensuite affiché par show() ; elle nécessite trois arguments : le contexte de l'application, le texte à afficher et une durée à choisir entre LENGTH_SHORT et LENGTH_LONG.

Un *toast* peut être personnalisé. Consulter la documentation de la classe Toast pour voir les possibilités de positionnement sur l'écran, de contrôle de la durée d'affichage...

Si un simple message ne suffit pas, il est même possible de provoquer l'affichage d'une *view* ou d'un *layout* plus complexe.

Références

Site Google officiel : http://www.android.com/
Site officiel de développement Android : http://developer.android.com/
Site officiel des sources Android : http://source.android.com/
Site de l'OHA : http://www.openhandsetalliance.com/
Communauté francophone autour d'Android : http://www.frandroid.com/,
et le site collaboratif associé : http://wiki.frandroid.com/wiki/Accueil
« Développement d'applications professionnelles avec Android 2 », Reto Meier – Pearson – ISBN : 978-2-7440-2452-8