IUT - Département Informatique

Android - 2013-2014

3ème séance - Base de données

Android TD3

1 Bases de données et quelques concepts élémentaires

Dans ce TD, nous allons suivre un autre tutoriel en ligne qui vous guidera dans la création d'un gestionnaire de notes textuelles. Ce tutoriel couvrira plusieurs aspects. Nous verrons en particulier :

— comment créer et utiliser une base de donnée embarquée dans l'application (SQLite) ;
— comment afficher une liste d'items résultant d'une requête SQL (via `ListActivity`[1] et `SimpleCursorAdapter`[2]) ;
— et comment ajouter des actions dans le menu d'une application.

Ce tutoriel est très facile à suivre, mais il est long ! Il y a surtout beaucoup d'explications à lire en anglais. Donc ne perdez pas de temps, cherchez à capter l'essentiel et n'hésitez pas à demander de l'aide lorsque vous êtes bloqués (y compris pour l'anglais).

0a Téléchargez l'archive `td3.zip` à l'adresse que nous vous avons indiqué et décompressez-là dans votre répertoire de travail Android. Cette archive contient le squelette d'une application, à partir duquel vous travaillerez. Nous y avons apporté quelques modifications par rapport au tutoriel en ligne que vous allez suivre pour le rendre compatible avec les derniers SDK. L'archive contient aussi une version électronique de cette feuille de TD.

0b Rendez-vous sur la page du tutoriel : `http://developer.android.com/training/notepad/notepad-ex1.html` et commencez à suivre le tutoriel. Ce dernier se décompose en plusieurs étapes (Step 1, Step 2, etc.). Pensez à revenir à cette feuille de TD après (ou pendant) chaque étape, car nous ferons souvent le point sur les notions parcourues.

Step 1 Importez le projet Notepadv1 à partir de notre archive (td3.zip). Dans Android Studio : File > Import Project.

Step 2 (à retenir) Toutes les informations concernant la base de données sont généralement encapsulées dans une même classe. Cette classe définit, entre autres, des variables java qui représentent le schéma de la base (noms des tables, des champs, etc.. qui ont vocation à être utilisés ensuite à la place des valeurs en dur, pour plus de modularité). Sont aussi définis les requêtes SQL permettant de supprimer ou (re)créer les tables de la base lors du déploiement de l'application, ainsi que toutes les opérations de manipulation de données (encapsulées sous forme de méthodes). A noter aussi que les opérations SQL de base, telles que l'ajout, la suppression ou la mise à jour d'un enregistrement sont assistées par des méthodes dédiées `insert()`, `delete()`, `update()`. Les insertions et les mises à jour utilisent des tables d'association clés/valeurs (classe `ContentValues`[3]) pour simplifier les requêtes.

Step 3 (à retenir) Il est impératif d'utiliser l'espace de nommage :
`xmlns:android="http://schemas.android.com/apk/res/android"` dans les fichiers de définition d'activités.

Step 4 (à retenir) La manière de désigner l'identifiant de la liste et du champs texte sont différentes de ce qu'on a vu la dernière fois (absence du '+' après le '@'). La raison est que nous ne voulons pas, ici, générer de nouveaux identifiants, mais leur affecter des identifiants pré-définis dans le SDK (en l'occurrence, `android.R.id.list` et `android.R.id.empty`) qui lui permettront de manipuler ces objets de façon transparente pour vous. Plus généralement, il y a deux fichiers de ressources : celui qui est propre à votre application (répertoire gen, dans votre projet) et généré automatiquement lors de la compilation, et celui qui est général au SDK (`android.R`, dans `android.jar`).

Step 5 (à retenir) Les listes d'affichage utilisent un layout *pour chaque ligne*. Dans cet exemple, il s'agit d'un simple champs texte, mais vous pourriez tout aussi bien définir des lignes qui comportent plusieurs éléments organisées dans une hiérarchie de layouts.

Step 6 (à retenir) Lorsque l'activité consiste essentiellement en une liste d'affichage, on peut la faire hériter de `ListActivity`, qui comporte des fonctionnalités dédiées et simplifie fortement la gestion de la liste. La majorité de la programmation d'une application consiste à redéfinir (`@Override`) des méthodes de type « callback », qui sont appelées lorsque tel ou tel évènement se produit (p. ex. une étape de cycle de vie, un clique sur un élément donné ou la fin d'un traitement).

1. http ://developer.android.com/reference/android/app/ListActivity.html
2. http ://developer.android.com/reference/android/widget/SimpleCursorAdapter.html
3. `http://developer.android.com/reference/android/content/ContentValues.html`

Step 9 (à retenir) Le tutoriel vous a fait ajouter un élément au menu depuis le code java directement. Il est aussi possible de définir les éléments du menu depuis un fichier xml, tout comme nous le faisons pour l'interface graphique (répertoire `res/menu/`, non créé dans ce projet). Il est aussi possible d'associer une icône à chaque action, soit en plus, soit à la place du texte.

Step 12 (à corriger) L'API a évolué depuis la création de ce tutoriel et l'usage de certaines méthodes est maintenant découragé (d'ailleurs, Eclipse vous indique ces appels erronés). C'est le cas de la méthode `startManagingCursor`, qui (sauf erreur de notre part) est devenue inutile et dont l'appel peut simplement être supprimé. Le constructeur `SimpleCursorAdapter`, quant à lui, est déconseillé car il s'exécute dans le même thread que l'activité, pouvant ainsi bloquer l'interface graphique si la requête correspondant s'avère longue à exécuter. Nous ignorerons la « déprécation »en utilisant l'autre version du constructeur avec pour dernier paramètre la valeur 0. Une vrai correction consisterait à utiliser des `Loader`[4], mais leur mise en place demande trop de notions qui ne peuvent pas être connues dans ce TD.

Step 13 Ouf ! Tout ça pour ça ? Ben oui, mais vous avez appris pas mal de choses essentielles. Demandez-moi la feuille suivante pour passer à l'exercice 2 du tuto, où cette application commencera à faire des choses intéressantes.

4. http ://developer.android.com/guide/components/loaders.html