# Android et l’accès aux données – 2ème partie

## Objectifs

Finaliser notre interface graphique.

## Fonctionnalités de l’application

L’utilisateur doit pouvoir sélectionner des éléments dans une liste déroulante, ajoutez des éléments et les faire persister dans la base de données.

Pour ajouter des éléments, une nouvelle activité s’ouvre permettant à l’utilisateur de saisir la nouvelle compétence. Lorsqu’il valide il revient à l’écran principal avec la liste de compétences actualisée.

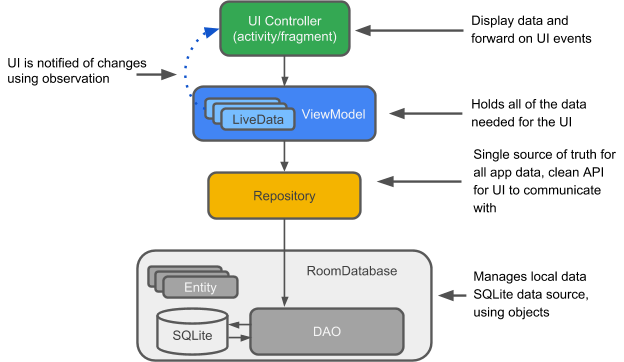
## Contenus

Environnement Android Studio –outil de build Gradle (packaging et dépendances) – DAO – Repository – ORM – SQL Lite - Room – interface – LiveData

Tests unitaires avec Junit – Génération de documentation.

**Pré requis :**

TP de la semaine dernière finalisé. Classes et interface complétées : Competence, CompetenceDao, CompetenceRoomDatabase, CompetenceRepository et CompetenceViewModel



Issu de google Codelabs, traduit par moi :

«**Entity:** une classe annotée qui décrit une table de la base de données.

**SQLite database: sur le smartphone physique, les données sont stockées dans une base de données SQLite. La librairie de persistence Room se charge de créer et maintenir la base de données.**

**DAO:** Data access object. Effectue un mapping, c’est à dire une correspondance entre les requêtes SQL et les méthodes. Quand on utilise le DAO avec Room, on appelle les méthodes et Room se charge du reste (accès aux données, récupération ….)

**Room database:** une surcouche à la base de données SQLite, qui sert de point d’accès. La base de données Room utilise les DAO pour produire les requêtes envoyées à SQLite

**Repository:** Une classe crée par le développeur pour pouvoir gérer plusieurs sources de données.

**ViewModel:** Permet de fournir les données à l’interface utilisateur. Permet de faire communiquer l’interface (UI) avec le Repository, en cachant également à l’UI la provenance des données. Une instance ViewModel persiste même après un changement de configuration.

**LiveData: c’est une classe qui contient des données et qui peut être observée. Elle contient toujours la version la plus à jour des données. A chaque fois qu’une donnée change, elle notifie les observateurs (observers).** »

**Partie 4 : création des layout et mise en forme XML**

Etape 1 : style et composants

Dans le fichier values/**styles.xml** (même emplacement que strings.xml), ajouter l’entrée suivante :

<**style name="competence\_title"**>  
 <**item name="android:layout\_width"**>match\_parent</**item**>  
 <**item name="android:layout\_height"**>26dp</**item**>  
 <**item name="android:textSize"**>24sp</**item**>  
 <**item name="android:textStyle"**>bold</**item**>  
 <**item name="android:layout\_marginBottom"**>6dp</**item**>  
 <**item name="android:paddingLeft"**>8dp</**item**>  
</**style**>

Remarque : c’est nécessaire pour avoir des éléments suffisamment visibles pour notre liste.

Etape 2 : ajout d’un nouveau Layout.

Ajoutez dans le répertoire res/layout un fichier nommé **recyclerview\_item.xml**, avec le code suivant :

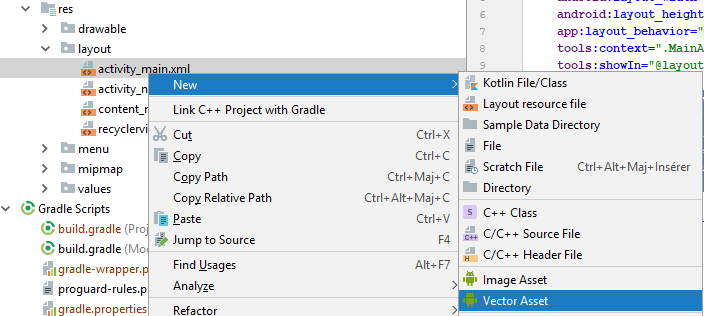
*<?***xml version="1.0" encoding="utf-8"***?>*<**LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"  
 android:orientation="vertical" android:layout\_width="match\_parent"  
 android:layout\_height="wrap\_content"**>  
  
 <**TextView  
 android:id="@+id/textView"  
 style="@style/competence\_title"  
 android:layout\_width="match\_parent"  
 android:layout\_height="wrap\_content"  
 android:background="@android:color/holo\_blue\_light"** />  
</**LinearLayout**>

Etape 3 : dans le fichier **content\_main.xml** , remplacer le TextView par un RecyclerView

<**android.support.v7.widget.RecyclerView  
 android:id="@+id/recyclerview"  
 android:layout\_width="match\_parent"  
 android:layout\_height="match\_parent"  
 android:background="@android:color/darker\_gray"  
 tools:listitem="@layout/recyclerview\_item"** />

Etape 4: remplacement du bouton d’action

1. Dans **activity\_main.xml**, effectuez clic droit puis New 🡪 Vector Asset



Puis dans la partie Icon, allez chercher le symbôle +

1. Enfin, dans le fichier **activity\_main.xml**, ajoutez l’attribut suivant aux propriétés de la widget **FloatingActionButton** :

**android:src="@drawable/ic\_add\_black\_24dp"**

**Partie 5 : ajout d’une classe CompetenceListAdapter**

**Introduction**: fonctionnement du RecyclerView

Sources :

<https://developer.android.com/reference/android/support/v7/widget/RecyclerView>

<https://developer.android.com/reference/android/support/v7/widget/RecyclerView.Adapter>

<https://android.jlelse.eu/understanding-recyclerview-a-high-level-insight-part-1-dc3f81af5720> : parties 1 er 2

Définition Android de base: “A flexible view for providing a limited window into a large data set.”

Définition peu claire et très large !

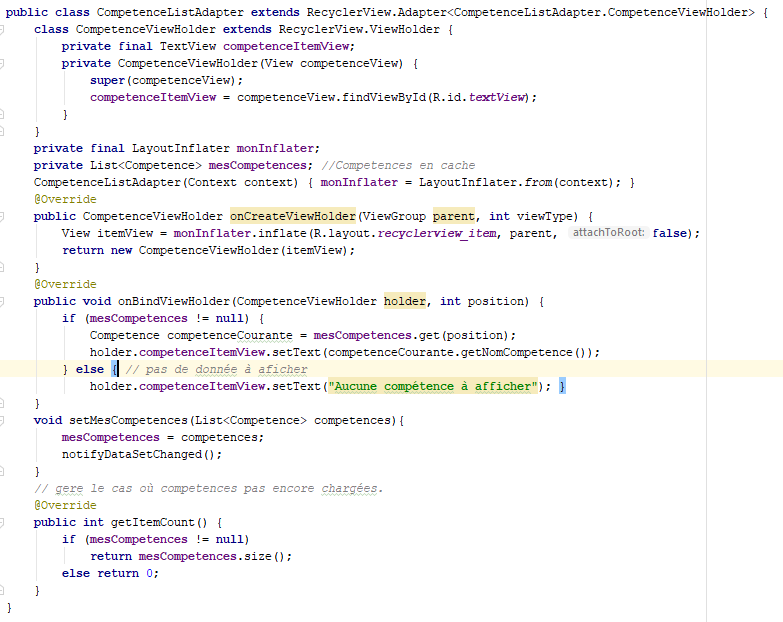
En fait, la classe et widget **RecyclerView** , et tout ce qui est lié (adapteur, layout manager, holder) ont été définis pour pallier les défaillances des **ListView**, widget et classes permettant de gérer des listes mais avec les défauts suivants : scrolling uniquement vertical, performances passables et difficulté à gérer des animations.

Un RecyclerView est un nouveau widget de type ViewGroup utilisable pour afficher n’importe quelle vue avec un support Adaptor sur une grille horizontale ou verticale.

* Un ViewGroup est une View spéciale capable de contenir d’autres View
* Une View avec support Adapter est manipulable par n’importe quel Adapter
* Un Adapter est une classe chargée de la création des View et d’attacher les données aux View

Etape 5: ajout d’une classe CompetenceListAdapter

Ajoutez une classe CompetenceListAdapter qui étend RecyclerView.Adapter , avec le code suivant :



Remarque :

On surcharge la méthode getItemCount() de RecyclerViewAdapter pour gérer le cas où les compétences n’ont pas encore été chargées.

**Questions 1** : analyse de la classe CompetenceListAdapter

1. Ligne 17, où se situe le textview récupéré via un findViewById ?
2. En vous basant sur l’extrait ci dessous, expliquez ce que font les 2 méthodes **onCreateViewHolder** et **onBindViewHolder** ?

***Extrait***issu d’une discussion StackoverFlow: <https://stackoverflow.com/questions/36042926/how-recyclerview-concept-works-on-android>

When a view is attached to the RecyclerView, it is removed from the pool, and when it is detached (scrolls beyond view, to some distance), it is added back to the pool of empty views--this is why it is important to reset everything when you populate your ViewHolders.

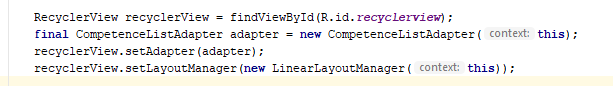
The [onCreateViewHolder()](http://developer.android.com/reference/android/support/v7/widget/RecyclerView.Adapter.html" \l "onCreateViewHolder(android.view.ViewGroup,%20int)) function is where a new, empty view (wrapped by a RecyclerView.ViewHolder) is created and added to the pool.

The [onBindViewHolder()](http://developer.android.com/reference/android/support/v7/widget/RecyclerView.Adapter.html" \l "onBindViewHolder(VH,%20int)) function gets a view from the empty pool and populates this view using the data you supplied to the adapter.\

You can use the [onViewRecycled()](http://developer.android.com/reference/android/support/v7/widget/RecyclerView.Adapter.html" \l "onViewRecycled(VH)) method to perform specific actions like setting an ImageView's bitmap to null (on detach) in order to reduce memory usage.

Etape 6: ajout du RecyclerView dans le MainActivity

A la fin de la méthode *onCreate()* de la classe **MainActivity**, ajoutez le code suivant :



**Question 2** : complétez les trous

* On récupère le widget recyclerview et on l’affecte à \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_
* Une nouvelle instance non modifiable de \_\_\_\_\_\_\_\_\_\_ est initialisée avec le contexte courant
* Le widget recyclerview est associé à \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ qui va gérer la partie données
* Il est également associé à un gestionnaire de Layout de type \_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Checkpoint n°1:**

Vérifiez que l’application se lance bien dans l’émulateur Genymotion ou interne, sans erreurs.

**Partie 6 : alimentation de la base de données**

Au début il n’y a pas de données dans la base.

L’alimentation de la base de données se fait de 2 manières :

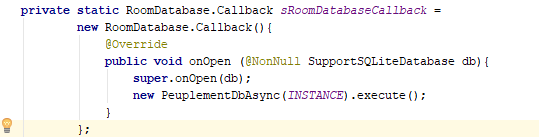
- ajout de données quand la base de données est ouverte

- ajout d’une activité permettant la saisie et l’ajout de nouvelle compétence

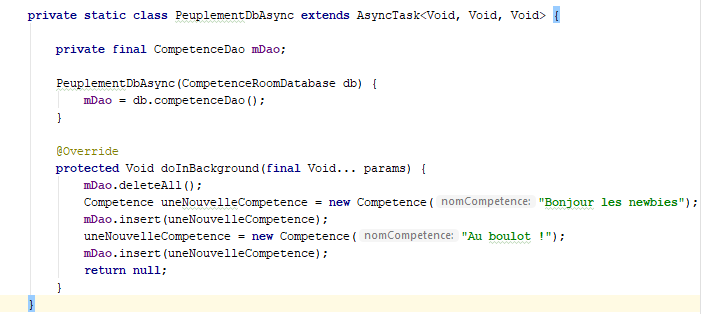
Etape 7: réinitialisation et « peuplement » initial de la base de données

Nous allons ajouter du code permettant de supprimer les données puis d’ajouter 2 messages à chaque fois que l’application est redémarrée.

1. Dans la classe **CompetenceRoomDatabase**, ajout d’une variable de classe de ***Callback*** :



1. Toujours dans la même classe, création d’une classe interne **PeuplementDbAsync**



1. Ajout du CallBack au moment de la construction de la base de données en modifiant le build() en ajoutant le callback : retrouvez et modifiez la ligne concernée :



**Question 3** :

1. Qu’est ce qu’un Callback ? Quand en avons nous déjà utilisé ?
2. Que fait la classe **PeuplementDbAsync ?** Comment est-elle appelée ?

Etape 8: ajout d’une activité pour ajouter des compétences

1. Ajoutez les entrées suivantes dans le fichier strings.xml



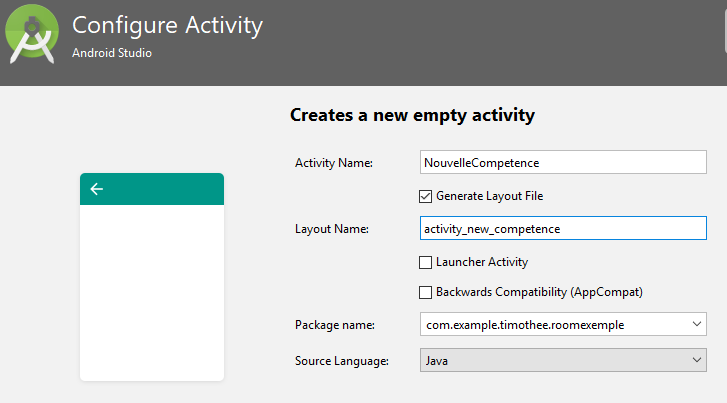
1. Ajout d’une couleur dans le fichier res/values/colors.xml



1. Ajout de resources de type dimension dans le fichier res/values/dimens.xml

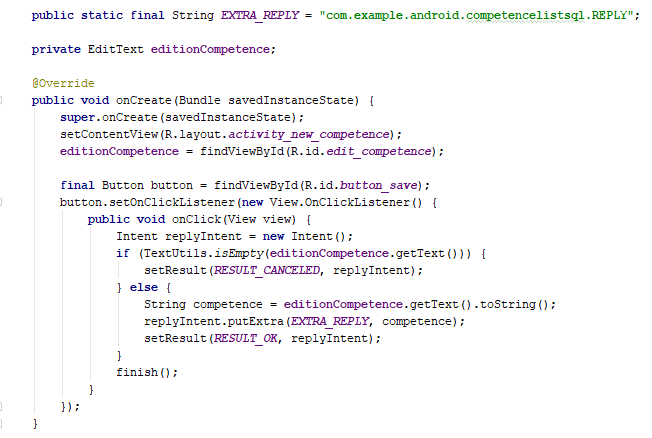


1. Créez une nouvelle activité, en utilisant le template Empty, par exemple



Ici je n’ai pas gardé le nom par défaut pour le layout.

1. Le contenu du Layout :
2. *<?***xml version="1.0" encoding="utf-8"***?>*<**LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"  
    android:orientation="vertical" android:layout\_width="match\_parent"  
    android:layout\_height="match\_parent"**>  
     
    <**EditText  
    android:id="@+id/edit\_competence"  
    android:layout\_width="match\_parent"  
    android:layout\_height="wrap\_content"  
    android:fontFamily="sans-serif-light"  
    android:hint="@string/hint\_competence"  
    android:inputType="textAutoComplete"  
    android:padding="@dimen/small\_padding"  
    android:layout\_marginBottom="@dimen/big\_padding"  
    android:layout\_marginTop="@dimen/big\_padding"  
    android:textSize="18sp"** />  
     
    <**Button  
    android:id="@+id/button\_save"  
    android:layout\_width="match\_parent"  
    android:layout\_height="wrap\_content"  
    android:background="@color/colorPrimary"  
    android:text="@string/button\_save"  
    android:textColor="@color/buttonLabel"** />  
     
   </**LinearLayout**>
3. Complétez le code pour la classe NouvelleCompetence



Remarques: RESULT\_OK et RESULT\_CANCELED sont des codes retour standard d’une activité fille vers l’activité mère qui l’a initialisé.

Cf: <https://developer.android.com/training/basics/intents/result> et <https://developer.android.com/reference/android/app/Activity> pour le cycle de vie des activités.

**Question 4** : complétez les trous

* On définit 2 variables, une \_\_\_\_\_\_\_\_ nommée \_\_\_\_\_\_\_\_\_\_ et un attribute de la classe \_\_\_\_\_\_\_\_ nommé \_\_\_\_\_\_\_\_\_\_\_
* A l’initialisation de l’activité, on récupère notre widget \_\_\_\_\_\_\_\_ affectée à \_\_\_\_\_\_\_\_\_
* On récupère également notre widget \_\_\_\_\_\_\_\_\_ auquel on associe un Listener de type \_\_\_\_\_\_
* Si le champ texte est \_\_\_\_ on renvoie une annulation
* Sinon on renvoie le \_\_\_\_\_\_\_\_\_\_\_\_\_\_

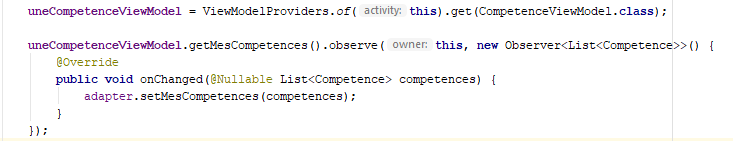
**Partie 7 : finalisation – connexion aux données**

Etape 9: connexion de l’UI avec la base de données

1. Dans MainActivity, ajout d’une variable **CompetenceViewModel**



1. A la fin de la méthode onCreate()



Remarques:

* Notre variable uneCompetenceViewModel est alimentée via une classe utilitaire ViewModelProviders. Cf <https://developer.android.com/reference/android/arch/lifecycle/ViewModelProviders>
* On ajoute enfin notre Observateur qui fonctionne en symbiose avec la classe LiveData

**Question 5** : lisez le début de la doc fournie en lien.

1. Quel constructeur est appelé ? Lorsqu’il y a plusieurs constructeurs, on appellee ça \_\_\_\_\_\_\_
2. Dans le code ci joint: **uneCompetenceViewModel** = ViewModelProviders.*of*(**this**).get(CompetenceViewModel.**class**);

Que renvoit ViewModelProviders.*of*(**this**) ?

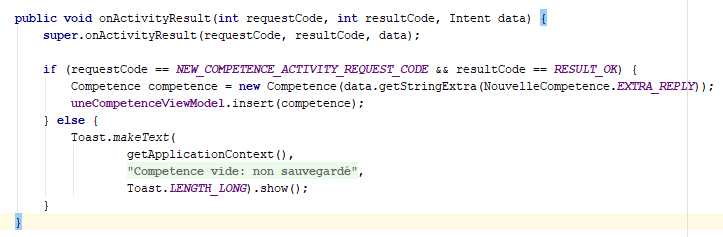
A quelle classe s’applique la méthode get() ? Que renvoie t’elle ?

1. Quel code est exécuté par l’observateur si des données changent ?

Etape 10: écrire la méthode onActivityResult()

Cette méthode va récupérer le résultat de l’activité NouvelleCompetence définie dans la partie précédente.

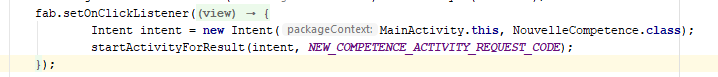
1. Ecrire le code de la méthode



1. Ajoutez la constante ***NEW\_COMPETENCE\_ACTIVITY\_REQUEST\_CODE***

**public static final int *NEW\_COMPETENCE\_ACTIVITY\_REQUEST\_CODE*** = 1;

1. Modifiez le code du listener du bouton



Remarque : attention ne modifiez que ces 2 lignes.

Vous voyez qu’on appelle une nouvelle activité exactement de la même manière qu’en PPE.

**Checkpoint n°2:**

1. Vérifiez que l’application se lance bien dans l’émulateur Genymotion ou interne, sans erreurs.
2. Vérifiez que vous pouvez ajouter de nouvelles compétences

**Exercice n°1:**

Si le checkpoint précédent est Ok, effectuez les modifications nécessaires pour persister les données même lorsque l’activité est redémarrée.