Université Paris Cité campus Grands Moulins
Programmation des composants mobiles
M1 informatique **Année 2022-2023**

TP n° 4

RecyclerView

Objectifs et ressources

Le but de tp est de faire un premier prototype d'une application qui permet de mémoriser les présences/absences en TP, TD ou en cours (en remplaçant une liste papier par la mémoire de mobile).

L'application complète utiliserait sans aucun doute une base de données. Mais pour notre prototype on se contentera juste d'une activité qui affiche la liste de tous les étudiants et permet de sélectionner les présents, ajouter et supprimer des étudiants. Par contre Le bouton `SAUVEGARDER` restera inactif : dans le listener associé vous ferez l'appel à la méthode `TODO()`.

Peut-être on aura l'occasion d'y revenir après le cours de bases de données android pour améliorer l'application et la rendre vraiment utile.

Dans la deuxième partie de TP on ajoutera le tri de la liste affichée.

L'application consiste de deux activités.

1 Activité principale

L'activité principale consiste d'une liste implémentée par un `RecyclerView` et de trois boutons : `Sauvegarder`, `Supprimer`, `Ajouter`.

L'utilisation de `RecyclerView` est impérative, l'application ne peut pas marcher correctement si la liste est affichée avec une `ListView`. N'oubliez pas d'ajouter dans `build.gradle (Module)` dans la section `dependencies` :

```
implementation 'androidx.recyclerview:recyclerview:1.2.1'
```

On utilisera aussi un `CardLayout` de androidx, et pour cela on ajoute dans la même section de fichier gradle `implementation 'androidx.cardview:cardview:1.0.0''`

Chaque item de la liste est composé de deux `TextView`s qui contiennent le nom et prénom d'un étudiant. Les deux `TextView` sont englobés dans un `CardView`.

1.1 Ressources fournies

Nous fournissons les fichiers suivants :

— `activity_main.xml` le fichier layout de l'activité principale,

— `activity_ajouter_etudiant.xml` le fichier layout de la deuxième activité,

— `item_layout.xml` le fichier layout d'un item affichée dans le `RecyclerView` de l'activité principale.

Pour de raisons esthétique l'élément à la racine de `item_layout.xml` est un `CardView` qui affiche un rectangle avec les angles arrondis. Mais bien plus important : il est possible

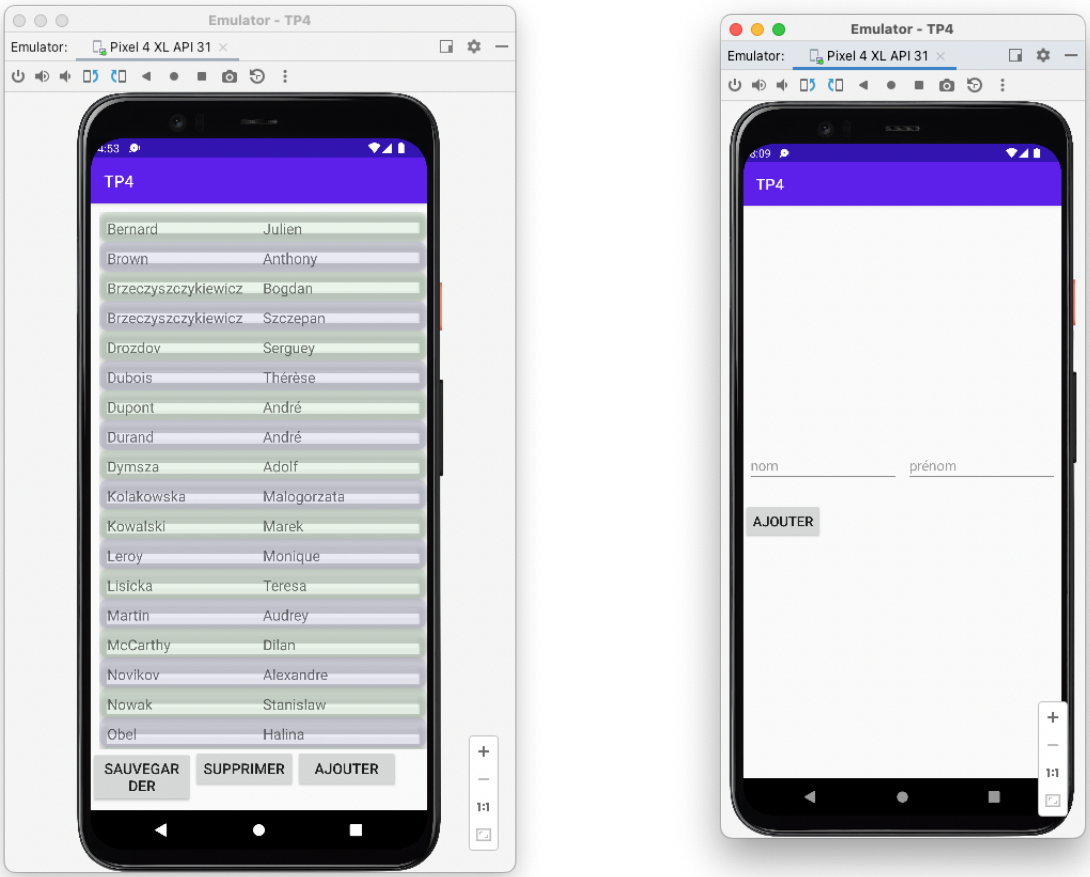

FIGURE 1 – Activité principale et l'activité `AjouterEtudiant`.

d'installer `View.OnClickListener` sur une `CardView`. Cela permet d'implémenter plus facilement les interactions avec l'utilisateur.

— Le fichier `strings.xml` contient deux tableaux de `String`s de même longueur. Le premier tableau contient les noms d'étudiants et le deuxième les prénoms. On suppose que la i-ème position dans chaque tableau donne respectivement le nom et le prénom de ième étudiant.
Copier-coller les définitions de deux tableau de `strings.xml` dans le fichier `res/values/strings.xml` de votre application.

— Le fichier `colors.xml` contient la définition de trois couleurs nommées `couleurPaire`, `couleurImpaire`, `couleurChecked`. Les deux premières couleurs serviront de couleurs de fond (background) pour les étudiants affichés sur les position paires et impaires sur la liste. La couleur `couleurChecked` sera utilisée comme la couleurs de fond pour les étudiants sélectionnés. Vous ajouterez ces définitions dans le fichier `res/values/colors.xml`

Les fichiers layout utilisent une taille de police que vous devez ajouter dans le fichier `res/values/dimens.xml`. Pour cela il suffit d' mettre la ligne `<dimen name="textSize">18sp</dimen>`.

Si les caractères s'avèrent trop grands/petits pour l'écran de votre appareil diminuez/augmenter la valeur de `textSize`. Si le fichier `dimens.xml` n'existe pas il faut le crée : New →

XML → Values XML file.

1.2 Class Etudiant

Créer un nouveau fichier kotlin dans lequel vous mettrez la définition de la classe `Etudiant` :

```
data class Etudiant(val prenom: String, val nom: String)
```

1.3 Lecture de tableaux ressources

Dans l'activité principal vous devez lire les tableau de noms et prénoms et fabriquer une liste/tableau d'étudiant. Cette liste servira à alimenter le `RecyclerView`.

1.4 `RecyclerView.Adapter`

Écrivez un `RecyclerView.Adapter`. Dans le constructeur de `RecyclerView.Adapter` vous passerez comme paramètres : la liste mutable d'étudiants, les trois couleurs `couleurPaire`, `couleurImpair`, `couleurChecked` (en tant que `Int`).

Remarque. Une fois vous avez écrit en-tête de la classe qui implémente `RecyclerView.Adapter` il suffit de positionner le curseur à l'intérieur de la classe et sélectionner `Code` → `Implement methods` pour que AndroidStudio génère les méthodes à implémenter.

Remarques. Si vous avez correctement écrit la fonction `onCreateViewHolder` de `Adapter` (et c'est presque copier-coller de l'exemple donné de cours) l'attribut `itemView` de `holder` contient la référence vers la View à la racine d'un item de la liste affichée par `RecyclerView`, c'est-à-dire `holder.itemView` est un `CardView`

Donc dans la méthode `onBindViewHolder()` il ne reste qu'a mettre correctement le nom et prénom de l'étudiant dans la view. Pour récupérer les références vers les `TextViews` pour le nom et prénom il suffit de récupérer le résultat de `holder.itemView.findViewById<TextView>( ... )` appliqué à l'identifiant de `TextViews` défini dans `item_layout.xml`.

Après cette étape votre activité devrait afficher la liste de tous les étudiants.

1.5 Améliorer l'affichage

Tous les éléments de la liste sont affichés de la même façon ce qui rend difficile la lecture (où commence un nouveau étudiant ?)

Modifier le code de la fonction `onBindViewHolder` pour que la couleur background de chaque item soit respectivement `couleurPaire` à la position paire et `couleurImpaire` à la position impaire.

Remarque. `CardView` possède la méthode `setCardBackgroundColor( color : Int )` dont le paramètre est le code de couleur de arrière-plan de `CardView`.

2 Gérer les étudiants sélectionnés par utilisateur

Le plus important devant nous.

Pour l'instant rien n'a été fait pour pouvoir sélectionner des étudiants sur la liste.

Quand utilisateur tape sur un étudiant sur la liste la couleur background de item correspondant de la liste doit devenir `couleurChecked`.

Bien sûr si l'utilisateur clique à nouveau sur un étudiant celui-ci doit être de-sélectionné : la propriété `checked` passe à `false` et la couleur redevient soit `couleurPaire` soit `couleurImpaire`.

Comme dans l'exemple du cours il faut installer un `OnClickListener` sur chaque objet `CardView` affiché sur la liste et, comme en cours, on le fera dans `onCreateViewHolder()`.

Dans `onCreateViewHolder()`, grâce à `LayoutInflater` on construit d'abord un `CardView`, et ensuite on crée un `ViewHolder`.

Pour écrire le listener il faut connaître la position de l'étudiant sur la liste d'étudiant. Hélas `onCreateViewHolder()` ne connaît pas la position de l'étudiant, en fait cette méthode est appelée pour créer le holder et la view associée et non pas pour remplir la view avec les données. La position de l'étudiant est connue dans `onBindViewHolder()`, mais cette méthode n'est pas appropriée pour ajouter un listener.

La solution est suivante : il s'avère qu'on peut obtenir la position de l'étudiant sur la liste d'étudiant à l'intérieur de listener grâce à la propriété `holder.absoluteAdapterPosition` du holder. Bien sûr cela implique que dans `onCreateViewHolder` il faut d'abord construire le holder et seulement ensuite créer le listener.

Notez aussi que `holder.absoluteAdapterPosition` donne la position de l'étudiant au moment de l'activation de listener, au moment de création de listener `holder.absoluteAdapterPosition` ne donne aucune valeur pertinente.

Après cette étape l'utilisateur pourra sélectionner et de-sélectionner les étudiants sur la liste.

3 Suppression

Quand l'utilisateur clique sur le bouton `SUPPRIMER` tous les étudiants sélectionnés doivent être supprimés de la liste. Sans aucun doute vous devez maintenir dans le `RecyclerView.Adapter` une liste des étudiants sélectionnés et la mettre à jour quand l'utilisateur sélectionne/de-sélectionne les étudiants.

4 Ajout

Quand l'utilisateur clique sur le bouton `AJOUTER` l'activité `AjoutEtudiant` doit être activée. Dans cette activité l'utilisateur donne le nom et prénom d'un nouveau étudiant et clique sur le bouton. Si les nom et prénom sont non-vide l'activité termine et passe les deux Strings dans l'activité principale. Dans l'activité principale il faut ajouter le nouveau étudiant sur la liste.

5 Liste triée

Cette section explique comment obtenir les listes triées dans un `RecyclerView`. La présentation, le sujet sera traité uniquement en TP.

Le but est d'avoir la liste d'étudiants triée par le nom et ensuite (pour le même nom) par le prénom.

Pour trier la liste `RecyclerView.Adapter` utilisera un objet de la classe
SortedList.

Mais d'abord pour obtenir une liste triée il nous faut un objet de la classe
SortedList.Callback
qui permet de comparer les objets à triés.

La définition de l'objet `callback` est suivante :

```
val callback = object : SortedList.Callback<Etudiant>() {

        override fun compare(o1: Etudiant?, o2: Etudiant?): Int {
            val cmpNoms = o1!!.nom.compareTo(o2!!.nom)
            val cmpPrenoms = o1!!.prenom.compareTo(o2!!.prenom)
            return if (cmpNoms != 0) cmpNoms else cmpPrenoms
        }

        override fun onInserted(position: Int, count: Int) {
            notifyItemRangeInserted(position, count)
        }

        override fun onRemoved(position: Int, count: Int) {
            notifyItemRangeRemoved(position, itemCount)
        }

        override fun onMoved(fromPosition: Int, toPosition: Int) {
            notifyItemMoved(fromPosition, toPosition)
        }

        override fun onChanged(position: Int, count: Int) {
            notifyItemRangeChanged(position, count)
        }

        override fun areContentsTheSame(oldItem: Etudiant?, newItem: Etudiant?): Boolean =
            (oldItem == null && newItem == null) || newItem!!.equals(oldItem)

        override fun areItemsTheSame(item1: Etudiant?, item2: Etudiant?): Boolean =
            item1 === item2
    }
```

On reconnaît que la méthode `compare` de l'objet défini l'ordre entre les étudiants, l'ordre de dictionnaire sur les noms et, pour les étudiants avec le même nom, l'ordre de dictionnaire sur le prénom.

Copiez cette définition dans votre `RecyclerView.Adapter` (comme une propriété). (Le code est donné aussi dans le fichier `callback.kt`).

Mais c'est quoi object ? Kotlin ne possède pas de classes anonymes. Mais il est possible de créer un objet qui implémente un interface ou une classes abstraite. Le code donné crée un tel objet qui implemente les méthodes abstraite de la classe `SortedList.Callback`.

Cette objet sera utilisé par le `RecyclerView.Adapter` pour obtenir la liste trée d'étudiants.

Dans le `RecyclerView.Adapter` vous ajouterez la propriété `sortedList` initialisée de façon sivante :

```
val sortedList = SortedList<Etudiant>(Etudiant::class.java, callback)
```

Le deuxième paramètre, `callback`, implémente `SortedList.Callback<Etudiant>`.

A partir de ce moment Android maintient automatiquement l'ordre dans la liste `sortedList`.

Et l'ordre d'élément dans `sortedList` sera automatiquement mis à jour quand on ajoute ou supprime les étudiants de `SortedList`.

Donc il suffit d'ajouter tous les étudiants dans la liste triée :

```
sortedList.addAll(etudiants)
```

et remplacer partout ailleurs dans le `RecyclerView.Adapter` les références vers la liste `etudiants` par les références vers la liste triée `sortedList`.