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

TP n° 5

RecyclerView (suite) et fichiers

Objectifs

Le but du TP est d'améliorer et d'ajouter des fonctionnalités à l'application développé en TP4. Mais d'abord **terminez le TP4**. Tout le monde doit apprendre comment implémenter un `RecyclerView`.

1 Tri par nom et par prénom

On veut faire en sorte que l'ordre du tri (nom ou prénom d'abord) corresponde au choix fait sur le spinner.

Vous devez ajouter dans `RecyclerView.Adapter` une propriété `sortColumn : String` dont la valeur (`nom` ou `prenom`) indique si on effectue le tri par le nom ou par le prénom de l'étudiant. La propriété `sortColumn` s'ajoutera aux paramètres du constructeur de votre `RecyclerView.Adapter`. Ensuite il faudra adapter la méthode `compare` du `callback`

Dans l'activité principale il faut récupérer la valeur de spinner et la passer au constructeur de `RecyclerView.Adapter`.

Il faut donc installer un listener

```
spinner.onItemSelectedListener = object : AdapterView.OnItemSelectedListener {
           override fun onItemSelected(p0: AdapterView<*>?,
                                       p1: View?, position: Int, p3: Long) {
               TODO("a vous d'implementer")
           }
           override fun onNothingSelected(p0: AdapterView<*>?) {
               TODO("on peut laisser non-implemente")
           }
       }
```

Dans `onItemSelected()` vous récupérez la valeur dans le spinner et, si elle est différente de la valeur précédente, vous devez créer un nouvel adapter de `RecyclerView` en lui passant la nouvelle valeur de spinner. Pour changer adapter de `RecyclerView` on peut faire `recyclerView.swapAdapter( newAdapter, false )` et le tour est joué, la liste s'affiche dans un autre ordre.

2 Optionnel : tri par la propriété `checked` de l'étudiant

Ajoutez dans le spinner la troisième entrée : `checked` et ajoutez ce nouveau cas à la méthode `compare` du `callback`

Quand l'utilisateur sélectionne `checked` la liste d'étudiants doit être triée tout d'abord par la valeur de la propriété `checked`, c'est-à-dire d'abord les étudiants avec la valeur `true` ensuite ceux avec la valeur `false`. Dans le groupe avec la même valeur `checked` les étudiants doivent être triés par le nom, et ensuite pour ceux avec le même nom par le prénom. Quand un étudiant est sélectionné/désélectionné sur la liste triée ainsi il doit automatiquement être déplacé sur la liste pour retrouver sa position selon l'ordre.

Problème et solution. Le tri par `checked` devrait maintenant fonctionner normalement. Par contre, quand dans ce mode du tri vous cliquez sur une ligne de la liste le résultat est imprévisible : en fait la `SortedList` ne sait pas qu'un élément a changé. Pour remédier à ça, il faut dans le listener attaché à `CardView`, quand un étudiant `e` (qui est à la position `pos`) change la valeur de `checked`, prévenir la `SortedList` en appelant `updateItemAt(pos, e)` pour qu'il soit déplacé à la bonne position sur la liste.

3 Sauvegarde et lecture de fichier

Ajouter dans le fichier layout de l'activité un bouton `lecture` à côté du bouton `sauvegarde`. Les deux boutons doivent être du même largeur et ensemble doivent s'étendre sur toute largeur de l'appareil.

Quand l'utilisateur clique sur le bouton `sauvegarde` on sauvegarde la liste d'étudiants avec les valeur `checked` dans un fichier texte (vous êtes libre de choisir le nom du fichier) dans la mémoire interne de l'appareil. Une ligne de fichier contient les données d'un étudiant : nom, prénom, et true au false en fonction de la valeur `checked`. Ces trois champs seront séparé par un point-virgule.

Pour tester si la sauvegarde fonctionne, quand l'utilisateur clique sur le bouton `lecture` on lit le fichier et avec `Log.d` on affiche sur le terminal.

Si l'affichage de fichier sur le terminal est correct, on modifie l'action de bouton `lecture` pour que l'affichage soit faite dans le `RecyclerView`.

Indication. Pour écrire dans le fichier il est commode d'avoir un `PrintWriter`. La construction `use` permet de le fermer à la fin

```
val file = File(filesDir, nom_fichier )
PrintWriter( file ).use{
    it.println( une chose)
    it.println( autre chose)
    ...
}
```

Pour lire un `Scanner` fait l'affaire :

```
val file = File(filesDir, nom_fichier )
Scanner(file).use {
   while(it.hasNext()){
     lire une donnee
     ....
   }
}
```