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

TP n° 3

Listes et adaptateurs

Objectif

On a souvent besoin de présenter dans une application Android une liste (ou autre spinner) avec un contenu dynamique. Dans les cas simples, le contenu correspond à un tableau ou une liste dans le programme Kotlin ; c'est ce qu'on fera aujourd'hui (dans les TPs suivants l'information affichée viendra d'une base de données).

1 Ce que l'on va programmer

On vous propose de programmer une application qui gère et affiche dans une `ListView` une liste de phrases.

Le bouton `Action` ajoute à la liste une phrase (saisie par l'utilisateur dans l'`EditText`) si le `RadioButton ajouter` est sélectionné. Ou supprime si c'est le `RadioButton supprimer`. Ensuite, le contenu d'`EditText` est effacé.

On affiche un `Toast` en cas d'erreur de types suivants :

— chaine vide ;
— tentative d'ajout d'une phrase déjà présente ;
— tentative de suppression d'une phrase qui n'est pas dans la liste.

Au lieu d'écrire une suite de `if ... else if ... else if` pour traiter les différents cas on préfère la construction avec `when` :

```
when{
  condition1 -> action1
  condition2 -> action2
  condition3 -> action3
  ....
}
```

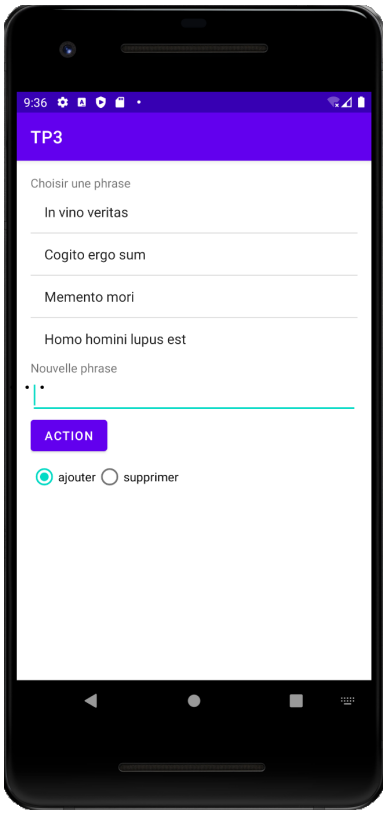

FIGURE 1 – L'application

2 Quelques indications

Le fichier `.xml` pour le design de l'activité principale est fourni.

Petites trivialités

— On teste l'état d'un `RadioButton` avec
`isChecked : Boolean`

— On récupère le texte saisi dans un `EditText e` avec `val s=e.text.toString()`, on l'efface avec `e.text.clear()`, on y met une chaine `s` avec `e.setText(s)`.

Comment gérer la liste de phrases Puisque les données affichées changent en fonction des actions de l'utilisateur il est commode de les stocker dans un objet `MutableList<String>`, comme cela est montré dans l'exemple étudié en cours.

On rappelle comment lier une liste Kotlin avec une `ListView` sur l'écran (voir cours 3 pour plus de détails)

— On instancie la liste des `String`
— On crée un `ArrayAdapter<String>` utilisant le layout fourni par Android `android.R.layout.simple_list` et basé sur la liste des phrases.
— on récupère la `ListView` comme on le fait avec tous le widgets graphique
— on connecte l'`Adapter` à cette `ListView`.
— on ajoute ou supprime des éléments avec les méthodes `add` et `remove` de l'`Adapter`. On en informe avec `notifyDataSetChanged`.

Vous devez à ce moment avoir une application fonctionnelle.

3 Initialisation avec les ressources

Initialement la liste de phrases sera initialisée avec les phrases qui se trouvent dans le tableau défini dans le fichier `strings.xml`. Vous devez copier ce tableau dans le fichier `res/values/strings.xml` de votre application.

Indication :
`resources.getStringArray(R.array.nom_de_tableau_dans_strings).asList()` pour récupérer la liste de mots depuis les ressources.

4 Permettre l'interaction avec la `ListView`

Quand l'utilisateur sélectionne (avec un doigt) un élément dans la `ListView` le `String` sélectionné doit être recopié dans `EditText` (ce qui permet ensuite de le supprimer sans taper la phrase dzans `EditText`). Inspirez-vous du cours pour implémenter cette fonctionnalité dans un petit listener de `ListView`.

5 Trier la liste : `FloatingActionButton`

Dans le fichier layout de l'activité ajouter comme le dernier élément de `LinearLayout` la balise

```
<com.google.android.material.floatingactionbutton.FloatingActionButton
    android:id="@+id/fab"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_gravity="bottom|end"
```

```
android:src="@drawable/sort"
android:layout_marginBottom="16dp"
android:contentDescription="@string/sort"
 />
```

et copier le fichier `sort.png` dans le répertoire `res/drawable` de votre projet. Le fichier png contient l'image affichée dans `FloatingActionButton` (l'attribut `android:src` dans la balise xml).

Ajouter import suivant dans le code Kotlin de votre l'activité

```
import com.google.android.material.floatingactionbutton.FloatingActionButton as FAB
```

Cela ajoute un nouveau `View` : `FloatingActionButton`.

Explication. `as FAB` dans `import` sert à définir un alias. L'alias permet de déclarer la variable qui contient la référence vers le `FloatingActionButton` comme une variable de type `FAB` au lieu d'écrire le nom complet de la classe.

Installer un listener (`OnClickListener`) sur le `FloatingActionButton` de façon à ce qu'un click sur le bouton fasse le tri de la liste de phrases.

Indication. `phrases.sort()` trie le liste de phrases sur place, à ne pas confondre avec `phrases.sorted()` qui retourne une nouvelle liste triées.

Est-ce que le tri est correct, en particulier est-ce que les majuscules et minuscules sont traitées correctement ? Si ce n'est pas le cas il faudra essayer `phrases.sortedWith` qui prend en paramètre un `Comparator` à implémenter par une lambda.

6 Confirmer la suppression

La suppression est une opération irréversible donc dangereuse. L'utilisateur doit confirmer la suppression à l'aide d'un `Snackbar` (`com.google.android.material.snackbar.Snackbar`).

```
Snackbar.make(view, "confirmer suppression", Snackbar.LENGTH_LONG)
                  .setAction("suprimer") { view ->
                      //le code pour supprimer une phrase
                  }
                  .show()
```

7 Changement d'orientation

Quand on tourne l'appareil le contenu de `ListView` disparaît. Vous devez remédier à ce problème.

Vous avez vu dans le cours et TP2, qu'il faut pour cela surcharger `onSaveInstanceState`, et sauvegarder dans son `Bundle` l'état de l'application. Il faut aussi restaurer cet état dans `onCreate`

Le problème est qu'il n'est possible de mettre dans un `Bundle` ni un objet `MutableList<String>` ni un `ArrayAdapter`.

Le `Bundle` possède la fonction `putStringArrayList(key: String, value: ArrayList<String>)` qui permet de mettre dans le `Bundle` un `ArrayList` de `String`s.

Pour fabriquer un `ArrayList<String>` à partir de `mutableList: MutableList<String>` il suffit :

```
val array = ArrayList<String>( mutableList )
bundle.putStringArrayList("key", array )
```

Dans le sens inverse, si `array` est une variable qui contient une référence vers une `ArrayList<String>` et `mutableList` contient une référence vers un objet `MutableList<Array>` alors

```
mutableList.addAll( array )
```

ajoute dans `mutableList` tous les `Strings` qui sont dans `array`