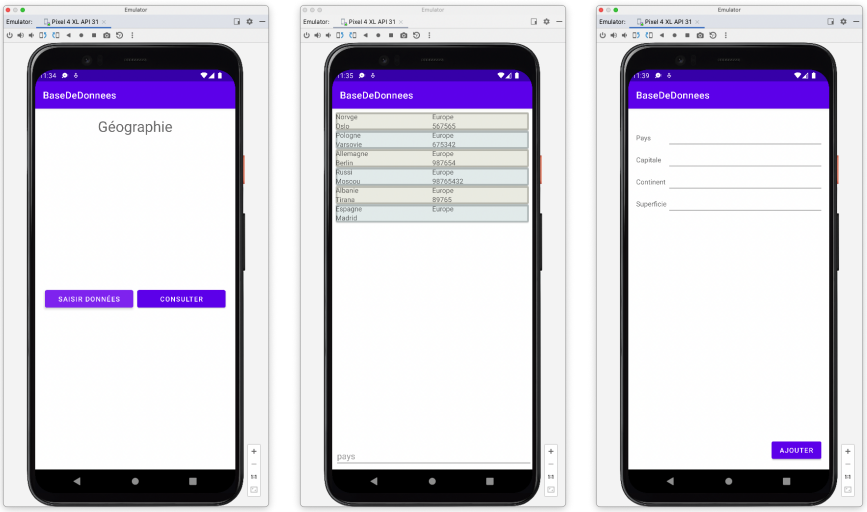
Université Paris Cité **Programmation des composants mobiles**
M1 informatique **Année 2022-2023**

TP n° 7

Base de données Room

Application à programmer

Le but est d'implémenter une simple base de donnée géographique avec une seule table et une petite application capable à faire des insertions et des requêtes SELECT.



L'unique table aura la structure suivante :

`Pays(pays,continent,capitale,superficie)`

Les trois premiers attributs sont de type `String`, le dernier `Int?`. Les noms des attributs indiquent clairement leur rôle. L'attribut `pays` est la clé primaire de la table.

1 Préparatifs et base de données

Reviser rapidement le cours 7. Créer une application avec 3 activités : `MainActivity`, `AjoutPaysActivity`, `RechercheActivity`. Appliquer les 3 layouts fournis.

1.1 On configure `build.gradle(module)`

Mettre en place `ViewBinding` en ajoutant

```
    buildFeatures{
    viewBinding true
}
```

dans la section `android` de `build.gradle` (Module).

Dans la section `plugins` ajouter

```
id 'com.google.devtools.ksp' version "1.7.10-1.0.6"
```

et dans la section `dependencies` ajouter

```
    implementation 'androidx.recyclerview:recyclerview:1.2.1'
    implementation "androidx.cardview:cardview:1.0.0"

    def lifecycle_version = '2.6.0-alpha02'
    // ViewModel
    implementation "androidx.lifecycle:lifecycle-viewmodel-ktx:$lifecycle_version"
    // LiveData
    implementation "androidx.lifecycle:lifecycle-livedata-ktx:$lifecycle_version"

    //Room
    def room_version = "2.4.3"
    implementation "androidx.room:room-ktx:$room_version"
    // To use Kotlin Symbol Processing (KSP)
    ksp "androidx.room:room-compiler:$room_version"
```

1.2 Définition de la table

Créer un nouveau fichier et définir la `data class Pays` avec toutes les annotations nécessaires pour construire la table `Pays`.

1.3 L'interface Dao

Créer, dans un autre fichier, l'interface annotée `@Dao` avec trois méthodes.

— La première doit insérer un `Pays` et retourner une `List<Long>`.
(Rappelons que la fonction d'insertion peut soit retourner rien (Unit) soit retourner une liste de `Long` qui contient les identifiants des éléments insérés (ou `-1L` pour les échecs d'insertion).

— La méthode `loadAll` devra faire une requête `SELECT` qui retournera un `LiveData<List<Pays>>` avec tous les pays.

— La méthode `loadPartialName(nom : String)` qui retourne une `List<Pays>` avec tous les pays dont le nom commence par `nom`.

1.4 Définition de la base de donnée

Maintenant vous pouvez ajouter une classe `GeoBD` pour définir la base de données (avec une unique table `Pays`). Inspirez-vous du cours pour faire des annotations et définir la classe en singleton (c-à-d une classe avec une seule instance qu'on obtient en appelant `getDatabase`)

Comme indiqué en cours vous devez également définir la classe `GeographyApplication` dérivée de `Application` dans laquelle on stockera une référence vers la base de données.

Et n'oubliez pas d'ajouter l'attribut `android:name=".GeographyApplication"` dans la balise `application` de `AndroidManifest.xml` pour que Android utilise votre application à la place de l'application par défaut.

2 Les activités

L'activité principale contient deux boutons qui permettent de lancer deux nouvelles activités `AjoutPaysActivity` et `RechercheActivity`.

2.1 Activité `AjoutPaysActivity` et `AjoutViewModel`

Comme on peut deviner l'activité `AjoutPaysActivity` permet d'ajouter un nouveau Pays dans la table.

La classe `AjoutViewModel` dérivée de `AndroidViewModel` associée à `AjoutPaysActivity` servira à stocker la référence vers `Dao`, la méthode `insert` qui fera l'appel à `insert` de `Dao` et sans doute un attribut de type `LiveData` pour

Les quatre `EditText` servent à entrer les valeurs de 4 attributs (propriétés). Avant de faire une insertion supprimer les espaces au début et à la fin de chaque valeur `String` (à l'aide de la méthode `trim`).

Vérifier que les noms de pays, continent et capitale ne soient pas vide. Si ce n'est pas le cas afficher un dialogue (pas un `Toast`) avec un message indiquant précisément quels champs obligatoires sont encore à remplir. Positionner le curseur sur le premier champs encore vide qui devait être rempli [1].

Si le `TextView` de superficie est vide alors insérer `null` comme la superficie. Après l'insertion, afficher dans un `Toast` un message qui indique soit l'échec d'insertion soit `id` d'élément inséré.

Remarque. La fonction d'insertion de `Dao` retourne une liste avec un seul nombre `Long` : soit $-1L$ en cas d'échec soit `id` d'élément inséré que vous afficherez dans le `Toast`. Rappelons que il est incorrect de faire les affichage sur l'interface graphique dans un autre thread que le thread principal.

Thread en Kotlin. Pour lancer un thread en Kotlin il suffit d'écrire

```
thread{ /* code de la methode run */ }
```

au lieu de

```
Thread{ /*code de la methode run*/ }.start()
```

La fonction `thread` de Kotlin appelle par défaut `start`.

Après l'insertion nettoyez tous les `EditText` et remettez le curseur dans `EditText` du pays.
Insérer au moins 4 `Pays` dans la table.

2.2 `RecyclerView.Adapter`

Pour pouvoir utiliser le `RecyclerView` de l'activité suivante, il faut ajouter et programmer la classe `MyRecAdapter`, en s'inspirant du TP4. Elle utilisera le `item_layout` fourni pour afficher une `List<Pays>`. Cette liste peut initialement être vide, et on pourra la remplacer avec une méthode `setListePays`, à programmer.

1. L'appel `view.requestFocus()` positionne le curseur sur `view`.

Vous devez utiliser le `ViewBinding` dans votre adapter. Regardez le petit document dans le dossier Cours 7 ou le code de l'application étudiée en cours 7.

Remarque. Au lieu d'écrire

```
binding.a....
binding.b....
binding.c....
binding.d....
```

on peut écrire

```
with(binding}{
  a....
  b....
  c....
  d....
}
```

Cela marche aussi à droit de = c'est-à-dire

```
with(binding){
   val x = a...
   val y = b...
   val z = c...
}
```

est équivalent à

```
   val x = binding.a...
   val y = binding.b...
   val z = binding.c...
```

Pour les question de lisibilité les items sur les positions paires et impaires doivent s'afficher avec des couleurs différentes (par exemple `Color.argb(0.1f, 0.3f, 0.3f, 0.0f)` et `Color.argb(0.1f, 0.0f, 0.3f, 0.3f)`.

2.3 Activité `RechercheActivity` er `RechercheViewModel`

Cette activité consiste d'un `RecyclerView` et un `EditText`.

2.3.1 RechercheViewModel

Le `RechercheViewModel` associé doit contenir :
— la référence vers `Dao`,
— une méthode `loadPartialName( nom: String )` qui fait appel à `loadPartialName` dans `dao` pour trouver une liste de pays selon le préfixe. Puisque `loadPartialName` du dao retourne une liste « nue », sans empaquetage dans `LiveData`, l'appel à `loadPartialName` du dao se fera dans un nouveau thread.
— un objet `MutableLiveData<List<Pays>>` pour stocker le résultat de l'appel à `loadPartialName` empaqueté dans `MutableLiveData`.

2.3.2 RechercheActivity

Initialement le `RecyclerView` doit afficher tous les pays présents dans la table.

Nous voulons implémenter le comportement suivant.

Quand utilisateur tape des lettres dans `EditText` c'est un préfixe d'un nom de pays.

Par exemple il commence par taper `R`. Le `RecyclerView` doit maintenant afficher les pays dont le nom commence par `R`. L'utilisateur ajoute la lettre `u` et `EditText` contient maintenant `Ru`. Le `RecyclerView` doit réagir immédiatement et afficher le pays dont le nom commence par `Ru`. L'utilisateur efface `u`, le `RecyclerView` affichera encore une fois les pays dont le nom commence par `R`.

La réaction de `RecyclerView` doit être immédiate après chaque changement de contenu de `EditText`. La technique conseillée consiste à installer un listener sur `EditText`, le bon listener s'appelle TextWatcher et s'installe avec `editText.addTextChangedListener()`.

Le `TextWatcher` possède trois fonctions à implémenter, dont deux, `beforeTextChanged` et `onTextChanged` n'ont pas d'importance pour nous, vous pouvez mettre des implémentations vides. La troisième fonction de `TextWatcher` est `afterTextChanged( editable : Editable )` dont le paramètre est un `Editable` qui contient le texte complet affiché dans `EditText` (appliquez `editable.toString()` pour obtenir un `String`).

Bien sûr vous pouvez [2] vous inspirer de l'exemple étudié en cours et disponible sur moodle.

2. et devez, sauf si vous trouvez mieux