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

TP n° 6

Android Jetpack: ViewBinding, ViewModel, LiveData

Android Jetpack permet d'écrire rapidement des applications bien structurées, en évitant certaines tâches mécaniques. Dans ce TP, sur l'exemple d'une application très très simple, on prendra en main des constituants du Jetpack : `ViewBinding` qui donne accès aux widgets de l'interface graphique sans le fastidieux `findViewById`, et le couple `ViewModel-LiveData` qui structure l'application et permet d'éviter la manipulation du `Bundle` pour pouvoir tourner l'écran.

Dans ce TP on interdit de récupérer les widgets ainsi que gérer le changement de configuration à la main, l'utilisation de Jetpack est obligatoire. Dans les TPs suivants et le projet vous pourrez choisir l'approche qui vous plait (mais l'usage de `LiveData` sera nécessaire pour travailler avec les BD).

1 L'application toute simple

On fera une application qui permet d'écrire un nombre en binaire à l'aide de boutons "0" et "1" et de convertir ce nombre simultanément en décimal. On peut également saisir un nombre décimal et le convertir en binaire en appuyant un bouton, ainsi qu'effacer les nombres décimal et binaire.

Créez l'application, prenez le fichier layout fourni. Ne programmez pas encore le reste.

Figure 1: L'application

2 On fait le binding

Configurez votre projet en ajoutant à la section `android` de `build.gradle(module)` le feature suivant:

```
buildFeatures{
        viewBinding true
        }
```

et n'oubliez pas de synchroniser. Maintenant vous aurez la classe générée `ActivityMainBinding` avec les références à toutes les views de votre layout `activity_main` munis d'un id.

Dans votre code Kotlin de `MainActivity` déclarez un attribut `binding: ActivityMainBinding`. Dans `onCreate` remplacez `setContent...` par

```
binding = ActivityMainBinding.inflate( layoutInflater )
setContentView( binding.root)
```

Ça y est, vous avez accès vers tous les éléments de votre interface graphique. Par exemple `binding.nombreBinaire` est une référence du `TextView` avec id `nombre_binaire`. Ainsi on peut faire

```
binding.boutonZero().setOnClickListener(){
        binding.root.setBackgroundColor(Coler.parsecolor("#FFFF00"))
}
```

Jouez avec le binding, par exemple faites en sorte que l'appui du bouton 1 mette le texte 2022 dans l'`EditText`.

3 On applique maintenant `ViewModel`

Le principe est le suivant. On crée une instance `model` de la classe `MyViewModel`. Toutes les données de l'application seront stockés dans des attributs de `model` de type `MutableLiveData`. Les actions sur l'interface (via des listeners usuels) modifient ces données. D'autre part, l'activité installe des observateurs de ces données, qui changent l'affichage quand la valeur change. Voyons ça en détail.

3.1 On identifie les données à stocker dans le modèle

L'idée de base est de garder l'état de l'application sous la forme d'une chaîne de caractères `cumulBinaire` qui contiendra la suite des "0" et des "1" tapés depuis le dernier effacement. Dans l'exemple de la capture d'écran, on a donc `"00010010"` dans `cumulBinaire`.

3.2 On configure le projet pour ViewModel et LiveData

Dans `build.gradle(module)` on ajoute dans la section `dependencies`

```
def lifecycle_version = "2.5.1"
implementation "androidx.lifecycle:lifecycle-viewmodel-ktx:$lifecycle_version"
implementation "androidx.lifecycle:lifecycle-livedata-ktx:$lifecycle_version"
implementation "androidx.lifecycle:lifecycle-runtime-ktx:$lifecycle_version"
implementation "androidx.lifecycle:lifecycle-viewmodel-savedstate:$lifecycle_version"
implementation "androidx.lifecycle:lifecycle-common-java8:$lifecycle_version"
```

et on synchronise (en demandant de l'aide en cas de problèmes).

3.3 On crée le modèle

Ajoutez à votre projet une classe `MyViewModel : ViewModel()` avec un attribut: une `String` initialisé à `""` et emballée comme `MutableLiveData`, comme ceci.

```
val cumulBinaire=MutableLiveData<String>("")
```

(si la valeur initiale n'était pas disponible on ferait un appel `by lazy`, mais ici ce n'est pas la peine).

3.4 On programme l'activité principale

On déclare l'atribut `model: MyViewModel` et puis dans `onCreate` on l'instancie:

```
model = ViewModelProvider(this).get(MyViewModel::class.java)
```

On programmera le bouton 0 pour qu'il ajoute 0 à la fin de la chaine `cumulBinaire` dans le modèle

```
binding.boutonZero.setOnClickListener {
            val s = model.cumulBinaire.value ?: ""
            model.cumulBinaire.setValue(s + '0')
        }
```

et les boutons 1, Effacer et Convertir (ce dernier lit le nombre décimal, le convertit en binaire et stocke dans `cumulBinaire`). Attention, tous ces listeners de boutons changent l'état mais n'affichent rien!

Maintenant il vous reste d'installer l'observateur de changement de données:

```
model.cumulBinaire.observe(this) {
// afficher le contenu de la String it en binaire et en decimal
...
}
```

Vérifiez que tout fonctionne

3.5 Mode paysage

Vérifiez que l'état de l' application est préservé quand vous tournez l'écran (miracle de LiveData). L'affichage en mode paysage est quand même moche.

On veut que, lorsque le téléphone est tourné, l'application utilise un autre layout.

Pour créer ce layout, dans l'onglet "Design" du fichier `XML` du layout, cliquez avec le bouton droit sur l'icône représentant un téléphone en train de tourner, et choisissez "create Landscape Variation". Il doit apparaître dans la fenêtre "Project", dans le répertoire `layout` avec le nom puis "(land)" entre parenthèses. (Si vous êtes curieux, vérifiez à partir du terminal qu'en fait le fichier est dans le répertoire `layout-land`!)

Ouvrez ce nouveau fichier, et réarrangez le layout pour que tous ses éléments soient visibles (ou utilisez le layout fourni).

4 S'il vous reste un peu de temps — on sauvegarde les préférences

Ajoutez maintenant le réglage de couleur du fond de l'écran, avec un petit `EditText` et un bouton `Appliquer`. L'utilisateur saisit une une couleur RGB codée en héxadécimale, par exemple, le jaune sera codé "FFFF00". Lorsqu'on appuie sur le bouton, cette couleur est appliquée, comme on a fait en section 2 de ce TP.

Faites en sorte que la couleur soit sauvegardée dans les préférences et réutilisée au prochain lancement de l'application (consultez les transparents du cours 5).