Haute École Bruxelles-Brabant
École Supérieure d'Informatique
Bachelor en Informatique

2021 – 2022
MOBG5
JLC - SDR

MOBG5 – Développement mobile

Application Android

Vers une première application

Consignes

Tous les exercices proposés dans ce TD sont disponibles sur `https://codelabs.developers.google.com/android-kotlin-fundamentals/`.
Nous ne ferons pas tous les exercices de ce tutoriel mais seulement une sélection d'exercices essentiels pour développer une première application.
Lisez les explications présentes dans ce TD et réalisez les exercices en ligne.
Les challenges et devoirs (Homeworks) proposés en ligne sont optionnels.

1 Installation de Android Studio

Pour développer une application Android nous proposons de travailler avec Android Studio. L'installation de cet `IDE` est décrite dans le lien ci-dessous.

Lien vers l'exercice

Android Kotlin Fundamentals 01.0 : Install Android Studio [a]

a. https://codelabs.developers.google.com/codelabs/kotlin-android-training-install-studio/index.html

2 Hello World

Ce premier exercice a pour but, comme le veut la tradition, de développer un *Hello World*. Vous pourrez grâce à cette application, constater les similitudes avec des développements réalisés dans d'autres cours.

Le moteur de production utilisé est Gradle.

Si `maven` est utilisé comme moteur de production dans les développements Java, Android utilise `Gradle`. Cet outil va permettre de construire votre projet et de gérer les dépendances via des fichiers `build.gradle`. Afin de faciliter le travail du développeur, Android Studio permet de gérer la majorité des actions sans se soucier des commandes propres à `Gradle`.

Version du document
8 septembre 2021

Afin de créer des écrans, les vues sont décrites via des fichiers XML auxquels sont associés des contrôleurs appelés des Activités.

Durant cet exercice vous aurez l'occasion d'installer l'application *Hello World* sur votre propre téléphone Android. Si vous n'en possédez pas, ou si vous travaillez sur les machines de l'école, passez cette étape de l'exercice.

Lien vers l'exercice

Android Kotlin Fundamentals 01.1 : Get started [a]

a. https://codelabs.developers.google.com/codelabs/kotlin-android-training-get-started/index.html

3 Application `DiceRoller`

Création d'une vue

Dans cet exercice vous allez créer une application de lancé de dé. Voici quelques points importants qui seront utilisés dans l'exercice.

Pour gérer le comportement d'une vue, une classe doit hériter de la classe `AppCompatActivity`.

C'est pourquoi l'en-tête de chaque activité ressemble à :

```
class MyActivity : AppCompatActivity()
```

De cette façon la classe `MyActivity` peut réécrire la méthode `onCreate()` contenue dans la classe parent `AppCompatActivity`. Cette méthode `onCreate()` contient les actions a effectuer lors de la création de la vue. Parmi ces actions on retrouve l'utilisation de ressources (images, sons, vidéo, fichiers xml, etc).

Les ressources sont associées à des entiers servant d'identifiants pour les utiliser dans le code.

Pour accéder à une ressource dans le code on n'utilise pas directement son nom ou son chemin, mais on utilise un identifiant associé à cette ressource. Ces identifiants sont générés lors du `build` et sont stockés sous forme de constantes dans un fichier nommé **R.java**.

Lien vers l'exercice

Android Kotlin Fundamentals 01.2 : Anatomy of Basic Android Project [a]

a. https://codelabs.developers.google.com/codelabs/kotlin-android-training-app-anatomy

Gestion des images

Dans l'exercice suivant vous allez mettre à jour votre application de lancé de dé pour ajouter l'image du dé.

> Les images locales de l'application sont conservées dans le dossier `drawable` du dossier des ressources.

Le dossier *ressources* est divisé en sous-dossiers dont chacun a une fonction précise :

- `color` : pour définir les couleurs ;
- `drawable` : pour sauvegarder les images ;
- `layout` : pour définir les écrans ;
- `values` : pour définir les constantes ;
- `font` : pour définir les fonte de caractères ;
- ...

En plus de ces dossiers de bases, nous verrons que chacun de ces dossiers peut lui-même avoir une sorte de copie. Par exemple vous trouverez les dossiers suivants dans des applications Android qui gèrent différents types de matériels :

- `drawable`
- `drawable-ldpi`
- `drawable-hdpi`

Le dossier `drawable-hdpi` contient les même images que le dossier `drawable`. Cependant elles sont enregistrées dans une meilleure résolution. Lorsque l'application a besoin d'une image, si l'appareil de l'utilisateur possède un écran `High Density`, Android utilisera l'image présente dans le dossier `drawable-hdpi`.

Les différents dossiers utilisés dans les *ressources* sont listés dans la documentation : https://developer.android.com/guide/topics/resources/providing-resources.

> Le `layout` utilisé pour insérer des images est un `ImageView`.

Après le `LinearLayout`, voici un nouveau `layout` a ajouter à votre liste des layout existants.

Les attributs des activités

Au sein du code `Kotlin` vous aurez besoin de conserver des informations en attributs. Cependant le développeur n'a pas accès au constructeur d'une activité, La méthode qui sert de point de départ à une activité est la méthode `onCreate()`. C'est pourquoi vous aurez besoin de préciser à `Kotlin` que les attributs utilisés seront initialisé en déhors du constructeur.

> Une variable non nul déclarée avec le mot clé `lateinit` peut être initialisée après l'instanciation.

Versions d'Android

Dans le `build.gradle` de l'application, vous trouverez la configuration des versions Android utilisées par votre application.

- ▷ `compileSdkVersion` : version utilisée pour compiler le code ;
- ▷ `targetSdkVersion` : version pour laquelle l'application a été pensée et testée ;
- ▷ `minSdkVersion` : version minimale de l'appareil pour utiliser l'application.

La version `minSdkVersion` détermine le nombre d'appareils pouvant utiliser votre application. Vous constaterez que certaines librairies utilisées lors du développement d'une application demandent de modifier ce numéro de version. Ces librairies réduisent le nombre d'utilisateurs potentiels. Il faut réfléchir à cet aspect lors de l'intégration d'une librairie à une application.

Lien vers l'exercice

01.3 : Image resources and compatibility [a]

a. https://codelabs.developers.google.com/codelabs/kotlin-android-training-images-compat/index.html

Utilisation de template

Cet exercice va parcourir les *templates* d'activités existants dans Android Studio et va vous permettre de changer l'icône de votre application.

Lien vers l'exercice

01.4 : Learn to help yourself [a]

a. https://codelabs.developers.google.com/codelabs/kotlin-android-training-available-resources/index.html

4 Application `AboutMe`

Nouveaux `layouts`

Le prochain exercice va vous guider afin d'apprendre à utiliser l'éditeur de `layout`. Il s'agit d'un outil similaire à `SceneBuilder` avec lequel vous êtes familiés. Ce sera l'occasion de découvrir de nouveaux `layouts`.

Un `ViewGroup` est un `layout` pouvant contenir d'autres layout afin de les organiser.

Après le `LinearLayout` et le `ImageView`, vous pouvez ajouter le `ViewGroup` à votre liste de `layout`.

Un `ScrollView` est un `layout` pouvant contenir un autre `layout` ou un `ViewGroup`. Il permet de faire défiler son contenu.

Comme vous le faisiez avec `JavaFX`, vous allez décomposer un écran en plusieurs `layouts`. Un assemblage de `ScrollView`, de `ViewGroup` et de `LinearLayout` permet d'organiser votre écran. Nous compléterons la liste des `layouts` au fur et à mesure mais on peut déjà citer le `ConstraintLayout` et le `GridView` dans les plus utilisés.

Lien vers l'exercice

02.1 : Linear layout using the Layout Editor [a]

a. https://codelabs.developers.google.com/codelabs/kotlin-android-training-linear-layout/index.html

Ajouter des interactions

Dans cet exercice vous apprendrez à gérer les actions associées à un bouton où à récupérer le texte entré par un utilisateur.

Ces notions sont similaires à celles utilisées avec `JavaFX`.

Lien vers l'exercice

02.2 : Add user interactivity [a]

a. https://codelabs.developers.google.com/codelabs/kotlin-android-training-interactivity/index.html

5 Application `ColorMyViews`

Le `ConstraintLayout`

Pour cet exercice vous allez créer une nouvelle application intitulée `ColorMyViews`. L'objectif de cet exercice est de comprendre un `layout` particulier : le `ConstraintLayout`.

Ce `layout` hérite de `ViewGroup`. Son rôle est d'organiser des éléments ensemble en les reliant par différentes contraintes et d'éviter de cette façon la création emboîtée de plusieurs `layouts` [1].

Un peu de style

Un style est un ensemble d'attributs qui caractérisent l'apparence d'une vue. Parmi ces attributs on retrouve, la police, la taille de la police, la couleur de la police, la couleur de fond,... Ces styles sont définis via le fichier `style.xml`.

1. Vous vous rappelez sans doute les imbrications de `VBox`, `HBox` et `GridPane` en `JavaFX`

Lorsque plusieurs styles s'appliquent à un élément, le résultat affiché provient d'une décision d'Android qui classe les styles suivants une certaine hiérarchie : https://developer.android.com/guide/topics/ui/look-and-feel/themes.

Vous découvrirez également la notion de style dans l'exercice suivant.

Lien vers l'exercice

02.3 : Constraint layout using the Layout Editor [a]

a. https://codelabs.developers.google.com/codelabs/kotlin-android-training-constraint-layout/index.html

6 Retour sur l'application `AboutMe`

Dans cet exercice vous allez mettre à jour l'application **`About me`** afin d'apprendre à utiliser le **Data Binding**. Ce mécanisme va nécessiter l'utilisation de **`data class`**.

Une **`data class`** a pour but de transporter des données. Les méthodes **`toString()`**, **`copy()`**, **`equals()`** et **`hashCode()`** sont générées automatiquement.

Principe du Data Binding

Imaginons qu'une vue possède le champs de texte d'id **`R.id.myTextView`** :

```
<TextView android:id="@+id/myTextView" ... />
```

Jusqu'à présent pour qu'une activité mette à jour ce champs de texte, la méthode **`findViewById()`** était utilisée de la manière suivante :

```
myTextView = findViewById(R.id.myTextView)

myTextView.setText("My Awesome Text")

myTextView.visibility = View.VISIBLE
```

Pour chaque composant a modifier, l'id de ce composant (**`R.id.myTextView`**) devait être utilisé. Cette façon de travailler est fastidieuse (le développeur doit rechercher dans le fichier XML l'id à utiliser) et peu performante. En effet la méthode **`findViewById()`** recherche à chaque appel l'objet demandé dans le fichier.

L'idée du **`Data Binding`** est l'existence d'un attribut de type **`androidx.databinding.DataBindingUtil`** dans l'activité. Cet attribut fait référence aux différents objets contenus dans la vue.

Par exemple, nommons cet attribut **`binding`**. Cet attribut permettrait de réécrire les mises à jours de la vue sans faire de recherche via **`findViewById()`** :

```
binding.myTextView.text = "Hello"
binding.myTextView.visibility = View.VISIBLE
```

Kotlin : Scope Function

Les appels à l'attribut de `Data Binding` étant multiples, on peut utiliser les `Scope Functions` de `Kotlin` pour améliorer la lisibilité du code.

De telles fonctions permettent d'appeler un groupe d'instruction dans le contexte d'un objet. Par exemple la méthode `apply` permet de réécrire le code précédent comme :

```
binding.apply {
    myTextView.text = "Hello"
    myTextView.visibility = View.VISIBLE
}
```

Les différentes instructions sont appliquées à l'instance de l'objet `binding`, autrement dit pour notre exemple, chaque appel à l'intérieur du `scope` de la méthode `apply` est précédée de `this.` .

```
binding.apply {
    this.myTextView.text = "Hello"
    this.myTextView.visibility = View.VISIBLE
}
```

De plus la méthode `binding.apply` retourne l'instance de l'objet `binding`. Des appels chaînés peuvent être construits facilement.

Sachez que plusieurs `Scoped Functions` existent : `let`, `with`, `run` et `also`. Elles se différencient par le contexte et l'objet retourné. Plus d'informations via https://kotlinlang.org/docs/reference/scope-functions.html.

Le second avantage du `Data Binding` est l'utilisation de `data class` provenant des activités **directement dans la vue**. Par exemple, il est possible de créer une `data class Person` pour l'utiliser dans la vue :

```
data class Person(var firstName: String = "", var lastName: String = "")
```

Si on instancie la classe `Person` dans l'activité et qu'on l'associe à l'attribut `person` via :

```
var person = Person("SpongeBob", "SquarePants")
```

L'attribut `person` est maintenant accessible dans la vue directement via le fichier XML :

```
<TextView android:id="@+id/myTextFirstView"
    android:text="@={person.firstName}"
 />
<TextView android:id="@+id/myTextSecondView"
    android:text="@={person.lastName}"
/>
```

Tout changement effectué dans l'attribut `person` est **automatiquement** reporté sur la vue. Il n'est plus utile d'appeler les mutateurs des objets `TextView` dans l'activité.

Comme vous le verrez dans l'exercice pour que ces liens entre activité et vue soient opérationnels des modifications sont à apporter :

- ▷ mise à jour du fichier `build.gradle` ;
- ▷ utilisation de la balise `layout` au sein de la vue ;
- ▷ utilisation de la balise `data` au sein de la vue.

Lien vers l'exercice

02.4 : Data binding basics [a]

a. https://codelabs.developers.google.com/codelabs/kotlin-android-training-data-binding-basics/index.html

7 Application `AndroidTrivia`

Utilisation des fragments

Durant cet exercice vous allez télécharger une nouvelle application et la modifier afin d'ajouter un `fragment`.

Le concept d'activité permet de concevoir des applications simples, mais dès que l'on souhaite développer une application comportant plusieurs écrans possédant des liens entre eux, la notion d'activité atteint rapidement ses limites.

La notion de fragment a été introduite avec l'arrivée des premières tablettes `Android`, lorsque le besoin d'interfaces plus élaborées s'est fait sentir. Un fragment est un 'morceau' d'interface, un `layout` et son `contrôleur`. Un écran est défini par une activité qui elle-même est composée d'un ou plusieurs fragments. Les fragments peuvent être manipulés, retirés ou ajoutés. Un fragment peut également être composé d'autres fragments.

Lien vers l'exercice

03.1 : Create a fragment [a]

a. https://codelabs.developers.google.com/codelabs/kotlin-android-training-create-and-add-fragment/index.html

Gestion de la navigation

Vous allez continuer le développement de l'application `AndroidTrivia` afin d'y ajouter plusieurs fragments. Le but de l'exercice va être de gérer les transitions d'un fragment à un autre.

Pour ce faire vous allez créer le **graphe de navigation** via un outil inclus dans `Android Studio`. Ce graphe sera enregistré dans une nouvelle ressource au format `xml`.

La gestion des transitions comprises, il sera alors temps de définir les actions associées aux boutons de retour arrière.

Lien vers l'exercice

03.2 : Define navigation paths [a]

a. https://codelabs.developers.google.com/codelabs/kotlin-android-training-add-navigation/index.html

Appeler une application extérieure

Afin de compléter l'application `AndroidTrivia`, vous allez ajouter la possibilité d'envoyer les résultats via mail.

Pour ce faire l'application de gestion de mail du téléphone doit être démarrée et des données doivent lui être transmises. Le transfert de ces données ce fait via un objet d'un nuveau type : `SafeArgs`.

Lien vers l'exercice

03.3 : Start an external activity [a]

a. https://codelabs.developers.google.com/codelabs/kotlin-android-training-start-external-activity/index.html