**HelHa**
Tournai
Bachelor en Informatique

2019 - 2020
Android

# Développement Android

## Première application

La majorité des travaux présentés dans ce cours suivent le livre [1]

**Android Programming : The Big Nerd Ranch Guide (3rd Edition)** [2].



1. https://www.bignerdranch.com/books/android-programming/
2. **Android Programming : The Big Nerd Ranch Guide (3rd Edition)**, de Bill Phillips, Chris Stewart et Kristin Marsicano, édité en 2017, Big Nerd Ranch, ISBN=978-0134706054

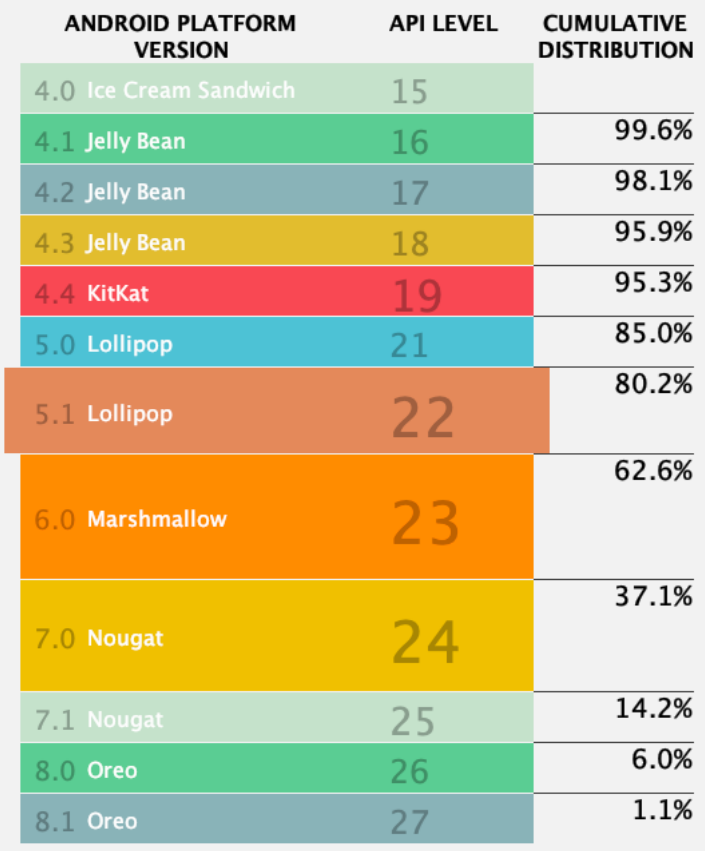


Version du document : 6 août 2019

# 1 Installation de l'environnement de travail

Installez Android Studio, l'IDE fourni par Google pour créer des applications Android. Il est disponible depuis l'adresse https://developer.android.com/studio.

Il faut aussi installer un SDK (Software Development Kit). Il en existe beaucoup de versions, comme le montre l'image suivante :



| ANDROID PLATFORM VERSION | API LEVEL | CUMULATIVE DISTRIBUTION |
|---|---|---|
| 4.0 Ice Cream Sandwich | 15 | |
| 4.1 Jelly Bean | 16 | 99.6% |
| 4.2 Jelly Bean | 17 | 98.1% |
| 4.3 Jelly Bean | 18 | 95.9% |
| 4.4 KitKat | 19 | 95.3% |
| 5.0 Lollipop | 21 | 85.0% |
| 5.1 Lollipop | 22 | 80.2% |
| 6.0 Marshmallow | 23 | 62.6% |
| 7.0 Nougat | 24 | 37.1% |
| 7.1 Nougat | 25 | 14.2% |
| 8.0 Oreo | 26 | 6.0% |
| 8.1 Oreo | 27 | 1.1% |

La colonne de droite montre le pourcentage de devices qui supportent la version dans le monde. Au plus nous prenons une version récente, au moins nous avons de devices sur lesquels cela peut fonctionner. Pour notre cours, nous allons utiliser sur le SDK 22 (sous le nom commercial : Android Lollipop 5.1).

Installez la version 22 du SDK depuis le manager de SDK inclus dans Android Studio.

# 2 Création d'un nouveau projet : MainActivity

Dans ce premier projet, vous allez implémenter une petite application permettant de jouer à un Quiz.


## 2.1 Choisir son type d'Activité

Il est désormais possible de créer des applications Android en Java et en Kotlin (un langage qui tent à remplacer Java dans les années futures). Nous vous demandons d'utiliser le langage Java dans ce cours.

Créez un nouveau projet 'Quiz' avec une 'Empty Activity'. Attendez que tout soit chargé.

Android Studio va vous créer un projet avec un tas de fichiers, dont un fichier java `MainActivity.java` qui est le contrôleur de votre application. Ce contrôleur est lié au layout qui est défini dans le fichier `/res/layout/activity_quiz.xml`. Recherchez et examinez ces deux fichiers [3].

3. Petite astuce : appuyez 2 fois sur votre touche `Shift` pour ouvrir un menu vous permettant de trouver rapidement un fichier via son nom ou son contenu.

Compilez et lancez cette première application. Vous allez devoir créer un émulateur ou le faire tourner sur un device Android physique (via USB). Pour l'émulateur, créez un machine virtuelle avec une version du SDK plus grand que 22 (par exemple 24, Nougat).

`git init`, `git commit`, `git push`.

Modifiez ensuite le texte affiché, la couleur de fond et la police et vérifiez que cela fonctionne.

`git commit`, `git push`.

## 2.2 Premier Layout

Affichez le fichier `activity_quiz.xml`

Et modifiez le comme suit :

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:gravity="center"
    android:orientation="vertical">
       <TextView
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:padding="24dp"
            android:text="@string/question_text"/>
       <LinearLayout
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:orientation="horizontal">
            <Button
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:text="@string/true_button" />
            <Button
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:text="@string/false_button" />
        </LinearLayout>
</LinearLayout>
```

Remarquez les valeurs des attributs `android:text`, par exemple `@string/false_button`. L'arobase indique qu'il s'agit d'une référence à une ressource de type `String`. On retrouve ces ressources dans le répertoire `res/values/strings.xml`.

Ajoutez à ce fichier les ressources `question_text` la question posée, `true_button` le label du bouton 'true', et `false_button` le label du bouton 'false'. Par exemple ajoutez la ligne

```
<string name="question_text">Canberra is the capital of Australia</string>
```

Les fichiers contenant des strings (il peut y en avoir plusieurs) doivent se trouver dans le répertoire `res/values`, et le document xml de ce fichier a comme racine l'élément `resources`.

Compilez et lancez votre application. L'écran s'affiche mais les boutons ne répondent pas (pas encore...).

`git commit`, `git push`.

**Question 1**

Expliquez les attributs ci-dessous :

- match_parent ;
- wrap_content ;
- android :orientation ;
- android :text.

# 3 Ressources et ID

Une ressource est une partie de votre application qui n'est pas du code (images, sons, vidéo, fichiers xml, etc). Les ressources de votre projet se trouvent dans le répertoire `app/res`. Pour accéder à une ressource dans le code (java) on utilise l'identifiant de cette ressource. Ces identifiants sont générés lors du *build* et sont stockés sous forme de constantes dans le fichier java `R.java`. Il ne faut donc jamais modifier 'à la main' ce fichier.

**Question 2**

1. Où se trouve le fichier `R.java` ? Utilisez la vue 'Project' (menu déroulant en haut à gauche), retournez ensuite à la vue par défaut 'Android'.
2. Dans le fichier `R`, quel est l'identifiant de la ressource de type *layout* se nommant `activity_quiz` ? Donnez sa valeur numérique et le nom de la variable associée (static et final).
3. Nous avons vu que les strings sont aussi des ressources. Dans le fichier `R`, quel est l'identifiant (valeur et nom de la constante) de la string `true_button` ?

## 3.1 Associer des IDs aux widgets

Vérifiez que la ressource `true_button` n'apparaît qu'une seule fois dans le fichier R. C'est la ressource de type string qui contient le texte du bouton.

Après être retourné à la vue 'Android' (menu dropdown en haut à gauche d'Android Studio), vous allez ajouter des IDs aux boutons. Dans `activity_quiz.xml` ajoutez l'attribut au bouton, par exemple pour le bouton 'true', ajoutez l'attribut :

`android:id="@+id/true_button"`

Faites de même pour le bouton 'false'. Le '+' indique au compilateur qu'il faut créer l'identifiant et donc l'ajouter au fichier `R.java`. Compilez votre projet et vérifiez que les nouveaux identifiants ont bien été créés dans le fichier R : vérifiez que la ressource `true_button` apparaît deux fois dans le fichier R, une fois pour la ressource de type string et une fois pour votre bouton.

Le fichier R est composé de plusieurs classes `static`.

**Question 3**

1. Dans quelle classe du fichier R apparaissent les `string` ?
2. Dans quelle classe du fichier R apparaissent les `id` de vos boutons ?
3. Dans quelle classe du fichier R apparaissent l'`id` de votre layout `activity_quiz` ?

## 3.2 Associer des objets aux widgets

Maintenant que les boutons ont des identifiants vous pouvez y accéder dans votre code java.

Pour cela, dans votre classe `MainActivity` ajoutez des attributs `mTrueButton` et `mFalseButton` de type `Button` :

```
private Button mTrueButton;
```

Le prefix 'm' dans 'mTrueButton' est une convention Android pour désigner les attributs (membres) d'une classe.

**Question 4**

1. Quelle est le package de la classe Button ?
2. Donnez 5 autres classes de ce package qui vous sont familières, par exemple des classes proches des classes Javafx.

Il faut maintenant récupérer les références de chaque bouton et les assigner aux attributs correspondant. Pour cela, on ajoute le code suivant à la méthode `onCreate` de la classe.

```
mTrueButton = findViewById(R.id.true_button);
mFalseButton = findViewById(R.id.false_button);
```

Notez l'utilisation de la classe `R`, on voit qu'on accède à la constante `true_button` de la classe static `id` qui est une classe imbriquée à la classe `R`.

**Question 5**

1. Dans quelle classe est définie la méthode `findViewById` ?
2. Quel est son type de retour ?

# 4 Gérer les événements

Nous avons maintenant accès aux boutons et nous allons ajouter la gestion des événements. Pour cela, on suit la manière standard en Java, il suffit d'ajouter des *Listener*.

Toujours dans la méthode `onCreate` du contrôleur, on ajoute aux boutons un écouteur :

```
mTrueButton.setOnClickListener(new View.OnClickListener()...
```

La fonction de complétion de code d'Android Studio vous permettra d'écrire ce bout de code correctement en utilisant une classe anonyme.

Dans la méthode `onClick` du Listener, vous ne faites encore rien, votre code ne réagit pas encore aux clics.

**Question 6**

1. Quel type d'élément est `View.OnClickListener`, une classe, une méthode, un constructeur, ou autre chose ?
2. Dans quel package se trouve cet élément ?

Ajoutez dans la méthode `onClick` le code suivant qui permet de produire un log :

```
Log.d("MainActivity", "true button clicked");
```

Ce log est visible dans la console 'Logcat' d'Android Studio. Compilez et lancez votre application afin de vérifier que le log apparaît bien dans la console.

`git commit`, `git push`.

**Question 7**

1. Quelle est le package de la classe `Log` ?
2. Quelles sont les méthodes disponibles dans la classe `Log` ?

# 5 Notifier l'utilisateur via un Toast

Nous allons annoncer à l'utilisateur s'il a donné ou non la bonne réponse via un *Toast*. Un Toast est un court message qui apparaît à l'écran et y reste quelques secondes

avant de disparaître. Votre application va faire apparaître le message 'Correct !' lorsque l'utilisateur presse sur 'TRUE' et le message 'Incorrect !' lorsque l'utilisateur clique sur 'FALSE'.

Tout d'abord ajoutez des strings à votre application. Dans le fichier `strings.xml` ajoutez la ligne :

```
<string name="correct"> Correct!</string>
```

Faites pareil pour le message incorrect.

Pour créer un toast on ajoute la ligne suivante à la méthode `onClick` du listener du bouton `true_button` :

```
Toast.makeText(MainActivity.this, R.string.correct, Toast.LENGTH_SHORT).show();
```

Faites apparaître un toast également lorsque le bouton 'FALSE' est cliqué, mais cette fois-ci avec le message 'Incorrect !'.

`git commit`, `git push`.

# 6 Modèle-Vue-Contrôleur (MVC)

Nous voulons enrichir l'application afin d'avoir un ensemble de questions, et non plus une seule question. Avant cela nous allons remanier légèrement le code. Nous allons introduire une architecture MVC. La vue (`activity_quiz.xml`) et le contrôleur (`MainActivity.java`) sont bien définis mais le contrôleur remplit également le rôle du modèle. Nous allons donc séparer le modèle du contrôleur.

## 6.1 Le modèle

Créez une nouvelle classe `Question` comprenant comme attributs

- `mTextResId` un entier qui est l'identifiant de la question. Plus exactement, l'ID de la ressource de type string comprenant le texte de la question ;
- `mAnswerTrue` un booléen qui indique si la réponse est 'true' ou non.

Ajoutez le constructeur initialisant les 2 attributs ainsi que les accesseurs (getters).

`git commit`, `git push`.

## 6.2 Mise à jour de la vue

Nous allons maintenant mettre à jour la vue, nous allons :

1. Ajouter un bouton 'Next' pour passer à la question suivante.
2. Donner accès au TextView pour pouvoir modifier le texte de la question.
3. Ajouter le texte des questions dans le fichier `strings.xml`.

Dans le fichier `activity_quiz.xml` ajoutez le code suivant définissant le bouton 'Next' :

```
<?xml version="1.0" encoding="utf-8"?>
        <LinearLayout ... >
                <LinearLayout ... >
                        (...)
                </LinearLayout>
                <Button
                        android:id="@+id/next_button"
                        android:layout_width="wrap_content"
                        android:layout_height="wrap_content"
                        android:text="@string/next_button" />
        </LinearLayout>
```

Ajoutez également la ressource string pour le label du bouton (`@string/next_button`) dans le fichier `strings.xml`.

Pour donner accès au `TextView` nous devons lui donner un id. Toujours dans le fichier XML, ajoutez au `TextView` l'attribut :

`android:id="@+id/question_text_view"`

**Question 8**

1. Où allons-nous utiliser l'id du bouton 'Next' ?
2. Quand allons-nous utiliser l'id du TextView ?

Il nous reste à ajouter les questions dans le fichier `strings.xml`.

```
<string name="question_australia">
Canberra is the capital of Australia.
</string>
<string name="question_oceans">
The Pacific Ocean is larger than the Atlantic Ocean.
</string>
<string name="question_mideast">
The Suez canal connects the Red Sea and the indian Ocean.
</string>
<string name="question_africa">
The source of the Nile River is in Egypt.
</string>
<string name="question_americas">
The Amazon river is the longest river in the Americas.
</string>
<string name="question_asia">
Lake Baikal is the world\'s oldest and deepest freshwater lake.
</string>
```

Compilez et lancez votre application, le nouveau bouton doit apparaître.

`git commit`, `git push`.

### 6.3 Mise à jour du contrôleur

Il nous reste à mettre à jour le contrôleur. Dans `MainActivity.java` ajoutez les attributs suivants :

- `mNextButton` une référence pour le bouton next ;
- `mQuestionTextView` une référence pour le TextView ;
- `mCurrentIndex` le numéro de la question courante, initialisé à 0 ;
- `mQuestionBank` un tableau contenant toutes les questions du quiz.

Le tableau de questions est directement initialisé comme suit :

```
private Question[] mQuestionBank = new Question[] {
        new Question(R.string.question_australia, true),
        new Question(R.string.question_oceans, true),
        new Question(R.string.question_mideast, false),
        new Question(R.string.question_africa, false),
        new Question(R.string.question_americas, true),
        new Question(R.string.question_asia, true),
};
```

L'initialisation des attributs se fait dans la méthode `onCreate`.

1. Initialisez l'attribut `mQuestionTextView` en assignant le TextView à l'aide de la méthode `findViewById` que vous connaissez déjà. Initialisez le texte du TextView en récupérant le texte de la première question.
2. Initialisez l'attribut `mNextButton` en lui assignant l'objet de type `Button` correspondant. Gérez le clic à l'aide d'un Listener. Lorsque le bouton est cliqué, on passe à la question suivante (`mCurrentIndex`) et on l'affiche dans le TextView.
3. Finalement mettez à jour les listener des boutons 'TRUE' et 'FALSE' pour qu'il affiche le bon message en fonction de la réponse à la question courante.

Compilez et testez votre application.

`git commit`, `git push`.

## 7 Exercices

Retravaillez votre application afin d'avoir une architecture MVC complète. Entre autres, gérez la liste de questions dans le modèle et non dans le contrôleur comme c'est le cas actuellement.

Vous pouvez ajouter une *Façade* par laquelle passe toute l'interaction avec le contrôleur.

On vous demande d'ajouter le score du joueur pour la session en cours (par exemple 5/8).

Ajoutez également un bouton 'end session' pour terminer la session (réinitialise le score à 0).