

Développement Android

Labo 1 : 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	Installation de l'environnement de travail	2
2	Installation du SDK	2
3	Création d'un nouveau projet : MainActivity	3
3.1	Choisir son type d'Activité	3
3.2	Premier Layout	4
4	Ressources et ID	5
4.1	Associer des IDs aux widgets	6
4.2	Associer des objets aux widgets	6
5	Gérer les événements	7
6	Notifier l'utilisateur via un Toast	8
7	Modèle-Vue-Contrôleur (MVC)	9
7.1	Le modèle	9
7.2	Mise à jour de la vue	9
7.3	Mise à jour du contrôleur	10
8	Exercices	11

1. https://www.amazon.com/Android-Programming-Ranch-Guide-Guides/dp/0135245125/ref=dp_ob_title_bk

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

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>.

Aide

Une fois installé, suivez le texte ci-dessous. Si vous êtes perdu ou si vous voulez plus d'informations, vous pouvez suivre la vidéo d'explications sur ce labo disponible ici ^a :

<https://youtu.be/8MCuCdf5wos>

Le code résultat de ce labo est disponible ici :

<https://github.com/fpluquet/ue3103-android-labo1>

a. La vidéo date de 3 ans mais l'essentiel est identique. Elle sera refaite l'année prochaine.

2 Installation du SDK

Il faut 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
4.4	KitKat	19	
5	Lollipop	21	99,7%
5.1	Lollipop	22	99,6%
6	Marshmallow	23	98,8%
7	Nougat	24	97,4%
7.1	Nougat	25	96,4%
8	Oreo	26	95,4%
8.1	Oreo	27	93,9%
9	Pie	28	89,6%
10	Q	29	81,2%
11	R	30	67,6%
12	S	31	48,6%
13	T	33	33,9%
14	U	34	13,0%

Last updated: May 1, 2024

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 24 (sous le nom commercial : Android Nougat).

Installez la version 24 du SDK depuis le manager de SDK inclus dans Android Studio.

3 Création d'un nouveau projet : MainActivity

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



3.1 Choisir son type d'Activité

Il est désormais possible de créer des applications Android en Java et en Kotlin (un langage qui tend à remplacer Java dans les années futures). Nous allons utiliser le langage Java dans ce cours mais vous pouvez utiliser Kotlin si vous le désirez.

Créez un nouveau projet **Quiz** avec une **Empty Views Activity** (et pas **Empty Activity**) et choisissez bien le langage Java. Attendez que tout soit chargé (cela peut prendre quelques minutes la première fois).

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_main.xml**. Recherchez et examinez ces deux fichiers³.

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 une machine virtuelle avec une version du SDK plus grand que 24 (par exemple 34).

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.

git init, git commit, git push⁴.

Modifiez ensuite le texte affiché, la couleur de fond et la police et vérifiez que cela fonctionne. Remarquez que vous pouvez modifier le fichier .xml manuellement ou utiliser l'éditeur d'interface intégré à Android Studio.

git commit, git push.

3.2 Premier Layout

Affichez le fichier activity_main.xml

Et modifiez le comme suit :

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/main"
    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 que les valeurs des attributs `android:text` sont de la forme `@string/key`, comme par exemple `@string/false_button`. La partie `@string/` indique qu'il s'agit d'une référence à une ressource de type `String`. Cette ressource se trouve dans le répertoire `res/values/strings.xml`. C'est un fichier XML qui va reprendre l'ensemble des clés et leurs valeurs. Ici, comme les clés `false_button`, `true_button`

4. Nous mettons à des moments clés l'enregistrement de la version de votre projet sur un repository git de votre choix. Ce n'est pas obligatoire mais cela vous habitue à utiliser git.

et `question_text` ne sont pas définies, Android Studio les souligne en rouge. Nous allons donc les ajouter dans le fichier correspondant.

Nous aimerions que la valeur de :

- `question_text` soit La capitale de l'Australie est Canberra.,
- `true_button` soit Vrai,
- et `false_button` soit Faux.

Pour les ajouter, il y a deux possibilités :

- soit vous cherchez le fichier `res/values/strings.xml` et vous l'éditez en y ajoutant les valeurs suivantes :

```
<string name="question_text">La capitale de l'Australie est Canberra.</string>
<string name="true_button">Vrai</string>
<string name="false_button">Faux</string>
```

- soit vous cliquez droit sur les clés soulignées en rouge, sélectionnez **Show Context Actions** puis **Create string resource value ...** et vous renseignez la bonne valeur pour la clé correspondante.
Vérifiez que le fichier `res/values/strings.xml` a alors été correctement modifié.

Sachez que vous pouvez avoir plusieurs fichiers contenant des strings, qu'ils doivent se trouver dans le répertoire `res/values`, et le document XML de ce fichier a comme racine l'élément `resources`.

Il y a beaucoup d'avantages à séparer les strings de l'XML de la vue : la simplicité de maintenance, la séparation des responsabilités, l'internationalisation, la réutilisabilité, ...

Compilez et lancez votre application. Vérifiez que l'écran s'affiche correctement. Les boutons ne répondent pas (...encore :)).

`git commit`, `git push`.

Question 1

Expliquez les termes suivants, utilisés dans la définition XML du layout :

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

4 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 `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 un fichier Java `R.java`. Il ne faut donc jamais modifier *à la main* ce fichier.

Ce fichier est difficilement accessible en fouillant le fichier `.dex` du build de votre application (qui correspond à une archive contenant tous les fichiers de votre application).

Retenez que lors du build, chaque ressource a été liée à un identifiant entier, que nous allons utiliser pour faire le lien avec notre code Java.

4.1 Associer des IDs aux widgets

Un widget est une partie de l'application (un label, un bouton, ...). Chaque widget a des propriétés communes (comme un identifiant) et des propriétés propres. Nous allons fixer les IDs pour nos différents widgets.

Dans `activity_main.xml`, choisissez la vue **Design**, sélectionnez le bouton **Vrai** et cherchez la propriété `id`. Fixez alors sa valeur comme `true_button`.

Allez désormais voir le fichier dans la vue **Code**. Nous pouvons remarquer l'ajout de l'attribut `android:id="@+id/true_button"` dans le bouton. Le `+` indique au compilateur qu'il faut créer l'identifiant et donc l'ajouter automatiquement au fichier `R.java`.

Encore une fois, le build va associer un identifiant entier à l'id `true_button` que nous allons pouvoir utiliser dans notre code Java.

Faites de même pour le bouton **Faux** (id : `false_button`) et le label **Question** (id : `question_text_view`).

4.2 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;  
private Button mFalseButton;
```

Le préfixe `m` dans `mTrueButton` est une convention Android pour désigner les attributs (membres) non publics et non statiques d'une classe. Le style de code à suivre pour Android est défini ici : <https://source.android.com/setup/contribute/code-style>.

Question 2

1. Quel 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` ^a.

^a. Pour parcourir les différentes classes du package, il faut demander à Android Studio de télécharger les sources du SDK, car par défaut, il ne contient que les classes compilées.

Comme nous l'avons vu en Java, la déclaration d'attributs ne les initialise pas. Il faut maintenant récupérer les références de chaque bouton et les assigner aux attributs correspondants. Pour cela, on ajoute le code suivant à la méthode `onCreate` de la classe, après l'appel de `setContentView`.

```
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 statique `id` qui est une classe imbriquée à la classe `R`.

Question 3

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

Pour comprendre ce qu'il s'est passé :

- nous avons un fichier XML qui définit les différents widgets de la vue,
- ce fichier est chargé via la méthode `setContentView(R.layout.activity_main)` ; qui va lire le fichier, transformer les informations contenues en objets Java (instances de sous-classes de `View`) et y appliquer tous les attributs renseignés,
- on récupère les objets qui nous intéressent (comme les deux boutons) en utilisant la méthode `findViewById(id)`.

Important

Il faut toujours utiliser `findViewById` après avoir appelé `setContentView` puisque `setContentView` crée les objets depuis l'XML tout en les associant avec leur id. La méthode `findViewById` ne pourra donc retrouver l'objet associé à un id qu'après que celui-ci soit créé.

5 Gérer les événements

Nous avons maintenant accès aux boutons et nous allons ajouter la gestion des événements. Pour cela, 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.

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

Question 4

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", "Clic sur le bouton Vrai");
```

Ce log est visible dans la console `Logcat` d'Android Studio (en bas à gauche).

Compilez et lancez votre application afin de vérifier que le log apparaît bien dans la console.

`git commit`, `git push`.

Question 5

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

6 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 **Vrai** et le message **Incorrect** ! lorsque l'utilisateur clique sur **Faux**.

Tout d'abord, ajoutez des strings à votre application. Dans le fichier `strings.xml`, ajoutez les lignes :

```
<string name="good_answer">Correct !</string>
<string name="wrong_answer">Incorrect !</string>
```

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.good_answer, Toast.LENGTH_SHORT).show();
```

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

git commit, git push.

7 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_main.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.

7.1 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 contenant le texte de la question ;
- `mAnswerTrue` un booléen qui indique si la réponse est vraie ou fausse.

Ajoutez le constructeur initialisant les 2 attributs ainsi que les accesseurs (getters) (clic droit, **Generate**, **Constructor**, ...). La convention de nommage des getters/setters est d'omettre le préfixe `m`. Donc pour `mTextResId`, nous aurons donc `getTextResId()` (et non `getmTextResId()`) et `setTextResId(int)`. Pour les variables booléennes, un getter ne commencera pas par `get` mais par `is` : `isAnswerTrue()` et `setAnswerTrue(boolean)`.

Mettez la classe `MainActivity` dans un package `quiz.controllers` (clic droit, **Refactor**, **Move class...**, ...) et la classe `Question` dans un package `quiz.models`.

git commit, git push.

7.2 Mise à jour de la vue

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

1. ajouter un bouton **Suivant** 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 `activity_main.xml`, ajoutez le code suivant définissant le bouton **Suivant** :

```
<?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`.

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

```
<string name="question_australia">
    Canberra est la capitale de l\'Australie.</string>
<string name="question_oceans">
    L\'océan Pacifique est plus large que l\'océan Atlantique.</string>
<string name="question_mideast">
    Le canal de Suez relie la mer rouge et l\'océan Indien.</string>
```

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

`git commit`, `git push`.

7.3 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 Suivant,
- `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),
};
```

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 `Vrai` et `Faux` 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.`

8 Exercices

- Retravaillez votre application afin d'avoir une meilleure séparation modèle/-contrôleur. Entre autres, gérez la liste de questions dans le modèle et non dans le contrôleur, comme c'est le cas actuellement : ajoutez une classe `Quiz` qui gère les questions, la question courante, ...
- Ajoutez le score du joueur pour la session en cours (par exemple 2/3).
- Ajoutez également un bouton `Terminer` pour terminer la session (réinitialise le score à 0).
- Faites en sorte que le joueur ne puisse donner qu'une seule réponse (désactivez les boutons).