

# COURS ANDROID

## Configuration de l'environnement

---

Avant de commencer, nous allons paramétrer légèrement l'IDE selon vos goûts. Pour ce faire, cliquez en bas sur **Configure** puis sur **Preferences**. N'oubliez pas de cliquer sur le bouton **Apply** en bas de la fenêtre pour appliquer les modifications.

### Thème

Pour configurer le thème principal, rendez-vous dans **Appearance & Behavior > Appearance > UI Options > Theme**. Vous aurez le choix entre le thème par défaut, d'apparence clair, et le thème *Darcula*, d'apparence sombre. Si vous voulez passer pour un vrai pro, choisissez le thème *Darcula*.

## Création du projet

---

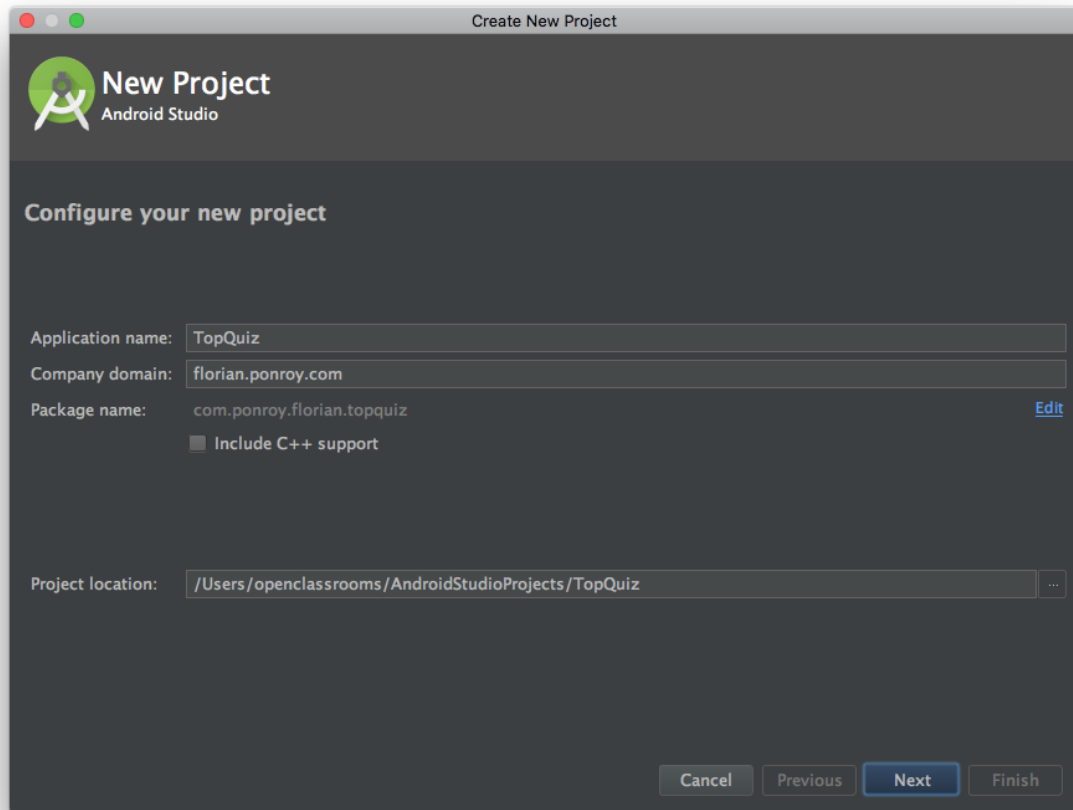
Après avoir lancé Android Studio, cliquez sur **Start a new Android Studio project**.

### Configuration du projet

Dans le champ **Application name**, vous allez saisir le nom de l'application. Par défaut, ce sera le nom qui apparaîtra en dessous de l'icône de l'application sur l'écran d'accueil du téléphone, et dans la barre de titre de l'application. Il vous sera tout à fait possible de le modifier par la suite. Saisissez *TopQuiz* (ou tout autre nom que vous trouvez mieux).

Le champ **Company domain** permet de déterminer quel nom de paquetage utiliser pour votre application. Cela permet par la suite de distinguer votre application d'une autre application qui porterait le même nom. Par convention, la notation inverse est utilisée. Par exemple, si vous travaillez dans la société *WorldCompany*, vous pourriez préciser *android.world-company.com*. Après, vous êtes libre de préciser le nom de votre choix. Évitez simplement d'utiliser un nom de domaine qui ne vous appartient pas, afin de ne pas être confronté-e à un doublon le jour où vous souhaitez publier votre application sur le Google Play Store.

Laissez la case **Include C++ support** décochée, modifiez éventuellement le chemin du projet dans **Project location** puis appuyez sur **Next**.



## Plates-formes cibles

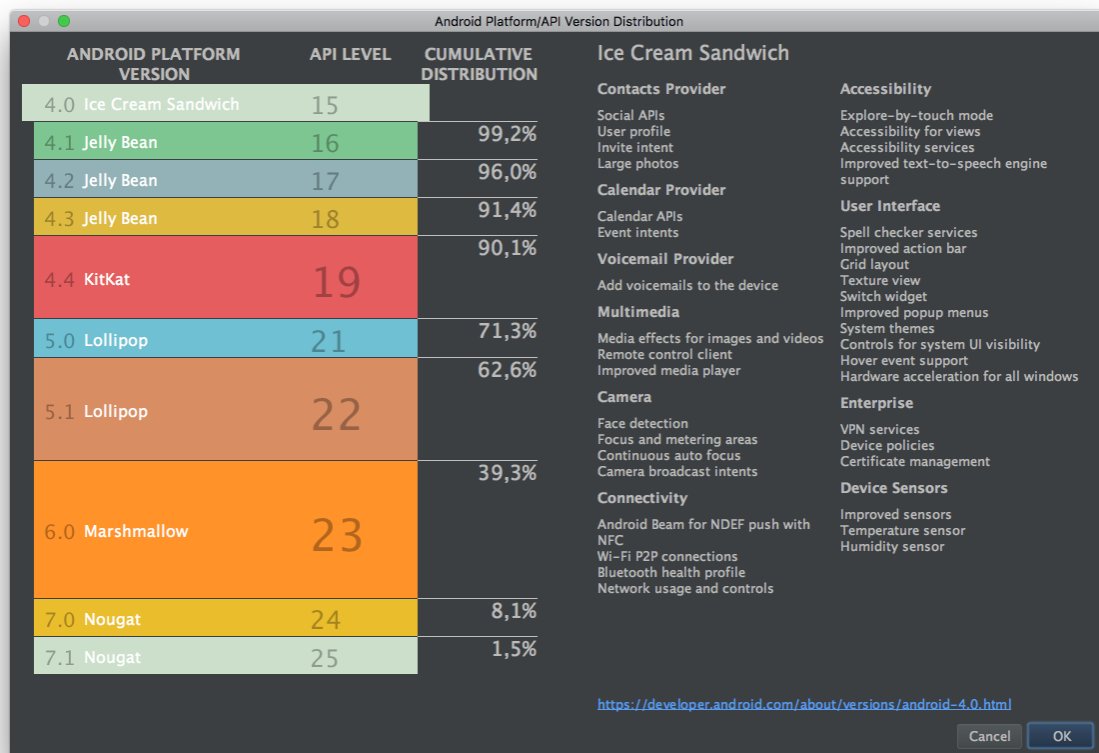
Dans cet écran, vous avez la possibilité de définir la ou les cibles sur lesquelles votre application va fonctionner. Cela permet de déterminer les SDKs à installer dans votre projet.

Vous avez le choix entre :

- **Phone and Tablet** : les téléphones et tablettes. C'est l'option par défaut ;
- **Wear** : les montres et objets connectés ;
- **TV** : la télévision, en l'occurrence les équipements qui supportent [Android TV](#) ;
- **Android Auto** : la voiture.

Pour l'instant, seuls les téléphones et tablettes nous intéressent, vous allez donc laisser la première case cochée et ne pas cocher les autres.

Vous devez également préciser la version minimale de SDK à utiliser pour votre projet, dans le champ **Minimum SDK**. Sans trop rentrer dans les détails, utiliser une API élevée (donc récente) vous permet de bénéficier des dernières fonctionnalités proposées par Android. Toutefois, les anciens appareils présents sur le marché qui ne sont pas à jour ne pourront pas faire fonctionner votre application. Pour vous aider dans votre choix, cliquez sur le lien **Help me choose**. Vous verrez apparaître l'écran suivant :



Ce graphique vous permet d'un coup d'œil de vérifier le pourcentage d'appareils qui seront capables d'installer et de lancer votre application. Plus vous vous rapprochez des 100%, mieux c'est, mais au détriment de fonctionnalités récentes. Quel dilemme !

A ce jour, en choisissant l'API 15 : *Android 4.0.3 (IceCreamSandwich)*, Android Studio précise que l'application sera en mesure de fonctionner sur 100% des appareils. Parfait ! Cliquez sur **Next**. Une nouvelle fenêtre s'affiche, vous indiquant que les composants nécessaires à votre projet sont installés. Cliquez de nouveau sur **Next**.

## Choix de l'activité principale

Une activité (ou *Activity* en anglais) est une brique fondamentale dans l'interaction avec l'utilisateur. C'est elle qui va contenir l'ensemble des éléments graphiques du type champ texte, bouton, titre, etc.

Pour faciliter la tâche du développeur, l'assistant de création permet de créer automatiquement des activités pré-définies. Par exemple, une activité pour afficher une carte Google Map ou une activité pour un affichage en mode paysage sur tablette.

Dans notre cas, nous allons choisir une activité "vide", en sélectionnant **Empty Activity**. Eh oui, nous allons tout développer nous-mêmes, c'est le meilleur moyen d'apprendre ! Cliquez sur **Next**.

## Configuration de l'activité principale

Par défaut, le nom de l'activité proposée est *MainActivity*. Il est parfaitement trouvé, car ce sera notre activité principale. Nul besoin de le modifier.

La case **Generate Layout File** permet de déterminer si un fichier de "mise en page" doit être généré pour l'activité. Ce fichier de "mise en page", couramment appelé *fichier layout*, permet de déterminer quels sont les éléments graphiques à afficher. Nous verrons plus en détail ses caractéristiques dans le prochain chapitre. Pour le moment, laissez la case cochée.

Le champ **Layout Name** permet de nommer le fichier layout. Par convention, s'il est lié à une activité, son nom commence toujours par **activity**, suivi du nom de l'activité. Dans notre cas, le nom proposé par défaut nous convient parfaitement.

Laissez la case **Backwards Compatibility (AppCompat)** cochée. Nous verrons un peu plus tard de quoi il s'agit.

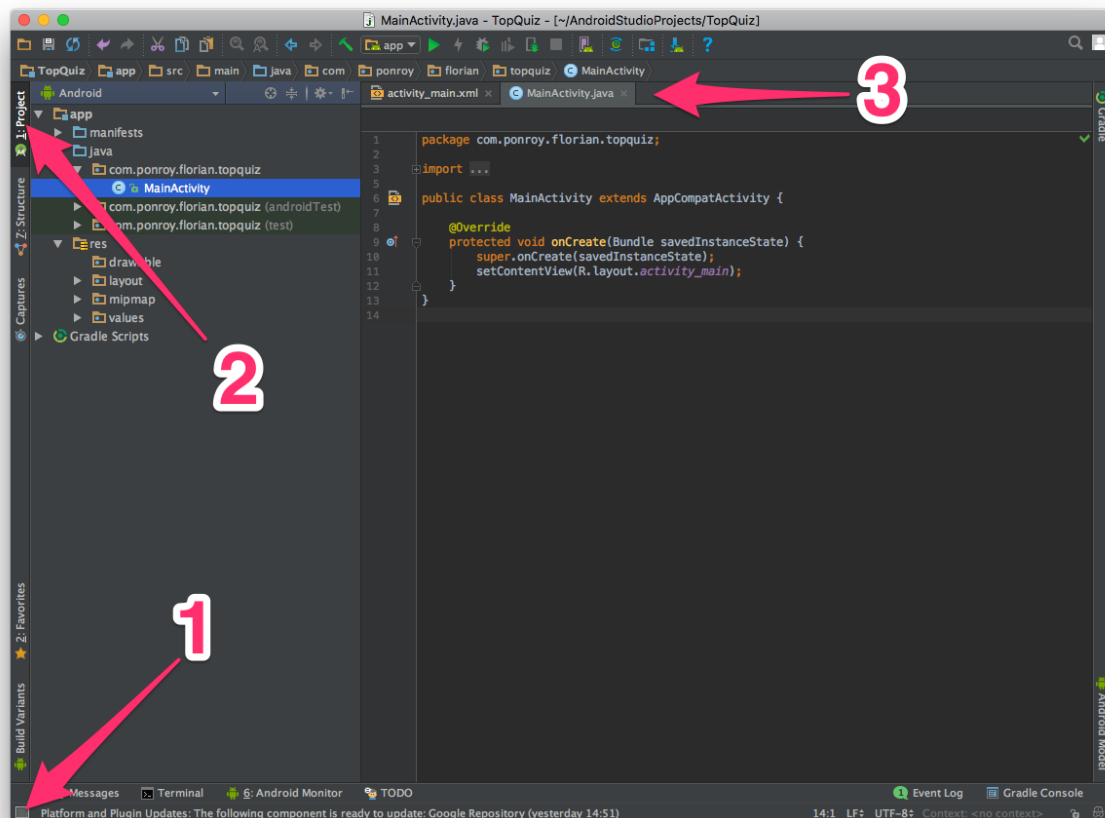
Cliquez sur le bouton **Finish**. Une fenêtre intitulée **Building 'TopQuiz' Gradle project** s'affiche, avec une barre de chargement. Patientez encore un peu, le temps que les fichiers nécessaires au bon fonctionnement de votre projet soient téléchargés, puis vous verrez apparaître l'écran principal d'Android Studio !

## Découverte d'Android Studio

---

Lorsqu'un IDE s'affiche pour la première fois, on a toujours la même sensation : on a l'impression de se trouver dans un cockpit d'avion, avec des centaines de boutons et des lumières qui clignotent dans tous les sens. Premier réflexe : faire demi-tour et partir en courant, de peur de casser quelque chose. N'ayez crainte, nous allons y aller pas à pas.

### L'écran principal



D'une façon tout à fait classique, vous avez sur le côté gauche l'arborescence des fichiers, et sur la droite le contenu du fichier en cours d'édition.

Vous constaterez qu'en plus des traditionnels boutons en haut de l'écran, plusieurs boutons ornent les bords de la fenêtre principale : quatre sur le bord gauche, six sur le bord du bas et deux sur le bord de droite. Eh oui, c'est une fonctionnalité assez originale d'Android Studio !

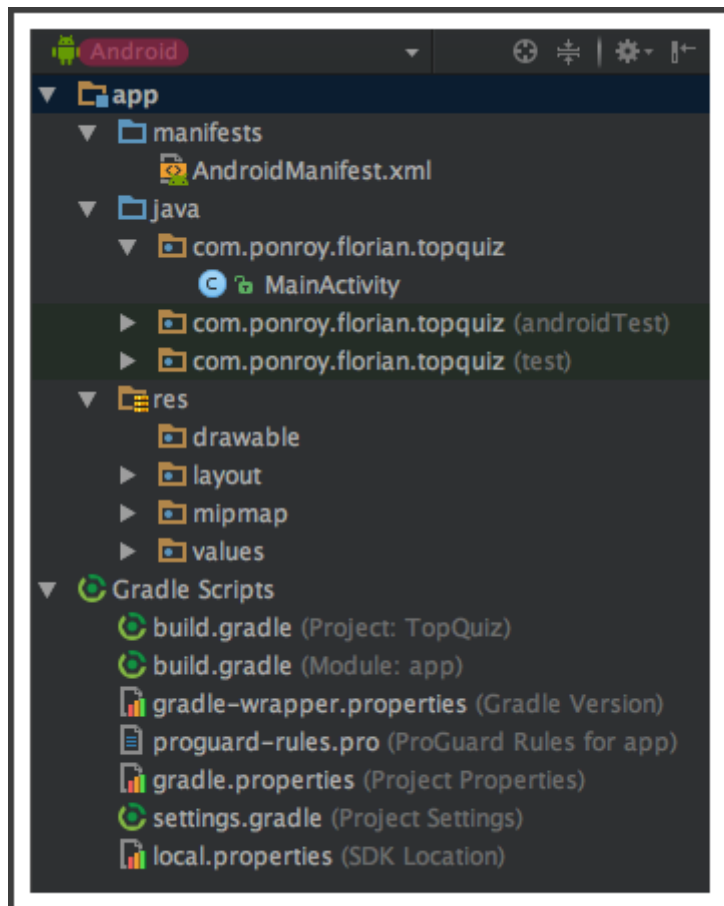
Pour l'instant, le bouton qui vous intéresse est le bouton n°2 (voir capture d'écran ci-dessus) : il permet d'afficher l'arborescence des fichiers du projet. Ensuite, chaque fichier ouvert s'affiche dans un onglet (label n°3 sur la capture d'écran).

Soyez curieux et amusez-vous à cliquer sur chaque bouton présent sur les bords pour faire apparaître le contenu. Si vous cliquez de nouveau dessus, la fenêtre sera de nouveau cachée.

Si vous ne voyez plus aucun bouton sur les bords, ne paniquez pas. Nul besoin d'appeler le 112 non plus. Il vous suffit simplement de cliquer sur le petit carré en bas à gauche de la fenêtre (bouton n°1 sur la capture d'écran ci-dessus).

## L'arborescence des fichiers

L'affichage des fichiers de votre projet doit ressembler à celui-ci :



Vous avez la possibilité d'afficher différentes "vues". Une vue correspond à un affichage pré-déterminé d'informations spécifiques. Par exemple, dans l'exemple ci-dessus, c'est la vue **Android** (surlignée en rose sur la capture d'écran) qui est sélectionnée. Elle permet d'afficher l'essentiels des fichiers spécifiques au développement Android. Par exemple, si vous souhaitez afficher l'ensemble des paquetages de l'application, vous pouvez sélectionner la vue **Packages**. De même, amusez-vous à afficher les différentes vues pour vous familiariser avec l'IDE.

Les trois répertoires principaux d'Android sont **manifests**, **java** et **res**. Explorons leur contenu.

### Le répertoire *manifests*

Ce répertoire contient généralement un seul fichier : le fichier *AndroidManifest.xml*. Ce fichier est la carte d'identité de votre application. Il permet entre autres de préciser le nom de l'application, l'icône à utiliser, quelle activité lancer au démarrage, etc.

### Le répertoire *java*

Ce répertoire contient l'ensemble du code source Java ou Kotlin de l'application, ainsi que les différents tests associés. Dans notre exemple, nous voyons apparaître le fichier *MainActivity* (l'extension *.java* est automatiquement masquée par l'IDE). Au fur et à mesure de l'avancement du projet, ce répertoire se remplira de fichier, voire de sous-dossiers afin d'isoler les composants fonctionnels entre eux.

*Hey professeur, je comprends pas là. C'est quoi Kotlin ?*

Attention, accrochez-vous : Java n'est plus le seul langage de développement disponible pour développer des applications Android. Google a récemment annoncé que le langage Kotlin était désormais officiellement supporté. Si vous êtes curieux, [n'hésitez pas à aller voir par ici](#).

## Le répertoire *res*

Ce répertoire contient toutes les ressources de l'application, et comprend quatre sous-répertoires :

- Le dossier *drawable*, qui contient l'ensemble des images et contenus à afficher à l'écran (par exemple une image de bouton ou un logo) ;
- Le dossier *layout*, qui contient l'ensemble des fichiers layout de votre application ;
- Le dossier *mipmap*, qui contient principalement l'icône de l'application ;
- Le dossier *values*, qui contient différents paramétrages et valeurs, par exemple les couleurs à utiliser dans l'application, les différentes traductions à utiliser ou les styles graphiques à appliquer.

Voilà, vous en savez assez pour commencer. Ah si, dernière petite astuce :

Pour fermer rapidement un fichier, au lieu de cliquer sur la petite croix de l'onglet correspondant, maintenez la touche *shift* de votre clavier enfoncée et cliquez sur l'onglet ! Merci qui ?

C'est parti !

## Conclusion

---

Android Studio vous guide naturellement dans la création d'un projet. Le choix de l'API est souvent crucial : dans votre futur emploi, vous serez peut-être contraints d'utiliser une certaine version de l'API afin de garantir la compatibilité avec les librairies existantes.

## Dessinez l'interface utilisateur de votre première activité

---

### Introduction

---

Dans ce chapitre, nous allons dessiner l'interface utilisateur du premier écran de l'application. En d'autres termes, cela consistera à déterminer de quels éléments graphiques nous aurons besoin et comment nous les positionnerons à l'écran.

Dans ce premier écran, nous souhaitons accueillir l'utilisateur en lui demandant de saisir son prénom. De fait, cet écran va être composé d'un champ texte, d'une zone de saisie et d'un bouton. Le résultat attendu est le suivant :

### Activité et layout

---

Une **activité**, ou **Activity** en anglais, est une brique fondamentale d'Android. C'est le point d'entrée de n'importe quelle application Android.

Une activité a pour rôle principal d'interagir avec l'utilisateur. C'est une classe Java ou Kotlin, qui hérite obligatoirement de la classe Android **Activity** ou **AppCompatActivity**.

*Hey professeur, quelle différence entre **Activity** et **AppCompatActivity** ?*

Par définition, une activité Android hérite toujours (plus ou moins directement) de la classe **Activity**. Sur les versions d'Android un peu plus anciennes, certaines fonctionnalités récentes ne sont pas officiellement supportées. De ce fait, hériter d'**AppCompatActivity** permet de corriger ce problème en "émulant" ces nouvelles fonctionnalités.

Si vous souhaitez avoir deux écrans dans votre application, par exemple un écran de connexion et un écran de tutoriel, vous aurez *généralement* deux activités : la première qui gère la partie connexion et la seconde qui gère l'affichage du tutoriel. Par convention, le nom d'une activité est toujours suffixé par *Activity* et écrit en **CamelCase**. Ainsi, vous nommerez vos activités *LoginActivity* et *TutorialActivity*.

Pourquoi ai-je dit "vous aurez *généralement* deux activités" ? Car nous verrons plus tard, dans un autre cours, qu'il n'est pas forcément obligatoire d'avoir à chaque fois une activité distincte pour gérer chaque écran. Mais il en est encore trop tôt pour en parler :-)

Pour interagir avec l'utilisateur, il faut lui présenter des éléments graphiques et **des éléments de contrôle ou widgets** pour qu'il puisse s'amuser avec ses petits doigts. Ces widgets peuvent être des boutons, des zones de saisie ou des menus déroulants par exemple.

Afin de déterminer quels éléments graphiques utiliser et comment les positionner à l'écran, nous utilisons **un fichier layout**. Un fichier layout est un fichier XML que l'activité va charger après avoir été instanciée. Ce fichier XML est toujours stocké dans le répertoire **res/layout** de votre projet. Par convention, s'il est lié à une activité, il est toujours préfixé par *activity*, suivi du nom de l'activité, le tout en minuscule et séparé par un underscore (\_). Ainsi, le fichier layout associé à *MainActivity* est *activity\_main.xml*. De la même façon, si nous avons eu une activité nommée *LoginActivity*, nous aurions créé le fichier layout associé *activity\_login.xml*.



## Construction de l'interface

---

### L'éditeur graphique

Sous Android Studio, naviguez dans l'arborescence du projet et ouvrez le fichier *activity\_main.xml* situé dans **res/layout** en double-cliquant dessus.

Par défaut, Android Studio ouvre l'éditeur en mode **Design**, c'est à dire que vous pouvez placer et configurer les différents éléments graphiques avec votre souris et générer automatiquement le contenu XML. Cela semble séduisant, mais nous allons plutôt utiliser le mode **Text**, permettant d'avoir une meilleure maîtrise de l'ensemble. Pour ce faire, cliquer sur l'onglet **Text** en bas de l'écran, et vous verrez apparaître le contenu XML.

Le contenu par défaut peut varier suivant les versions d'Android Studio, mais vous devez avoir quelque chose qui ressemble peu ou prou à cela :

```
<?xml version="1.0" encoding="utf-8"?>

<android.support.constraint.ConstraintLayout

    xmlns:android="http://schemas.android.com/apk/res/android"

    xmlns:app="http://schemas.android.com/apk/res-auto"

    xmlns:tools="http://schemas.android.com/tools"

    android:layout_width="match_parent"

    android:layout_height="match_parent"

    tools:context="com.ponroy.florian.topquiz.MainActivity">

    <TextView

        android:layout_width="wrap_content"

        android:layout_height="wrap_content"

        android:text="Hello World!"

        app:layout_constraintBottom_toBottomOf="parent"

        app:layout_constraintLeft_toLeftOf="parent"

        app:layout_constraintRight_toRightOf="parent"

        app:layout_constraintTop_toTopOf="parent"></TextView>
```

```
</android.support.constraint.ConstraintLayout>
```

Cela n'est pas forcément très parlant de prime abord. Pour voir le résultat en temps-réel, cliquez sur le bouton **Preview** situé sur le bord droit de la fenêtre. Vous verrez automatiquement apparaître le rendu du contenu XML, c'est à dire un écran avec le texte *Hello World!*.

Cliquez ici pour afficher la fenêtre de pré-visualisation du layout

## Les conteneurs

Afin de pouvoir afficher des éléments à l'écran, il est impératif d'utiliser un *conteneur*. Un conteneur est un élément particulier permettant d'organiser les éléments qu'il contient entre eux.

Pour mieux comprendre le principe de conteneur, transposons cela dans le monde réel : imaginez que vous souhaitiez accrocher au mur vos différentes photos de vacances. Vous allez très probablement acheter un grand cadre dans lequel vous allez coller ou punaiser toutes vos photos.

Dans le monde virtuel d'Android, vos photos correspondent aux éléments que vous souhaitez montrer à l'utilisateur (un champ texte ou une zone de saisie), et le grand cadre correspond à un conteneur.

Dans le fichier XML de notre projet, le premier élément XML que nous voyons est du type **android.support.constraint.ConstraintLayout**. Cet élément est un conteneur. Android suffixe toujours le nom des conteneurs par **Layout**. Dans cet exemple, le conteneur contient un élément **TextView**, utilisé pour afficher le texte *Hello World!*.

Parmi les conteneurs proposés par Android, nous pouvons noter par exemple :

- **FrameLayout** : permet de positionner les éléments les uns au dessus des autres ;
- **LinearLayout** : permet de positionner les éléments les uns à la suite des autres, dans le sens horizontal ou vertical ;
- **RelativeLayout** : permet de positionner les éléments les uns par rapport aux autres ;
- **ConstraintLayout** : comme le **RelativeLayout**, mais avec des règles de positionnement beaucoup plus puissantes.

Il est trop tôt pour utiliser le **ConstraintLayout**, nous allons donc le remplacer par **LinearLayout**, beaucoup plus simple. Pour ce faire, remplacez la balise

XML **android.support.constraint.ConstraintLayout** par **LinearLayout**, puis supprimez tout ce qui se rattache au tag *app*. Vous devez obtenir le résultat suivant :

```
<?xml version="1.0" encoding="utf-8"?>

<LinearLayout

    xmlns:android="http://schemas.android.com/apk/res/android"

    xmlns:tools="http://schemas.android.com/tools"

    android:layout_width="match_parent"

    android:layout_height="match_parent"
```

```
tools:context="com.ponroy.florian.topquiz.MainActivity">

<TextView

    android:layout_width="wrap_content"

    android:layout_height="wrap_content"

    android:text="Hello World!"></TextView>

</LinearLayout>
```

De façon littérale, nous venons d'écrire :

- *Je souhaite afficher mes éléments graphiques les uns à la suite des autres. Le premier élément graphique que je souhaite afficher est un champ texte.*

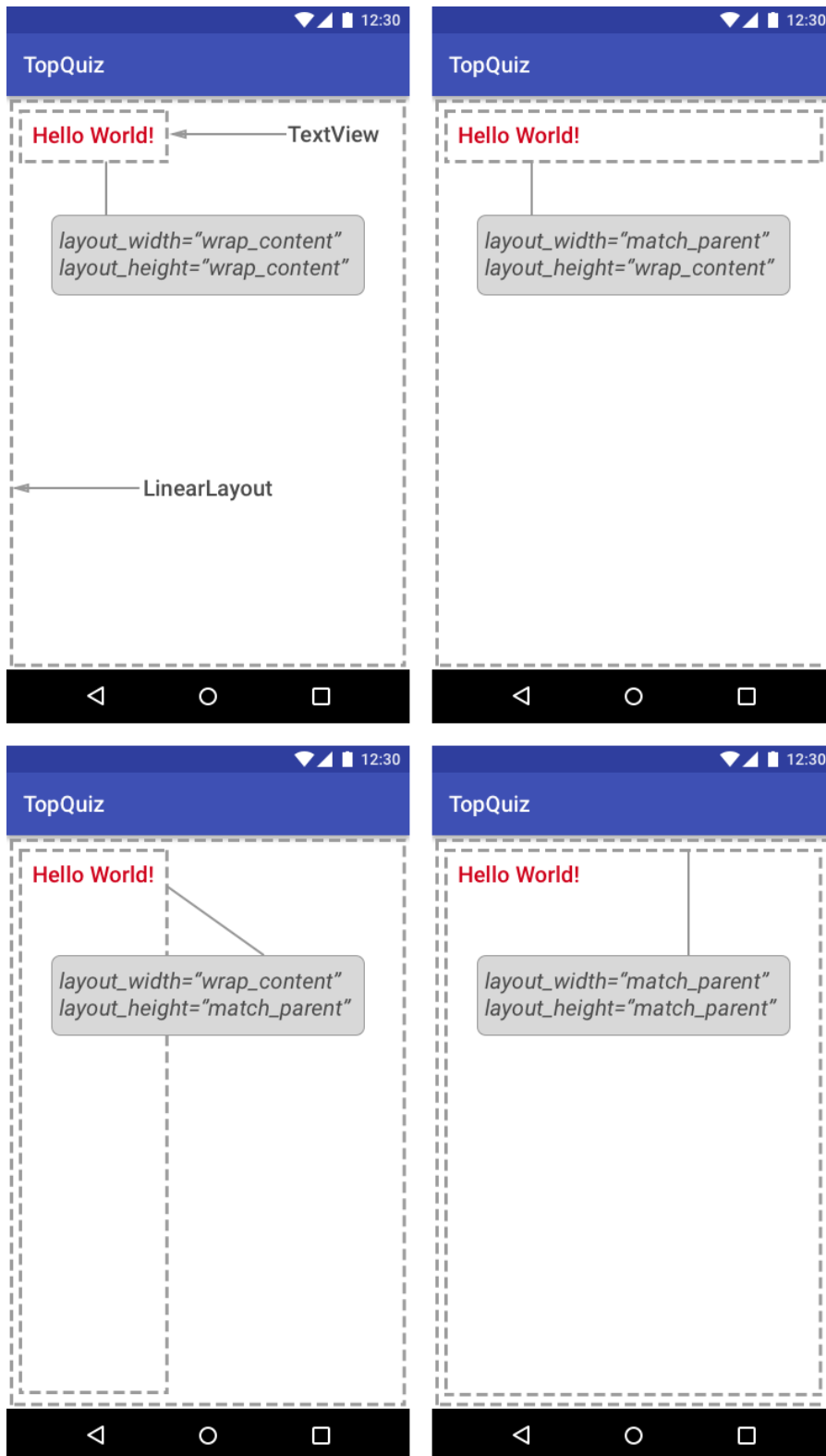
## Les attributs

### Occupation de l'espace

Chaque balise XML possède plusieurs attributs. A minima, les deux attributs fondamentaux sont **layout\_width** et **layout\_height**. Ils permettent de déterminer comment afficher un élément au sein de son conteneur. Les deux valeurs possibles sont :

- *match\_parent* : l'élément doit s'étendre le plus possible afin d'occuper le maximum d'espace disponible offert par son parent (vous pourriez voir apparaître de temps en temps *fill\_parent* au détour d'un tutorial ou d'un site web : c'est un attribut obsolète, ancêtre de *match\_parent*) ;
- *wrap\_content* : l'élément doit s'étendre le moins d'espace possible et n'occuper que la place nécessaire à l'affichage de son contenu.

Un dessin valant mieux qu'un long discours, voici la forme que prendrait l'élément `TextView` suivant les valeurs de ces deux attributs :



:

Variation des attributs d'occupation de l'espace

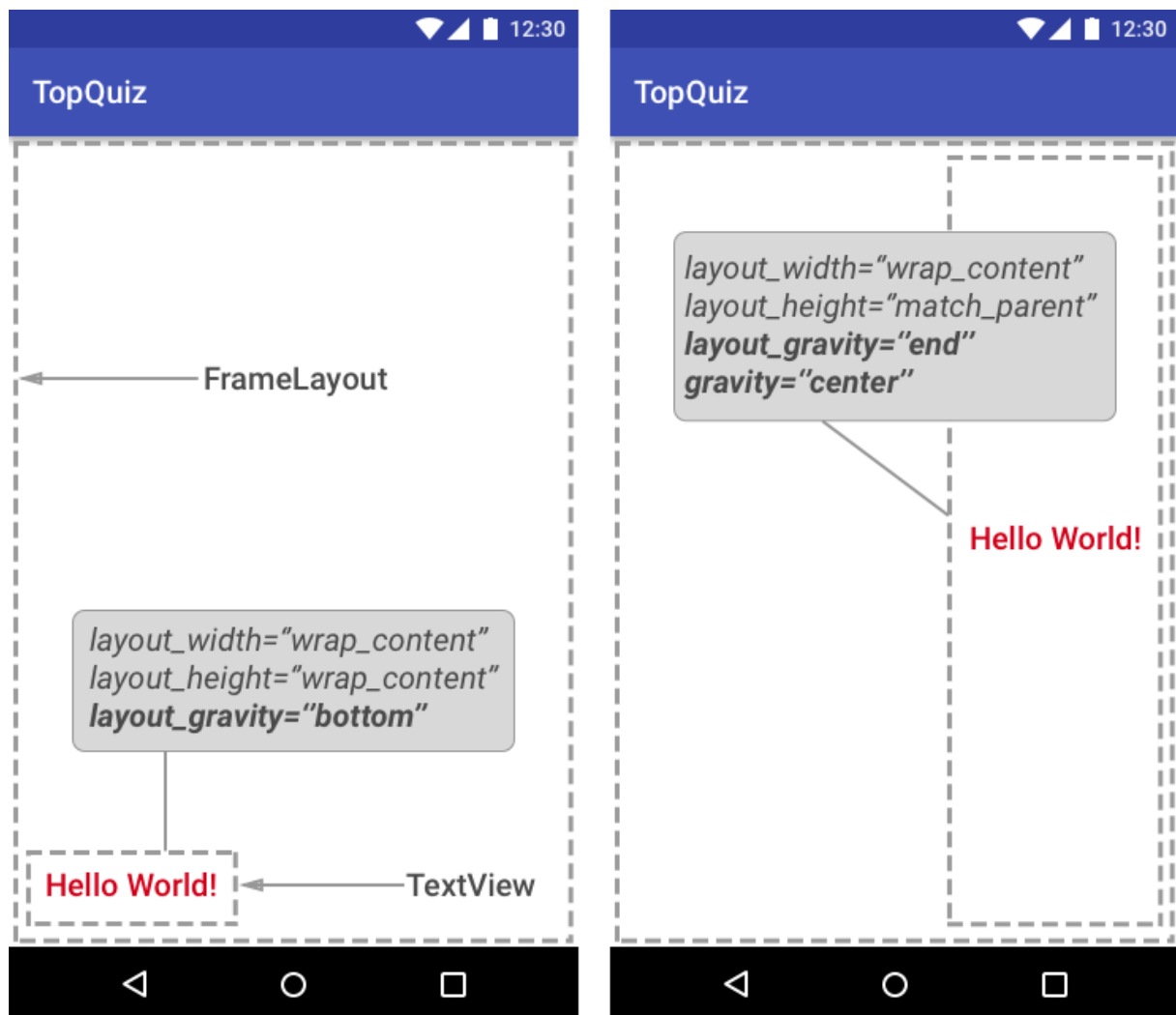
Gravitation

La notion de gravitation peut s'appliquer à un élément ou à son contenu. Elle permet de déterminer comment positionner un élément par rapport à son conteneur, ou comment positionner le contenu d'un élément, par exemple le titre d'un bouton.

Pour définir le positionnement d'un élément, c'est l'attribut **android:layout\_gravity** qu'il faut utiliser. Les valeurs possibles sont nombreuses : *left, right, center, center\_vertical, center\_horizontal, etc.*

Pour définir le positionnement d'un titre au sein d'un bouton ou d'un champ texte par exemple, c'est l'attribut **android:gravity** qu'il faut utiliser. Les valeurs possibles sont identiques à l'attribut **layout\_gravity**.

Pour bien saisir la différence, regardez le résultat produit selon les valeurs utilisées :



### Gravitation des éléments

#### Texte

Nous allons remplacer le texte de l'élément `TextView` afin d'accueillir convenablement l'utilisateur. Il vous suffit pour cela de remplacer la valeur de l'élément **android:text**, par exemple : *"Bienvenue dans TopQuiz. Quel est votre prénom ?"*. Le texte dans le rendu de la zone *Preview* doit automatiquement se mettre à jour.

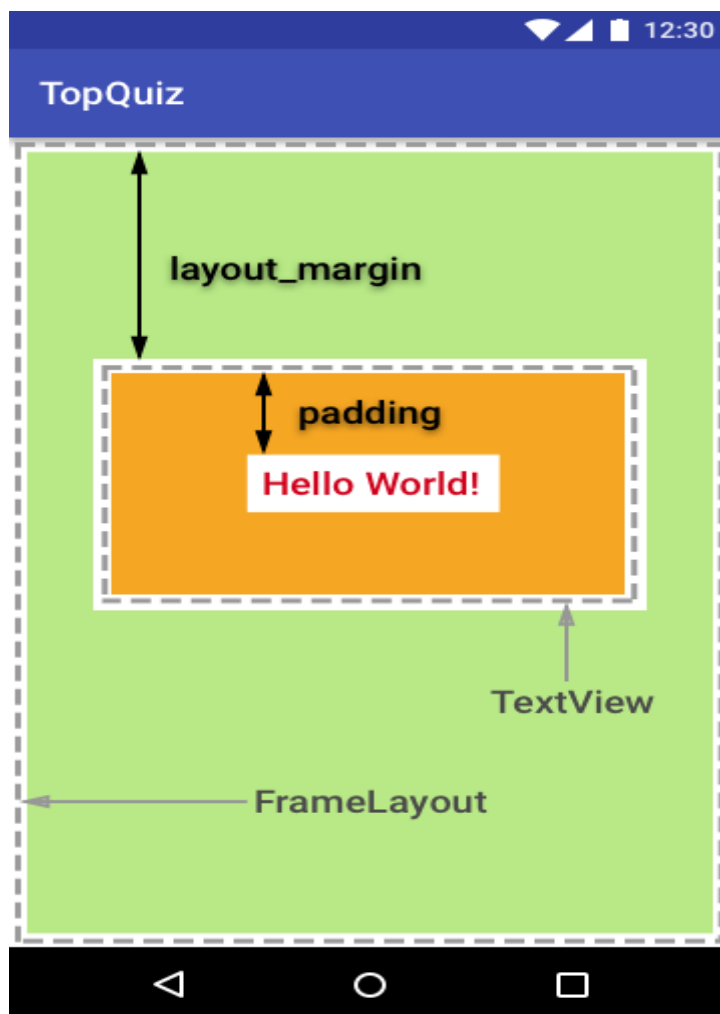
## Marge

Vous constatez que le texte de l'élément TextView est collé en haut à gauche de son conteneur, ce qui n'est pas l'idéal. Pour le décoller légèrement du bord, il est nécessaire d'ajouter l'attribut **android:layout\_margin**, permettant de préciser une grandeur de marge (Simpson). La valeur se mesure en *dp*, pour *Density-independent pixels*. Par exemple, en précisant une valeur de *10dp*, vous verrez l'élément s'éloigner des bords. Si vous ne souhaitez modifier qu'une seule marge, vous pouvez utiliser les versions suffixées suivantes : **layout\_marginTop**, **layout\_marginBottom**, **layout\_marginStart** ou **layout\_marginEnd**.

Utiliser l'unité de mesure *dp* permet de s'affranchir des incohérences d'affichage suivant la résolution des téléphones. Dit autrement, imaginez que vous définissiez un élément qui fasse 20 pixels de haut. Sur un téléphone dont l'écran fait 100 pixels de haut (si si, je vous assure, cela existait dans les années 80), l'élément occuperait 20% de l'espace, alors que son affichage n'occuperait que 2% d'espace sur un téléphone récent. L'élément serait donc illisible. [Je vous invite à lire cette page très bien expliquée pour mieux comprendre cette notion.](#)

## Rembourrage

Le rembourrage, ou *padding* en anglais, consiste à ajouter de l'espace entre le contenu d'un élément et les bords de cet élément. J'adore les images, je suis sûr que vous allez tout de suite comprendre :



Différence entre marge et rembourrage

En ajoutant une marge et du rembourrage à l'élément **TextView** de votre projet, vous devez obtenir le résultat suivant (seul l'élément TextView est présenté) :

```
<TextView

    android:layout_width="wrap_content"

    android:layout_height="wrap_content"

    android:layout_margin="10dp"

    android:padding="20dp"

    android:text="Welcome! What's your name?"/>
```

Voilà, vous en savez suffisamment sur la mise en page des éléments, nous allons maintenant étudier les éléments de contrôle pour interagir avec l'utilisateur.

## Les éléments de contrôle

### Zone de saisie

Pour que l'utilisateur puisse taper son prénom, il faut lui présenter un élément lui permettant de saisir du texte. Sous Android, c'est l'élément **EditText** qui porte ce rôle. Pour ce faire, toujours dans le fichier *activity\_main.xml*, ajoutez un élément **EditText**.

Dès que vous commencez à saisir le chevron et les premières lettres, Android Studio vous propose automatiquement tous les choix possibles. Dès que vous voyez apparaître EditText, sélectionnez-le et appuyez sur la touche de tabulation. Android Studio vous ajoute ensuite automatiquement les attributs *layout\_width* et *layout\_height*. Il vous positionne également le curseur sur la première valeur. Commencez à saisir les premières lettres et complétez automatiquement en appuyant sur tabulation. Appuyez de nouveau sur tabulation pour passer à l'attribut suivant, et ainsi de suite. C'est magique !

Toutefois, le champ de saisie est comprimé entre le texte et le bord droit de l'écran. Il aurait été plus judicieux qu'il soit positionné en dessous de l'élément TextView, n'est-ce pas ? Si vous vous souvenez, nous avons dit que le conteneur LinearLayout permettait de positionner les éléments les uns à la suite des autres. Par défaut, ce positionnement s'effectue horizontalement. Pour changer ce comportement, il vous suffit d'ajouter l'attribut **orientation** à l'élément LinearLayout, avec la valeur *"vertical"* (de la même façon, vous pouvez simplement saisir les premières lettres et Android Studio se chargera de compléter l'ensemble pour vous).

Tout comme le champ texte, nous pouvons ajouter une marge afin d'éviter que la zone de saisie ne soit trop proche des bords, par exemple à gauche et à droite.

Il est également possible d'ajouter un indice à l'aide de l'attribut **android:hint**. Cet indice apparaît dans le champ texte pour apporter une information à l'utilisateur, puis disparaît dès qu'il commence à saisir du texte.

Vous devez obtenir le résultat suivant (seul l'élément EditText est présenté) :

```
<EditText

    android:layout_width="match_parent"

    android:layout_height="wrap_content"

    android:layout_marginStart="10dp"

    android:layout_marginEnd="10dp"

    android:hint="Please type your name"/>
```

## Bouton

Maintenant que l'utilisateur s'est présenté, il n'a plus qu'à appuyer sur un bouton pour commencer à jouer. L'élément XML à utiliser est **Button** (comme Benjamin... désolé, je n'ai pas pu m'en empêcher).

Comme pour l'élément TextView, l'attribut à utiliser pour spécifier le titre est **android:text**. Nous décidons de nommer le bouton *"Let's play"*. Vous pouvez modifier la marge ou le rembourrage pour positionner le bouton comme bon vous semble. Pour ma part, j'ai décidé de le centrer horizontalement, et d'ajouter un peu de rembourrage pour qu'il paraisse plus important. Voici le résultat :

```
<Button

    android:layout_width="wrap_content"

    android:layout_height="wrap_content"

    android:layout_marginTop="20dp"

    android:padding="30dp"

    android:layout_gravity="center_horizontal"

    android:text="Let's play"/>
```

## Conclusion

---

Voilà, l'interface graphique est prête. Ce n'est pas la plus jolie, certes, mais elle est fonctionnelle !

L'outil de pré-visualisation de layout d'Android Studio est très puissant : il vous permet d'un coup d'œil de visualiser le résultat final. N'hésitez pas à modifier les attributs des éléments de votre layout afin de bien comprendre leur fonctionnement.

Rendez-vous dans le prochain chapitre pour commencer à coder !



## Référez les éléments graphiques dans votre activité

---

L'interface graphique est prête, il est temps maintenant de se pencher sur le code Java de l'activité. Ouvrez le fichier **MainActivity** situé dans le répertoire **java/nom.de.votre.paquetage**.

### Affichage de l'interface

---

La classe *MainActivity* a été créée automatiquement par Android Studio lors de la création du projet, lorsque nous avons choisi le modèle **Empty Activity**. La seule méthode implémentée est la suivante :



```
@Override
protected void onCreate(Bundle savedInstanceState) {

    super.onCreate(savedInstanceState);

    setContentView(R.layout.activity_main);
}
```

La méthode **onCreate()** est appelée lorsque l'activité est créée (nous verrons plus en détail le cycle de vie d'une activité à la fin de ce cours). La ligne qui nous intéresse ici est la ligne n°4 : la méthode **setContentView()** permet de déterminer quel fichier layout utiliser.

Le fichier layout à utiliser est précisé avec une syntaxe particulière : *R.layout.nom\_du\_fichier* (sans l'extension XML). Petite explication : lors de la compilation du projet, Android Studio génère une classe Java appelée **R** (pour **Resources**), qui contient l'ensemble des identifiants de toutes les ressources du projet. Ces ressources peuvent être des fichiers layout, des chaînes de caractères, etc. Nous verrons à quoi ressemble cette classe après avoir compilé le projet.

Petite astuce : vous pouvez naviguer facilement dans le code source en utilisant la souris et en cliquant sur une méthode, une classe ou un paramètre. Pour ce faire, positionnez le curseur sur le mot-clé qui vous intéresse, maintenez la touche  enfoncée sur Mac ou la touche  sous Windows et faites un clic gauche. Essayez dans le fichier *MainActivity.java* en cliquant sur **AppCompatActivity**, **setContentView** ou **activity\_main** !

## Référencement des éléments graphiques

---

### Déclaration des variables

Pour rappel, nous avons trois éléments graphiques dans notre interface :

- Le texte d'accueil ;
- Le champ de saisie du prénom ;
- Le bouton de validation.

Afin de pouvoir interagir avec ces éléments, il est nécessaire de les référencer dans le code. Commençons par ajouter dans la classe *MainActivity* les trois variables correspondantes :

```
private TextView mGreetingText;

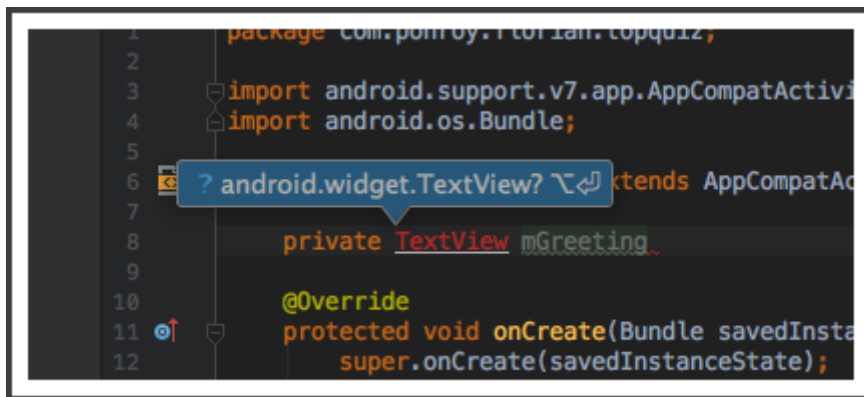
private EditText mNameInput;

private Button mPlayButton;
```

Lorsque vous commencerez à saisir un type (par exemple **TextView**), vous constaterez qu'il apparaîtra en rouge. Pourquoi ? Parce que ce type est inconnu : il faut importer la classe où ce type est défini. Par exemple, pour le type **TextView**, il faut importer la classe **TextView** située dans le paquetage **android.widget** en saisissant la ligne suivante en haut du fichier :

```
import android.widget.TextView;
```

Heureusement, Android Studio est là pour vous prêter main forte : dès qu'il détecte la saisie d'un type inconnu (il faut toujours se méfier des types inconnus, mes parents me l'ont toujours dit), il vous propose d'importer automatiquement la classe correspondante en appuyant sur **ALT + Entrée**. Ainsi, le type devient connu et vos parents sont rassurés.



Import automatique

Par convention, il est fréquent en Java, et notamment sur Android, de préfixer les attributs avec la lettre *m* (pour *member* en anglais). Cela permet d'un coup d'œil, en lisant le code, de savoir qu'une variable correspond à une donnée membre. De même, les variables statiques sont préfixées par la lettre *s*. [Vous pourrez trouver davantage d'informations ici.](#)

## Identification des éléments graphiques

Il manque néanmoins une petite chose dans le fichier layout : imaginez que vous ayez cinq éléments **TextView** les uns à la suite des autres. Comment les distinguer ? En leur ajoutant un identifiant bien sûr ! Pour cela, l'attribut à utiliser est **android:id**, et sa valeur doit être **"@+id/votre\_identifiant"**. Notez bien que l'identifiant doit toujours être préfixé par **"@+id/"**. Par exemple :

```
<TextView

    android:id="@+id/activity_main_greeting_txt"

    android:layout_width="wrap_content"

    android:layout_height="wrap_content"
```

```

        android:padding="20dp"

        android:text="Welcome! What's your name?"/>

<EditText

    android:id="@+id/activity_main_name_input"

    android:layout_width="match_parent"

    android:layout_height="wrap_content"

    android:layout_marginStart="20dp"

    android:layout_marginEnd="20dp"

    android:hint="Please type your name"/>

<Button

    android:id="@+id/activity_main_play_btn"

    android:layout_width="wrap_content"

    android:layout_height="wrap_content"

    android:layout_marginTop="20dp"

    android:padding="30dp"

    android:layout_gravity="center_horizontal"

    android:text="Let's play"/>

```

Vous êtes libres d'utiliser le nom d'identifiant que vous souhaitez, mais attention : un identifiant doit être unique **au niveau du projet**, ce qui signifie que vous ne pouvez pas utiliser le même identifiant dans deux fichiers séparés. Pour pallier cette limitation, je vous propose d'utiliser la convention de nommage suivante : préfixez par le nom du fichier, puis ajoutez la description, puis suffixez par le type d'élément (*input* pour une zone de saisie, *btn* pour un bouton, *text* pour un champ texte, etc).

## Branchement des variables

Voilà, nous pouvons maintenant référencer les trois éléments graphiques qui nous intéressent dans le code. La méthode à utiliser pour cela est **findViewById()**. Elle prend en paramètre l'identifiant de la vue qui nous intéresse, et renvoie la vue correspondante. Comme pour le fichier

layout, la syntaxe à utiliser pour le paramètre est spécifique : il faut préciser *R.id.identifiant\_de\_vue*.

Notez également que le type renvoyé par la méthode **findViewById()** est **View**. Il faut donc effectuer une conversion vers le bon type (ou *cast* en anglais).

Voici l'exemple de code que vous pouvez ajouter dans la méthode **onCreate()** :

```
mGreetingText = (TextView) findViewById(R.id.activity_main_greeting_txt);  
  
mNameInput = (EditText) findViewById(R.id.activity_main_name_input);  
  
mPlayButton = (Button) findViewById(R.id.activity_main_play_btn);
```

Nos trois éléments graphiques sont référencés, nous allons pouvoir commencer à jouer avec !

## Conclusion

---

Dans ce chapitre, nous avons vu comment référencer les éléments graphiques dans l'activité. Référencer les éléments graphiques dans le code est une étape indispensable pour pouvoir implémenter la logique métier de notre application, pouvoir modifier leur propriétés ou réagir à leurs changements.

## Gérez les actions de l'utilisateur

---

Dans ce chapitre, nous allons voir comment interagir avec l'utilisateur et répondre à ses différentes actions.

### Gestion des actions

---

Les deux actions les plus importantes à implémenter sont les suivantes :

1. Vérifier la saisie du nom de l'utilisateur ;
2. Détecter le clic sur le bouton pour démarrer le jeu.

#### Saisie utilisateur

Il faut s'assurer que l'utilisateur ait saisi son prénom avant d'aller plus loin. Pour l'empêcher d'aller plus loin, la façon la plus simple consiste à désactiver le bouton de démarrage de jeu au lancement de l'application, puis de l'activer lorsqu'il a saisi son prénom. Tout d'abord, ajoutez la ligne suivante dans la méthode **onCreate()** :

```
mPlayButton.setEnabled(false);
```

Elle permet de désactiver le bouton de l'interface. Assurez-vous d'ajouter cette ligne après le branchement de la variable **mPlayButton**, sinon vous appellerez la méthode **setEnabled()** sur un objet non défini, ce qui va naturellement faire planter l'application.

Ensuite, il faut pouvoir être notifié lorsque l'utilisateur commence à saisir du texte dans le champ **EditText** correspondant. Pour ce faire, nous allons appeler la méthode **addTextChangedListener()** sur l'instance d'EditText, en utilisant une implémentation d'interface sous forme de classe anonyme.

Voici le code que vous allez ajouter dans **onCreate()**, à la suite du code existant :

```
mNameInput.addTextChangedListener(new TextWatcher() {

    @Override

    public void beforeTextChanged(CharSequence s, int start, int count, int after) {

    }

    @Override

    public void onTextChanged(CharSequence s, int start, int before, int count) {

        // This is where we'll check the user input
```

```

    }

    @Override

    public void afterTextChanged(Editable s) {

    }

});

```

Afin d'éviter de tout taper, Android Studio vous simplifie encore un peu la vie. D'une part, il va rapidement vous proposer des suggestions lorsque vous commencerez à taper **addText...** vous pourrez alors appuyez sur la touche de tabulation pour compléter automatiquement. D'autre part, lorsque vous commencerez à saisir ce qu'il y a après entre les parenthèses, arrêtez-vous après avoir tapé le **T** de **TextWatcher** : Android Studio va gentiment vous proposer d'implémenter automatiquement l'interface associée. Appuyez sur tabulation et là, miracle !



Implémentation automatique d'une interface sous forme de classe anonyme

La méthode qui va nous intéresser est celle du milieu : **onTextChanged**.

Si vous constatez que les méthodes générées par Android Studio possèdent des signatures étranges (telle que celle ci-dessous), c'est qu'il vous manque le code source d'Android.

```
beforeTextChanged(CharSequence charSequence, int i1, int i2, int i3)
```

En effet, lorsque vous demandez à Android Studio d'implémenter l'interface pour vous, il va analyser le code source de l'interface pour récupérer les méthodes à implémenter, et récupérer les noms des paramètres correspondants. Si le code source n'est pas installé, il va générer des noms sans aucun sens pour le développeur.

Pour installer le code source, rendez-vous dans les préférences d'Android (raccourci `⌘ + virgule` sous Mac ou `CTRL + ALT + S` sous Windows) puis dans la section **Appearance & Behavior > System Settings > Android SDK**, ou lancez directement le SDK Manager depuis l'IDE (l'icône avec la tête d'Android et une flèche bleue derrière elle) :



Ensuite, dans l'onglet **SDK Platforms**, activez l'option **Show Package Details** en bas à droite, cochez la case **Sources for Android 25** (ou le numéro correspondant à votre version), puis appuyez sur le bouton **Apply**.

À chaque fois que l'utilisateur saisira une lettre, la méthode **onTextChanged** sera appelée. Cela nous permettra de déterminer si l'utilisateur a commencé à saisir son prénom, et ainsi activer le bouton de démarrage de jeu. La logique à appliquer est donc simple : si au moins une lettre a été saisie, alors le bouton doit être actif (bon, je suis d'accord avec vous, un prénom d'une seule lettre, c'est assez rare. Sauf pour Q dans James Bond, mais je ne vous garantis pas qu'il utilisera l'application).

Ajoutez simplement la ligne suivante, et cela fera parfaitement l'affaire :

```
mPlayButton.setEnabled(s.toString().length() != 0);
```

## Clic sur le bouton

Une fois le bouton activé, l'utilisateur peut cliquer dessus pour lancer le jeu (nous verrons cela dans la partie suivante). Pour détecter que l'utilisateur a cliqué sur le bouton, il est nécessaire d'implémenter un *listener*. Le principe est identique à la gestion de la saisie présentée au dessus. Il faut pour cela appeler la méthode **setOnClickListener()** sur l'objet `mPlayButton`, puis d'implémenter l'interface **OnClickListener** de la méthode **View**, comme ceci :

```
mPlayButton.setOnClickListener(new View.OnClickListener() {  
  
    @Override  
  
    public void onClick(View v) {  
  
        // The user just clicked  
  
    }  
  
});
```

La méthode **onClick()** est appelée chaque fois que l'utilisateur appuie sur le bouton. C'est l'endroit idéal pour démarrer le jeu, ce que nous verrons dans les chapitres suivants.

## Conclusion

Au delà de la détection d'un clic sur un bouton ou d'une entrée dans une zone de saisie, vous verrez plus tard qu'il est possible de détecter un nombre d'événements très important : lorsque

l'utilisateur fait défiler l'écran, lorsqu'il penche son téléphone d'un côté ou d'un autre, lorsqu'il effectue un clic court ou un clic long, etc.



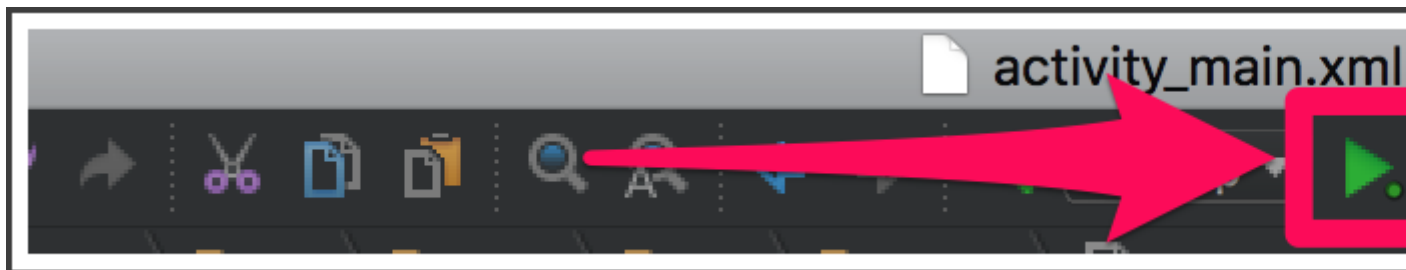
## Lancez l'application sur l'émulateur ou sur un équipement réel

Vous avez saisi du code, c'est super, mais concrètement, cela donne quoi ? Ce chapitre vous présente comment lancer votre application. Vous avez le choix : soit sur un émulateur si vous ne possédez pas d'Android, soit sur votre téléphone (ou tablette) si vous en possédez un (ou une).

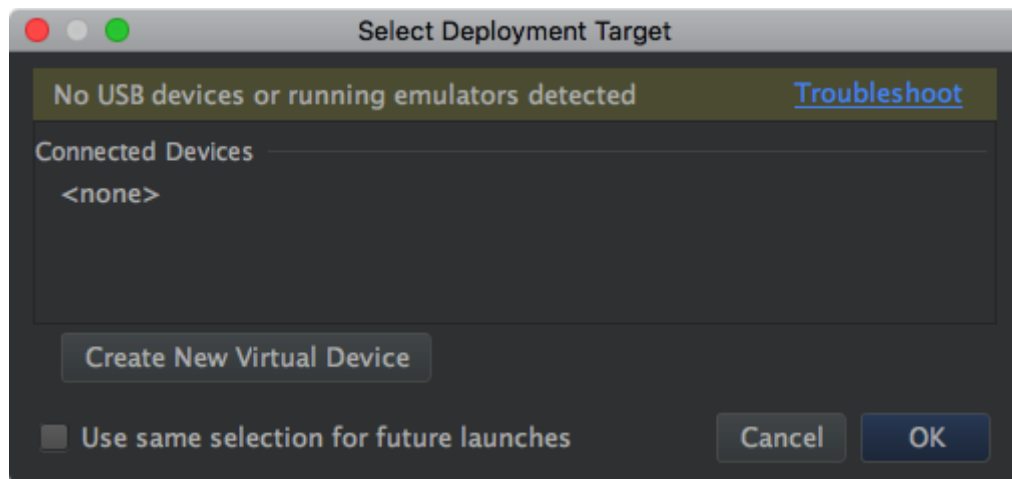
Utiliser l'émulateur offre l'avantage de générer facilement différentes configurations, avec des tailles d'écran différentes, une mémoire vive limitée ou une ancienne version d'Android par exemple. Après, rien ne vaut un test sur un équipement réel, afin de s'assurer que l'expérience utilisateur est la meilleure possible.

### Lancement sur l'émulateur

Pour lancer l'application, il suffit de cliquer sur le bouton de lecture vert situé dans la barre de l'IDE :



Ensuite, la fenêtre suivante s'affiche :



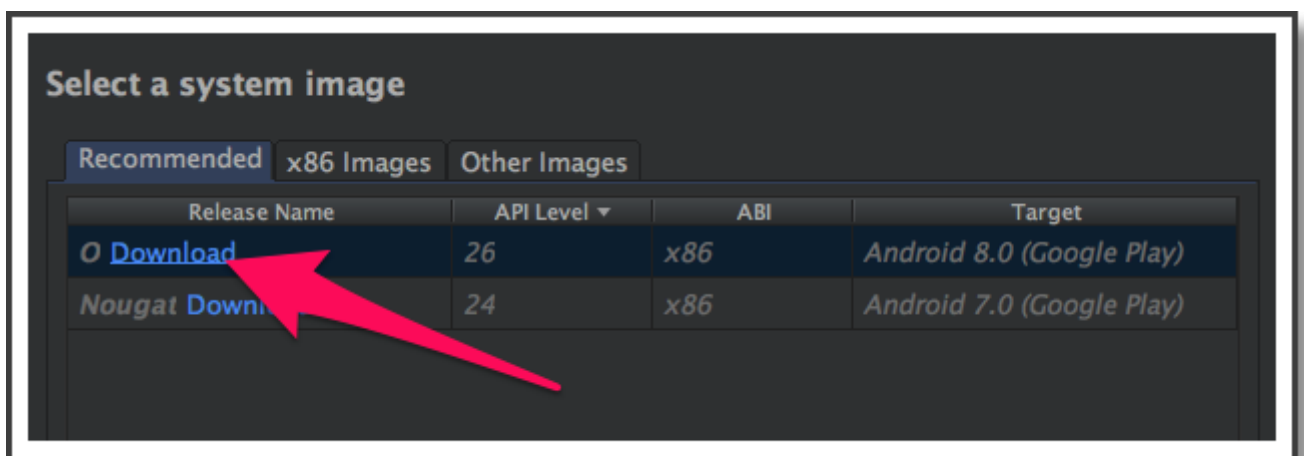
### Création d'un équipement virtuel

Afin de pouvoir lancer l'application sur l'émulateur, il est nécessaire de créer un équipement virtuel. Un équipement virtuel correspond à l'association d'un type d'équipement (la partie matérielle) avec une version d'Android (la partie logicielle). Pour cela, cliquez sur le bouton **Create New Virtual Device**.

Dans la partie gauche de la fenêtre, vous pouvez sélectionner le type d'équipement. Dans notre cas, ce sera **Phone**, mais à l'avenir vous pourriez choisir **Tablet** si vous souhaitiez tester l'application sur une tablette.

La partie du milieu liste l'ensemble des équipements disponibles. Chaque équipement possède des caractéristiques matérielles spécifiques (taille de l'écran, résolution, mémoire disponible, etc). Si un jour vous souhaitiez émuler un équipement très spécifique (avec un écran très grand ou peu de mémoire), vous pourriez le créer en cliquant sur le bouton **New Hardware Profile**. Pour notre besoin, vous pouvez sélectionner le *Nexus 5*, qui fera parfaitement l'affaire. Cliquez ensuite sur le bouton **Next**.

L'image système correspond à la version d'Android à installer sur l'équipement virtuel. Je vous conseille de rester dans l'onglet *Recommended* dans un premier temps. Il est possible que vous deviez installer une image système pour pouvoir aller plus loin : cliquez pour cela sur le lien **Download** à côté de l'image que vous souhaitez installer.



Lorsque vous souhaitez installer une nouvelle image, une fenêtre s'affiche. Vous devez explicitement cliquer sur chaque licence à gauche puis l'accepter en activant le bouton radio en bas de la fenêtre (pas forcément facile à voir), puis cliquer sur **Next**. Si vous n'acceptez pas la licence, le bouton Next sera désactivé. Vous n'avez pas le choix !

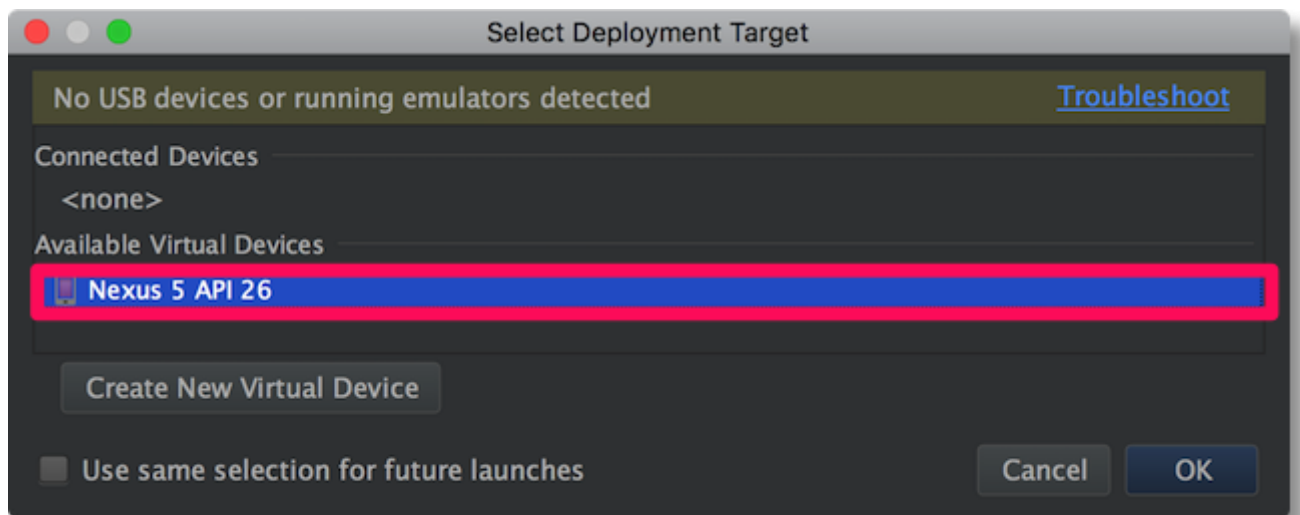


L'écran **Component Installer** s'affiche ensuite, le temps de télécharger les fichiers nécessaires. Les fichiers étant assez volumineux, vous avez sûrement le temps de vous faire un café. Ou un thé. Ou une tisane. À la fin de l'installation, cliquez sur le bouton **Finish**.

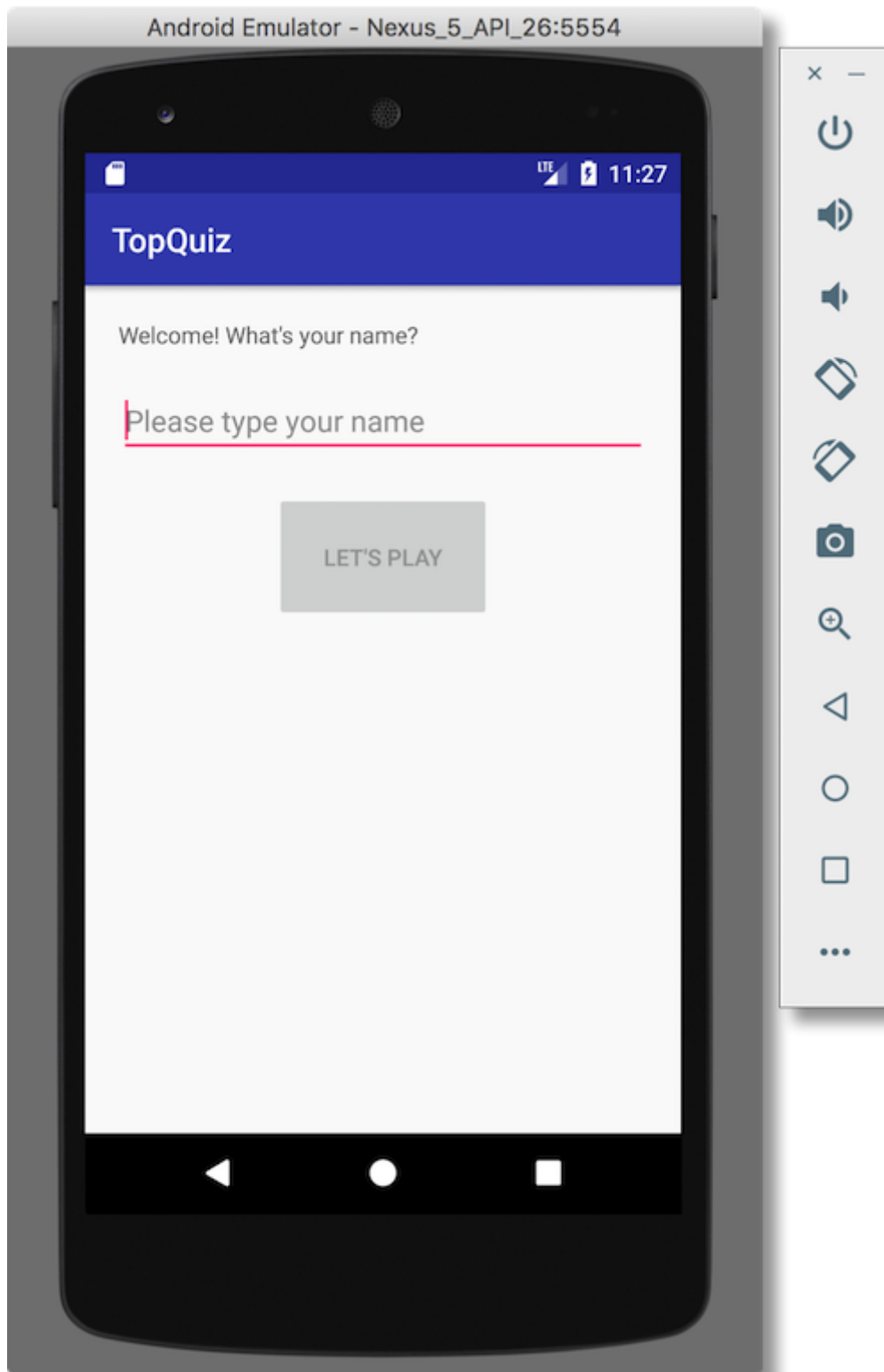
De retour sur l'écran **System Image**, cliquez sur le bouton **Next** pour arriver sur l'écran **Android Virtual Device (AVD)**. Cet écran vous permet de vérifier la configuration de votre appareil virtuel, voire d'en changer certains paramètres. Pour l'instant, je vous conseille de conserver les paramètres par défaut. Cliquez sur **Finish**.

## Lancement

Vous devez maintenant voir apparaître votre nouvel équipement virtuel dans la fenêtre. Si la fenêtre n'est pas affichée, cliquez sur le bouton de lecture vert présenté en tout début de chapitre.



Sélectionnez l'équipement puis cliquez sur **OK**. Normalement, l'émulateur devrait se lancer, et après le chargement d'Android, vous devriez voir apparaître la fenêtre suivante :



La barre d'outils sur le côté droit de l'émulateur vous permet de modifier certains paramètres. Ils sont tous assez explicites. Soyez curieux, testez chacun d'eux ! Vous pouvez même cliquer sur les trois petits points en bas pour afficher davantage de paramètres.

En lançant l'application sur l'émulateur, le focus est fait par défaut sur le premier champ de saisie : vous devriez voir le curseur positionné et clignoter sur le champ de saisie du prénom. Vous pouvez directement utiliser le clavier de votre ordinateur pour saisir votre prénom (si vous

souhaitez faire s'afficher le clavier virtuel du téléphone, il vous suffit de cliquer sur le champ de saisie avec votre souris).

Ensuite vous pouvez passer automatiquement à l'élément suivant (en l'occurrence le bouton) en appuyant sur la touche de tabulation et en appuyant sur la touche **Entrée** pour simuler le clic sur le bouton. Vous devriez voir le bouton s'animer. Il ne se passe rien d'autre, c'est normal, car nous ne l'avons pas encore développé. Cette technique de navigation à partir du clavier est très pratique pour tester un formulaire par exemple.

## Lancement sur équipement réel

---

Rien ne vaut un test sur un équipement réel ! Si vous avez la chance d'avoir un équipement Android sous le coude (téléphone ou tablette), c'est le moment de dégainer son câble USB et de le brancher à votre ordinateur.

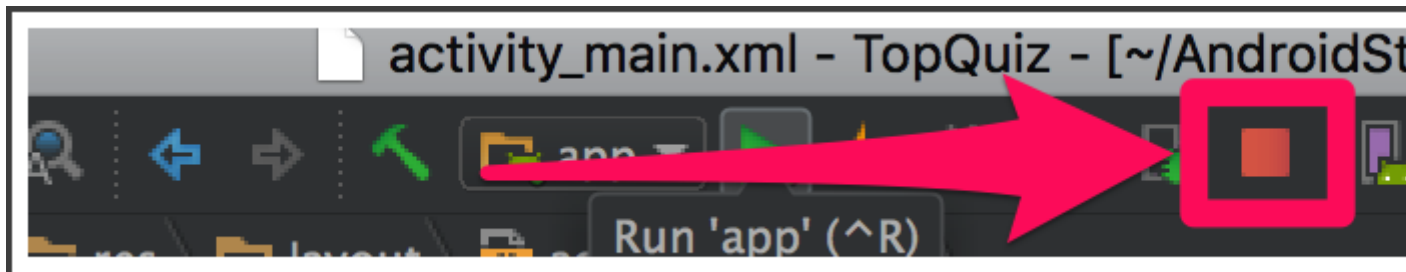
### Activation du mode développeur

Afin de permettre à Android Studio de communiquer avec votre équipement, il est nécessaire d'activer le mode développeur. Pour ce faire, suivez les instructions de la section **Enable developer options** [détaillées sur cette page](#).

Ensuite, vous devez permettre à votre système d'exploitation de détecter l'équipement. Si vous êtes sur Mac, profitez car vous n'avez rien à faire ! Si vous êtes sur Windows ou Linux, suivez les instructions détaillées dans la section **Set up a device for development** [de cette page](#).

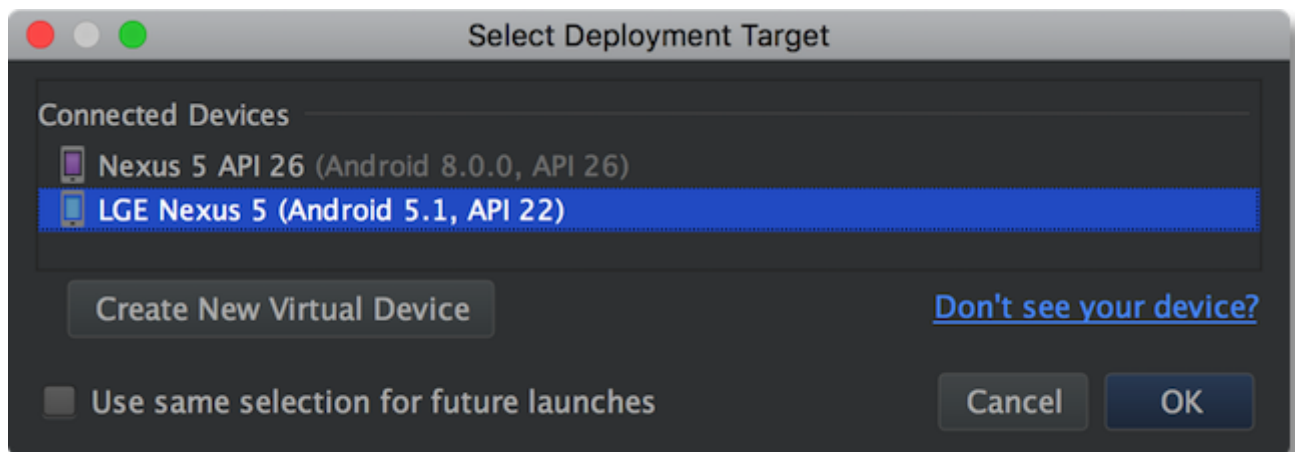
### Lancement

Si l'application est déjà lancée sur l'émulateur, arrêtez-la en cliquant sur le bouton **Stop** (carré rouge situé sur la barre d'outils d'Android Studio).



Cliquez-ici pour arrêter l'application

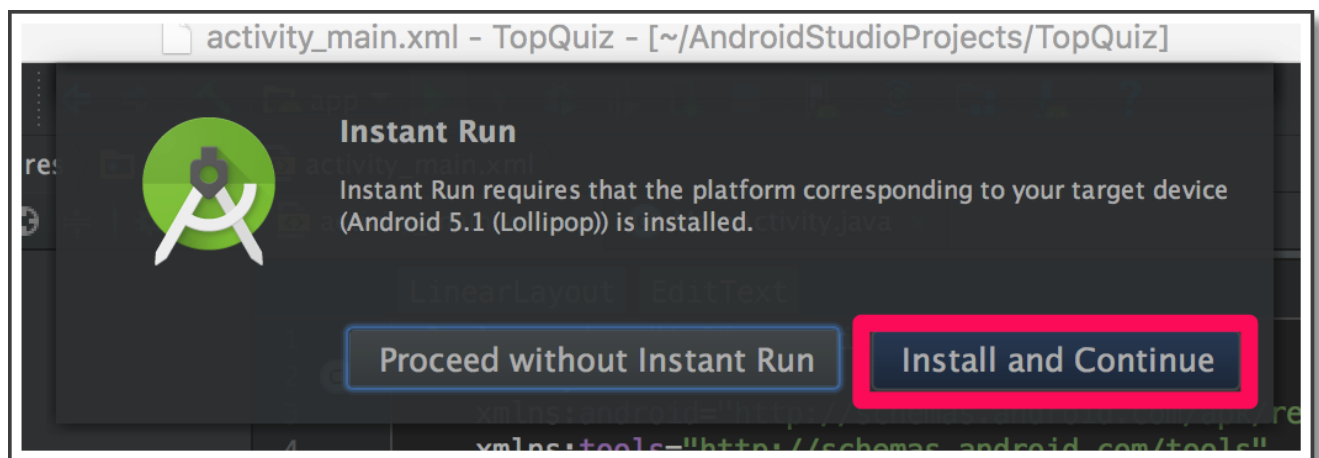
Ensuite, cliquez de nouveau sur le bouton de lancement. Vous devriez voir apparaître la fenêtre suivante, vous permettant de sélectionner si vous souhaitez lancer l'application sur l'émulateur ou votre équipement réel. Sélectionnez votre équipement puis cliquez sur le bouton **OK**.



## La fonctionnalité Instant Run

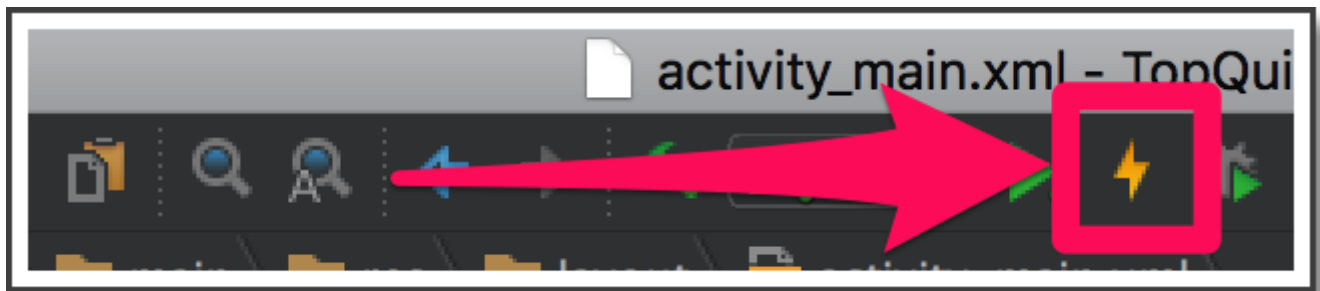
Historiquement, pour tester la moindre modification de votre code source, Android Studio devait recompiler l'ensemble de votre application et la déployer de nouveau sur l'émulateur ou l'équipement. Cela devenait vite pénible pour tester un simple changement de texte par exemple.

Heureusement, depuis Android Studio 2.3, il est possible de répercuter les modifications sur l'application lancée sans recompiler l'ensemble. Cela permet de tester plus rapidement son code. Si vous lancez l'application pour la première fois sur votre équipement, il se peut que vous voyiez apparaître la fenêtre ci-dessous.



Dans ce cas, cliquez sur le bouton **Install and Continue**. Une nouvelle fenêtre s'affiche (*Component Installer*) le temps d'installer les fichiers manquants. Appuyez sur le bouton **Finish**, et l'application devrait se lancer.

À partir de maintenant, vous pouvez cliquer sur le bouton **Apply Changes** de la barre d'outils, afin d'appliquer les modifications sans avoir à tout recompiler.



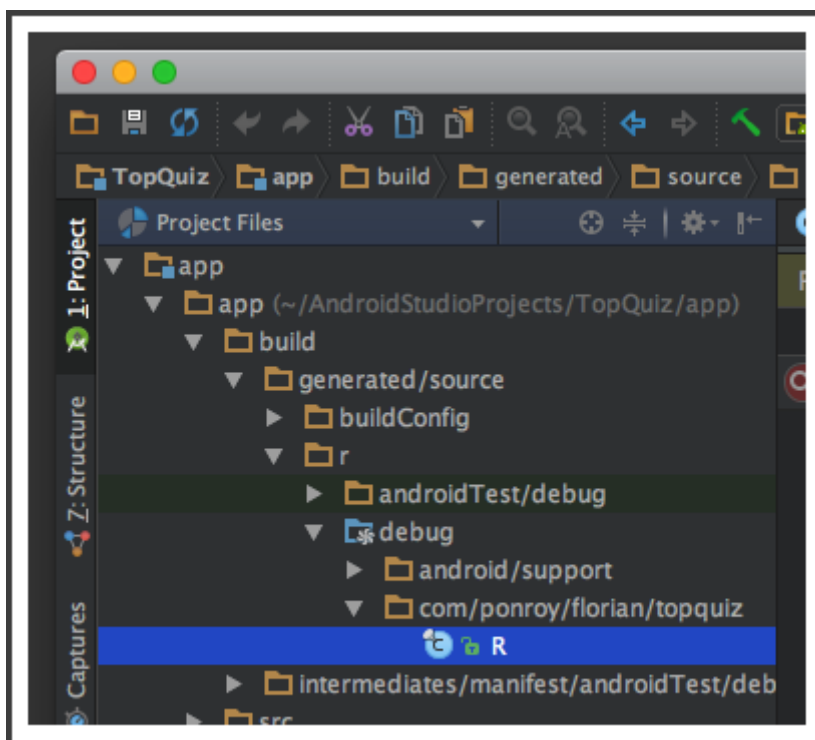
Si vous souhaitez en savoir plus sur les subtilités d'*Instant Run*, [c'est par ici](#).

## Classe R

---

Comme nous l'avons dit dans le chapitre précédent, lors de la compilation de votre application, Android Studio génère une classe Java appelée **R**. Cette classe contient l'ensemble des identifiants de toutes les ressources du projet.

Si vous souhaitez mieux comprendre son utilisation et voir à quoi elle ressemble, sélectionnez la vue **Project Files** dans l'arborescence de gauche, puis ouvrez le fichier situé dans l'arborescence présentée ci-dessous. Vous allez y trouver les identifiants que vous avez saisis dans le fichier layout, par exemple **activity\_main\_name\_input**.



## Conclusion

---

Voilà, vous savez désormais lancer votre application sur l'émulateur ou sur un équipement réel, bravo ! N'oubliez pas que tout le monde n'utilise pas forcément le même téléphone que vous : n'hésitez donc pas à utiliser l'émulateur pour tester votre application sur une ancienne version d'Android, ou sur un équipement avec un tout petit écran.



