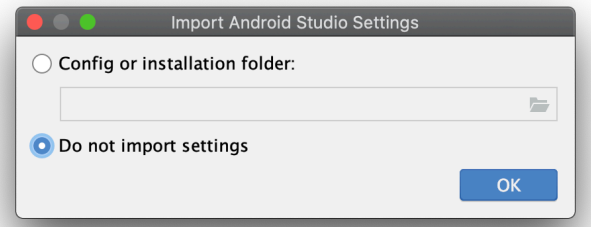
TME Android (1,2,3,4,5)

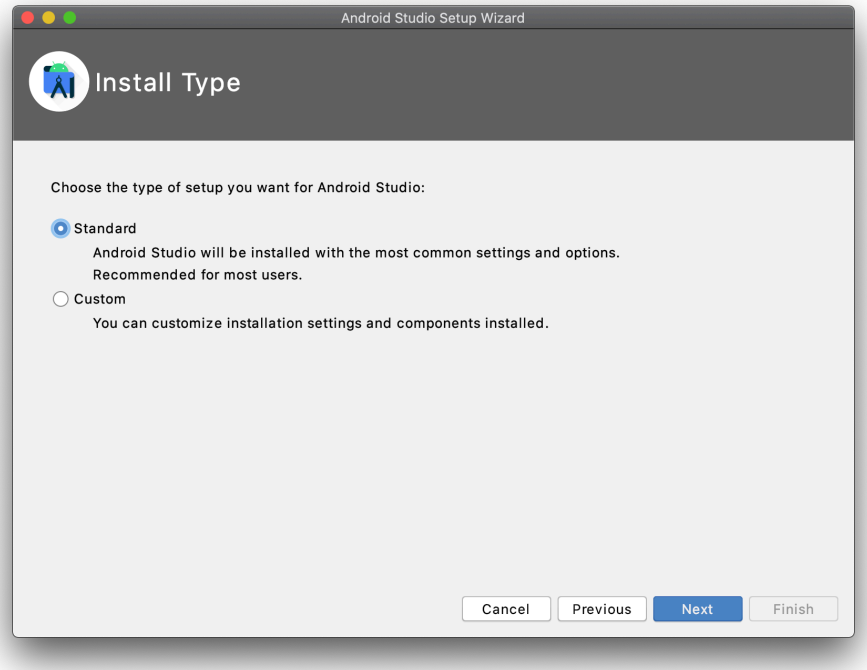
Partie I:

1. Installer Android Studio

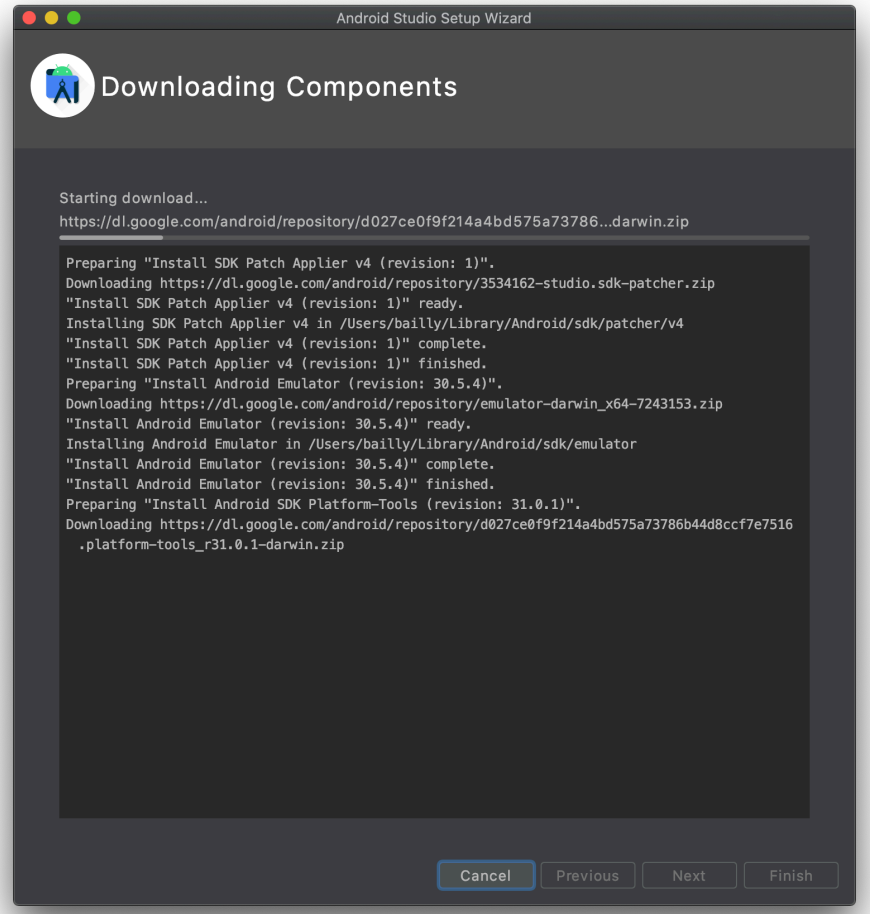
Android Studio est un logiciel pour créer et tester vos applications Android. Installer et configurer Android Studio avant la séance de TME. Lisez toutes les étapes avant de commencer.

1. Vérifiez que vous avez suffisamment de place sur votre ordinateur. Il faut au moins 2Giga, 4 recommandé.
2. Lancez Android Studio.
3. Laissez vous guider pour l'installation, mais attention, ca prend beaucoup de place.





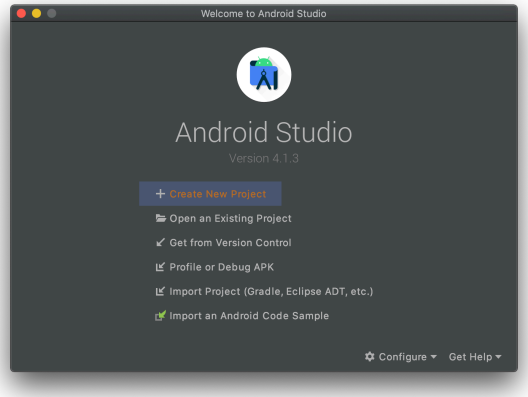
Si vous avez un problème de place et que vous avez un téléphone/tablette Android, vous pouvez tenter une installation Custom pour ne pas installer le simulateur qui occupe pas mal de place. Je vais avoir des difficultés pour vous aider en cas de problèmes.

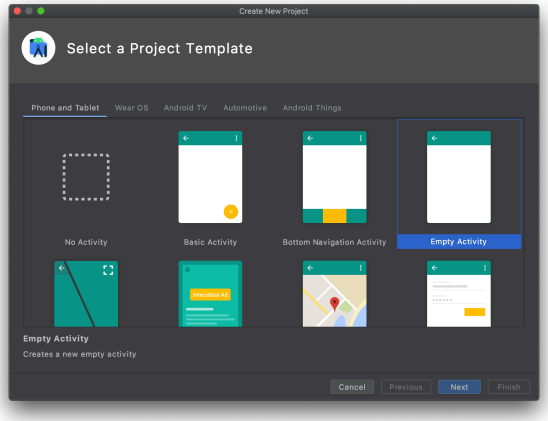


2. Hello world

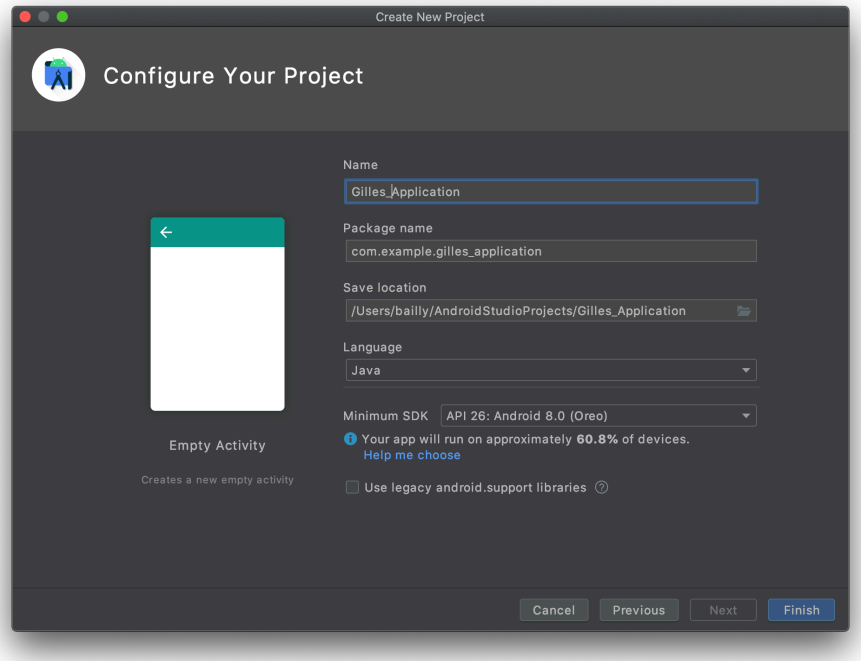
Nous allons commencer par faire un Classique Hello World sur Android

1. Créez un **nouveau project** et choisir **Empty activity**





2. Configuration du projet. Vous avez le choix du langage. Choisissez Java (Kotlin semble plus approprié maintenant, mais vous êtes probablement plus familiers avec Java). Pour l'API, choisissez API 26 : Android 8 :0 (Oreo)

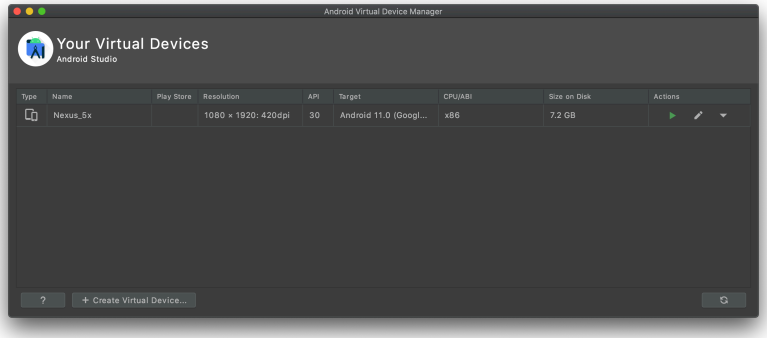


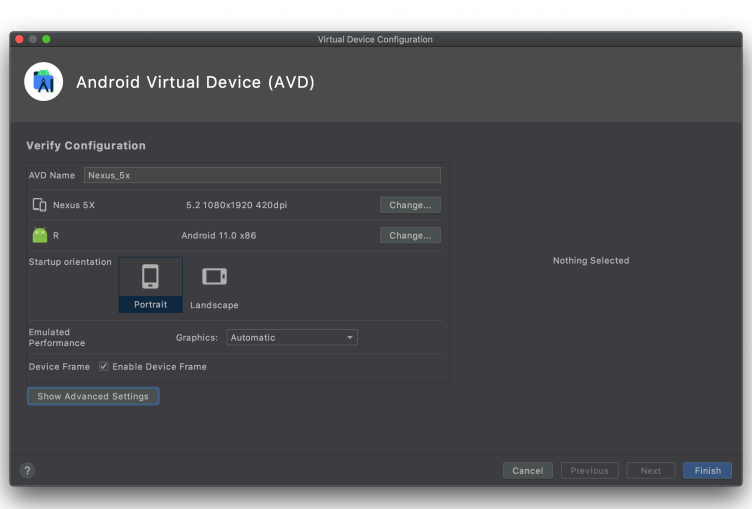
3. Vous pouvez maintenant compiler le projet via Build > Make Project

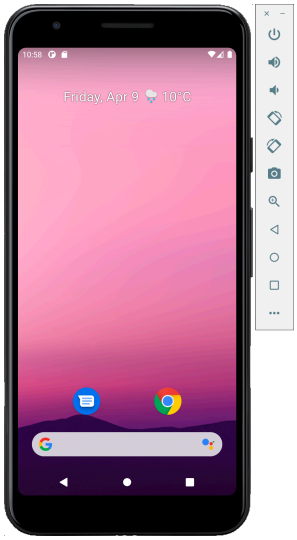
3. Préparation d'un émulateur

Si vous avez besoin d'un émulateur pour tester votre application, vous pouvez créer un nouveau **A**ndroid **V**irtual **D**evice (AVD) :

2. Dans Tools > AVD Manager, choisissez un périphérique. Il y a une icone « crayon » pour "edit this AVD". J'ai choisi un Nexus 5X.



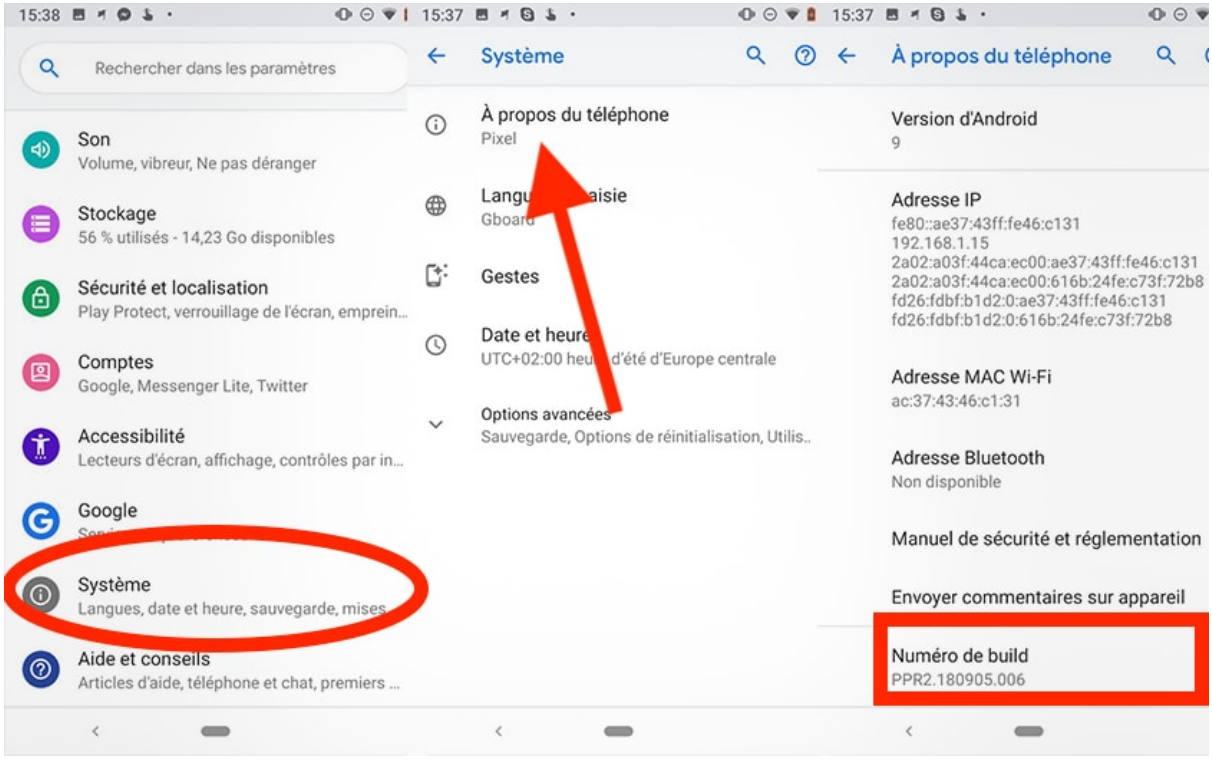




3. Si tout se passe bien, vous pouvez voir/interagir avec l'émulateur
4. Lancer votre application Run > Run app. Vous devriez voir le nom de l'application et « Hello World » dans l'émulateur
5. Vous pouvez tout quitter !

1. Utilisation de son téléphone

Si vous utilisez votre propre telephone Android, ca évite d'utiliser le simulateur qui prend énormément de place. Bien penser à le mettre en mode développeur. La manipulation est un peu bizarre. Elle consiste à aller dans "General > A propos du telephone" et cliquer sur "Numero du Build" 7 ou 8 fois. Vous obtenez une demande de confirmation. Une fois en mode développeur, quand vous retournez dans "Système", vous avez "option pour les développeurs" et dedans activez debuggage USB. Ensuite Android Studio devrait pouvoir alors détecter automatiquement le telephone.



2. Slides

des slides disponibles par James Eagan.

https://hci.isir.upmc.fr/wp-content/uploads/2019/02/1—Intro-Android.pdf

https://hci.isir.upmc.fr/wp-content/uploads/2019/02/2—Activités-Intents.pdf

Partie II: Enoncé du TME

L'objectif principal de ce TME est de se familiariser avec l'environnement de développement Android, notament :

- Créer un nouveau projet Android,
- Créer un émulateur pour tester le projet
- Comprendre l'organisation des composants d'une application Android
- Composer une interface graphique basique, et
- Y associer des fonctionalités

3. Créer un projet

Nous allons maintenant créer un premier projet Androïd. Notre application sera un convertisseur de monnaies. Dans sa première version, il y aura trois étiquettes, trois champs de texte, et deux boutons : convertir et RÀZ (remet à zéro). Commençons.

- Choisir Empty Activity, puis faire « Next ».
- Pour le nom de l'application, saisir quelque chose comme « Sous ».
- Pour package name, mettre « fr.upmc.votre_nom.sous ».
- Pour la localisation du projet sur le disque, choisir un dossir raisonnable.
- Vérifier que la configuration choisir Java et API 26, puis faire «Finish».

Si tout s'est bien déroulé, vous avez un joli projet Androïd. Aller dans le fichier app -> java -> fr.upmc.votre_nam.sous-> MainActivity. Tester le programme en cliquant sur Run > Run App pour le compiler et lancer. Si tout va bien, l'environnement lance l'émulateur. Au bout de quelques minutes, une fois l'émulateur démarré vous devrez voir « Hello World ».

4. Organisation d'un projet Android

Regardons un peu plus proche le projet qu'Android Studio nous a créé. Vous avez peut-être remarqué qu'il y a un ou deux million de dossiers dans notre projet, dont quelques uns qui nous concerne aujourd'hui :

- `app` — c'est ici où il y aura tous les contenus de notre application
- `gradle` — l'outil utilisé par Androïd pour compiler et empaqueter nos applications. C'est un concept analogue aux `Makefile`s
- `java` — ce dossier contient le code source pour notre application, dont
  - `MainActivity.java` qui est notre écran principal. Une activité correspond à peu près à un écran en Androïd.
- `res` — contient les ressources non-code de l'application
  - `drawable-*` — les images brutes à composer dans l'interface. Les versions différentes correspondent à la résolution du dispositif.
  - `layout` — les descriptions de l'interface sous format XML, dont notre

        - `activity_main.xml` qui décrit l'interface pour `MainActivity.java`.
    - `values` — nous ignorons ces fichiers sauf :
        - `strings.xml` qui contient des définitions des chaines de caractères qu'on utilise dans l'interface

La première chose qui est un peu bizarre est cette notion de séparation des chaines de caractères dans leur propre fichier. Cette approche permet de facilement créer des localisations et traductions de texte dans le projet (pareil que tr() dans Qt).

Les fichiers `layout/*.xml` sont un peu comme les designs qu'on fait dans Qt Designer. C'est un fichier XML qui décrit l'interface qui sera affichée sur l'écran. Commençons là.

5. Composer une interface

Ouvrir `layout/activity_main.xml`. Android Studio nous propose un éditeur graphique pour composer l'interface. C'est un très bon outil, mais pour mieux apprendre ce qui se passe réellement, nous allons dans un premier temps utiliser l'interface XML. Cliquer l'onglet `Code` en haut à droite de l'éditeur pour passer à la vue XML.

L'interface qu'on voit corresponde à notre « Hello World », créé automatiquement lorsque l'on crée un nouveau projet. Dans un premier temps, changer le `androix.constrainLayout.widget.ConstainLayout` en `LinearLayout` et modifier les attributs pour que ça ressemble à :

```
    <LinearLayout 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"
    android:orientation="vertical"
    tools:context=".MainActivity">
```

Tester pour vérifier que nous n'avons pas cassé l'interface. Si vous ne voyez pas « Hello World », déboguer le layout jusqu'à ce que ça marche.

Notre première version de l'interface contiendra quatre rangs : un pour une étiquette et champs de texte pour saisir le montant en euros, un pour saisir le taux d'échange, un pour indiquer la valeur équivalente en dollars, et un pour les deux boutons.

Pour créer ces rangs, il suffit de mettre un layout à l'intérieur du `LinearLayout`, créant une hiérarchie de widgets. Le `LinearLayout` que nous avons déjà créé a une orientation verticale. Pour créer les rangs, ça suffit donc de mettre

d'autres `LinearLayouts` dedans, mais avec un orientation horizontale. (Bien entendu, il y a d'autres layouts possibles.)

Ensuite, rajouter un autre `LinearLayout` dans le premier. Il n'y aura plus besoin des attributs `xmlns:*` et `tools:context`. Virer-les, et changer l'orientation en `horizontal`. De plus, on ne veut plus que la hauteur soit celle du parent. Changer-la donc en `wrap_content`, nous laissant avec :

```
<LinearLayout
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:orientation="horizontal" >
```

Déplacer le `TextView` dans ce rang et changer son texte en `@string/euros_label`. Si vous essayez de lancer l'application, Android Studio se plaindra que cette valeur n'est pas encore définie. Le compilateur est vigilant, mais nous ne sommes pas esclave à la machine ! On la définira tout à l'heure, quand on le voudra. On peut aussi supprimer les app:layout_constraintX_Y.

Ensuite, rajouter un champs de texte :

```
<EditText android:id="@+id/edit_euros"
    android:inputType="numberDecimal"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:hint="@string/euros_label" />
```

Noter que nous avons donné un `android:id` attribut à cet item. Cet attribut permettra d'accéder à ce widget dans notre code java. Le + avant `id` indique que l'on définie un nouvel objet avec cet identifiant.

Tester encore l'interface pour vérifier que ça … ne marche pas parce que nous n'avons pas encore définie la `@string/euros_label`.

Ouvrir le fichier `strings.xml` dans `res/values` et y rajouter une entrée pour `euros_label` avec la valeur « Euros » en suivant le modèle des autres strings définies dans le fichier. Tester encore et déboguer si nécessaire.

Rajouter un autre rang pour la taux d'échange. Re-compiler et lancer pour tester. Puis rajouter un rang pour le résultat. Indice : pour créer un champ de texte non-modifiable, utiliser les attributs `android:enabled="false".` et `android:focusable="false"`. Tester l'application encore une fois.

Remarquer que la mise-en-place de l'interface n'est pas terrible. Les widgets sont dimensionnés par la taille de leur contenu et non par la place disponible. Il existe une autre option, `android:layout_weight` qui donne un poids relatif au widget dans le layout. C'est comme faire un cocktail : le poids et relatif aux autres composants (e.g. pour faire une interface délicieuse, faire une part `TextView` pour deux parts `EditText`.!). Par défaut, un widget à un poids 0. Quand on utilise un `android:layout_weight` pour une dimension, on ne veut plus calculer la taille en fonction des contenus, donc on met aussi `android:layout_width` (ou `height`, le cas échéant) à `0dp` :

```
<EditText android:id="@+id/edit_euros"
    android:inputType="numberDecimal"
    android:layout_weight="2"
    android:layout_width="0dp"
    android:layout_height="wrap_content"
    android:hint="@string/euros_label" />
```

Dans un nouveau rang, rajouter les boutons. Un bouton s'écrit comme ça :

```
<Button
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="@string/button_convert"
    android:onClick="convert" />
```

L'attribut `android:onClick` relie le bouton à une méthode de l'activité associée. On peut voir cette activité dans l'attribut de `tools:context=`. Nous allons définir cette méthode prochainement. Mais d'abord, ne pas oublier de définir les strings dans `strings.xml`.

6. Implementer la fonctionnalité

La méthode de rappel (« callback ») associé à un widget via `android:onClick` devrait être void et devrait accepter un seul paramètre, la `View` qui à déclenché l'action : `public void maMethode(View sender) {}`

Implementer les méthodes de rappel référencée dans le layout. Pour retrouver un widget défini dans le layout par son identifiant, utiliser :

```
EditText euros_text = (EditText) findViewById( R.id.edit_euros );
```

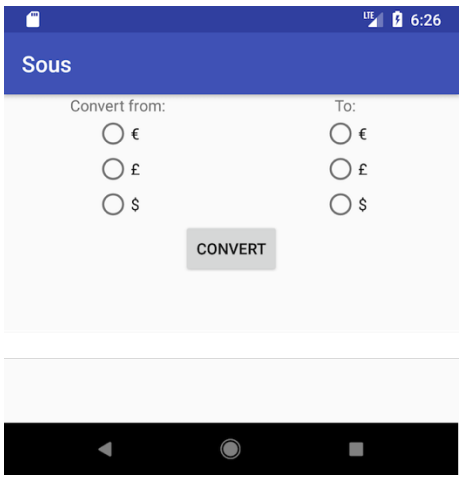
Pour retrouver et modifier la valeur d'un champ de texte, les méthodes `editText.getText().toString()` et `editText.setText(blaBlaString)` pourraient être utiles.

Partie III

L'objectif de cette partie est de découvrir comment relier plusieurs Activity en utilisant des Intents.

Nous allons créer une interface basique pour renseigner les taux d'échange où l'utilisateur choisit la monaie source et cible (le taux de change sera câblé en dur). Ceci n'est pas forcément l'interface la plus élégante, mais elle va nous donner un peu d'expérience avec des `Activity` et `Intent`.

Cette nouvelle `CurrencyChooserActivity` présentera deux colonnes de radio boutons: la monnaie source, et la monnaie cible. Voici un exemple :



1. Créer une nouvelle Activity

Rajoutez un nouvel `Activity` à votre projet (File > New > Activity). On peut l'appeler `CurrencyChooserActivity`. Dans son layout XML, créer l'interface dans l'esprit de l'interface dessus. Vous devrez utiliser, en particulier, des RadioGroup qui hérite de Layout.

```
<RadioGroup android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:id="@+id/currency_from"
    android:checkedButton="@+id/currency_euro_from">
```

Et des RadioButton qui hérite de Button

```
<RadioButton android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:id="@+id/currency_euro_from"
    android:text="@string/currency_euros" />
```

Vous pouvez basculer entre les vues code et Design de AndroidStudio pour voir l'affichage sans devoir lancer l'application.

2. Relier les deux activités

Tout d'abord, il faut créer un bouton dans MainActivity (comme précédemment) qui lorsque l'utilisateur tapera dessus, lancera la CurrencyChooserActivity.

Il faut ensuite créer une Intent. Une Intent exprime une action et ses paramètres. Elle permet dans notre cas de naviguer entre des Activitys. Ajouter ce code dans MainActivity.

```
public void choose_currency(View sender){
    Intent intent = new Intent(this, CurrencyChooserActivity.class);
    startActivity(intent);
}
```

Tester votre code.

On souhaite également que l'activité retourne la devise source et la devise cible. Commencez tout d'abord par ajouter une variable d'instance pour définir l'identifiant de ChooseCurrencyActivity dans MainActivity

```
static final int CURRENCY_CHOOSER_REQUEST = 1;
```

Sa valeur a peu d'importance. Il faut simplement qu'elle soit unique si vous aviez des activités différentes à démarrer.

Ensuite, modifiez l'appel de création de la *CurrencyChooserActivity* en utilisant cette nouvelle méthode :

```
public void choose_currency(View sender){
    Intent intent = new Intent(this, CurrencyChooserActivity.class);
    startActivityForResult( intent, CURRENCY_CHOOSER_REQUEST );
}
```

Avec ce nouvel appel, la méthode

```
protected void onActivityResult( int requestCode, int resultCode, Intent data){}
```

sera appelée lorsque la l'activité sera terminée. et permettra de récupérer le résultat. Ajouter cette méthode dans MainActivity, mais on ajoutera son contenu plus tard.

Pour l'instant, rendez-vous dans la *CurrencyChooserActivity*. Lorsque l'utilisateur tap sur le bouton CONVERT, il faut que l'activité se termine et retourne les deux monnaies.

Pour ce faire, il est nécessaire de nouveau de créer, dans le callback du bouton CONCERT, un **Intent** (nous avions déjà utilisé cette classe pour demander le démarrage de la CurrencyChooserActivity depuis la *MainActivity*). Dans le cas présent, nous allons nous en servir pour lui "attacher" la monnaie source et la monnaie cible, afin de pouvoir les renvoyer à la première activité.

Tout d'abord, pour "attacher" une valeur à un **Intent**, il est nécessaire d'utiliser la méthode **putExtra()** qui nécessite la définition de clés (qui seront récupérées dans *MainActivity)* :

```
public static final String BUNDLE_EXTRA_CURRENCY_FROM = "CURRENCY_FROM";
public static final String BUNDLE_EXTRA_CURRENCY_TO = "CURRENCY_TO";
```

Maintenant, on peut créer un **Intent**, attacher les paramètres à l'intent, indiquer au système que l'activité s'est correctement terminée et finalement terminer l'activité.

```
Intent res_intent = new Intent();
res_intent.putExtra(BUNDLE_EXTRA_CURRENCY_FROM, "dollar");
res_intent.putExtra(BUNDLE_EXTRA_CURRENCY_TO, "pound");
setResult(RESULT_OK, res_intent);
finish();
```

Dans l'exemple ci-dessus, les monnaies sources et cibles retounées sont codées en dur. Récupérer celles des RadioButton sélectionnés à l'aide de la méthode *getCheckedRadioButtonId()* de *RadioGroup* qui retourne l'identifiant du radioButton sélectionné.

Revenons mainenant dans MainActivity pour implémenter *onActivityResult* :

```
protected void onActivityResult( int requestCode, int resultCode, Intent data){
    if ( requestCode == CURRENCY_CHOOSER_REQUEST ){
        if ( resultCode == RESULT_OK ){
            String currency_from = data.getStringExtra("CURRENCY_FROM");
            String currency_to = data.getStringExtra("CURRENCY_TO");
```

où vous verifiez l'identifiant de la requête et si l'activity s'est correctement terminée. Vous pouvez alors récupérez les valeurs retournées grâce à leurs identifiants.

Finalisez l'application pour mettre à jour le taux de change (que vous pouvez coder en dur).

Partie IV

Félicitations, votre application Androïd marche. S'il vous reste du temps, penser à améliorer l'interface. Google propose des bons guides sur :

https://developer.android.com

- Layouts : https://developer.android.com/guide/topics/ui/declaring-layout.html
- Widgets : https://developer.android.com/guide/topics/ui/look-and-feel/themes

Quelles modifications feriez-vous ?

Des choses que vous pouvez tester

Il est possible d'utiliser le vibreur du téléphone (si un vibreur est accessible). Pour le faire vibrer pendant 1 seconde (1000 millisecondes), on fait :

```
Vibrator vibrator = (Vibrator) getSystemService(Context.VIBRATOR_SERVICE);
vibrator.vibrate(VibrationEffect.createOneShot(200,
VibrationEffect.DEFAULT_AMPLITUDE));
```

Rajoutez cela dans le code et testez sur l'émulateur et sur le téléphone.

Si tout va *bien*, l'application va planter avec une exception de type java.lang.SecurityException. Cela est normal : pour le système Android, l'accès à certaines fonctionnalités doit être signalé à l'utilisateur. Il faut donc rajouter la permission appropriée dans le *manifest* de l'application. La page de documentation, https://developer.android.com/reference/android/os/Vibrator.html, de la classe Vibrator nous indique effectivement que la fonction vibrate() a besoin de la permission VIBRATE.