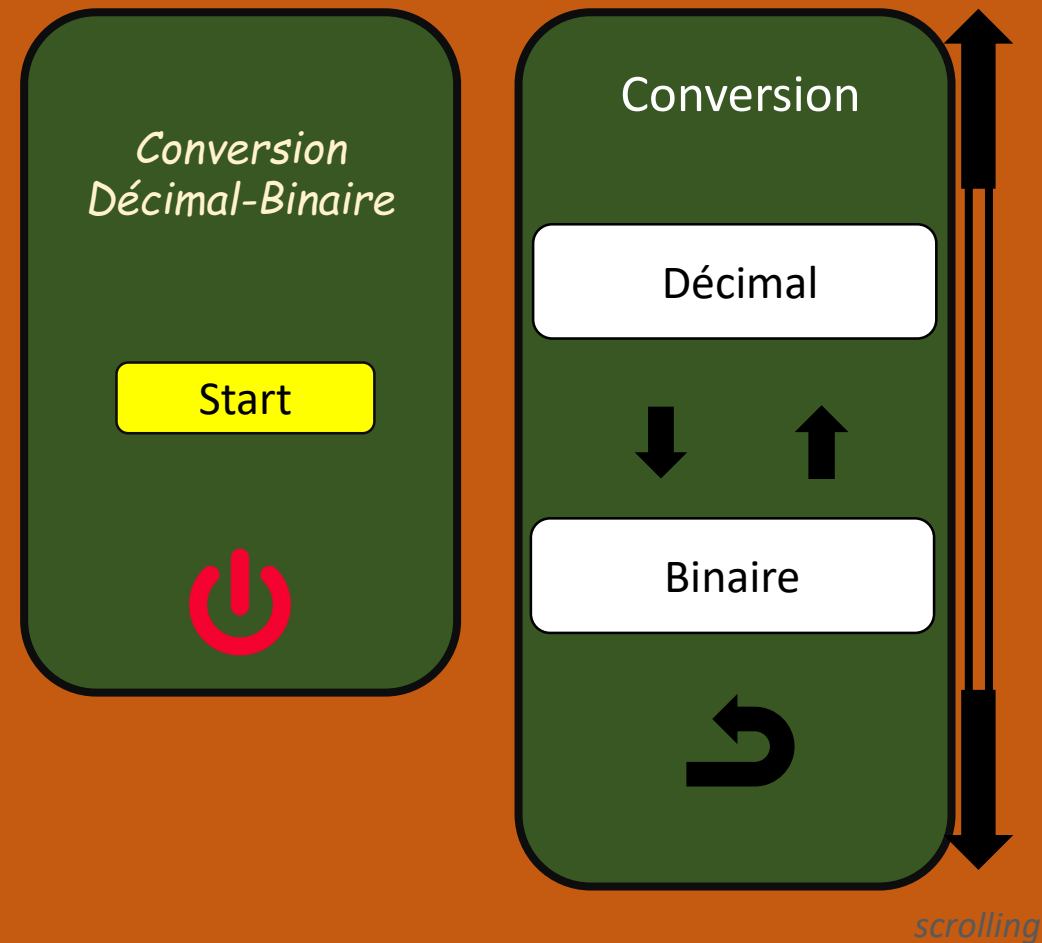


Deuxième exercice



IHM de la maquette

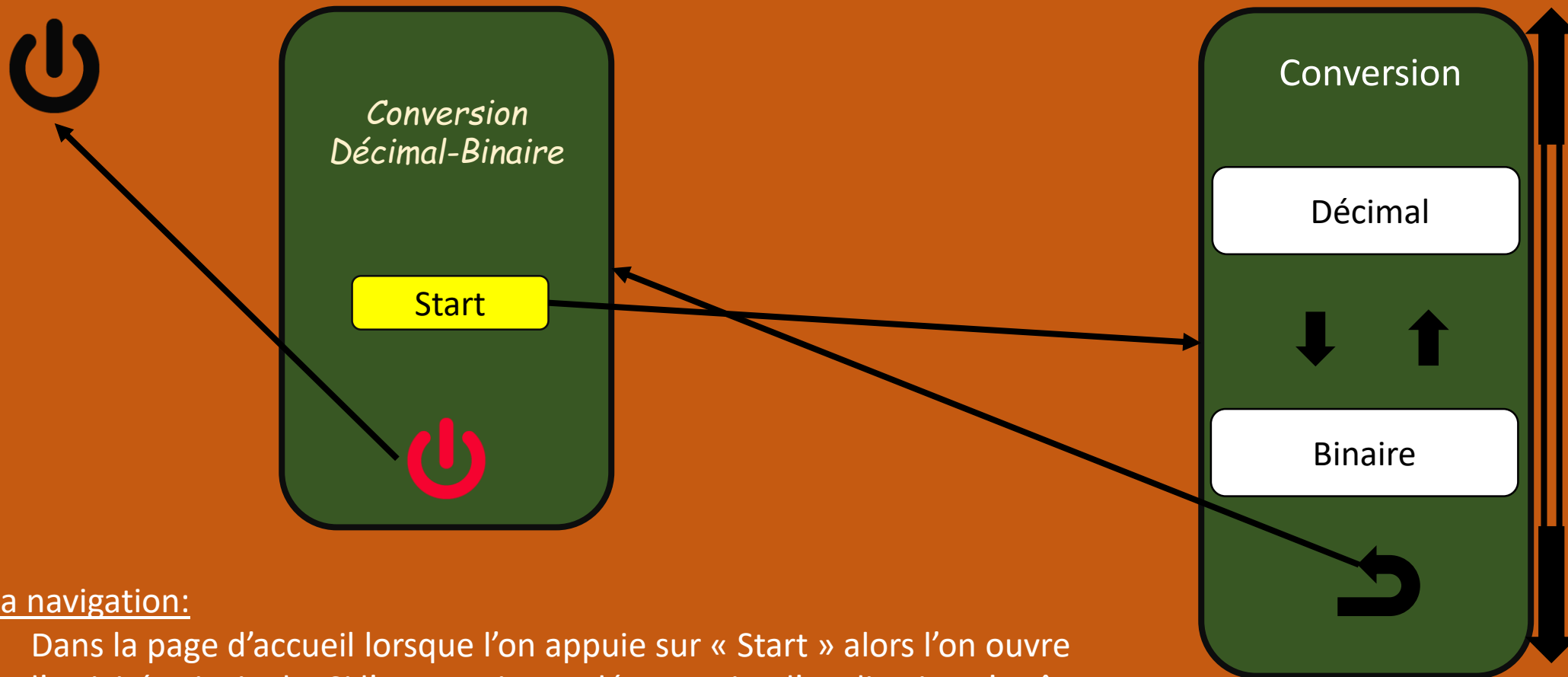


Titre: « Conversion décimal – binaire - hexadécimal »

Description:

- Dans cet exercice détaillé et guidé, il s'agit de développer une application simple, pour convertir des valeurs décimales entières en valeurs binaire et inversement.
- Nous allons développer l'application étape par étape. Elle est composée de deux activités. L'activité d'ouverture et la page d'accueil et la deuxième activité et la principale pour les conversions.
- L'application doit être responsive -> prendre en considération la rotation de l'écran et les différentes tailles d'écrans.

Navigation



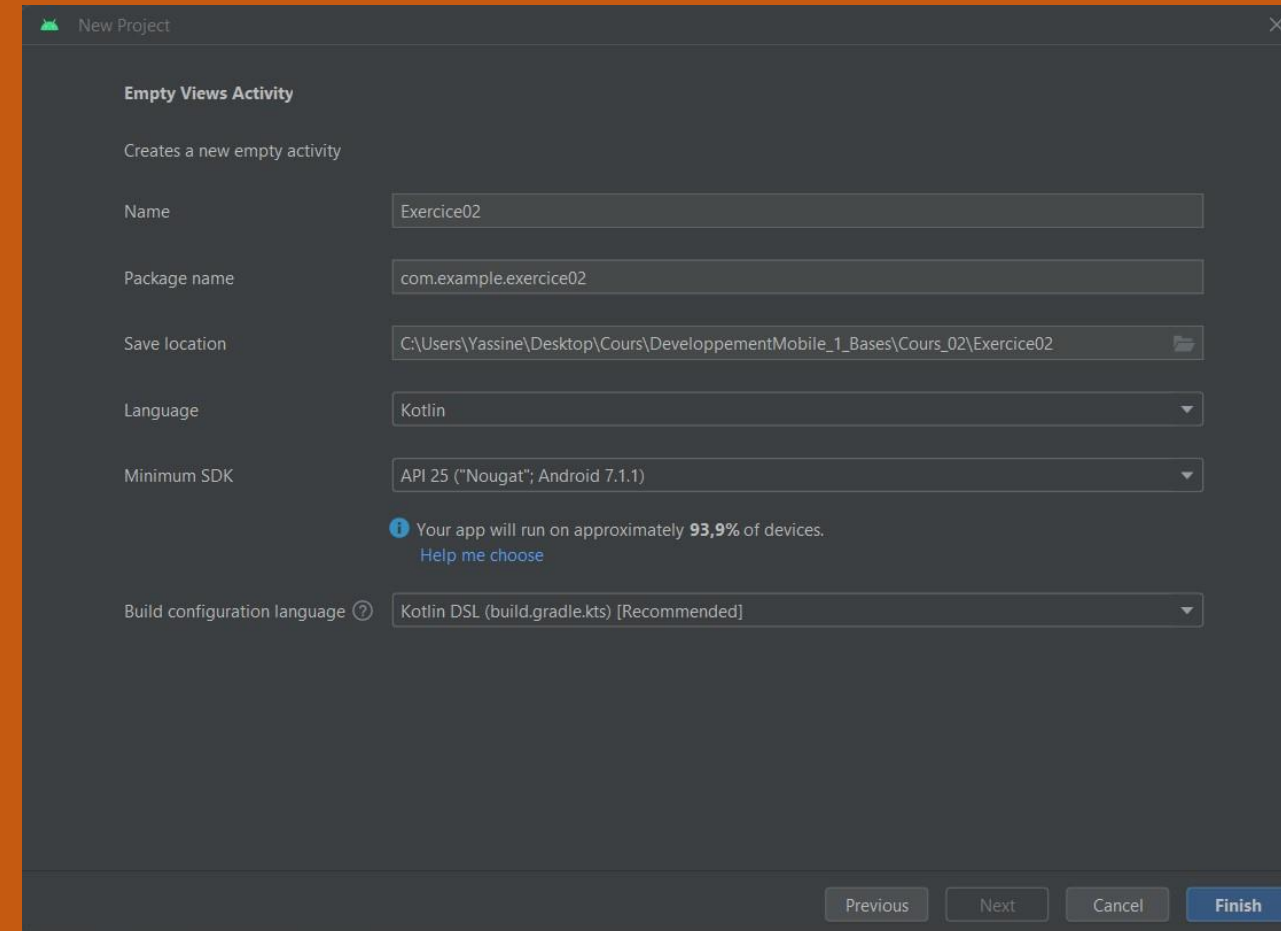
La navigation:

- Dans la page d'accueil lorsque l'on appuie sur « Start » alors l'on ouvre l'activité principale. Si l'on appuie sur déconnexion l'application s'arrête.
- Dans l'activité principale si l'on appuie sur retour alors l'on est redirigé vers la page d'accueil.

Commencer un nouveau projet

Tout d'abord ouvrons l'outil SDK Android-Studio

1. Choisir une « Empty view activity » dans les projets pour smartphones
2. Commençons un nouveau projet pour smartphones: donnons lui le titre Exercice02.
3. Choisissez où sauvegarder le dossier du projet
4. Le langage de programmation est « Kotlin ».
5. Le niveau d'API dépendent de votre version d'Android – choisissez une version qui soit cohérente et inférieur avec votre smartphone (ou la machine virtuelle que vous installée).



L'IHM de la page d'accueil part 1/6

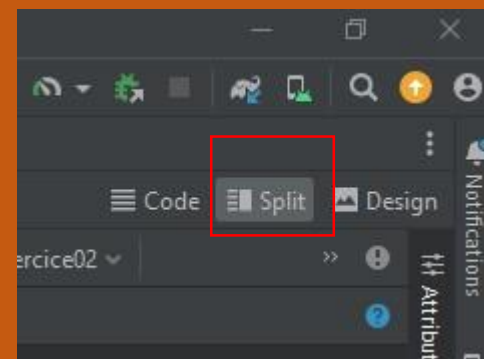
Commençons par définir l'IHM (Interface Humain-Machine) de la page d'accueil:

- Sélectionnons le fichier activity_main.xml
- Supprimons le TextView déjà présent
- Choisir le mode « Split », il permet de composer l'IHM via le code et peut s'avérer utile pour vérifier et valider l'agencement.
- Ensuite remplacer `androidx.constraintlayout.widget.ConstraintLayout` par `RelativeLayout`, cela nous permettra d'agencer plus librement nos composants sur l'écran.
- Nous souhaitons que notre application puisse offrir l'option de scroller l'écran au cas où des composants n'apparaîtraient pas sur l'écran (dû aux différentes tailles d'écrans). Pour scroller il faut ajouter un Container « ScrollView », nous allons le faire directement dans le code.
- D'abord il faut savoir que ce container ne peut avoir qu'un seul enfant. Ici l'enfant sera « RelativeLayout », et « RelativeLayout » aura lui plusieurs enfants -> les composants.
- Dans le code ajouter juste en dessous de `<?xml version="1.0" encoding="utf-8"?>` -> `<ScrollView` une erreur s'affiche ce qui est logique, nous allons déplacer ces trois ligne depuis la balise Layout dans la balise « ScrollView »

```
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"
```

- Ajoutez devant « layout_width = "match_parent" » et « layout_height = "wrap_content" » `android:`
- et n'oubliez pas de refermer la balise tout à la fin `</ScrollView>`.

Voilà « ScrollView » est le Layout principal de notre IHM et il englobe tous les autres Layouts.



L'IHM de la page d'accueil part 2/6

Vous devriez avoir un code qui ressemble à cela:

```
1  <?xml version="1.0" encoding="utf-8"?>
2  <ScrollView
3      xmlns:android="http://schemas.android.com/apk/res/android"
4      xmlns:app="http://schemas.android.com/apk/res-auto"
5      xmlns:tools="http://schemas.android.com/tools"
6      android:layout_width="match_parent"
7      android:layout_height="wrap_content">
8
9      <RelativeLayout
10         android:layout_width="match_parent"
11         android:layout_height="wrap_content"
12         tools:context=".MainActivity">
13
14     </RelativeLayout>
15 </ScrollView>
```

L'IHM de la page d'accueil part 3/6

Les valeurs « match_parent » et « wrap_content » permettent d'étendre le layout sur la largeur de l'écran du smartphone utilisé ou selon la quantité de composant à afficher. Il permettent donc un affichage responsive_design.

Maintenant nous allons ajouter les trois composants de cette IHM, c'est à dire un titre, un bouton Start et un bouton avec une image par dessus:

- D'abord changer la couleur d'arrière fond du Layout général « ScrollView » avec l'attribut `android:background="@android:color/holo_green_dark"`, vous pouvez choisir la couleur qui vous convient, ici la maquette indique du vert.
- Maintenant ajoutez un titre à votre composition « Conversion Décimal-Binaire », il s'agit d'un « TextView » `<TextView` à l'intérieur du « RelativeLayout ». Choisissez une hauteur (unité dp), la largeur doit être « match_parent ». Définissez une marge par rapport au haut, ainsi qu'une taille, une couleur de police et un alignement des caractères. Pour prévisualiser ces options il vous suffit d'écrire une nouvelle ligne à l'intérieur des balises en commençant par `android:`, après les deux points ils suffit d'écrire approximativement l'attribut que l'on souhaite modifier pour le trouver si il existe.
- On oublie pas de fermer la balise `</>`
- Avant d'ajouter les deux boutons, ajouter aussi un attribut `android:scrollbars="vertical"` à l'intérieur de la balise `<ScrollView` pour spécifier la direction scroller.
- N'oubliez pas de modifier l'id de vos composant ici appelez le `android:id="@+id/text_title"`

L'IHM de la page d'accueil part 4/6

La balise TextView devra ressembler à ceci:

```
11 <RelativeLayout
12     android:layout_width="match_parent"
13     android:layout_height="wrap_content"
14     tools:context=".MainActivity">
15
16     <TextView
17         android:id="@+id/text_title"
18         android:layout_width="match_parent"
19         android:layout_height="100dp"
20         android:layout_alignParentTop="true"
21         android:layout_alignParentEnd="true"
22         android:layout_centerHorizontal="true"
23         android:layout_marginTop="125dp"
24         android:text="Conversion Décimal-Binaire"
25         android:textAlignment="center"
26         android:textColor="@color/white"
27         android:textSize="40sp" />
28
29 </RelativeLayout>
```



Conversion
Décimal-Binaire

L'IHM de la page d'accueil part 5/6

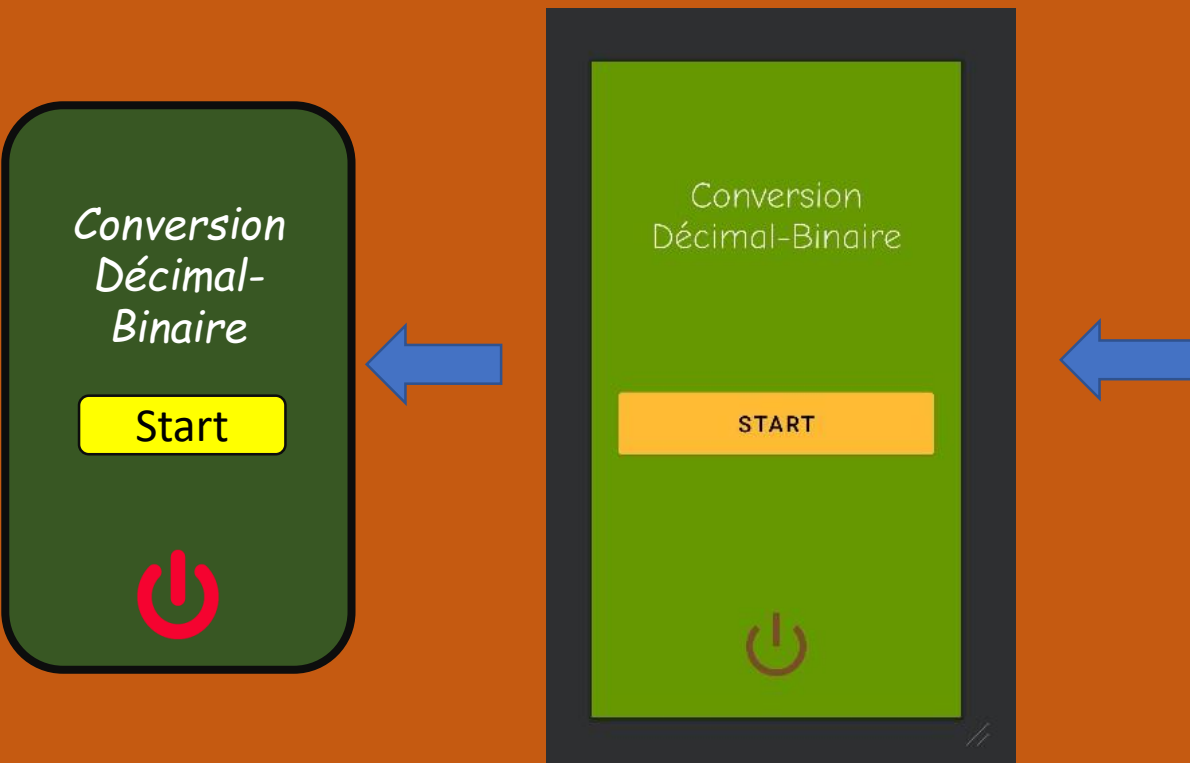
Maintenant nous allons ajouter les deux boutons, ils sont différents l'un est basique l'autre est aussi une image:

- Ajouter après le TextView un « Button », toujours dans « RelativeLayout » `<Button` On peut lui attribuer l'id « button_start »
- Choisissez les dimensions du bouton – vous devez le placer en dessous du Titre dans le visuel, relié les côtés du boutons aux côtés du titre pour que les deux composants soient solidaires. Ainsi peu importe la taille des écrans, si l'un bouge l'autre s'aligne avec. Si le titre prend toute la largeur de l'écran alors automatique le bouton va définir deux marges par rapport à la largeur de l'écran.
- Pour changer la couleur d'arrière fond d'un bouton il faut utiliser l'attribut `android:backgroundTint="@android:color/holo_orange_light"`
- On peut aussi changer la couleur et la taille du texte (unité sp).
- Maintenant ajouter un nouveau bouton avec une image « ImageButton » `<ImageButton` On peut lui attribuer l'id « button_deconnexion » -> choisir une image qui correspond à la déconnexion d'une application parmi les choix.
- Pareil on lui choisit des dimensions quelconque, une couleur d'arrière plan, puis on relie ces marges à celle du composant juste au dessus de lui, c'est-à-dire le bouton « Start ».
- Lui ajouter l'attribut « scaleType » qui permet si la valeur est “fitCenter” d'agrandir l'image et de la centrer.
- En ce qui concerne l'importation de l'image pour ce bouton si une erreur il y'a alors il faut avoir comme attribue:

```
app:srcCompat="@android:drawable/ic_lock_power_off"
```

L'IHM de la page d'accueil part 6/6

Les balises pour les deux boutons doivent être comme ceci:

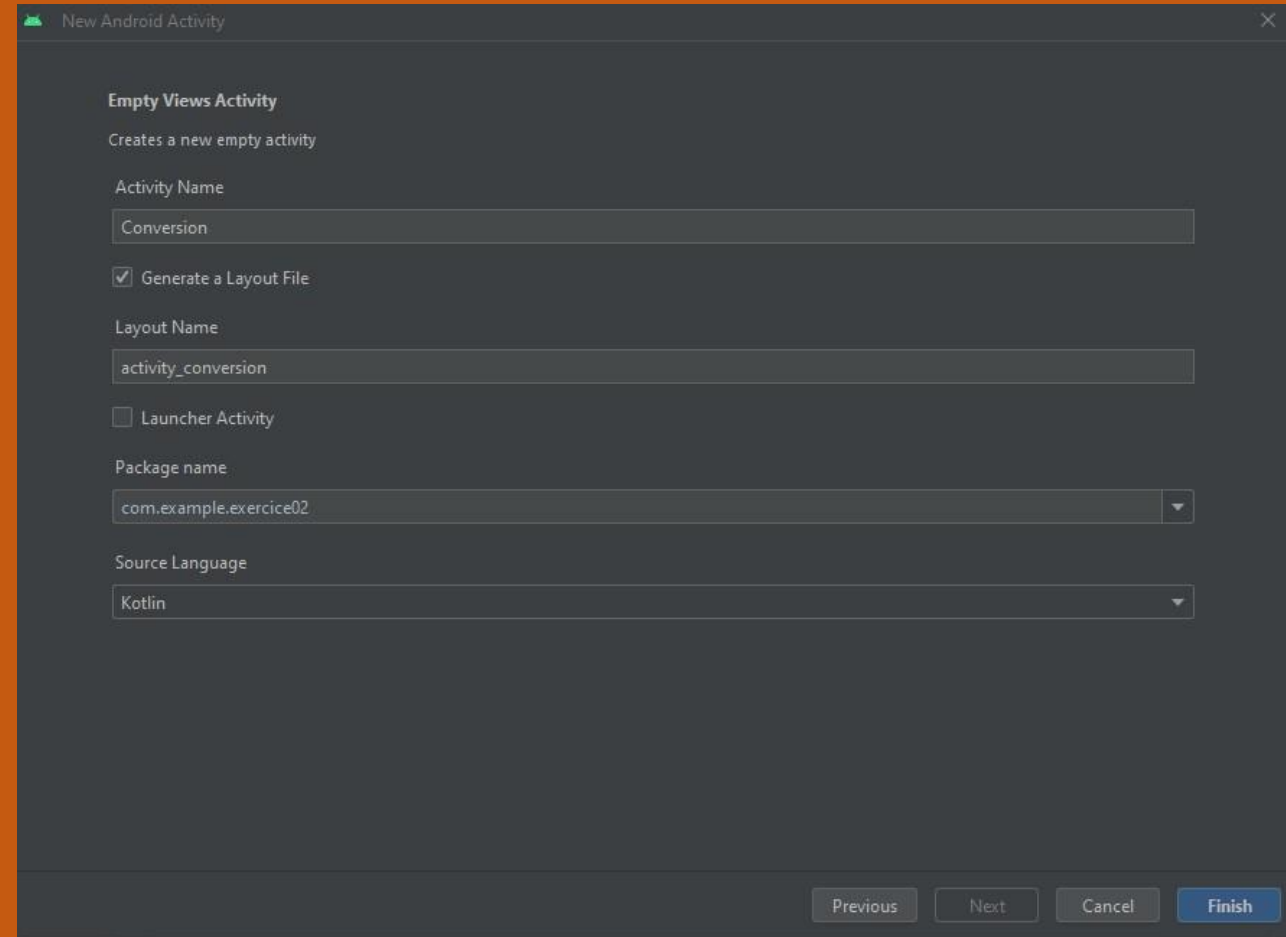


```
29 <Button
30     android:id="@+id/button_start"
31     android:layout_width="241dp"
32     android:layout_height="81dp"
33     android:layout_below="@+id/text_title"
34     android:layout_alignEnd="@+id/text_title"
35     android:layout_alignParentStart="true"
36     android:layout_marginStart="30dp"
37     android:layout_marginTop="138dp"
38     android:layout_marginEnd="30dp"
39     android:backgroundTint="@android:color/holo_orange_light"
40     android:text="@string/start"
41     android:textColor="@color/black"
42     android:textSize="25sp" />
43
44 <ImageButton
45     android:id="@+id/button_deconnexion"
46     android:layout_width="117dp"
47     android:layout_height="108dp"
48     android:layout_below="@+id/button_start"
49     android:layout_alignStart="@+id/button_start"
50     android:layout_alignEnd="@+id/button_start"
51     android:layout_marginStart="60dp"
52     android:layout_marginTop="154dp"
53     android:layout_marginEnd="60dp"
54     android:backgroundTint="@android:color/holo_green_dark"
55     android:contentDescription="@string/bouton_d_connexion"
56     android:scaleType="fitCenter"
57     app:srcCompat="@android:drawable/ic_lock_power_off"
58     app:tint="@color/black" />
59
60 </RelativeLayout>
```

Activité d'ouverture – principale : la navigation

Maintenant que nous avons notre page d'accueil, créons notre activité principale pour ensuite pouvoir l'ouvrir en appuyant sur le bouton Start.

- Pour créer une nouvelle activité faite un clique droit dans la fenêtre avec l'arborescence des fichiers, sélectionner New/Activity/Gallery/
- Choisir encore une « Empty view activity ». Donnez lui le nom « Conversion ».
- Ajouter un TextView dans le fichier qui viens de se créer : activity_conversion.xml à l'intérieur du « ConstraintLayout »
- Écrire dans ce texte le mot « Conversion »



Activité d'ouverture – principale : la navigation

Maintenant que nous avons une deuxième activité nous pouvons faire une navigation.

- Dans le fichier kotlin: MainActivity.kt nous devons instancier le bouton « Start » afin de pouvoir l'utiliser.
- Dans ce fichier l'on remarque qu'une fonction nommé « onCreate » existe déjà, en effet elle s'applique lors de la création de l'application c'est-à-dire son ouverture. Et à l'intérieur on remarque la ligne `setContentView(R.layout.activity_main)` qui permet de relier ce fichier kotlin au layout « activity_main ». Donc si cette activité (le fichier Kotlin) s'ouvre alors le layout qui y est associé s'ouvre aussi.
- Écrivez juste en dessous de `class MainActivity : AppCompatActivity() {` la ligne suivante: `private lateinit var start: Button` cette ligne nous permettra de donner une valeur et de la recharger plus tard pour la variable modifiable « start ». L'intérêt de déclarer son existence ici est de pouvoir l'utiliser dans toutes les fonction de cette classe « MainActivity » sans avoir besoin de la redéfinir.
- À l'intérieur de la fonction « onCreate() » nous allons attribuer les valeurs initiales à nos variables de classe. start est une variable de type Button, pour lui attribuer un bouton associé il faut écrire : `start = findViewById<Button>(R.id.button_start)` R.id correspond dans les ressources à un fichier qui recense tous les identifiant et leurs objets associés pour les utiliser.
- Maintenant nous devons déclencher une interaction lorsque l'on appuie sur le bouton. Au lieu de définir cette événement dans « onCreate() », on va créer une fonction pour les actions qui consiste à appuyer sur des boutons. Créez une fonction privée du nom de « buttonClick() ». Utilisez la méthode « .setOnClickListener{ } » qui permet lors d'un click d'activer une interaction. Ajoutons une autre variable de la même façon que « start » que l'on nomme « open » qui est du type Intent(), c'est une instance qui permet de charger des activités et les ouvrir. `private lateinit var open: Intent` Importer la librairie Intent () si besoin. Puis dans le « onCreate » `open = Intent(this@MainActivity, Activity2::class.java)` et `buttonClick()` pour appliquer la fonction.
- Puis l'on peut utiliser la méthode start dans la fonction « buttonClick() » pour commencer une nouvelle activité.

```
private fun buttonClick() {  
    start.setOnClickListener {  
        startActivity(open)  
    }  
}
```

Activité d'ouverture – principale : la navigation

Maintenant en appuyant sur le bouton « Start » on peut ouvrir la nouvelle activité.

Votre code devrait ressembler à ceci:

- Ajouter une troisième variable associé au bouton de déconnexion, de la même façon que le bouton « start ». Lorsque le click est détecté il faut utiliser la fonction « finish() ». Cette fonction permet de stopper l'activité en cours, si il s'agit de la première activité alors de stopper l'application.

```
1 package com.example.exercice02
2
3 import android.content.Intent
4 import androidx.appcompat.app.AppCompatActivity
5 import android.os.Bundle
6 import android.widget.Button
7
8 class MainActivity : AppCompatActivity() {
9
10     private lateinit var start: Button
11     private lateinit var open: Intent
12
13     override fun onCreate(savedInstanceState: Bundle?) {
14         super.onCreate(savedInstanceState)
15         setContentView(R.layout.activity_main)
16         start = findViewById(R.id.button_start)
17         open = Intent(packageContext, this@MainActivity, Conversion::class.java)
18         buttonClick()
19     }
20
21     private fun buttonClick() {
22         start.setOnClickListener { it: View!
23             startActivity(open)
24         }
25     }
26
27 }
```

Activité d'ouverture – principale : la navigation

Maintenant en appuyant sur le bouton de déconnexion l'on peut fermer l'application.

Votre code devrait maintenant ressembler à ceci:

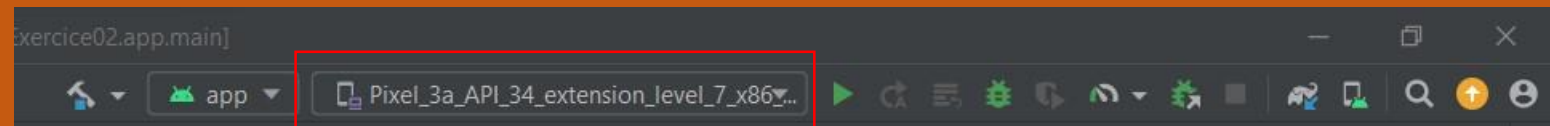
```
8
9 class MainActivity : AppCompatActivity() {
10
11     private lateinit var start: Button
12     private lateinit var open: Intent
13     private lateinit var close: ImageButton
14
15     override fun onCreate(savedInstanceState: Bundle?) {
16         super.onCreate(savedInstanceState)
17         setContentView(R.layout.activity_main)
18         start = findViewById(R.id.button_start)
19         open = Intent(packageContext, this@MainActivity, Conversion::class.java)
20         close = findViewById(R.id.button_deconnexion)
21         buttonClick()
22     }
23
24     private fun buttonClick() {
25         start.setOnClickListener { it: View!
26             startActivity(open)
27         }
28         close.setOnClickListener { it: View!
29             finish()
30         }
31     }
32
33 }
```


Tester cette première partie du code

Nous allons maintenant téléverser cette application pour la tester et la valider.

Si vous avez un smartphones avec l'OSMobile Android:

1. Déboguer votre smartphone -> aller dans vos paramètres/à propos/ cherchez le numéro de série
2. Appuyer 7 fois pour débloquent le mode développeur
3. Dans le mode développeur chercher et activer débogage USB
4. Maintenant dans cette petite fenêtre vous devriez voir le nom de votre smartphone



5. Appuyer sur Play pour téléverser l'application, et la retéléverser lorsque l'on applique des changements.
6. Si besoin ouvrir la fenêtre « Devices / Physical » pour associer votre smartphone à l'SDK Android.

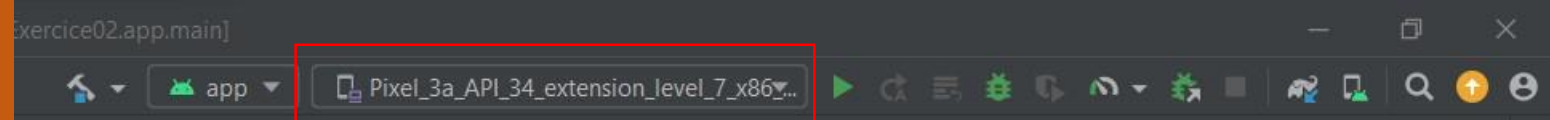
Si vous n'avez pas smartphones avec l'OSMobile Android:

Tester cette première partie du code 1/2

Nous allons maintenant téléverser cette application pour la tester et la valider.

Si vous avez un smartphones avec l'OSMobile Android:

1. Déboguer votre smartphone -> aller dans vos paramètres/à propos/ cherchez le numéro de série
2. Appuyer 7 fois pour débloquent le mode développeur
3. Dans le mode développeur chercher et activer débogage USB
4. Maintenant dans cette petite fenêtre vous devriez voir le nom de votre smartphone



5. Appuyer sur Play pour téléverser l'application, et la retéléverser lorsque l'on applique des changements.
6. La première fois cela prend un peu de temps pour téléverser l'application.
7. *Si besoin ouvrir la fenêtre « Device Manager / Physical » pour associer votre smartphone à l'SDK Android.*

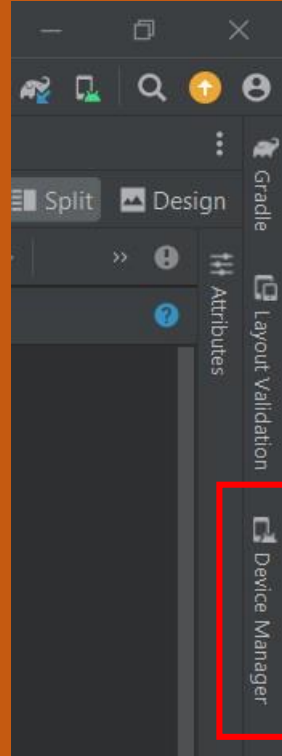
Si vous n'avez pas smartphones avec l'OSMobile Android:



Tester cette première partie du code 2/2

Si vous n'avez pas smartphones avec l'OSMobile Android:

1. Installons une machine virtuelle dans notre SDK Android Studio, ouvrir la fenêtre « Device Manager »
2. Sélectionner la fenêtre Virtual, puis cliquer sur « Create Device »
3. Il faut installer HAXM pour la machine virtuelle
4. À vous de choisir un model de smartphone pour simuler vos application dessus, elles sont différentes tailles d'écrans.
5. Cela prend un peu de temps lors du téléchargement de la machine virtuelle et de son ouverture.
6. Lorsque la machine virtuelle est allumé vous pouvez la sélectionner pour téléverser votre application dessus, cela prend un peu de temp.



Changer les couleurs des thématiques 1/2

Maintenant que nous pouvons tester l'application sur la machine virtuelle ou le smartphone nous pouvons tester les différentes interactions pour les valider.

- Votre application présente peut-être une barre de couleur noir avec le titre de votre application. Nous pouvons la supprimer. Nous pouvons aussi changer les couleurs des barres d'action et des boutons du system.
- Ces paramètres sont définis dans les deux fichiers: `values/themes/themes.xml` & `themes.xml(night)`
- Les deux fichiers correspondent aux deux modes: Clair/Sombre de votre smartphone et qui sont appliqués selon l'utilisateur du smartphone. Pour l'instant nous allons faire en sorte que les deux fichiers soient similaire, mais lors de vos futur projets garder en tête que le mode sombre à pour objectif d'améliorer le confort de la lecture et de rendre les contraste plus accessible.


Changer les couleurs des thématiques 2/2

- Pour supprimer la barre du titre dans les deux fichiers theme.xml vous devez vérifier que la valeur de l'attribut parent dans la balise style précise « NoActionBar » `parent="Theme.MaterialComponents.DayNight.NoActionBar">`
- Pour appliquer une couleur choisit dans la barre d'action vous devait ajouter un nouvel item avec la balise `<item name="android:statusBarColor">` à l'intérieur de ces balises nous pouvons définir une nouvelle couleur.
- Pour choisir une couleur vous pouvez écrire « @color/ » puis choisir la couleur noir, ensuite cliquer sur le petit carré de couleur noir pour le changer, dans la liste déroulante de couleur -> dans « android » vous avez un plus grand choix.
- Pour la barre des boutons du system le processus est similaire mais avec pour valeur dans l'attribut name -> « android:navigationBarColor »
- Vous pouvez aussi définir des couleurs dites primaires qui vont définir le style de votre application avec les balises suivantes:

```
<item name="colorPrimary">#4CAF50</item>
<item name="colorPrimaryVariant">@android:color/holo_green_dark</item>
<item name="colorOnPrimary">@color/black</item>
```
- Les couleurs peuvent être définit en hexadécimal avec #, il vous suffit sur une page web de récupérer la valeur.
- Vous pouvez ensuite réutilisé ces couleurs prédéfini, en appelant la valeur de leurs attributs name, ex: « ?attr/colorPrimary »

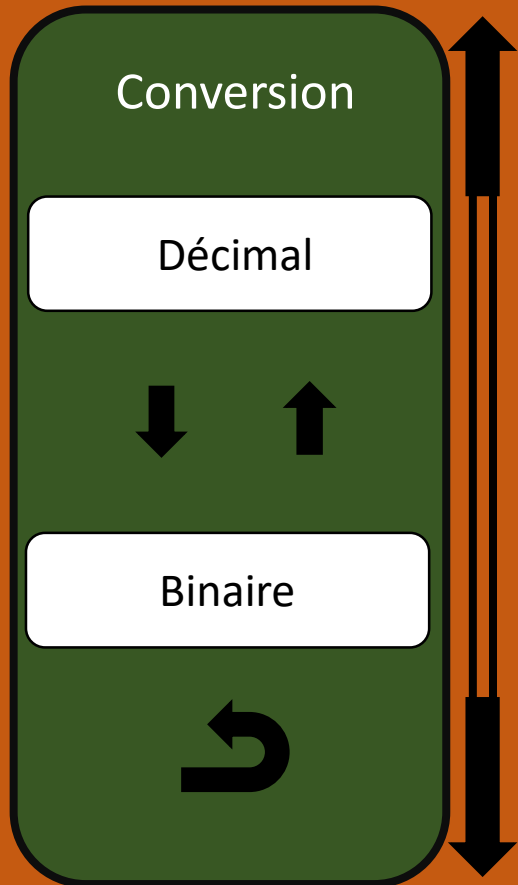
Changer l'icône de votre application

Avant de développer la suite de l'application nous pouvons tout de suite changer l'icône de l'application et son titre pour qu'ils soient plus explicite.

- Vous avez remarqué que l'icône de l'application est la suivante:
- 
- Télécharger deux images de votre choix, cohérentes avec la thématique des conversion de valeur. La première image correspondra au logo de votre application (format png avec arrière fond transparent), et la deuxième image à l'arrière fond. La deuxième image n'est pas obligatoire vous pouvez définir une couleur directement depuis Android Studio pour l'arrière fond.
 - Faites un clique-droit dans la fenêtre de l'arborescence puis sélectionné: New/Image Asset
 - Dans « Foreground » trouvez le chemin vers votre Logo et dans « Background » définir l'arrière plan
 - Faites en sorte que le logo soit bien centré dans le cercle, ensuite appuyer sur next
 - Dans la nouvelle fenêtre « Confirm Icon Path » dans la liste déroulante il y'a peut être présélectionné « debug », vous devez choisir « main » puisque nous souhaitons appliquer le Logo au projet.
 - Pour changer le titre afficher avec le Logo ouvrir le fichier: values/strings.xml. Dans la balise pour le nom de l'application choisir un titre cohérent avec le projet.
 - Maintenant si vous actualiser l'application vous devriez observer les changement avec un nouveau logo et un nouveau titre.

L'IHM de l'activité principale

Maintenant nous allons commencer la composition du Layout de la deuxième activité: activity_conversion.xml



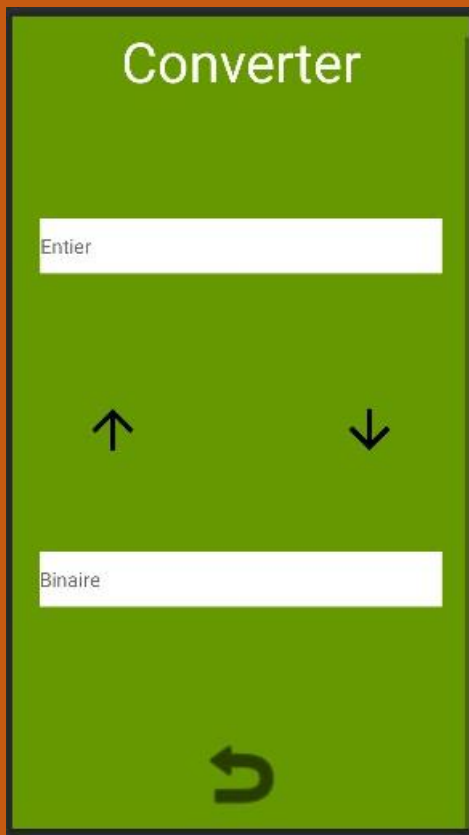
- Comme pour la première activité nous souhaitons scroller la page.
- Cette fois si nous avons deux Text de type EditText (décimal), vous pouvez les ajouter et les faire glisser depuis la fenêtre en mode Design.
- Il faut ajouter l'attribut suivant pour que les entrées saisissables soient uniquement des valeurs entières: `android:inputType="number"`
- IL y'a aussi deux boutons avec des flèches pour choisir le sens de la conversion et retourner dans la page d'accueil.
- Inspirez vous du Layout de la première page pour concevoir cette deuxième page, sachant que je vais vous donner les différents id de chaque composants puisque nous en auront besoin juste après.

Les ID à associé aux composants:

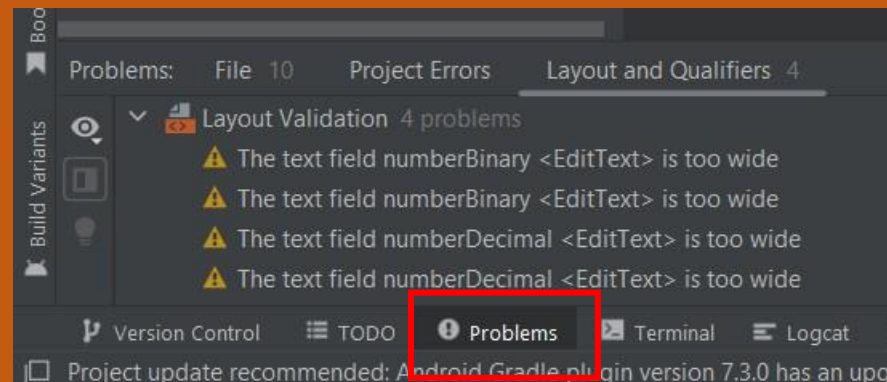
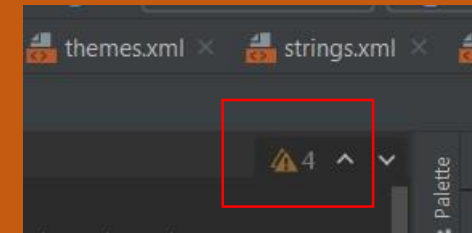
1. `android:id="@+id/title_activity_2"`
2. `android:id="@+id/numberDecimal"`
3. `android:id="@+id/button_towardDecimal"`
4. `android:id="@+id/button_towardBinary"`
5. `android:id="@+id/numberBinary"`
6. `android:id="@+id/button_back"`

L'IHM de l'activité principale

La correction de cette exercice vous sera envoyé à la fin du cours – vous devriez avoir une IHM qui ressemble à peu près à cela:



- Vous pouvez ajouter des images pour vos bouton dans: New/Image Vecteur
- Si vous avez des erreurs ignorables corriger les quand même pour améliorer les performances de votre application.



L'activité principale: le programme 1/10

Maintenant nous allons programmer l'activité principale avec l'IHM que vous veniez de concevoir. L'objectif est de vérifier si l'utilisateur à entrez une valeur entière dans la première zone de saisie et une valeur binaire dans la deuxième, puis de faire la conversion dans les deux sens.

- Dans un premier temps instancier les objets dans le fichier conversion.kt :
 1. Nous avons trois composants de type ImageButton et deux de type EditText
 2. En déclarant les variables « var » en privée « private » et « lateinit » pour les utiliser dans toute la classe seulement et les initialiser plus tard.
 3. Ensuite Initialisez les variables dans la fonction « onCreate() » en leurs attribuant leurs objet avec la ligne « findViewById(R.id.) »
 4. Voici le nom des variables à instancier: importer des libraires si besoin pour les types EditText et ImageButton

```
class Conversion : AppCompatActivity() {  
  
    private lateinit var decimal: EditText  
    private lateinit var binary: EditText  
    private lateinit var towardDecimal: ImageButton  
    private lateinit var towardBinary: ImageButton  
    private lateinit var towardBack: ImageButton
```

```
    override fun onCreate(savedInstanceState: Bundle?) {  
        super.onCreate(savedInstanceState)  
        setContentView(R.layout.activity_conversion)  
        decimal = findViewById(R.id.numberDecimal)  
        binary = findViewById(R.id.numberBinary)  
        towardDecimal = findViewById(R.id.button_towardDecimal)  
        towardBinary = findViewById(R.id.button_towardBinary)  
        towardBack = findViewById(R.id.button_back)
```

L'activité principale: le programme 2/10

Passons à la au développement des différentes fonctions

- Il y'a deux type d'interaction que l'utilisateur peut effectuer sur cette page de l'application, la première écrire dans les zones de saisie des textes, puisque nous avons précisé que le type du texte est une valeur décimal et un nombre entier, alors normalement le clavier de l'utilisateur ne devrait lui permettre de saisir que des nombres entier. Cette saisie des textes est automatique géré par l'application et l'outil Android Studio nous n'avions pas besoin de le préciser dans le programme.
- La deuxième interaction que peut effectuer l'utilisateur est le fait d'appuyer sur les trois boutons, cette fois si nous devons préciser que en cas de click une action se passe. Comme pour la précédente activité ajouter une fonction bouton ou vous allez associé aux trois boutons un Listener « `setOnClickListener` ».

L'activité principale: le programme 3/10

C'est maintenant que nous allons passer aux choses sérieuses, nous allons créer développer la méthode de conversion

Pour convertir une valeur décimale entière vers une valeur binaire il faut deux fonctions:

- La première fonction privée « `convertToBinary (){ ...}` » vérifie qu'une valeur est saisie dans la variable « décimal »

```
private fun convertToBinary()
```

- Pour vérifier que la variable n'est pas vide il faut utiliser une condition avec la méthode `isEmpty()`, le `.text` précise que l'on récupère le texte de type `Char()`

```
decimal.text.isEmpty()
```

- Si aucune valeur n'est saisie alors l'on affiche un message Pop-UP avec la ligne suivante: importer des libraires si besoin pour Toast

```
Toast.makeText(this, "Pas de valeur décimale à convertir!", Toast.LENGTH_SHORT).show()
```

- Toast permet d'afficher un message avec une certaine durée via la méthode `makeText`. Dans ces attributs nous devons définir dans quel contexte nous souhaitons l'utiliser « This » pour cette activité, la valeur (le texte) qu'il doit afficher et sa durée (`Toast.LENGTH_SHORT` || `LONG`)
- Une fois le Toast crée l'on précise que l'on souhaite l'afficher « `show()` »

L'activité principale: le programme 4/10

La deuxième fonction vérifie si le nombre saisi dans la zone de saisie des valeurs binaire est binaire et si elle est vide

- Créer une deuxième fonctions « `convertToDecimal() {...}` », puis comme pour la précédente nous avons besoin d'une première condition pour vérifier qu'un nombre est saisi, sinon on affiche un Toast.

```
private fun convertToDecimal(){
```

- Dans cette fonction nous avons besoin d'une deuxième condition si une valeur est saisie, car en effet nous pouvons écrire d'autres chiffres que « 0 » et « 1 », hors nous souhaitons un nombre binaire. Pour vérifier que le nombre est binaire nous allons récupérer les chiffres du `char()` précédemment convertit en type `String()` un par un. D'abord on convertit encore ce `String()` en une liste de `String()`. Puis l'on vérifie qu'il s'agit soit des chiffres 0 et 1 en convertissant encore chaque éléments de la liste en un `Int()`, ensuite on les ajoute dans une liste de `Int`, cette liste correspond à notre nombre binaire c'est-à-dire les enchaînement de 0 et de 1.
- Voici les variables pour la fonction « `convertToDecimal() { }` » ainsi qu'un algorithme pour vérifier et créer la liste. Cette algorithme peut être modifier et améliorer il s'agit d'une façon de faire parmi temps d'autres.
- Pour les conversions on utilise: `toString()`, `toInt()`, `toList()`



L'activité principale: le programme 5/10

La fonction `conversionInverse()` est la prochaine que nous allons créer.

```
50     private fun convertToDecimal(){//Vérifier valeur binaire
51         val laList = binary.text.toString().toList()
52         val newList = arrayListOf<Int>()
53         var check = true
54         var number : Int
55
56         if (binary.text.isEmpty()) {
57             Toast.makeText( context: this, text: "Pas de valeur binaire à convertir!", Toast.LENGTH_SHORT).show()
58         }
59         else{
60             for (char in laList){
61                 number = char.toString().toInt()
62                 if (number == 0 || number == 1) {
63                     newList.add(number)
64                 }
65                 else{
66                     check = false
67                 }
68             }
69             if(check){
70                 decimal.setText(conversionInverse(newList))
71             }
72             else{
73                 Toast.makeText( context: this, text: "Pas de valeur binaire à convertir!", Toast.LENGTH_SHORT).show()
74             }
75         }
76     }
77 }
```

L'activité principale: le programme 6/10

```
40  
41 private fun convertToBinary(){//Vérifier valeur décimal - entière  
42     if (decimal.text.isEmpty()) {  
43         Toast.makeText( context: this, text: "Pas de valeur décimale à convertir!", Toast.LENGTH_SHORT).show()  
44     }  
45     else{  
46         binary.setText(euclideanConversion(decimal.text.toString()))  
47     }  
48 }
```

La fonction euclideanConversion() est la prochaine que nous allons aussi créer.

L'activité principale: le programme 7/10

Maintenant nous pouvons vérifier que les valeurs sont entrées et qu'il s'agit bel et bien de valeur binaire pour la conversion vers le décimale. Une fois ces vérifications terminées nous pouvons convertir les valeurs!

- Pour convertir les valeurs nous allons créer deux nouvelles fonctions de conversions qui doivent retourner un `String ()` pour être affichées directement dans la zone des textes. Ces fonctions feront les différentes conversions en `Int()` et en `String()` dans leurs programmes internes, parce qu'elles récupéreront aussi en attributs des variables de type `String()`
- Commençons par convertir les valeurs décimales en binaire, nous allons utiliser la méthode euclidienne qui consiste à vérifier si la valeur est pair ou impair avec « %2 ». Si la valeur est impaire alors on soustrait 1 et l'on écrit dans une liste la valeur "1", sinon l'on écrit "0", ensuite l'on divise la valeur par deux. On recommence ce procédé jusqu'à ce qu'il reste la valeur 0 à diviser grâce à une boucle « `while()` ».
- La liste binaire de type `String()` contient des « 0 » et « 1 » on l'inverse pour avoir le bon sens de lecture avec la méthode « `.reversed()` ».
- Attention : il y a une limite aux valeurs décimales que l'on peut saisir et à la valeur binaire que l'on peut convertir. On peut utiliser le type « `toLong` » sur la liste de `String()` pour agrandir les valeurs convertissables. Dans ce cas les valeurs des variables se voient ajoutées un « L » pour préciser leurs types.
- Créez une fonction « `euclideanConversion() {...}` » qui reçoit une variable de type `String` et qui est appliquée depuis la fonction « `convertToBinary() {...}` »

```
private fun euclideanConversion(decimalValue: String): String {
```

L'activité principale: le programme 8/10

Voici une version de la fonction: essayer de la développer sans cette aide avec la logique expliqué précédemment.

```
78     private fun euclideanConversion(decimalValue: String): String {
79         var list = ""
80         var value = decimalValue.toLong()
81         while(value != 0L){
82             if(value%2 == 0L){
83                 value /= 2
84                 list += "0"
85             }
86             else{
87                 value -= 1
88                 value /= 2
89                 list += "1"
90             }
91         }
92         return list.reversed()
93     }
```

L'activité principale: le programme 9/10

Nous avons bientôt terminé, la prochaine fonction est la dernière elle va convertir les nombres binaires en décimales.

- Créez la dernière fonction « `conversionInverse` » qui reçoit en attribut une variable une liste d'entier `Int()`, appelé depuis la fonction « `convertToDecimal () {...}` »

```
private fun conversionInverse(list: List<Int>): String {
```

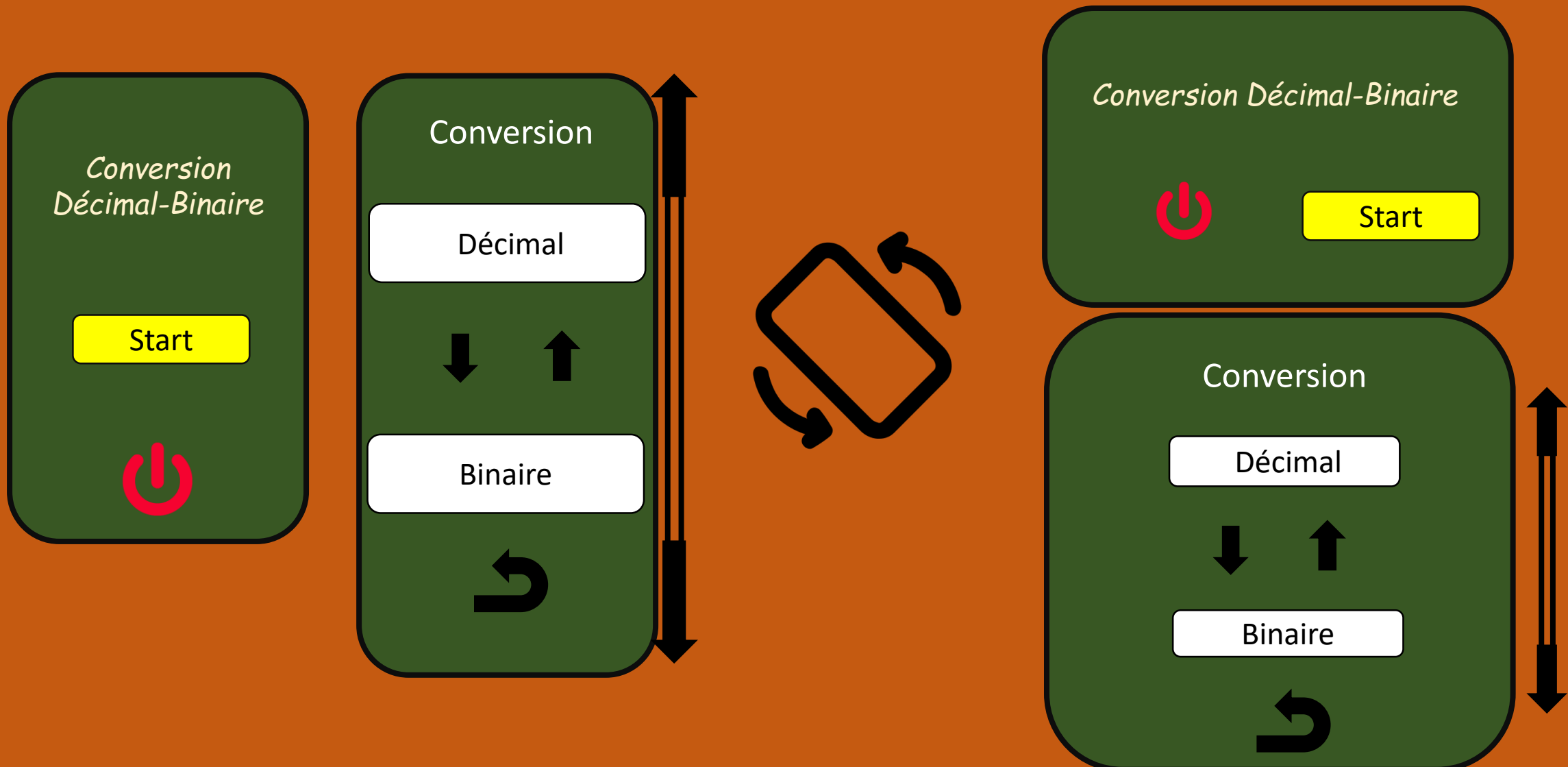
- Cette fonction reçoit une liste d'entier qui correspond à un nombre binaire depuis « `convertToDecimal () {...}` ». Pour la convertir en un nombre décimal pour chaque valeur (0 ou 1) et sa position dans la liste nous allons ajouter à une nouvelle variable de type `Double()` la valeur de 2 à la puissance (la position du chiffre binaire dans la liste). Cela grâce à un compteur initialiser à la taille de la liste et auquel on soustrait 1 (l'itération est à l'envers dû à sa lecture), et à une itération de la taille de la liste.
- Ensuite on retourne une valeur décimal convertit sous la forme d'un `String()` pour pouvoir l'afficher dans l'EditText de la saisie des décimales.
- Tentez d'imaginer et de produire l'algorithme, la correction est dans la diapositives suivante :



L'activité principale: le programme 10/10

```
95     private fun conversionInverse(list: List<Int>): String {  
96         var value = 0.0  
97         var compteur = list.size.toDouble()  
98         for(number in list){  
99             compteur -= 1  
100            value += number* 2.0.pow(compteur)  
101        }  
102        return value.toLong().toString()  
103    }
```


IHM & Design (La rotation)

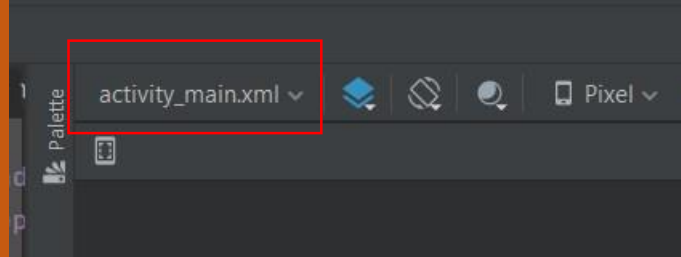


IHM & Design (La rotation)

Maintenant que le programme est terminée vous pouvez le tester sur votre smartphone.

Nous allons améliorer le responsive Design, les utilisateurs peuvent activer la rotation de leurs écrans, sachant que l'on peut scroller dans nos deux pages d'activités tous les composants sont accessibles. Si en plus les dimensions de vos composants sont responsives, c'est-à-dire que vous avait fais des combinaisons de valeurs « match_parent » et « wrap_content », alors la rotation ne devrait pas poser de problème.

Cependant vous pouvez réagencer les éléments selon la rotation de l'écran si vous le souhaitez, pour cela vous devez créer un nouveau layout(land) qui est associé au mode paysage:



Nouvelle exercice

Si vous avez terminé l'exercice 02, voici un exercice conçu par les développeurs de chez Android que j'ai testé et jugé intéressant pour comprendre le cycle de vie d'une application. Par « cycle de vie » j'entend les moments où vos applications se créent, s'ouvrent, commencent, se mettent en pause, s'arrêtent et sont détruites. Ces différentes étapes doivent être prises en compte pour y inclure des interactions selon vos besoins, par exemple si l'on ferme une application en ayant oublié de sauvegarder des données etc.

Voici le lien de l'exercice: <https://developer.android.com/codelabs/basic-android-kotlin-compose-activity-lifecycle?hl=fr#6>