Université Paris Cité **Programmation des composants mobiles**
M1 **Année 2022-2023**

TP n° 2

Premières activités

Références principales :

- les cours de Wieslaw Zielonka sur Moodle.
- Le site officiel d'Android
  - la page `https://developer.android.com/docs` ;
  - la documentation de l'API :
    urlhttps ://developer.android.com/reference/packages
  - guides sur des thèmes précis : `https://developer.android.com/guide`
- la page de Kotlin `https://kotlinlang.org/docs/home.html`

1 Un peu de configuration

Vous devez maintenant apprendre à utiliser Android Studio pour faire des applications Android, et les lancer, soit sur votre téléphone, soit sur l'émulateur.

Hello world

Commencez un projet : Choisissez "Start a new Android Studio project". À l'écran suivant, choisissez "Empty Activity", et appuyez "Next". Mettez le nom d'application "TP0", un nom de package sympa (par exemple `fr.uparis.dupont`), le langage Kotlin, appuyez sur "Finish" et patientez. L'écran de travail d'Android Studio doit s'ouvrir durant ce processus.

Connectez votre téléphone (optionnel) C'est souvent plus rapide et plus amusant d'exécuter vos programmes sur un vrai téléphone (ou tablette) sous Android. C'est très facile

`https://developer.android.com/studio/debug/dev-options`

et un peu magique :

- Sur votre téléphone allez à **Paramètres** → **A propos** → **Numéro de version**, que vous taperez 7 fois. Le message "vous êtes sous mode développeur" apparaîtra.
- Allez maintenant dans la nouvelle rubrique **Paramètres** → **Options pour développeurs** et activez le Débogage USB.
- Connectez le téléphone à la machine par cable USB.

Si vous n'avez pas de téléphone disponible configurez un émulateur – A faire une seule fois Dans le menu choisissez `Tools->Device Manager`. Dans la fenêtre qui s'ouvre cliquez sur "create virtual device", Sélectionnez un type de téléphone, par exemple Nexus 4 (plus petit) , ou Nexus 6. Choisissez une version d'Android à y mettre (par exemple R), puis "Next" et enfin "Finish". Il y a un peu d'attente avant d'avoir une fenêtre "Your virtual devices", fermez cette fenêtre (vous pouvez lancer l'émulateur).

Lancez l'application (et si nécessaire l'émulateur) : Cliquez sur la flèche verte. Choisissez votre téléphone ou l'émulateur. Patientez, puis testez l'application. **L'émulateur met un certain temps à charger, ne l'éteignez surtout pas** jusqu'à la fin du TP, vous pourrez lancer vos applications dessus rapidement.

Bien utiliser votre Android Studio

Il s'agit d'un environnement de développement puissant, voici quelques conseils d'utilisation.

— restez connectés à l'Internet
— soyez patients, certaines actions prennent du temps.
— lisez les messages d'erreur dans l'onglet Logcat (en bas) ! Vous pouvez filtrer l'information utile dans cet onglet
— créez vos classes/activités/fragments etc avec le menu New
— soyez attentif au suggestions du Studio, souvent il vous génère du code utile. Utilisez le menu Code pour générer du code.
— vous pouvez activer les imports automatiques dans `Preferences->Editor->General->Auto-Import->Kotlin`
— l'onglet `Problems` (en bas) permettra d'améliorer votre style et trouver des bugs cachés.

2 Ce qu'on veut programmer aujourd'hui

Dans ce TP vous allez programmer votre première vraie application Android. Il s'agit d'une application avec trois activités pour tester les connaissances de maths et du français du joueur, qui ressemble aux trois captures d'écran :

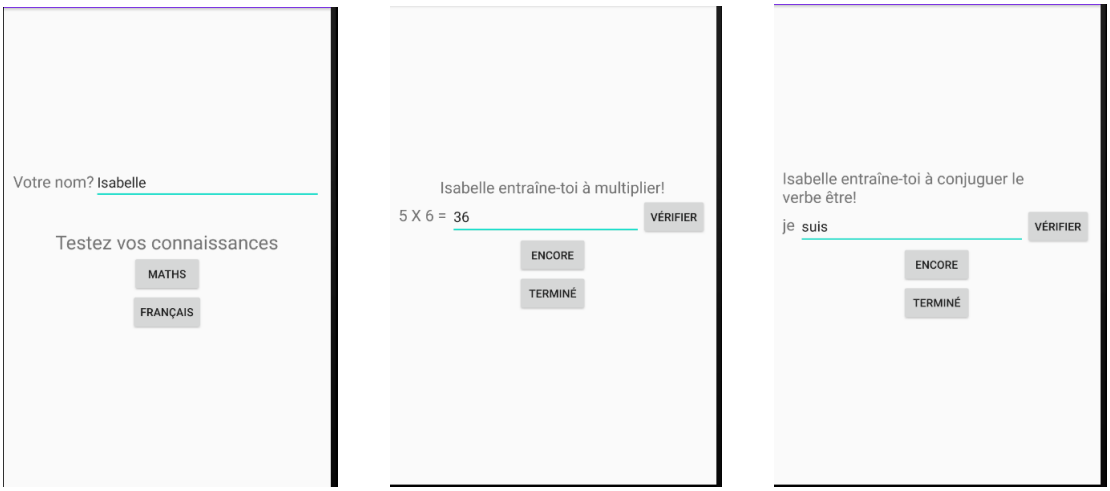

FIGURE 1 – De gauche à droite : écran d'accueil ; quiz de multiplication ; quiz de français

3 On crée le projet et programme l'écran d'accueil

Dans Android Studio, créez l'application TP2 en commençant par "Empty Activity", laissez Android Studio générer tous les fichiers

3.1 Mise en page (layout)

Ouvrez le fichier contenu dans `res/layout`, cliquez sur le bouton "Code" en haut à gauche pour voir le fichier XML. Effacez-le et copiez-collez à la place le fichier `activity_main.xml` que vous récupérerez sur Moodle. (Ouvrez le fichier de Moodle sous `emacs` ou `gedit`, pas avec un navigateur web qui n'afficherait pas le code en entier.) Pour que le fichier s'indente correctement, utilisez "Code/Reformat code" (utile aussi avec le code Kotlin). Examinez le texte XML, vérifiez que vous le comprenez.

Copiez aussi le contenu de `strings.xml` dans le fichier de même nom dans `res/values`

Lancez l'application sur l'émulateur (attention, ne le fermez pas jusqu'à la fin du TP) ou le téléphone. Essayez d'appuyer sur un bouton.

3.2 On gère les boutons

Récupérer un élément d'IG Pour récupérer un `EditText` on utilise toujours la méthode `findViewById(R.id.editNom)` qui doit être appelé après `setContentView(...)`. Plusieurs techniques d'utilisation sont possible.

— en variable locale immuable dans `onCreate` (inutilisable par ailleurs) :

```
val edit : EditText=  findfindViewById(R.id.editNom)
```

— en attribut de classe non-initialisé

```
lateinit var edit: EditText
```

et puis dans `onCreate` :

```
edit= findfindViewById(R.id.editNom)
```

— ou en attribut de classe immuable avec l'initialisation paresseuse

```
val edit: EditText by lazy { findViewById(R.id.editNom) as EditText }
```

Gérer un élément d'IG Des techniques pour gérer un bouton

— Déclarer `onClick` dans le xml (dans notre cas `faireMath`). Programmer le gestionnaire `fun faireMath(view: android.view.View)`

— récupérer le bouton et y associer un listener en utilisant une lambda-expression.

```
val bMath:Button=findViewById(R.id.faireMath)
bMath.setOnClickListener{ faire qch}
```

— On peut aussi faire un seul listener qui gère plusieurs boutons (voir le cours).

1. Dans le code Kotlin de `MainActivity` récupérez un objet pour `EditText` et éventuellement les deux boutons.
2. Faites un gestionnaire provisoire du bouton Maths qui affiche un Toast "Test de math". Pareil pour l'autre bouton. Testez.

Afficher un Toast Ça se fait avec

```
Toast.makeText(this, "tralala",Toast.LENGTH_SHORT).show()
```

3. Améliorez vos gestionnaires de boutons pour que le `Toast` affiche aussi le nom de l'utilisateur, p.ex. "Test de math pour Eugene".
   Vous pouvez lire le nom saisi dans l'EditText comme ceci `val nom=edit.getText().toString()`, ou mieux comme cela `val nom=edit.text.toString()` [1] Pensez au cas où l'utilisateur n'a pas saisi son nom.

1. la propriété `text` est ici en fait un raccourci pour l'accesseur `getText()`. On peut faire pareil en écriture.

4 Seconde activité : on programme le quiz de maths

1. Demandez à Android Studio d'ajouter à votre projet une nouvelle activité (pour le quiz de math) et faites son interface graphique (copiez le contenu du fichier `activity_multiply.xml` fourni sur Moodle).
2. Maintenant les choses intéressantes commencent :
   (a) On veut faire en sorte que l'appui du bouton "Maths" sur l'écran d'accueil lance une autre activité — le quiz. Ce n'est pas direct, on doit créer une intention (explicite), et demander au système à lancer l'application, par exemple comme ça :

   ```
   val iii = Intent(this, MultiplyActivity::class.java)
   startActivity(iii)
   ```

   Remarques :
   — La classe `Intent` est une représentation abstraite d'une opération à effectuer. Ici on lui donne le nom de l'activité à exécuter.
   — la première ligne est un appel à un constructeur : il n'y a pas de `new` en Kotlin.
   — `startActivity` lance l'exécution.
   Testez que ça marche — c-à-d que l'activité de quiz, pas encore fonctionnelle, est lancée dès qu'on appuie le bouton "Maths".

   (b) L'activité principale veut transmettre le nom du joueur (qui est saisi dans le widget `EditText`) à l'activité de quiz. Pour ceci, avant le `StartActivity`, on met le nom dans l'`Intent`, avec ceci : `iii.putExtra("username",nom)`. En fait, grâce à la méthode `putExtra`, on peut ajouter à un `Intent` autant de couples clé-valeur que l'on veut, avec un grand nombre de types disponibles pour la valeur.
   Pour récupérer le nom, l'activité de quiz (dans sa méthode `onCreate`) peut procéder comme suit :

   ```
   val iii = intent //appel implicite a getIntent() de Activity
   val nom = iii.getStringExtra("username")
   ```

   Remarque :
   — `getIntent()` (ou tout simplement `intent` en Kotlin) retourne le Intent qui a démarré cette activité. Ici c'est donc l'intent avec lequel on a appelé l'activité de multiplication. Il est donc normal qu'il contienne le nom.
   Programmez cette communication et faites en sorte que le quiz utilise le nom du joueur pour le saluer (dans le `TextView`).

   (c) Programmez le quiz qui génère aléatoirement les deux nombres à multiplier, reçoit la réponse du joueur et affiche un `Toast` avec le message “Bravo” ou “Erreur”. De plus, on veut savoir, à combien de questions on a répondu et combien de réponses étaient justes. Ça nous permettra de calculer le pourcentage des réponses justes ensuite.
   Indications :
   — pour engendrer un entier aléatoire entre 1 et 9, faites : `Random.nextInt(1, 9)`
   — vous avez intérêt à déclarer comme propriétés les deux entiers engendrés, et à faire une fonction `genereQuestion()` qui génère ces nombres et affiche la question. (Vous allez la réutiliser.)

   (d) Souvent une activité A souhaite lancer une activité B pour qu'elle fasse son travail et à la fin renvoie un résultat. Ceci est possible à travers des `Intent`. Plus précisément, l'activité A prépare un `Intent intAB` pour B, et un objet `launcher` de type `ActivityResultLauncher<Intent>`, puis lance l'application B avec le `launcher.launch(intAB)`.

Fabriquer un launcher et son callback Comme ceci (voir aussi Cours 3)

```
val launcher: ActivityResultLauncher<Intent> =
    registerForActivityResult (ActivityResultContracts.StartActivityForResult())
        { //listener
      if (it.resultCode == Activity.RESULT_OK){
         //recuperer le resultat dans it?.data?
         //faire le traitement souhaite
        }
    }
```

Quand l'activité B termine son travail, elle crée un nouvel `Intent intBA`, met dedans toute information utile (ici, le pourcentage de réponses justes un entier), et puis appelle

```
setResult(RESULT_OK,intBA)
finish()
```

(Le code de résultat peut aussi être `RESULT_CANCELED` ou un autre entier.) Finalement, l'activité A récupère le résultat et le traite à travers la méthode callback déclarée dans le launcher.

Remarque. Pour appeler une méthode, sur un nullable, ici `data`, il est possible de faire :

```
if (data != null) {
  data.uneMethode() //sans ‘‘?’’ car on a fait la verification,
}
```

Le fait que le test `(data != null)` est fait est vérifié à la compilation.

Ajoutez ce retour de résultat à votre application. Par exemple, quand le joueur a fini, l'activité du quiz ferme, et l'écran d'accueil affiche quelque chose du genre “Vous avez 75% de réponses justes".

5 S'il vous reste du temps

Ajoutez l'activité pour le quiz du français : il s'agit d'interroger sur la conjugaison du verbe “être” au présent.

Indications utiles :

— les questions et les réponses peuvent être mis dans des tableaux

```
val sujets = arrayOf("je", "tu", "elle/il", "nous", "vous", "ils/elles")
```

— vous pouvez ignorer l'accent sur “vous êtes”.

— si vous avez programmé de manière propre le quiz précédent, les changements sont assez peu nombreux.

6 Il vous reste encore du temps ?

Vérifiez et corrigez le comportement de votre programme lorsqu'on tourne l'écran, comme c'est expliqué en Cours 2 (avec l'objet `savedInstanceState`)