

TP 1

TP1 : hello world

Ce TP est un TP d'introduction à Android au cours duquel nous aurons l'occasion de mettre en œuvre une activité simple de type *Hello World* sur laquelle nous allons (beaucoup) cliquer.

Installation d'Android Studio


On suppose que l'on a déjà installé Android Studio sur notre machine de travail. La sous-section qui suit concerne uniquement les utilisateurs d'une machine de salle de TP.

Utilisation d'Android Studio dans une salle de TP

Si vous utilisez un PC d'une salle de TP, vous avez deux possibilités :

- Utiliser la version installée par le CRI accessible avec la commande depuis un terminal
`android-studio`
- Utiliser une version plus récente que le CRI disponible avec le chemin (ce qui est conseillé)

`/home/ens/chilowi/export/andstu`

 Dans tous les cas, ne vous avisez pas d'installer vous-même Android Studio sur votre compte utilisateur : vous dépasseriez assurément votre quota disque.

Lors du premier lancement, acceptez l'import de la configuration précédente (1er dialogue). Si Android Studio vous invite à télécharger le SDK, la configuration par défaut a mal été importée ; supprimez le répertoire dont le nom commence par `.AndroidStudio` à la racine de votre compte utilisateur et relancez Android Studio.

Création d'un nouveau projet

Pour créer un projet, on utilise le dialogue File>New>New project. L'assistant nous interroge :

- sur le nom de l'application et le paquetage de l'application (qui doit être unique en vue de la publication sur un store)
- l'emplacement du projet sur le disque
- le langage par défaut utilisé pour le projet : Kotlin ou Java (que l'on pourra changer par la suite)

Ensuite nous sommes questionnés sur le niveau minimum d'API que nous souhaitons supporter. Ce choix est crucial ; le `minSdk` est écrit dans le fichier `build.gradle` et peut être changé par la suite. Une application refusera de s'installer sur une machine disposant d'une version d'Android inférieure au *minSdk* indiqué. La meilleure stratégie consiste à choisir le *minSdk* le plus faible possible pour assurer le maximum de compatibilité et à augmenter si besoin sa valeur si des fonctionnalités essentielles plus récentes nous font défaut pour le développement de notre application.

Notons qu'utiliser un niveau minimal faible ne nous interdit pas d'utiliser une fonctionnalité d'un niveau plus élevé. Par exemple le Bluetooth Low Energy (BLE) n'est disponible qu'à partir de la version d'API 18. Si dans le fichier `build.gradle` le `minSdkVersion` est inférieur, il faut tester systématiquement le niveau d'API avant d'utiliser cette fonctionnalité au risque d'être confronté à une exception :

```
if (android.os.Build.VERSION.SDK_INT >= Build.VERSION_CODES.JELLY_BEAN_MR2)
{
    // we can execute the BLE code here...
}
```

Il est possible aussi de développer des applications pour des éditions spécifiques d'Android pour les montres, TV ou voitures. Il existe même une version à destination des objets connectés : Android Things (mais le développement de cette version a été malheureusement arrêté !). Cette version se concentre sur le support de différents bus de communication et supporte des interfaces utilisateur sans écran.

Continuons dans l'assistant. Il nous est demandé maintenant de créer une activité. Il s'agit d'un composant de base d'une application Android : un écran permettant

d'interagir avec l'utilisateur. Ne choisissons pour l'instant aucune activité (nous en ajouterons une plus tard).

Terminons l'assistant : nous disposons d'une application Android mais totalement vide.

Sur la gauche, Android Studio nous propose une vue avec les fichiers du projet. Trois types de fichiers sont à distinguer :

- les sources Java de l'application
- les ressources qui sont des fichiers XML ou binaires contenant des données utiles pour l'application (définition des écrans, chaînes de caractères internationalisées, images, sons...)
- le fichier où nous déclarons des informations concernant l'application à destination du système : son nom, les écrans (activités) qu'elle contient, ses éventuels autres composants (, ,), les permissions que requiert l'application pour s'exécuter...

AndroidManifest.xml

Service

BroadcastReceiver

ContentProvider

- les fichiers de construction Gradle contenant des déclaration destinées au compilateur (numéro minimum d'API, numéro d'API ciblé, dépendances nécessaires pour la compilation)

Création de la 1ère activité

Créons maintenant une première activité nommée HelloWorldActivity. Le projet sélectionné sur la vue de gauche (app), avec File>New>Activity>Empty Activity, nous créons une activité vide qui utilisera l'API graphique de base. Indiquons son nom dans le dialogue. On constate qu'un layout (définition XML des composants graphiques de l'activité) va être créé (gardons le nom suggéré).

Nous pouvons indiquer qu'il s'agit d'une Launcher activity. Cela signifie que l'activité apparaîtra dans le launcher (écran principal). Si cette option n'est pas demandée, l'utilisateur ne pourra lancer l'activité qu'à partir d'une autre activité et non directement du launcher. Validons le dialogue. Deux nouveaux fichiers ont été créés :

- Un fichier contenant la classe héritant de (ou)

HelloWorldActivity.java

Activity

AppCompatActivity

- Un fichier de ressource dans pour définir en XML le contenu de l'écran (composants graphiques)

res/layout

☞ Si vous avez oublié de cocher *Launcher Activity*, vous pouvez plus tard rajouter dans le manifeste un *intent filter* dans la balise *Activity* afin de rendre accessible l'activité depuis le launcher :

```
<activity ...>
  <intent-filter>
    <action android:name="android.intent.action.MAIN" />
    <category android:name="android.intent.category.LAUNCHER" />
  </intent-filter>
</activity>
```

Première version du layout

Ouvrez le fichier XML de layout correspondant à l'activité créée. Vous pouvez alors soit éditer graphiquement le layout en ajoutant, supprimant ou modifiant des composants graphiques ; soit modifier le contenu XML à la main (onglet XML en bas). Par défaut le layout parent est un *ConstraintLayout* dans lequel nous ajoutons des composants avec l'usage de contraintes de placement par rapport au parent et/ou aux composants frères.

Ajoutez un composant *TextView* pour afficher du texte. On ajoutera des contraintes de placement sur le haut du layout parent ainsi que sur *start* et *end*. On fixera la largeur à 0dp : le composant se place en haut de l'écran sur toute la largeur. On change le texte (propriété *text*) pour afficher *Hello World* ; on centre le texte (propriété *gravity*).

Ajoutez ensuite un bouton prenant toute la largeur de l'écran en bas du layout avec l'intitulé *Quit*.

Lancez la compilation et l'exécution de l'application avec cette activité avec Run>Run app. On vous demandera sur quelle machine vous souhaitez exécuter l'application. Il faut soit employer une machine physique, soit une image sur l'émulateur.

Test de l'application

Test sur une machine physique

L'emploi d'une machine physique nécessite préalablement l'activation du mode développeur sur celle-ci. Rendez-vous dans les paramètres ; si une entrée de menu *Developer options* n'est pas présente, il faut l'activer. Pour cela, nous allons dans *About* et appuyons de multiples fois sur le *Build number* (normalement 10 fois devraient suffire). Dans le menu des options de développement il faut ensuite activer le débogage Android ADB via USB.

Normalement l'appareil devrait être reconnu dès sa première connexion en USB. N'oubliez pas dans le tiroir de notification d'activer le mode transfert de fichier pour initier la communication (par défaut un téléphone connecté en USB ne fait que se recharger sans activation de la connexion de données). Un dialogue devrait demander sur l'appareil une autorisation d'accès qu'il faut accepter. Si cela ne fonctionne pas, il est possible qu'un pilote spécifique puisse être nécessaire sous Windows. Sous Linux, les problèmes sont généralement dûs à des droits non disponibles pour l'utilisateur courant ; il faut alors configurer *udev* pour attribuer automatiquement des droits pour l'utilisateur lors de la connexion de l'appareil (cette page de la documentation Android explique la marche à suivre). Si vous utilisez une machine de la salle et que votre appareil Android n'est pas reconnu, vous ne pouvez malheureusement pas changer les règles *udev* vous-même (il faut être administrateur) : seul le CRI pourra le réaliser.

Test avec une image sur l'émulateur

Dans Tools>Android>AVD manager nous pouvons accéder au gestionnaire d'appareils Android virtuels. Demandez la création d'un AVD, choisissez le profil matériel que vous souhaitez : il est préférable de ne pas être trop gourmand en résolution (inférieure à celle de votre écran). Choisissez l'image système à utiliser : employez de préférence une image x86 64 bits pour plus de performance (émuler un processeur ARM sur un

processeur hôte x86 est moins performant). Vous pouvez aussi configurer en détails les options de configuration : on peut choisir d'équiper l'appareil de caméras (virtuelles ou alors connectées à la webcam de l'ordinateur hôte), on peut simuler une bande passante et une latence de différentes technologies réseau et cellulaire, on peut employer plusieurs coeurs du CPU, configurer la quantité de RAM (à modérer en fonction de la RAM de la machine hôte), la taille du tas de la VM, le stockage interne et simuler une carte SD insérée...

On peut ensuite démarrer l'AVD créé. Il est préférable de ne pas fermer intempestivement l'AVD si nous en avons encore besoin (un démarrage étant généralement long).

⚠ Si vous utilisez un ordinateur de la salle de TP, voici quelques conseils :

- Il est requis de configurer l'accélération graphique en mode *software* afin que l'émulateur puisse se lancer.
- N'essayez pas de télécharger une image : choisissez un image disponible préinstallée par le CRI (image sans lien *download*).
- Soyez raisonnable pour la résolution de la machine (qui devrait être inférieure à celle du moniteur).
- Dans les options avancées, configurez l'image en mode *Cold boot* plutôt que *Quick boot* pour économiser votre précieux quota disque ; le démarrage sera plus lent car le contenu de la RAM ne sera pas sauvegardé sur le disque à la fermeture de la machine virtuelle.

Quittons l'activité

Quittons pour de faux...

Rajoutons maintenant une action sur le bouton Quit que nous avons créé.

Pour l'instant nous allons afficher un Toast : il s'agit d'un petit message surgissant sur l'écran pendant quelques secondes. On affichera le message Let's quit the activity. Voici le code pour afficher un tel Toast en Java :

```
Toast t = Toast.makeText(this, "Let's quit the activity", Toast.LENGTH_SHORT);  
t.show();
```

this désigne un contexte (classe Context dont hérite Activity) : il s'agit généralement de l'activité courante à partir de laquelle on demande l'affichage. Beaucoup d'appels de l'API nécessitent de passer en premier argument un tel contexte. Il ne faut pas oublier d'appeler show() pour afficher le Toast.

Mais où placer ce code ? Nous avons deux possibilités :

- Soit créer une méthode de signature dans la classe de notre activité et y mettre le code d'affichage du . Il faut ensuite relier cette méthode au bouton : on se rendra dans le layout XML et on ajoutera en propriété du bouton le nom de la méthode.

```
public void onQuitClicked(View v)
```

```
Toast
```

```
onClick
```

- Soit ajouter programmatiquement un listener sur le bouton. On le fera dans la méthode , méthode où l'on initialise les composants graphiques de l'activité ainsi que les divers champs dont nous avons besoin. A la fin de cette méthode, on rajoutera le code suivant :

```
onCreate(...)
```

```
Button b = (Button)findViewById(R.id.quitButton); // pour trouver le bouton à partir de son ID défini dans le layout XML
b.setOnClickListener(button -> { ...}); // expression lambda appelée lorsque l'on clique sur le bouton
```

On constate que l'on peut obtenir une référence vers un composant graphique défini dans le layout XML avec la méthode findViewById en passant en paramètre l'id du bouton contenu dans la classe R.java. Cette classe est automatiquement générée lors de la compilation du projet (en utilisant aapt) : elle référence toutes les ressources du projet en leur attribuant un identifiant entier. Pour que cela fonctionne, il faut attribuer un nom d'identifiant aux composants graphiques : ici on renseignera la propriété id du Button.

Au moindre souci relatif à une ressource de l'application (généralement un problème syntaxique dans un fichier XML), aapt ne peut plus générer le fichier R.java : les

identifiants ne sont plus reconnus.

Compilation plus rapide en RAM plutôt que sur le disque

Nous pouvons aussi indiquer un chemin spécifique pour la compilation du projet dans le fichier build.gradle général du projet. Par exemple, on pourra demander la compilation des projets dans /tmp (on suppose que l'on est sur un OS Unix-like) plutôt que dans le répertoire du projet :

```
allprojects {  
    buildDir = "/tmp/${rootProject.name}/${project.name}"  
}
```

Ceci est très utile si votre répertoire utilisateur est sur un disque lent (par exemple avec un accès NFS comme sur les machines des salles) et que le répertoire /tmp est monté en RAM (beaucoup plus rapide).

Quittons pour de vrai...

Pour quitter réellement l'activité, nous pouvons appeler la méthode finish() après avoir affiché le toast. L'activité se ferme alors et nous retournons normalement sur l'écran d'accueil si nous avons lancé l'activité depuis le launcher.

Internationalisation de l'activité

Pour l'instant notre activité est monolingue avec les chaînes de caractères codées en dur aussi bien dans le fichier XML de layout (propriétés text) que dans le code java (texte affiché sur le toast). Nous voulons désormais adapter notre application à d'autres langues.

Pour cela nous allons d'abord définir les chaînes dans le fichier res/values/strings.xml. On remplacera ensuite les versions en dur dans le XML par l'identifiant de la chaîne @string/name_of_the_string. De la même façon, il est possible d'obtenir programmatiquement en Java la chaîne correspondante à un identifiant avec getResources().getString(R.string.name_of_the_string) (pratique pour le toast).

Si une chaîne en dur est présente dans le source ou un fichier XML de layout ; une ampoule s'affichant en marge permet automatiquement d'extraire la chaîne en dur et de

la déplacer dans le fichier `res/values/strings.xml` : c'est ce que vous ferez pour les chaînes *Hello World* et *Quit*.

Maintenant que les chaînes ne sont plus en dur, adaptons notre programme en français. Vous pouvez pour cela créer un nouveau fichier `res/values-fr/strings.xml` automatiquement grâce à l'éditeur de traductions (bannière s'affichant lorsque vous éditez le fichier `strings.xml`). Rajoutez la locale *français* (fr) : une nouvelle colonne apparaît dans l'éditeur de traductions où vous pouvez indiquer la valeur de la chaîne en français.

Si vous relancez votre activité après avoir changé la langue du système dans les paramètres, vous pourrez constater que les chaînes sont adaptées. S'il n'y a pas de chaînes disponibles pour la langue actuelle du système, la version par défaut des chaînes sera employée (vous pouvez par exemple tester en changeant la langue en japonais ; si vous ne lisez pas couramment le japonais, il est conseillé d'avoir auparavant mémorisé la position du menu de configuration ;).

Gérons les erreurs

Errare humanum est... Il est peu probable que vous échappiez aux terribles `NullPointerException` en développant sous Android sauf si vous développez avec le langage Kotlin qui limite ce risque. Néanmoins d'autres exceptions tout aussi méchantes peuvent vous menacer telles que les `SecurityException` si vous oubliez d'accorder certaines permissions à votre application. Malheureusement le message surgissant en cas d'exception est peu informatif et se contente d'indiquer que l'application s'est arrêtée. Pour en savoir plus, vous devrez utiliser *logcat* sur la machine de développement. Une fenêtre *logcat* est accessible depuis *Android Studio*. La quantité de logs produite peut être impressionnante car y sont agglomérés les journaux de toutes les applications du système. C'est pourquoi on cherchera à filtrer les entrées avec des mots-clés ainsi que par niveau de sévérité.

Les logs sont indispensables pour débbugger. Pour tester, provoquez artificiellement une exception puis consultez en cours d'exécution *logcat* : vous y verrez la pile d'appel de l'exception. Il est possible aussi d'ajouter des entrées de log à la main avec les méthodes statiques de la classe Log. Elles sont du type `Log.x(String tag, String message)`, *tag* étant une étiquette pour filtrer plus facilement (généralement on

utilisera `this.getClass().getName()`), et *message* le message à enregistrer. Il existe une méthode pour chaque niveau de sévérité :

- `Log.v` pour le niveau le moins grave (verbose)
- `Log.d` pour indiquer des messages de debug
- `Log.i` pour le niveau info
- `Log.w` pour le niveau warning
- `Log.e` pour le niveau error qui est le niveau de plus grande sévérité (en fait ce n'est pas tout à fait vrai car il existe également un niveau *wtf*)

Sous Android, on essaiera de ne plus utiliser `System.out` et `System.err` qui n'ont pas vraiment de sens car l'application ne s'exécute pas en terminal ; en fait leur sortie est automatiquement redirigée vers *logcat* (niveau info pour `System.out` et niveau warning pour `System.err`). Utilisons désormais les méthodes de `Log`.

Il est possible aussi de consulter les journaux directement en terminal avec la commande `adb logcat`.

Rajoutez quelques appels à des méthodes de `Log` dans la méthode `onCreate` de votre activité ainsi que dans le listener de clic sur le bouton pour quitter. Vérifiez que vous parvenez bien à les visualiser avec *logcat*.

Essayez de provoquer une exception (par exemple en déréférençant une référence nulle) et observez comment l'application est arrêtée et quels sont les messages affichés avec *logcat*.

Rajoutons une `ImageView` de planisphère

Rajoutez une composant `ImageView` au centre de l'activité affichant [un planisphère que vous trouverez ici](#).

Associez un listener de clic à l'`ImageView` pour faire en sorte d'incrémenter un compteur à chaque clic. Ce compteur est un entier qui est conservé comme champ de l'activité. A chaque clic, le compteur est incrémenté et sa valeur est affichée à la place du message d'accueil *hello world* dans le `TextView` en haut de l'activité.

Stylons le `TextView`

On souhaite maintenant modifier le style du TextView :

- La couleur de la police sera blanche
- La couleur du fond changera en fonction de la valeur du compteur de clics. Sous Android, une couleur peut être représentée par un entier ; il est possible d'en créer une avec en indiquant la quantité de chaque composante primaire (rouge, vert et bleu) entre et . Par exemple, la couleur pourra être noire pour 1 à 10 clics, bleue pour 11 à 20 clics, rouge à partir de 21 clics...

`Color.rgb(float r, float g, float b)`

`0.0f`

`1.0f`

Jouons avec des villes

Il est temps de jouer plus sérieusement ! Notre objectif sera de proposer un jeu consistant à localiser sur le planisphère différentes villes du monde.

Pour cela, nous chargeons en mémoire une liste de villes. Téléchargez [ici](#) un fichier texte avec une liste de villes (1 ville par ligne accompagnée de leurs coordonnées géographiques). Placez ce fichier dans le répertoire `src/res/raw` (si le répertoire `raw` n'existe pas, il faut le créer dans `res`).

Nous souhaitons charger cette liste de villes en mémoire au démarrage de l'activité dans une `List<City>`. Pour cela, vous pouvez vous aider de la classe fournie [ici](#) : chargez la liste dans la méthode `onCreate` afin qu'un champ `List<City> cities` contienne toutes les villes.

Attention : le fichier `Clty.kt` que vous avez téléchargé cherche le fichier de villes dans les assets de l'application et pas dans le répertoire `res`. Vous pouvez créer une méthode `loadFromRes` s'inspirant de `loadFromAsset` mais prenant en paramètre un `Context` ainsi qu'un `resourceId` de type `Int` plutôt qu'un chemin en `String`. Vous pourrez ensuite ouvrir le fichier de villes avec `context.resources.openRawResource(resourceId)`.

Après le chargement de la liste, nous tirons au sort une ville de celle-ci et affichons son nom dans le TextView du haut. Le joueur devra cliquer sur le planisphère pour la localiser. Vous ne pouvez plus utiliser un simple `OnClickListener` sur l'`ImageView` du planisphère pour gérer le clic. En effet, il est nécessaire de connaître la position du

pointeur lors du clic ce que ce listener ne nous permet pas d'obtenir. Nous utiliserons donc à la place un `OnTouchListener` que nous installerons ainsi (en Kotlin) :

```
imageView.setOnTouchListener { _, event ->
    when (event.actionMasked) {
        MotionEvent.ACTION_DOWN -> {
            Log.i("action_down: ${event.x}, ${event.y}")
        }
        else -> { /* do nothing */ }
    }
    true // important to return true to continue to receive the following events
}
```

Nous pouvons maintenant obtenir avec `event.x` et `event.y` les coordonnées du clic. Sachant que vous pouvez obtenir les dimensions de l'`ImageView` avec ses méthodes `getWidth()` et `getHeight()`, vous pouvez convertir les dimensions cartésiennes X et Y en pixels en coordonnées angulaires sur le globe terrestre (latitude et longitude en degrés). Cette conversion réalisée, il vous reste à la comparer avec les coordonnées connues de la ville. On affiche dans le `TextView` la distance entre le point cliqué et le point réel. Vous pouvez calculer la distance entre deux points géographiques avec la méthode de l'API `Location.distanceBetween(...)`.

Afin de permettre au joueur de continuer à jouer, vous pouvez aussi indiquer dans le `TextView` outre la distance une nouvelle ville dont il faudra déterminer la position en re cliquant sur la carte. N'oubliez pas aussi d'afficher le nombre de clics correspondant aux nombre de parties réalisées. Lorsque l'utilisateur sera fatigué de jouer, il pourra cliquer sur le bouton *Quit* afin de clôturer le jeu.

Une fortune avant de quitter

Avant de quitter définitivement le jeu, nous souhaiterions afficher un aphorisme.

Avant de quitter l'activité de jeu avec `finish()`, nous lançons une nouvelle activité chargée d'afficher la citation proverbiale.

La fortune sera récupérée a l'adresse suivante (que vous pouvez tester depuis un navigateur web classique) : <https://fortuneapi.herokuapp.com/>

Créez une nouvelle *empty activity* nommée `FortuneActivity` avec un `TextView` au centre où sera affiché le proverbe et un bouton permettant de quitter l'activité.

Pour lancer cette activité depuis l'activité de jeu, nous pouvons créer un Intent pour la nouvelle activité et demander son lancement :

```
Intent intent = new Intent(this, FortuneActivity.class);
startActivity(intent)
```

Dans la méthode onCreate() de FortuneActivity, nous allons demander la récupération d'un proverbe sur une API web. Pour cela nous allons employer la bibliothèque Volley. Tout d'abord nous devons la déclarer comme dépendance de notre application Android dans le fichier build.gradle de app :

```
dependencies {
    ...
    implementation 'com.android.volley:volley:1.1.1'
}
```

Il faut d'autre part déclarer la permission *INTERNET* pour notre application Android afin de l'autoriser à réalisation des communications réseau. Pour cela nous l'indiquons dans le fichier AndroidManifest.xml :

```
<manifest ...>

    <uses-permission android:name="android.permission.INTERNET" />
    ....
</manifest>
```

On peut ensuite réaliser une requête vers le serveur. Voici un exemple inspiré de ce qui est présenté dans [la documentation de Volley](#) :

```
final TextView textView = (TextView) findViewById(R.id.text);
// ...

// Instantiate the RequestQueue.
RequestQueue queue = Volley.newRequestQueue(this);
String url = "https://fortuneapi.herokuapp.com/";

// Request a string response from the provided URL.
```

```
StringRequest stringRequest = new StringRequest(Request.Method.GET, url,
        new Response.Listener<String>() {
            @Override
            public void onResponse(String response) {
                textView.setText(response);
            }
        }, new Response.ErrorListener() {
            @Override
            public void onErrorResponse(VolleyError error) {
                textView.setText("That didn't work because of this error: " + error);
            }
        });

// Add the request to the RequestQueue.
queue.add(stringRequest);
```

Vous remarquerez sans doute que la fortune récupérée est entourée par des guillemets et que des caractères sont *escaped* (retour à la ligne notamment) pour être au format d'une *string* JSON. Pour convertir le String reçu en String sans caractères spécialisés, nous pouvons utiliser cette fonction :

```
public static String unescapeJSONString(String v) {
    return new JSONTokener(v).nextValue().toString();
}
```

Félicitations, nous sommes arrivés au terme du premier TP !