université de BORDEAUX

Développement Android

Visite guidée

TD1

Android est un système d'exploitation tournant sur différents types d'appareils : téléphones, tablettes voire même téléviseurs. Les applications Android sont développées en Java et l'IDE de prédilection est Android Studio (surcouche d'IntelliJ développée par Google). Avant de commencer, allez faire un tour sur `http://developer.android.com`, le site officiel des développeurs Android. La plupart de la documentation se trouve là, ainsi que de nombreux exemples de code.

Lorsqu'une question est précédée d'un "▷ ", vous devez consigner votre réponse dans un fichier texte (à envoyer à la fin de la séance).

1 Prise en main du studio

Commencez par lancer l'Android Studio et choisissez les options par défaut pour finaliser l'installation. Avant de créer un nouveau projet, rendez-vous dans le menu "Configure" > "Project Defaults" > "Project Structure" et vérifiez que le JDK est bien celui d'oracle ou sun (7 ou 8). Vous pouvez ensuite créer un nouveau projet. Appelez votre application `Hello`. Cliquez sur `Next`. Choisissez la version minimale du SDK que votre application supportera. Plus la version est récente et plus l'API est riche (mais moins de téléphones peuvent faire tourner votre application). Jetez à oeil au lien `Help me choose`. Nous vous conseillons de laisser le choix par défaut. Cliquez sur `Next`. Choisissez `Empty Activity`, puis `Next` et enfin `Finish`.

1.1 Première exécution

Vous voici avec le projet créé. Nous allons l'exécuter en cliquant sur la flèche verte dans la barre d'outils. Une boite de dialogue s'ouvre et propose de lancer un émulateur.

Une fois l'émulateur lancé, vous pouvez déverrouillez son interface et voir votre application apparaître. Un conseil : **ne fermez pas votre émulateur**, la même instance sera utilisée pour les exécutions futures.

1.2 Contrôle basique de l'émulateur

Vous pouvez utiliser l'émulateur de l'intérieur, comme s'il s'agissait d'un vrai périphérique. Mais vous pouvez aussi en contrôler certains aspect depuis l'extérieur. Dans le studio, cherchez le menu *Tools* > *Android*, puis choisissez l'*Android Device Monitor*. Explorez les menus de cet outil, en particulier l'onglet *Emulator Control*.

→ Envoyez un SMS à votre périphérique virtuel, puis simulez un appel entrant. Vous pouvez ensuite fermez la fenêtre de l'Android Device Monitor (nous y reviendrons plus tard).

1.3 Arborescence du projet

Regardez l'arborescence de votre projet et cherchez à comprendre à quoi sert chaque élément (ignorez le répertoire Gradle Scripts). Vous pouvez constater qu'un projet Android contient des fi-

chiers très variés (autres que le code source java), par exemple dans le répertoire `manifests` et `res`.
▷ Pour chaque élément (dossier ou fichier, selon ce qui vous parait pertinent), expliquez son rôle avec vos propres mots.

1.4 Les activités

Les *activités* sont les composants centraux d'une application Android. Une activité peut être vue comme une "fenêtre" permettant à l'utilisateur d'effectuer une action précise. La plupart des applications Android sont composées de plusieurs activités (ou de plusieurs *fragments*, qui sont des versions plus légères s'exécutant dans une même activité, un peu comme des onglets). Chaque activité (ou fragment) possède une interface graphique décrite dans un fichier de layout (en XML). En fait, on peut même associer un layout XML à n'importe quel composant (bouton, champ texte, etc.).
▷ Quel est le nom du fichier de layout que votre activité utilise ?
▷ Comment l'activité sait-elle qu'il faut utiliser celui là plutôt qu'un autre ? (Recopiez l'instruction correspondante)
▷ Observez l'existence de la classe `R`. Émettez une hypothèse sur son rôle.

1.5 Les layouts XML

Ainsi, Android permet de définir l'interface graphique des activités (et autres composants graphiques) dans des fichiers séparés en XML. Cela permet d'alléger beaucoup le code tout en séparant la logique du visuel de l'application. Ouvrez le fichier XML dans *res/layout*, et regardez son code source dans l'onglet *Text*. Observez la structure de l'interface décrite : elle est composée d'un layout racine (de type `RelativeLayout`) dans lequel se trouve un champ texte (`TextView`).

→ Dans le rendu de l'onglet *Design*, sélectionnez le textview pour lui donner l'identifiant `txtHello` (via la liste des propriétés à droite : champ `id`). Allez ensuite voir les changements dans le code XML (onglet *Text*).

▷ Quelle ligne a été ajoutée ?

→ Allez maintenant dans le code java de votre activité, et ajoutez les deux instructions suivantes à la méthode onCreate() :

```
TextView tv = (TextView) findViewById(R.id.txtHello);
tv.setText("Hello from the code!");
```

Exécutez votre application à nouveau, comprenez bien ce que vous avez fait et le rôle de chaque élément dans ces instructions.
▷ Expliquez brièvement le rôle de chaque élement.
▷ Tentez de placer ces instructions avant ou après l'instruction `setContentView`. Quelle différence observez-vous ? Pourquoi ?

1.6 Traces d'exécution avec LogCat

D'outil LogCat (intégré au studio, fenêtre du bas) trace l'exécution de votre programme.

Les messages affichés dans LogCat ont différents niveaux d'importance. On peut d'ailleurs les filtrer selon six niveaux.
▷ Quels sont ces niveaux, du moins important au plus important ?

Dans `onCreate()`, écrivez un message quelconque à l'aide de `System.out.println()`.
▷ Quel niveau d'importance a ce message ? (essayez les différents filtres)

Pour écrire un message d'un niveau donné, on peut utiliser la classe `Log`, par exemple

Log.i("TAG", "message informatif") ;

Ici le message sera de niveau INFO car on a utilisé la méthode `i()`. Le premier argument est un tag que l'on peut ajouter pour organiser nos messages de manière plus fine.
▷ Quelles sont les méthodes correspondant à chaque niveau d'importance ?
▷ A quoi correspond la méthode `wtf()` ;)

1.7 Traçage du cycle de vie de votre activité

Dans le code java, observez la déclaration de votre classe `MainActivity`.

▷ De quelle classe cette classe hérite-t-elle ? Et cette dernière à son tour ?

▷ Remontez la hiérarchie jusqu'à la classe `Context`, dont vous lirez le chapeau "Class Overview" dans la javadoc. Reformulez ici ce que vous comprenez de cette dernière classe.

Vous avez certainement noté que votre classe descend de la classe `Activity`. Elle possède, à ce titre, un grand nombre de méthodes que vous ne voyez pas ici. La liste peut être consultée via le menu *Code* > *Override Methods* (Ctrl+o).

→ Surchargez les principales méthodes du cycle de vie de l'application, à savoir `onCreate` (déjà fait), `onStart, onResume, onPause, onStop, onRestart,` et `onDestroy`.

→ Rajoutez dans chacune de ces méthodes une instruction de log pour faire afficher leur nom par LogCat quand elles s'exécutent. Vous utiliserez à chaque fois le tag `"CV"` (pour cycle de vie).

Le cycle de vie d'une activité est illustré au dos de cette page (Figure 1). Ci-dessous nous allons générer une série d'évènement pour observer les effets sur le cycle de vie de notre application.

→ Réalisez les scénarios suivants et observez, via LogCat, l'évolution du cycle de vie.
▷ Indiquez, pour chaque action, les méthodes du cycle de vie qui sont exécutées. Indiquez lorsque quelque chose vous parait étrange.

1. Lancez l'application
2. Appuyez sur le bouton BACK
3. Lancez de nouveau l'application
4. Simulez un appel entrant, décrochez, puis raccrochez
5. Revenez à l'accueil (bouton HOME)
6. Réactivez l'application
7. Tuez l'application depuis la liste des fenêtres (bouton en bas à droite)
8. Relancez l'application puis changez l'orientation de l'écran (Ctrl+F12 dans l'émulateur)
9. Tuez brutalement l'application depuis le téléphone
   (*Settings* > *Apps* > *All* > *Hello* > *Force Stop*)

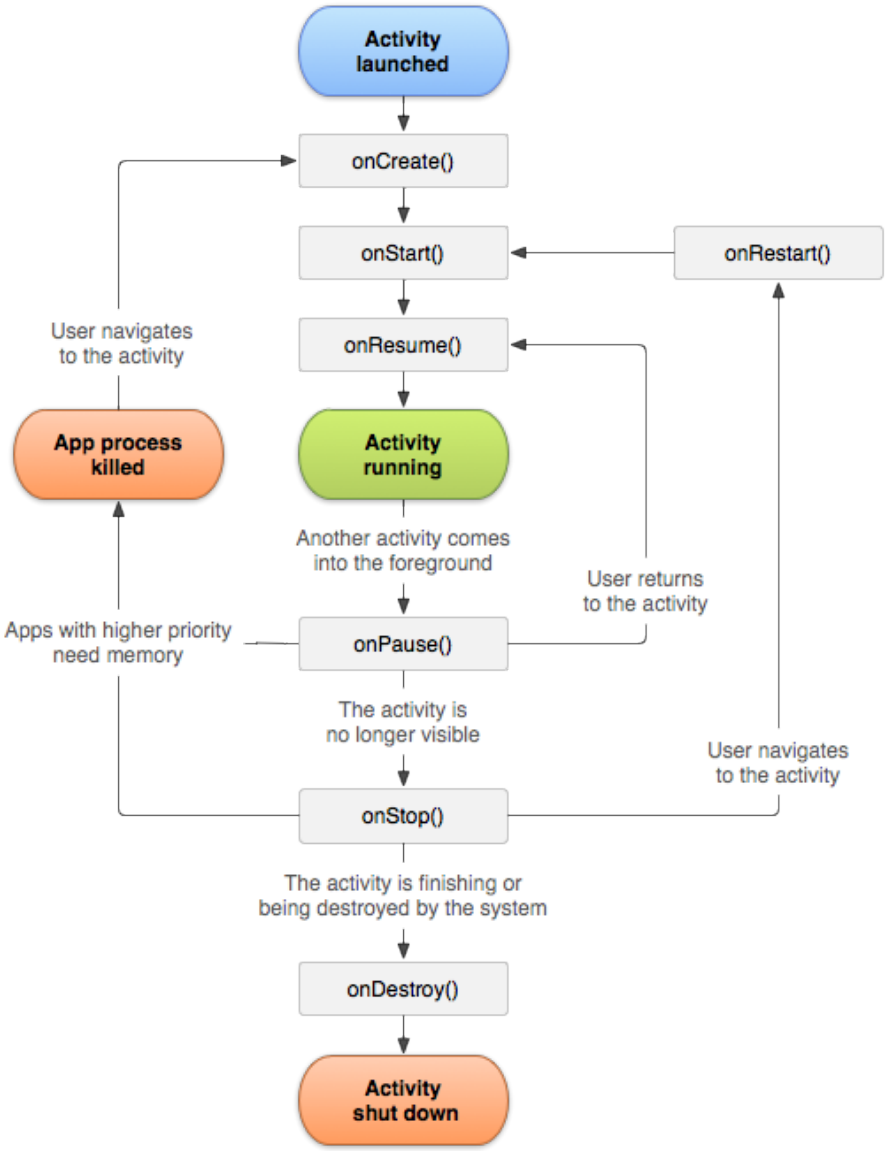

FIGURE 1 – Cycle de vie d'une activité Android

À rendre par courriel :

Une archive zip nommée `td1-nom1-nom2.zip` où `nom1` est le nom de famille alphabétiquement le plus petit de votre binôme, et `nom2` le suivant. L'archive doit contenir :

1. Votre fichier de réponse `.txt`
2. Votre fichier de layout `.xml`
3. Le fichier source de votre activité `.java`