A la découverte d'Android

inspiré du cours d'Arnaud Casteigts

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.

Avant de commencer, visitez `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.

1 Démarrage

La commande `android-studio` lance l'IDE et déploie sur votre compte une configuration toute prête.

Vous pouvez maintenant créer un nouveau projet. Appelez votre application `Hello`. Vérifiez que l'emplacement du projet est bien dans votre espace de stockage (∼login/espaces/travail). Choisissez Java comme langage. Choisissez la version minimale de l'API que votre application supportera. Plus la version est récente et plus l'API est riche (mais moins de téléphones peuvent faire tourner votre application). Nous vous conseillons dans un premier temps de laisser le choix par défaut. Cliquez sur `Next`. Choisissez `Empty Activity`, puis `Next` et enfin `Finish`.

2 Première exécution

Une fois le projet créé on l'exécute en cliquant sur la flèche verte dans la barre d'outils. Vous devez préalablement créer un émulateur en choisissant le téléphone et le système émulé. Ne téléchargez pas de nouvelle image système mais utilisez une image déjà installée (qui ne sera pas forcément proposée parmi les *"recommandées"*).

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.

Vous pouvez aussi si vous disposez d'un téléphone android et d'un cable usb tester l'application sur votre téléphone. Pour cela il faut tout d'abord autoriser le débogage usb, qui se trouve dans les options pour développeur, cachées par défaut. Pour faire apparaitre ces options il faut aller dans `Paramètres` > `A propos` et cliquer sur le numéro de build plusieurs fois, le menu sera alors disponible dans les paramètres.

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 fichiers très variés (autres que le code source java), par exemple dans les répertoire `manifests` et `res`.

4 Les activités

Les *activités* sont les composants centraux d'une application Android. Une activité peut être vue comme les "fenêtres" 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.).

Remarquez le nom du fichier de layout que votre activité utilise et comment l'activité sait qu'il faut utiliser celui là plutôt qu'un autre.

Observez aussi l'existence de la classe R et essayez de trouver son rôle.

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 partie graphique de l'application. Ouvrez le fichier XML situé 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 `ConstraintLayout`) dans lequel se trouve un champ texte (`TextView`).

À partir 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).

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. Tentez de placer ces instructions avant ou après l'instruction setContentView.

6 Traces d'exécution avec LogCat

L'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.

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. Trouvez les méthodes correspondant à chaque niveau d'importance.

7 Les autres outils importants

Même si vous n'avez pas encore besoin de ces outils, notez leur présence puisqu'ils sont directement intégrés dans Android Studio. Et ils vous seront très certainement très utiles avant la fin de l'année... par exemple lorsque qu'une opération prend "anormalement" beaucoup de temps. Les boutons correspondant à ces outils sont à droite de la flèche permettant de lancer l'application.

— Le `débogueur` : essayez de placer un breakpoint, visualiser des valeurs, et exécuter pas à pas.
— Le `profiler` : il permet de visualiser l'état des ressources machines en cours d'exécution. Vous pouvez tracer les allocations mémoire (Memory) et l'utilisation du processeur (CPU).

8 Traçage du cycle de vie de votre activité

Dans le code java, observez la déclaration de la classe `MainActivity`. Remontez la hiérarchie jusqu'à la classe `Context`, dont vous lirez le chapeau "Class Overview" dans la javadoc. Vous avez certainement noté que cette 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 $Code > OverrideMethods$ (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 dans la Figure 1.

Ci-dessous nous allons générer une série d'évènements 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.

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
8. Relancez l'application puis changez l'orientation de l'écran
9. Tuez brutalement l'application depuis le téléphone ($Settings > Apps > All > Hello > ForceStop$)

9 Afficher/modifier une image

La classe `Bitmap` du package `android.graphics` permet de manipuler des images matricielles : accès à la taille de l'image, lecture/écriture des valeurs des pixels.
Voir `http://developer.android.com/reference/android/graphics/Bitmap.html`.

1. Créez une nouvelle application qui affiche une image que vous aurez au préalable stockée dans les ressources de l'application (trouvez dans quel répertoire).

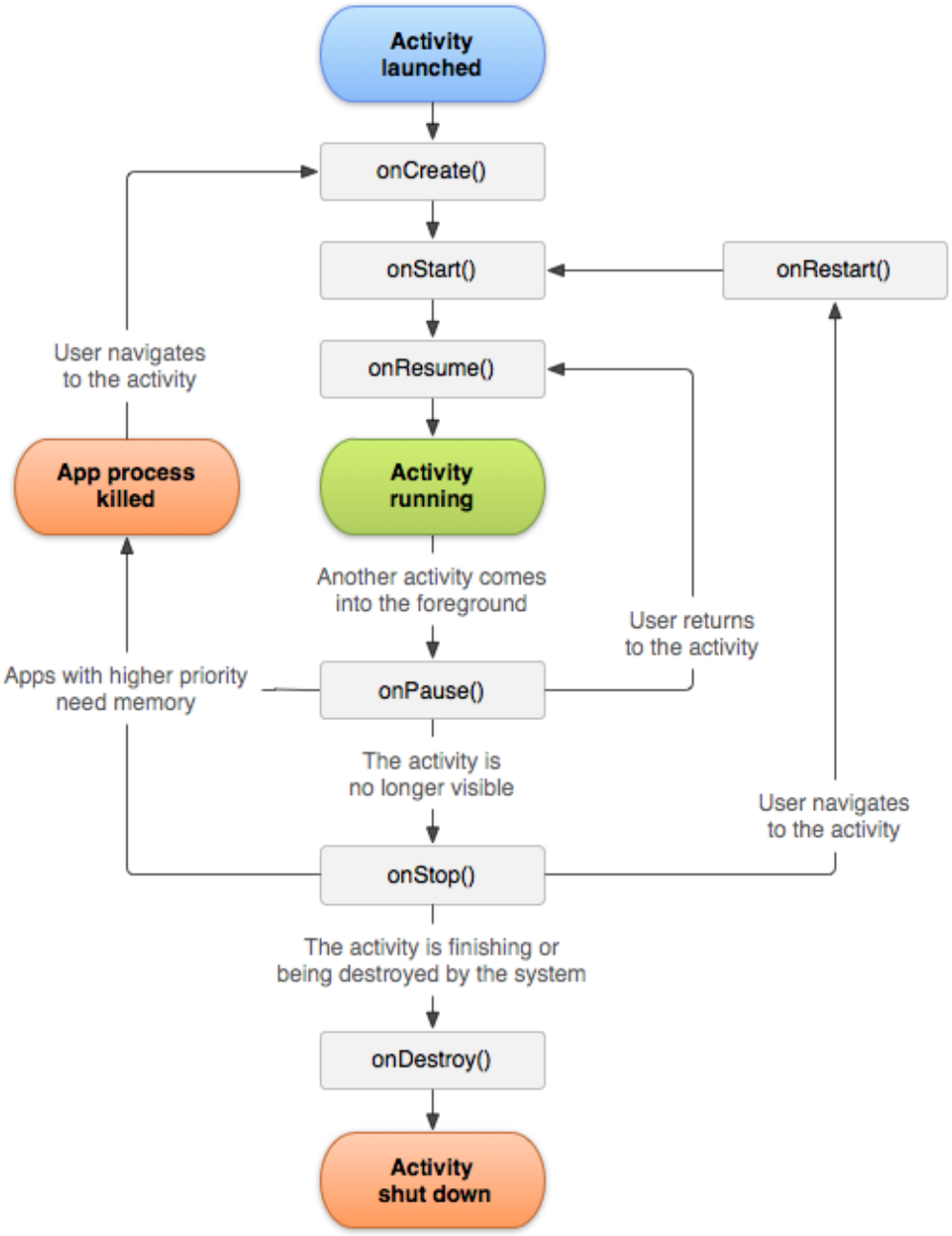

FIGURE 1 – Cycle de vie d'une activité android

Indication : Le composant graphique qui permet d'afficher une image à l'écran est `ImageView`. Pour créer un objet de type `Bitmap` à partir d'un fichier image, vous pouvez utiliser la classe `BitmapFactory`.

Jouez sur les propriétés `layout_width` et `layout_height` de l'ImageView pour afficher l'image à différentes tailles.

2. Créez un champ de texte qui affiche la taille de l'image initiale. Vérifiez que la taille affichée est correcte.

   Indication : Voir la classe `BitmapFactory.Options`.

3. Testez la modification de la valeur d'un pixel en traçant une ligne de pixels noirs au milieu de l'image.

10 Un premier traitement : griser l'image

La valeur d'un pixel est stockée sur un entier (int) qui regroupe les trois composantes Rouge, Vert et Bleu. Le quatrième octet correspond à la transparence (voir la classe `android.graphics.Color`). Dans une image RGB, un pixel dont les trois composantes RGB sont égales apparaît gris, c'est-à-dire qu'on ne percoit qu'une luminosité entre le blanc et le noir. Pour griser un pixel, on calcule sa luminosité comme moyenne de ses trois composantes. Cette valeur est ensuite affectée à chacune des trois composantes du pixel. On peut utiliser la moyenne pondérée $0.3R + 0.59G + 0.11B$.

1. Ecrire une méthode `void toGray(Bitmap bmp)` qui grise tous les pixels du Bitmap en utilisant les méthodes `getPixel` et `setPixel` de la classe `Bitmap`.
2. Ajoutez à l'application un bouton permettant de griser l'image affichée comme dans l'exemple ci-dessous.

   Indication : utiliser le composant graphique `Button`. Comme `ImageView Button` fait partie du package `android.widget`.

Avant

Après

3. Ecrire une seconde version de la méthode `toGray` qui utilise cette fois les méthodes `getPixels` et `setPixels` de la classe `Bitmap` : on manipule les valeurs des pixels via un tableau d'entiers.
4. Comparer l'efficacité des deux versions de la méthode `toGray` en utilisant le profiler. Voir `https://developer.android.com/studio/profile/`.