Ce chapitre est très important. Il vous permettra d'enfin mettre la main à la pâte, mais surtout on abordera la notion de cycle d'une activité, qui est la base d'un programme pour Android. Si pour vous un programme débute forcément par un main, vous risquez d'être surpris !

On va tout d'abord voir ce qu'on appelle des activités et comment les manipuler. Sachant que la majorité de vos applications (si ce n'est toutes) contiendront plusieurs activités, il est indispensable que vous maîtrisiez ce concept ! Nous verrons aussi ce que sont les vues et nous créerons enfin notre premier projet — le premier d'une grande série — qui n'est pas, de manière assez surprenante, un Hello World! . Enfin presque !

Activité et vue

**Qu'est-ce qu'une activité ?**

Si vous observez un peu l'architecture de la majorité des applications Android, vous remarquerez une construction toujours à peu près similaire. Prenons par exemple l'application du Play Store. Vous avez plusieurs fenêtres à l'intérieur même de cette application : si vous effectuez une recherche, une liste de résultats s'affichera dans une première fenêtre et si vous cliquez sur un résultat, une nouvelle fenêtre s'ouvre pour vous afficher la page de présentation de l'application sélectionnée. Au final, on remarque qu'une application est un assemblage de fenêtres entre lesquelles il est possible de naviguer.

Ces différentes fenêtres sont appelées des activités. Un moyen efficace de différencier des activités est de comparer leur interface graphique : si elles sont radicalement différentes, c'est qu'il s'agit d'activités différentes. De plus, comme une activité remplit tout l'écran, votre application ne peut en afficher qu'une à la fois. La figure suivante illustre ce concept.

En cliquant sur un élément de la liste à gauche, on ouvre une nouvelle activité

Faire une recherche ouvre une liste de resultats, comme dans l'écran de gauche. Toucher un résultat de la recherche ouvre un nouvel écran, dont l'interface graphique n'a rien à voir avec le précédent. Il s'agit donc de deux activités différentes.

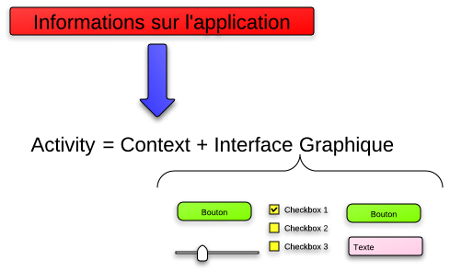
Voici un autre example. Imaginez que vous naviguiez sur OpenClassrooms avec votre téléphone, le tout en écoutant de la musique sur ce même téléphone. Il se passe deux choses dans votre système :

* La navigation sur internet, permise par une interface graphique (la barre d'adresse et le contenu de la page web, au moins) ;
* La musique, qui est diffusée en fond sonore, mais qui n'affiche pas d'interface graphique à l'heure actuelle puisque l'utilisateur consulte le navigateur.

On a ainsi au moins deux applications lancées en même temps ; cependant, le navigateur affiche une activité alors que le lecteur audio n'en affiche pas.

En d'autres termes, une activité est un support sur lequel nous allons greffer une interface graphique. Cependant, ce n'est pas le rôle de l'activité que de créer et de disposer les éléments graphiques, elle n'est que l’échafaudage sur lequel vont s'insérer les objets graphiques et elle va établir les liens entre l'interface graphique et le logique derrière les éléments qui la constitue.

De plus, une activité contient des informations sur l'état actuel de l'application : ces informations s'appellent le context. Ce context constitue un lien avec le système Android ainsi que les autres activités de l'application, comme le montre la figure suivante.

Une activité est constituée du contexte de l'application et d'une seule et unique interface graphique

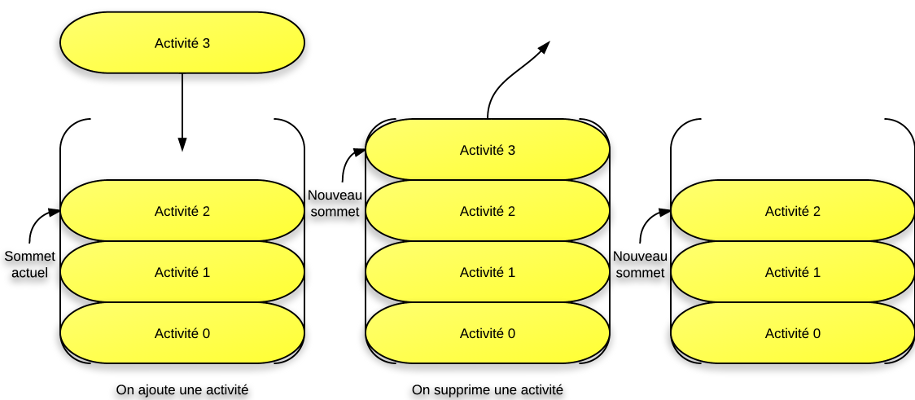
**États d'une activité**

Si un utilisateur reçoit un appel, il devient plus important qu'il puisse y répondre que de faire en sorte qu'il puisse écouter la chanson que votre application diffuse. Pour pouvoir toujours répondre à ce besoin, les développeurs d'Android ont eu recours à un système particulier :

* À tout moment votre application peut laisser place à une autre application, qui a une priorité plus élevée. Si votre application utilise trop de ressources système, alors elle empêchera le système de fonctionner correctement et Android l'arrêtera sans vergogne.
* Votre activité existera dans plusieurs états au cours de sa vie, par exemple un état actif pendant lequel l'utilisateur l'exploite, et un état de pause quand l'utilisateur reçoit un appel.

Pour être plus précis, quand une application se lance, elle se met tout en haut de ce qu'on appelle la pile d'activités.

Une pile est une structure de données de type « LIFO », c'est-à-dire qu'il n'est possible d'avoir accès qu'à un seul élément de la pile, le tout premier élément, aussi appelé **sommet**. Quand on ajoute un élément à cette pile, le nouvel élément prendra la première place et deviendra le nouveau sommet. Quand on veut récupérer un élément, ce sera le sommet qui sera récupéré, sorti de la liste et l'objet en deuxième place deviendra le nouveau sommet, comme illustré à la figure suivante.

Fonctionnement de la pile d'activités

L'activité que voit l'utilisateur est celle qui se trouve au-dessus de la pile. Ainsi, lorsqu'un appel arrive, il se place au sommet de la pile et c'est lui qui s'affiche à la place de votre application, qui n'est plus qu'à la deuxième place. Votre activité ne reviendra qu'à partir du moment où toutes les activités qui se trouvent au-dessus d'elle seront arrêtées et sorties de la pile. On retrouve ainsi le principe expliqué précédemment, on ne peut avoir qu'une application visible en même temps sur le terminal, et ce qui est visible est l'interface graphique de l'activité qui se trouve au sommet de la pile.

Une activité peut se trouver dans trois états qui se différencient surtout par leur visibilité :

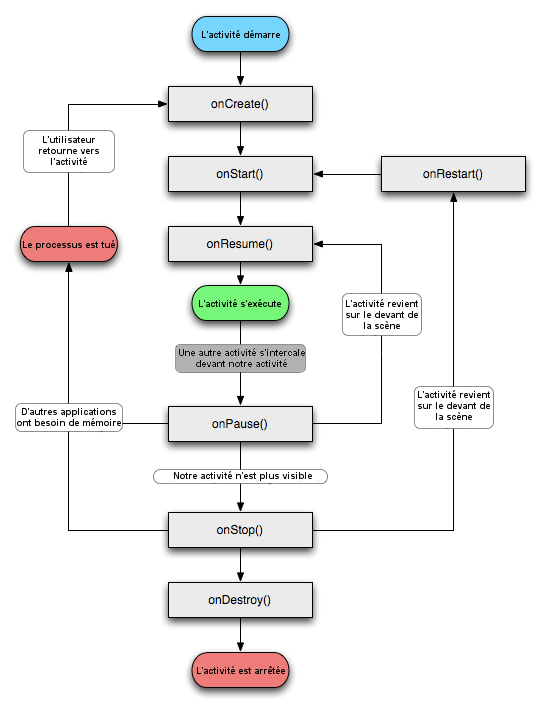
| **État** | **Visibilité** | **Description** |
| --- | --- | --- |
| Active  (« active» ou «running») | L'activité est visible en totalité. | Elle est sur le dessus de la pile, c'est ce que l'utilisateur consulte en ce moment même et il peut l'utiliser dans son intégralité. C'est cette application qui a le *focus*, c'est-à-dire que l'utilisateur agit directement sur l'application. |
| Suspendue (« paused») | L'activité est partiellement visible à l'écran. C'est le cas quand vous recevez un SMS et qu'une fenêtre semi-transparente se pose devant votre activité pour afficher le contenu du message et vous permettre d'y répondre par exemple. | Ce n'est pas sur cette activité qu'agit l'utilisateur.  L'application n'a plus le focus, c'est l'application au-dessus qui l'a. Pour que notre application récupère le focus, l'utilisateur devra se débarrasser de l'application partiellement au-dessus pour que l'utilisateur puisse à nouveau interagir avec notre activité. Si le système a besoin de mémoire, il peut très bien tuer l'application (cette affirmation n'est plus vraie si vous utilisez un SDK avec l'API 11 minimum). |
| Arrêtée  («stopped») | L'activité est tout simplement invisible pour l'utilisateur, car une autre activité prend toute la place sur l'écran. | L'application n'a évidemment plus le focus, et puisque l'utilisateur ne peut pas la voir, il ne peut pas agir dessus.  Le système retient son état pour pouvoir reprendre, mais il peut arriver que le système tue votre application pour libérer de la mémoire système. |

*Mais j'ai pourtant déjà vu des systèmes Android avec deux applications visibles en même temps !*

Ah oui, c'est possible. Mais il s'agit d'un artifice, il n'y a vraiment qu'une application qui est active. Pour faciliter votre compréhension, je vous conseille d'oublier ces systèmes.

**Cycle de vie d'une activité**

Une activité n'a pas de contrôle direct sur son propre état (et par conséquent vous non plus en tant que programmeur), il s'agit plutôt d'un cycle rythmé par les interactions avec le système et d'autres applications. Voici un schéma qui présente ce que l'on appelle **le cycle de vie d'une activité**, c'est-à-dire qu'il indique les étapes que va traverser notre activité pendant sa vie, de sa naissance à sa mort. Vous verrez que chaque étape du cycle est représentée par une méthode. Nous verrons comment utiliser ces méthodes en temps voulu.



Cycle de vie d'une activité

Les activités héritent de la classe [Activity](http://developer.android.com/reference/android/app/Activity.html). Or, la classe Activity hérite de l'interface [Context](http://developer.android.com/reference/android/content/Context.html)dont le but est de représenter tous les composants d'une application. On les trouve dans le package android.app.Activity.

Pour rappel, un package est un répertoire qui permet d'organiser notre code source, un récipient dans lequel nous allons mettre nos classes de façon à pouvoir trier votre code et différencier des classes qui auraient le même nom. Concrètement, supposez que vous ayez à créer deux classes X — qui auraient deux utilisations différentes, bien sûr. Vous vous rendez bien compte que vous seriez dans l'incapacité totale de différencier les deux classes si vous deviez instancier un objet de l'une des deux classes X, et Java vous houspillera en déclarant qu'il ne peut pas savoir à quelle classe vous faites référence. C'est exactement comme avoir deux fichiers avec le même nom et la même extension dans un même répertoire : c'est impossible car c'est incohérent.

Pour contrer ce type de désagrément, on organise les classes à l'aide d'une hiérarchie. Si je reprends mon exemple des deux classes X, je peux les placer dans deux packages différents Y et Z par exemple, de façon à ce que vous puissiez préciser dans quel package se trouve la classe X sollicitée. On utilisera la syntaxeY.X pour la classe X qui se trouve dans le package Y et Z.X pour la classe X qui se trouve dans le packageZ. Dans le cas un peu farfelu du code source d'un navigateur internet, on pourrait trouver les packagesWeb.Affichage.Image, Web.Affichage.Video et Web.Telechargement.

Les **vues** (que nos amis anglais appellent ***view***), sont ces fameux composants qui viendront se greffer sur notre échafaudage, il s'agit de l'unité de base de l'interface graphique. Leur rôle est de fournir du contenu visuel avec lequel il est éventuellement possible d'interagir. À l'instar de [l'interface graphique en Java](http://www.siteduzero.com/tutoriel-3-10601-apprenez-a-programmer-en-java.html#part_10599), il est possible de disposer les vues à l'aide de conteneurs, nous verrons comment plus tard.

Les vues héritent de la classe View. On les trouve dans le package android.view.View.

Un non-Hello world!

Vous trouverez les fichiers créés dans le panneau de gauche (voir figure suivante). Vous pouvez aussi rechercher la classe à laquelle vous souhaitez accéder avec le raccourcis **CTRL** + **N**.

Le navigateur de fichiers

Android Studio (IntelliJ IDEA pour être précis) est un IDE célèbre pour sa superbe collection de raccourcis claviers. Je vous recommande chaudement d'apprendre le plus de raccourcis possibles. Les meilleurs développeurs n'utilisent même plus la souris pour coder, c'est beaucoup trop lent !

On y trouve notre premier grand répertoire src/, celui qui contiendra tous les fichiers sources .java. Ouvrez le seul fichier qui s'y trouve, chez moi PremiereActivite.java (en double cliquant dessus). Vous devriez avoir un contenu plus ou moins similaire à celui-ci :

1package com.openclassrooms.fr.premierprojet;

2

3import android.app.Activity;

4import android.os.Bundle;

5import android.view.Menu;

6import android.view.MenuItem;

7

8

9public class PremiereActivite extends Activity {

10

11    @Override

12    protected void onCreate(Bundle savedInstanceState) {

13        super.onCreate(savedInstanceState);

14        setContentView(R.layout.activity\_premiere\_activite);

15    }

16

17

18    @Override

19    public boolean onCreateOptionsMenu(Menu menu) {

20        // Inflate the menu; this adds items to the action bar if it is present.

21        getMenuInflater().inflate(R.menu.premiere\_activite, menu);

22        return true;

23    }

24

25    @Override

26    public boolean onOptionsItemSelected(MenuItem item) {

27        // Handle action bar item clicks here. The action bar will

28        // automatically handle clicks on the Home/Up button, so long

29        // as you specify a parent activity in AndroidManifest.xml.

30        int id = item.getItemId();

31        if (id == R.id.action\_settings) {

32            return true;

33        }

34        return super.onOptionsItemSelected(item);

35    }

36}

Ah ! On reconnaît certains termes que je viens tout juste d'expliquer ! Cependant, Android Studio a fait du zèle et a créé des choses que je n'expliquerai que plus tard, alors je vais faire quelque chose que je n'aime pas faire, mais je vais vous demander de modifier du code sans comprendre pourquoi. Supprimez les méthodes onCreateOptionsMenu et onOptionsItemSelected afin d'obtenir :

1package com.openclassrooms.fr.premierprojet;

2

3import android.app.Activity;

4import android.os.Bundle;

5import android.view.Menu;

6import android.view.MenuItem;

7

8

9public class PremiereActivite extends Activity {

10

11    @Override

12    protected void onCreate(Bundle savedInstanceState) {

13        super.onCreate(savedInstanceState);

14        setContentView(R.layout.activity\_premiere\_activite);

15    }

16}

Je vais prendre toutes les lignes une par une, histoire d'être certain de ne déstabiliser personne.

1package com.openclassrooms.fr.premierprojet;

Là, on déclare que notre programme se situe dans le packagecom.openclassrooms.fr.premierprojet. Si on veut faire référence à notre application, il faudra faire référence à ce package.

1import android.app.Activity;

2import android.os.Bundle;

3import android.view.Menu;

4import android.view.MenuItem;

On importe des classes qui se trouvent dans des packages différents : les classes Activity, Bundle, et Menu et MenuItem qui se trouvent dans le même package. Deux de ces packages sont inutiles car nous avons supprimé des fonctions dans le code, comme le montre l'image suivante.

Les imports inutiles sont en gris

Pour résoudre ce problème, on va utiliser le raccourci clavier le plus magique d'Android Studio : **ALT** + **ENTREE**. Ce raccourci permet d'effectuer des correstions d'erreurs rapidement, ainsi que d'autres actions rapides. Ici, on vous propose de Optimize Imports. Appuyez sur **ENTREE** pour valider votre sélection.

1public class PremiereActivite extends Activity {

2  //…

3}

On déclare ici une nouvelle classe, PremiereActivity, et on la fait dériver de Activity, puisqu'il s'agit d'une activité.

1@Override

2protected void onCreate(Bundle savedInstanceState)

3  //…

4}

Le petit @Override permet d'indiquer que l'on va redéfinir une méthode qui existait auparavant dans la classe parente, ce qui est logique puisque vous saviez déjà qu'une activité avait une méthode void onCreate() et que notre classe hérite de Activity.

L'instruction @Override est facultative. Cependant, elle permet au compilateur d'optimiser le bytecode alors faites y attention.

Cette méthode est la première qui est lancée au démarrage d'une application, mais elle est aussi appelée après qu'une application a été tuée par le système en manque de mémoire ! C'est à cela que sert le paramètre de type Bundle :

* S'il s'agit du premier lancement de l'application ou d'un démarrage alors qu'elle avait été quittée normalement, il vaut null.
* Mais s'il s'agit d'un retour à l'application après qu'elle a perdu le focus et redémarré, alors il se peut qu'il ne soit pas null si vous avez fait en sorte de sauvegarder des données dedans, mais nous verrons comment dans quelques chapitres, puisque ce n'est pas une chose indispensable à savoir pour débuter.

Dans cette méthode, vous devez définir ce qui doit être créé à chaque démarrage, en particulier l'interface graphique.

1super.onCreate(savedInstanceState);

L'instruction super signifie qu'on fait appel à une méthode ou un attribut qui appartient à la superclasse de la méthode actuelle, autrement dit la classe juste au-dessus dans la hiérarchie de l'héritage — la classe parente, c'est-à-dire la classe Activity.

Ainsi, super.onCreate fait appel au onCreate de la classe Activity, mais pas au onCreate deMainActivity. Il gère bien entendu le cas où le Bundle est null. Cette instruction est obligatoire.

L'instruction suivante :

1setContentView(R.layout.activity\_premiere\_activite);

sera expliquée en partie dans quelques instants.

En attendant, vous pouvez remplacer le contenu du fichier par celui-ci :

1//N'oubliez pas de déclarer le bon package dans lequel se trouve le fichier !

2

3import android.app.Activity;

4import android.os.Bundle;

5import android.widget.TextView;

6

7public class PremiereActivite extends Activity {

8

9    @Override

10    protected void onCreate(Bundle savedInstanceState) {

11        super.onCreate(savedInstanceState);

12

13        TextView texte = new TextView(this);

14        texte.setText("Bonjour, vous me devez 1 000 000€.");

15        setContentView(texte);

16    }

17}

Nous avons créé un object que j'ai appelé texte . Cet object est de type TextView, j'imagine que le nom est déjà assez explicite. :D Il s'agit d'une vue (View)… qui représente un texte (Text). J'ai changé le texte qu'affichera cette vue avec la méthode void setText(String texte).

La méthode void setContentView (View vue) permet d'indiquer l'interface graphique de notre activité. Si nous lui donnons un TextView, alors l'interface graphique affichera ce TextView et rien d'autre.

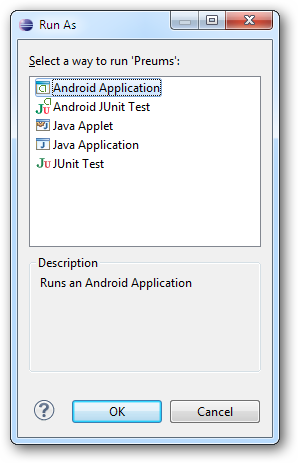
Lancement de l'application

Souvenez-vous, je vous ai dit précédemment qu'il était préférable de ne pas fermer l'AVD, celui-ci étant long à se lancer. Si vous l'avez fermé, ce n'est pas grave, il s'ouvrira tout seul.

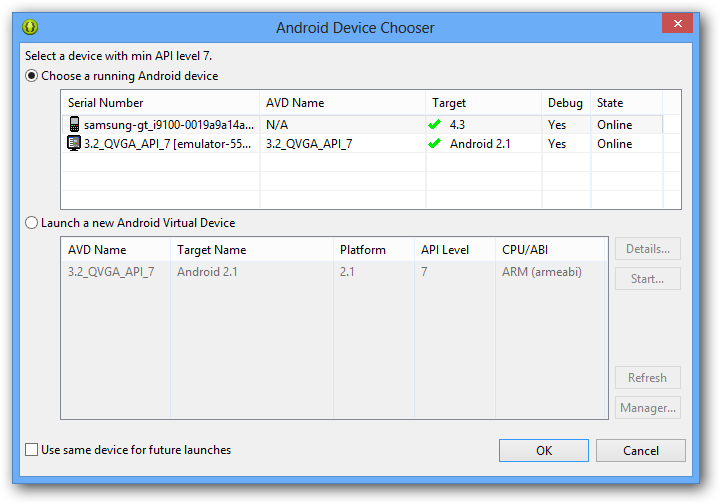
Pour lancer notre application, regardez la barre d'outils d'Eclipse et cherchez l'encart visible à la figure suivante.

Les outils pour exécuter votre codeLes outils pour exécuter votre code

Il vous suffit de cliquer sur le deuxième bouton (celui qui ressemble au symbole « *play* »). Si une fenêtre s'ouvre (voir figure suivante), sélectionnez Android Application.

Sélectionnez « Android Application »

Si vous avez un ou plusieurs AVD lancés et/ou un terminal de connecté, alors cette écran apparaîtra :

Vous pouvez choisir où lancer l'application

A partir de celui-ci, il vous est possible de choisir où vous souhaitez que l'application soit exécutée. Par exemple, moi je désire qu'elle le soit sur mon téléphone, je clique donc sur samsung-gt puis sur ok.

Et voilà ! L'utilisateur vous doit 1 000 000 € !

La présentation peut être différente chez vous en fonction de votre version d'Android

Après vérification auprès d'un avocat, ce n'est pas légalement valable :( .

* Pour avoir des applications fluides et optimisées, il est essentiel de bien comprendre le cycle de vie des activités.
* Chaque écran peut être considéré comme une Activity, qui est constitué d'un contexte et d'une interface graphique. Le contexte fait le lien entre l'application et le système alors que l'interface graphique se doit d'afficher à l'écran des données et permettre à l'utilisateur d'interagir avec l'activité.
* Pour concevoir une navigation impeccable entre vos différentes activités, vous devez comprendre comment fonctionne la pile des activités. Cette structure retirera en premier la dernière activité qui aura été ajoutée.