De Java à Android

version 0.1

David Roche

Note de l'auteur

Ce document est à l'origine destiné aux élèves de 1ère S du lycée G Fichet (Bonneville, Haute Savoie) qui ont choisi de suivre l'atelier scientifique. La plupart de ces élèves ont commencé en seconde leur découverte de la programmation orientée objet à l'aide du logiciel Alice[1]. En début de première, les élèves se sont initiés[2] à Java.
Ce document n'a pas la prétention d'être une "bible" du développement d'application sous Android, beaucoup de notions ne sont pas abordées parce que jugées trop complexes dans le cadre d'une première approche.
Les lecteurs désirant approfondir leurs connaissances devront se tourner vers des ouvrages beaucoup plus complets (mais peut-être plus difficile pour un débutant). Si à la fin de la lecture de ce document, l'élève est pris d'une "irrésistible" envie de se procurer un de ces ouvrages, j'aurais atteint mon objectif !

David Roche
Novembre 2010

[1] voir www.alice.org
voir www.animanum.com/alice
voir http://www.epi.asso.fr/revue/articles/a1006d.htm

[2] voir http://www.epi.asso.fr/revue/articles/a1009b.htm

Sommaire

Chapitre 1
Introduction et installation

Android ?

Android est un système d'exploitation, basé sur un noyau linux, destiné aux appareils mobiles (smartphone principalement, mais c'est en train d'évoluer à vitesse grand V, un nombre important de tablettes tactiles grand format (10 ou 7 pouces), tournant sous Android, devraient sortir dans les prochains mois !).
Android est développé par un consortium : l'Open Handset Alliance.
L'Open Hanset Alliance regroupe beaucoup de sociétés liées aux nouvelles technologies (Intel, HTC, Motorola, Garmin,.....), mais le principal contributeur est Google !
En théorie Android est un système libre (tout comme GNU/Linux), mais sans vouloir entrer dans la polémique qui agite le net, je vais dire que.... c'est un peu plus compliqué que cela, Android est libre sur le papier (licence "Apache Open source licence v2"), mais pas vraiment dans les faits !

Développer des applications sous Android.

Des milliers (voir beaucoup plus !) de personnes (plus ou moins professionnelles) développent des applications sous Android. Beaucoup de ces applications se retrouvent sur le grand supermarché des applications Android mis en place par Google : l'Android Market.
Beaucoup d'applications sont gratuites, d'autres, payantes (en général, quelques euros), certaines, sont de grande qualité, d'autres, beaucoup moins.
Il est relativement "facile" de créer une application (simple) pour Android (ceci explique sans doute le nombre d'applications de qualité très "moyenne" que l'on trouve sur l'Andoid Market !), il "suffit" d'avoir de bonnes bases en programmation Java (normalement cela devrait être votre cas) et d'être prêt à faire les efforts nécessaires pour acquérir certaines méthodes particulières au développement sous Android.
Les programmes destinés à être utilisés sous Android, sont écrits en Java, mais il faut bien avoir à l'esprit qu'un programme fonctionnant sur la machine virtuelle Java développée par Sun (JVM : Java Virtual Machine, celle que nous avons utilisée jusqu'à présent), ne fonctionnera pas sous Android.
Android étant destiné à des appareils mobiles ayant peu de puissances (par rapport à un ordinateur classique), Google a développé sa propre machine virtuelle : Dalvik.

Certaines classes disponibles sous JVM ne le seront pas sous Dalvik, prudence donc.
En plus de Java, le développement d'applications sous Android, demande quelques connaissances en XML.

XML, c'est quoi ça ? Encore un truc compliqué à apprendre ?

Pas de panique, il vous suffira de quelques minutes pour comprendre les bases du XML.
XML signifie eXtensible Markup Language (en français : langage extensible de balisage). XML n'est pas un langage de programmation, il n'y a pas de boucle for, de if, de while,..... .
Il est presque exclusivement utilisé pour stocker (ou transférer d'un programme à un autre) des données (du texte) de façon structurée.
C'est un langage qui utilise des balises (comme le HTML pour ceux qui connaissent !).

Les balises sont ouvrantes, <balise_ouvrante> ou fermantes, </balise_fermante>. Quand vous ouvrez une balise, vous devez à un moment ou un autre la refermer.

Il est possible de mettre du texte entre la balise ouvrante et la balise fermante :
<balise1> Voici le texte inséré </balise1>

Les balises peuvent être imbriquées : on peut insérer un ou plusieurs couples de balises (ouvrante et fermante) entre 2 balises (ouvrante+fermante)
voici un exemple permettant de stocker des informations sur votre cinémathèque :

```
<cinematheque>
	<film>
		<nom>Les deux tours</nom>
		<realisateur>P Jackson</realisateur>
		<annee_sortie>2002</annee_sortie>
	</film>
	<film>
		<nom>Bladerunner</nom>
		<realisateur>R Scott</realisateur>
		<annee_sortie>1982</annee_sortie>
	</film>
</cinematheque>
```

la balise <cinematheque> est appelée "élément racine", la balise <film> est le "premier enfant", etc.
Un fichier XML doit posséder un "élément racine".

Lorsqu'il n'y a pas de texte entre la balise ouvrante et la balise fermante, on peut remplacer :
<balise> </balise> par <balise/> (qui est à la fois la balise ouvrante et la balise fermante)

Une balise peut contenir des attributs :
<balise nom="Toto" prénom="Jean Pierre"/>
nom et prénom sont des attributs qui seront utilisés par un programme tiers.

Un fichier XML doit toujours commencer par une ligne appelée prologue :

<?xml version=*"1.0"* encoding=*"utf-8"*?>
Cette ligne précise la version de XML utilisée (ici, c'est la version 1.0) et le type d'encodage pour le

document (inutile d'entrer dans les détails)

Pour terminer cette introduction sur XML, voici un des fichiers XML que nous allons utiliser très fréquemment lors du développement d'applications sous Android.

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    >
<TextView
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="@string/hello"
      />
</LinearLayout>
```

Nous n'allons pas entrer dans les détails (pour l'instant !), mais vous pouvez d'ores et déjà remarquer, la ligne de prologue (1ère ligne), les balises : LinearLayout et TextView (ouvrante et fermante), la balise TextView (qui est ouvrante et fermante) est comprise entre les balises LinearLayout ouvrante et LinearLayout fermante.
Chaque balise contient des attributs (orientation, layout_width,....)
Vous avez aussi sans doute remarqué :
`xmlns:android="http://schemas.android.com/apk/res/android"`

Cette ligne est liée à la notion de "namespace", tout comme les "`android:`" des lignes suivantes. Nous n'aborderons pas cette notion ici, pour plus d'informations, vous pouvez consulter l'excellent site du zéro et plus particulièrement le tutoriel consacré au langage XML :
http://www.siteduzero.com/tutoriel-3-33440-le-point-sur-xml.html

Installation du SDK et configuration d'Eclipse

Pour développer des applications Android, nous allons utiliser Eclipse (comme pour le développement d'applications Java "classique")
Nous allons devoir télécharger 2 autres composants :

- le SDK (*software development kit*) Andoid qui va contenir tous les "outils" nous permettant de "construire" des applications sous Android.
- Le plugin "Android pour Eclipse" ADT, qui adapte Eclipse au développement d'applications sous Android.

Le SDK doit être téléchargé à l'adresse suivante:
http://developer.android.com/sdk/index.html

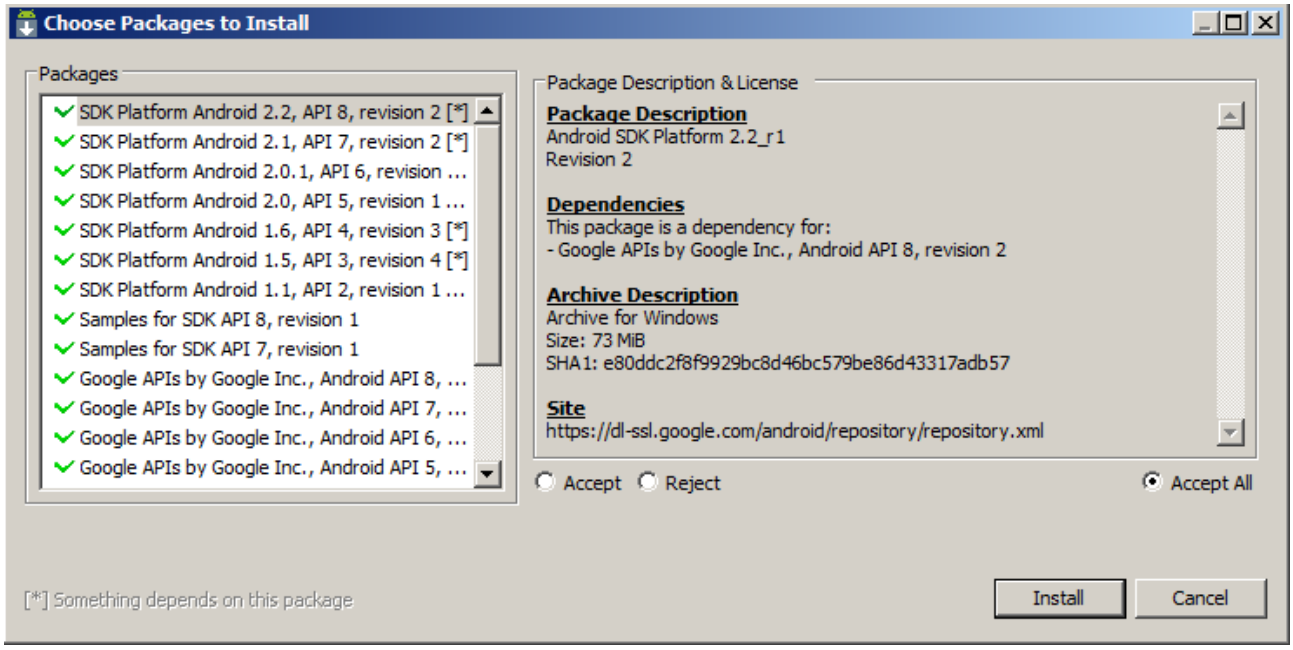
Platform	Package	Size	MD5 Checksum
Windows	android-sdk_r07-windows.zip	23669664 bytes	69c40c2d2e408b623156934f9ae574f0
Mac OS X (intel)	android-sdk_r07-mac_x86.zip	19229546 bytes	0f330ed3ebb36786faf6dc72b8acf819
Linux (i386)	android-sdk_r07-linux_x86.tgz	17114517 bytes	e10c75da3d1aa147ddd4a5c58bfc3646

Choisissez le fichier "android-sdk_r07-windows"
(attention pour les versions à venir d'Android, le fichier changera de nom, ne soyez pas surpris si vous ne trouvez plus "android-sdk_r07-windows.zip")

Une fois téléchargée, ouvrez le fichier "android-sdk_r07-windows.zip", "dézippez" le dossier "android-sdk-windows", et installez le SDK (double clic sur le fichier SDK Manager).

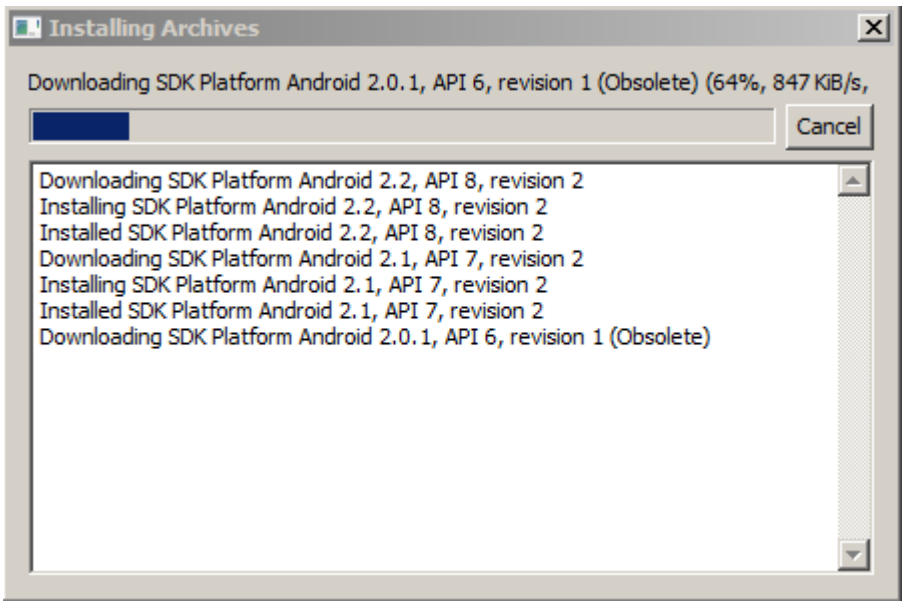
Dossier android-sdk-windows

Vous devriez voir apparaître la fenêtre suivante :

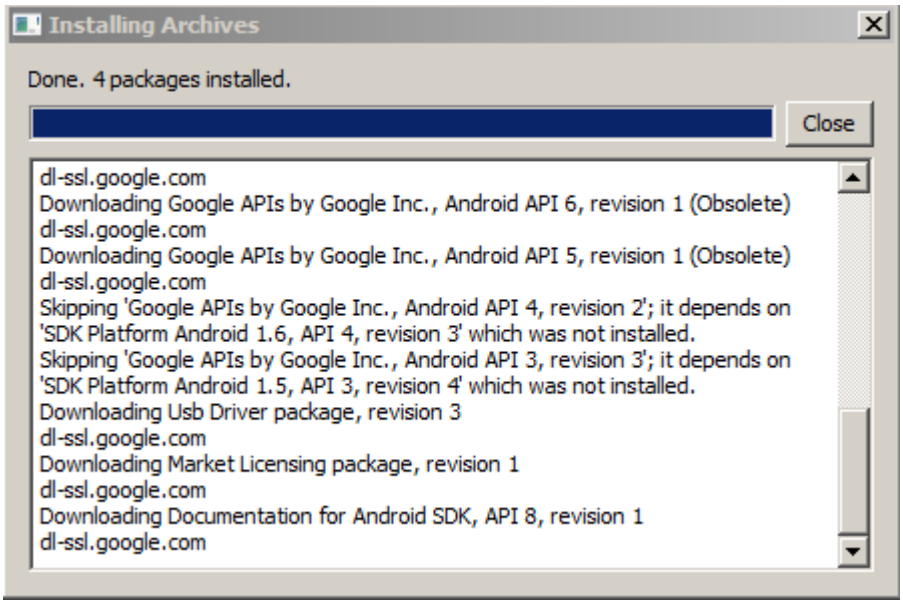


Vérifiez que "Accept All" est bien coché et cliquez sur "Install"

Ensuite, il n'y a plus qu'à attendre.



Parfois, longtemps... (cela dépend de la qualité de la connexion internet)
Une fois l'installation terminée, cliquez sur le bouton close

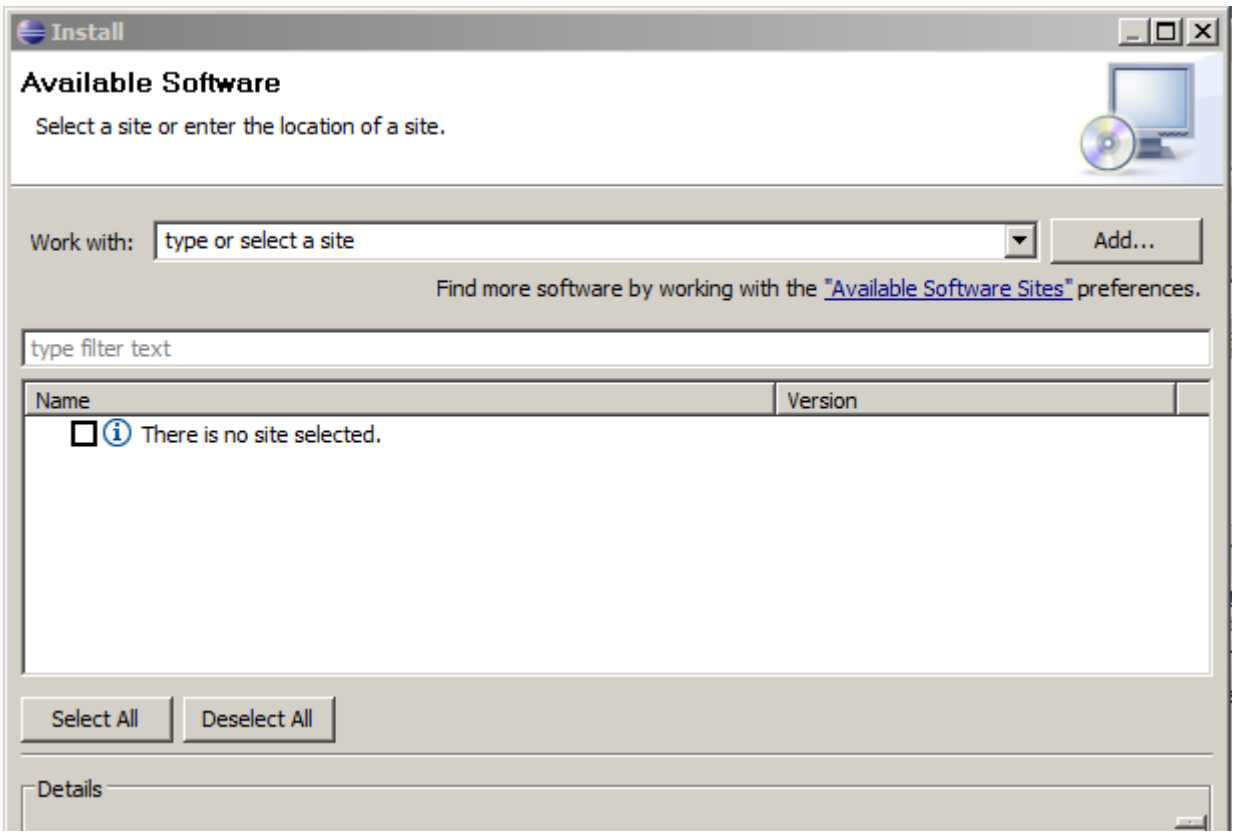


Fermez ensuite la fenêtre "Android SDK and AVD Manager"

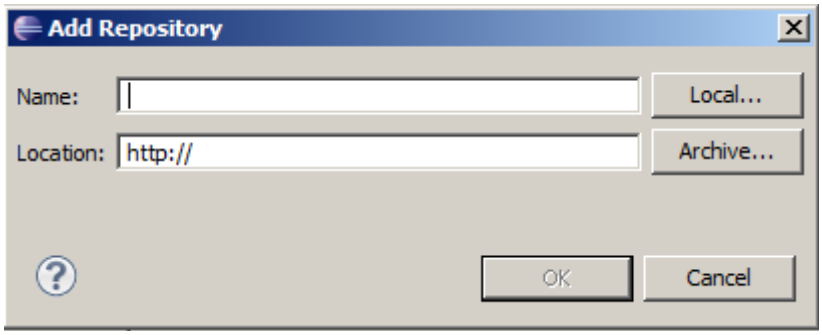
Le SDK est installé, passons maintenant à l'installation du plugin pour Eclipse, le module ADT.

Ouvrez Eclipse et choisissez "Install New Software..." dans le menu "Help"

Vous devriez voir la fenêtre suivante s'ouvrir:



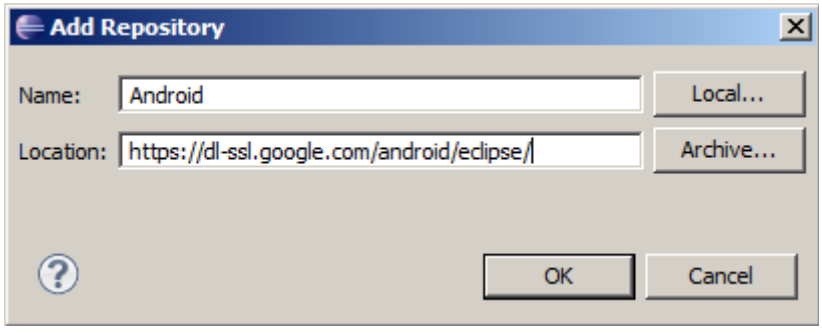
Cliquez sur le bouton "Add..."



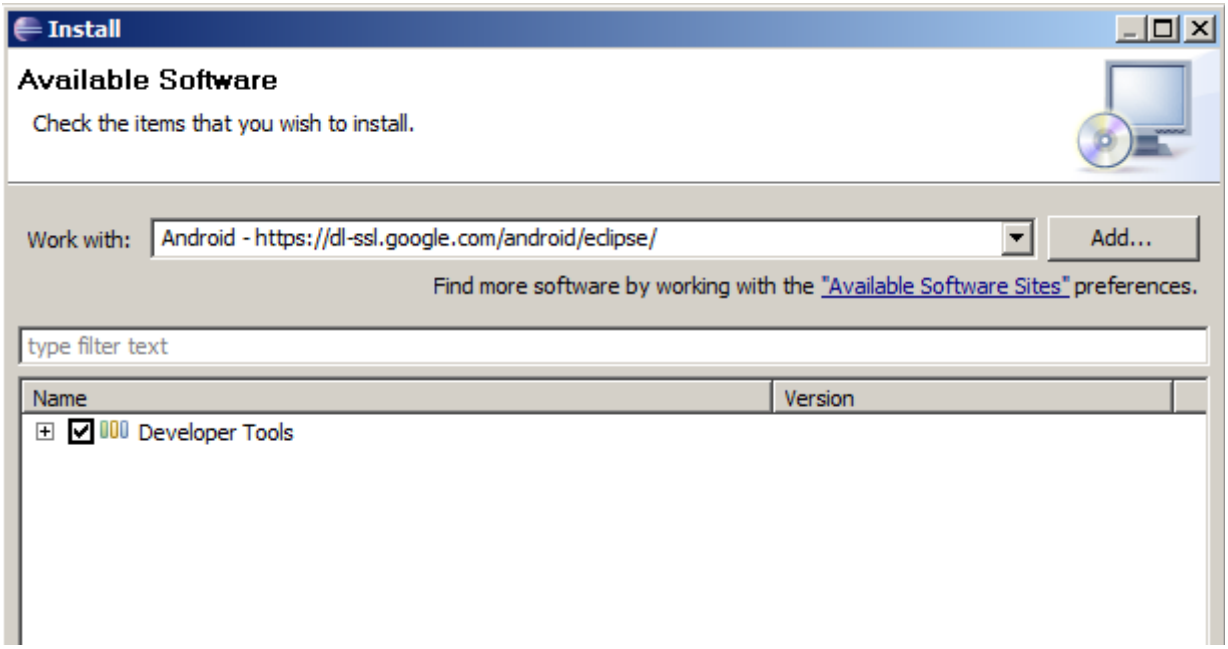
Dans la fenêtre "Name:", tapez "Android". Dans la fenêtre "Location:" entrez l'adresse suivante :

```
https://dl-ssl.google.com/android/eclipse/
```

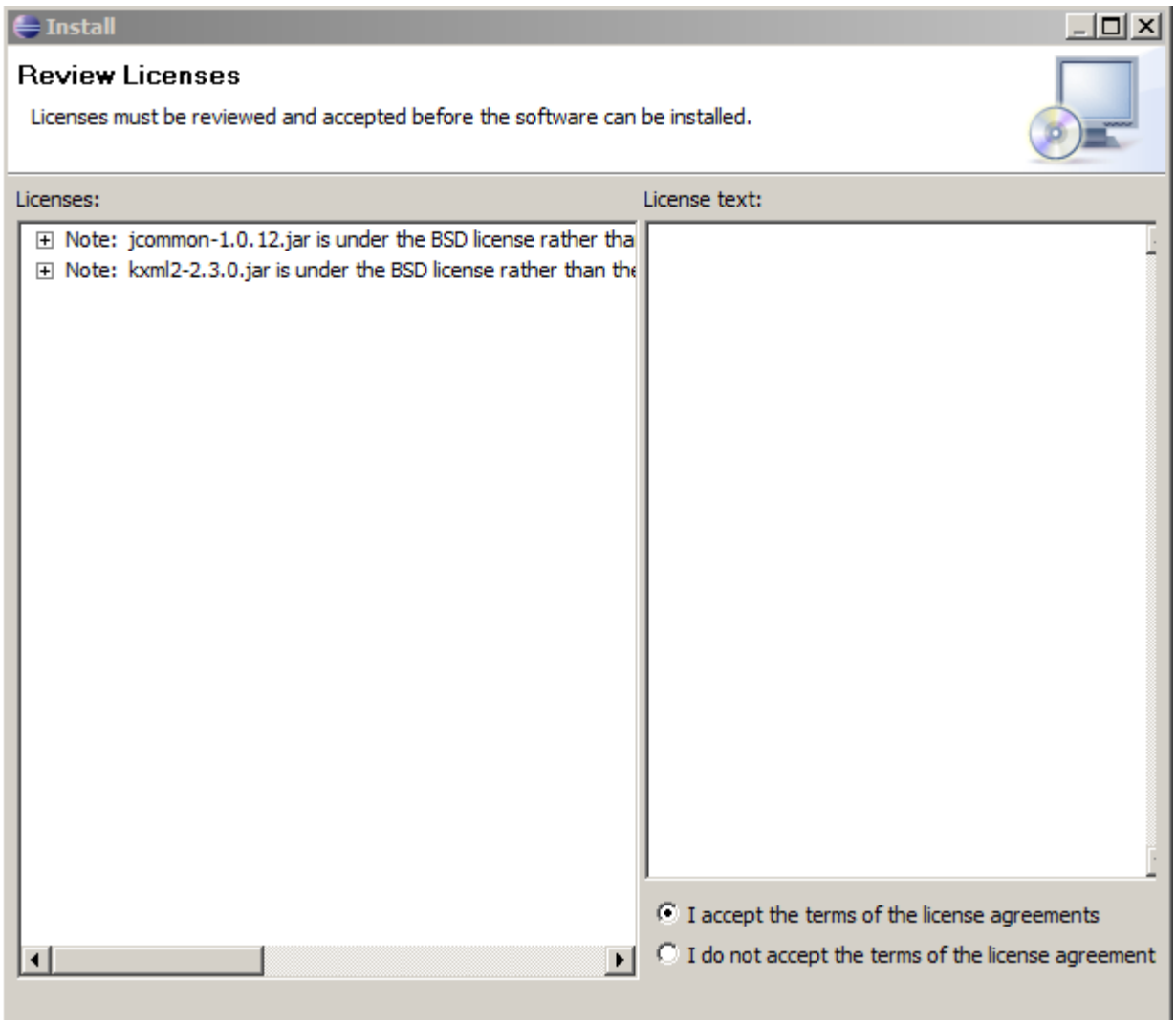
Vous devriez donc obtenir ceci:



Cliquez sur OK

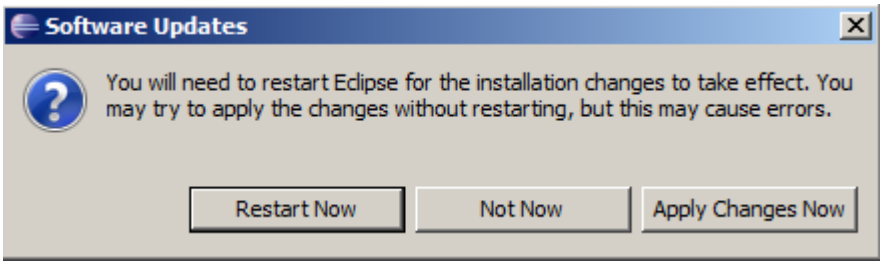


Après avoir coché "Developer Tools", cliquez sur "Next >" 2 fois.



Cochez "I accept the terms of the license agreements", puis cliquez sur "Finish"

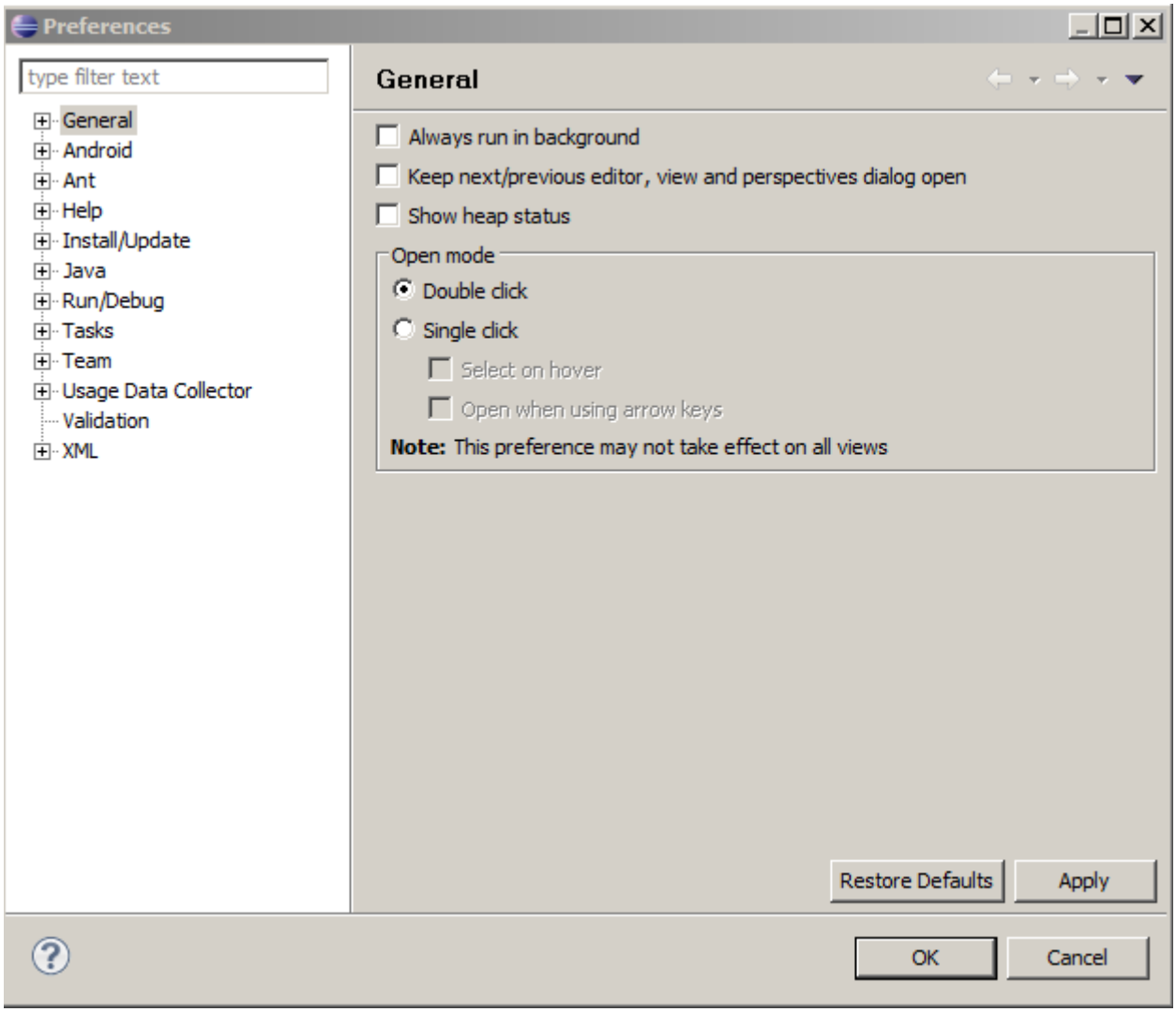
Après encore un peu de patience (et peut-être une fenêtre d'avertissement, si c'est le cas, cliquez sur OK), Eclipse devra être redémarré.



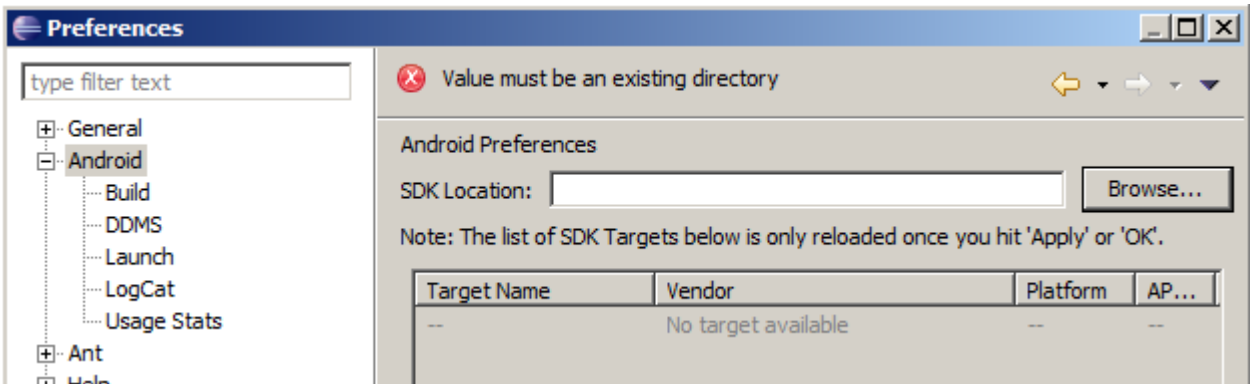
Cliquez donc sur "Restart Now"

Eclipse se relance.

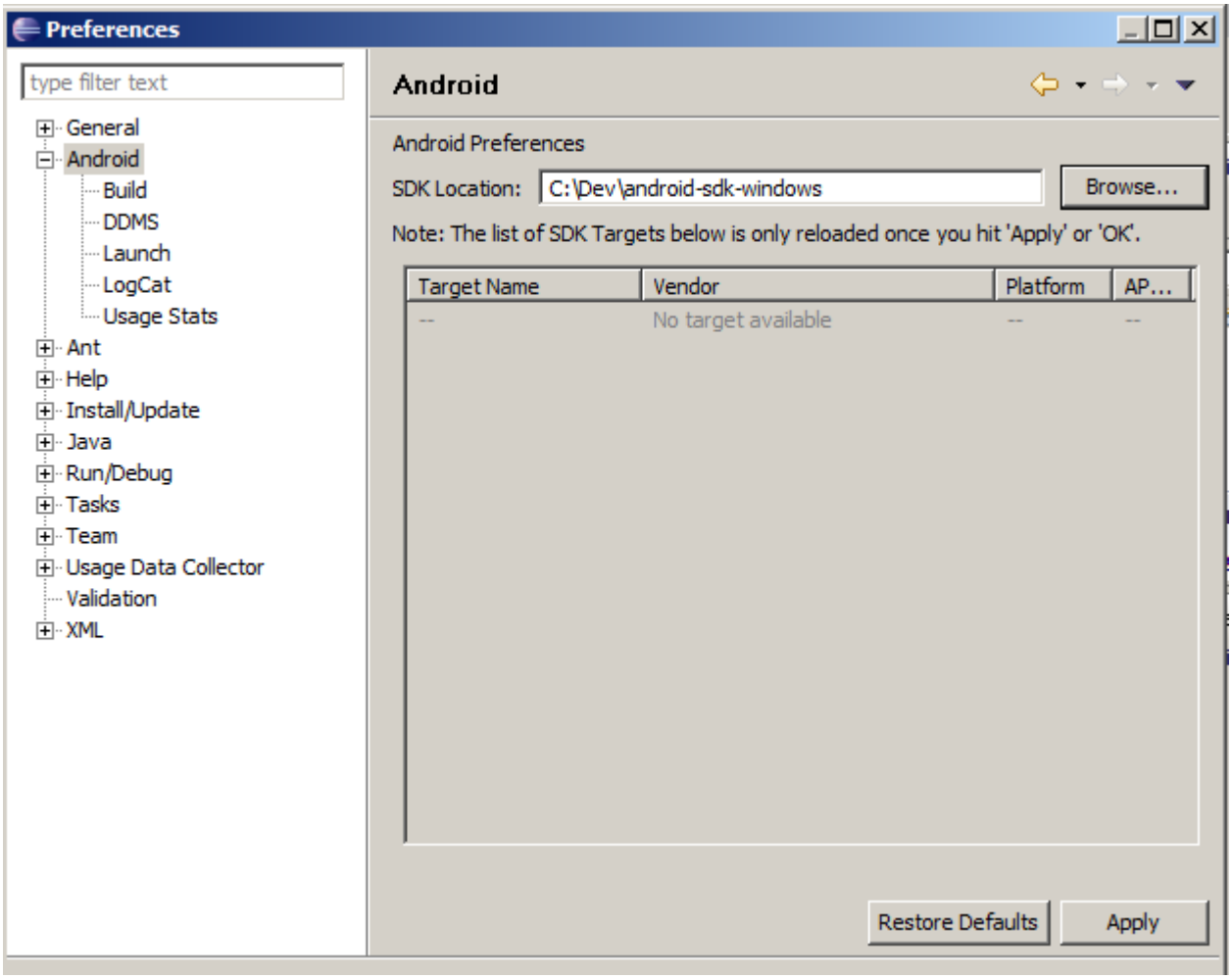
Une fois qu'Eclipse a redémarré, cliquez sur "Preferences" dans le menu "Window".



Cliquer sur "Android" dans le menu

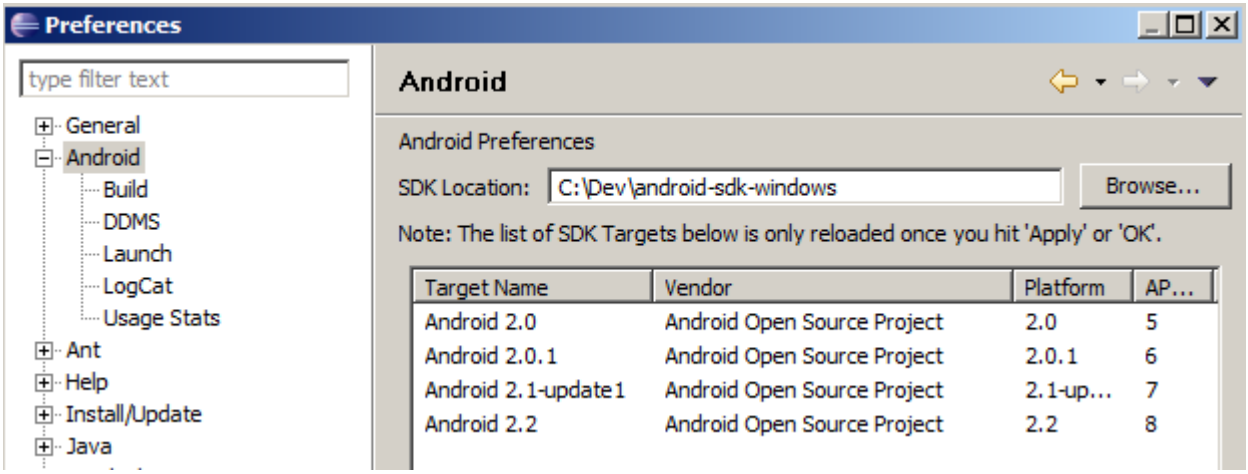


Cliquez sur "Browse.." pour indiquer à Eclipse le chemin d'accès au dossier "android-sdk-windows"



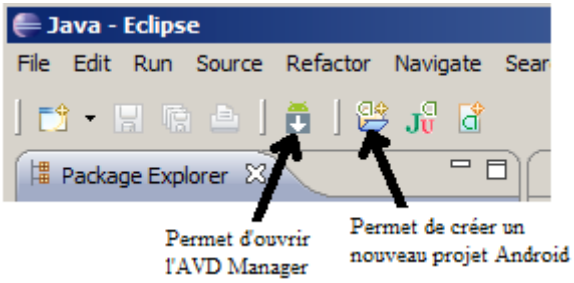
(Attention "mon" chemin ne sera pas forcément le votre)

N'oubliez surtout pas de cliquer sur Apply



Cliquez sur OK, l'installation est maintenant terminée.

Normalement, plusieurs icônes ont dû faire leur apparition dans Eclipse.

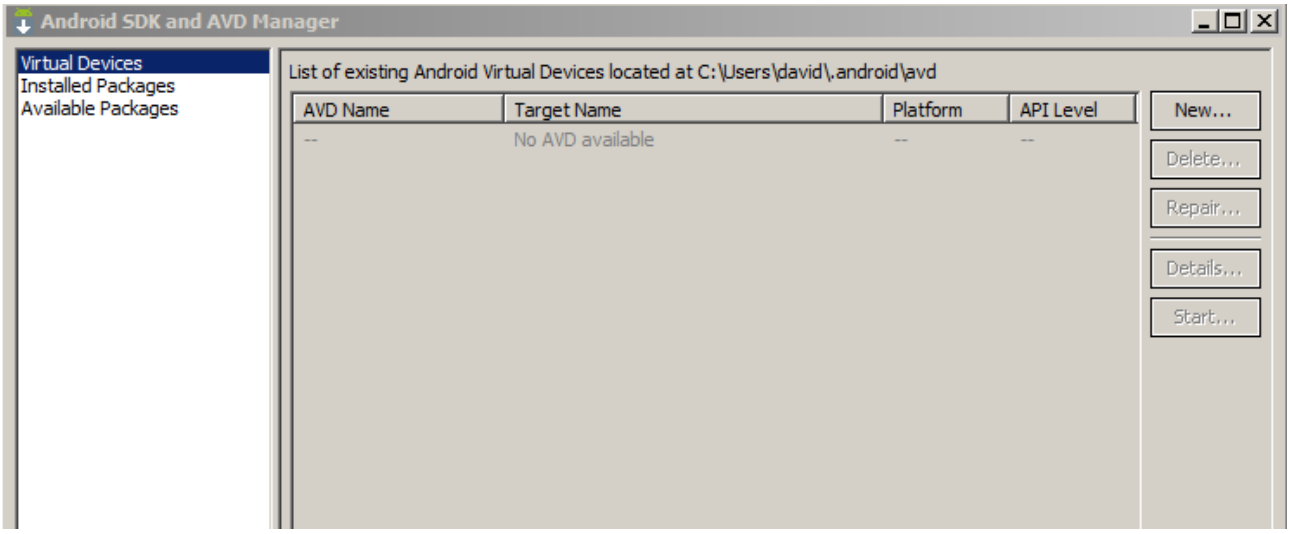


La méthode d'installation du SDK et du plugin Eclipse pouvant évoluer avec le temps, n'hésitez pas à consulter la documentation officielle sur :
http://developer.android.com/sdk/index.html

Création d'un émulateur sous Eclipse

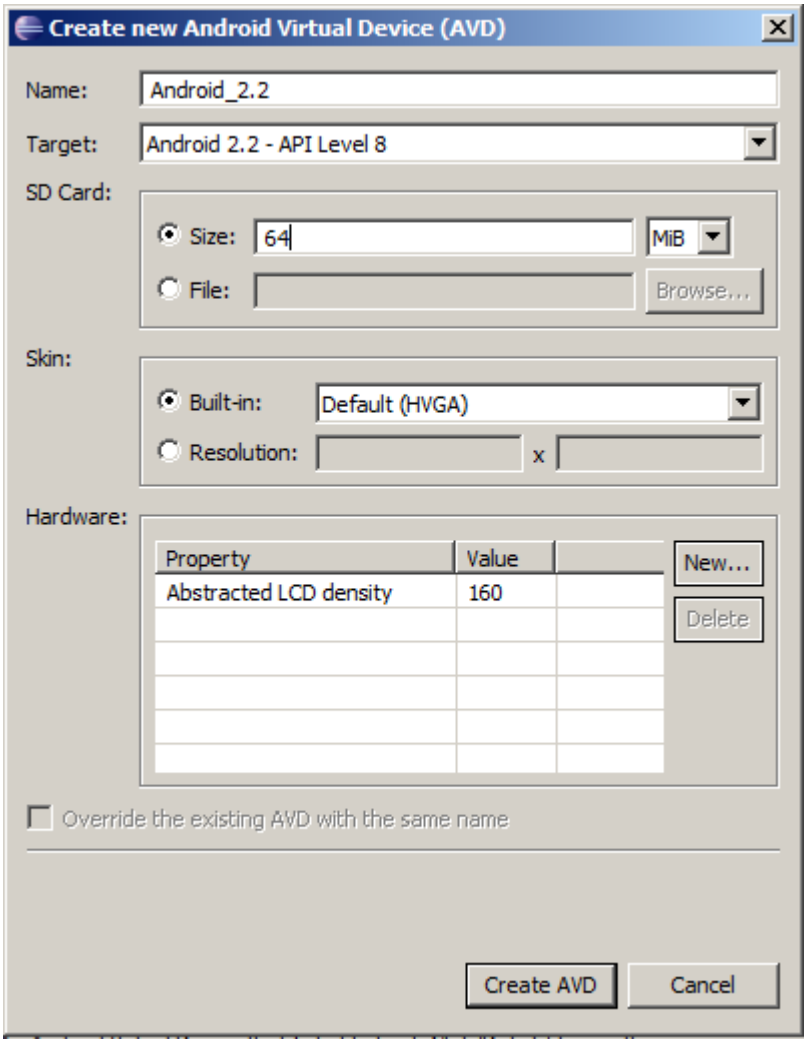
Vous n'avez sans doute pas de smartphone Android à votre disposition, pas d'inquiétude, les concepteurs d'Andoid ont pensé à vous.
Il est possible de créer un émulateur de smartphone (simulateur de smartphone) dans Eclipse, qui va vous permettre de tester vos applications (à quelques exceptions près).

Pour créer un émulateur, ouvrez l'AVD Manager.



Cliquez sur "New..."

Une nouvelle fenêtre apparaît alors



Complétez les champs "Name", "Target" et "Size" comme indiqué ci-dessus puis cliquez sur "Create AVD"

Pour tester votre émulateur, dans la fenêtre "Android SDK and AVD Manager" et après avoir sélectionné l'émulateur que vous venez de créer, cliquez sur "Start".

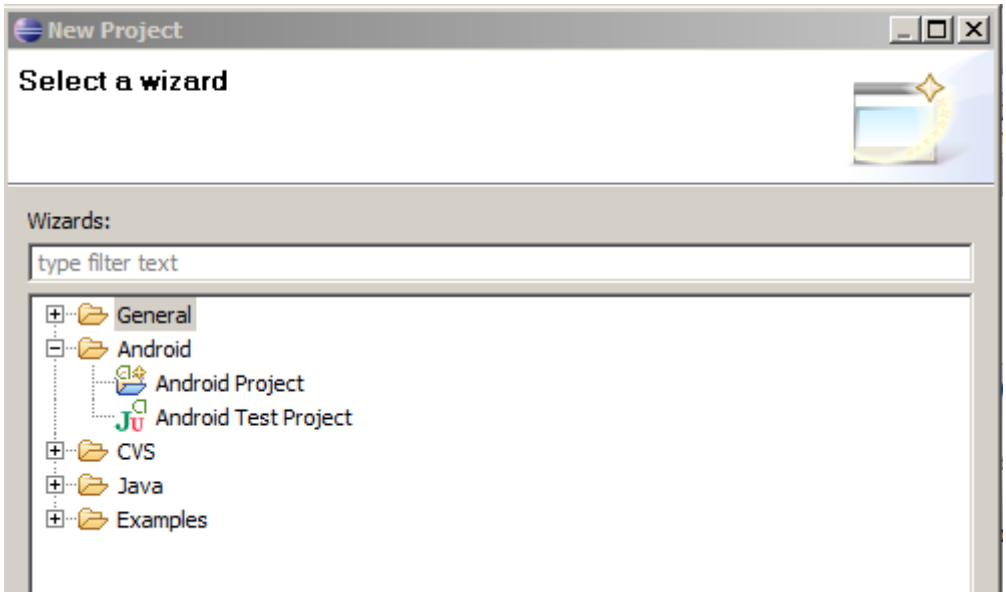
Au bout de quelques minutes (le premier démarrage est très long !), vous devriez voir apparaître ceci:

Vous pouvez commencer à "jouer" avec votre téléphone virtuel sous Android 2.2

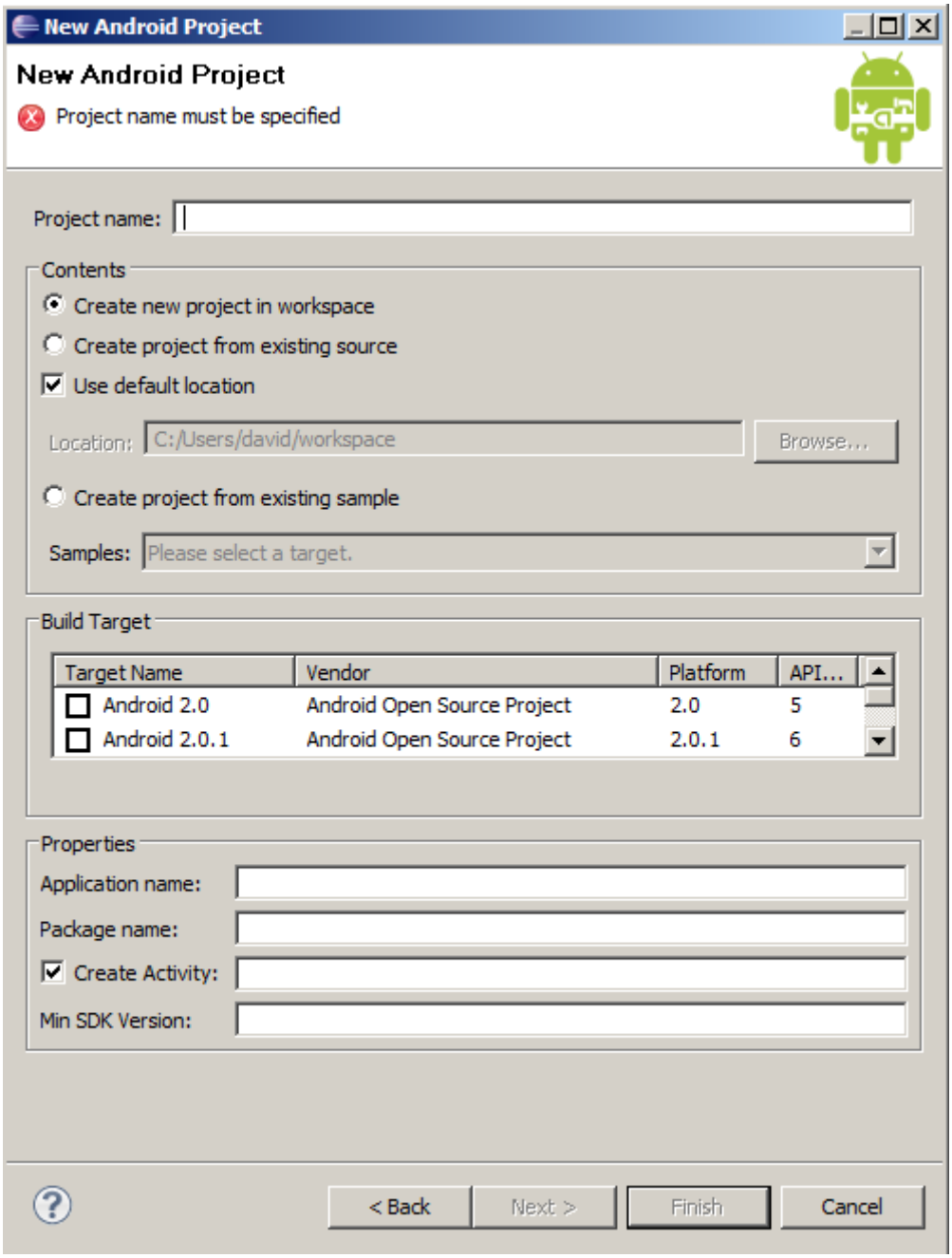
Voilà tout est prêt, passons aux choses sérieuses !

Chapitre 2
1ère application

Nous allons créer notre 1ère application, pour cela, dans Eclipse, dans le menu "File" choisissez "New" -> "Project"



Sélectionnez alors "Android Project", puis cliquez sur "Next >", une autre fenêtre apparaît :



Complétez le champ "Project name :" avec le nom de votre projet (pour nous cela sera "ex2.1").

Dans "Build Target": choisissez Android 2.2 (ou plus si un SDK plus récent est sorti !)

Dans "Properties" :

"Application name:" : comme son nom l'indique, donnez un nom à votre application

"Package name" : ici, c'est un peu plus compliqué. Imaginez que vous décidiez d'écrire une application qui porte le nom de "Toto", tellement fier de votre oeuvre, vous publiez votre application sur l'Android Market. Mais à l'autre bout du monde (en Australie par exemple, mais bon, cela n'a aucune importance !) Mr Smith, décide lui aussi de publier une application "Toto" (mais qui, à part le nom, n'a aucun rapport avec la vôtre) : Problème ?
Pas vraiment, car, en théorie, vos 2 applications n'auront pas le même "Package name". Un package

name a, en général (mais ce n'est pas une obligation), la forme suivante :
org.education.lgf.atelier
Oui, votre première impression est la bonne, cela ressemble beaucoup a une adresse internet à l'envers. A la place de "org", vous auriez pu mettre "fr" ou "com". Notre package name indique que notre application à un rapport avec l'éducation, avec le lycée G Fichet (lgf) et avec l'atelier scientifique.
Il y a très peu de chances pour que notre ami australien, Mr Smith, ait le même package name. Nos applications pourront donc être distinguées.

La case "Create Activity" doit être cochée, vous devrez donner un nom à votre activité dans le champ qui suit.
Mais, qu'est-ce qu'une activité ?

L'activité (Activity) est l'unité de base des applications sous Android. Pour résumer (un peu brutalement), nous allons dire qu'à un écran correspond une activité (et vice versa). Si une application se compose de plusieurs écrans, en général, on aura une activité pour chaque écran. Une activité est une classe qui étend la classe de base "Activity", nous aurons donc un "extends Activity"

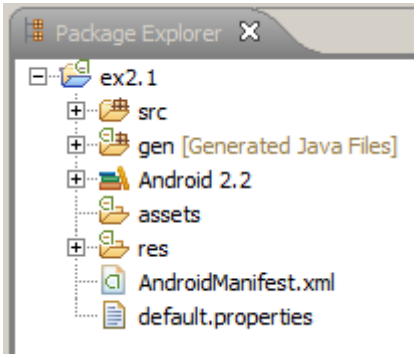
Nous appellerons ici notre activité "Principale".

Enfin, laissez le champ "Min SDK Version" vide (pour l'instant).

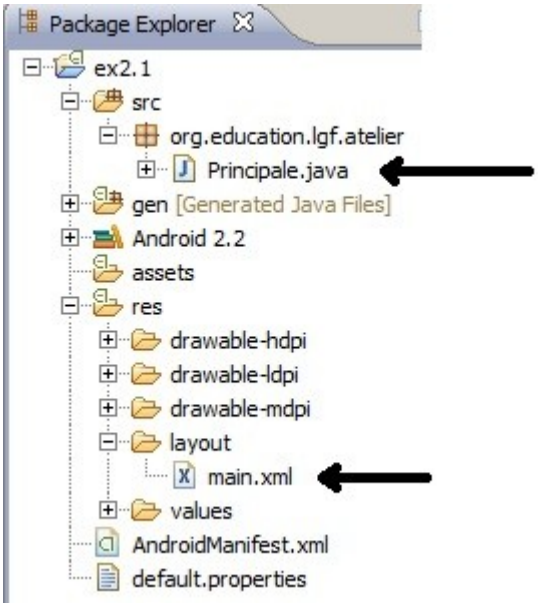
Cliquez sur "Finish", vous venez de créer votre premier projet Android, félicitations ! (en cas d'erreur, fermez et redémarrez Eclipse)

Étude de la première application

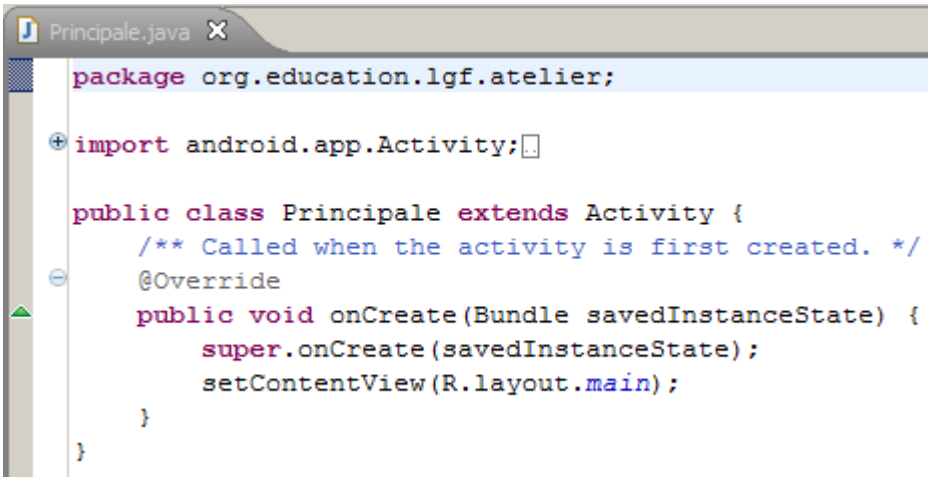
Observons attentivement la fenêtre "Package Explorer" d'Eclipse



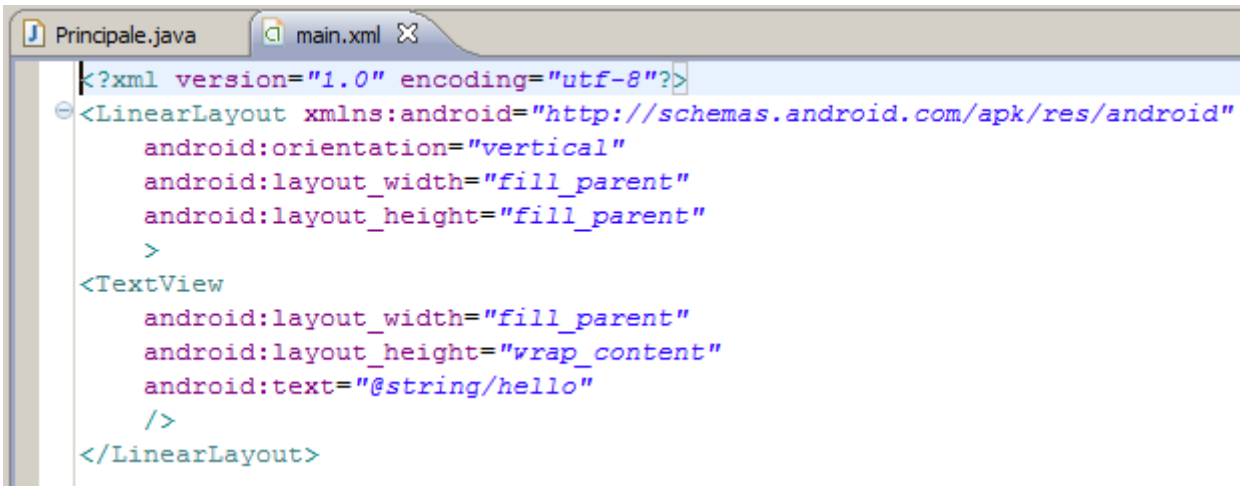
Eclipse a créé une arborescence (complexe), dans un premier temps, nous allons uniquement nous intéresser à 2 fichiers : "Principale.java" et "main.xml"



Double cliquez sur "Principale.java", vous devriez alors obtenir ceci :

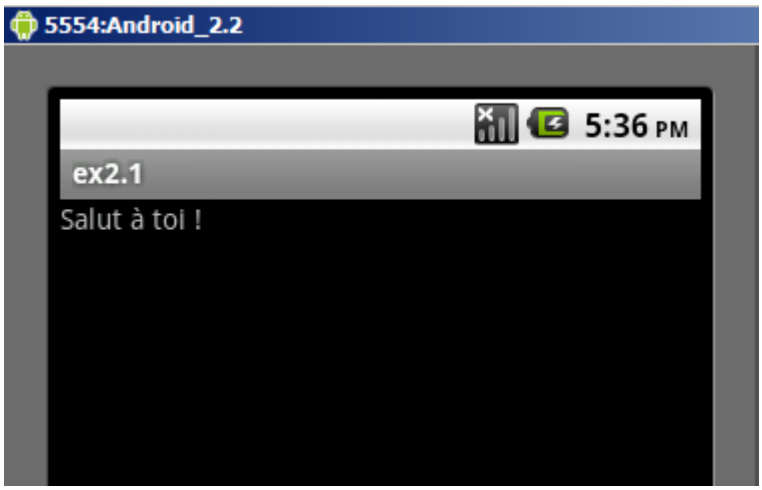


Double cliquez sur "main.xml", vous devriez obtenir ceci :



Comme vous avez pu le remarquer, un projet Android créé par Eclipse n'est pas une coquille vide, il y a déjà du code Java dans le fichier "Principale.java" et XML dans le fichier "main.xml". Nous allons nous baser sur ce code, écrit par défaut lors de la création d'un projet Android sous Eclipse, pour essayer de comprendre les bases du développement d'applications sous Android.

Par souci de simplification, vous allez légèrement modifier le fichier "main.xml" en remplaçant à la dixième ligne `android:text="@string/hello"` par `android:text="Salut à toi !"`
Testez ce projet en cliquant sur "Run" dans le menu "Run" d'Eclipse. Après une attente plus ou moins longue (je vous conseille de ne pas fermer l'émulateur entre 2 tests), et après avoir déverrouillé le "téléphone", vous devriez obtenir ceci :



Commençons par étudier le fichier "main.xml"

Ex 2.1 main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    >
<TextView
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="Salut à toi !"
    />
</LinearLayout>
```

Ce fichier va nous permettre de définir l'interface graphique de votre application (les boutons, les textes, les images.....). Pour tout vous dire, il est aussi possible de coder cette interface en java, mais nous n'aborderons pas cette possibilité dans ce cours.

Étudions en détail ce fichier:

```
<?xml version="1.0" encoding="utf-8"?>
```
Rien à signaler, c'est le prologue

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    >
```

Nous ouvrons une balise nommée "LinearLayout". Qu'est-ce qu'un LinearLayout ?
Un LinearLayout est un layout, mais qu'est ce qu'un layout ?
Pour répondre à cette question, je pourrai vous renvoyer vers le chapitre 6 de "D'Alice à Java", mais pour résumer, un layout est un conteneur qui va accueillir vos boutons, images, champs de texte,... bref tout ce que l'on appelle des widgets.
Il existe plusieurs types de layout, nous utilisons ici le plus courant le "LinearLayout".
Le choix du type layout (et des attributs qui lui sont associés) est très important, car c'est lui qui "gère" le positionnement des widgets (bouton, champ de texte,...) sur l'écran.
Nous reviendrons plus en détail dans le prochain chapitre, qui se nomme d'ailleurs "layouts et widgets" (nous expliquerons alors les attributs "`orientation`", "`layout_width`" et "`layout_height`" de la balise ouvrante LinearLayout).

La suite du fichier main.xml :

```
<TextView
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="Salut à toi !"
  />
```

Entre la balise ouvrante et fermante du LinearLayout, nous allons placer des widgets. Ici, nous allons placer un seul widget : TextView. La balise est à la fois ouvrante et fermante.
Comme son nom l'indique, TexView permet d'afficher du texte à l'écran (plus précisément le texte attribué à "text" : `text="Salut à toi !"`)
Comme pour les layouts, nous reviendrons en détail sur les widgets (notamment sur les attributs "`layout_width`" et "`layout_height`") dans le prochain chapitre.

Enfin pour terminer, la dernière ligne du fichier main.xml ne devrait pas vous poser de problème, c'est la balise fermante du LinearLayout.

Passons maintenant à l'étude du fichier "Principale.java"

```
package org.education.lgf.atelier;

import android.app.Activity;
import android.os.Bundle;

public class Principale extends Activity {
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
    }
}
```

Passons sur les imports et le package, qui sont obligatoires quelque soit votre projet Android ("`import android.app.Activity`" permet d'utiliser la classe "Activity", "`import android.os.Bundle`" permet d'utiliser un objet de type "Bundle" comme paramètre de la méthode "onCreate").

```
public class Principale extends Activity {
```
Avec cette ligne nous créons une classe qui se nomme "Principale" (notez que c'est le nom donné à notre activité) qui hérite de la classe Activity.

Je reviendrai un peu plus loin sur le : `@Override`

```
public void onCreate(Bundle savedInstanceState) {
```
"onCreate" est une des méthodes de la classe Activity. Nous avons donc ici affaire à une redéfinition de la méthode "onCreate" (si vous avez besoin de vous rafraîchir la mémoire sur cette notion de redéfinition, consultez le chapitre 5 du document "d'Alice à Java"). A ce propos, nous pouvons maintenant revenir sur le `@Override` :
Le @override n'est pas vraiment une instruction Java, c'est une annotation.
Avec "onCreate", nous avons bien affaire à une redéfinition pas à une surcharge (nous avons dans les 2 méthodes "onCreate" (celle de la classe parente "Activity" et celle de la classe qui hérite d'Activity, "Principale"), un seul paramètre : une instance (qui se nomme "savedInstanceState") de la classe "Bundle", nous reviendrons dessus un peu plus loin). Si par hasard, sans le vouloir, en modifiant le nombre ou le type de paramètres de la méthode "onCreate", vous ne faites plus une redéfinition, mais une surcharge, votre programme ne fonctionnera pas normalement, mais comme aucune véritable erreur n'aura était commise, aucun message ne sera renvoyé par le compilateur. @override, évite ce genre de chose : en mettant cette annotation, vous "dites" au compilateur :"Attention, la méthode qui suit ("onCreate" dans notre cas) est une redéfinition, pas une surcharge, si un petit malin s'amuse à faire une surcharge, il faudra lui dire !".
@override n'est pas obligatoire, mais fortement conseillé !

L'instance "savedInstanceState" de la classe "Bundle" permet de retrouver votre activité dans l'état où vous l'avez laissé. Oui, je sais, ce n'est pas très clair, mais ce n'est pas fondamental pour comprendre la suite, je ne chercherai donc pas à insister là-dessus. Nous nous contenterons de dire que la méthode "onCreate" doit recevoir un objet de type "Bundle" comme paramètre, vous devriez pouvoir vous en sortir uniquement avec cela !

```
super.onCreate(savedInstanceState);
```
Ici nous appelons la méthode "onCreate" définie au niveau de la classe parente, "Activity" (toujours avec l'objet de type "Bundle", "savedInstanceState" comme paramètre).
J'attire votre attention sur le fait que la méthode "onCreate" de la classe "Activity" est une méthode qui comporte sans doute plusieurs dizaines (voir centaines ?) de lignes. Grâce à la ligne `super.onCreate(savedInstanceState);`, tout se passe comme si ces dizaines de lignes étaient insérées dans notre redéfinition de la méthode "onCreate". Mais "la magie" de la POO permet de ne pas s'en préoccuper.
Mais que fait cette méthode "onCreate" ?
Pour faire simple (nous préciserons les choses dans le chapitre "Cycle de vie d'une activité" dans la 2ème partie), la méthode "onCreate" sera la première méthode exécutée au moment du lancement de notre activité "Principale".

```
setContentView(R.layout.main);
```
"setContentView" est une méthode static de la classe "Activity" (donc ici aussi de la classe "Principale" puisque "Principale" hérite d'"Activity").

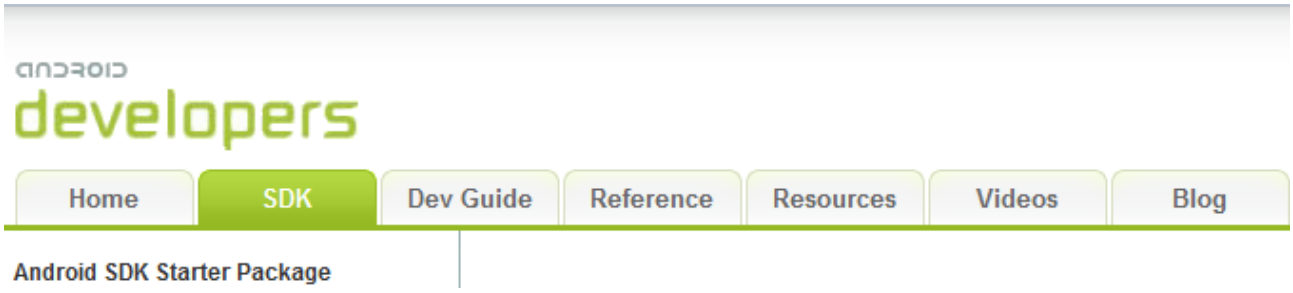
Quelle est le rôle de cette méthode "setContentView" ?
Vous avez sans doute remarqué les mots "layout" et "main" dans le paramètre passé à cette méthode "layout" correspond au répertoire layout de l'arborescence de notre projet Android, "main" correspond au fichier qui se trouve dans ce répertoire layout, c'est-à-dire le fichier "main.xml" que

nous avons étudié un peu au-dessus.
Cette méthode indique au système qu'il faudra utiliser le fichier main.xml se trouvant dans le répertoire layout pour "fabriquer" l'interface graphique de notre activité.
En fait les choses sont un peu plus compliquées que cela (avec notamment la présence de ce mystérieux "R"), mais pour l'instant, nous nous contenterons de cette explication : pour indiquer au système que nous voulons utiliser l'interface graphique définie dans le fichier "toto.xml" (ce fichier se trouvant dans le répertoire "res/layout" de l'arborescence de notre projet), il suffira d'écrire :

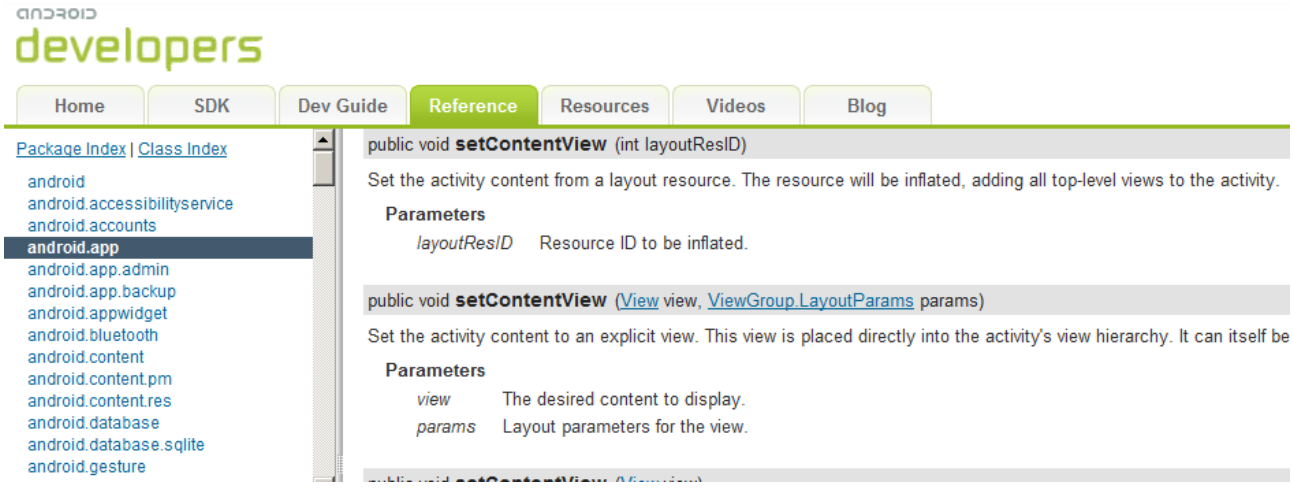
```
setContentView(R.layout.toto);
```

La documentation du SDK

Pour terminer ce chapitre, quelques mots sur la documentation officielle du SDK.
Ce document n'a pas la prétention de faire de vous des pro d'Android, il est même très probable qu'un jour ou l'autre, il ne vous suffira plus pour pouvoir poursuivre votre progression. Vous pourrez alors acheter des livres (je donne une liste d'ouvrage à la fin de ce document), mais il sera aussi très judicieux de consulter la documentation officielle du SDK sur internet (on la trouve aussi "offline" dans le répertoire d'installation du SDK) : http://developer.android.com/sdk/index.html



Vous trouverez sans doute votre bonheur dans cette documentation officielle (en anglais, mais cela ne devrait pas vous poser de problème ;)) . N'hésitez pas à la parcourir dès maintenant pour vous familiariser avec.
J'attire particulièrement votre attention sur l'onglet "Reference" qui référence toutes les classes et toutes les méthodes disponibles.



Vous avez ci-dessus la documentation de la méthode "setContentView".

Même si cela vous paraît un peu rébarbatif au départ, astreignez-vous à consulter cette documentation à chaque fois que nous rencontrerons une nouvelle méthode.

Chapitre 3
les layouts

Un layout (ou conteneur en bon français) est un objet, au sens POO de terme, qui va contenir des widgets (ou d'autres layouts) comme les Texview (vus au chapitre précédent), les boutons, ... et vous permettre d'organiser la disposition de ces widgets sur l'écran comme bon vous semble.
Il existe plusieurs types de layout (LinearLayout, RelativeLayout, TableLayout,...), chacun avec des caractéristiques différentes. Par souci de simplification nous allons ici uniquement étudier le LinearLayout (rassurez-vous le LinearLayout est le plus couramment utilisé, les autres étant, plus....exotiques !).
Comme dit plus haut, les layouts sont des classes, des classes qui héritent toutes d'une même classe parente, la classe ViewGroup.
Le LinearLayout va aligner les widgets les uns après les autres à la verticale (en colonne) si vous choisissez "`android:orientation="vertical"`" et à l'horizontale (en ligne) si vous choisissez "`android:orientation="horizontal"`"
ex 3.1 : main.xml

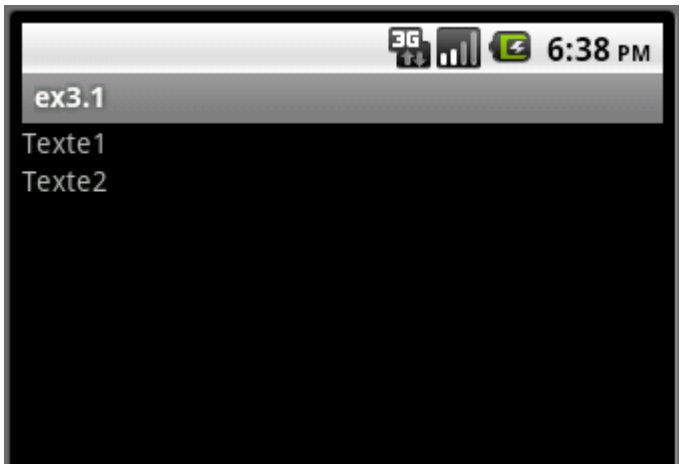
```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    >
<TextView
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="Texte1"
    />
<TextView
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="Texte2"
    />
</LinearLayout>
```

ex 3.1 : Principale.java

```
package org.education.lgf.atelier;

import android.app.Activity;
import android.os.Bundle;

public class Principale extends Activity {
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
    }
}
```

N. B. Dans tout ce chapitre, "Principale.java" ne sera pas modifié.

Ex 3.1 : résultat



Modifions main.xml : "`android:orientation="horizontal"`" (pour que cela fonctionne correctement, nous avons aussi modifié 2 autres lignes, trouvez-les !)

ex :3.2 : main.xml

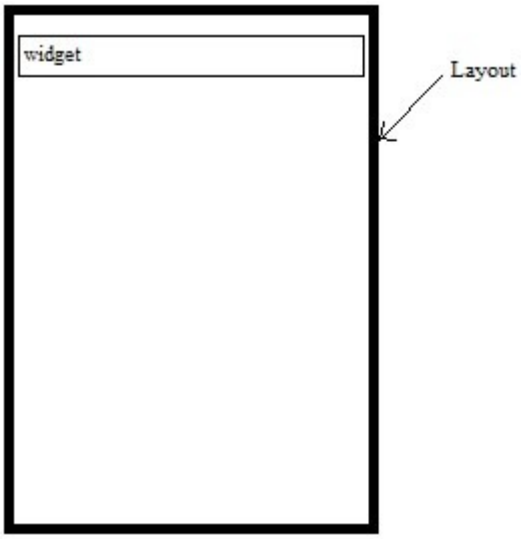
```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="horizontal"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    >
<TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Texte1"
    />
<TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Texte2"
    />
</LinearLayout>
```

ex 3.2 : résultat

Avez-vous trouvé les 2 autres modifications de l'exemple 3.2 ?
Dans les 2 balises TextView, nous avons remplacé "`android:layout_width="fill_parent"`" par "`android:layout_width="wrap_content"`".
A quoi correspond ce "layout_width" (et ce "layout_height") ?
Un widget peut soit :

- prendre uniquement la place dont il a besoin
- prendre toute la place disponible

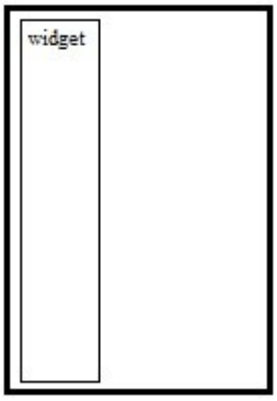
Si vous choisissez "`android:layout_width="fill_parent"`" le widget prendra toute la place restante disponible en largeur



Si vous choisissez "`android:layout_width="wrap_content"`", le widget prendra uniquement la place dont il a besoin (en largeur) (pour TextView, la place du texte). Le reste de la place reste disponible pour d'autres widgets.

Vous comprenez maintenant pourquoi nous avons remplacé "`android:layout_width="fill_parent"`" par "`android:layout_width="wrap_content"`" dans l'exemple 3.2 : Avec `"fill_parent"`, il ne restait plus de place "à côté" de "Texte1" pour accueillir "Texte2", "Texte1" prenant toute la place en largeur.

Il existe la même la chose en hauteur : "`android:layout_height="fill_parent"`", le widget prendra toute la place restante disponible en hauteur.



Que se passera-t-il si dans l'exemple 3.1 vous remplacez :

```
<TextView
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="Texte1"
    />
```

par

```
<TextView
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:text="Texte1"
    />
```

Réponse :

Pourquoi ?

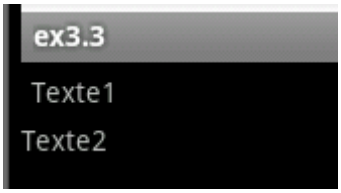
Padding

Pour l'instant nos widgets sont serrés les uns contre les autres, ce qui n'est pas très esthétique. Vous pouvez augmenter l'espace entre vos widgets en ajoutant une ligne au fichier "main.xml" : "`android:padding="5px"` ". Cette ligne va créer une bordure de 5 pixels de largeur autour du widget.

ex 3.3 : main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    >
<TextView
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="Texte1"
    android:padding="5px"
    />
<TextView
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="Texte2"
    />
</LinearLayout>
```

ex 3.3 : résultat



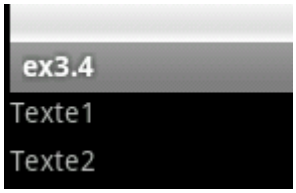
Vous pouvez constater la bordure invisible de 5 pixels autour de "Texte1".

Il est aussi possible de créer un décalage uniquement sur l'un des côtés du widget (à gauche avec paddingLeft, à droite avec paddingRight, en haut avec paddingTop ou en bas avec paddingBottom)

ex 3.4 : main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    >
<TextView
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="Texte1"
    android:paddingBottom="5px"
    />
<TextView
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="Texte2"
    />
</LinearLayout>
```

ex 3.4 : résultat



Gravité

Comme vous avez pu le constater, les widgets se placent dans le layout en haut et à gauche par défaut.
Pour placer votre widget autre part, il faudra utiliser l'attribut : "`android:gravity=`"
Cet attribut peut prendre différentes valeurs :

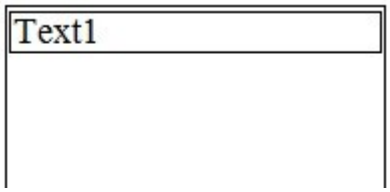
- `android:gravity="center_horizontal"`
- `android:gravity="center_vertical"`
- `android:gravity="top"`
- `android:gravity="bottom"`
- `android:gravity="left"`
- `android:gravity="right"`

Il est même possible de combiner les valeurs grâce au symbole | (touche 6 + Alt Gr) :

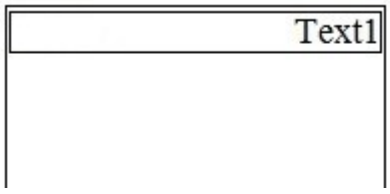
```
android:gravity="top|right"
```

Personnellement, je trouve que la gravité est relativement complexe à appréhender, mais je vais quand même essayer de vous donner 2 ou 3 infos sur son utilisation.

Par défaut, le texte de notre widget se place en haut à gauche de l'espace qui lui est réservé. En utilisant l'attribut "gravity", vous pouvez modifier cet état de fait.
Prenons un exemple :
Si vous choisissez "`android:layout_width="fill_parent"`" le widget prendra toute la place restante disponible en largeur.
si vous n'utilisez pas l'attribut "gravity", vous obtiendrez ceci :



Si dans ce même cas, vous utilisez "`android:gravity="right"`" , vous obtiendrez ceci :



En revanche, si vous utilisez "`android:gravity="bottom"`" vous n'obtiendrez aucun changement, "Text1" est déjà au plus bas possible de l'espace qui lui est réservé.

Petit test tout de même :

ex 3.5 : main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    >
<TextView
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:gravity="right"
    android:text="Text1"
    />
</LinearLayout>
```

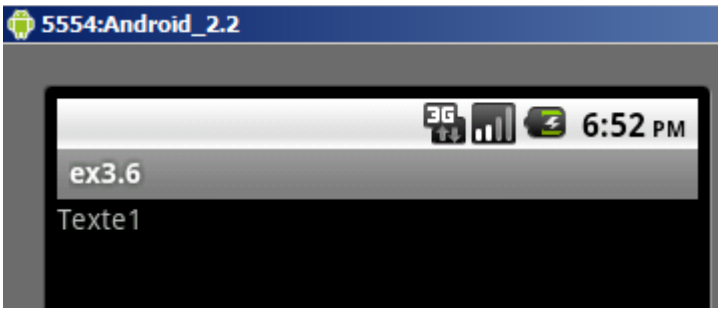
ex 3.5 : résultat

exemple en utilisant "bottom"

ex 3.6 : main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    >
<TextView
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:gravity="bottom"
    android:text="Texte1"
    />
</LinearLayout>
```

ex 3.6 : résultat

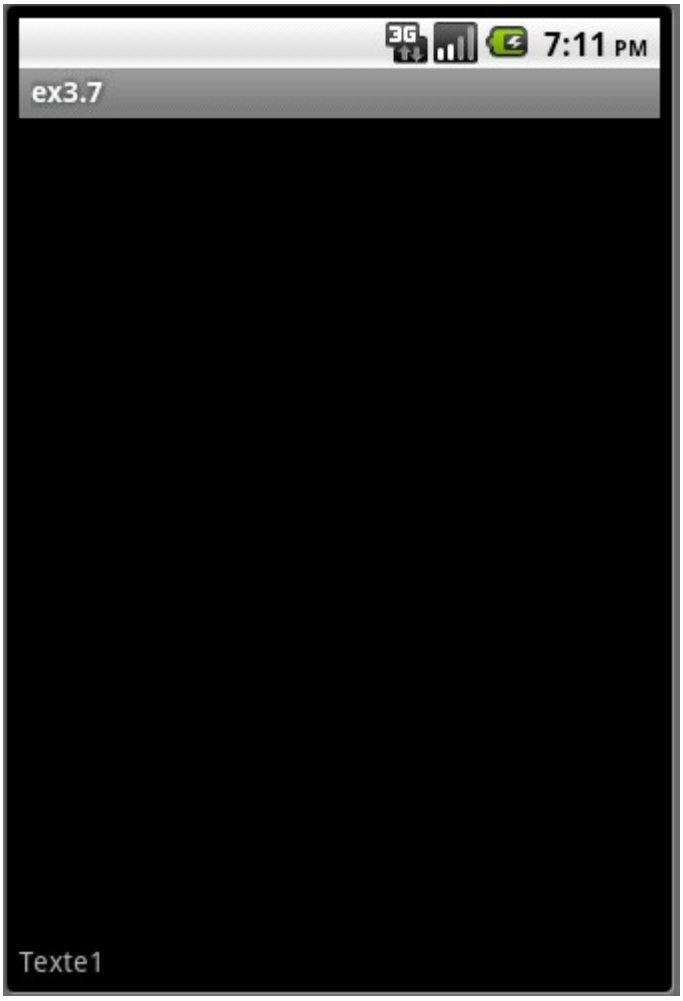


Deux exemples avec "`android:layout_width="wrap_content"`" et "`android:layout_height="fill_parent"`" :

ex 3.7 : main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    >
<TextView
    android:layout_width="wrap_content"
    android:layout_height="fill_parent"
    android:gravity="bottom"
    android:text="Texte1"
    />
</LinearLayout>
```

ex 3.7 : résultat



Si nous "tentons" un "`android:gravity="bottom"`" : aucun effet.
Essayez de comprendre pourquoi ?

ex 3.8 : main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    >
<TextView
    android:layout_width="wrap_content"
    android:layout_height="fill_parent"
    android:gravity="right"
    android:text="Texte1"
    />
</LinearLayout>
```

ex 3.8 : résultat

Cela veut donc dire que si vous utilisez "`android:layout_width="wrap_content"` " et "`android:layout_height="wrap_content"`", l'attribut "gravity" n'aura aucun effet.

Il est aussi possible d'utiliser "gravity" dans la balise linearLayout. C'est alors votre layout qui sera affecté par l'attribut "gravity" (pour les exemples, voir le chapitre 4, les exemples sont en effet plus parlant en utilisant des widgets "Button").

Il existe aussi un attribut "layout_gravity" à placer dans les balises des widgets. Nous verrons un exemple, là aussi avec le widget "Button", dans le chapitre suivant.

Poids (weight)

Prenons tout de suite un exemple :

ex 3.11 : main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="horizontal"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    >
<TextView
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="Texte1"
    />
<TextView
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="Texte2"
    />
</LinearLayout>
```

ex3.11 : résultat

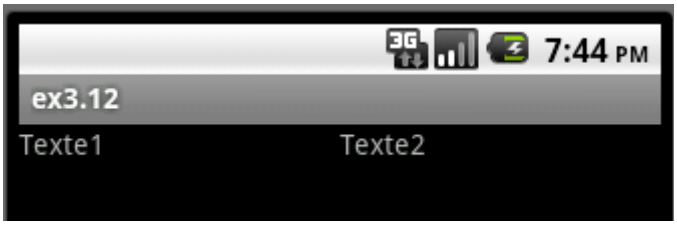
Comme vous pouvez le constater, aucune trace de "Texte2" !
Logique me direz-vous, nous avons un "`android:layout_width="fill_parent"`" qui permet à "Texte1" de prendre toute la place disponible. Il ne reste donc plus de place pour "Texte2"

En utilisant l'attribut "layout_weight", il est possible de "partager" l'espace disponible.

ex3.12 : main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="horizontal"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    >
<TextView
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:layout_weight="1"
    android:text="Texte1"
    />
<TextView
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:layout_weight="1"
    android:text="Texte2"
    />
</LinearLayout>
```

ex3.12 : résultat



Les 2 widgets se sont "partagés" l'espace libre à égalité, car dans les 2 cas "`android:layout_weight="1"`". Si nous avions eu "2" pour "Texte1" et "1" pour "Texte2", "Texte1" aurait eu 2 fois plus de place que "Texte2". Ce qui compte, ce ne sont pas les valeurs de "`android:layout_weight`" mais les proportions de l'un par rapport à l'autre.

Chapitre IV
Les widgets

Les widgets sont eux aussi des objets au sens POO du terme. Chaque type de widgets est une classe qui dérive de la classe "View" (d'ailleurs, la classe "Viewgroup" hérite elle aussi de la classe "View"). Nous placerons systématiquement nos widgets dans des layouts. Commençons donc à étudier quelques widgets.

Les Boutons

Voici un fichier main.xml qui permettra d'afficher un bouton

ex4.1 : main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    >
<Button
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="Un Bouton"
    />
</LinearLayout>
```

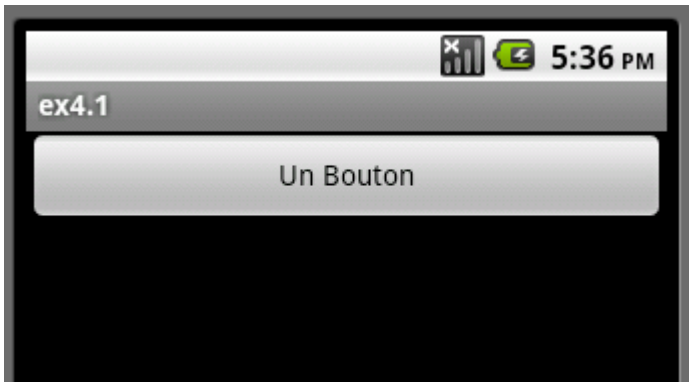
Nous avons une balise "`Button`" : les attributs "`layout_width`" et "`layout_height`" ne devraient pas vous poser de problème (rien de nouveau). L'attribut "`text`", vous permet d'afficher du texte dans votre bouton.

ex4.1 : Principale.java

```
package org.education.lgf.atelier;

import android.app.Activity;
import android.os.Bundle;

public class Principale extends Activity {
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
    }
}
```

N. B. Dans tout ce chapitre, "Principale.java" ne sera pas modifié.

ex4.1 : résultat

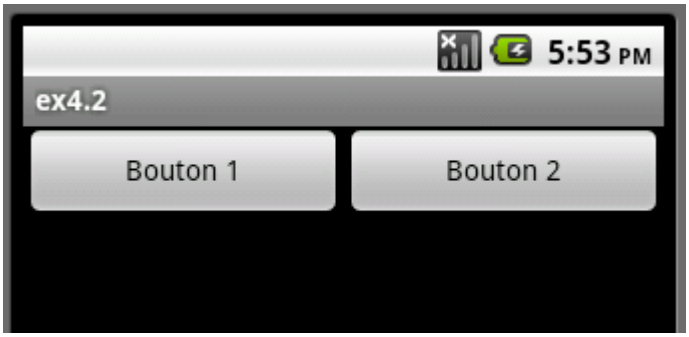


Nous allons maintenant pouvoir réutiliser tout ce que nous avons dans le chapitre précédent (gravity, weight, padding)
Utilisation de layout_weight :

ex4.2 : main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="horizontal"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    >
<Button
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="Bouton 1"
    android:layout_weight="1"
    />
<Button
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="Bouton 2"
    android:layout_weight="1"
    />
</LinearLayout>
```

ex4.2 : résultat

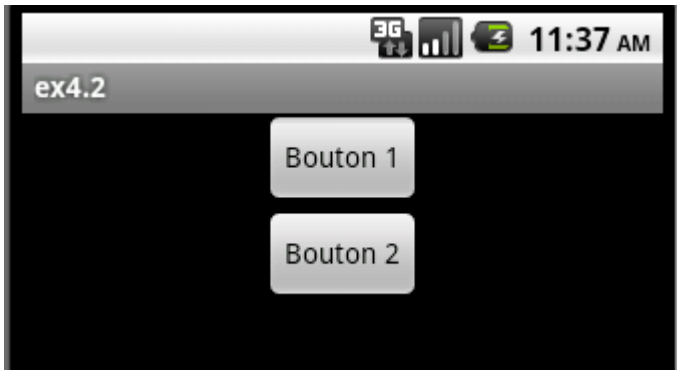


Maintenant un peu de gravity dans la balise "LinearLayout" et uniquement du "*wrap_content*" pour les boutons.

ex4.3 : main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:gravity="center_horizontal"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    >
<Button
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Bouton 1"
    />
<Button
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Bouton 2"
    />
</LinearLayout>
```

ex4.3 : résultat



autre exemple, toujours avec "gravity" dans la balise "LinearLayout" et uniquement du "*wrap_content*" pour les boutons.

ex4.4 : main.xml

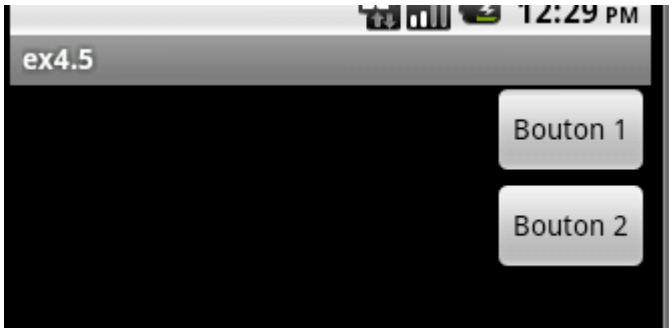
```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
      android:gravity="center"
    android:orientation="horizontal"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    >
<Button
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Bouton 1"
    />
<Button
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Bouton2"
    />
</LinearLayout>
```

ex4.4 : résultat

ex4.5 : main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:gravity="right"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    >
<Button
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Bouton 1"
    />
<Button
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Bouton 2"
    />
</LinearLayout>
```

ex4.5 : résultat



Pour terminer avec les boutons, un exemple avec un "layout_gravity"

ex4.6 : main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="horizontal"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    >
<Button
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_gravity="center"
    android:text="Bouton 1"
    />
<Button
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Bouton 2"
    />
</LinearLayout>
```

ex4.6 : résultat

Je vous conseille de faire vous même vos essais pour pouvoir vous approprier toutes ces notions (gravity, weight,.....)

Les "TextView"

Nous avons déjà vu les "TextView" au chapitre précédent, juste quelques mots, pour vous signaler l'existence de quelques attributs supplémentaires à éventuellement ajouter à vos balises "TextView"
L'attribut "`android:textStyle`" peut prendre les valeurs "normal", "bold" (gras), "italic" (italique)
exemple : "`android:textStyle="italic"`"
L'attribut "`android:typeface`" vous permet de choisir la police de caractères ("sans", "serif", "monospace")
exemple : `android:typeface="sans"`

Il existe bien d'autres attributs pour "TextView" (couleur du texte, taille de la police.....) n'hésitez pas à consulter la documentation officielle.

NB : Les attributs de "TextView" sont aussi utilisables avec le widget "Button".

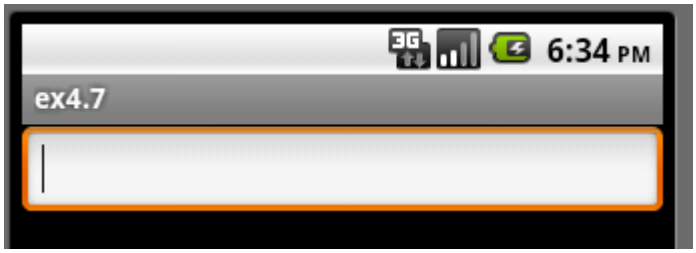
Les "EditText"

Le widget "EditText" va vous permettre de saisir du texte

ex4.7 : main.xml

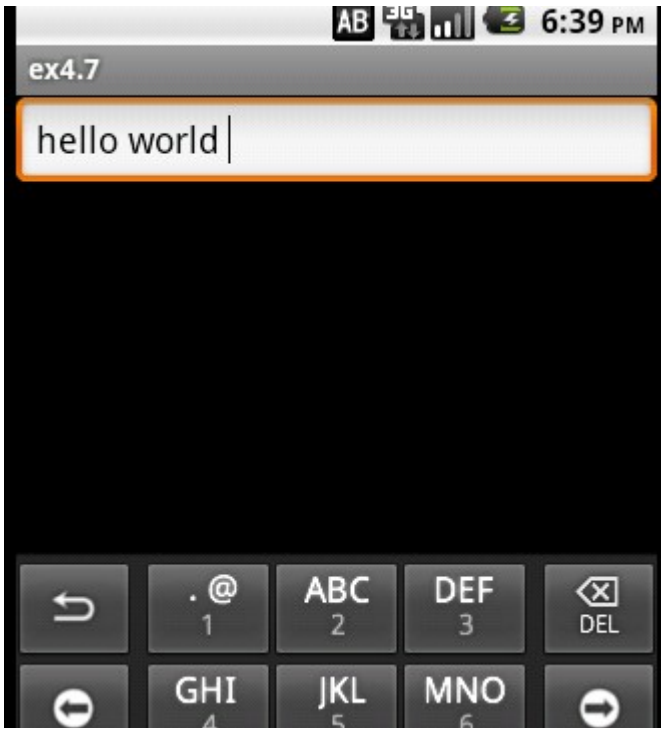
```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    >
<EditText
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"

    />
</LinearLayout>
```

ex4.7 : résultat 1



Rien à signaler de spécial, en cliquant sur le champ que vous venez de créer, vous faites automatiquement apparaître un clavier virtuel. Vous pouvez alors taper votre texte.

ex4.7 : résultat 2

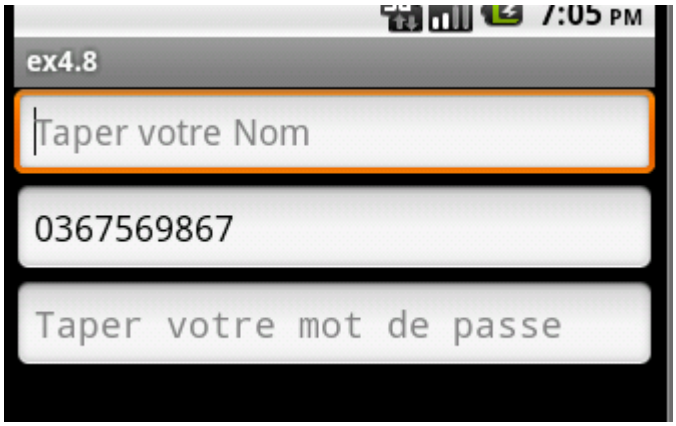


Nous allons maintenant ajouter différents attributs à notre balise "EditText".

ex4.8 : main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    >
<EditText
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:hint="Taper votre Nom"
    />
<EditText
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:phoneNumber="true"
    android:text="0367569867"
    />
<EditText
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:hint="Taper votre mot de passe"
    android:password="true"
    />
</LinearLayout>
```

ex4.8 : résultat 1



L'attribut "`android:hint`" permet d'indiquer à l'utilisateur le type de données attendues dans le champ (ici l'utilisateur doit taper son nom).
L'attribut "`android:phoneNumber`" "force" l'utilisateur à entrer un numéro de téléphone quand il est égal à "`true`"
L'attribut "`android:text`" permet de pré remplir le champ avec une valeur par défaut (libre à l'utilisateur de changer ou non cette valeur).

Comme vous pouvez le constater, le dernier champ attend un mot de passe. Entrons donc notre mot de passe.

Ex4.8 : résultat 2

Comme vous pouvez le constater, l'attribut "`android:password="true"`" permet de rendre invisible ce qu'est en train de taper l'utilisateur.

Nous avons vu ici que quelques-uns des attributs de la balise "EditText". Une fois de plus, je compte sur vous pour creuser la question en consultant la documentation officielle d'Android !

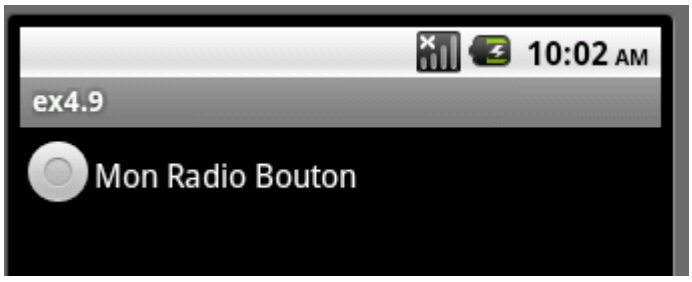
Les boutons radio (balises "RadioButton" et "RadioGroup")

Nous allons maintenant étudier les boutons radio et la balise "RadioButton"

ex4.9 : main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    >
<RadioButton
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="Mon Radio Bouton"
    android:checked="false"
    />
</LinearLayout>
```

ex4.9 : résultat



Rien de bien compliqué ici :

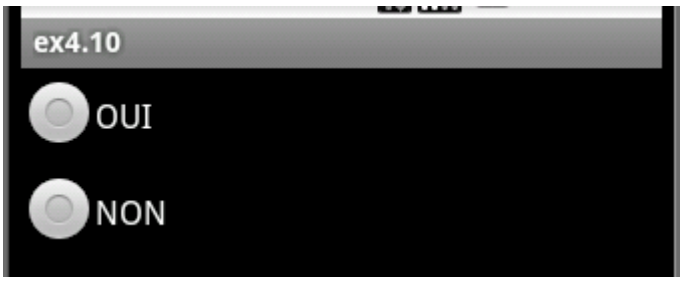
- l'attribut "`android:text`" détermine le texte qui sera associé à votre bouton radio
- l'attribut "`android:checked`" vous permet de choisir validé (="true") ou non validé (="false").

Dans la plupart des cas les boutons vont au moins par paire (voir plus). On peut regrouper les boutons radio grâce à la balise "RadioGroup". Cela permet de lier l'état des boutons (à un instant donné, un seul des boutons peut-être sélectionné (à condition de ne pas avoir utilisé l'attribut "`android:checked`")).

ex4.10 : main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    >
<RadioGroup
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:orientation="vertical">
    <RadioButton
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="OUI"
    />
    <RadioButton
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="NON"
    />
</RadioGroup>

</LinearLayout>
```

ex4.10 : résultat



Bien évidemment, tout ce que nous avons vu pour les autres widgets reste valable (gravity, style,...)

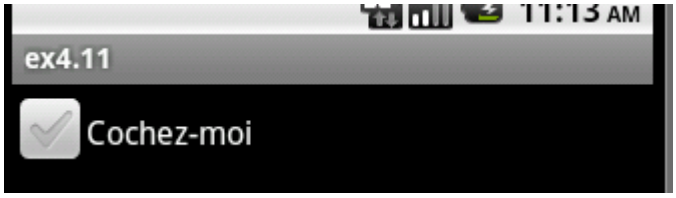
Les "CheckBox"

Les "CheckBox" sont des cases à cocher et à décocher :

ex4.11 : main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    >
<CheckBox
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Cochez-moi"
    />
</LinearLayout>
```

ex4.11 : résultat



Promis, c'est la dernière fois que je le répète : tout ce que nous avons vu avant reste valable.....

DatePicker

Passons tout de suite à l'exemple

ex4.12 : main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    >
<DatePicker
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    />
</LinearLayout>
```

ex4.12 : résultat

Rien à signaler, il existe l'équivalent pour l'heure : "TimePicker", je vous laisse le tester par vous même

Encore une fois, il existe d'autres widgets qui pourraient vous être utiles, n'hésitez pas à consulter la documentation.

Chapitre 5
Les listeners

Nous venons de voir de nombreux widgets, mais pour l'instant, quand vous appuyez sur un bouton, quand vous entrez du texte ou quand vous cochez une case, il ne se passe rien, désespérément rien ! Dans ce chapitre nos widgets vont enfin "prendre vie"

Avant d'allez plus loin, je vous conseille de revoir le dernier chapitre du document "d'Alice à Java". Si vous ne maitrisez pas, au moins dans les grandes lignes, ce chapitre, vous risquez de très rapidement vous retrouver en difficulté en lisant ce qui suit.

Avant d'entrer dans le vif du sujet, nous allons apprendre à fabriquer des "toasts".
Un "toast" est un message "surgissant" qui normalement permet au système de donner une information à l'utilisateur, sans trop le déranger (le "toast" s'affiche tout au plus quelques secondes puis disparaît sans intervention de l'utilisateur). Pour étudier les listeners, nous allons principalement utiliser des "toasts".
Un "toast" est une instance de la classe "Toast", voici une première application qui ne fait qu'afficher un "toast".

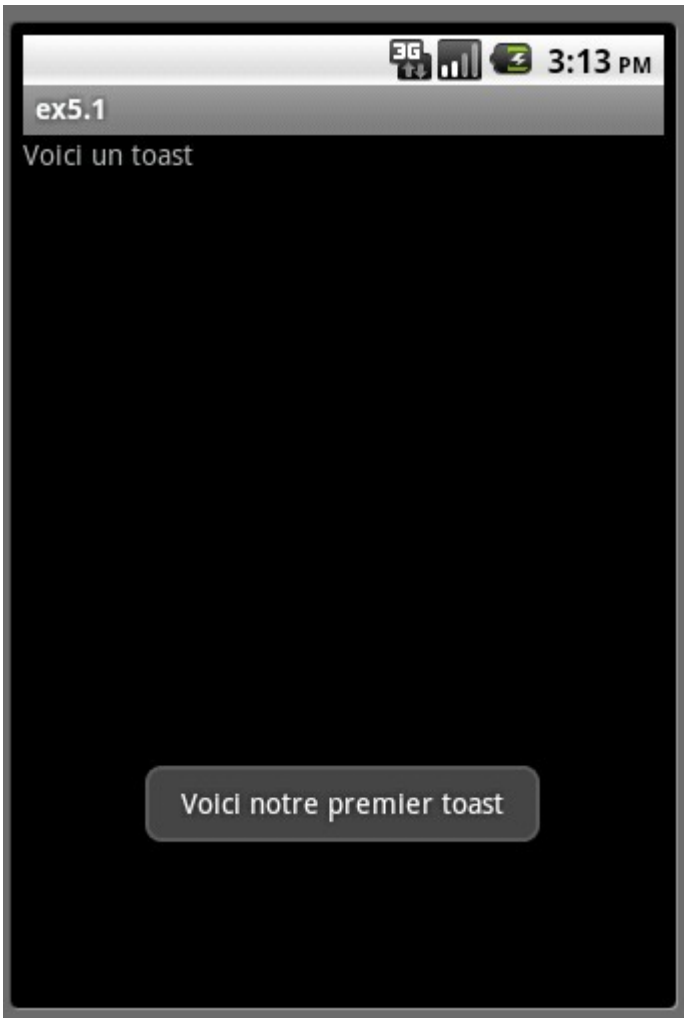
ex5.1 : main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    >
<TextView
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="Voici un toast"
    />
</LinearLayout>
```

ex5.1 : Principale.java

```
package org.eduction.lgf.atelier;

import android.app.Activity;
import android.os.Bundle;
import android.widget.Toast;

public class Principale extends Activity {
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        Toast toast= Toast.makeText(this,"Voici notre premier toast",
Toast.LENGTH_LONG);
        toast.show();
    }
}
```

ex5.1 : résultat



Deux lignes nous intéressent ici :

```
Toast toast= Toast.makeText(this,"Voici notre premier toast", Toast.LENGTH_LONG);
toast.show();
```

1ère ligne :
"`Toast toast=` " : Nous créons un objet de type "Toast", nous appelons cette instance "toast"
"makeText" est une méthode static de la classe "Toast" elle doit donc être appliquée à la classe "Toast" et non pas à notre instance "toast" d'où le : "`Toast.makeText`".
Cette méthode static "makeText" renvoie comme "valeur" une instance de la classe "Toast" que nous "rangeons" dans l'instance "toast" que nous venons de créer d'où le :
"`Toast toast= Toast.makeText`"
"makeText" attend 3 paramètres : `(this,"Voici notre premier toast", Toast.LENGTH_LONG)`

- l'activité qui "accueille" notre "toast", ici c'est notre classe "Principale", donc "this" !
- Le texte de notre "toast" : "`"Voici notre premier toast"`"
- Une constante static qui donne la durée de l'affichage du toast : "`Toast.LENGTH_LONG`" permet d'afficher le "toast" 5 secondes, "`Toast.LENGTH_SHORT`" affiche le "toast" 2 secondes.

2ème ligne :
Affichage du toast grâce à la méthode "show ()" (on applique la méthode show () à notre instance "toast" : `toast.show();`)

Vous pouvez aussi tout comprimer en une seule ligne sans créer d'instance :
`Toast.makeText(this,"Voici notre premier toast", Toast.LENGTH_LONG).show();`
Enfin, attention de ne pas oublier d'ajouter la ligne : "`import android.widget.Toast`"
Il nous reste une chose à voir avant d'aborder les listeners. Pour l'instant nos widgets ne peuvent pas être identifiés dans notre fichier "Principale.java" (nos widgets non pas de "nom"). Nous allons commencer par leur donner un identifiant dans le fichier "main.xml" grâce à l'attribut "id".

ex5.2 : main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    >
<Button
      android:id="@+id/bouton1"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="Mon Bouton 1"
    />
<Button
      android:id="@+id/bouton2"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="Mon Bouton 2 "
    />
</LinearLayout>
```

Nous avons créé 2 boutons : un bouton identifié par "bouton1" et l'autre identifié par "bouton2" (ne tenez pas compte du "@+id/", il est obligatoire de commencer par cette suite de caractères, c'est tout ce que vous avez à savoir !)

Je vous ai dit que les widgets étaient des objets (par exemple, nous avons la classe "Button" qui hérite de la classe "View"). Dans notre fichier "Principale.java" nous allons créer 2 instances de la classe "Button", la première instance "bouton1" représentera le bouton ayant pour "id" "`@+id/bouton1`" (nous avons le même nom pour l'instance (fichier "Principale.java") et pour l'id

(fichier "main.xml") , mais cela n'est pas obligatoire), la deuxième instance "bouton2" représentera le bouton ayant pour "id" "`@+id/bouton2`".

ex5.2 : Principale.java

```
package org.education.lgf.atelier;

import android.app.Activity;
import android.os.Bundle;
import android.widget.Button;

public class Principale extends Activity {
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        Button bouton1=(Button)findViewById(R.id.bouton1);
        Button bouton2=(Button)findViewById(R.id.bouton2);
    }
}
```

Pour commencer, j'attire votre attention sur la nouvelle ligne "import" :
"`import android.widget.Button;`"

Ensuite, la seule nouveauté :
"`Button bouton1=(Button)findViewById(R.id.bouton1);`"

Nous créons une instance ("bouton1") de la classe "Button" : "`Button bouton1`"
"findViewById" est une méthode de la classe "View" qui renvoie un objet de type "View". Ici, nous ne voulons pas tout à fait un objet de type "View" mais un objet qui dérive de "View", un objet de type "Button". Le "`(Button)findViewById`" permet donc de préciser que nous attendons en retour un objet de type "Button".
Enfin, nous devons passer en paramètre de cette méthode l'id que nous avons choisi dans notre fichier "main.xml", ici "R.id.bouton1" (ici aussi ne vous préoccupez pas du "R.id.", il ne faut pas l'oublier, c'est tout). Voilà, nous avons créé un objet "bouton1" grâce a cette méthode "findViewById".

Comme avec swing nous allons implémenter une interface, ici notre interface se nomme "`View.OnClickListener`". Cette interface possède une seule méthode (nous n'aurons donc qu'une méthode à redéfinir : "`onClick`").
Pour "écouter" un bouton, nous allons définir un "listener" grâce à la méthode "`setOnClickListener`".
Avec ces nouvelles méthodes (et cette nouvelle interface), nous allons devoir ajouter une ligne dans les "**`import`**" : "**`import`** `android.view.View`"

ex5.3 : Principale.java

```
package org.education.lgf.atelier;

import android.app.Activity;
import android.os.Bundle;
import android.widget.Button;
import android.widget.Toast;
import android.view.View;

public class Principale extends Activity implements View.OnClickListener {
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        Button monbouton =(Button)findViewById(R.id.bouton);
        monbouton.setOnClickListener(this);
    }
    public void onClick(View view){
      Toast.makeText(this,"BRAVO", Toast.LENGTH_SHORT).show();
    }
}
```

"`monbouton.setOnClickListener(this)`" permet "d'écouter" "monbouton" (on définit un listener sur l'objet "monbouton". La méthode "`setOnClickListener`" attend un paramètre, l'objet qui va "écouter" le bouton. Dans la plupart des cas cela sera l'activité courante, d'où le "this".
Ensuite nous avons la redéfinition de la méthode "onClick". Cette méthode attend comme argument "l'objet" qui vient d'être "cliqué" (ici l'objet "view" sera donc "monbouton"). Enfin, la méthode "onClick" affiche un "toast".

Le fichier "main.xml" ne devrait pas vous poser de problème.

ex5.3 : main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    >
<Button
      android:id="@+id/bouton"
      android:layout_gravity="center"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Appuyer"
    />
</LinearLayout>
```

Comment gérer la présence de 2 boutons ?

Il suffit de tester l'objet "view" passé à la méthode "onClick", grâce, par exemple, à un "if/else"
Prenons un exemple:

ex5.4 : main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="horizontal"
    android:gravity="center"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    >
<Button
      android:id="@+id/boutonA"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Bouton A"
    />
<Button
      android:id="@+id/boutonB"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Bouton B"
    />
</LinearLayout>
```

Rien à signaler pour le fichier "main.xml"

ex5.4 : Principale.java

```
package org.education.lgf.atelier;

import android.app.Activity;
import android.os.Bundle;
import android.widget.Button;
import android.widget.Toast;
import android.view.View;

public class Principale extends Activity implements View.OnClickListener {
    /** Called when the activity is first created. */
      Button monboutonA;
      Button monboutonB;
      @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        monboutonA =(Button)findViewById(R.id.boutonA);
        monboutonA.setOnClickListener(this);
        monboutonB =(Button)findViewById(R.id.boutonB);
        monboutonB.setOnClickListener(this);
    }
    public void onClick(View view){
      if (view==monboutonA){
            Toast.makeText(this,"Bouton A", Toast.LENGTH_SHORT).show();}
      else {Toast.makeText(this,"Bouton B", Toast.LENGTH_SHORT).show();}
    }
}
```

L'exemple est, je pense, suffisamment parlant, il suffit de comparer l'objet qui vient de subir un clic (l'objet "view") avec "monboutonA". Si "view = = monboutonA" renvoie "true" alors le "toast" "Bouton A" sera affiché, sinon c'est le "toast" "Bouton B" qui sera affiché.
Avant d'en terminer avec cet exemple, j'aimerai attirer votre attention sur le fait que les variables de

type "Button", "monboutonA" et "monboutonB" ont été défini en dehors de la méthode "onCreate", pourquoi d'après vous ?
Si vous ne trouvez pas la réponse, essayez de les définir dans "onCreate" et vous verrez bien ce qui va se passer ! D'une façon générale, il est de bon ton de déclarer les objets de type "View" (les widgets) en dehors de toute méthode.

Pour mettre en place des listeners il est possible, comme avec swing, d'utiliser les classes anonymes (là aussi une petite révision avec "d'Alice à Java" s'impose peut-être !). Voici donc le même exemple que précédemment, sans interface, mais avec une classe anonyme.

ex5.5 : Principale.java

```
package org.education.lgf.atelier;

import android.app.Activity;
import android.os.Bundle;
import android.widget.Button;
import android.widget.Toast;
import android.view.View;
import android.view.View.OnClickListener;

public class Principale extends Activity {
    /** Called when the activity is first created. */
       private Button monboutonA;
       private Button monboutonB;
       @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        monboutonA =(Button)findViewById(R.id.boutonA);
        monboutonA.setOnClickListener(new OnClickListener (){
               public void onClick(View view){
                      Toast.makeText(Principale.this,"Bouton A", Toast.LENGTH_SHORT).show();
               }
        });
        monboutonB =(Button)findViewById(R.id.boutonB);
        monboutonB.setOnClickListener(new OnClickListener (){
               public void onClick(View view){
                      Toast.makeText(Principale.this,"Bouton B", Toast.LENGTH_SHORT).show();
               }
        });
       }

}
```

Nous avons ajouté une ligne "import" : "`import android.view.View.OnClickListener`" car nous allons utiliser la classe "OnClickListener".
Cette classe "OnClickListener" possède une méthode "onclick". C'est cette méthode que nous allons redéfinir :

```
monboutonA.setOnClickListener(new OnClickListener (){
               public void onClick(View view){
```

Remarquez une petite différence dans notre "toast" :

```
Toast.makeText(Principale.this,"Bouton B", Toast.LENGTH_SHORT).show()
```

Vous avez trouvé ?

Nous avons remplacé "`this`" par "`Principale.this`", pourquoi ?

L'instance qui "produit" le "toast" n'est plus issue de la classe "Principale" (notre premier "this") mais la classe "Principale.OnClickListener" (notre instance courante (le "this") est maintenant du type "OnClickListener").

A vous de choisir entre ces 2 méthodes (interface ou classe anonyme)

EditText

Nous n'allons pas parler d'un veritable listener, mais de la méthode à employer pour "récupérer" une chaîne de caractères.
Voici un exemple avec un bouton "Valider" et un "EditText" ("Entrez votre nom").

ex5.6 : main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    >
<EditText
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:id="@+id/nom_texte"
    android:hint="Entrez votre nom"
    />
<Button
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:id="@+id/bouton"
      android:layout_gravity="center_horizontal"
      android:text="Valider"
      />
</LinearLayout>
```

L'interface graphique ne devrait pas poser de problème.

ex 5.6 : Principale.java

```
package org.education.lgf.atelier;

import android.app.Activity;
import android.os.Bundle;
import android.widget.Button;
import android.widget.EditText;
import android.widget.Toast;
import android.view.View;
import android.view.View.OnClickListener;

public class Principale extends Activity {
    /** Called when the activity is first created. */
       private Button bouton;
       private EditText texte_nom;

       @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        bouton=(Button)findViewById(R.id.bouton);
        bouton.setOnClickListener(new OnClickListener (){
               public void onClick (View view) {
                      texte_nom=(EditText)findViewById(R.id.nom_texte);
                      String nom = texte_nom.getText().toString();
                      Toast.makeText(Principale.this,"Bonjour "+nom, Toast.LENGTH_LONG).show();
               }
        });
    }
}
```

"texte_nom" est un objet de type "EditText" qui correspond à la balise "nom_texte" du fichier main.xml.

```
String nom = texte_nom.getText().toString();
```

Voici la grande nouveauté de cet exemple :
La méthode "toString" est utilisée pour renvoyer la "description" sous forme de chaîne de caractères d'une instance (ici de l'instance "texte_nom"). La méthode "getText" récupère, parmi toutes les chaînes renvoyées par "toString", la chaîne de caractères correspondant au texte tapé par l'utilisateur (ici, normalement, son nom). Tout cela est peut-être un peu complexe, mais vous devez uniquement retenir que l'association : "instance_de_type_EditText.getText ( ).toString ( )" va renvoyer le texte tapé dans l'EditText par l'utilisateur. Cette chaîne de caractères est "rangée" dans une variable (de type String bien entendu !) : "nom" dans notre exemple.

Nous allons maintenant étudier d'autres listeners (les listeners de certains des widgets vus au chapitre 4).

CheckBox

La méthode "setOnCheckedChangeListener" va créer le listener; la classe "OnChekedChangeListener" sera notre classe anonyme.

ex5.7 : main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    >
<CheckBox
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:id="@+id/box"
    android:text="Cocher ou décocher la case"
    />
</LinearLayout>
```

RAS au niveau du fichier main.xml

ex5.7 : Principale.java

```
package org.education.lgf.atelier;

import android.app.Activity;
import android.os.Bundle;
import android.widget.CheckBox;
import android.widget.CompoundButton;
import android.widget.Toast;

public class Principale extends Activity {
    /** Called when the activity is first created. */
       CheckBox maboite;
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        maboite=(CheckBox)findViewById(R.id.box);
        maboite.setOnCheckedChangeListener(new CheckBox.OnCheckedChangeListener (){
               public void onCheckedChanged (CompoundButton buttonView, boolean isChecked){
                      if (isChecked==true){
                             Toast.makeText(Principale.this,"La case est cochée",
Toast.LENGTH_SHORT).show();
                      }
                      else {
                             Toast.makeText(Principale.this,"La case est décochée",
Toast.LENGTH_SHORT).show();
                      }

               }

        });

    }
}
```

Le listener que nous mettons en place avec "`setOnCheckedChangeListener`" surveille le changement d'état de notre "CheckBox".
La méthode "`onCheckedChanged`" est l'équivalent de la méthode "onClick" pour les boutons. Cette méthode accepte 2 paramètres : "`buttonView`" de type "`CompoundButton`" correspond à la "CheckBox" concernée par le clic (ici "`buttonView`" correspond à "`maboite`") et "`isChecked`" de type "**`boolean`**" qui est égal à "true" si la case est cochée et à "false" si la case est décochée.
Le reste ne devrait pas vous poser de problème.

Voici un second exemple qui illustre l'utilisation de "`buttonView`"
et de l'interface "`CompoundButton.OnCheckedChangeListener`".

ex5.8 : main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    >
<CheckBox
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:id="@+id/box_1"
    android:text="Box 1"
    />
<CheckBox
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:id="@+id/box_2"
    android:text="Box 2"
    />
</LinearLayout>
```

ex5.8 : Principale.java

```
package org.education.lgf.atelier;

import android.app.Activity;
import android.os.Bundle;
import android.widget.CheckBox;
import android.widget.CompoundButton;
import android.widget.Toast;

public class Principale extends Activity implements CompoundButton.OnCheckedChangeListener {
    /** Called when the activity is first created. */
        CheckBox box1;
        CheckBox box2;
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        box1=(CheckBox)findViewById(R.id.box_1);
        box1.setOnCheckedChangeListener(this);
        box2=(CheckBox)findViewById(R.id.box_2);
        box2.setOnCheckedChangeListener(this);

        }
    public void onCheckedChanged(CompoundButton buttonView, boolean isChecked){
        if (buttonView==box1){
                if (isChecked){
                        Toast.makeText(Principale.this,"Box 1 cochée", Toast.LENGTH_LONG).show();
                }
                else {
                        Toast.makeText(Principale.this,"Box 1 décochée", Toast.LENGTH_LONG).show();
                }
        }
        else {
                if (isChecked){
                        Toast.makeText(this,"Box 2 cochée", Toast.LENGTH_LONG).show();
                }
                else {
                        Toast.makeText(this,"Box 2 décochée", Toast.LENGTH_LONG).show();
                }

        }

    }
}
```

Je vous laisse analyser seul ce code, prenez votre temps et cela ne devrait pas vous poser de problème.

Il existe d'autres méthodes à utiliser avec "CheckBox", consulter la documentation pour en savoir plus !

RadioButton et RadioGroup

ex5.9 : main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    >
<RadioGroup
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:id="@+id/monRadioGroup">
    <RadioButton
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:id="@+id/RaOUI"
      android:text="OUI">
    </RadioButton>
    <RadioButton
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:id="@+id/RaNON"
      android:text="NON">
    </RadioButton>
</RadioGroup>
</LinearLayout>
```

ex5.9 : Principale.java

```
package org.education.lgf.atelier;

import android.app.Activity;
import android.os.Bundle;
import android.widget.RadioButton;
import android.widget.RadioGroup;
import android.widget.Toast;

public class Principale extends Activity {
        RadioGroup radiog;
        RadioButton radiob_oui;
        RadioButton radiob_non;
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        radiog=(RadioGroup)findViewById(R.id.monRadioGroup);
        radiob_oui=(RadioButton)findViewById(R.id.RaOUI);
        radiob_non=(RadioButton)findViewById(R.id.RaNON);
        radiog.setOnCheckedChangeListener(new RadioGroup.OnCheckedChangeListener() {

                        public void onCheckedChanged(RadioGroup group, int checkedId) {
                                if ((RadioButton)findViewById(checkedId)==radiob_oui){
                                        Toast.makeText(Principale.this,"OUI",
Toast.LENGTH_SHORT).show();
                                }
                                else{
                                        Toast.makeText(Principale.this,"NON",
Toast.LENGTH_SHORT).show();
                                }
                        }
                });

    }
}
```

Remarquez que la méthode qui crée le listener, "`setOnCheckedChangeListener`" est identique à l'exemple précédent. Sinon rien de spécial, la structure commence à être connue.

En revanche, le "`if ((RadioButton)findViewById(checkedId)==radiob_oui)`" est un peu plus intéressant à étudier.
"checkedId" (paramètre de type "int" de la méthode "`onCheckedChanged`") est égal à l'Id du "RadioButton" qui vient d'être cliqué. Si vous venez de cliquer sur "OUI", "checkedId" est égal à "R.id.RaOUI".
"`(RadioButton)findViewById(checkedId)`" renvoi donc un objet de type "RadioButton". Si vous venez de cliquer sur "OUI" cet objet sera identique à "radiob_oui", d'où l'intérêt du "if/else"

DatePicker

La structure du listener est un peu différente de ce que nous venons de voir.
ex5.10 : main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    >
<DatePicker
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:id="@+id/date"
    />
</LinearLayout>
```

ex5.10 : Principale.java

```
import android.app.Activity;
import android.os.Bundle;
import android.widget.DatePicker;
import android.widget.Toast;

public class Principale extends Activity {
       DatePicker choixDate;
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        choixDate=(DatePicker)findViewById(R.id.date);
        choixDate.init(2010,10,22,new DatePicker.OnDateChangedListener() {

                    public void onDateChanged(DatePicker view, int year, int monthOfYear,
                            int dayOfMonth) {
                        Toast.makeText(Principale.this,"Voici la nouvelle Date :\nAnnée :
"+year+" Mois : "+monthOfYear+" Jour : "+dayOfMonth, Toast.LENGTH_LONG).show();

                    }
            });

    }
}
```

La méthode "init" est appliquée à notre instance (de type "DatePicker") "choixDate". Cette méthode accepte pour paramètres :

- l'année affichée par défaut
- le mois affiché par défaut
- le jour affiché par défaut

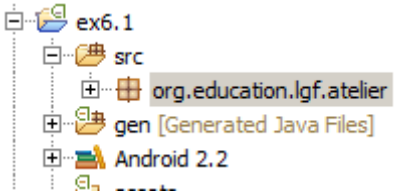
Attention, le numéro du mois commence à 0 (janvier = 0, février = 1,.....) (ce n'est pas le cas des années et des jours)
Enfin, dernier paramètre d'"init" le listener qui va détecter le changement de date ("`new DatePicker.OnDateChangedListener()`"). Le reste est classique.

Chapitre 6
Les intents

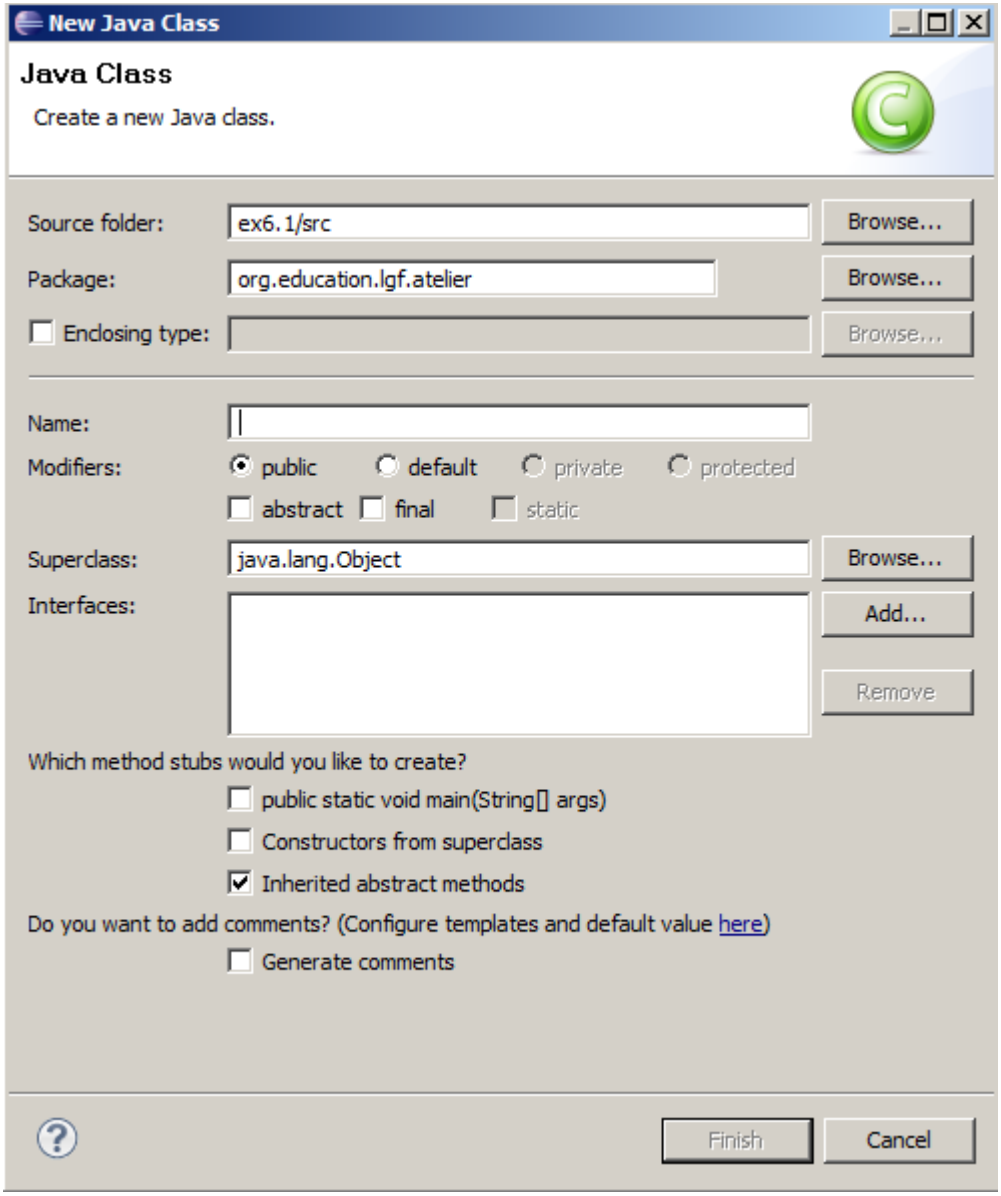
Avertissement
Ce chapitre est difficile, n'hésitez pas à prendre votre temps et à poser des questions !

Jusqu'à présent, nos applications ne comportaient qu'une seule activité (un seul écran). Il suffit de "jouer" 5 minutes avec une tablette ou un smartphone pour se rendre compte que la plupart des logiciels se composent de plusieurs écrans (donc de plusieurs activités).
Pour commencer ce chapitre, nous allons créer une deuxième activité (en plus de notre activité habituelle : "Principale").

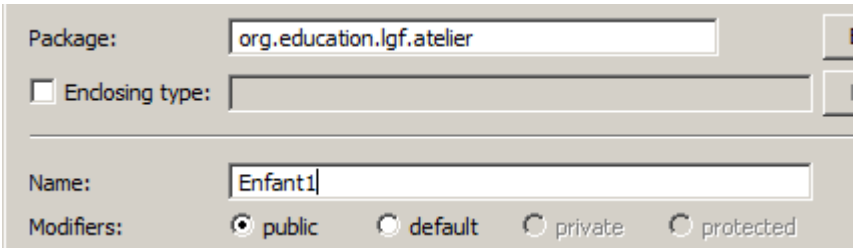
Nous allons appeler notre deuxième activité "Enfant1" (selon les actions de l'utilisateur, d'autres activités pourront être lancées depuis l'activité "Principale", nous appellerons donc ces activités, "activité enfant").
Après avoir créé notre projet et avoir demandé la création de notre activité "Principale" (comme nous le faisons depuis le début de ce document), nous allons faire un "clic droit" sur le package "org.education.lgf.atelier" de notre projet "ex6.1".



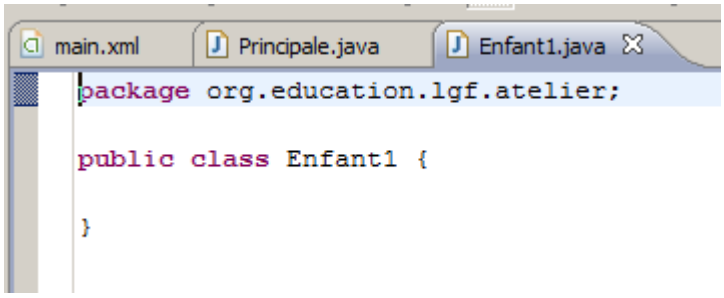
Choisir ensuite "New" puis "Class"



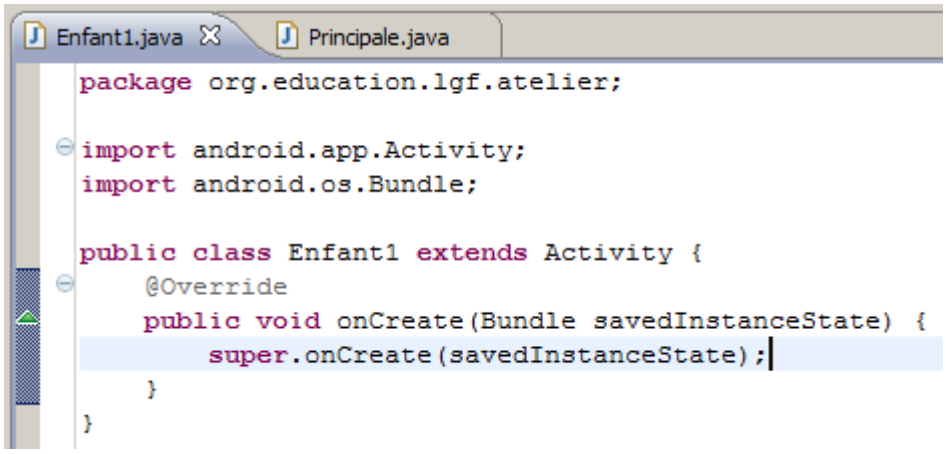
Il vous reste ensuite à compléter le champ "Name" avec "Enfant1" (sans toucher aux autres paramètres).



Cliquez sur "Finish" et c'est presque terminé.



Pour l'instant, nous avons une classe Java classique, pas une activité au sens "Android" du terme. Pour cela, il faut que notre classe "Enfants" hérite de la classe "Activity" (et nous allons aussi en profiter pour écrire le début de la méthode "onCreate")



Voici le code du projet ex6.1 :

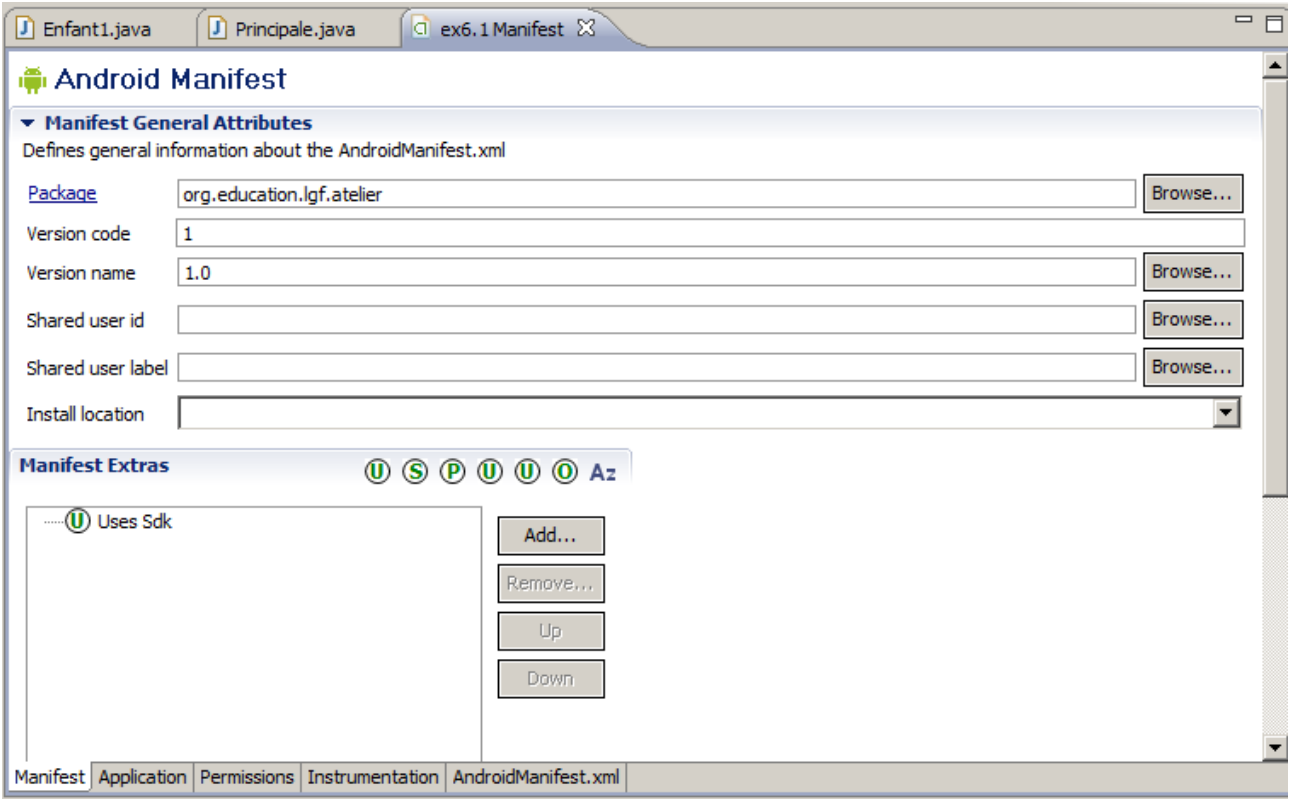
ex6.1 : Enfant1.java

```
package org.education.lgf.atelier;

import android.app.Activity;
import android.os.Bundle;

public class Enfant1 extends Activity {
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
    }
}
```

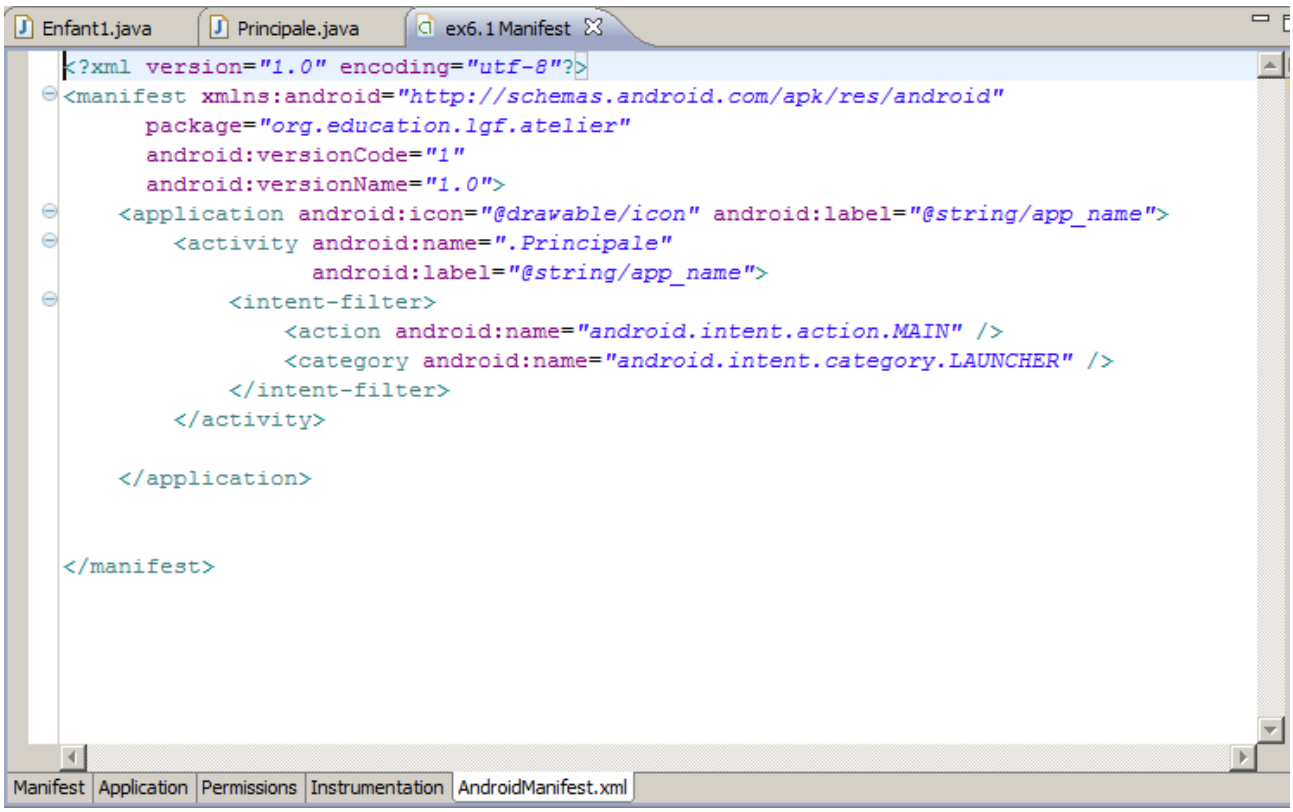
Terminé ?
Pas tout à fait. Notre nouvelle activité doit être déclarée dans un fichier xml dont je ne vous ai pas encore parlé (pourtant, ce fichier a une importance capitale dans un projet Android) : "AndroidManifest.xml"

Ce fichier n'est pas très difficile à trouver :

Ouvrez ce fichier "AndroidManifest.xml" en double cliquant dessus



Même s'il était possible de gérer ce fichier depuis cette fenêtre, par souci "d'universalité", nous allons plutôt utiliser le ficher ".xml" classique.
Cliquez donc sur l'onglet "AndroidManifest.xml" (il se trouve en bas)



Ouf, nous retrouvons une structure plus "familière".

ex6.1 : AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
      package="org.education.lgf.atelier"
      android:versionCode="1"
      android:versionName="1.0">
    <application android:icon="@drawable/icon" android:label="@string/app_name">
        <activity android:name=".Principale"
                  android:label="@string/app_name">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>

    </application>


</manifest>
```

Nous n'allons pas étudier ce fichier en détail (nous verrons chaque partie de ce fichier au fur et à mesure de notre progression).

Pour déclarer notre activité "Enfant1" nous allons faire un copier-coller de la balise <Activity> en supprimant la balise <intent-filter>. Pour l'instant, nous dirons seulement que cette balise contient des informations permettant à Android de connaître l'activité à lancer au démarrage de l'application (dans notre exemple, l'activité "Principale"). Notre activité "Enfant1" ne doit pas être lancée au démarrage de l'application, nous supprimons donc cette balise.
Ensuite, nous allons changer "`android:name=".Principale"`"

en "android:name=".Enfant1"".
Enfin, modifions le label : "label="@string/app_name"" en "label="sous_activité"".

Voici donc notre nouveau fichier "AndroidManifest.xml"
ex6.1 : AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
      package="org.education.lgf.atelier"
      android:versionCode="1"
      android:versionName="1.0">
    <application android:icon="@drawable/icon" android:label="@string/app_name">
        <activity android:name=".Principale"
                  android:label="@string/app_name">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
        <activity android:name=".Enfant1"
                  android:label="sous_activité">
        </activity>
    </application>
</manifest>
```

Maintenant que notre deuxième activité est créée, comment allons-nous procéder pour la "lancer" depuis notre activité "Principale" ?

Et bien, Android utilise des objets "intent" de la classe "Intent". Certains traduisent intent par intention, mais bon.....je ne suis pas convaincu par cette traduction et nous utiliserons donc la version originale "intent".
N'oubliez pas votre "petit tour" habituel dans la documentation officielle pour voir ce qui est dit sur ce sujet !

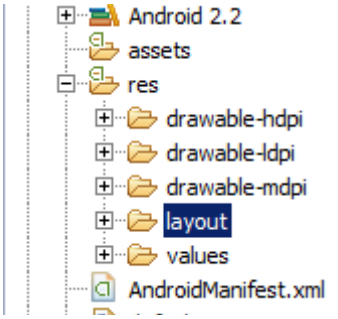
Pour appeler notre activité "Enfant1" depuis notre activité "Principale", nous allons créer une instance d'un objet de type "Intent". Les paramètres du constructeur sont :
- l'endroit où l'intent est créé (notre instance courante, c'est-à-dire "this")
- l'objectif de notre objet "intent", pour nous ici, notre activité "Enfant1.class"

Comme l'intent est une sorte de messager, à qui on donne l'adresse de l'expéditeur ("this") et l'adresse du destinataire ("Enfant1.class"), il nous faut une méthode pour "propulser" ce messager vers sa destination. Pour l'instant, nous utiliserons la méthode "startActivity".

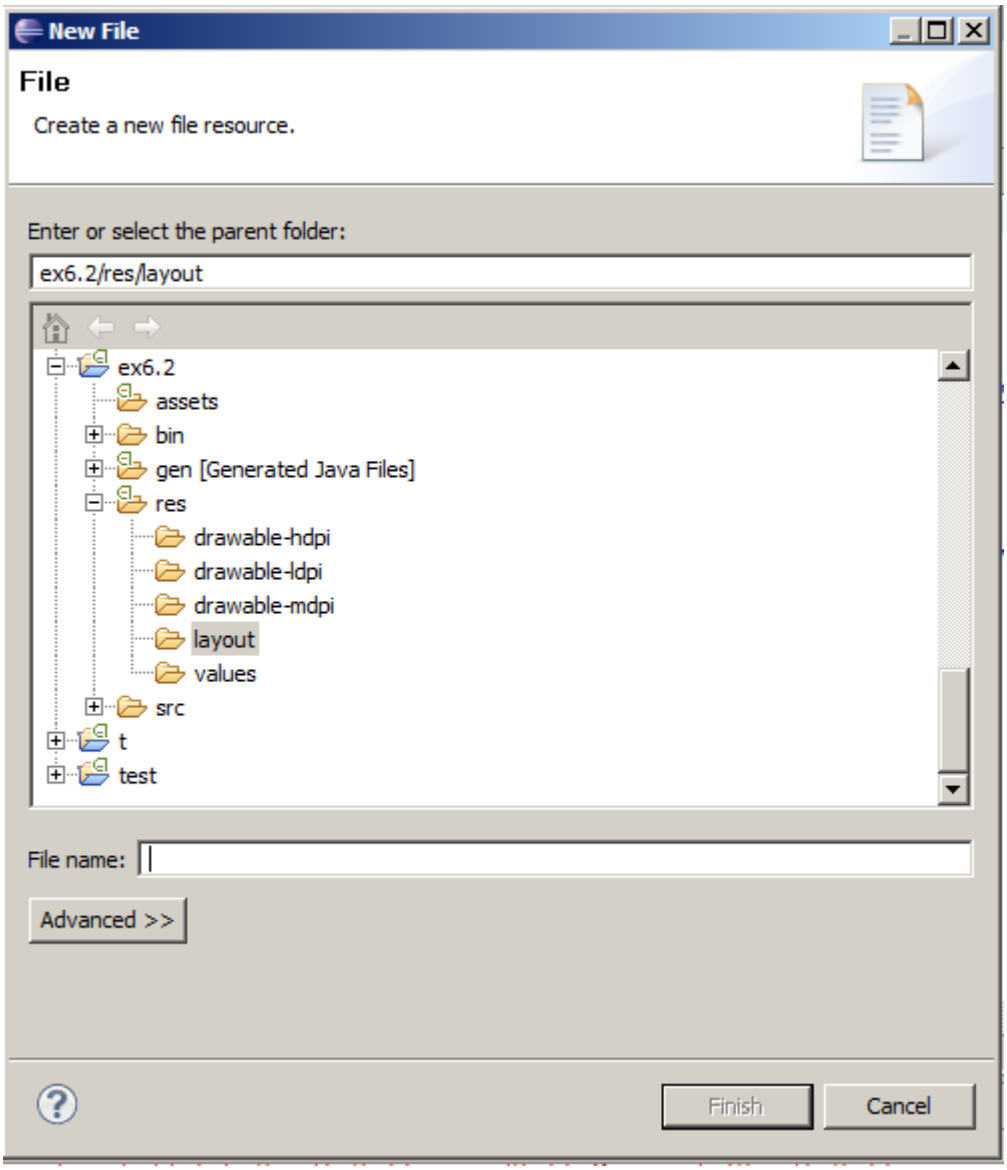
Voici un exemple pour illustrer tout cela:

Le point de départ de notre projet "ex6.2" sera le projet "ex6.1". Nous allons ensuite le modifier (nous ne toucherons pas au fichier "AndroidManifest.xml") notamment en créant un fichier "main1.xml" dans le répertoire "res/layout". Ce fichier nous permettra de définir une interface graphique pour notre activité "Enfant1". C'est parti :

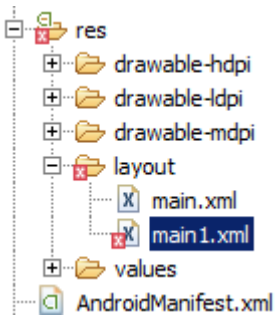
Pour créer un fichier "main1.xml", sélectionner le répertoire "layout"



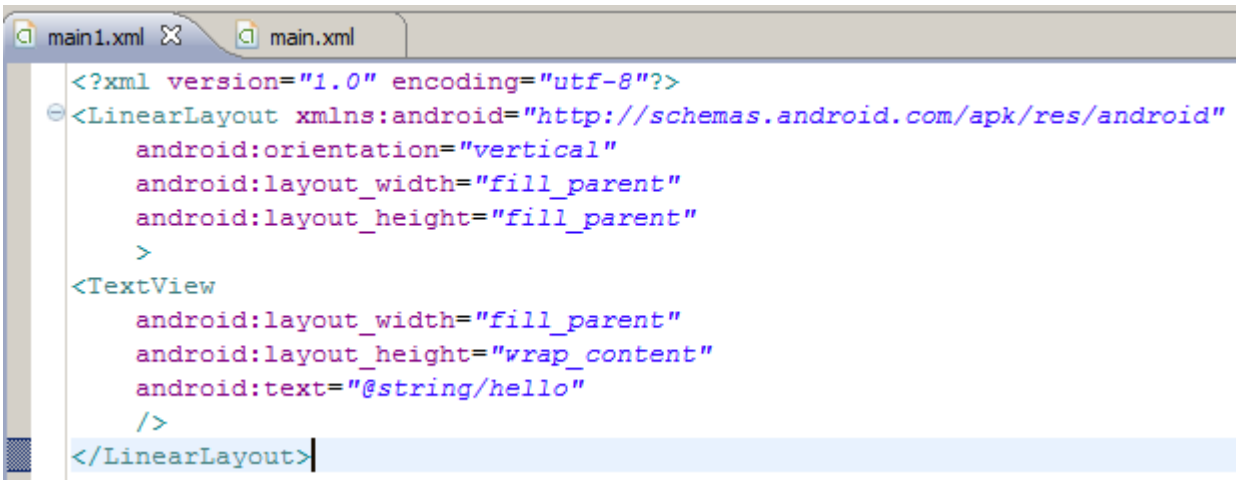
Faites un clic droit puis choisissez "New" suivit de "File". Vous devriez obtenir la fenêtre suivante :



Tapez "main1.xml" dans le champ "File name" puis cliquez sur "Finish"
Ne tenez pas compte de l'éventuelle erreur (croix rouge sur votre nouveau fichier)



Pour faire disparaître cette erreur, copiez le contenu du fichier "main.xml" dans le fichier "main1.xml".



Tous les éléments étant en place, passons à l'écriture des différents fichiers :

ex6.2 : main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    >
<Button
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="Appel de l'activité Enfant1"
     android:id="@+id/bouton"
    />
</LinearLayout>
```

ex6.2 : main1.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    >
<TextView
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="Bienvenu dans l'activité Enfant1 "

    />
</LinearLayout>
```

ex6.2 : Principale.java

```
package org.education.lgf.atelier;

import android.app.Activity;
import android.content.Intent;
import android.os.Bundle;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;

public class Principale extends Activity {
    Button bouton_1;
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        bouton_1=(Button)findViewById(R.id.bouton);
        bouton_1.setOnClickListener(new OnClickListener(){
               public void onClick (View view){
               Intent intent=new Intent(Principale.this,Enfant1.class);
               startActivity(intent);

               }
        });
    }
}
```

ex6.2 : Enfant1.java

```
package org.education.lgf.atelier;

import android.app.Activity;
import android.os.Bundle;

public class Enfant1 extends Activity {
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main1);
    }
}
```

Que de codes ! Et pourtant, très peu de nouveautés :
"`Intent intent=new Intent(Principale.this,Enfant1.class);`"
Nous créons un objet "intent" (on aurait pu choisir un autre nom) de type "Intent". Le premier paramètre du constructeur de la classe "Intent" correspond à l'instance en cours de notre activité "Principale" d'où le "`Principale.this`" le deuxième paramètre correspond à la cible de notre "intent" c'est-à-dire notre nouvelle activité "`Enfant1.class`" (qui n'est rien d'autre qu'une classe, d'où le "`.class`").

"`startActivity(intent);`"
Nous envoyons notre intent vers son destinataire avec la méthode "`startActivity`" l'unique paramètre étant notre instance "intent".
Si maintenant vous voulez revenir à l'activité "Principale", il suffit d'utiliser le bouton retour en arrière du téléphone.

Nous avons progressé, car nous sommes maintenant capable d'écrire des applications "multiactivités".
Mais il n y'a pour l'instant aucune "communication" entre nos activités, notamment, l'activité "Principale" n'a aucun retour de la part de l'activité "Enfant1".
Pour lancer l'activité "Enfant1" depuis l'activité "Principale" avec retour d'information à la fin de l'exécution d'"Enfant1", il faut utiliser la méthode "startActivityForResult" à la place de la méthode "StartActivity".

La méthode "startActivityForResult" demande un paramètre de plus que la méthode "StartActivity": un "requestCode".
Le "requestCode" est un nombre envoyé par l'activité "parent" à l'activité "enfant" (pour nous de "Principale" à "Enfant1". En retour, une fois terminée son exécution, l'activité "enfant" renverra le "requestCode" précédemment reçu vers l'activité "parent" (le "requestCode" aura fait un aller et retour).
A quoi cela sert ?
Et bien à identifier l'activité "enfant" qui vient de se terminer (lorsqu'une activité enfant se termine, la "main" est rendue à l'activité "parente"). En imaginant par exemple qu'il existe plusieurs activités "enfants", comment identifier l'activité "enfant" qui vient de se terminer ?
Grâce à son "requestCode" bien sûr !

La méthode "enfant", en se terminant, ne va pas uniquement retourner un "requestCode", elle va aussi retourner un "resultCode".
Pour un peu simplifier, un "resultCode" pourra prendre 2 valeurs : "RESULT_OK" si l'activité "enfant" va jusqu'au bout de son exécution ou "RESULT_CANCELED" si l'activité "enfant" a été interrompue d'une façon ou d'une autre (par exemple bouton "Cancel" ou appui sur la touche retour arrière du téléphone).

Dans l'activité "enfant", c'est la méthode "setResult" qui renverra vers l'activité "parent" le "resultCode"

Du côté "parent", c'est la méthode "onActivityResult" qui permettra de traiter le "requestCode" et le "resultCode". Cette méthode à 3 paramètres :

- le "requestCode"
- le "resultCode"
- un objet de type "Intent" qui sera chargé de "transporter" des informations supplémentaires (voir un peu plus loin).

Illustrons toutes ces nouvelles notions par un exemple:
Nous allons créer une activité supplémentaire "Enfant2" (avec son fichier "main2.xml")

ex6.3 main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    >
<Button
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="Appel de l'activité Enfant1"
     android:id="@+id/bouton1"
    />
<Button
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="Appel de l'activité Enfant2"
     android:id="@+id/bouton2"
    />
</LinearLayout>
```

ex6.3 main1.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    >
<TextView
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="Bienvenus dans l'activité Enfant1 "
   />
<Button
        android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:id="@+id/boutonOK1"
    android:text="OK"
    />
<Button
        android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:id="@+id/boutonANNULER1"
    android:text="ANNULER"
    />

</LinearLayout>
```

ex6.3 main2.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    >
<TextView
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="Bienvenus dans l'activité Enfant2 "
   />
<Button
        android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:id="@+id/boutonOK2"
    android:text="OK"
    />
<Button
        android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:id="@+id/boutonANNULER2"
    android:text="ANNULER"
    />
</LinearLayout>
```

ex6.3 Principale.java

```
package org.education.lgf.atelier;

import android.app.Activity;
import android.content.Intent;
import android.os.Bundle;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;
import android.widget.Toast;

public class Principale extends Activity {
    Button bouton_1;
    Button bouton_2;
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        bouton_1=(Button)findViewById(R.id.bouton1);
        bouton_2=(Button)findViewById(R.id.bouton2);


suite du code page suivante
```

suite de "Principale.java"

```
        bouton_1.setOnClickListener(new OnClickListener(){
                public void onClick (View view){
                Intent intent=new Intent(Principale.this,Enfant1.class);
                startActivityForResult(intent,1);
                }
         });
         bouton_2.setOnClickListener(new OnClickListener(){
                public void onClick (View view){
                Intent intent=new Intent(Principale.this,Enfant2.class);
                startActivityForResult(intent,2);
                }
         });
    }
    public void onActivityResult (int requestCode, int resultCode, Intent data){
        switch (requestCode){
        case (1):
                switch (resultCode){
                case RESULT_OK:
                        Toast.makeText(this,"Enfant1 OK", Toast.LENGTH_SHORT).show();
                        return;
                case RESULT_CANCELED:
                        Toast.makeText(this,"Enfant1 Annuler", Toast.LENGTH_SHORT).show();
                        return;
        }
        case (2):
                switch (resultCode){
                case RESULT_OK:
                        Toast.makeText(this,"Enfant2 OK", Toast.LENGTH_SHORT).show();
                        return;
                case RESULT_CANCELED:
                        Toast.makeText(this,"Enfant2 Annuler", Toast.LENGTH_SHORT).show();
                        return;
        }
        }

    }
}
```

ex6.3 Enfant1.java

```
package org.education.lgf.atelier;

import android.app.Activity;
import android.os.Bundle;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;

public class Enfant1 extends Activity {
Button B_OK;
Button B_AN;
    @Override
    public void onCreate(Bundle savedInstanceState) {
         super.onCreate(savedInstanceState);
         setContentView(R.layout.main1);
         B_OK=(Button)findViewById(R.id.boutonOK1);
         B_AN=(Button)findViewById(R.id.boutonANNULER1);
         B_OK.setOnClickListener (new OnClickListener () {
                public void onClick (View view) {
                        setResult(RESULT_OK);
                        finish();
                }
         });
         B_AN.setOnClickListener (new OnClickListener () {
                public void onClick (View view) {
                        setResult(RESULT_CANCELED);
                        finish();
                }
         });
    }
}
```

ex6.3 Enfant2.java

```
package org.education.lgf.atelier;

import android.app.Activity;
import android.os.Bundle;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;

public class Enfant2 extends Activity {
Button B_OK;
Button B_AN;
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main2);
        B_OK=(Button)findViewById(R.id.boutonOK2);
        B_AN=(Button)findViewById(R.id.boutonANNULER2);
        B_OK.setOnClickListener (new OnClickListener () {
               public void onClick (View view) {
                       setResult(RESULT_OK);
                       finish();
               }
        });
        B_AN.setOnClickListener (new OnClickListener () {
               public void onClick (View view) {
                       setResult(RESULT_CANCELED);
                       finish();
               }
        });
    }
}
```

Ouf, c'était long !
Que fait ce programme ?
Au démarrage vous avez 2 boutons : un bouton "*Appel de l'activité Enfant1*" et un bouton "*Appel de l'activité Enfant2*".
Si vous cliquez sur le premier bouton, vous créez un "intent" qui a pour paramètre "`Enfant1.class`" puis vous lancez l'activité "`Enfant1`" grâce à la méthode "`startActivityForResult`" (avec "1" comme "requestCode").
Une fois l'activité "`Enfant1`" lancée, vous vous retrouvez avec un texte et 2 boutons (OK et ANNULER). Si vous appuyer sur "OK" le "resultCode" "*RESULT_OK*" est envoyé à l'activité "Principale" grâce à la méthode "`setResult(`*RESULT_OK*`)`", enfin, l'activité "`Enfant1`" se termine avec la méthode "`finish()`". Si vous appuyez sur "ANNULER" le "resultCode" "*RESULT_CANCELED*" est envoyé à l'activité "Principale" grâce à la méthode "`setResult(`*RESULT_CANCELED*`)`", comme ci-dessus, l'activité "`Enfant1`" se termine avec la méthode "`finish()`".
Il se passe le même enchaînement avec cette fois-ci l'activité "`Enfant2`", si dans la méthode "Principale" vous cliquez sur le bouton "*Appel de l'activité Enfant2*".

Une fois de retour dans l'activité "Principale" la méthode "`onActivityResult`" est exécutée. Pour "changer un peu", nous avons utilisé le couple "switch/case" à la place du couple "if/else".
Nous avons 2 "switch/case" imbriqués l'un dans l'autre. Un qui gère le "`requestCode`" et l'autre qui gère le "`resultCode`".

Je vous laisse analyser ce programme qui commence un peu à se compliquer (encore une fois, prenez bien votre temps et n'hésitez pas à poser des questions).

Transporter des informations avec les intents

Les intents peuvent aussi vous permettre de "transporter" des informations grâce à la méthode "putExtra ()". Cette méthode utilise le principe couple clé/valeur. A chaque "valeur" correspond une

"clé".
La méthode "putExtra ()" est appliquée sur un objet de type "Intent" (l'intent va donc appeler la méthode "enfant" et transporter l'information). Cette méthode a deux paramètres : la clé et la valeur : "intent.putExtra (clé,valeur)".
Pour extraire la valeur de l'intent, il faut utiliser un objet de type "Bundle". Je ne vais pas entrer dans les détails, vous allez utiliser "l'enchaînement" :
"objet_de_type_Bundle = this.getIntent().getExtras()"
Ensuite pour placer la valeur transportée dans une variable nous appliquerons la méthode "getXXX" à l'"objet_de_type_Bundle" que nous avons créé juste au-dessus. La méthode "getXXX" accepte un paramètre, la clé associée à la valeur transportée. Vous vous doutez bien que la méthode "getXXX" n'existe pas vraiment. En fait, il faut remplacer "XXX" par le type de la valeur transportée. On pourra donc avoir un "getString" si la valeur transportée est de type String, un "getInt" si la valeur est de int.....

Un exemple ?

Voici:

ex6.4 main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent">
<EditText
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:hint="Entrez votre nom"
     android:id="@+id/editT"
    />
<Button
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="Valider"
     android:id="@+id/bouton"
    />
</LinearLayout>
```

ex6.4 main1.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    >
<TextView
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
       android:id="@+id/textV"
   />
<Button
       android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:id="@+id/boutonOK"
    android:text="OK"
    />
<Button
       android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:id="@+id/boutonANNULER"
    android:text="ANNULER"
    />
```

L'activité "Principale" est constituée d'un "EditText" qui attend le nom de l'utilisateur et d'un bouton (qui lancera l'activité "Enfant1")

L'activité "Enfant1" sera constituée d'un "TextView" (vide, pas de "android:text"), d'un bouton "OK" et d'un bouton "ANNULER" (ces 2 boutons ont les mêmes fonctions que dans l'exemple "ex6.3")

ex6.4 Principale.java

```
package org.education.lgf.atelier;

import android.app.Activity;
import android.content.Intent;
import android.os.Bundle;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;
import android.widget.EditText;
import android.widget.Toast;

public class Principale extends Activity {
    Button bouton;
    EditText texte_nom;
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        bouton=(Button)findViewById(R.id.bouton);
        bouton.setOnClickListener(new OnClickListener(){
               public void onClick (View view){
                       texte_nom=(EditText)findViewById(R.id.editT);
                       String nom = texte_nom.getText().toString();
                       Intent intent=new Intent(Principale.this,Enfant1.class);
                       intent.putExtra("monNom", nom);
                       startActivityForResult(intent,1);
               }
        });
    }
    public void onActivityResult (int requestCode, int resultCode, Intent data){
       switch (resultCode){
       case RESULT_OK:
               Toast.makeText(this,"Enfant1 OK", Toast.LENGTH_SHORT).show();
               return;
        case RESULT_CANCELED:
                       Toast.makeText(this,"Enfant1 Annuler", Toast.LENGTH_SHORT).show();
                       return;
       }
    }
}
```

"`String nom = texte_nom.getText().toString();`" permet de récuperer le texte tapé par l'utilisateur.
Nous définissons notre intent de la même façon que dans l'exemple "ex6.3" :
"`Intent intent=new Intent(Principale.this,Enfant1.class);`"
"`intent.putExtra("monNom", nom);`" nous ajoutons une information à l'intent
(clé = "monNom", valeur = la variable de type String définie juste au-dessus : "nom")

ex6.4 Enfant1.java

```
package org.education.lgf.atelier;

import android.app.Activity;
import android.os.Bundle;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;
import android.widget.TextView;

public class Enfant1 extends Activity {
Button B_OK;
Button B_AN;
TextView textV;
private String nom;
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main1);
        Bundle infoNom = this.getIntent().getExtras();
        if(infoNom != null){
               nom=infoNom.getString("monNom");
        }
        textV=(TextView)findViewById(R.id.textV);
        textV.setText("Bonjour "+nom+" bienvenu dans l'activité Enfant1");

        B_OK=(Button)findViewById(R.id.boutonOK);
        B_AN=(Button)findViewById(R.id.boutonANNULER);
        B_OK.setOnClickListener (new OnClickListener () {
               public void onClick (View view) {
                      setResult(RESULT_OK);
                      finish();
               }
        });
        B_AN.setOnClickListener (new OnClickListener () {
               public void onClick (View view) {
                      setResult(RESULT_CANCELED);
                      finish();
               }
        });
    }
}
```

Création d'un objet de type "Bundle" qui permet d'extraire l'information de l'intent :
"`Bundle infoNom = this.getIntent().getExtras();`"

```
if(infoNom != null){
       nom=infoNom.getString("monNom");
        }
```

Après avoir vérifié que l'objet de type "Bundle" ("infoNom") n'est pas vide, la méthode "getString" (avec comme paramètre la clé "`"monNom"`") permet de donner une valeur à la variable de type String "`nom`".
"`textV.setText("Bonjour "+nom+" bienvenu dans l'activité Enfant1");`" la méthode "`setText`" est l'équivalent d'un "android:text" dans un fichier layout, "main.xml".

Il est bien évidemment aussi possible de faire passer des informations de l'activité "enfant" vers l'activité "parente".
Il suffit de créer un objet de type Intent "vide" dans l'activité "enfant", d'utiliser la méthode "putExtra" pour "remplir" l'objet Intent avec les informations que vous voulez transmettre.
La méthode "setResult" aura alors 2 paramètres : le "requestCode" et l'objet Intent que vous venez de créer.
L'activité "parente" utilisera la méthode "getXXXExtra" (il faut bien sûr remplacer les XXX par le type de valeur transportée, String, Int,....)
Voici un exemple :

ex6.5 : main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent">
<Button
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="Vers Enfant1"
     android:id="@+id/bouton"
    />
</LinearLayout>
```

Nous avons conservé un unique bouton qui envoie vers "Enfant1"

ex6.5 : main1.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    >
<EditText
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
        android:id="@+id/editT"
        android:hint="Entrez votre age"
   />
<Button
        android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:id="@+id/boutonOK"
    android:text="OK"
    />
<Button
        android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:id="@+id/boutonANNULER"
    android:text="ANNULER"
    />

</LinearLayout>
```

L'activité "Enfant1" possède maintenant un "EditText" et 2 boutons.

Ex6.5 : Enfant1.java

```
package org.education.lgf.atelier;

import android.app.Activity;
import android.content.Intent;
import android.os.Bundle;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;
import android.widget.EditText;

public class Enfant1 extends Activity {
Button B_OK;
Button B_AN;
EditText votre_age;
private String age;
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main1);
        B_OK=(Button)findViewById(R.id.boutonOK);
        B_AN=(Button)findViewById(R.id.boutonANNULER);
        votre_age=(EditText)findViewById(R.id.editT);
        B_OK.setOnClickListener (new OnClickListener () {

suite du code sur la page suivante
```

```
                public void onClick (View view) {
                        age=votre_age.getText().toString();
                        Intent resultat= new Intent ();
                        resultat.putExtra("maclé",age);
                        setResult(RESULT_OK, resultat);
                        finish();
                }
        });
        B_AN.setOnClickListener (new OnClickListener () {
                public void onClick (View view) {
                        setResult(RESULT_CANCELED);
                        finish();
                }
        });
    }
}
```

"`Intent resultat= new Intent ();`" nous créons une instance de type "Intent" (remarquez la parenthèse vide : pas d'activité "cible" et pas de "requestCode", cet intent va uniquement nous servir à "transporter" une information).
"`resultat.putExtra("maclé",age);`" permet d'ajouter une donnée de type "String" à notre intent "resultat"
"`setResult(RESULT_OK, resultat);`" nous renvoyons notre intent vers l'activité "Principale"

ex6.5 : Principale.java

```
package org.education.lgf.atelier;

import android.app.Activity;
import android.content.Intent;
import android.os.Bundle;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;
import android.widget.Toast;

public class Principale extends Activity {
    Button bouton;
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        bouton=(Button)findViewById(R.id.bouton);
        bouton.setOnClickListener(new OnClickListener(){
                public void onClick (View view){
                        Intent intent=new Intent(Principale.this,Enfant1.class);
                        startActivityForResult(intent,1);
                }
        });
    }
    public void onActivityResult (int requestCode, int resultCode, Intent data){
       switch (resultCode){
       case RESULT_OK:
                String age=data.getStringExtra("maclé");
                Toast.makeText(this,"Vous avez "+age+" ans", Toast.LENGTH_SHORT).show();
                return;
        case RESULT_CANCELED:
                        Toast.makeText(this,"Vous ne voulez pas donner votre age",
Toast.LENGTH_SHORT).show();
                        return;
       }
    }
}
```

Seule la méthode "`onActivityResult`" nous intéresse ici.
La ligne "`String age=data.getStringExtra("maclé");`" permet de récupérer la chaîne correspondant à la clé "maclé", ensuite, rien de neuf.

Les intents en mode implicite

Les intents permettent aussi de démarrer d'autres applications. Android propose une méthode originale pour lancer d'autres applications : le mode implicite.
Avec le mode implicite, vous n'avez pas besoin de connaître le nom de l'application, il suffit de "dire" à Android : "Peux-tu démarrer une application qui me permettra de lire une vidéo" ou "Lance une application qui me permettra d'appeler le numéro de téléphone suivant : 0450087856". Vous n'avez pas besoin de préciser l'application à lancer, Android se charge de tout !

Notre objet de type "Intent" devra recevoir 2 paramètres, une constante et un objet de type "Uri" ??????

La constante

Le système Android propose plusieurs actions natives, chaque action est représentée par une constante static de la classe "Intent". Voici quelques exemples (voir la documentation officielle pour une liste plus exhaustive) :
ACTION_DIAL : permet de lancer l'interface de composition des numéros (le numéro de téléphone peut-être prérempli)
ACTION_CALL : même chose que ci-dessus, si le numéro est prérempli, ce numéro est automatiquement appelé.
ACTION_SENDTO : permet d'envoyer un message à un des contacts entrés dans le téléphone.
ACTION_VIEW : Permet de visualiser l'élément identifié par l'uri (voir ci-dessous pour une explication sur l'uri). C'est largement l'action la plus utilisée.

L'uri

Que dit wikipedia sur les uri ?
"Un **URI**, de l'anglais ***Uniform Resource Identifier***, soit littéralement *identifiant uniforme de ressource*, est une courte chaîne de caractères identifiant une ressource sur un réseau (par exemple une ressource Web) physique ou abstraite, et dont la syntaxe respecte une norme"

Source Wikipedia

Je pense que la définition est suffisamment claire. Les fameuses adresses "url" sont des "uri" "http://www.google.fr" est une "url" donc un "uri".

Pour transformer une chaîne de caractères en "uri" exploitable par un objet de type "Intent", il faut utiliser la méthode "parse("la chaîne de caractères à transformer en uri")".
Dans Android les "uri" sont des objets de type "Uri". Pour créer une instance de type "uri" il faudra donc utiliser la ligne :
Uri uri = Uri.parse("la chaîne de caractères à transformer en uri")

Voici un premier exemple (avec une url comme uri !)

ex6.6 : main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    >
<TextView
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="Cliquez sur le bouton pour afficher la page web www.google.fr"
    />
<Button
        android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:id="@+id/bouton"
    android:text="www.google.fr"
    />
</LinearLayout>
```

RAS : un TextView et un bouton.

ex6.6 : Principale.java

```
package org.education.lgf.atelier;

import android.app.Activity;
import android.content.Intent;
import android.net.Uri;
import android.os.Bundle;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;

public class Principale extends Activity {
Button bouton;
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        bouton=(Button)findViewById(R.id.bouton);
        bouton.setOnClickListener(new OnClickListener () {
               public void onClick (View view) {
                       Uri uri=Uri.parse("http://www.google.fr/");
                       Intent intent = new Intent (Intent.ACTION_VIEW, uri);
                       startActivity(intent);
               }

        });
    }
}
```

Rien de bien compliqué, tout a été expliqué au-dessus :
"uri" est une instance de la classe "Uri", la méthode "parse" transforme la chaîne de caractères en uri : "`Uri uri=Uri.parse("http://www.google.fr/");`".
Nous créons ensuite un intent avec comme paramètres du constructeur de la classe "Intent", la constante static et l'uri :"`Intent intent = new Intent (Intent.ACTION_VIEW, uri);`"

Autre exemple avec un numéro de téléphone et "ACTION_CALL"

ex6.7 : main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    >
<EditText
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:id="@+id/numT"
    android:hint="Entrez un numéro de téléphone"
    />
<Button
        android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:id="@+id/bouton"
    android:text="APPEL"
    />
</LinearLayout>
```

ex6.7 : main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    >
<Button
        android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:id="@+id/bouton"
    android:text="APPEL"
    />
</LinearLayout>
```

Un simple bouton

ex6.7 : Principale.java

```
package org.education.lgf.atelier;

import android.app.Activity;
import android.content.Intent;
import android.net.Uri;
import android.os.Bundle;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;

public class Principale extends Activity {
Button bouton;
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        bouton=(Button)findViewById(R.id.bouton);
        bouton.setOnClickListener(new OnClickListener () {
               public void onClick (View view) {
                       Uri uri=Uri.parse("tel:65674453");
                       Intent intent = new Intent (Intent.ACTION_CALL,uri);
                       startActivity(intent);
               }

        });
    }
}
```

Rien à signaler non plus, l'uri doit être de la forme "tel:XXXXXX".

Si vous lancez le programme tel quel, cela ne fonctionnera pas !
Pourquoi ?
Par souci de sécurité, par défaut, l'OS interdit certaines actions aux applications,comme par exemple les appels téléphoniques depuis une application tierce.
Pour "autoriser" une application "à passer un coup de fil", il faut modifier le fichier "AndroidManifest.xml".

ex 6.7 AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
      package="org.education.lgf.atelier"
      android:versionCode="1"
      android:versionName="1.0">
    <uses-permission android:name="android.permission.CALL_PHONE"/>
    <application android:icon="@drawable/icon" android:label="@string/app_name">
        <activity android:name=".Principale"
                  android:label="@string/app_name">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>

    </application>
</manifest>
```

Une seule ligne a été ajoutée : "`<uses-permission android:name="android.permission.CALL_PHONE"/>`"

Une fois la modification effectuée, le programme fonctionne.

Pour terminer avec les intents en mode implicite, je vous signale que tout ce que nous avons vu plus haut sur l'utilisation de la méthode "putExtra" reste valable. Vous pouvez donc "passer" des informations supplémentaires grâce à cette méthode, même en mode implicite.

Encore une fois je vous encourage à consulter la documentation pour plus de détails sur les intents, les autorisations,...