Programmation sous Android

Table des matières

Android est un système d'exploitation mobile pour smartphones, tablettes tactiles, PDA, smartwatches et terminaux mobiles. C'est un système open source utilisant le noyau Linux. Il a été lancé par une startup du même nom rachetée par Google en 2005.

1. Introduction

1.1. À propos d'Android

À l'origine, « Android » était le nom d'une PME américaine, créée en 2003 puis rachetée par Google en 2005, qui avait la ferme intention de s'introduire sur le marché des produits mobiles. La gageure, derrière Android, était de développer un système d'exploitation mobile plus intelligent, qui ne se contenterait pas uniquement de permettre d'envoyer des SMS et transmettre des appels, mais qui devait permettre à l'utilisateur d'interagir avec son environnement.

Créée en novembre de l'année 2007 l'OHA[1], qui comptait à sa création 35 entreprises évoluant dans l'univers du mobile, a eut pour but de développer un système open source pour l'exploitation sur mobile et ainsi concurrencer les systèmes propriétaires, tels Windows Mobile et iOS. Cette alliance a pour logiciel vedette Android.

Il existe un certain nombre de bonnes pratiques qu'il faut absolument respecter dans le développement des applications. Sans quoi, l'utilisateur aura tendance à vouloir les désinstaller.

- Ne jamais bloquer le smartphone. N'oubliez pas qu'il fait aussi autre chose lorsque vous exécutez vos applications.
- Optimiserles algorithmes : votre smartphone n'est pas comparable à votre ordinateur en terme de performance.
- Adapter les interfaces à tous les types d'écran : les terminaux sont nombreux.
- Penser les interfaces pour les doigts de l'utilisateur final. S'il possède des gros doigts et que vous faites des petits boutons, l'expérience utilisateur en sera altérée.

1.2. Installation et configuration des outils

1.2.1 JDK, Eclipse, ADT et le SDK Android

Il faut télécharger les outils indispensables pour développer les applications Android :

- Le JDK[2], qui contient le JRE[3] et un ensemble d'outils pour compiler et déboguer le code.
- L'IDE[4] Eclipse, un environnement de développement spécialisé dans le développement Java.
- Le plugin ADT[5], qui est une extension d'Eclipse afin de développer des applications Android.
- Le SDK[6] Android, des outils pour gérer l'installation d'Android sur votre système.

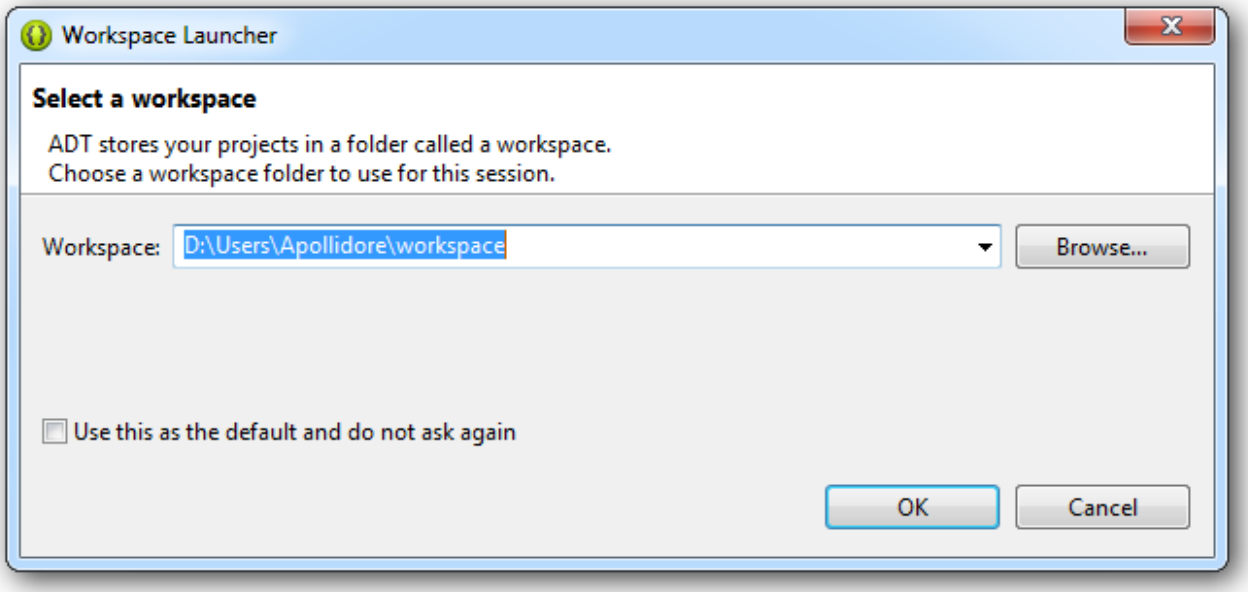
Pour télécharger le JDK suivez ce lien, puis cliquez sur Download en dessous de JDK :

1 Open Handset Alliance
2 Java Development Kit
3 Java Runtime Environment
4 Integrated Development Environment
5 Android Developer Tools
6 Software Development Kit

Pour télécharger le pack Eclipse, ADT, SDK suivez ce lien, puis cliquez sur Download the SDK :

Une fois le téléchargement terminé et l'archive décompressée rendez vous dans le répertoire adt-bundle-xxx/eclipse et ouvrez Eclipse :

Tout d'abord, une fenêtre va s'ouvrir pour vous demander où vous souhaitez installer le workspace, c'est-à-dire l'endroit où vous souhaitez que vos projets soient sauvegardés :

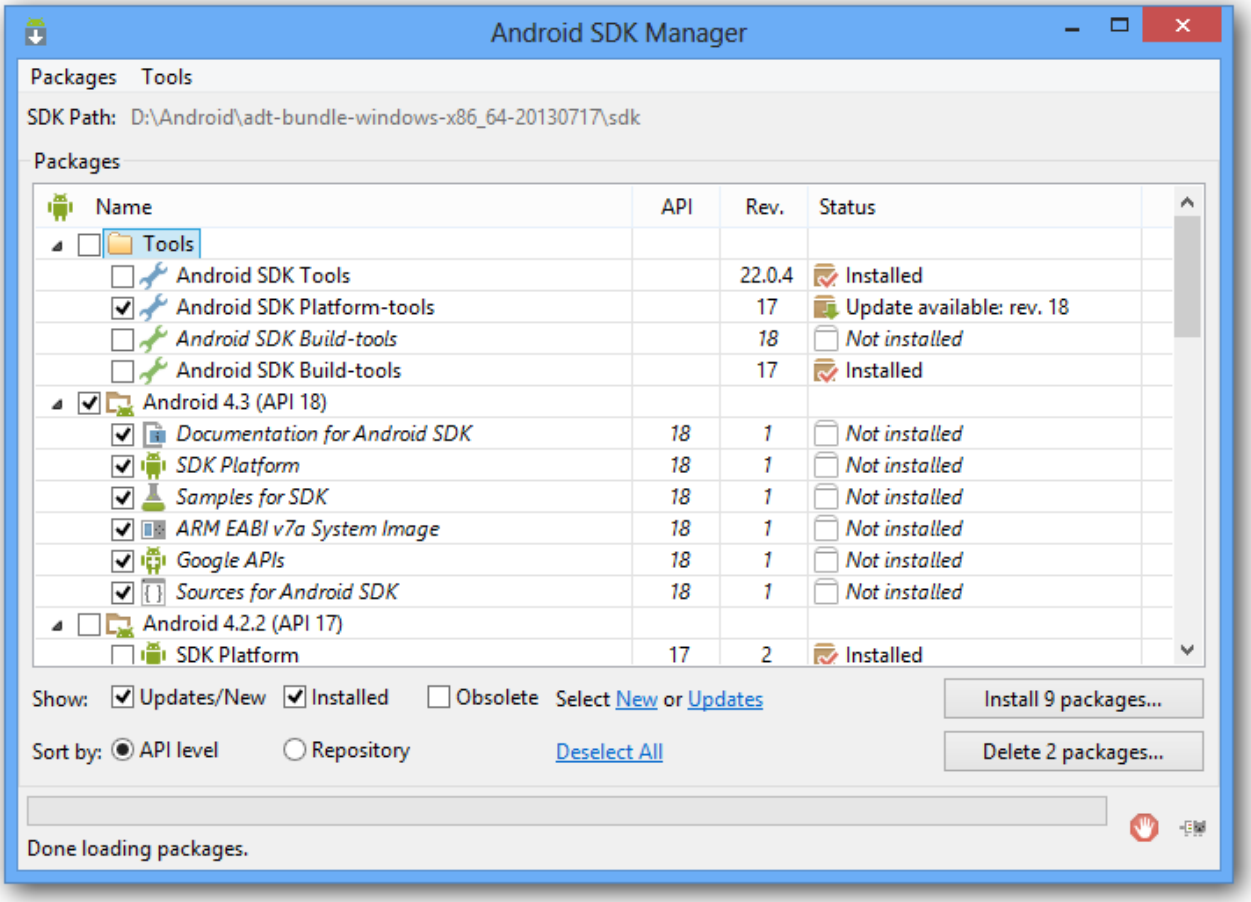


Une fois le logiciel ouvert, cliquez sur le bouton Android SDK Manager pour ouvrir l'outil de

gestion du SDK d'Android :

Le bouton Android SDK Manager est celui de gauche

Le Android SDK Manager s'ouvre et vous tomberez sur un écran similaire à celui-ci :



Cliquez sur « Install xx packages.... » et valider les licences des fichiers à télécharger

Chaque ligne correspond à un paquet, c'est-à-dire des fichiers qui seront téléchargés pour ajouter de nouvelles fonctionnalités au SDK d'Android. Par exemple vous pouvez voir que le paquet Android SDK Tools est déjà installé (installed). En revanche, Documentation for Android SDK n'est pas installé (Not installed).

Le nom des paquets suivent un certain motif. Il est écrit à chaque fois Android [un nombre] (API [un autre nombre]). La présence de ces nombres s'explique par le fait qu'il existe plusieurs versions de la plateforme Android qui ont été développées depuis ses débuts et qu'il existe donc plusieurs versions différentes en circulation.

Le premier nombre correspond à la version d'Android et le second à la version de l'API[7] Android associée. Quand on développe une application, il faut prendre en compte ces numéros, puisqu'une application développée pour une version précise d'Android ne fonctionnera pas pour les versions

7 Application Programming Interface, ensemble de règles à suivre pour pouvoir dialoguer avec d'autres applications

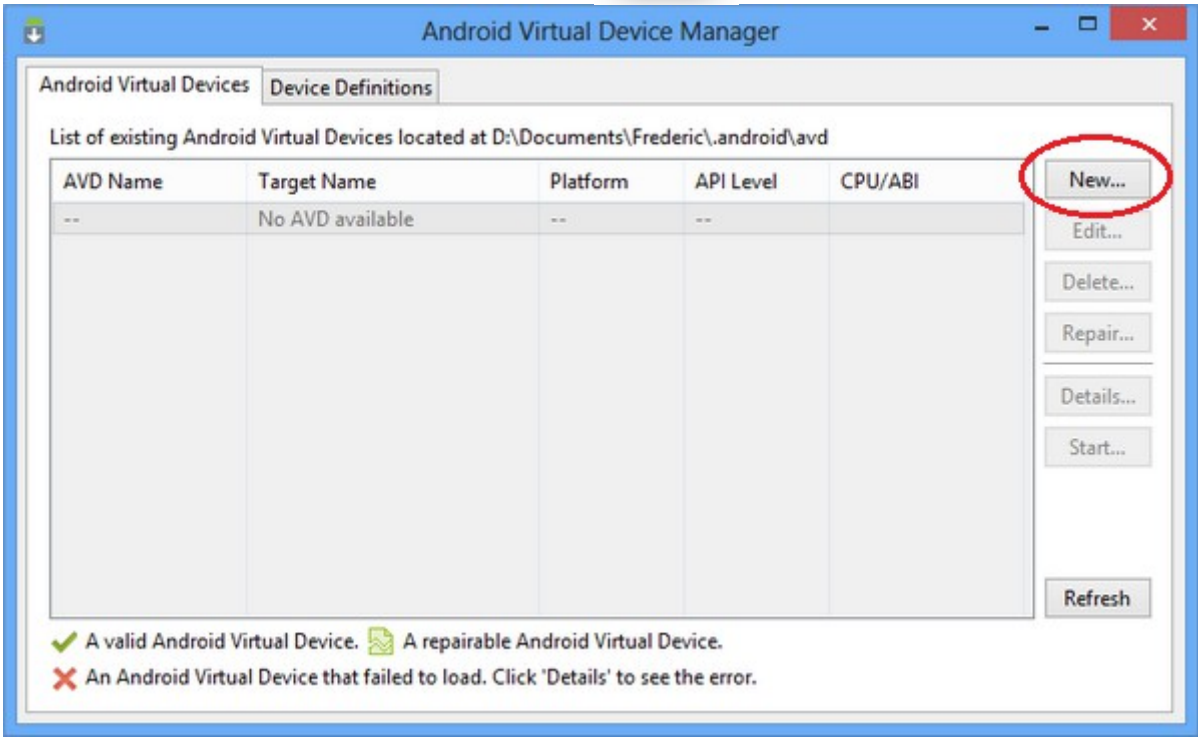
précédentes.

Les API dont le numéro est compris entre 11 et 13 sont normalement destinées aux tablettes. En théorie, vous n'avez pas à vous en soucier, les applications développées avec les API numériquement inférieures fonctionneront, mais il y aura des petits efforts à fournir en revanche en ce qui concerne l'interface graphique.

1.2.2. L'émulateur de téléphone : ADV

L'AVD[8] est un émulateur de terminal sous Android, c'est-à-dire que c'est un logiciel qui se fait passer pour un appareil sous Android à votre ordinateur. C'est la raison pour laquelle vous n'avez pas besoin d'un périphérique sous Android pour développer et tester la plupart de vos applications ! En effet, une application qui affiche un calendrier par exemple peut très bien se tester dans un émulateur, mais une application qui exploite le GPS doit être éprouvée sur le terrain pour que l'on soit certain de son comportement.

Ouvrez l'interface de gestion d'AVD avec l'icone 



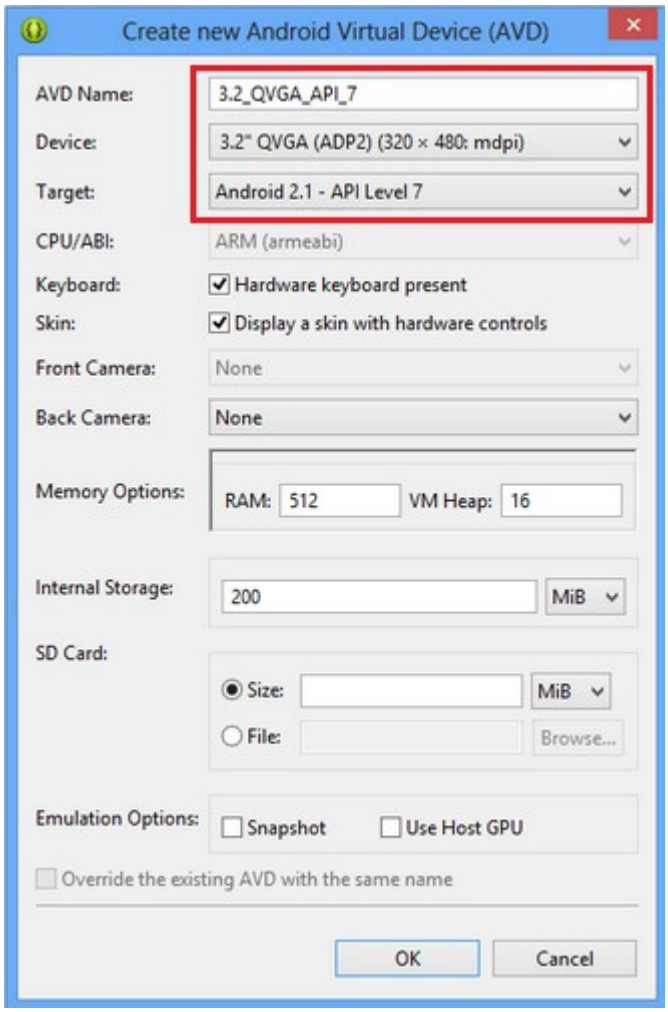
Dans le premier onglet, cliquez sur New… pour ajouter un nouvel AVD

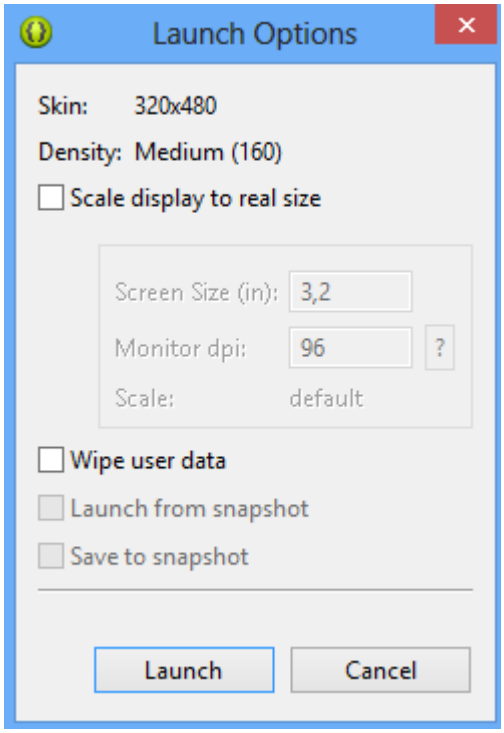
Vous pouvez voir deux onglets : Android Virtual Devices et Device Definitions. Le premier onglet recense tous vos AVD et vous permet d'en créer avec précision, alors que le second onglet vous permet de créer des onglets qui ressemblent à des machines qui existent réellement, par exemple le Nexus S de Google.

8 Android Virtual Device

Une fenêtre s'ouvre, vous proposant de créer votre propre émulateur. Indiquer un nom pour l'AVD (ex : 3.2_QVGA_API_7 : la taille de l'écran et la version de l'API sous laquelle fonctionnera cet AVD). Notez que certains caractères comme les caractères accentués et les espaces ne sont pas autorisés. Vous pouvez choisir la taille de l'écran à l'aide de Device (ex : 3.2" pour une résolution de 320 x 480).

Dans Target, choisissez Android 2.1 - API Level 7, puisque nous ferons des applications avec la version 7 de l'API et sans le Google API.





Une fois l'AVD créé, cliquez sur « le bouton Start… » dans l'AVD Manager.

Puis cliquez sur « Launch ».

Il se peut que l'émulateur soit **très lent** car l'ordinateur doit émuler un téléphone et exécuter des instructions processeurs qui respectent l'architecture ARM[9].

La liste de droite contient les boutons permettant d'imiter les boutons d'un téléphone :

contrôles de base

: Diminuer le volume.

: Augmenter le volume.

boutons de navigation

: Retourner sur le dashboard

: Ouvrir le menu.

: Retour arrière.

: Effectuer une recherche

Vous pouvez manipuler la partie de gauche avec votre souris, ce qui simulera le tactile. Faites glisser le verrou sur la gauche pour déverrouiller la machine. Vous vous retrouverez sur l'accueil. Cliquez sur le bouton MENU à droite pour accéder à l'option Settings.

1.3. Première application

Une application Android affiche affiche souvent plusieurs fenêtres. Ces différentes fenêtres sont appelées des activités. Un moyen efficace de différencier des activités est de comparer leur interface graphique : si elles sont radicalement différentes, c'est qu'il s'agit d'activités différentes.

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

9 Advanced RISC Machines

Le cadre bleu montre les limites de l'activité

Le cadre bleu montre les limites de l'activité

Si un utilisateur navigue sur un site avec son téléphone, le tout en écoutant de la musique sur ce même téléphone. Il se passe deux choses dans votre système :

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

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

Si l'utilisateur reçoit un appel, il devient plus important qu'il puisse y répondre que d'émettre la chanson que votre application diffuse. Ainsi :

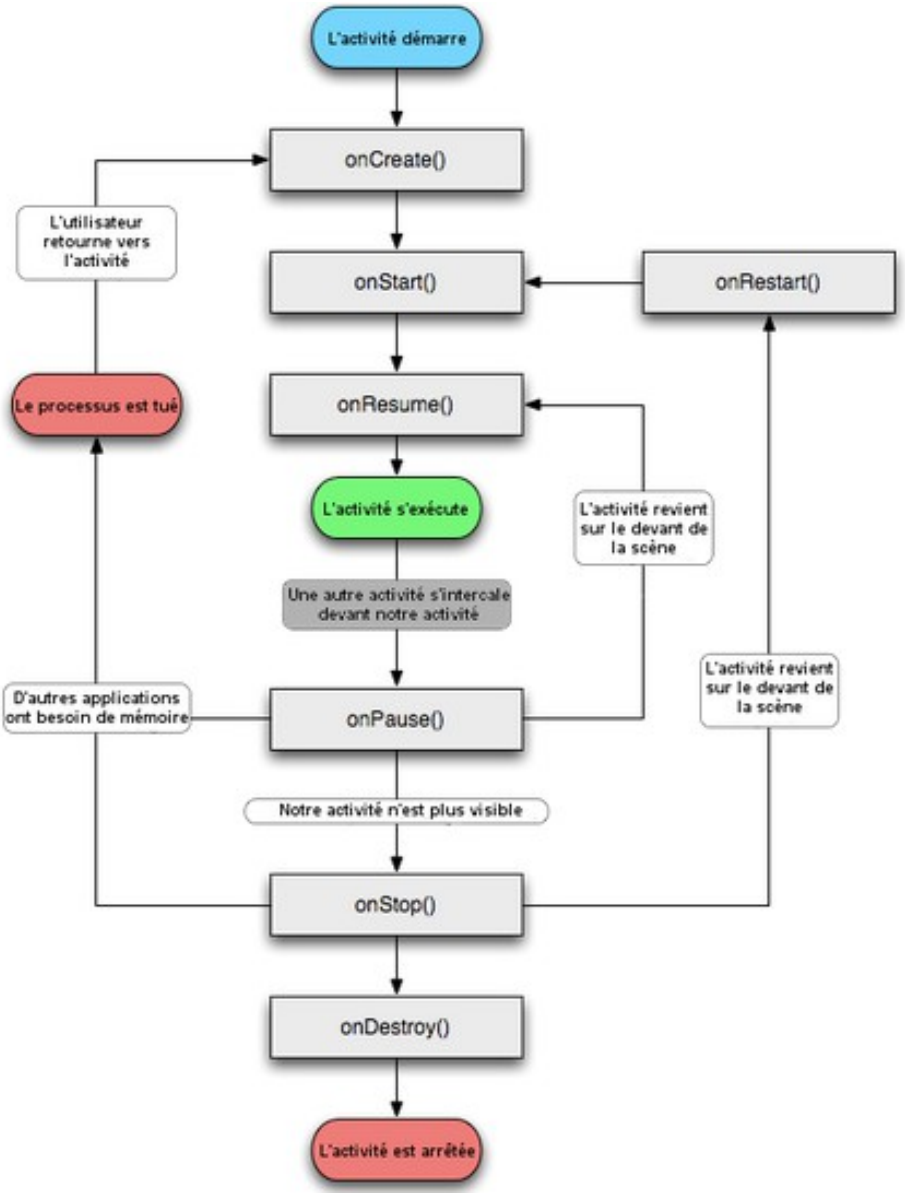
- À tout moment l'application peut laisser place à une autre application, qui a une priorité plus élevée. Si l'application utilise trop de ressources système, alors elle empêchera le système de fonctionner correctement et Android l'arrêtera sans vergogne.
- L'activité existera dans plusieurs états au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

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

État	**Visibilité**	**Description**
Active (« active » ou « running »)	L'activité est visible en totalité.	C'est cette application qui a le focus, c'est-à-dire que l'utilisateur agit directement sur l'application.
Suspendue (« paused »)	L'activité est partiellement visible à l'écran.	Ce n'est pas sur cette activité qu'agit l'utilisateur. L'application n'a plus le focus, c'est l'application sus-jacente qui l'a. Pour que notre application récupère le focus, l'utilisateur devra se

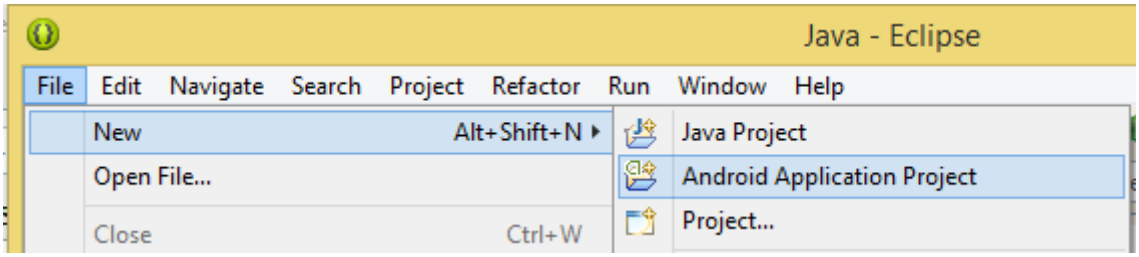
		débarrasser de l'application qui l'obstrue, puis l'utilisateur pourra à nouveau interagir avec.
Arrêtée (« stopped »)	L'activité est tout simplement oblitérée par une autre activité, on ne peut plus la voir du tout.	L'application n'a évidemment plus le focus, et puisque l'utilisateur ne peut pas la voir, il ne peut pas agir dessus. Le système retient son état pour pouvoir reprendre, mais il peut arriver que le système tue l'application pour libérer de la mémoire système.

Une activité n'a pas de contrôle direct sur son propre état, il s'agit plutôt d'un cycle rythmé par les interactions avec le système et d'autres applications. Voici un schéma qui présente ce que l'on appelle le cycle de vie d'une activité, c'est-à-dire qu'il indique les étapes que va traverser l'activité pendant sa vie, de sa naissance à sa mort. Chaque étape du cycle est représentée par une méthode.



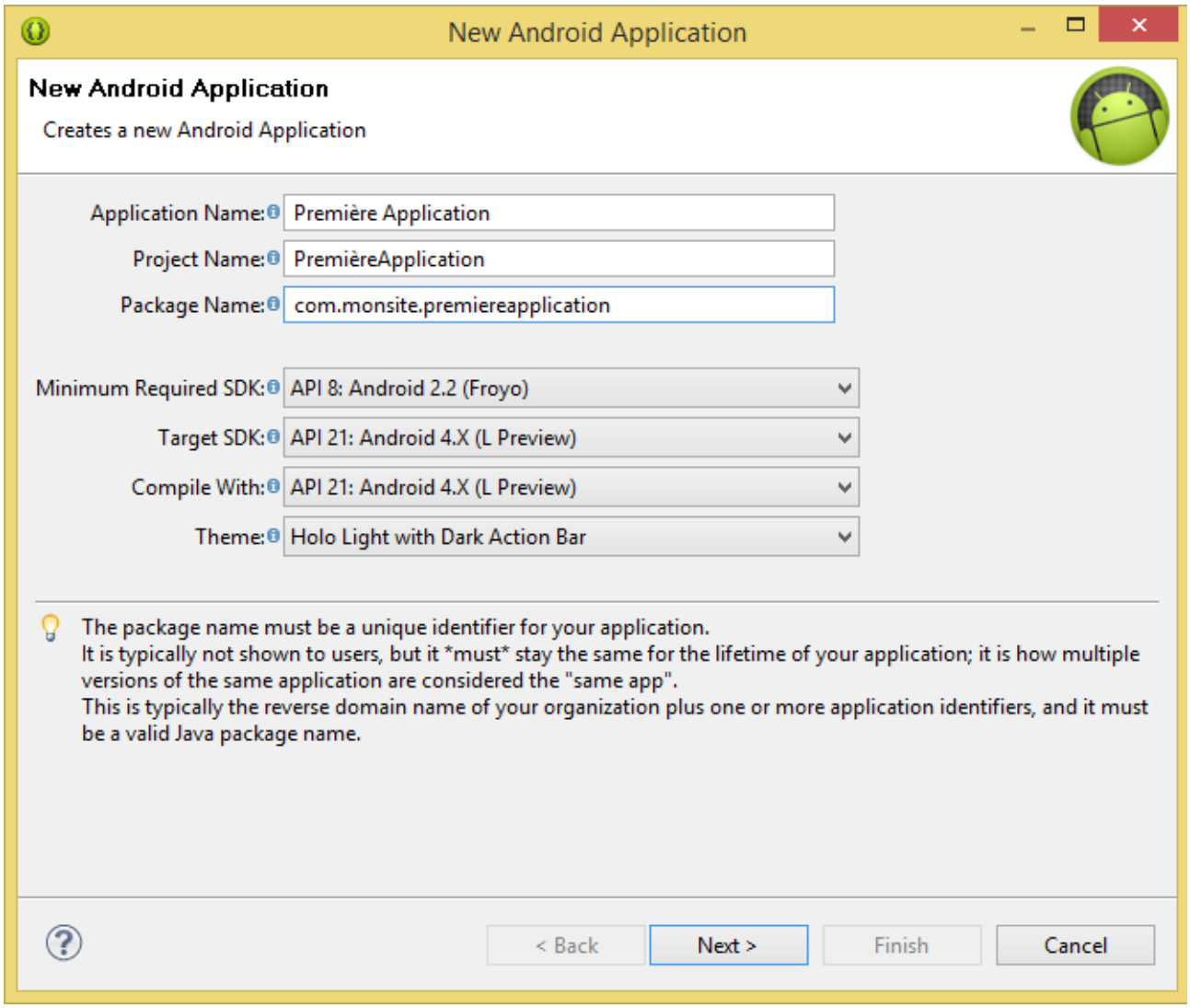
Les activités héritent de la classe Activity. Or, la classe Activity hérite de l'interface Context dont le but est de représenter tous les composants d'une application. On les trouve dans le package[10] android.app.Activity.

Une fois Eclipse démarré, cliquez sur File/New/Android Application Projet



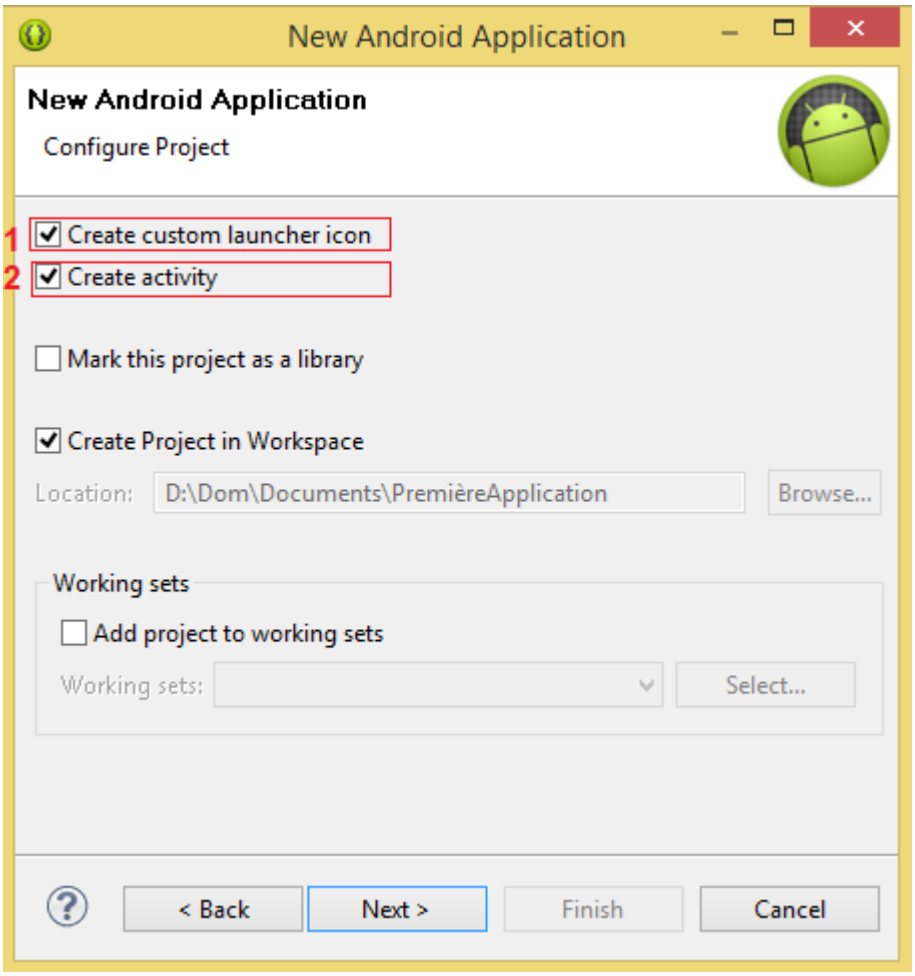
Une nouvelle fenêtre s'ouvre :

10 répertoire qui permet d'organiser le code source



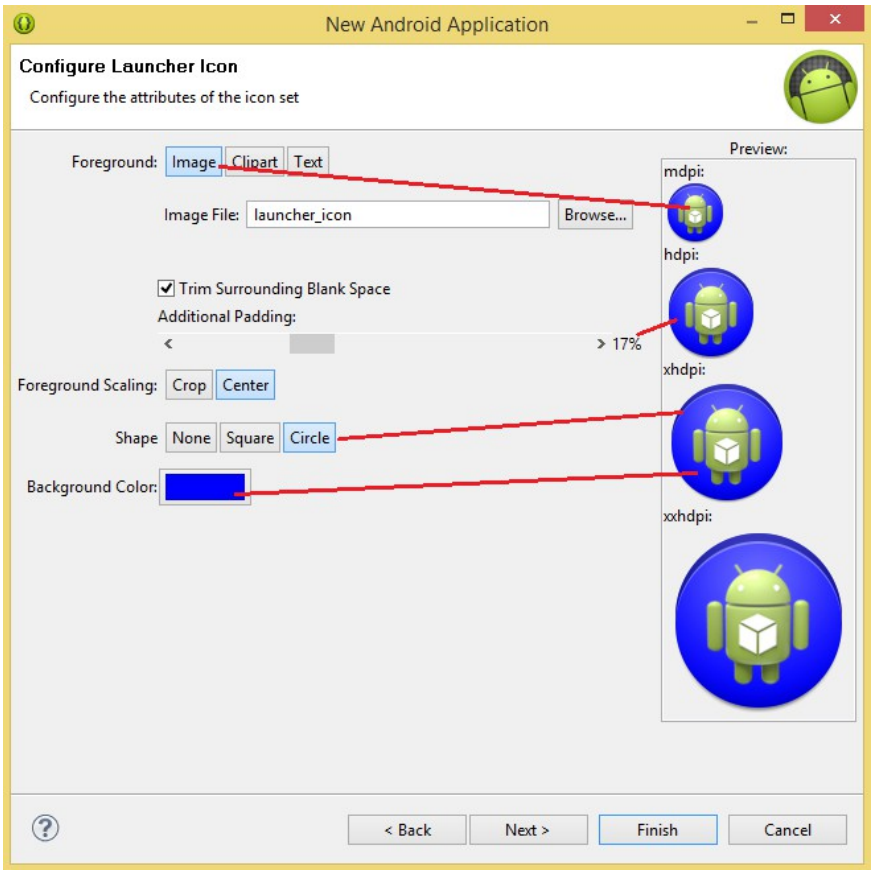
- Application name : nom qui apparaîtra sur l'appareil et sur Google Play pour les futures applications.
- Project name : nom du projet pour Eclipse. Ce champ n'influence pas l'application en elle-même, il s'agit du nom sous lequel Eclipse la connaîtra.
- Project name : package où ira l'application. Ce package agira comme une sorte d'identifiant pour l'application sur le marché d'applications, faire en sorte qu'il soit unique et constant pendant toute la vie de l'application. En général on se base sur le nom de domaine de son entreprise pour le constitué.
- Minimum Required SDK : version minimale pour laquelle l'application est destinée. Cette information sera utilisée sur Google Play pour proposer vos applications à des clients.
- Target SDK : inverse de Minimum Required SDK, il s'agit de la version maximale sous laquelle l'application fonctionne.
- Compile With : permet de choisir pour quelle version du SDK on va compiler l'application. Il s'agit cette fois de la version minimale nécessaire pour utiliser votre application.

Cliquez sur Next :

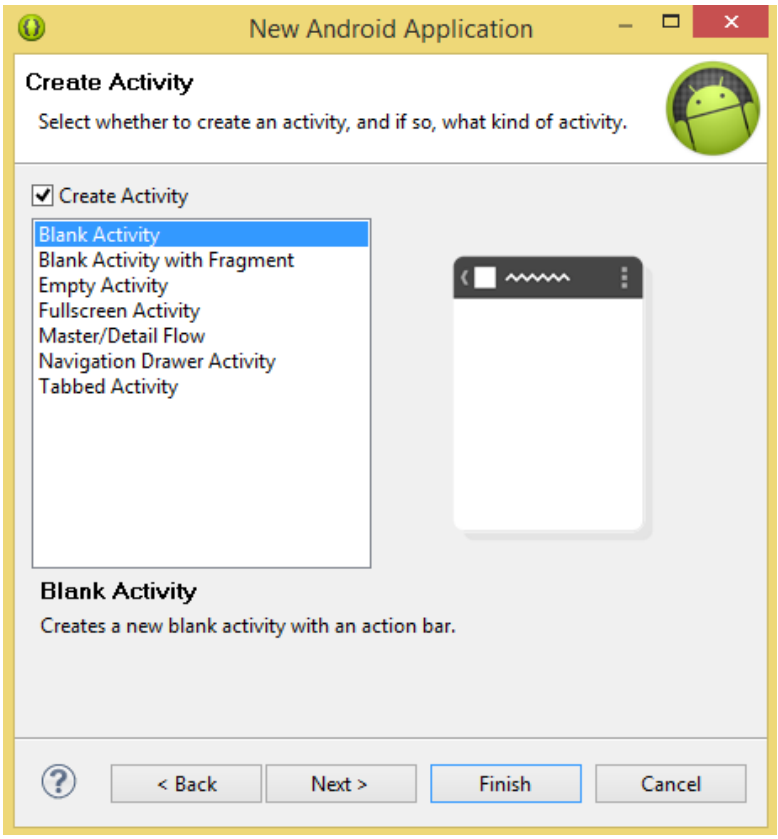


1. Create custom launcher icon, ouvrira à la fenêtre suivante un outil pour vous aider à construire une icône pour l'application à partir d'une image préexistante.
2. Create activity, permet de vous faciliter le développement de l'application en faisant faire une partie par Eclipse.

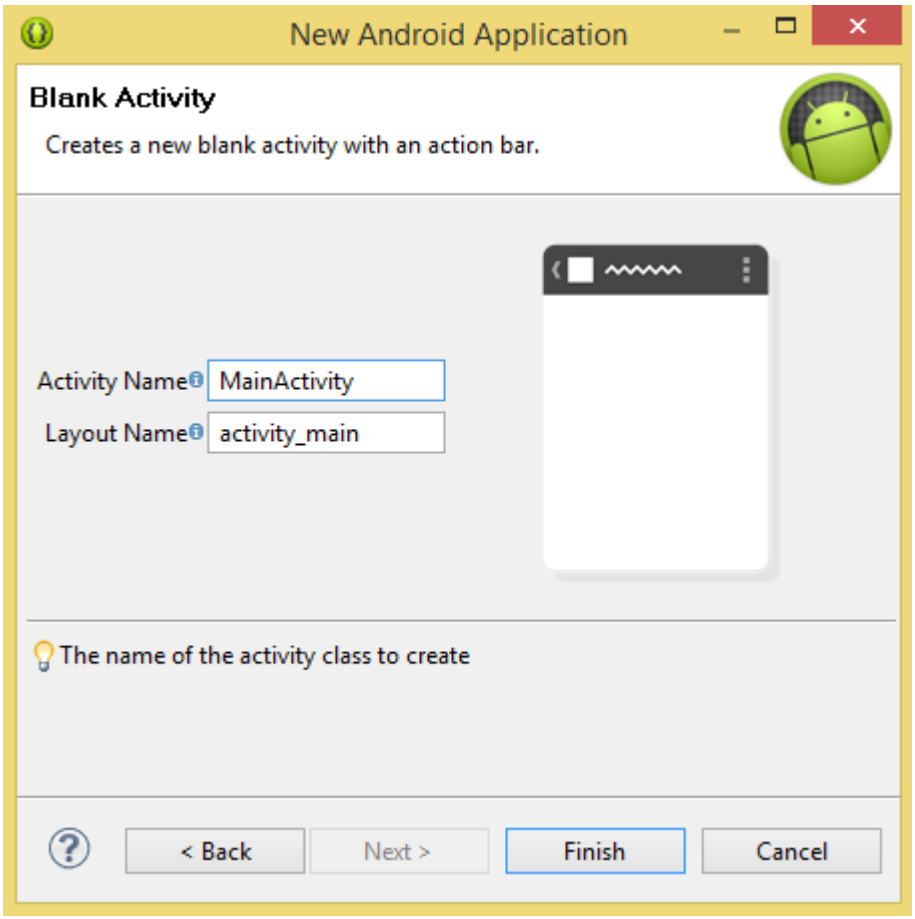
Cliquez sur Next :



Cliquez sur Next :

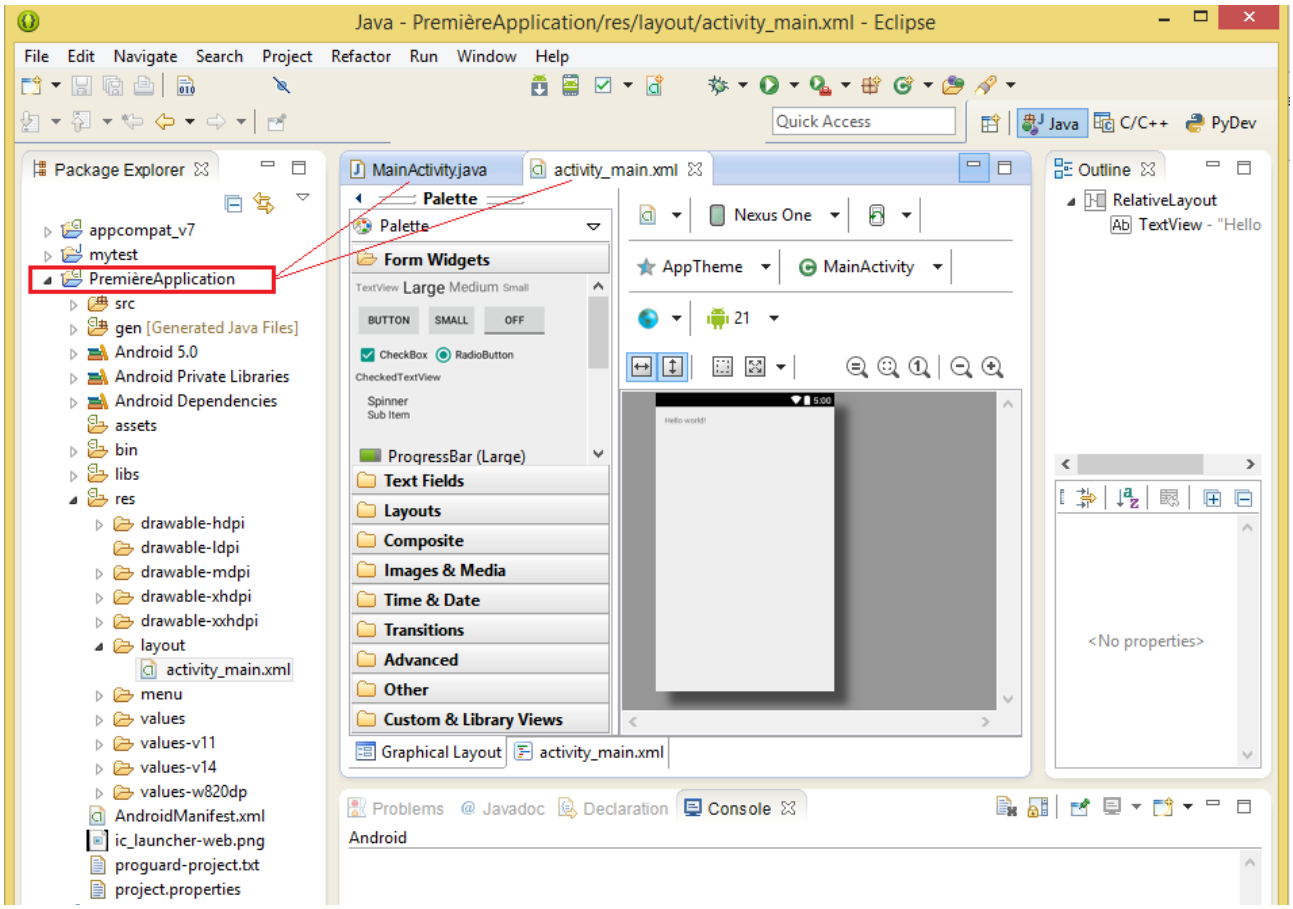


Sélectionner BlankActivity pour rester maître de la mise en page, puis Cliquer sur Next :



- Activity Name : indique le nom de la classe Java qui contiendra votre activité, ce champ doit donc respecter la syntaxe Java standard.
- Layout Name : nom du fichier qui contiendra l'interface graphique qui correspondra à cette activité.
- Navigation Type : permet de définir facilement comment s'effectueront les transitions entre plusieurs activités.

Pour finaliser la création, cliquez sur Finish :



Les fichiers créés sont visibles par le Package Explorer. On y trouve le répertoire src/, celui qui contient tous les fichiers sources .java. Ouvrez le fichier MainActivity.java qui s'y trouve :

```
package com.monsite.premiereapplication;

import android.support.v7.app.ActionBarActivity;
import android.os.Bundle;
import android.view.Menu;
import android.view.MenuItem;

public class MainActivity extends ActionBarActivity {

	@Override
	protected void onCreate(Bundle savedInstanceState) {
		super.onCreate(savedInstanceState);
		setContentView(R.layout.activity_main);
	}

	@Override
	public boolean onCreateOptionsMenu(Menu menu) {
		// Inflate the menu; this adds items to the action bar if present.
		getMenuInflater().inflate(R.menu.main, menu);
		return true;
	}

	@Override
	public boolean onOptionsItemSelected(MenuItem item) {
		// Handle action bar item clicks here. The action bar will
```

```
            // automatically handle clicks on the Home/Up button, so long
            // as you specify a parent activity in AndroidManifest.xml.
            int id = item.getItemId();
            if (id == R.id.action_settings) {
                  return true;
            }
            return super.onOptionsItemSelected(item);
      }
}
```

Commentons le code :

```
package com.monsite.premiereapplication;
```

On déclare que notre programme se situe dans le package com.monsite.premiereapplication, comme expliqué précédemment. Pour faire référence à l'application, il faudra faire référence à ce package.

```
import android.support.v7.app.ActionBarActivity;
import android.os.Bundle;
import android.view.Menu;
import android.view.MenuItem;
```

On importe des classes qui se trouvent dans des packages différents.

```
public class MainActivity extends ActionBarActivity {
      // ...
}
```

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

```
@Override
protected void onCreate(Bundle savedInstanceState) {
      // ...
}
```

@Override permet d'indiquer que l'on va redéfinir une méthode qui existait auparavant dans la classe parente, puisqu'une activité avait une méthode void onCreate() et que la classe hérite de ActionBarActivity (l'instruction @Override est facultative).

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

- S'il s'agit du premier lancement de l'application ou d'un démarrage alors qu'elle avait été quittée normalement, il vaut null.
- S'il s'agit d'un retour à l'application après qu'elle a perdu le focus et redémarré, alors il se peut qu'il ne soit pas null si des données ont été sauvegardées dedans.

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

```
super.onCreate(savedInstanceState);
```

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

Ainsi, super.onCreate fait appel au onCreate de la classe Activity, mais pas au onCreate de

MainActivity. Il gère bien entendu le cas où le Bundle est null. Cette instruction est obligatoire.

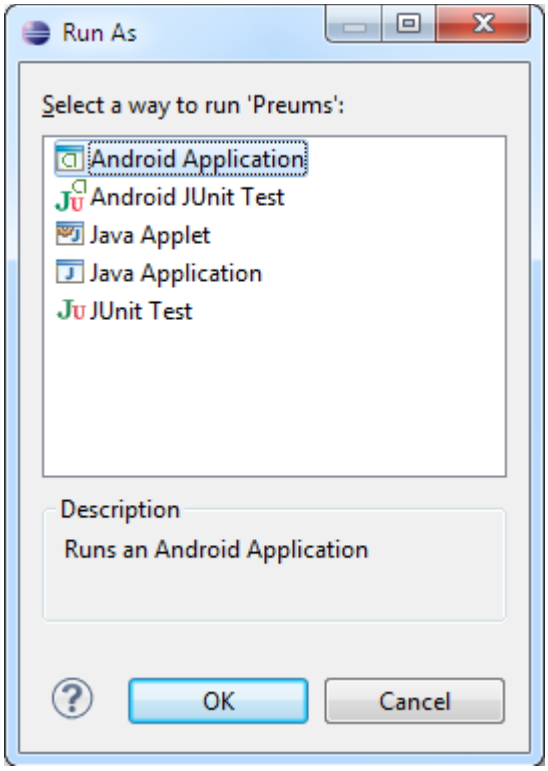
Vous pouvez remplacer le contenu du fichier par celui-ci :

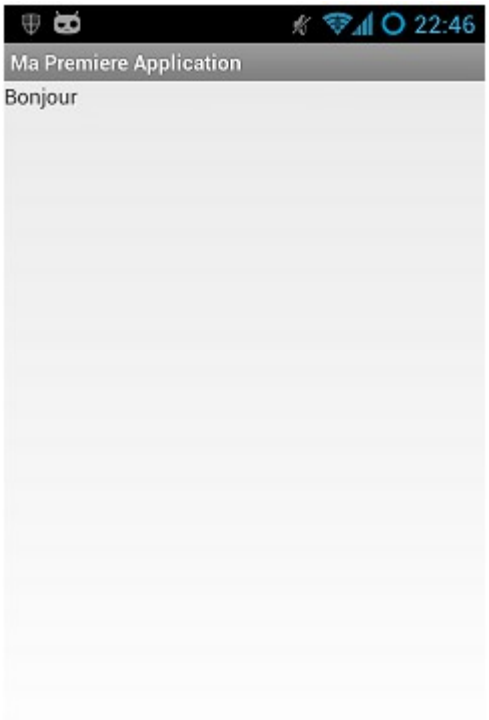
```
package com.monsite.premiereapplication;

import android.support.v7.app.ActionBarActivity;
import android.os.Bundle;
import android.widget.TextView;

public class MainActivity extends ActionBarActivity {
	private TextView texte = null;

	@Override
	protected void onCreate(Bundle savedInstanceState) {
		super.onCreate(savedInstanceState);

		texte = new TextView(this);
		texte.setText("Bonjour");
		setContentView(texte);
	}
}
```

Un attribut de classe de type TextView a été ajouté. Il s'agit d'une vue (View)… qui représente un texte (Text) qui s'affichera avec la méthode void setText(String texte).

La méthode void setContentView (View vue) permet d'indiquer l'interface graphique de l'activité.

Pour lancer l'application, il suffit de cliquer sur le bouton de la barre d'outils d'Eclipse qui ressemble au symbole « play »  . Si une fenêtre s'ouvre, sélectionnez Android Application.



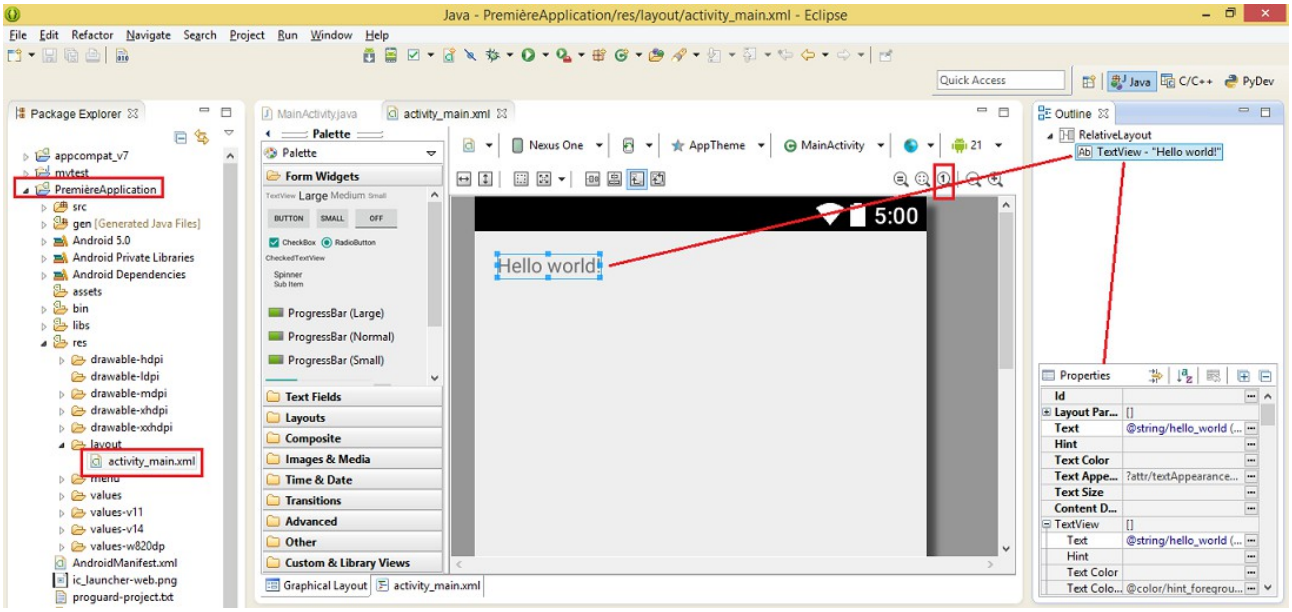


2. Création d'interfaces graphiques

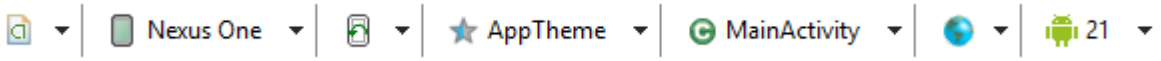
Avec la diversité des machines sous lesquelles fonctionne Android, il faut exploiter les ressources pour développer des applications qui fonctionneront sur la majorité des terminaux. Une application Android polyvalente possède un fichier XML pour chaque type d'écran, de façon à pouvoir s'adapter.

2.1. Constitution des interfaces graphiques

Ouvrez le fichier activity_main.xml qui se trouve dans le répertoire res/layout :



Le menu en haut permet d'observer le résultat avec différentes options :



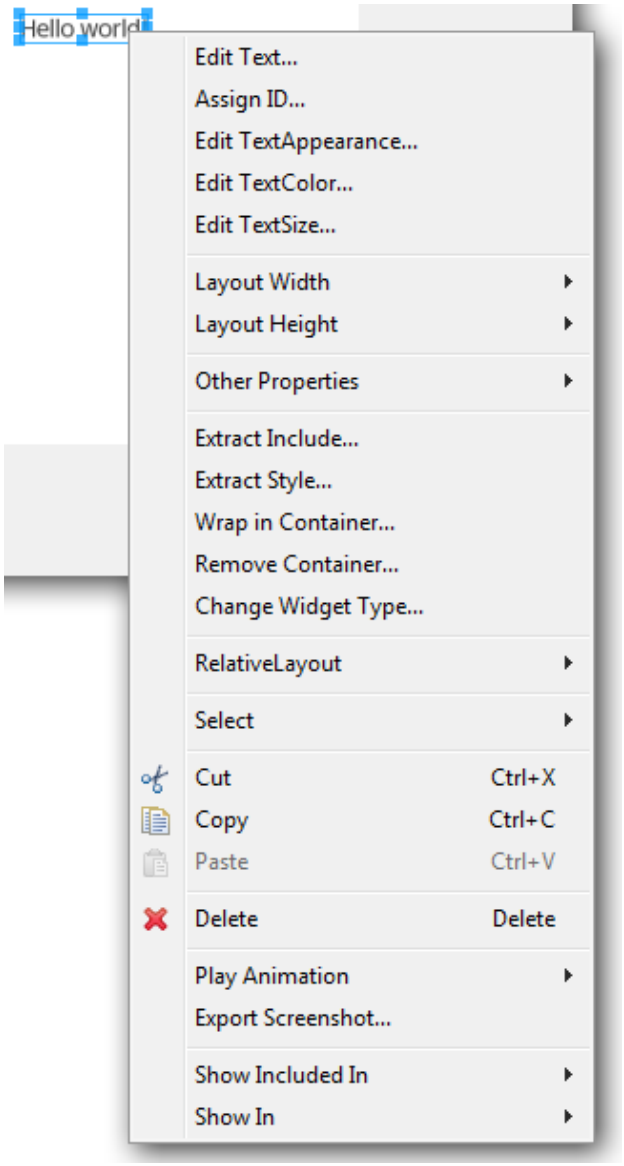
- La première liste déroulante permet de naviguer rapidement entre les répertoires de layouts.
- La deuxième permet d'observer le résultat en fonction de différentes résolutions. Le chiffre indique la taille de la diagonale en pouces et la suite de lettres en majuscules la résolution de l'écran.
- La troisième permet d'observer l'interface graphique en fonction de certains facteurs (mode portrait ou en mode paysage, matériel d'amarrage, ...)
- La suivante permet d'associer un thème à votre activité.
- L'avant-dernière permet de choisir une langue.
- La dernière vérifie le comportement en fonction de la version de l'API.

Les boutons suivants sont spécifiques à un composant et à son layout parent, contrairement aux boutons précédents qui étaient spécifiques à l'outil.

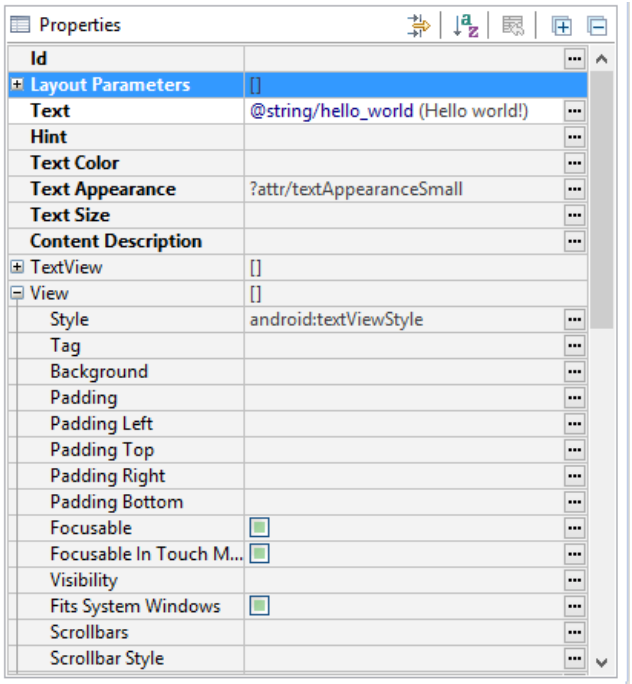
Boutons de droite :

- Le premier bouton permet de modifier l'affichage en fonction d'une résolution.
- Le deuxième fait en sorte que l'interface graphique fasse exactement la taille de la fenêtre dans laquelle elle se trouve.
- Le suivant remet le zoom à 100%.
- Enfin les deux suivants permettent respectivement de dézoomer et de zoomer.

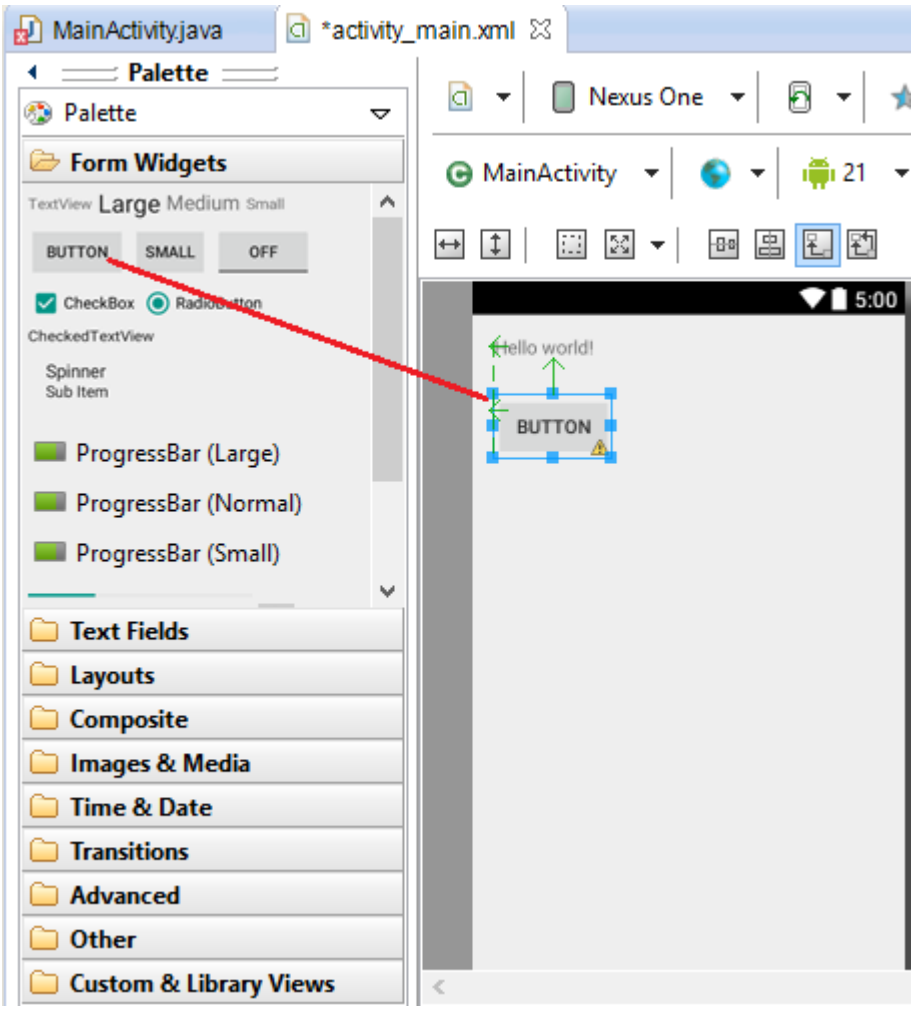
En faisant un click droit sur l'objet TextView, on obtient le menu contextuel qui permet de modifer les propriétés de la vue :



Ou bien en utilisant le paneau Properties :



De plus, il est possible de placer différentes vues en cliquant dessus depuis le menu de gauche, puis en les déposant sur l'activité :



2.1.1. Règles générales sur les vues

Eclipse a créé un fichier XML par défaut :

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
  xmlns:tools="http://schemas.android.com/tools"
  android:layout_width="fill_parent"
  android:layout_height="fill_parent" >

  <TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_centerHorizontal="true"
    android:layout_centerVertical="true"
    android:padding="@dimen/padding_medium"
    android:text="@string/hello_world"
    tools:context=".MainActivity" />

</RelativeLayout>
```

La racine possède deux attributs similaires : xmlns:android="http://schemas.android.com/apk/res/android" et xmlns:tools="http://schemas.android.com/tools". Ces deux lignes permettent d'utiliser des attributs spécifiques à Android. Si vous ne les mettez pas, vous ne pourrez pas utiliser les attributs et le

fichier XML sera un fichier XML banal au lieu d'être un fichier spécifique à Android. De plus, Eclipse refusera de compiler.

On trouve ensuite une racine qui s'appelle RelativeLayout et qui englobe un autre nœud qui s'appelle TextView.

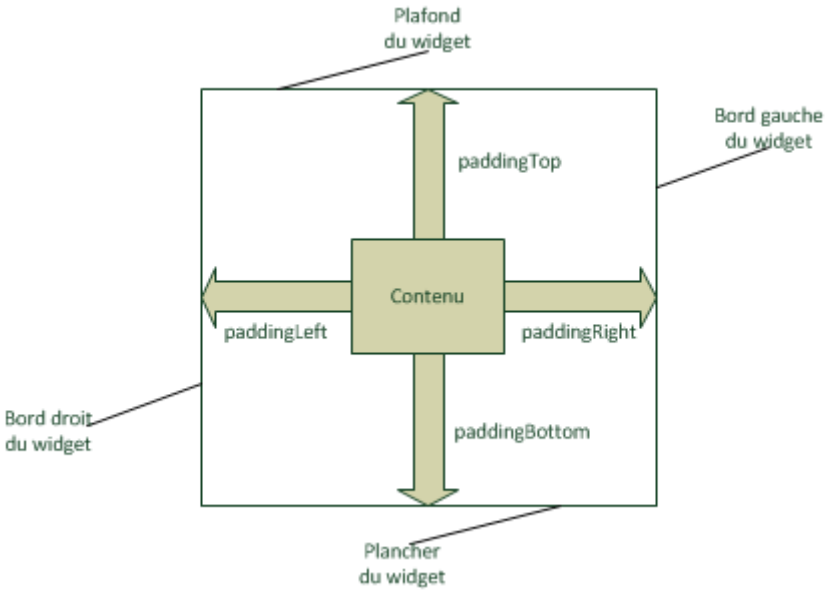
Un layout[11] est donc une vue spéciale qui peut contenir d'autres vues et qui n'est pas destinée à fournir du contenu ou des contrôles à l'utilisateur. Les layouts se contentent de disposer les vues d'une certaine façon. Les vues contenues sont les enfants, la vue englobante est le parent, comme en XML. Une vue qui ne peut pas en englober d'autres est appelée un widget[12].

Un layout hérite de ViewGroup (classe abstraite, qu'on ne peut donc pas instancier), et ViewGroup hérite de View.

Comme beaucoup de nœuds en XML, une vue peut avoir des attributs, qui permettent de moduler certains de ses aspects. Certains de ces attributs sont spécifiques à des vues, d'autres sont communs. Parmi ces derniers, les deux les plus courants sont layout_width, qui définit la largeur que prend la vue (la place sur l'axe horizontal), et layout_height, qui définit la hauteur qu'elle prend (la place sur l'axe vertical). Ces deux attributs peuvent prendre une valeur parmi les trois suivantes :

- fill_parent : signifie qu'elle prendra autant de place que son parent sur l'axe concerné ;
- wrap_content : signifie qu'elle prendra le moins de place possible sur l'axe concerné. Par exemple si votre vue affiche une image, elle prendra à peine la taille de l'image, si elle affiche un texte, elle prendra juste la taille suffisante pour écrire le texte ;
- Une valeur numérique précise avec une unité.

Il est possible de définir une marge interne pour chaque widget, autrement dit l'espacement entre le contour de la vue et son contenu :



Ci-dessous avec l'attribut android:padding dans le fichier XML pour définir un carré d'espacement ; la valeur sera suivie d'une unité, 10.5dp par exemple.

```
<TextView
```

11 gabarit
12 composant

```
  android:layout_width="fill_parent"
  android:layout_height="wrap_content"
  android:padding="10.5dp"
  android:text="@string/hello" />
```

La méthode Java équivalente est public void setPadding (int left, int top, int right, int bottom).

```
textView.setPadding(15, 105, 21, 105);
```

En XML on peut aussi utiliser des attributs android:paddingBottom pour définir uniquement l'espacement avec le plancher, android:paddingLeft pour définir uniquement l'espacement entre le bord gauche du widget et le contenu, android:paddingRight pour définir uniquement l'espacement de droite et enfin android:paddingTop pour définir uniquement l'espacement avec le plafond.

2.1.2. Identifier et récupérer des vues

Il est possible d'accéder à une ressource à partir de son identifiant à l'aide de la syntaxe @X/Y. Le @ signifie qu'on va parler d'un identifiant, le X est la classe où se situe l'identifiant dans R.java et enfin, le Y sera le nom de l'identifiant. Bien sûr, la combinaison X/Y doit pointer sur un identifiant qui existe. Ainsi, dans la classe créée par défaut :

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
  xmlns:tools="http://schemas.android.com/tools"
  android:layout_width="fill_parent"
  android:layout_height="fill_parent" >

  <TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_centerHorizontal="true"
    android:layout_centerVertical="true"
    android:padding="@dimen/padding_medium"
    android:text="@string/hello_world"
    tools:context=".MainActivity" />

</RelativeLayout>
```

D'après la ligne surlignée, le TextView affichera le texte de la ressource qui se trouve dans la classe String de R.java et qui s'appelle hello_world.

Afin de créer un identifiant, on peut rajouter à chaque vue un attribut android:id. La valeur doit être de la forme @+X/Y. Le + signifie qu'on parle d'un identifiant qui n'est pas encore défini. En voyant cela, Android sait qu'il doit créer un attribut.

La syntaxe @+X/Y est aussi utilisée pour faire référence à l'identifiant d'une vue créée plus tard dans le fichier XML.

Le X est la classe dans laquelle sera créé l'identifiant. Si cette classe n'existe pas, alors elle sera créée. Traditionnellement, X vaut id et Y est le nom de l'identifiant. Cet identifiant doit être unique au sein de la classe.

Exemple : on va appeler le TextView « text » et changer le padding pour qu'il vaille 25.7dp[13].

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
  xmlns:tools="http://schemas.android.com/tools"
  android:layout_width="fill_parent"
```

13 ou dip : il s'agit d'une unité qui est indépendante de la résolution de l'écran.

```
  android:layout_height="fill_parent" >

  <TextView
    android:id="@+id/text"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_centerHorizontal="true"
    android:layout_centerVertical="true"
    android:padding="25.7dp"
    android:text="@string/hello_world"
    tools:context=".MainActivity" />

</RelativeLayout>
```

Le fichier R sera modifié automatiquement lors de la sauvegarde :

```
public final class R {
  public static final class attr {
  }
  public static final class dimen {
    public static final int padding_large=0x7f040002;
    public static final int padding_medium=0x7f040001;
    public static final int padding_small=0x7f040000;
  }
  public static final class drawable {
    public static final int ic_action_search=0x7f020000;
    public static final int ic_launcher=0x7f020001;
  }
  public static final class id {
    public static final int menu_settings=0x7f080000;
  }
  public static final class layout {
    public static final int activity_main=0x7f030000;
  }
  public static final class menu {
    public static final int activity_main=0x7f070000;
  }
  public static final class string {
    public static final int app_name=0x7f050000;
    public static final int hello_world=0x7f050001;
    public static final int menu_settings=0x7f050002;
    public static final int title_activity_main=0x7f050003;
  }
  public static final class style {
    public static final int AppTheme=0x7f060000;
  }
}
```

Enfin, on peut utiliser cet identifiant dans le code, comme avec les autres identifiants. Pour cela, on utilise la méthode public View findViewById (int id). Attention, cette méthode renvoie une View, il faut donc la convertir[14] dans le type de destination.

Ensuite, on pourra déclarer que l'activité utilise comme interface graphique la vue que l'on désire à l'aide de la méthode void setContentView (View view). Dans l'exemple suivant, l'interface graphique est référencée par R.layout.activity_main, il s'agit donc du layout d'identifiant main, autrement dit celui que nous avons manipulé un peu plus tôt :

14 Technique du casting

```
import android.app.Activity;
import android.os.Bundle;
import android.widget.TextView;

public class TroimsActivity extends Activity {
  TextView monTexte = null;

  @Override
  public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);

    monTexte = (TextView)findViewById(R.id.text);
    monTexte.setText("Le texte de notre TextView");
  }
}
```

Puis on peut modifier le padding à posteriori :

```
import android.app.Activity;
import android.os.Bundle;
import android.widget.TextView;

public class TroimsActivity extends Activity {
  TextView monTexte = null;

  @Override
  public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);

    monTexte = (TextView)findViewById(R.id.text);
    // N'oubliez pas que cette fonction n'utilise que des entiers
    monTexte.setPadding(50, 60, 70, 90);
  }
}
```

À chaque fois que l'on récupère un objet depuis un fichier XML dans le code Java, on procède à une opération qui s'appelle la désérialisation. Concrètement, la désérialisation, c'est transformer un objet qui n'est pas décrit en Java − dans ce cas l'objet est décrit en XML − en un objet Java réel et concret. C'est à cela que sert la fonction View findViewById (int id). Le problème est que cette méthode va aller chercher dans un arbre de vues, qui est créé automatiquement par l'activité. Or, cet arbre ne sera créé qu'après le setContentView ! Donc le findViewById retournera null puisque l'arbre n'existera pas et l'objet ne sera donc pas dans l'arbre. On va à la place utiliser la méthode static View inflate (Context context, int id, ViewGroup parent). Cette méthode va désérialiser l'arbre XML au lieu de l'arbre de vues qui sera créé par l'activité.

```
import android.app.Activity;
import android.os.Bundle;
import android.widget.RelativeLayout;
import android.widget.TextView;

public class TroimsActivity extends Activity {
  RelativeLayout layout = null;
  TextView text = null;

  @Override
```

```
  public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);

    // On récupère notre layout par désérialisation. La méthode inflate retourne
un View
    // C'est pourquoi on caste (on convertit) le retour de la méthode avec le
vrai type de notre layout, c'est-à-dire RelativeLayout
    layout = (RelativeLayout) RelativeLayout.inflate(this,
R.layout.activity_main, null);
    // … puis on récupère TextView grâce à son identifiant
    text = (TextView) layout.findViewById(R.id.text);
    text.setText("Et cette fois, ça fonctionne !");
    setContentView(layout);
    // On aurait très bien pu utiliser « setContentView(R.layout.activity_main)
» bien sûr !
  }
}
```

2.2. Les widgets de base

2.2.1. Les widgets

Un widget est un élément de base qui permet d'afficher du contenu à l'utilisateur ou lui permet d'interagir avec l'application. Chaque widget possède un nombre important d'attributs XML et de méthodes Java.

TextView : permet d'afficher une chaîne de caractères que l'utilisateur ne peut modifier.

XML
```
<TextView
  android:layout_width="fill_parent"
  android:layout_height="wrap_content"
  android:text="@string/textView"
  android:textSize="8sp"
  android:textColor="#112233" />
```

La syntaxe @string/textView signifie qu'on utilise une ressource de type string. On précise la taille des caractères avec android:textSize, et la couleur du texte avec android:textColor (la notation avec un # permet de décrire des couleurs à l'aide de nombres hexadécimaux).

Java
```
TextView textView = new TextView(this);
textView.setText(R.string.textView);
textView.setTextSize(8);
textView.setTextColor(0x112233);
```

EditText : Ce composant est utilisé pour permettre à l'utilisateur d'écrire des textes. Il hérite de TextView, ce qui signifie qu'il peut prendre les mêmes attributs.

XML
```
<EditText
  android:layout_width="fill_parent"
  android:layout_height="wrap_content"
  android:hint="@string/editText"
  android:inputType="textMultiLine"
  android:lines="5" />
```

- Au lieu d'utiliser android:text, on utilise android:hint car android:hint affiche juste un texte d'indication, qui n'est pas pris en compte par l'EditText en tant que valeur.
- On précise quel type de texte contiendra notre EditText avec android:inputType. Dans ce cas précis un texte sur plusieurs lignes. Cet attribut change la nature du clavier qui est

proposé à l'utilisateur, par exemple si vous indiquez que l'EditText servira à écrire une adresse e-mail, alors l'arobase sera proposé tout de suite à l'utilisateur sur le clavier

- Enfin, on peut préciser la taille en lignes que doit occuper l'EditText avec android:lines.

Java
```
EditText editText = new EditText(this);
editText.setHint(R.string.editText);
editText.setInputType(InputType.TYPE_TEXT_FLAG_MULTI_LINE);
editText.setLines(5);
```

Button : simple bouton, même s'il s'agit en fait d'un TextView qui hérite de TextView.

XML
```
<Button
  android:layout_width="fill_parent"
  android:layout_height="wrap_content"
  android:text="@string/button" />
```
Java
```
Button button = new Button(this);
editText.setText(R.string.button);
```

CheckBox : case qui peut être dans deux états : cochée ou pas. Elle hérite de Button.

XML
```
<CheckBox
  android:layout_width="fill_parent"
  android:layout_height="wrap_content"
  android:text="@string/checkBox"
  android:checked="true" />
```

android:checked="true" signifie que la case est cochée par défaut.

Java
```
CheckBox checkBox = new CheckBox(this);
checkBox.setText(R.string.checkBox);
checkBox.setChecked(true)
if(checkBox.isChecked())
  // Faire quelque chose si le bouton est coché
```

RadioButton et **RadioGroup** : Même principe que la CheckBox, à la différence que l'utilisateur ne peut cocher qu'une seule case. Il est recommandé de les regrouper dans un RadioGroup. RadioButton hérite de Button.

XML
```
<RadioGroup
  android:layout_width="wrap_content"
  android:layout_height="wrap_content"
  android:orientation="horizontal" >
  <RadioButton
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:checked="true" />
  <RadioButton
    android:layout_width="wrap_content"
    android:layout_height="wrap_content" />
  <RadioButton
    android:layout_width="wrap_content"
    android:layout_height="wrap_content" />
</RadioGroup>
```
Java
```
RadioGroup radioGroup = new RadioGroup(this);
RadioButton radioButton1 = new RadioButton(this);
RadioButton radioButton2 = new RadioButton(this);
RadioButton radioButton3 = new RadioButton(this);

// On ajoute les boutons au RadioGroup
radioGroup.addView(radioButton1, 0);
radioGroup.addView(radioButton2, 1);
```

```
radioGroup.addView(radioButton3, 2);

// On sélectionne le premier bouton
radioGroup.check(0);

// On récupère l'identifiant du bouton qui est coché
int id = radioGroup.getCheckedRadioButtonId();
```

2.2.2. Gérer les évèvements

Il existe plusieurs façons d'interagir avec une interface graphique. Par exemple cliquer sur un bouton, entrer un texte, sélectionner une portion de texte, etc. Ces interactions s'appellent des évènements. Pour pouvoir réagir à l'apparition d'un évènement, il faut utiliser un objet qui va détecter l'évènement et afin de vous permettre le traiter. Ce type d'objet s'appelle un listener. Un listener est une interface qui vous oblige à redéfinir des méthodes de callback et chaque méthode sera appelée au moment où se produira l'évènement associé.

Par exemple, pour intercepter l'évènement clic sur un Button, on appliquera l'interface View.OnClickListener sur ce bouton. Cette interface contient la méthode de callbackvoid onClick(View vue) — le paramètre de type View étant la vue sur laquelle le clic a été effectué, qui sera appelée à chaque clic et qu'il faudra implémenter pour déterminer que faire en cas de clic. Par exemple pour gérer d'autres évènements, on utilisera d'autres méthodes (liste non exhaustive) :

- View.OnLongClickListener pour les clics qui durent longtemps, avec la méthode boolean onLongClick(View vue). Cette méthode doit retourner true une fois que l'action associée a été effectuée.
- View.OnKeyListener pour gérer l'appui sur une touche. On y associe la méthode boolean onKey(View vue, int code, KeyEvent event). Cette méthode doit retourner true une fois que l'action associée a été effectuée.

Il faut indiquer à Android quand vous souhaitez que l'évènement soit considéré comme traité, achevé. En effet, il est possible qu'un évènement continue à agir dans le temps. Un exemple simple est celui du toucher. Le toucher correspond au fait de toucher l'écran, pendant que vous touchez l'écran et avant même de lever le doigt pour le détacher de l'écran. Si vous levez ce doigt, le toucher s'arrête et un nouvel évènement est lancé : le clic, mais concentrons-nous sur le toucher. Quand vous touchez l'écran, un évènement de type onTouch est déclenché. Si vous retournez true au terme de cette méthode, ça veut dire que cet évènement toucher a été géré, et donc si l'utilisateur continue à bouger son doigt sur l'écran, Android considérera les mouvements sont de nouveaux évènements toucher et à nouveaux la méthode de callbackonTouch sera appelée pour chaque mouvement. En revanche, si vous retournez false, l'évènement ne sera pas considéré comme terminé et si l'utilisateur continue à bouger son doigt sur l'écran, Android ne considérera pas que ce sont de nouveaux évènements et la méthode onTouch ne sera plus appelée. Il faut donc réfléchir en fonction de la situation.

Enfin pour associer un listener à une vue, on utilisera une méthode du type setOn[Evenement]Listener(On[Evenenement]Listener listener) avec Evenement l'évènement concerné, par exemple pour détecter les clics sur un bouton on fera :

Bouton b = new Button(getContext());

b.setOnClickListener(notre_listener);

On va faire implémenter un listener à notre classe, ce qui veut dire que l'activité interceptera d'elle-

même les évènements. Il faut nécessairement implémenter toutes les méthodes de cette interface mais il n'est pas indispensable de gérer tous les évènements d'une interface.

Exemple :

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

// Notre activité détectera les touchers et les clics sur les vues qui se sont
inscrites
public class Main extends Activity implements View.OnTouchListener,
View.OnClickListener {
  private Button b = null;

  @Override
  public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);

    setContentView(R.layout.main);

    b = (Button) findViewById(R.id.boutton);
    b.setOnTouchListener(this);
    b.setOnClickListener(this);
  }

  @Override
  public boolean onTouch(View v, MotionEvent event) {
    /* Réagir au toucher */
    return true;
  }

  @Override
  public void onClick(View v) {
    /* Réagir au clic */
  }
}
```

Cependant, un problème se pose. À chaque fois qu'on appuiera sur un bouton, quel qu'il soit, on rentrera dans la même méthode, et on exécutera donc le même code… C'est pas très pratique, si nous avons un bouton pour rafraîchir un onglet dans une application de navigateur internet et un autre pour quitter un onglet, on aimerait bien que cliquer sur le bouton de rafraîchissement ne quitte pas l'onglet et vice-versa. Heureusement, la vue passée dans la méthode onClick(View) permet de différencier les boutons. En effet, il est possible de récupérer l'identifiant de la vue sur laquelle le clic a été effectué. Ainsi, nous pouvons réagir différemment en fonction de cet identifiant :

```
public void onClick(View v) {
  // On récupère l'identifiant de la vue, et en fonction de cet identifiant…
  switch(v.getId()) {

    // Si l'identifiant de la vue est celui du premier bouton
    case R.id.bouton1:
    /* Agir pour bouton 1 */
```

```
    break;

    // Si l'identifiant de la vue est celui du deuxième bouton
    case R.id.bouton2:
    /* Agir pour bouton 2 */
    break;

    /* etc. */
  }
}
```

L'inconvénient principal de la technique précédente est qu'elle peut très vite allonger les méthodes des listeners, ce qui fait qu'on s'y perd un peu s'il y a beaucoup d'éléments à gérer. C'est pourquoi il est préférable de passer par une classe anonyme dès qu'on a un nombre élevé d'éléments qui réagissent au même évènement.

Exemple d'une classe anonyme qui implémente View.OnClickListener() :

```
widget.setTouchListener(
  new View.OnTouchListener() {
  /**
   * Contenu de ma classe
   * Comme on implémente une interface, il y aura des méthodes à implémenter,
dans ce cas-ci
   * « public boolean onTouch(View v, MotionEvent event) »
  */
      }
  ); // Et on n'oublie pas le point-virgule à la fin !
```

Exemple :

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

public class AnonymousExampleActivity extends Activity {
  // On cherchera à détecter les touchers et les clics sur ce bouton
  private Button touchAndClick = null;
  // On voudra détecter uniquement les clics sur ce bouton
  private Button clickOnly = null;

  @Override
  public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);

    touchAndClick = (Button)findViewById(R.id.touchAndClick);
    clickOnly = (Button)findViewById(R.id.clickOnly);

    touchAndClick.setOnLongClickListener(new View.OnLongClickListener() {
      @Override
      public boolean onLongClick(View v) {
        // Réagir à un long clic
        return false;
      }
    });
```

```
    touchAndClick.setOnClickListener(new View.OnClickListener() {
      @Override
      public void onClick(View v) {
        // Réagir au clic
      }
    });

    clickOnly.setOnClickListener(new View.OnClickListener() {
      @Override
      public void onClick(View v) {
        // Réagir au clic
      }
    });
  }
}
```

La méthode ci-dessous est dérivée de la méthode précédente : on implémente des classes anonymes dans des attributs de façon à pouvoir les utiliser dans plusieurs éléments graphiques différents qui auront la même réaction pour le même évènement. C'est la méthode à privilégier dès que l'on a, par exemple, plusieurs boutons qui utilisent le même code :

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

public class Main extends Activity {
  private OnClickListener clickListenerBoutons = new View.OnClickListener() {
    @Override
    public void onClick(View v) {
      /* Réagir au clic pour les boutons 1 et 2*/
    }
  };

  private OnTouchListener touchListenerBouton1 = new View.OnTouchListener() {
    @Override
    public boolean onTouch(View v, MotionEvent event) {
      /* Réagir au toucher pour le bouton 1*/
      return onTouch(v, event);
    }
  };

  private OnTouchListener touchListenerBouton3 = new View.OnTouchListener() {
    @Override
    public boolean onTouch(View v, MotionEvent event) {
      /* Réagir au toucher pour le bouton 3*/
      return super.onTouch(v, event);
    }
  };

  Button b1 = null;
  Button b2 = null;
  Button b3 = null;

  @Override
  public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
```

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

    b1 = (Button) findViewById(R.id.bouton1);
    b2 = (Button) findViewById(R.id.bouton2);
    b3 = (Button) findViewById(R.id.bouton3);

    b1.setOnTouchListener(touchListenerBouton1);
    b1.setOnClickListener(clickListenerBoutons);
    b2.setOnClickListener(clickListenerBoutons);
    b3.setOnTouchListener(touchListenerBouton3);
  }
}
```

2.3. Organiser son interface avec des layouts

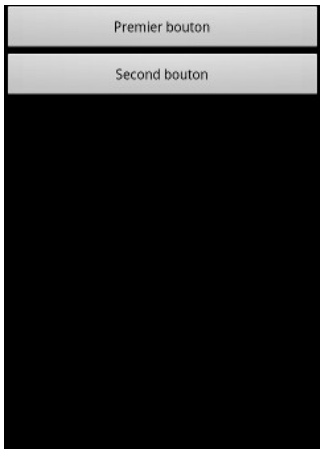
2.3.1. LinearLayout : placer les éléments sur une ligne

Ce layout se charge de mettre les vues sur une même ligne, selon une certaine orientation. L'attribut pour préciser cette orientation est android:orientation. On peut lui donner deux valeurs :

- vertical pour que les composants soient placés de haut en bas (en colonne) ;
- horizontal pour que les composants soient placés de gauche à droite (en ligne).

Exemple 1 :

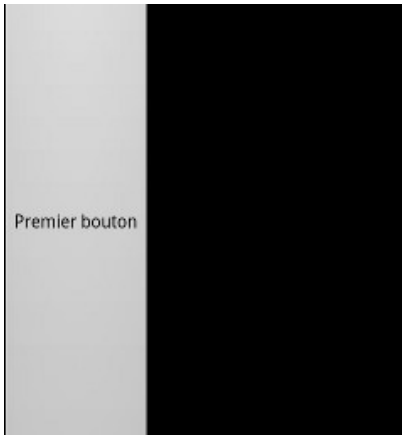
```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
  android:orientation="vertical"
  android:layout_width="fill_parent"
  android:layout_height="fill_parent" >
  <Button
    android:id="@+id/premier"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="Premier bouton" />

  <Button
    android:id="@+id/second"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="Second bouton" />
</LinearLayout>
```



- Le LinearLayout est vertical et prend toute la place de son parent.
- Le premier bouton prend toute la place dans le parent en largeur et uniquement la taille nécessaire en hauteur (la taille du texte, donc !).
- Le second bouton fait de même.

Exemple 2 :

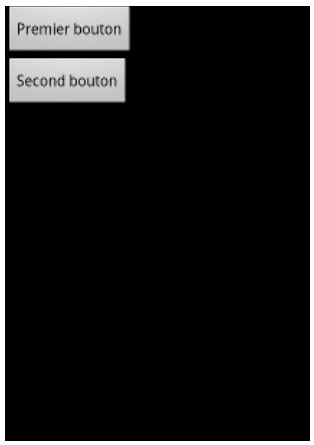
```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
  android:orientation="vertical"
  android:layout_width="fill_parent"
  android:layout_height="fill_parent" >

  <Button
    android:id="@+id/premier"
    android:layout_width="wrap_content"
    android:layout_height="fill_parent"
    android:text="Premier bouton" />

  <Button
    android:id="@+id/second"
    android:layout_width="wrap_content"
    android:layout_height="fill_parent"
    android:text="Second bouton" />
</LinearLayout>
```



- Le LinearLayout est vertical et prend toute la place de son parent.
- Le premier bouton prend toute la place de son parent en hauteur et uniquement la taille nécessaire en largeur.
- Comme le premier bouton prend toute la place, alors le pauvre second bouton se fait écraser. C'est pour cela qu'on ne le voit pas.

Exemple 3 :

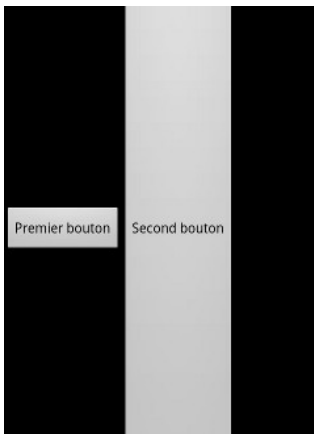
```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
  android:orientation="vertical"
  android:layout_width="wrap_content"
  android:layout_height="wrap_content" >
  <Button
    android:id="@+id/premier"
    android:layout_width="wrap_content"
    android:layout_height="fill_parent"
    android:text="Premier bouton" />
  <Button
    android:id="@+id/second"
    android:layout_width="wrap_content"
    android:layout_height="fill_parent"
    android:text="Second bouton" />
</LinearLayout>
```



- Le LinearLayout est vertical et prend toute la place en largeur mais uniquement la taille nécessaire en hauteur : dans ce cas précis, la taille nécessaire sera calculée en fonction de la taille des enfants.
- Le premier bouton prend toute la place possible dans le parent. Comme le parent prend le moins de place possible, il doit faire de même.
- Le second bouton fait de même.

Exemple 4 :

```
<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:id="@+id/premier"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Premier bouton" />
  <Button
    android:id="@+id/second"
    android:layout_width="wrap_content"
    android:layout_height="fill_parent"
    android:text="Second bouton" />
</LinearLayout>
```



- Le LinearLayout est horizontal et prend toute la place de son parent.
- Le premier bouton prend uniquement la place nécessaire.
- Le second bouton prend uniquement la place nécessaire en longueur et s'étend jusqu'aux bords du parent en hauteur.

Vous remarquerez que l'espace est toujours divisé entre les deux boutons, soit de manière égale, soit un bouton écrase complètement l'autre. Et si on voulait que le bouton de droite prenne deux fois plus de place que celui de gauche par exemple ?

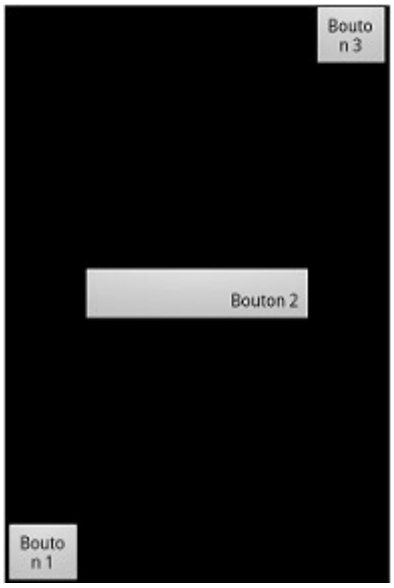
Pour cela, il faut attribuer un poids au composant. Ce poids peut être défini grâce à l'attribut android:layout_weight. Pour faire en sorte que le bouton de droite prenne deux fois plus de place, on peut lui mettre android:layout_weight="1" et mettre au bouton de gauche android:layout_weight="2". C'est alors le composant qui a la plus faible pondération qui a la priorité.

Et si, dans l'exemple précédent où un bouton en écrasait un autre, les deux boutons avaient eu un poids identique, par exemple android:layout_weight="1" pour les deux, ils auraient eu la même priorité et auraient pris la même place. Par défaut, ce poids est à 0.

Exemple 5 :

Dernier attribut particulier pour les widgets de ce layout, android:layout_gravity, qu'il ne faut pas confondre avec android:gravity. android:layout_gravity vous permet de déterminer comment se placera la vue dans le parent, alors que android:gravity vous permet de déterminer comment se placera le contenu de la vue à l'intérieur même de la vue (par exemple, comment se placera le texte dans un TextView ? Au centre, en haut, à gauche ?).

```
<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:id="@+id/bouton1"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:layout_gravity="bottom"
    android:layout_weight="40"
    android:text="Bouton 1" />
  <Button
    android:id="@+id/bouton2"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:layout_gravity="center"
    android:layout_weight="20"
    android:gravity="bottom|right"
    android:text="Bouton 2" />
  <Button
    android:id="@+id/bouton3"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:layout_gravity="top"
    android:layout_weight="40"
    android:text="Bouton 3" />
</LinearLayout>
```



Comme le bouton 2 a un poids deux fois inférieur aux boutons 1 et 3, alors il prend deux fois plus de place qu'eux. De plus, chaque bouton possède un attribut android:layout_gravity afin de que l'on détermine sa position dans le layout. Le deuxième bouton présente aussi l'attribut android:gravity, qui est un attribut de TextView et non layout, de façon à mettre le texte en bas (bottom) à droite (right).

2.3.2. RelativeLayout : placer les éléments les uns en fonction des autres

Ce layout propose plutôt de placer les composants les uns par rapport aux autres. Il est même possible de les placer par rapport au RelativeLayout parent.

Si on veut par exemple placer une vue au centre d'un RelativeLayout, on peut passer à cette vue l'attribut android:layout_centerInParent="true". Il est aussi possible d'utiliser android:layout_centerHorizontal="true" pour centrer, mais uniquement sur l'axe horizontal, de même avec android:layout_centerVertical="true" pour centrer sur l'axe vertical.

Exemple 1 :

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

  <TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Centré dans le parent"
    android:layout_centerInParent="true" />
  <TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Centré verticalement"
    android:layout_centerVertical="true" />
  <TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Centré horizontalement"
    android:layout_centerHorizontal="true" />

</RelativeLayout>
```

On observe ici une différence majeure avec le LinearLayout : il est possible d'empiler les vues. Ainsi, le TextView centré verticalement s'entremêle avec celui centré verticalement et horizontalement.

Il existe d'autres contrôles pour situer une vue par rapport à un RelativeLayout. On peut utiliser :

- android:layout_alignParentBottom="true" pour aligner le plancher d'une vue au plancher du RelativeLayout ;
- android:layout_alignParentTop="true" pour coller le plafond d'une vue au plafond du RelativeLayout ;
- android:layout_alignParentLeft="true" pour coller le bord gauche d'une vue avec le bord gauche du RelativeLayout ;
- android:layout_alignParentRight="true" pour coller le bord droit d'une vue avec le bord droit du RelativeLayout.

Exemple 2 :

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
  android:layout_width="fill_parent"
  android:layout_height="fill_parent" >
  <TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="En haut !"
    android:layout_alignParentTop="true" />
  <TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="En bas !"
    android:layout_alignParentBottom="true" />
  <TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="A gauche !"
    android:layout_alignParentLeft="true" />
  <TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="A droite !"
    android:layout_alignParentRight="true" />
  <TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Ces soirées là !"
    android:layout_centerInParent="true" />
</RelativeLayout>
```

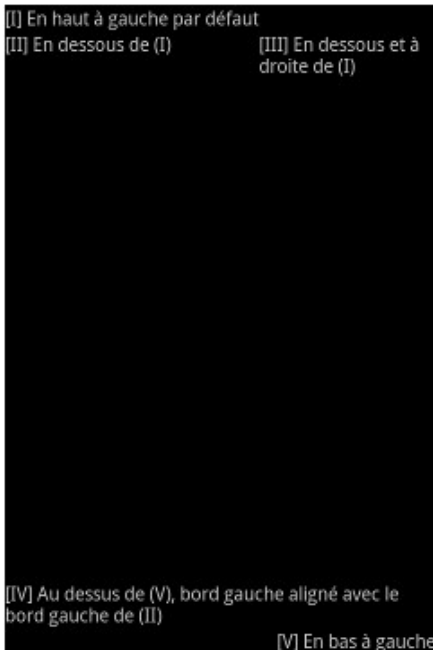
On remarque tout de suite que les TextView censés se situer à gauche et en haut s'entremêlent, mais c'est logique puisque par défaut une vue se place en haut à gauche dans un RelativeLayout. Donc, quand on lui dit « Place-toi à gauche » ou « Place-toi en haut », c'est comme si on ne lui donnait pas d'instructions au final.

Enfin, il ne faut pas oublier que le principal intérêt de ce layout est de pouvoir placer les éléments les uns par rapport aux autres. Pour cela il existe deux catégories d'attributs :

- Ceux qui permettent de positionner deux bords opposés de deux vues différentes ensemble. On y trouve android:layout_below (pour aligner le plafond d'une vue sous le plancher d'une autre), android:layout_above (pour aligner le plancher d'une vue sur le plafond d'une autre), android:layout_toRightOf (pour aligner le bord gauche d'une vue au bord droit d'une autre) et android:layout_toLeftOf (pour aligner le bord droit d'une vue au bord gauche d'une autre).
- Ceux qui permettent de coller deux bords similaires ensemble. On trouve android:layout_alignBottom (pour aligner le plancher de la vue avec le plancher d'une autre), android:layout_alignTop (pour aligner le plafond de la vue avec le plafond d'une autre), android:layout_alignLeft (pour aligner le bord gauche d'une vue avec le bord gauche d'une autre) et android:layout_alignRight (pour aligner le bord droit de la vue avec le bord droit d'une autre).

Exemple 3 :

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
  android:layout_width="fill_parent"
  android:layout_height="fill_parent" >
  <TextView
    android:id="@+id/premier"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="[I] En haut à gauche par défaut" />
  <TextView
    android:id="@+id/deuxieme"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="[II] En dessous de (I)"
    android:layout_below="@id/premier" />
  <TextView
    android:id="@+id/troisieme"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="[III] En dessous et à droite de (I)"
    android:layout_below="@id/premier"
    android:layout_toRightOf="@id/premier" />
  <TextView
    android:id="@+id/quatrieme"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="[IV] Au dessus de (V), bord gauche aligné avec le bord gauche
de (II)"
    android:layout_above="@+id/cinquieme"
    android:layout_alignLeft ="@id/deuxieme" />
  <TextView
    android:id="@+id/cinquieme"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="[V] En bas à gauche"
    android:layout_alignParentBottom="true"
    android:layout_alignParentRight="true" />
</RelativeLayout>
```



Remarque : on ne fait pas référence au dernier TextView comme aux autres, il faut préciser un + dans son attribut android:layout_above. En effet, pour faire référence à une vue qui n'est définie que plus tard dans le fichier XML, il faut ajouter un + dans l'identifiant, sinon Android pensera qu'il s'agit d'une faute et non d'un identifiant qui sera déclaré après.

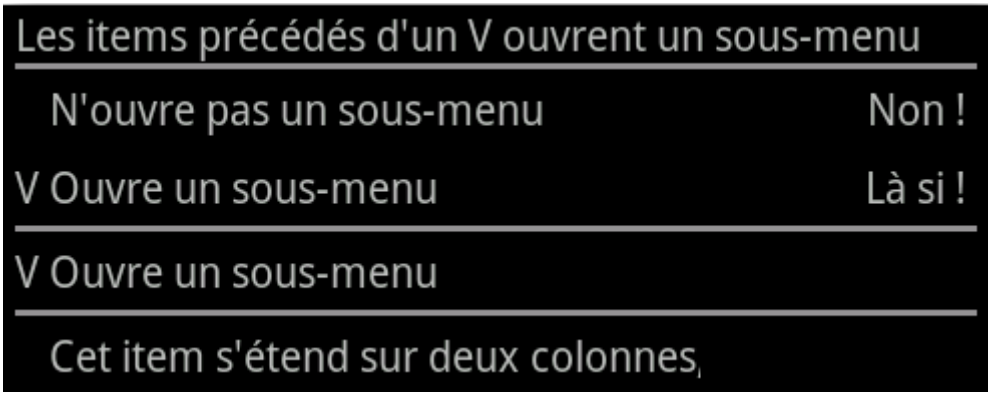
2.2.3. TableLayout : placer les éléments comme dans un tableau

Ce layout permet d'organiser les éléments en tableau, comme en HTML, mais sans les bordures.

Exemple :

```
<?xml version="1.0" encoding="utf-8"?>
<TableLayout xmlns:android="http://schemas.android.com/apk/res/android"
  android:layout_width="fill_parent"
  android:layout_height="fill_parent"
  android:stretchColumns="1">
  <TextView
    android:text="Les items précédés d'un V ouvrent un sous-menu"
```

```
  />
  <View
    android:layout_height="2dip"
    android:background="#FF909090"
  />
  <TableRow>
    <TextView
      android:text="N'ouvre pas un sous-menu"
      android:layout_column="1"
      android:padding="3dip"
    />
    <TextView
      android:text="Non !"
      android:gravity="right"
      android:padding="3dip"
    />
  </TableRow>
  <TableRow>
    <TextView
      android:text="V"
    />
    <TextView
      android:text="Ouvre un sous-menu"
      android:layout_column="1"
      android:padding="3dip"
    />
    <TextView
      android:text="Là si !"
      android:gravity="right"
      android:padding="3dip"
    />
  </TableRow>
  <View
    android:layout_height="2dip"
    android:background="#FF909090"
  />
  <TableRow>
    <TextView
      android:text="V"
    />
    <TextView
      android:text="Ouvre un sous-menu"
      android:padding="3dip"
    />
  </TableRow>
  <View
    android:layout_height="2dip"
    android:background="#FF909090"
  />
  <TableRow>
    <TextView
      android:layout_column="1"
      android:layout_span="2"
      android:text="Cet item s'étend sur deux colonnes"
      android:padding="3dip"
    />
  </TableRow>
</TableLayout>
```



On observe tout d'abord qu'il est possible de mettre des vues directement dans le tableau, auquel cas elles prendront toute la place possible en longueur. En fait, elles s'étendront sur toutes les colonnes du tableau. Cependant, si on veut un contrôle plus complet ou avoir plusieurs éléments sur une même ligne, alors il faut passer par un objet <TableRow>.

```
<TextView
  android:text="Les items précédés d'un V ouvrent un sous-menu" />
```

Cet élément s'étend sur toute la ligne puisqu'il ne se trouve pas dans un <TableRow>

```
<View
  android:layout_height="2dip"
  android:background="#FF909090" />
```

Moyen efficace pour dessiner un séparateur — n'essayez pas de le faire en dehors d'un <TableLayout> ou votre application plantera.

Une ligne est composée de cellules. Chaque cellule peut contenir une vue, ou être vide. La taille du tableau en colonnes est celle de la ligne qui contient le plus de cellules. Dans notre exemple, nous avons trois colonnes pour tout le tableau, puisque la ligne avec le plus cellules est celle qui contient « V » et se termine par « Là si ! ».

```
<TableRow>
  <TextView
    android:text="V"
  />
  <TextView
    android:text="Ouvre un sous-menu"
    android:layout_column="1"
    android:padding="3dip"
  />
  <TextView
    android:text="Là si !"
    android:gravity="right"
    android:padding="3dip"
  />
</TableRow>
```

Cette ligne a trois éléments, c'est la plus longue du tableau, ce dernier est donc constitué de trois colonnes.

Il est possible de choisir dans quelle colonne se situe un item avec l'attribut android:layout_column. Attention, l'index des colonnes commence à 0. Dans notre exemple, le dernier item se place directement à la deuxième colonne grâce à android:layout_column="1".

```
<TableRow>
```

```
  <TextView
    android:text="N'ouvre pas un sous-menu"
    android:layout_column="1"
    android:padding="3dip"
  />
  <TextView
    android:text="Non !"
    android:gravity="right"
    android:padding="3dip"
  />
</TableRow>
```

On veut laisser vide l'espace pour la première colonne, on place alors les deux TextView dans les colonnes 1 et 2.

La taille d'une cellule dépend de la cellule la plus large sur une même colonne. Dans notre exemple, la seconde colonne fait la largeur de la cellule qui contient le texte « N'ouvre pas un sous-menu », puisqu'il se trouve dans la deuxième colonne et qu'il n'y a pas d'autres éléments dans cette colonne qui soit plus grand.

Enfin, il est possible d'étendre un item sur plusieurs colonnes à l'aide de l'attribut android:layout_span. Dans notre exemple, le dernier item s'étend de la deuxième colonne à la troisième. Il est possible de faire de même sur les lignes avec l'attribut android:layout_column.

```
<TableRow>
  <TextView
    android:layout_column="1"
    android:layout_span="2"
    android:text="Cet item s'étend sur deux colonnes"
    android:padding="3dip"
  />
</TableRow>
```

Ce TextView débute à la deuxième colonne et s'étend sur deux colonnes, donc jusqu'à la troisième.

Sur le nœud TableLayout, on peut jouer avec trois attributs (attention, les rangs débutent à 0) :

- android:stretchColumns pour que la longueur de tous les éléments de cette colonne passe en fill_parent, donc pour prendre le plus de place possible. Il faut préciser le rang de la colonne à cibler, ou plusieurs rangs séparés par des virgules.
- android:shrinkColumns pour que la longueur de tous les éléments de cette colonne passe en wrap_content, donc pour prendre le moins de place possible. Il faut préciser le rang de la colonne à cibler, ou plusieurs rangs séparés par des virgules.
- android:collapseColumns pour faire purement et simplement disparaître des colonnes du tableau. Il faut préciser le rang de la colonne à cibler, ou plusieurs rangs séparés par des virgules.

3. Les capteurs

La majorité des appareils modernes sont bien plus que de simples outils pour communiquer ou naviguer sur internet. Ils ont des capacités sensorielles, matérialisées par leurs capteurs. Ces capteurs nous fournissent des informations brutes avec une grande précision, qu'il est possible d'interpréter pour comprendre les transitions d'état que vit le terminal. On trouve par exemple des accéléromètres, des gyroscopes, des capteurs de champ magnétique, etc. Tous ces capteurs nous

permettent d'explorer de nouvelles voies, d'offrir de nouvelles possibilités aux utilisateurs.

3.1. Les différents capteurs

On peut répartir les capteurs en trois catégories :

- Les capteurs de mouvements : en mesurant les forces d'accélération et de rotation sur les trois axes, ces capteurs sont capables de déterminer dans quelle direction se dirige l'appareil. On y trouve l'accéléromètre, les capteurs de gravité, les gyroscopes et les capteurs de vecteurs de rotation.
- Les capteurs de position : évidemment, ils déterminent la position de l'appareil. On trouve ainsi les capteurs d'orientation et le magnétomètre.
- Les capteurs environnementaux : ce sont trois capteurs (baromètre, photomètre et thermomètre) qui mesurent la pression atmosphérique, l'illumination et la température ambiante.

D'un point de vue technique, on trouve deux types de capteurs. Certains sont des composants matériels, c'est-à-dire qu'il y a un composant physique présent sur le terminal. Ils fournissent des données en prenant des mesures. Certains autres capteurs sont uniquement présents d'une manière logicielle. Ils se basent sur des données fournies par des capteurs physiques pour calculer des données nouvelles.

Il n'est pas rare qu'un terminal n'ait pas tous les capteurs, mais seulement une sélection. Par exemple, la grande majorité des appareils ont un accéléromètre ou un magnétomètre, mais peu ont un thermomètre. De plus, il arrive qu'un terminal ait plusieurs exemplaires d'un capteur, mais calibrés d'une manière différente de façon à avoir des résultats différents.

Ces différents capteurs sont représentés par une valeur dans la classe Sensor. On trouve ainsi :

Nom du capteur	**Valeur système**	**Type**	**Description**	**Utilisation typique**
Accéléromètre	TYPE_ACCELEROMETER	Matériel	Mesure la force d'accélération appliquée au terminal sur les trois axes (x, y et z), donc la force de gravitation (m/s^2).	Détecter les mouvements.
Tous les capteurs	TYPE_ALL	Matériel et logiciel	Représente tous les capteurs qui existent.	
Gyroscope	TYPE_GYROSCOPE	Matériel	Mesure le taux de rotation sur chacun des trois axes en radian par seconde (rad/s).	Détecter l'orientation de l'appareil.
Photomètre	TYPE_LIGHT	Matériel	Mesure le niveau de lumière ambiante en lux (lx).	Détecter la luminosité pour adapter celle de l'écran de l'appareil.
Magnétomètre	TYPE_MAGNETIC_FIELD	Matériel	Mesure le champ géomagnétique sur les trois axes en microtesla (µT).	Créer un compas.
Orientation	TYPE_ORIENTATIO	Logiciel	Mesure le degré de rotation que	Déterminer la

	N		l'appareil effectue sur les trois axes.	position de l'appareil.
Baromètre	TYPE_PRESSURE	Matériel	Mesure la pression ambiante en hectopascal (hPa) ou millibar (mbar).	Surveiller les changements de pression de l'air ambiant.
Capteur de proximité	TYPE_PROXIMITY	Matériel	Mesure la proximité d'un objet en centimètres (cm).	Détecter si l'utilisateur porte le téléphone à son oreille pendant un appel.
Thermomètre	TYPE_TEMPERATURE	Matériel	Mesure la température de l'appareil en degrés Celsius (°C).	Surveiller la température.

3.2. Opérations génériques

3.2.1. Demander la présence d'un capteur

Il se peut que votre application n'ait aucun sens sans un certain capteur. Pour indiquer qu'on ne veut pas qu'un utilisateur sans accéléromètre puisse télécharger votre application, il vous faudra ajouter une ligne de type <uses-feature> dans votre Manifest :

```
<uses-feature android:name="android.hardware.sensor.accelerometer"
  android:required="true" />
```

android:required="true" sert à préciser que la présence de l'accéléromètre est absolument indispensable. S'il est possible d'utiliser l'application sans l'accéléromètre mais qu'il est fortement recommandé d'en posséder un, alors il suffit de mettre à la place android:required="false".

3.2.2. Identifier les capteurs

La classe qui permet d'accéder aux capteurs est SensorManager. Pour en obtenir une instance, il suffit de faire :

SensorManager sensorManager =
(SensorManager)getSystemService(Context.SENSOR_SERVICE);

Les capteurs sont représentés par la classe Sensor. Si vous voulez connaître la liste de tous les capteurs existants sur l'appareil, il vous faudra utiliser la méthode List<Sensor> getSensorList(int type) avec type qui vaut Sensor.TYPE_ALL. De même, pour connaître tous les capteurs qui correspondent à une catégorie de capteurs, utilisez l'une des valeurs vues précédemment dans cette même méthode. Par exemple, pour connaître la liste de tous les magnétomètres :

ArrayList<Sensor> liste = sensorManager.getSensorList(Sensor.TYPE_MAGNETIC_FIELD);

Il est aussi possible d'obtenir une instance d'un capteur. Il suffit d'utiliser la méthode Sensor getDefaultSensor(int type) avec type un identifiant présenté dans le tableau précédent. Comme je vous l'ai déjà dit, il peut y avoir plusieurs capteurs qui ont le même objectif dans un appareil, c'est pourquoi cette méthode ne donnera que l'appareil par défaut, celui qui correspondra aux besoins les plus génériques.

Si le capteur demandé n'existe pas dans l'appareil, la méthode getDefaultSensor renverra null.

```
Sensor accelerometre =
sensorManager.getDefaultSensor(Sensor.TYPE_ACCELETOMETER);
if(accelerometre != null)
  // Il y a au moins un accéléromètre
else
  // Il n'y en a pas
```

Il est ensuite possible de récupérer des informations sur le capteur, comme par exemple sa consommation électrique avec float getPower() et sa portée avec float getMaximumRange().

3.2.3. Détection des changements des capteurs

L'interface SensorEventListener permet de détecter deux types de changement dans les capteurs :

- Un changement de précision du capteur avec la méthode de callbackvoid onAccuracyChanged(Sensor sensor, int accuracy) avec sensor le capteur dont la précision a changé et accuracy la nouvelle précision. accuracy peut valoir SensorManager.SENSOR_STATUS_ACCURACY_LOW pour une faible précision, SensorManager.SENSOR_STATUS_ACCURACY_MEDIUM pour une précision moyenne, SensorManager.SENSOR_STATUS_ACCURACY_HIGH pour une précision maximale et SensorManager.SENSOR_STATUS_ACCURACY_UNRELIABLE s'il ne faut pas faire confiance à ce capteur.
- Le capteur a calculé une nouvelle valeur, auquel cas se lancera la méthode de callbackvoid onSensorChanged(SensorEvent event). Un SensorEvent indique à chaque fois quatre informations contenues dans quatre attributs : l'attribut accuracy indique la précision de cette mesure (il peut avoir les mêmes valeurs que précédemment), l'attribut sensor contient une référence au capteur qui a fait la mesure, l'attribut timestamp est l'instant en nanosecondes où la valeur a été prise, et enfin les valeurs sont contenues dans l'attribut values.

values est un tableau d'entiers. Si dans le tableau précédent j'ai dit que les valeurs correspondaient aux trois axes, alors le tableau a trois valeurs : values[0] est la valeur sur l'axe x, values[1] la valeur sur l'axe y et values[2] la valeur sur l'axe z. Si le calcul ne se fait pas sur trois axes, alors il n'y aura que values[0] qui contiendra la valeur. Attention, cette méthode sera appelée très souvent, il est donc de votre devoir de ne pas effectuer d'opérations bloquantes à l'intérieur. Si vous effectuez des opérations longues à résoudre, alors il se peut que la méthode soit à nouveau lancée alors que l'ancienne exécution n'avait pas fini ses calculs, ce qui va encombrer le processeur au fur et à mesure.

```
final SensorEventListener mSensorEventListener = new SensorEventListener() {
  public void onAccuracyChanged(Sensor sensor, int accuracy) {
    // Que faire en cas de changement de précision ?
  }
  public void onSensorChanged(SensorEvent sensorEvent) {
    // Que faire en cas d'évènements sur le capteur ?
  }
};
```

Une fois l'interface écrite, il faut déclarer au capteur que nous sommes à son écoute. Pour cela, on va utiliser la méthode boolean registerListener(SensorEventListener listener, Sensor sensor, int rate) de SensorManager, avec le listener, le capteur dans sensor et la fréquence de mise à jour dans rate. Il est possible de donner à rate les valeurs suivantes, de la fréquence la moins élevée à la plus élevée :

- SensorManager.SENSOR_DELAY_NORMAL (0,2 seconde entre chaque prise) ;

- SensorManager.SENSOR_DELAY_UI (0,06 seconde entre chaque mise à jour, délai assez lent qui convient aux interfaces graphiques) ;
- SensorManager.SENSOR_DELAY_GAME (0,02 seconde entre chaque prise, convient aux jeux) ;
- SensorManager.SENSOR_DELAY_FASTEST (0 seconde entre les prises).

Le délai que vous indiquez n'est qu'une indication, il ne s'agit pas d'un délai très précis. Il se peut que la prise se fasse avant ou après le moment choisi. De manière générale, la meilleure pratique est d'avoir la valeur la plus lente possible, puisque c'est elle qui permet d'économiser le plus le processeur et donc la batterie.

Enfin, on peut désactiver l'écoute d'un capteur avec void unregisterListener(SensorEventListener listener, Sensor sensor). Il ne faut pas oublier de désactiver vos capteurs pendant que l'activité n'est pas au premier plan (donc il faut le désactiver pendant onPause() et le réactiver pendant onResume()), car le système ne le fera pas pour vous. De manière générale, désactivez les capteurs dès que vous ne les utilisez plus.

```
private SensorManager mSensorManager = null;
private Sensor mAccelerometer = null;
final SensorEventListener mSensorEventListener = new SensorEventListener() {
  public void onAccuracyChanged(Sensor sensor, int accuracy) {
    // Que faire en cas de changement de précision ?
  }
  public void onSensorChanged(SensorEvent sensorEvent) {
    // Que faire en cas d'évènements sur le capteur ?
  }
};
@Override
public final void onCreate(Bundle savedInstanceState) {
  super.onCreate(savedInstanceState);
  setContentView(R.layout.activity_main);
  mSensorManager = (SensorManager) getSystemService(Context.SENSOR_SERVICE);
  mAccelerometer = mSensorManager.getDefaultSensor(Sensor.TYPE_ACCELEROMETER);
}
@Override
protected void onResume() {
  super.onResume();
  mSensorManager.registerListener(mSensorEventListener, mAccelerometer,
SensorManager.SENSOR_DELAY_NORMAL);
}
@Override
protected void onPause() {
  super.onPause();
  mSensorManager.unregisterListener(mSensorEventListener, mAccelerometer);
}
```

2.3.4. Les capteurs de mouvements

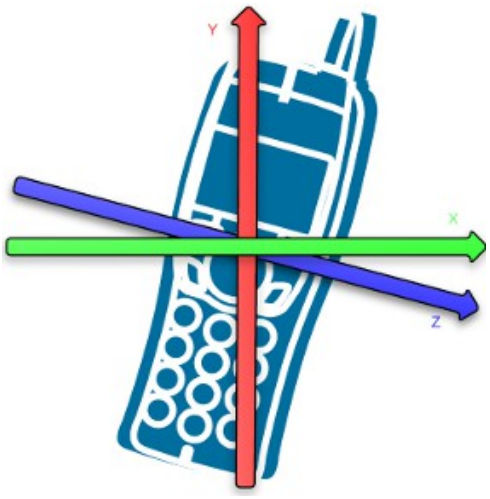
Pour l'API 7, on trouve surtout l'accéléromètre, mais les versions suivantes supportent aussi le gyroscope, ainsi que trois capteurs logiciels (gravitationnel, d'accélération linéaire et de vecteurs de rotation). De nos jours, on trouve presque tout le temps un accéléromètre et un gyroscope. Pour les capteurs logiciels, c'est plus complexe puisqu'ils se basent souvent sur plusieurs capteurs pour déduire et calculer des données, ils nécessitent donc la présence de plusieurs capteurs différents.

On utilise les capteurs de mouvements pour détecter les… mouvements : inclinaisons, aux

secousses, aux rotations ou aux balancements. Typiquement, les capteurs de mouvements ne sont pas utilisés pour détecter la position de l'utilisateur, mais si on les utilise conjointement avec d'autres capteurs, comme par exemple le magnétomètre, ils permettent de mieux évaluer ses déplacements.

Tous ces capteurs retournent un tableau de float de taille 3, chaque élément correspondant à un axe différent :

- La première valeur, values[0], se trouve sur l'axe x, il s'agit de l'axe de l'horizon, quand vous bougez votre téléphone de gauche à droite. Ainsi, la valeur est positive et augmente quand vous déplacez le téléphone vers la droite, alors qu'elle est négative et continue à diminuer plus vous le déplacez vers la gauche.
- La deuxième valeur, values[1], correspond à l'axe y, c'est-à-dire l'axe vertical, quand vous déplacez votre téléphone de haut en bas. La valeur est positive quand vous déplacez le téléphone vers le haut et négative quand vous le déplacez vers le bas.
- Enfin, la troisième valeur, values[2], correspond à l'axe z, il s'agit de l'axe sur lequel vous pouvez éloigner ou rapprocher le téléphone de vous. Quand vous le rapprochez de vous, la valeur est positive et, quand vous l'éloignez, la valeur est négative.



Toutes ces valeurs respectent un schéma identique : un 0 signifie pas de mouvement, une valeur positive un déplacement dans le sens de l'axe et une valeur négative un déplacement dans le sens inverse de celui de l'axe.

Enfin, l'accéléromètre ne mesure pas du tout la vitesse, juste le changement de vitesse. Pour obtenir la vitesse depuis les données de l'accéléromètre, il faudra intégrer l'accélération sur le temps (que l'on peut obtenir avec l'attribut timestamp) pour obtenir la vitesse. Et pour obtenir une distance, il faudra intégrer la vitesse sur le temps.

2.3.5. Les capteurs de position

On trouve trois capteurs qui permettent de déterminer la position du terminal : le magnétomètre, le capteur d'orientation et le capteur de proximité (c'est le moins puissant, il est uniquement utilisé pour détecter quand l'utilisateur a le visage collé au téléphone, afin d'afficher le menu uniquement quand l'utilisateur n'a pas le téléphone contre la joue). Le magnétomètre et le capteur de proximité sont matériels, alors que le capteur d'orientation est une combinaison logicielle de l'accéléromètre et du magnétomètre.

Le capteur d'orientation et le magnétomètre renvoient un tableau de taille 3, alors que pour le capteur de proximité c'est plus compliqué. Parfois il renvoie une valeur en centimètres, parfois juste une valeur qui veut dire « proche » et une autre qui veut dire « loin » ; dans ces cas-là, un objet est considéré comme éloigné s'il se trouve à plus de 5 cm.

Cependant, le magnétomètre n'est pas utilisé que pour déterminer la position de l'appareil. Si on l'utilise conjointement avec l'accéléromètre, il est possible de détecter l'inclinaison de l'appareil. Et pour cela, la seule chose dont nous avons besoin, c'est de faire de gros calculs trigonométriques.

Dans tous les cas, il faut utiliser deux capteurs et donc déclarer les deux dans deux listeners différents. Une fois les données récupérées, il est possible de calculer ce qu'on appelle la méthode Rotation avec la méthode statique static boolean SensorManager.getRotationMatrix(float[] R, float[] I, float[] gravity, float[] geomagnetic) avec R le tableau de taille 9 dans lequel seront stockés les résultats, I un tableau d'inclinaison qui peut bien valoir null, gravity les données de l'accéléromètre et geomagnetic les données du magnétomètre.

La matrice rendue s'appelle une matrice de rotation. À partir de celle-ci, vous pouvez obtenir l'orientation de l'appareil avec static float[] SensorManager.getOrientation(float[] R, float[] values) avec R la matrice de rotation et values le tableau de taille 3 qui contiendra la valeur de la rotation pour chaque axe :

```
SensorManager sensorManager =
(SensorManager)getSystemService(Context.SENSOR_SERVICE);

Sensor accelerometre = sm.getDefaultSensor(Sensor.TYPE_ACCELEROMETER);
Sensor magnetometre = sm.getDefaultSensor(Sensor.TYPE_MAGNETIC_FIELD);

sensorManager.registerListener(accelerometreListener, accelerometre,
SensorManager.SENSOR_DELAY_UI);
sensorManager.registerListener(magnetometreListener, magnetometre,
SensorManager.SENSOR_DELAY_UI);

// …

float[] values = new float[3];
float[] R = new float[9];
SensorManager.getRotationMatrix(R, null, accelerometreValues,
magnetometreValues);
SensorManager.getOrientation(R, values);

Log.d("Sensors", "Rotation sur l'axe z : " + values[0]);
Log.d("Sensors", "Rotation sur l'axe x : " + values[1]);
Log.d("Sensors", "Rotation sur l'axe y : " + values[2]);
```

On suppose que le téléphone portable est posé sur une table, le haut qui pointe vers vous, l'écran vers le sol. Ainsi, l'axe z pointe de bas en haut, l'axe x de droite à gauche et l'axe y de "loin devant vous" à "loin derrière vous" (en gros c'est l'axe qui vous traverse) :

- La rotation sur l'axe z est le mouvement que vous faites pour ouvrir ou fermer une bouteille de jus d'orange posée verticalement sur une table. C'est donc comme si vous englobiez le téléphone dans votre main et que vous le faisiez tourner sur l'écran. Il s'agit de l'angle entre le nord magnétique et l'angle de l'axe y. Il vaut 0 quand l'axe y pointe vers le nord magnétique, 180 si l'axe y pointe vers le sud, 90 quand il pointe vers l'est et 270 quand il pointe vers l'ouest.
- La rotation autour de l'axe x est le mouvement que vous faites quand vous ouvrez la bouteille de jus d'orange posée sur une table mais que le bouchon pointe vers votre droite. Enfin, ce n'est pas malin parce que vous allez tout renverser par terre. Elle vaut -90 quand vous tournez vers vous (ouvrir la bouteille) et 90 quand vous tournez dans l'autre sens (fermer la bouteille).
- La rotation sur l'axe y est le mouvement que vous faites quand vous ouvrez une bouteille de jus d'orange couchée sur la table alors que le bouchon pointe vers vous. Elle vaut 180 quand vous tournez vers la gauche (fermer la bouteille) et -180 quand vous tournez vers

la droite (ouvrir la bouteille).

Enfin, il est possible de changer le système de coordonnées pour qu'il corresponde à vos besoins. C'est utile si votre application est censée être utilisée en mode paysage plutôt qu'en mode portrait par exemple. On va utiliser la méthode static boolean SensorManager.remapCoordinateSystem(float[] inR, int X, int Y, float[] outR) avec inR la matrice de rotation à transformer (celle qu'on obtient avec getRotationMatrix), X désigne la nouvelle orientation de l'axe x, Y la nouvelle orientation de l'axe y et outR la nouvelle matrice de rotation (ne mettez pas inR dedans). Vous pouvez mettre dans X et Y des valeurs telles que SensorManager.AXIS_X qui représente l'axe x et SensorManager.AXIS_MINUS_X son orientation inverse. Vous trouverez de même les valeurs SensorManager.AXIS_Y, SensorManager.AXIS_MINUS_Y, SensorManager.AXIS_Z et SensorManager.AXIS_MINUS_Z.

2.3.6. Les capteurs environnementaux

Il n'y a pas tellement à dire sur les capteurs environnementaux. On en trouve trois dans l'API 7 : le baromètre, le photomètre et le thermomètre. Ce sont tous des capteurs matériels, mais il est bien possible qu'ils ne soient pas présents dans un appareil. Tout dépend du type d'appareil, pour être exact. Sur un téléphone et sur une tablette, on en trouve rarement (à l'exception du photomètre qui permet de détecter automatiquement la meilleure luminosité pour l'écran), mais sur une station météo ils sont souvent présents. Il faut donc redoubler de prudence quand vous essayez de les utiliser, vérifiez à l'avance leur présence.

Tous ces capteurs rendent des valeurs uniques, pas de tableaux à plusieurs dimensions.

- La majorité des appareils sous Android utilisent des capteurs pour créer un lien supplémentaire entre l'utilisateur et son appareil. On rencontre trois types de capteurs : les capteurs de mouvements, les capteurs de position et les capteurs environnementaux.
- La présence d'un capteur dépend énormément des appareils, il faut donc faire attention à bien déclarer son utilisation dans le Manifest et à vérifier sa présence au moment de l'utilisation.
- Les capteurs de mouvements permettent de détecter les déplacements que l'utilisateur fait faire à son appareil. Il arrive qu'ils soient influencés par des facteurs extérieurs comme la gravité, il faut donc réfléchir à des solutions basées sur des calculs physiques quand on désire avoir une valeur précise d'une mesure.
- Les capteurs de position sont capables de repérer la position de l'appareil par rapport à un référentiel. Par exemple, le capteur de proximité peut donner la distance entre l'utilisateur et l’appareil. En pratique, le magnétomètre est surtout utilisé conjointement avec des capteurs de mouvements pour détecter d'autres types de mouvements, comme les rotations.
- Les capteurs environnementaux sont vraiment beaucoup plus rares sur un téléphone ou une tablette, mais il existe d'autres terminaux spécifiques, comme des stations météorologiques, qui sont bardés de ce type de capteurs.