Table des matières

Pourquoi Développer Sous ANDROID ?

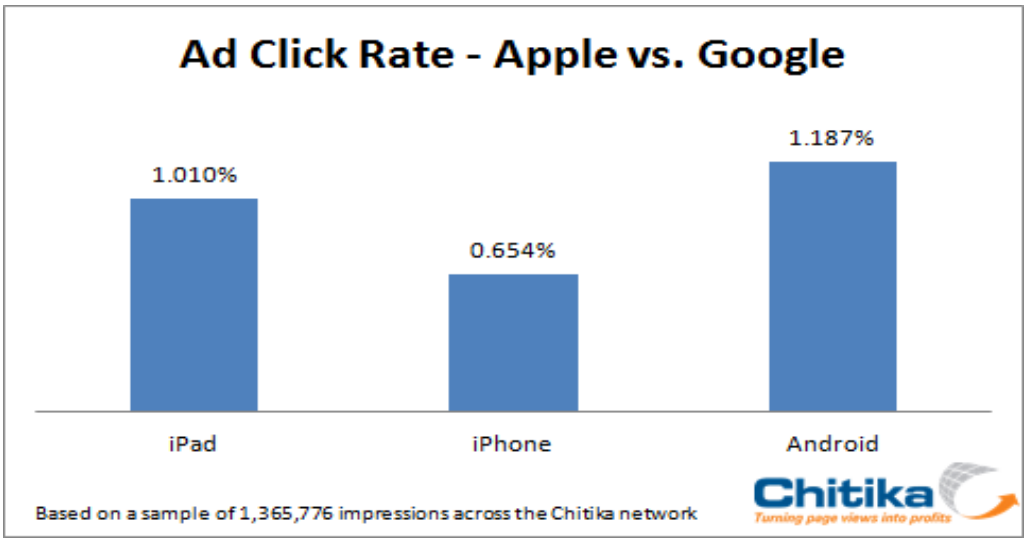
Adroid : System Open source utiliser par plusieurs sociétés (Sumsung, Sony, Google Nexus , BlackBerry …)

Tout d'abord, au niveau des outils :
Une application est développée à l'aide du SDK android et de eclipse ou androidStudio avec le plugin crée par google. À ce niveau-là, le développeur n'a pas besoin de débourser d'argent.

Une appli android est codé à l'aide de deux langages :
Tout d'abord, l'interface se développe en XML et le reste de l'appli en java. Deux langages déjà très populaire et dont les notions sont utiles pas seulement pour android.

Aucun OS spécifiques requit :
Étant donné eclipse ou androidStudio est codé en java, cette IDE fonctionne sur tous les OS qui supportent une machine virtuelle java (Qui est présent sous Windows, Mac OSX, ainsi que la plupart des distributions Linux).

Possible rémunération même si l'appli est gratuite :
Il est possible d'insérer des pubs ad-sense pour avoir une petite rémunération pour une application gratuite. À ce niveau, Google possède une certaine avance, grâce à son expérience dans la pub en ligne, ainsi qu'une étude récente prouvant que les utilisateurs android cliquent plus sur ses pubs que ceux possédant un iphone.



La possibilité de tester l'appli que l'on développe gratuitement :
Alors que sous IOS, la validation de l'application pour être utilisé sur son propre iphone est obligatoire (et payante) sous android on peut tester son application sans aucune validation.

Une visibilité accessible :
Le contrôle de publication d'une application sur l'android market est beaucoup moins stricte que chez les concurrents avec un tarif de 25 $ par an ce qui est quatre fois moins cher que pour l'app store.

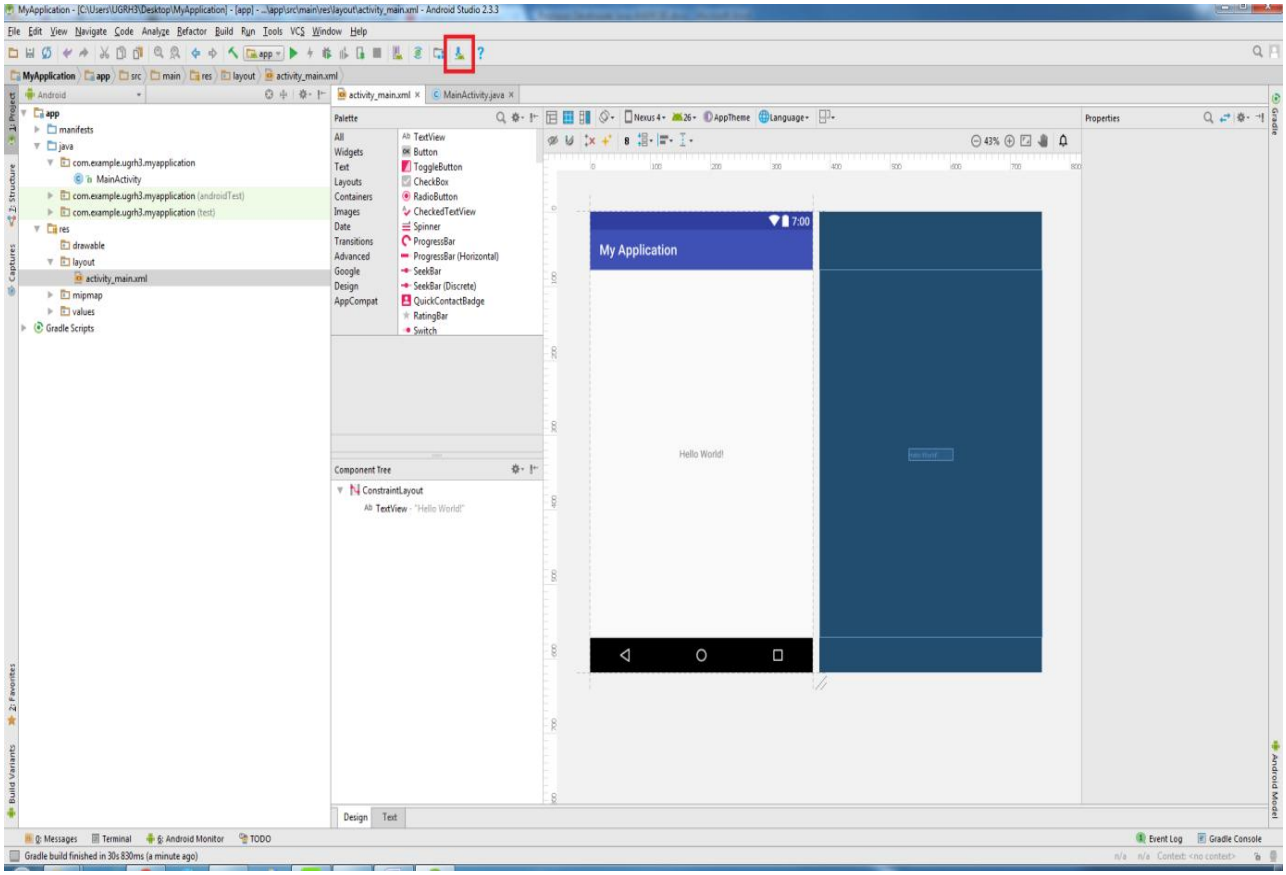
Un public de plus en plus large :
2010 est une année record pour android, qui a tripler sa part de marcher depuis 2009. À la fin de l'année, android pourrait être la deuxième plate-forme la plus populaire derrière Symbian.

Company	2Q10 Units	2Q10 Market Share (%)	2Q09 Units	2Q09 Market Share (%)
Symbian	25,386.8	41.2	20,880.8	51.0
Research In Motion	11,228.8	18.2	7,782.2	19.0
Android	10,606.1	17.2	755.9	1.8
iOS	8,743.0	14.2	5,325.0	13.0
Microsoft Windows Mobile	3,096.4	5.0	3,829.7	9.3
Linux	1,503.1	2.4	1,901.1	4.6
Other OSs	1,084.8	1.8	497.1	1.2
Total	**61,649.1**	**100.0**	**40,971.8**	**100.0**

Les produits Google :
Les services Google comme Maps, Navigation, etc. peuvent très facilement être utilisé dans chaque appli.

Paramètres de l'environnement

Sélectionner SDK Manager

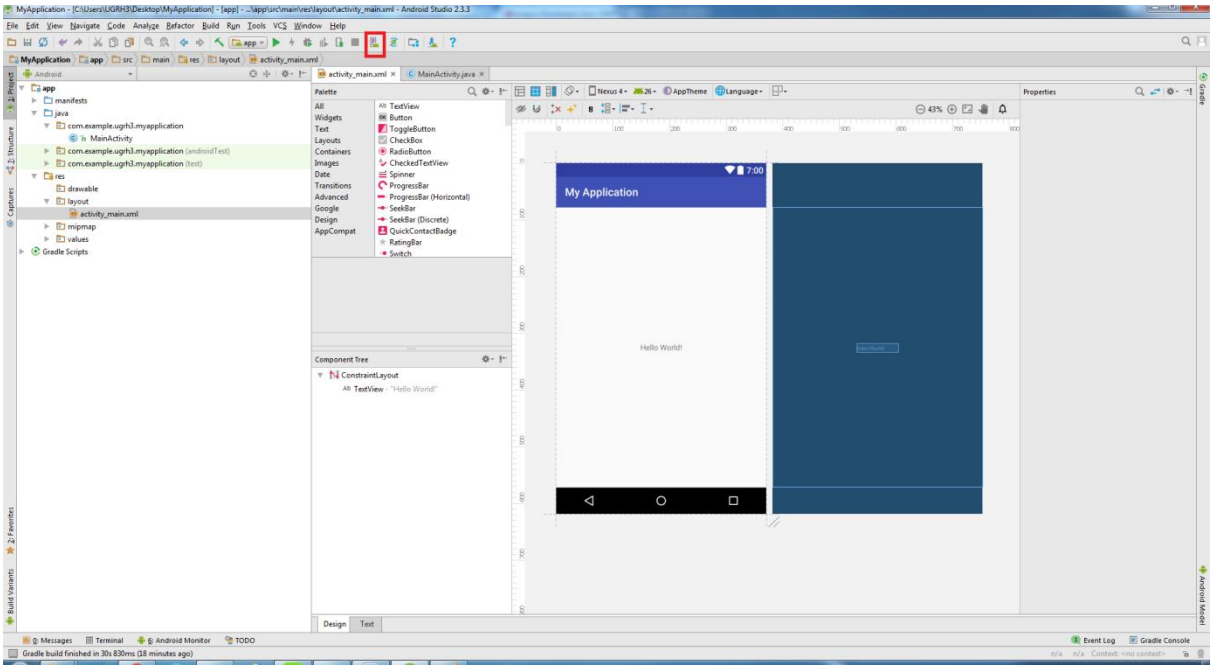


Puis dans l'onglet SDK Platforms sélectionner les versions android que vous voulez.

Activer l'onglet SDK Touls et vérifier si ces quatre objets sont sélectionnées, si non veuillez les installés

1. Google Play service
2. Google USB Driver
3. Google Repository
4. Intel x86 Emulateur Accelerateur (HAXM INSTALER)

Pour crée un appareil android virtuel Cliquer sur AVD Manager



Puis create viruel device, sélectionner le type que vous voulez et installer le.

N’oubliez pas que c’est un émulateur virtuel, donc si vous développer une application qui exécute un code sur la carte sim par exemple cet émulateur ne fonctionne pas.

Les widgets les plus simples

Ce chapitre traitera uniquement des widgets, c'est-à-dire des vues qui *fournissent* un contenu et non qui le *mettent en forme* — ce sont les layouts qui s'occupent de ce genre de choses.
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, c'est pourquoi je ne les détaillerai pas, mais vous pourrez trouver toutes les informations dont vous avez besoin sur la documentation officielle d'Android.

TEXTVIEW

Elle vous permet d'afficher une chaîne de caractères que l'utilisateur ne peut modifier. Vous verrez plus tard qu'on peut aussi y insérer des chaînes de caractères formatées, à l'aide de balises HTML.

Exemple en XML

```
<TextView
    android:id="@+id/textView"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="TextView"
    android:textSize="8pt"
    android:textColor="#112233" />
```

- **android:text** : le Texte à afficher
- **android:textSize** : la taille du texte
- **android:textColor** : La couleur du texte on peut aussi ecrire la couleur de cette facon :

```
android:textColor="@android:color/holo_red_dark"
```

Exemple en Java

```
TextView textView = (TextView)findViewById(R.id .textView);
textView.setText(R.string.textView);
textView.setTextSize(8);
textView.setTextColor(0x112233);
```

Exemple en Java

```
TextView textView = (TextView)findViewById(R.id .textView);

textView.setText(R.string.textView);

textView.setTextSize(12);

textView.setTextColor(Color.BLUE);
```

EDITTEXT

Ce composant est utilisé pour permettre à l'utilisateur d'écrire des textes. Il s'agit en fait d'un TextView éditable. Il hérite de TextView, ce qui signifie qu'il peut prendre les mêmes attributs que TextView en XML et qu'on peut utiliser les mêmes méthodes Java.

Exemple en XML

```
<EditText
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
```

```
    android:hint="Tapez votre Message"
    android:inputType="textMultiLine"
    android:lines="5" />
```

- Au lieu d'utiliser android:text, on utilise android:hint. Le problème avec android:text est qu'il remplit l'EditText avec le texte demandé, alors qu'android:hint affiche juste un texte d'indication, qui n'est pas pris en compte par l'EditText en tant que valeur (si vous avez du mal à comprendre la différence, essayez les deux).
- 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.

Exemple en Java

```
EditText editText = (EditText)findViewById(R.id.editText1)
editText.setHint("Votre Texte ici");
editText.setInputType(InputType.TYPE_TEXT_FLAG_MULTI_LINE);
editText.setLines(5);
```

BUTTON

Un simple bouton, même s'il s'agit en fait d'un TextView cliquable. Il hérite de TextView, ce qui signifie qu'il peut prendre les mêmes attributs que TextView en XML et qu'on peut utiliser les mêmes méthodes Java.

Exemple en XML

```
<Button
    android:id="@+id/button"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Afficher" />
```

Exemple en Java

```
Button btn = (Button)findViewById(R.id.button);
btn.setText("Enregistrer");
```

CHECKBOX

Une case qui peut être dans deux états : cochée ou pas.
Elle hérite de Button, ce qui signifie qu'elle peut prendre les mêmes attributs que Button en XML et qu'on peut utiliser les mêmes méthodes Java.

Exemple en XML

```
<CheckBox
    android:id="@+id/checkBox"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Remember me"
    android:checked="false"/>
```

Exemple en Java

```
CheckBox chk =  (CheckBox)findViewById(R.id.checkBox);
chk.setText("Remember me");
```

```
chk.setChecked(true);
if(chk.isChecked());
```

RadioButton et RadioGroup

Même principe que la CheckBox, à la différence que l'utilisateur ne peut cocher qu'une seule case. Il est plutôt recommandé de les regrouper dans un RadioGroup.
RadioButton hérite de Button, ce qui signifie qu'il peut prendre les mêmes attributs que Button en XML et qu'on peut utiliser les mêmes méthodes Java.
Un RadioGroup est en fait un layout, mais il n'est utilisé qu'avec des RadioButton, c'est pourquoi on le voit maintenant. Son but est de faire en sorte qu'il puisse n'y avoir qu'un seul RadioButton sélectionné dans tout le groupe.

Exemple en XML

```
<RadioGroup
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:orientation="horizontal">

    <RadioButton
        android:id="@+id/radioButton3"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="RadioButton" />

    <RadioButton
        android:id="@+id/radioButton2"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="RadioButton" />

    <RadioButton
        android:id="@+id/radioButton"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="RadioButton" />
</RadioGroup>
```

Exemple en Java

```
RadioButton rb1 = (RadioButton) findViewById(R.id.radioButton4);
rb1.setText("Texte 1");
if(  rb1.isChecked());
```

Organiser son interface avec des layouts

Il vous suffit de regarder n'importe quelle application Android pour réaliser que toutes les vues ne sont pas forcément organisées comme cela et qu'il existe une très grande variété d'architectures différentes. C'est pourquoi nous allons maintenant étudier les différents layouts, afin d'apprendre à placer nos vues comme nous le désirons. Nous pourrons ainsi concevoir une application plus attractive, plus esthétique et plus ergonomique !

LinearLayout : placer les éléments sur une ligne

Comme son nom l'indique, 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).

Premier exemple

```
<LinearLayout
    android:layout_width="match_parent "
    android:layout_height="match_parent "
    android:orientation="vertical"
    tools:layout_editor_absoluteY="8dp"
    tools:layout_editor_absoluteX="8dp">

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

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

Le rendu de ce code se trouve à la figure suivante.

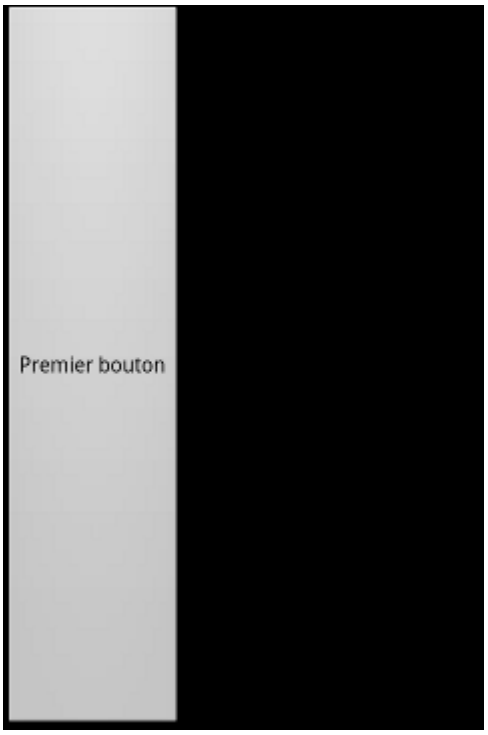
Les deux boutons prennent toute la largeur

- Le LinearLayout est vertical et prend toute la place de son parent (vous savez, l'invisible qui prend toute la place dans l'écran).
- 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.

Deuxième exemple

```
<LinearLayout
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    tools:layout_editor_absoluteY="8dp"
    tools:layout_editor_absoluteX="8dp">

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

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

Le rendu de ce code se trouve à la figure suivante.



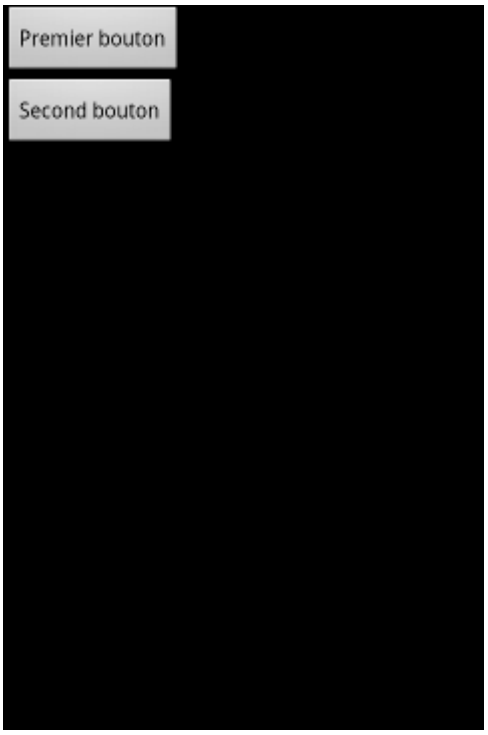
Le premier bouton fait toute la hauteur, on ne voit donc pas le deuxième bouton

- 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 second bouton se fait écraser. C'est pour cela qu'on ne le voit pas.

Troisième exemple

```
<LinearLayout
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:orientation="vertical"
    tools:layout_editor_absoluteY="8dp"
    tools:layout_editor_absoluteX="8dp">

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

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

Le rendu de ce code se trouve à la figure suivante.

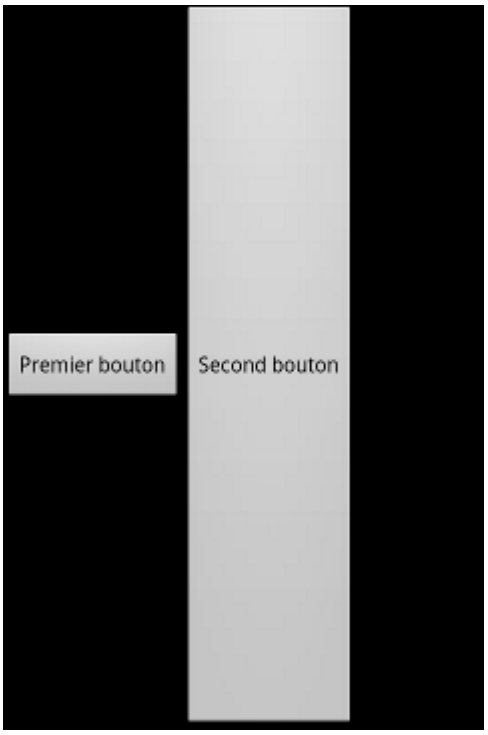


- Le LinearLayout est vertical et prend toute la place en largeur mais uniquement la taille nécessaire en hauteur : dans ce cas précis, la taille nécessaire sera calculée en fonction de la taille des enfants.
- Le premier bouton prend toute la place possible dans le parent. Comme le parent prend le moins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

Quatrième exemple

```
<LinearLayout
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="horizontal"
    tools:layout_editor_absoluteX="8dp"
    tools:layout_editor_absoluteY="8dp">

    <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="match_parent"
        android:text="Second bouton" />
</LinearLayout>
```

Le rendu de ce code se trouve à la figure suivante.



Le premier bouton prend uniquement la place nécessaire et le deuxième toute la hauteur

- 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 grande 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.

Pour ne pas trouver des dificulter si vous voulez travailler avec android:layout_weight veuillez mettre la valeur 0 à : **android:layout_width="0dp".** Voir Cinquième exemples

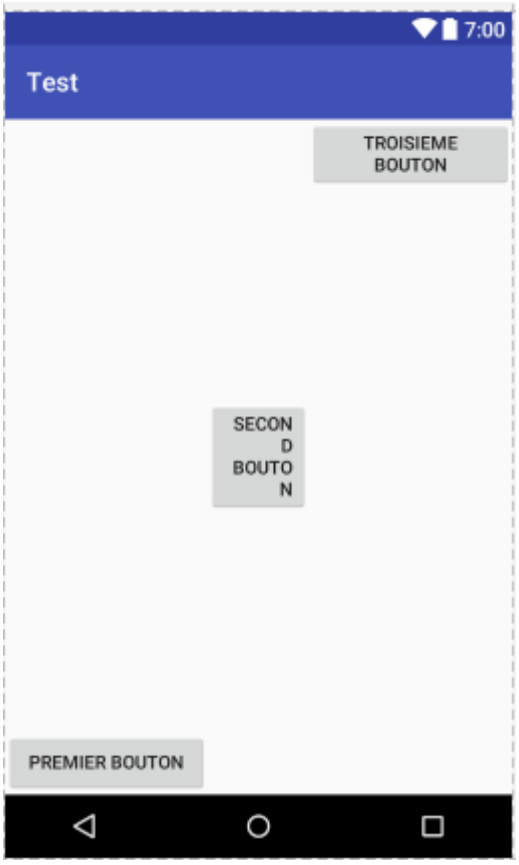
Dernier attribut particulier pour les widgets de ce layout, android:layout_gravity, qu'il ne faut pas confondre avec android:gravity. android:layout_gravity vous permet de déterminer comment se placera la vue dans le parent, alors que android:gravity vous permet de déterminer comment se placera le contenu 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 ?).

Vous prendrez bien un petit exemple pour illustrer ces trois concepts ?

Cinquième exemples

```
<Button
    android:id="@+id/Premier"
    android:layout_width="0dp"
    android:layout_height="wrap_content"
    android:layout_gravity="bottom"
    android:layout_weight="40"
    android:text="Premier bouton" />

<Button
    android:id="@+id/Second"
    android:layout_width="0dp"
    android:layout_height="wrap_content"
    android:layout_gravity="center"
    android:gravity="bottom|right"
    android:layout_weight="20"
    android:text="Second bouton" />

<Button
    android:id="@+id/Troisieme"
    android:layout_width="0dp"
    android:layout_height="wrap_content"
    android:layout_gravity="top"
    android:layout_weight="40"
    android:text="Troisieme bouton" />
```

Le rendu de ce code se trouve à la figure suivante.



Trois boutons placés différemment

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).

RELATIVELAYOUT : PLACER LES ÉLÉMENTS LES UNS EN FONCTION DES AUTRES

De manière totalement différente, 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.

Premier exemple

```
<RelativeLayout
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:layout_editor_absoluteY="8dp"
    tools:layout_editor_absoluteX="8dp">
    <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>
```

Le rendu de ce code se trouve à la figure suivante.

Deux vues sont empilées

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.

Deuxième exemple

```
<RelativeLayout
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:layout_editor_absoluteY="8dp"
    tools:layout_editor_absoluteX="8dp">
    <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>
```

Le rendu de ce code se trouve à la figure suivante.

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).
- android:layout_toLeftOf (pour aligner le bord droit d'une vue au bord gauche d'une autre).

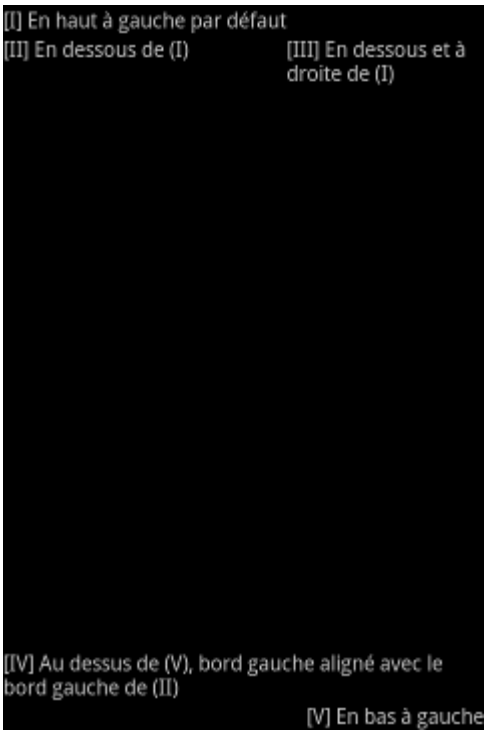
Ceux qui permettent de coller deux bords similaires ensemble. On trouve :

- 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).
- android:layout_alignRight (pour aligner le bord droit de la vue avec le bord droit d'une autre).

Troisième exemple

```
<RelativeLayout
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:layout_editor_absoluteY="8dp"
    tools:layout_editor_absoluteX="8dp">
    <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>
```

Le rendu de ce code se trouve à la figure suivante.



Le problème de ce layout, c'est qu'une petite modification dans l'interface graphique peut provoquer de grosses modifications dans tout le fichier XML, il faut donc savoir par avance très précisément ce qu'on veut faire.

Il s'agit du layout le plus compliqué à maîtriser, et pourtant le plus puissant tout en étant l'un des moins gourmands en ressources. Je vous encourage fortement à vous entraîner à l'utiliser.

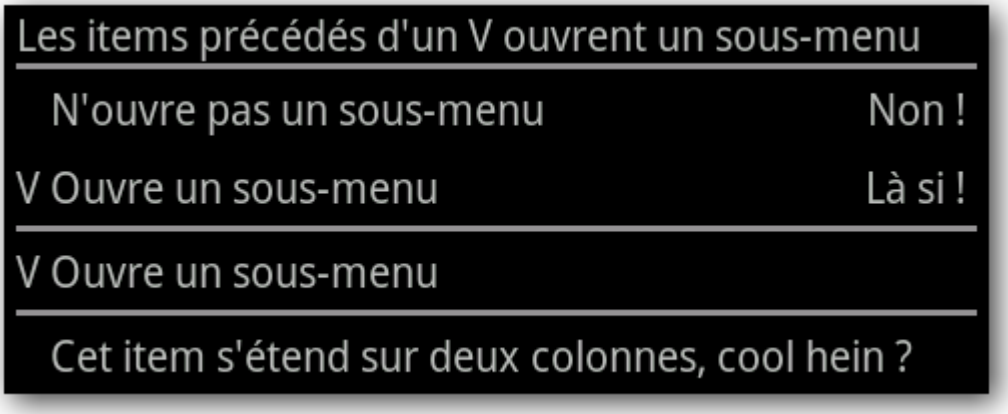
TableLayout : placer les éléments comme dans un tableau

Il permet d'organiser les éléments en tableau, comme en HTML, mais sans les bordures. Voici un exemple d'utilisation de ce layout :

```
<TableLayout
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:layout_editor_absoluteY="8dp"
    tools:layout_editor_absoluteX="8dp">
    android:stretchColumns="1">
    <TextView
        android:text="Les items précédés d'un V ouvrent un sous-menu"
        />
    <View
        android:layout_height="2dip"
        android:background="@android:color/black"
        />
    <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="@android:color/black"
        />
    <TableRow>
        <TextView
            android:text="V"
            />
        <TextView
            android:text="Ouvre un sous-menu"
            android:padding="3dip"
            />
    </TableRow>
    <View
        android:layout_height="2dip"
        android:background="@android:color/black"
        />
    <TableRow>
        <TextView
            android:layout_column="1"
            android:layout_span="2"
            android:text="Cet item s'étend sur deux colonnes, cool hein ?"
            android:padding="3dip"
            />
    </TableRow>
</TableLayout>
```

Ce qui donne la figure suivante.



Le contenu est organisé en tableau

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="@android:color/black" />
```

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, cool hein ?"
        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 match_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.

GRIDLAYOUT

Le GridLayout permet de placer des éléments sur une grille, très pratique pour des tableaux par exemple ici, pour un morpion. Deux options s'avère nécessaire : android:columnCount et android:rowCount qui correspondent au nombre de lignes et de colonnes sur notre grille.

Exemple du Morpion


```
<GridLayout
    android:id="@+id/gridtest"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:columnCount="3"
    android:padding="60dp"
    android:rowCount="3">

    <Button
        android:id="@+id/button1"
        android:text="X" />

    <Button
        android:id="@+id/button2"
        android:text="O" />

    <Button
        android:id="@+id/button3"
        android:text="X" />

    <Button
        android:id="@+id/button4"
        android:text="X" />

    <Button
        android:id="@+id/button5"
        android:text="O" />

    <Button android:text="X" />

    <Button
        android:id="@+id/button7"
        android:text="X" />

    <Button
        android:id="@+id/button8"

        android:text="O" />

    <Button
        android:id="@+id/button9"
        android:text="O" />
</GridLayout>
```

FRAMELAYOUT : UN LAYOUT UN PEU SPÉCIAL

Ce layout est plutôt utilisé pour afficher une unique vue. Il peut sembler inutile comme ça, mais ne l'est pas du tout ! Il n'est destiné à afficher qu'un élément, mais il est possible d'en mettre plusieurs dedans puisqu'il s'agit d'un ViewGroup. Si par exemple vous souhaitez faire un album photo, il vous suffit de mettre plusieurs éléments dans le FrameLayout et de ne laisser qu'une seule photo visible, en laissant les autres invisibles grâce à l'attribut android:visibility (cet attribut est disponible pour toutes les vues). Pareil pour un lecteur de PDF, il suffit d'empiler toutes les pages dans le FrameLayout et de n'afficher que la page actuelle, celle du dessus de la pile, à l'utilisateur. Cet attribut peut prendre trois valeurs :

ScrollView : faire défiler le contenu d'une vue

Elle est par ailleurs un peu particulière puisqu'elle fait juste en sorte d'ajouter une barre de défilement verticale à un autre layout. En effet, si le contenu de votre layout dépasse la taille de l'écran, une partie du contenu sera invisible à l'utilisateur. De façon à rendre ce contenu visible, on peut préciser que la vue est englobée dans une ScrollView, et une barre de défilement s'ajoutera automatiquement.

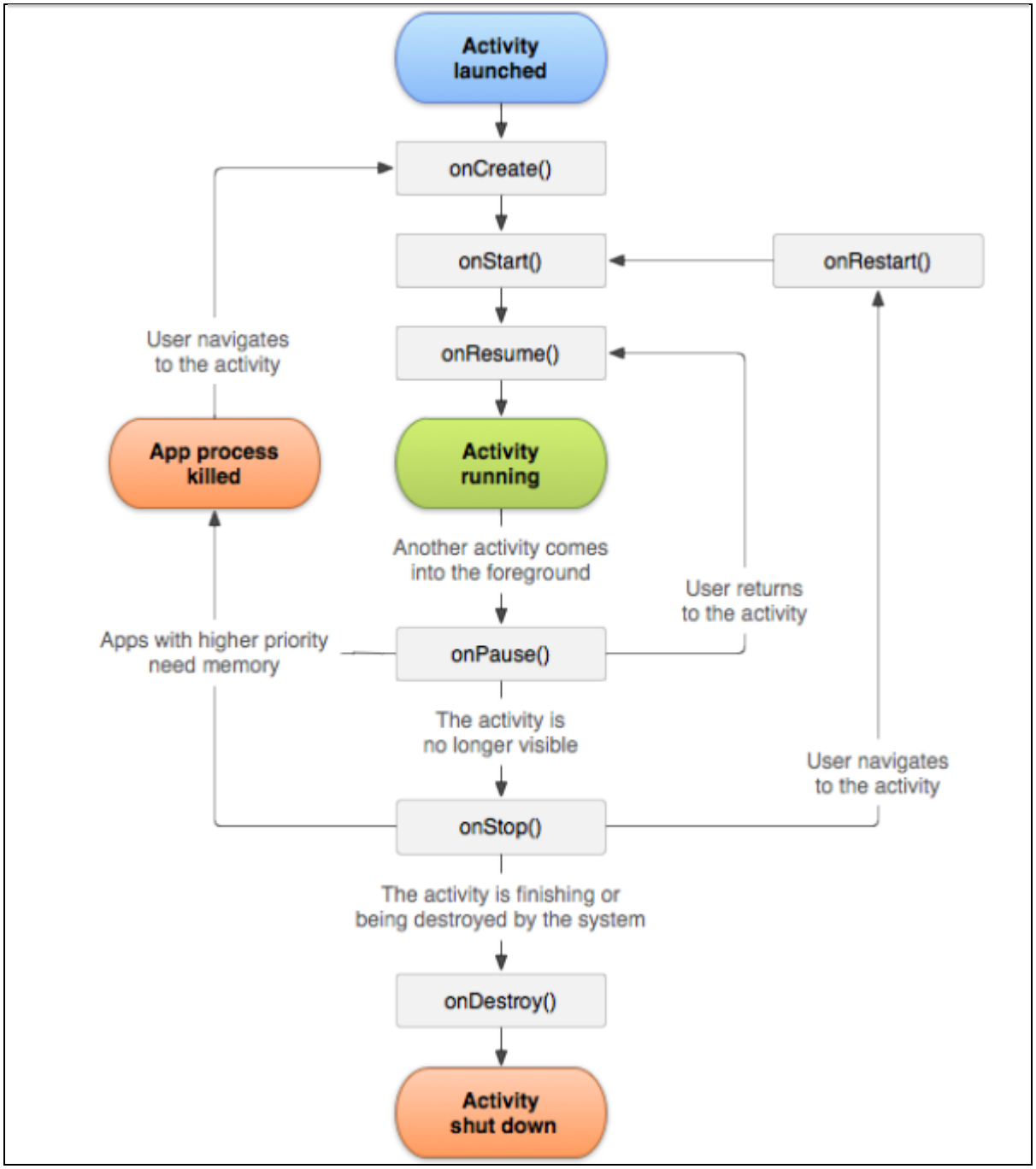
Ce layout hérite de FrameLayout, par conséquent il vaut mieux envisager de ne mettre qu'une seule vue dedans.
Il s'utilise en général avec LinearLayout, mais peut être utilisé avec tous les layouts… ou bien des widgets ! Par exemple :

```
<ScrollView
    android:layout_width="368dp"
    android:layout_height="495dp"
    tools:layout_editor_absoluteY="8dp"
    tools:layout_editor_absoluteX="8dp">

    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:orientation="vertical" />
</ScrollView>
```

Attention cependant, il ne faut pas mettre de widgets qui peuvent déjà défiler dans une ScrollView, sinon il y aura conflit entre les deux contrôleurs et le résultat sera médiocre. Nous n'avons pas encore vu de widgets de ce type, mais cela ne saurait tarder.

Cycle de vie d'une activité

Toute activité passe par plusieurs états durant son cycle de vie, il est important de connaitre ce cycle ainsi que les méthodes qui sont appelées à chaque fois que l'application bascule d'un état vers l'autre. Le diagramme ci-dessous résume le cycle de vie d'une activité



onCreate() / onDestroy(): permet de gérer les opérations à faire avant l'affichage de l'activity, et lorsqu'on détruit complètement l'activity de la mémoire. On met en général peu de code dans **onCreate()** afin d'afficher l'activity le plus rapidement possible.

onStart() / onStop(): ces méthodes sont appelées quand l'activity devient visible/invisible pour l'utilisateur.
onPause() / onResume(): une activité peut rester visible mais être mise en pause par le fait qu'une autre activité est en train de démarrer, par exemple B. **onPause()** ne doit pas être trop long, car B ne sera pas créé tant que **onPause()** n'a pas fini son exécution.
onRestart(): cette méthode supplémentaire est appelée quand on relance une activité qui est passée par **onStrop()**. Puis **onStart()** est aussi appelée. Cela permet de différencier le premier lancement d'un relancement.
Pour utiliser ces méthodes il faut juste les implémenter dans la classe java. Exemple:

```
@Override
protected  void  onPause()
{
    super.onPause();
    Toast.makeText(this, "Methodes pnPause", Toast.LENGTH_SHORT).show();
}
```

Répondre aux évènements

Pour répondre à un appui sur le bouton il suffit de définir un attribut android:onClick pour le bouton en lui donnant comme valeur le nom de la méthode qui devrait être appelée quand le bouton est appuyé, et d'implémenter cette méthode de réponse dans la classe principale de l'activité.
Dans le fichier xml du *layout*, rajoutez l'attribut android:onClick à l'élément bouton tel que :

```
<Button
    android:id="@+id/btnValider"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Valider"
    android:onClick="LancerMessage" />
```

Dans la classe Principale rajoutez la méthode

```
public void LancerMessage(View view)
{
    Toast.makeText(this, "Message affiché lorsqu'on a cliqué sur valider",
Toast.LENGTH_SHORT).show();
}
```

Il faut absolument respecter cette signature pour la méthode afin que le système puisse l'associer au nom donné par android:onClick. Le paramètre view est rempli par le système et correspond à l'élément qui a généré l'évènement (le bouton Envoi dans notre cas).

LES LISTENER SOUS ANDROID

Les listener permettent d'écouter, de surveiller certains objets (graphique ou matériel) et de nous avertir dès qu'un évènement particulier vient à ce produire. L'avantage étant de ne pas avoir à s'occuper nous même de cette surveillance dans notre application, mais de laisser cette tâche au système !

Evénement onTextChanged du EditText :

```
<EditText
    android:id="@+id/txtTest"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:ems="10"
    android:inputType="textPersonName"

    />
```

Dans la classe java rajoutez le code suivant dans la méthode onCreate :

```
final EditText txt=(EditText)findViewById(R.id.txtTest);
txt.addTextChangedListener(new TextWatcher() {

    @Override
public void beforeTextChanged(CharSequence charSequence, int i, int i1, int i2)
{    }
    @Override
public void onTextChanged(CharSequence text, int start, int lengthBefore, int
lengthAfter) {
        Toast.makeText(MainActivity.this, txt.getText().toString(),
```

```
Toast.LENGTH_LONG).show();
    }
    @Override
    public void afterTextChanged(Editable editable) {    }
});
```

Evénement OnCheckedChange du CheckBox:

```
<CheckBox
    android:id="@+id/checkBox"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:text="Remember me" />
```

Dans la classe java rajoutez le code suivant dans la méthode onCreate :

```
final CheckBox chk=(CheckBox)findViewById(R.id.checkBox);
     chk.setOnCheckedChangeListener(new CompoundButton.OnCheckedChangeListener()
{
         @Override
         public void onCheckedChanged(CompoundButton compoundButton, boolean b) {
             if (b)
             {
                 Toast.makeText(MainActivity.this, "Vous avez cocher le
Checkbox", Toast.LENGTH_SHORT).show();
             }
             else
             {
                 Toast.makeText(MainActivity.this, "Vous avez décocher le
Checkbox", Toast.LENGTH_SHORT).show();
             }
         }
     });
```

Evénement OnCheckedChangeListenerdu RadioGroup:

```
<RadioGroup
    android:id="@+id/groupCivilite"
    android:layout_width="match_parent"
    android:layout_height="wrap_content">

    <RadioButton
        android:id="@+id/rbMme"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_weight="1"
        android:text="Mme" />

    <RadioButton
        android:id="@+id/rbMlle"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_weight="1"
        android:text="Mlle" />

    <RadioButton
        android:id="@+id/rbMr"
        android:layout_width="wrap_content"
```

```
        android:layout_height="wrap_content"
        android:layout_weight="1"
        android:text="Mr" />
</RadioGroup>
```

Dans la classe java rajoutez le code suivant dans la méthode onCreate :

```
RadioGroup rg=(RadioGroup)findViewById(R.id.groupCivilite);
rg.setOnCheckedChangeListener(new RadioGroup.OnCheckedChangeListener() {
    @Override
    public void onCheckedChanged(RadioGroup radioGroup,  int i) {
        RadioButton rb=(RadioButton)radioGroup.findViewById(i);
        boolean cocher=rb.isChecked();
        if (cocher)
            Toast.makeText(MainActivity.this, rb.getText(),
Toast.LENGTH_SHORT).show();
    }
});
```

`OnCheckedChange`

Les ressources

Les ressources sont des fichiers qui contiennent des informations qui ne sont :

- Pas en Java (ce n'est donc pas du code).
- Pas dynamique (le contenu d'une ressource restera inchangé entre le début de l'exécution de votre application et la fin de l'exécution).

Android est destiné à être utilisé sur un très grand nombre de supports différents, et il faut par conséquent s'adapter à ces supports. L'avantage des ressources, c'est qu'elles nous permettent de nous adapter facilement à toutes ces situations différentes.

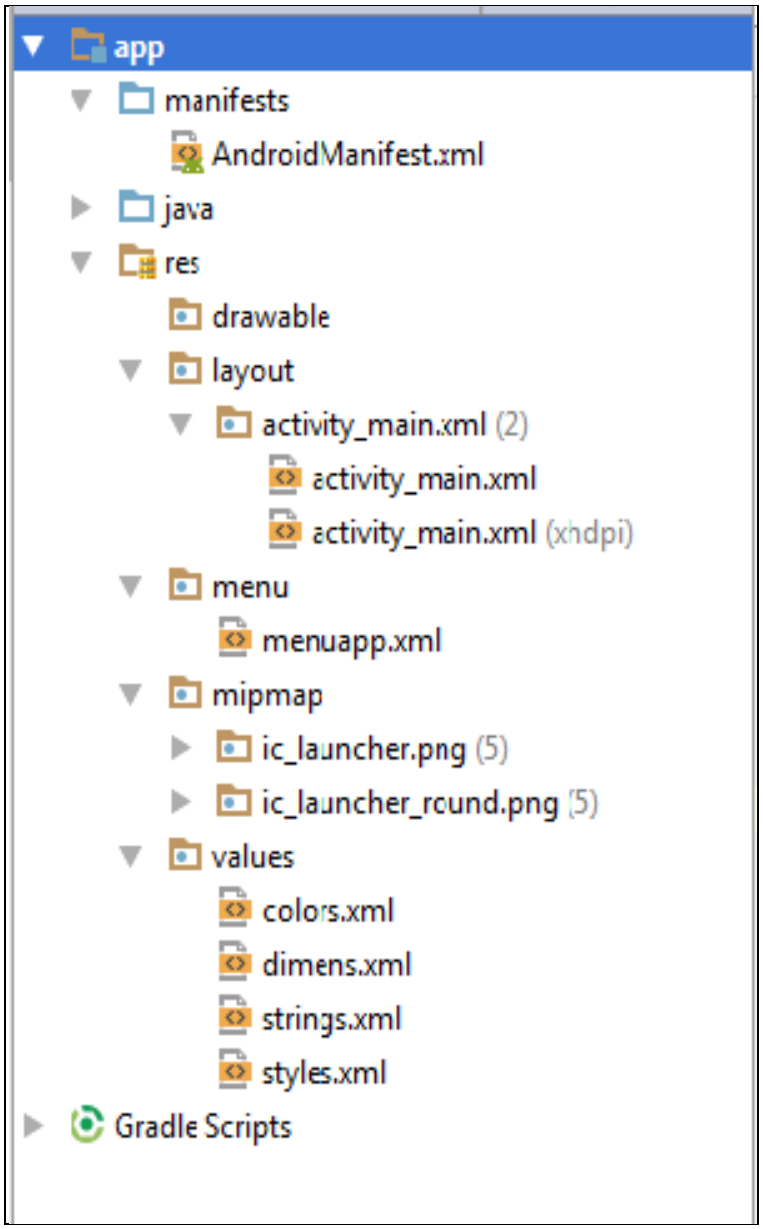
PRENONS QUELQUES EXEMPLES CONCRETS :

- Votre application affiche une image. Si cette image est matricielle, c'est-à-dire qu'elle est constituée de pixels comme une photo par exemple, alors il faudra adapter sa taille à l'écran du terminal. Si la photo est petite, il faudra agrandir l'image. Et qu'est-ce qui se passe quand on agrandit une image matricielle ? Elle perd en qualité. L'idéal sera donc de préparer à l'avance plusieurs images à afficher en fonction de la résolution de l'écran.
- Votre application fonctionne super bien en français, vous avez donc décidé de la traduire en anglais. Aïe, tous vos textes se trouvent dans le code, comment allez-vous faire pour retrouver chaque texte et le traduire efficacement en anglais ? Heureusement avec les ressources, vous pouvez organiser facilement les chaînes de caractères pour plus facilement travailler avec.

LES DIFFÉRENTS TYPES DE RESSOURCES

Les ressources sont des éléments capitaux dans une application Android. On y trouve par exemple des chaînes de caractères ou des images. Comme Android est destiné à être utilisé sur une grande variété de supports, il fallait trouver une solution pour permettre à une application de s'afficher de la même manière sur un écran 7" que sur un écran 10", ou faire en sorte que les textes s'adaptent à la langue de l'utilisateur. C'est pourquoi les différents éléments qui doivent s'adapter de manière très précise sont organisés de manière tout aussi précise, de façon à ce qu'Android sache quels éléments utiliser pour quels types de terminaux.

On découvre les ressources à travers une hiérarchie particulière de répertoires. Vous pouvez remarquer qu'à la création d'un nouveau projet, Android Studio crée certains répertoires par défaut, comme le montre la figure suivante.



L'emplacement des ressources au sein du projet

Pour permettre à Android de les retrouver facilement, chaque type de ressources est associé à un répertoire particulier. Voici un tableau qui vous indique les principales ressources que l'on peut trouver, avec le nom du répertoire associé.

Type	Description	Présence de fichiers XML
Dessin et image (res/drawable)	On y trouve les images matricielles (les images de type PNG, JPEG ou encore GIF) ainsi que des fichiers XML qui permettent de décrire des dessins (ce qui donne des images vectorielles qui ne se dégradent pas quand on les agrandit).	Oui
Mise en page ou interface graphique (res/layout)	Les fichiers XML qui représentent la disposition des vues (on abordera cet aspect, qui est très vaste, dans la prochaine partie).	Exclusivement
Menu (res/menu)	Les fichiers XML pour pouvoir constituer des menus.	Exclusivement
Différentes variables (res/values)	Il est plus difficile de cibler les ressources qui appartiennent à cette catégorie tant elles sont nombreuses. On y trouve entre autre des variables standards, comme des chaînes de caractères, des dimensions, des couleurs, etc.	Exclusivement

La colonnePrésence de fichiers XML indique la politique à adopter pour les fichiers XML de ce répertoire. Elle vaut :

- Exclusivement, si les fichiers de cette ressource sont tout le temps des fichiers XML.
- Oui, si les fichiers peuvent être d'un autre type que XML, en fonction de ce qu'on veut faire. Ainsi, dans le répertoire drawable/, on peut mettre des images ou des fichiers XML dont le contenu sera utilisé par un interpréteur pour dessiner des images.
- Le moins possible, si les fichiers doivent de préférence ne pas être de type XML. Pourquoi ? Parce que tous les autres répertoires sont suffisants pour stocker des fichiers XML. Alors, si vous voulez placer un fichier XML dans le répertoire raw/, c'est qu'il ne trouve *vraiment* pas sa place dans un autre répertoire.

Il existe d'autres répertoires pour d'autres types de ressources, mais je ne vais pas toutes vous les présenter, les principales sont déjà là.

LA GESTION DES STRINGS SOUS ANDROID (STRING.XML)

Nous allons aborder **la gestion des chaines de caractères sous Android**. Les **strings** servent à référencer des chaines de caractères que vous allez utiliser dans votre application.

➢ *Création de chaines de caractères*

Le fichier qui contiendra vos chaines de caractères ne doit pas forcement se nommer **strings.xml**. Vous pouvez lui donner le nom que vous souhaité, il doit seulement se trouver dans le dossier *values* et entre des balises <string>.
Voici un petit exemple :

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
```

```
    <string name="nom_string">texte_string</string>
</resources>
```

Vous pouvez utiliser une chaîne que vous avez déclaré depuis :

- Un fichier Java : R.string.nom_string
- Un fichier Xml : @string/nom_string

Gestion du pluriel

Pour vous permettre de gérer le pluriel de vos chaines de caractères, android intègre une balise très pratique (plurals).

Voici un petit exemple :

```
<plurals name="nombre_de_tutos">
    <item quantity="zero">Aucun tutoriel n\'est disponible</item>
    <item quantity="one">un tutoriel est disponible</item>
    <item quantity="other">Plusieurs tutoriaux sont disponibles</item>
</plurals>
```

L'attribut quantité peut avoir les valeurs suivantes :

- **zero**: Traitement du cas de 0 élément.
- **one** : Traitement du cas d'un élément.
- **two** : Traitement du cas de deux éléments.
- **few** : Traitement d'un petit nombre d'éléments (3, 4 par exemple).
- **many** : Traitement d'un grand nombre d'éléments (10, 12 par exemple).
- **other** : Traitement des autres cas

Pour pouvoir récupérer ces chaines de caractéres en Java, la méthode **getQuantityString** est disponible.

```
public String getQuantityString(int id, int quantity) throws Resources.NotFoundException {
  throw new RuntimeException("Stub!");
}
```

- Cette méthode retourne la chaine de caractére souhaitée.
- Le premier argument est l'identifiant de la chaine.
- Le deuxiéme argument est la quatité souhaitée.
- Cette méthode peut lancer l'exception ResourceNotFoundException, si aucune ressource ne correspond à votre id.

Voici un petit exemple correspondant au cas cité au dessus :

```
Resources res = getResources();
String numberOfTutos = res.getQuantityString(R.plurals.nombre_de_tutos, 2);
Toast.makeText(this, numberOfTutos, Toast.LENGTH_SHORT).show();
```

Formatage des chaines de caractères

Échapper les apostrophes

Si vous voulez inclure des **apostrophes** dans votre chaine de caractére, vous devez l'échaper. Voici un petit exemple :

```
<string name="apostrophe1">"L'exemple d'un échappement"</string>
<string name="apostrophe2">L\'exemple d\'un échappement </string>
```

Ajouter des arguments à vos chaines de caractères

Vous pouvez rajouter des arguments à vos chaines de caractère afin de les inclure plus tard.

Voici un petit exemple :

```
<string name="reception_message">%1$s! Vous avez reçu %2$d nouveaux
messages.</string>
```

- Nous avons rajouté deux arguments à notre chaine de caractère.
- Le premier correspond à une chaine de caractére et le deuxiéme à un entier.

Puis dans votre application, vous pouvez rajouter ces arguments à la volée.

```
Resources res = getResources();
String message = String.format(res.getString(R.string.reception_message),
"zakaria", 5);
Toast.makeText(this, message.toString(), Toast.LENGTH_SHORT).show();
```

Vous pouvez utiliser quelques balises HTML pour personnaliser vos chaines de caractères.

```
<string name="bienvenue">Bienvenue sur <b>votre application de test</b>!</string>
```

Les balises supportées sont :

- b : Gras
- i : Italic
- u : Souligné

Internationalisation

Vous pouvez internationaliser les chaines de caractères utilisées dans votre application. Pour cela il suffit de créer un dossier *values* par langues supportées.

Par exemple, si votre application supporte l'anglais, le français et l'espagnol. Vous devez avoir 3 dossiers values :

- values : Dossier par défaut utilisé pour l'Anglais
- values-fr : Dossier pour le français
- values-es : Dossier pour l'espagnol

Lors du lancement de l'application sur un device, la langue correspondante à celle du device sera sélectionnée. Si elle n'est pas supportée, la langue par défaut sera sélectionnée (anglais).

LA GESTION DES DIMENSIONS (DIMENS.XML)

Ce fichier va permettre de déclarer des dimensions dans notre application, je vous vois déjà avec vos grands yeux me dire « Bah quand nous avons ajouté des paddings, nous avons déclaré nos valeurs dans le fichier layout directement ». Oui mais, imaginez vous, vous décidez d'aligner plusieurs textes (une centaine) à 15 dip de la gauche de votre écran par exemple. Et d'un seul coup, vous vous dîtes mince c'est trop proche, j'aurais du mettre à 30dip, mais la flemme de repasser sur tout les textes. Bah voici le cas où il faut utiliser le fichier dimens.

On explique le fonctionnement ? Ce fichier, comme d'ailleurs la plupart des fichiers du dossier values est très simple à comprendre et à utiliser. Je prend une ligne du fichier pour vous l'expliquer.

```
<dimen name="activity_horizontal_margin">16dp</dimen>
```

Alors il faut comprendre que ce fichier fonctionne avec des couples clé/valeur. Nous allons définir la clé à l'aide de l'attribut name et la valeur sera comprise entre les deux tags dimen.

Pour utiliser cette dimension dans un layout, c'est très simple, vous utilisez « @dimen/clé ». Ca doit vous dire quelque chose, les paddings dans activity_main.xml était déclarer avec ce principe avant qu'on les changent.

Exemple :

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
         <dimen name="padding_small">8dp</dimen>
        <dimen name="padding_medium">8dp</dimen>
        <dimen name="padding_large">16dp</dimen>

        <dimen name="normal_text_size">15sp</dimen>
        <dimen name="normal_padding">20dp</dimen>
        <dimen name="large_padding">100dp</dimen>
        <dimen name="small_padding">10dp</dimen>
  </resources>
```

Pour utiliser ses valeurs vous devez affecter le nom de la dimension souhaité à la propriété de la view.

Exemple :

```
<TextView
    android:id="@+id/textView"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="TextView"
    android:textSize="@dimen/normal_text_size"
    android:paddingBottom="@dimen/large_padding"
 />
```

LA GESTION DES STYLES (STYLES.XML)

Un style c'est quoi?
Alors un style c'est une collection de propriétés (attributs) qui vont spécifier le format et le look d'un objet ou d'un conteneur. On va pouvoir spécifier dans le style la hauteur, la largeur, une couleur de texte, la taille du texte et plein d'autres choses.
Un exemple ça vaut peut être mieux que plein de blabla non ? Prenons un *TextView*

```
<TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="@string/on_teste_un_style"
    android:textColor="@color/bleu_creer"/>
```

Nous allons mettre tout les attributs dans un style sauf le texte. « Pourquoi mettons nous ces attributs dans un style ? ». Nous faisons ceci toujours dans les soucis dé réutilisabilité. De pouvoir réutiliser le code à plusieurs endroits. Nous imaginons toujours notre centaine de *TextView*, et on veut qu'ils soient tous écrit en bleu, d'une taille et d'une largeur spécifique. Bah dans ce cas là, nous allons utiliser un **style**.
Voici comment on déclare un style

```
<style name="NotreSyle">
    <item name="android:layout_width">wrap_content</item>
    <item name="android:layout_height">wrap_content</item>
    <item name="android:color">@color/bleu_creer</item>
</style>
```

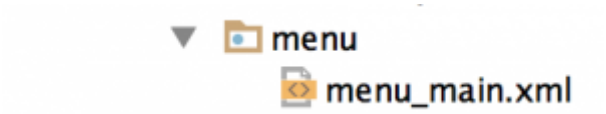
Donc on donne un nom à notre style qui sera la **clé** pour l'utiliser. Et on déclare chaque attribut comme des items. Voyons maintenant son utilisation sur notre *TextView*.

```
<TextView
    android:text="@string/on_teste_un_style"
        style="@style/NotreSyle"
     />
```

On voit que contrairement aux autres attributs que nous n'allons pas chercher notre style avec le préfixe android: On utilise ensuite notre style comme tous les autres couples clé/valeur.

LES MENUS

Que retrouve-on dans le dossier menu ? A quoi servent les fichiers contenus dans le répertoire ? Nous allons répondre à ces deux questions dans un instant.



Vous pouvez remarquer que le répertoire **menu** contient un fichier nommé « *menu_main.xml* ». Voici ce qu'il contient :

```
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:android="http://schemas.android.com/apk/res/android">

</menu>
```

La premiere chose a faire est de modifie cette ligne
<menu xmlns:android="http://schemas.android.com/apk/res/android">
Par celle-ci

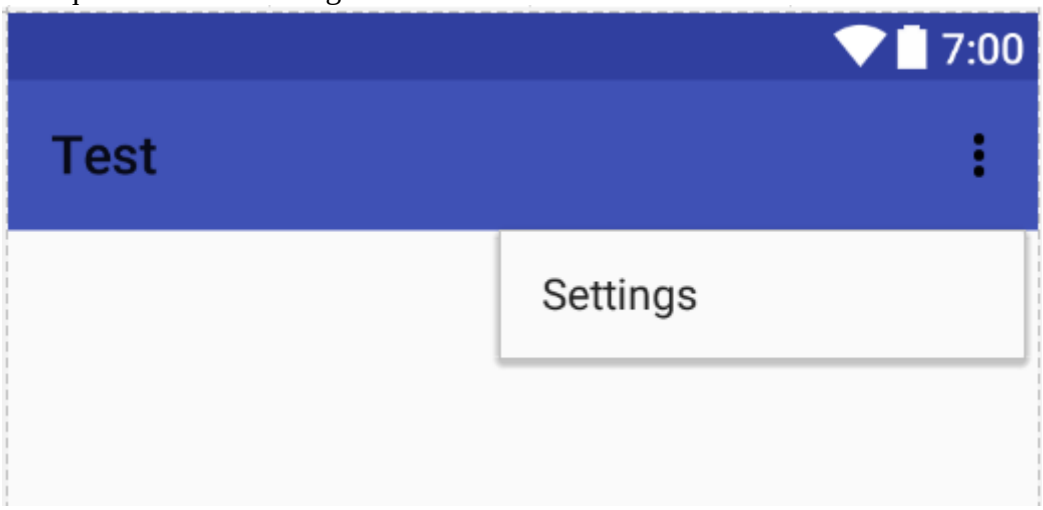
```
<menu xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto" >
```

Puis ajouter ce code

```
<item
    android:id="@+id/action_settings"
    android:title="Settings"
    android:orderInCategory="100"
    app:showAsAction="never" ></item>
```

Avant d'expliquer le contenu du fichier, est-ce que quelqu'un sait de quel menu nous sommes en train de parler ? Une petite précision s'impose.
Si vous avez un smartphone Android, vous avez pu constater que les boutons d'actions du style envoyer, partager, modifier etc… Se trouvent dans une barre se situant en haut de votre écran. Cette barre se nomme l'**ActionBar**. C'est dans celle-ci que nos fichiers menu vont s'afficher. Votre ActionBar devrait ressembler à ça

La question que tout le monde se pose, à quoi servent les trois petits points ? En langage Android on appelle cet element l'**overflow,** il contient pour le moment notre menu. Cliquez dessus, on voit un item qui se nomme Settings.



Je peux à présent retourner vous expliquez le contenu de notre fichier *« menu_main.xml »*.
Intéressons nous à certains attributs d'un item :

- android:title, permet de définir le nom de l'item qui sera affiché dans notre menu
- app:showAsAction, permet de spécifier comment et quand l'item apparaît hors de l'overflow, vous avez pour le moment **never** qui signifie que l'item apparaîtra toujours dans l'overflow, vous pouvez essayez de mettre **always** et voir le changement.
- android:icon, permet de mettre un icône correspondant à l'action (le texte sera affiché si l'item se trouve dans l'overflow)

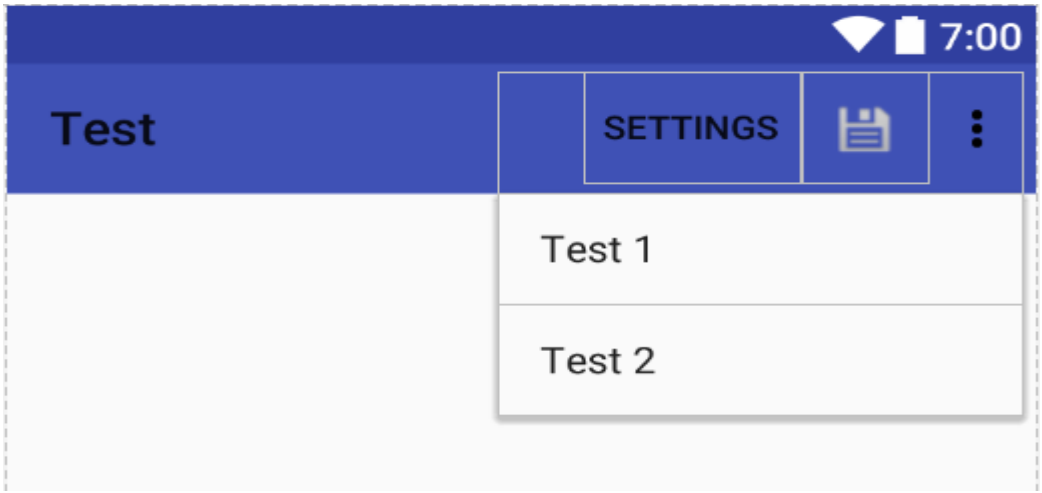
Réalisons maintenant un menu un peu plus évolué pour mettre en oeuvre nos nouvelles connaissances, voici ce que doit contenir le menu :

- L'item settings, n'appartenant plus à l'overflow
- Un item launcher, représenté par une icône (l'image ic_launcher fera l'affaire) et n'appartenant pas à l'overflow
- Et deux items test1,test2 affiché dans l'overflow

Je vous met ce que votre fichier *« menu_main.xml »* doit contenir et le résultat que vous devez obtenir dans l'application. Essayez par vous même, ce sera mieux pour votre compréhension.

```
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto" >
    <item
        android:id="@+id/action_settings"
        android:title="settings"
        app:showAsAction="always" />
```

```
    <item
        android:title="launcher"
        android:icon="@drawable/ic_menu_save"
        app:showAsAction="always" />
    <item
        android:title="Test 1"
        app:showAsAction="never" />
    <item
        android:title="Test 2"
        app:showAsAction="never" />
    </menu>
```



Maintenant que nous avons crée notre menu il ne reste que lui attacher à notre activity, pour cela il faut redéfinir une fonction dans la classe java :

```
@Override
public boolean onCreateOptionsMenu(Menu menu){
    getMenuInflater().inflate(R.menu.menu_main,menu);
    return true;
}
```

Menu_main est le nom de mon menu.

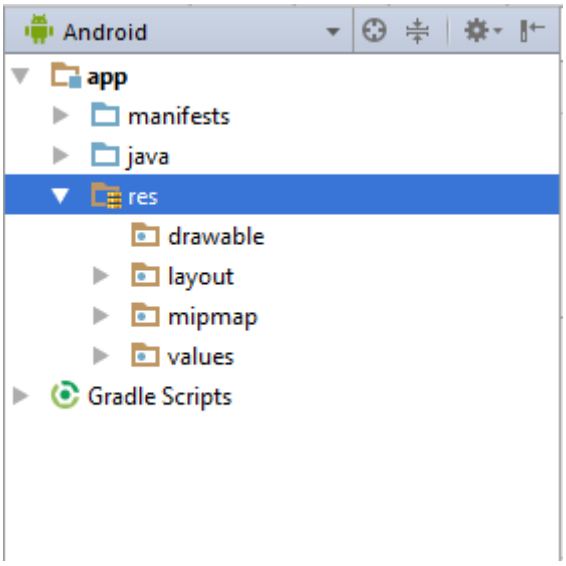
Peut être maintenant vous posez la question comment je vais savoir sur quel item l'utilisateur a cliqué ? Pour cela c'est simple, il faut redéfinir une autre fonction dans la class java :

```
@Override
public boolean onOptionsItemSelected(MenuItem item)
{
    int id=item.getItemId();
    if (id==R.id.action_settings)
        Toast.makeText(this, "settings is clicked", Toast.LENGTH_SHORT).show();
    if (id==R.id.test2)
        Toast.makeText(this, "test 2 is clicked", Toast.LENGTH_SHORT).show();
    return  true;
}
```

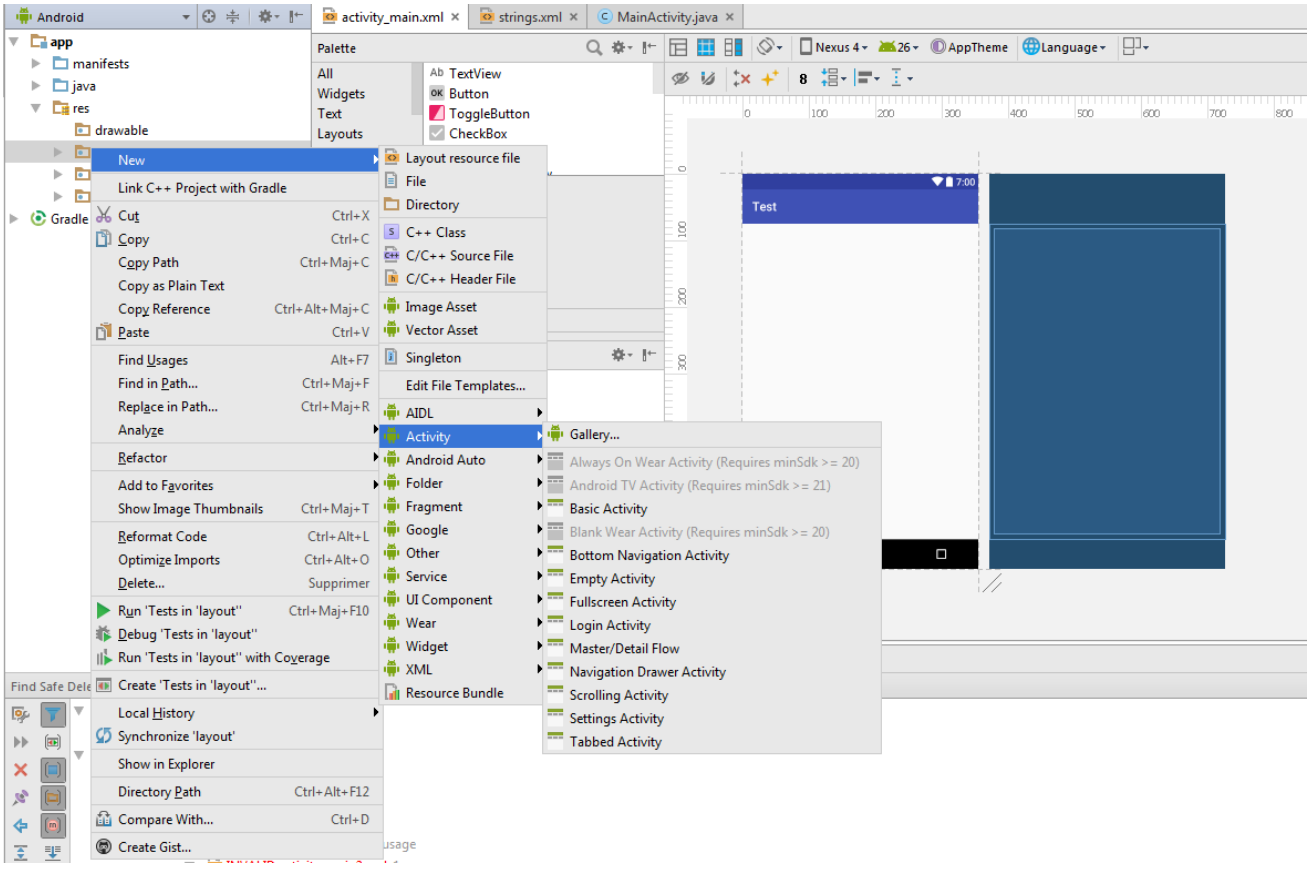
Passage de donnée d'une activity à une autre (Intent)

AJOUTER UNE NOUVELLE ACTIVITÉ AU PROJET EXISTANT

Dans votre projet d'application principal, allez dans votre dossier res.

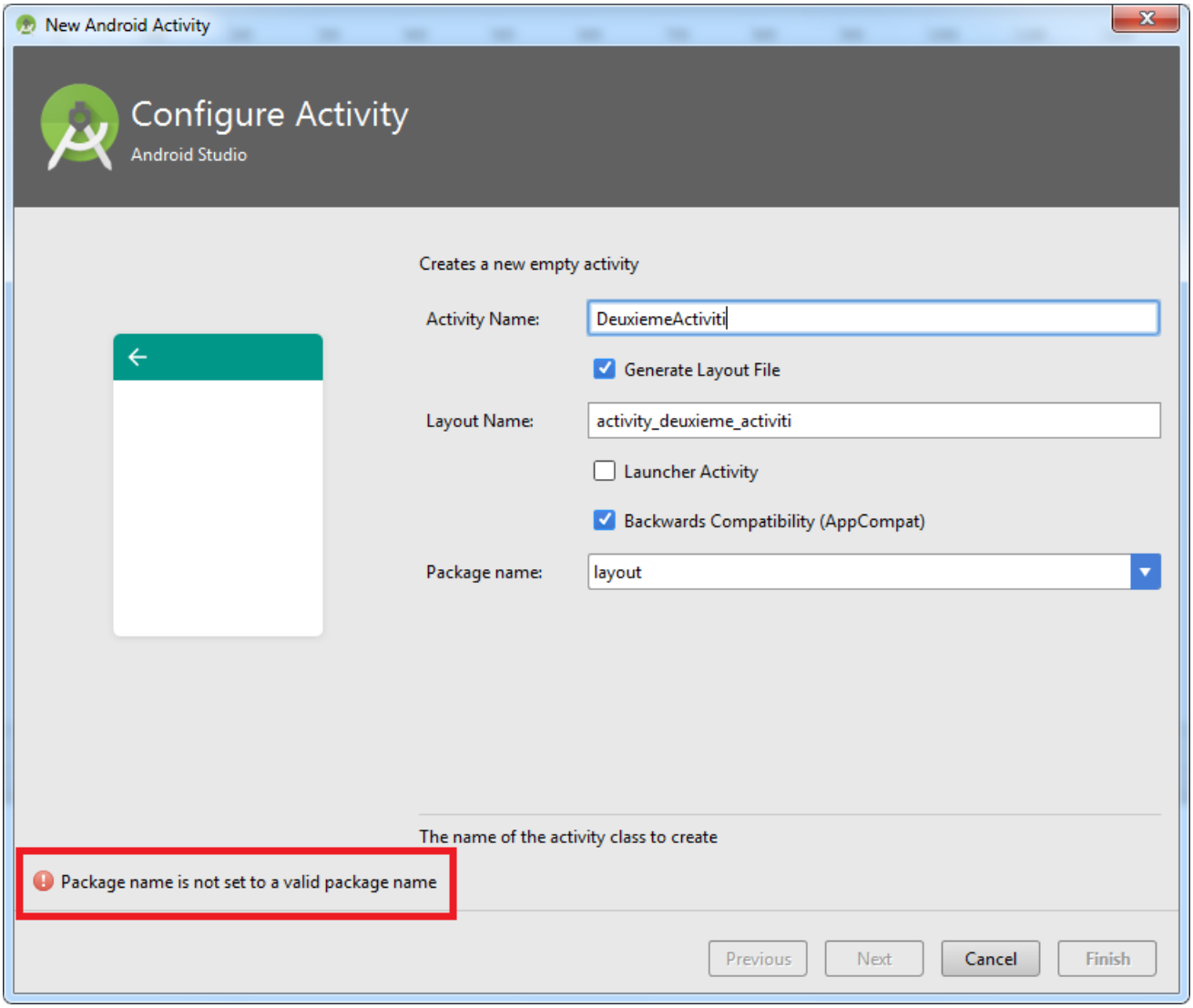


Clic droit sur le dossier layout puis new puis activity puis choisir le type activity que vous voulez.

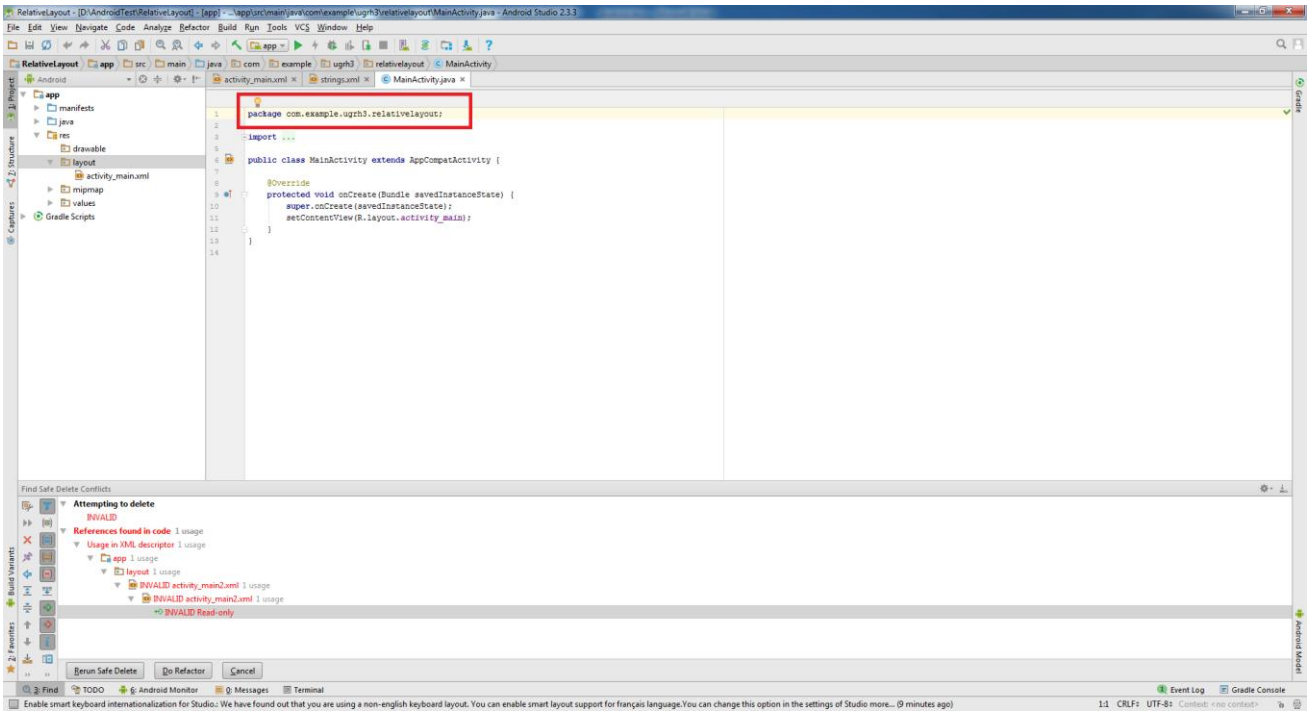


Dans notre exemple on va choisir « Empty Activity » qui est une activité vide.

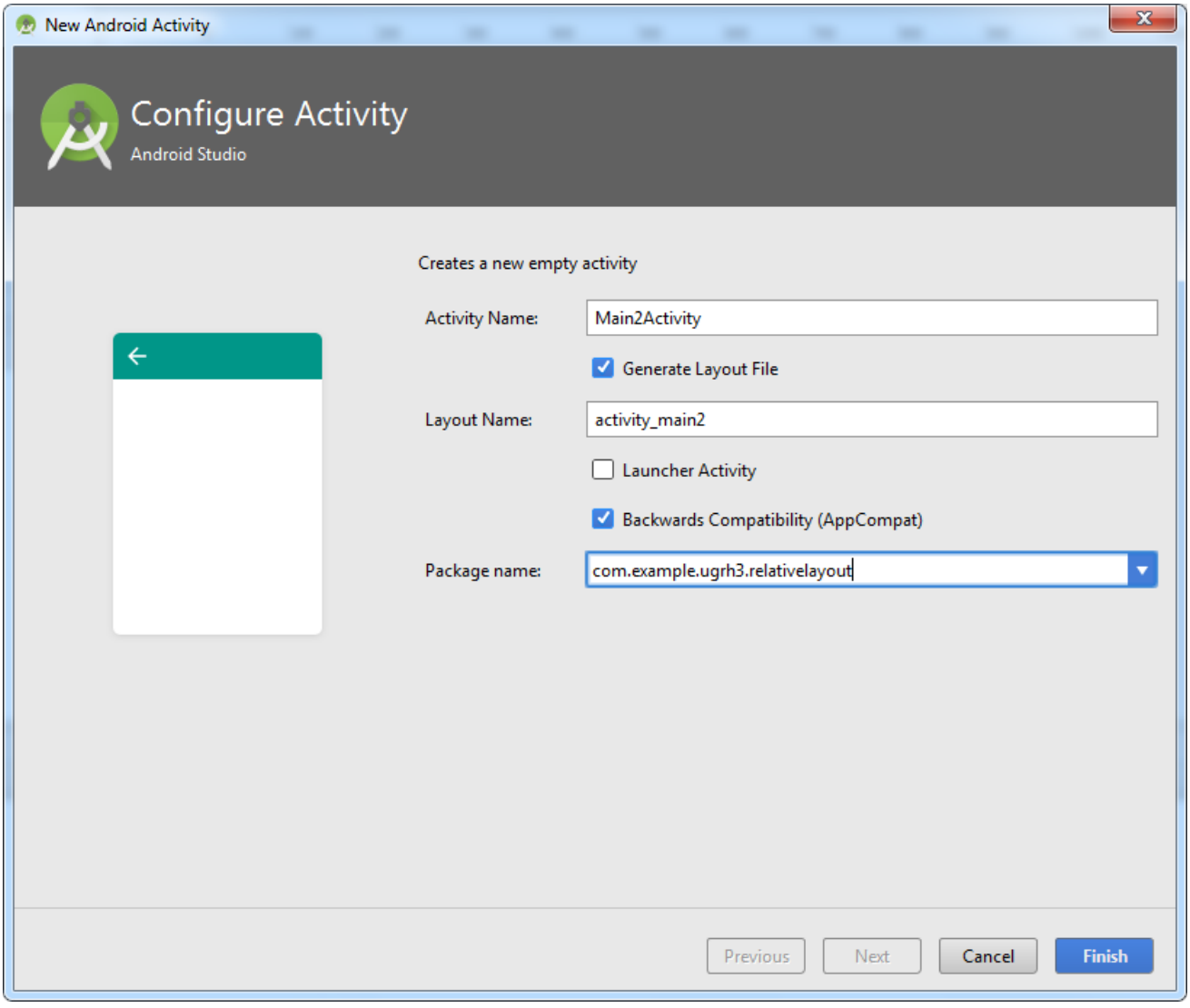
Tapez le nom de l'activity



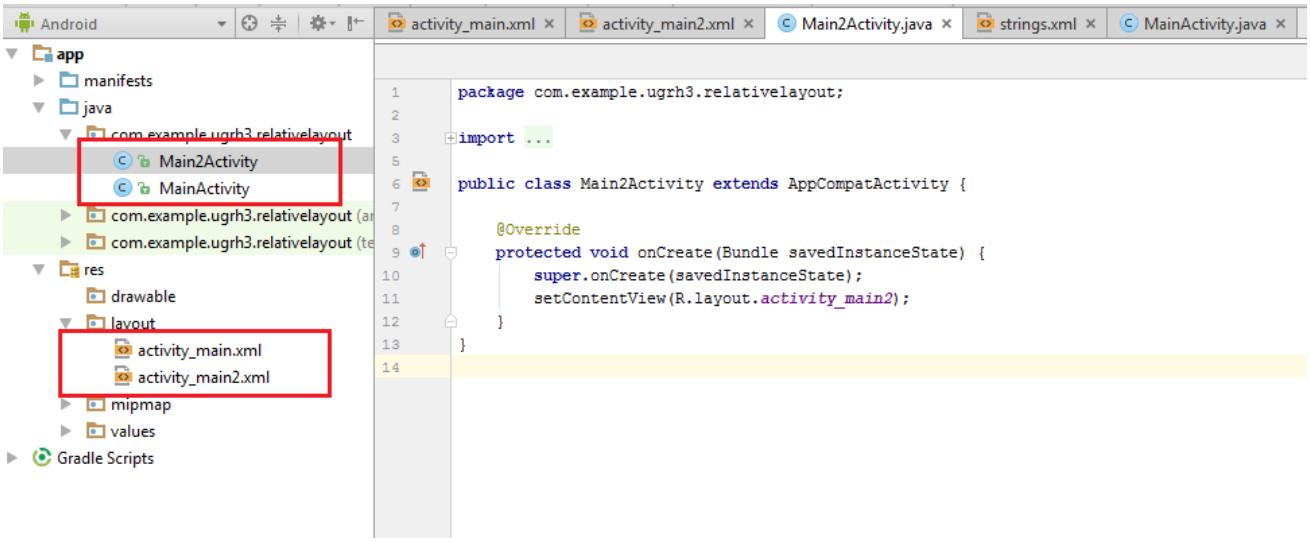
si vous trouvez un probleme du Package veuillez copie cette ligne sans point virgule



Et coller la dans le « Package name »



pour terminé appuyer sur Finish.



Comme vous allez remarquer android studio crée une autre activity avec son class java.

PASSAGE D'UNE ACTIVITY À UNE AUTRE

Un passage d'une activity à une autre signifie un changement d'activity avec passage de données si nécessaire entre les deux activity.

Pour bien comprendre cette partie, nous allons faire un exemple.

Nous allons créer une activity de connexion et une autre activity pour afficher le login et le mot de passe de l'utilisateur.

Nous allons mettre en place le passage d'une activité à une autre, donc pour cela revenons dans la classe "**MainActivity.java**".

Le passage d'une activité à une autre se fait en plusieurs parties :

On déclare un nouveau "**OnClickListener**" sur le bouton utilisé pour passer à la seconde activité (ici le bouton "se connecter").Pour le passage d'une activité à une autre c'est facile.

- On crée un nouveau **Intent** (Les Intents permettent de communiquer entre les différentes activités de notre application, mais aussi du téléphone. Ils sont en quelque sorte le « messager » pour lancer une activité. Ainsi une activité peut en lancer une autre soit en passant un intent vide, soit en y passant des paramètres.)
- Le premier argument représente le contexte (facilement récupérable à l'aide de l'activité de départ) et le second représente l'activité d'arrivée.
- On utilise la méthode "**startActivity**" avec comme argument l'intent crée précédemment.

Ce qui donnera pour la méthode **OnCreate** :

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    final Button loginButton = (Button) findViewById(R.id.Connexion);
    loginButton.setOnClickListener(new View.OnClickListener() {

        @Override
        public void onClick(View v) {
            Intent intent = new Intent(MainActivity.this, Main2Activity.class);
            startActivity(intent);
        }
```

```
    });
}
```

Si vous tester le code actuel, le passage de la première activité à la seconde s'effectue correctement mais les données utilisateur ne sont pas transmise.

PASSAGE DE DONNÉES D'UNE ACTIVITY À UNE AUTRE

Utilisation des Intent

Premier Activity

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="368dp"
    android:layout_height="495dp"
    android:orientation="vertical"
    android:paddingTop="@dimen/layout_padding_top"
    android:gravity="top|center"
  >
    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/email"
        android:textColor="@color/noir"
        android:textSize="@dimen/big_text_size"
        android:textStyle="bold" />

<EditText
    android:id="@+id/txt_login"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:gravity="center"
    android:hint="@string/hint_mail"
    android:textColor="@color/noir"
    android:textSize="@dimen/big_text_size" />
    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:paddingTop="@dimen/normal_padding"
        android:text="@string/password"
        android:textColor="@color/noir"
        android:textSize="@dimen/big_text_size"
        android:textStyle="bold" />

<EditText
    android:id="@+id/txt_pass"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:gravity="center"
    android:hint="@string/hint_pass"

    android:inputType="textPassword"
    android:textColor="@color/noir"
    android:textSize="@dimen/big_text_size" />
    <Button
        android:id="@+id/Connexion"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="se connecter"
```

```
        android:onClick="fctPassage"/>
</LinearLayout>
```

Deuxième activity

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="368dp"
    android:layout_height="495dp"
    android:orientation="vertical"
    tools:layout_editor_absoluteX="8dp"
    tools:layout_editor_absoluteY="8dp">

    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:orientation="horizontal">

        <TextView
            android:id="@+id/textView5"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_weight="1"
            android:text="Login  :" />

        <TextView
            android:id="@+id/AffichageMail"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_weight="1"
            android:text="TextView" />
    </LinearLayout>

    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:orientation="horizontal">

        <TextView
            android:id="@+id/textView3"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_weight="1"
            android:text="Mot de passe :" />

        <TextView
            android:id="@+id/AffichagePass"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_weight="1"
            android:text="TextView" />
    </LinearLayout>
</LinearLayout>
```

Pour transmettre ces données, il suffit de les ajouter à l'intent crée précédemment en leur spécifiant une clé permettant de les identifier. Ce qui donnera :

```
final String EXTRA_LOGIN = "Login";
final String EXTRA_PASS = "pass";
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    final EditText login = (EditText)findViewById(R.id.txt_login);
    final EditText pass = (EditText)findViewById(R.id.txt_pass);
    final Button btnLancerActivity=(Button)findViewById(R.id.Connexion);
    btnLancerActivity.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View view) {
            Intent intent = new Intent(MainActivity.this, Main2Activity.class);
            intent.putExtra(EXTRA_LOGIN, login.getText().toString());
            intent.putExtra(EXTRA_PASS, pass.getText().toString());
            startActivity(intent);
        }
    });
}
```

- On déclare les deux identifiants utiles pour le passage de données (EXTRA_LOGIN et EXTRA_ PASS)
- On initialise les composants utilises (Zones de saisies pour le login et password)
- Lors du clic sur le bouton, on récupère les textes saisies par l'utilisateur (**getText().toString**)
- Pour finir, on associe ces valeurs (textes) avec l'intent déclaré précédemment (méthode **putExtra**)

Maintenant, il faut récupérer les données transmises par l'intent et les assigner aux zones de texte, ce qui donnera :

```
final String EXTRA_LOGIN = "Login";
final String EXTRA_PASS = "pass";

@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main2);

        Intent intent = getIntent();
        TextView loginActiviti2 = (TextView)findViewById(R.id.AffichageMail);
        TextView passActivity2 = (TextView)findViewById(R.id.AffichagePass);

        if (intent != null) {
            loginActiviti2.setText(intent.getStringExtra(EXTRA_LOGIN));
            passActivity2.setText(intent.getStringExtra(EXTRA_PASS));
        }
}
```

- Il faut récupérer l'intent passé à la nouvelle activité à l'aide de la méthode getIntent et vérifier que ce dernier n'est pas null
- Initialiser les deux zones de textes servant à afficher les informations utilisateurs (login / password)
- Récupérer les deux informations à l'aide de leurs clés et de la méthode **getStringExtra**. Il faut utiliser la méthode **getTypeExtra** ou Type correspond au type de la donnée passé
- Lier les textes récupérés aux TextView à l'aide de la méthode **setText**

Utilisation des Bundle

On peut passer des données d'une activity à une autre on utilisons les Bundle, le résultat est le même mais la façon se diffère.

Pour transmettre les données :

```
final String EXTRA_LOGIN = "Login";
final String EXTRA_PASS = "pass";
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
  final   EditText login = (EditText)findViewById(R.id.txt_login);
    final EditText pass = (EditText)findViewById(R.id.txt_pass);
    final Button btnLancerActivity=(Button)findViewById(R.id.Connexion);
   btnLancerActivity.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View view) {
            Intent intent = new Intent(MainActivity.this, Main2Activity.class);
            Bundle bdl=new Bundle();
            bdl.putString(EXTRA_LOGIN,login.getText().toString());
            bdl.putString(EXTRA_PASS, pass.getText().toString());
            intent.putExtras(bdl);
            startActivity(intent);
        }
    });
}
```

Et pour la récupération des données :

```
final String EXTRA_LOGIN = "Login";
final String EXTRA_PASS = "pass";
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main2);
    Bundle extras=getIntent().getExtras();
        TextView loginActiviti2 = (TextView)findViewById(R.id.AffichageMail);
        TextView passActivity2 = (TextView)findViewById(R.id.AffichagePass);
        if (extras != null)
        {
            loginActiviti2.setText(extras.getString(EXTRA_LOGIN));
            passActivity2.setText(extras.getString(EXTRA_PASS));
        }
}
```

Préférences partagées

Utile voire indispensable pour un grand nombre d'applications, pouvoir enregistrer les paramètres des utilisateurs leur permettra de paramétrer de manière minutieuse vos applications de manière à ce qu'ils obtiennent le rendu qui convienne le mieux à leurs exigences.

LES DONNÉES PARTAGÉES

Le point de départ de la manipulation des préférences partagées est la classe SharedPreferences. Elle possède des méthodes permettant d'enregistrer et récupérer des paires de type identifiant-valeur pour les types de données primitifs, comme les entiers ou les chaînes de caractères. L'avantage réel étant bien sûr que ces données sont conservées même si l'application est arrêtée ou tuée. Ces préférences sont de plus accessibles depuis plusieurs composants au sein d'une même application.

Il existe trois façons d'avoir accès aux SharedPreferences :

- La plus simple est d'utiliser la méthode statique SharedPreferences PreferenceManager.getDefaultSharedPreferences(Context context).
- Si vous désirez utiliser un fichier standard par activité, alors vous pourrez utiliser la méthode SharedPreferences getPreferences(int mode).
- En revanche, si vous avez besoin de plusieurs fichiers que vous identifierez par leur nom, alors utilisez SharedPreferences getSharedPreferences (String name, int mode) où name sera le nom du fichier.

En ce qui concerne le second paramètre, mode, il peut prendre trois valeurs :

- Context.MODE_PRIVATE, pour que le fichier créé ne soit accessible que par l'application qui l'a créé.
- Context.MODE_WORLD_READABLE, pour que le fichier créé puisse être lu par n'importe quelle application.
- Context.MODE_WORLD_WRITEABLE, pour que le fichier créé puisse être lu *et* modifié par n'importe quelle application.

Afin d'ajouter ou de modifier des couples dans un SharedPreferences, il faut utiliser un objet de type SharedPreference.Editor. Il est possible de récupérer cet objet en utilisant la méthode SharedPreferences.Editor edit() sur un SharedPreferences.

Si vous souhaitez ajouter des informations, utilisez une méthode du genre SharedPreferences.Editor putX(String key, X value) avec X le type de l'objet, key l'identifiant et value la valeur associée. Il vous faut ensuite impérativement valider vos changements avec la méthode boolean commit().

NB : Les préférences partagées ne fonctionnent qu'avec les objets de type boolean, float, int, long et String.

Par exemple, pour conserver la couleur préférée de l'utilisateur, il n'est pas possible d'utiliser la classe Color puisque seuls les types de base sont acceptés, alors on pourrait conserver la valeur de la couleur sous la forme d'une chaîne de caractères :

```
SharedPreferences myPreference = getSharedPreferences("test",
Context.MODE_PRIVATE);
SharedPreferences.Editor Edit = myPreference.edit();
```

```
Edit.putString("Couleur pref", "FFABB4");
Edit.commit();
```

De manière naturelle, pour récupérer une valeur, on peut utiliser la méthode X getX(String key, X defValue) avec X le type de l'objet désiré, key l'identifiant de votre valeur et defValue une valeur que vous souhaitez voir retournée au cas où il n'y ait pas de valeur associée à key :

```
String couleur = myPreference.getString("Couleur pref", "FFFFFF");
Toast.makeText(this, couleur, Toast.LENGTH_SHORT).show();
```

Si vous souhaitez supprimer une préférence, vous pouvez le faire avec SharedPreferences.Editor removeString(String key).

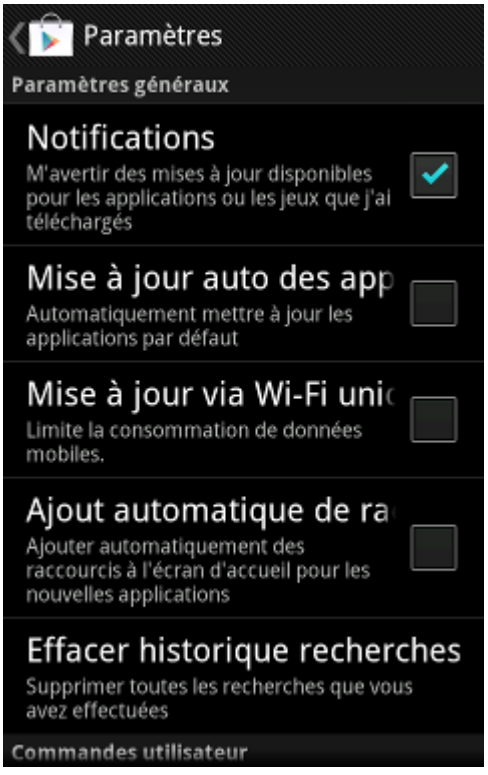
```
SharedPreferences myPreference = getSharedPreferences("test",
Context.MODE_PRIVATE);
SharedPreferences.Editor Edit = myPreference.edit();
Edit.remove("Couleur pref");
```

pour radicalement supprimer toutes les préférences, il existe aussi SharedPreferences.Editor clear().

```
SharedPreferences myPreference = getSharedPreferences("test",
Context.MODE_PRIVATE);
SharedPreferences.Editor Edit = myPreference.edit();
Edit.clear();
```

Des préférences prêtes à l'emploi

Pour enregistrer vos préférences, vous pouvez très bien proposer une activité qui permet d'insérer différents paramètres (voir figure suivante). Si vous voulez développer vous-mêmes l'activité, grand bien vous fasse, ça fera des révisions, mais sachez qu'il existe un framework pour vous aider. Vous en avez sûrement déjà vus dans d'autres applications. C'est d'ailleurs un énorme avantage d'avoir toujours un écran similaire entre les applications pour la sélection des préférences.



Ce type d'activités s'appelle les « `PreferenceActivity` ». Un plus indéniable ici est que chaque couple identifiant/valeur est créé automatiquement et sera récupéré automatiquement, d'où un gain de temps énorme dans la programmation. La création se fait en plusieurs étapes, nous allons voir la première, qui consiste à établir une interface graphique en XML.

Étape 1 : en XML

La racine de ce fichier doit être un `PreferenceScreen`.

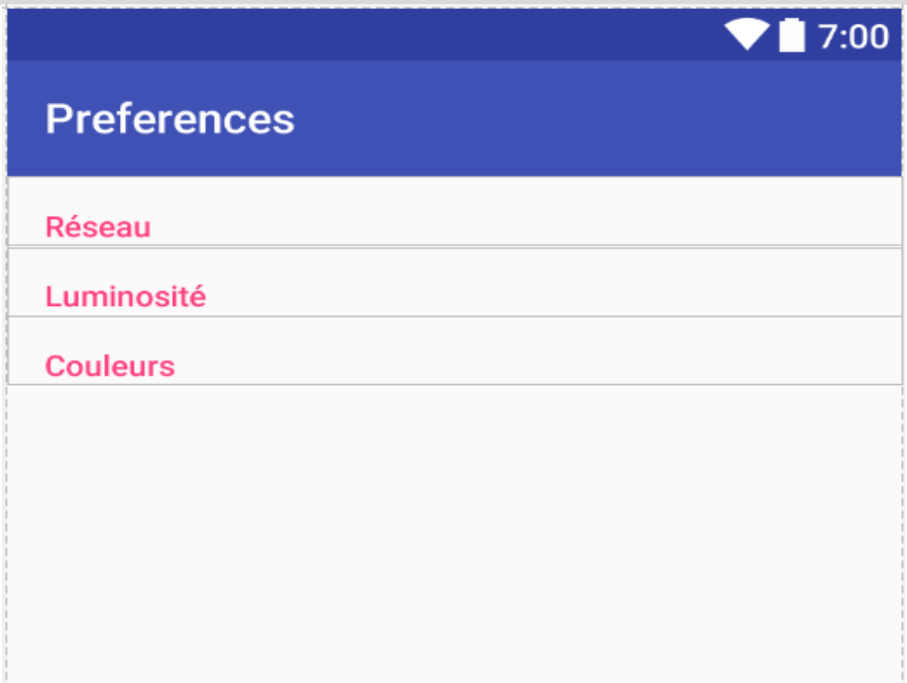
Comme ce n'est pas vraiment un layout, on le définit souvent dans `/xml/preference.xml`.
Pour ajouter ce Fichier :

1. Click droite dur dossier res
2. Choisir new puis Android ressource file
3. Dans ressource type choisir XML
4. Donner un nom au fichier et valider

Tout d'abord, il est possible de désigner des catégories de préférences. Une pour les préférences destinées à internet par exemple, une autre pour les préférences esthétiques, etc. On peut ajouter des préférences avec le nœud `PreferenceCategory`. Ce nœud est un layout, il peut donc contenir d'autre vues. Il ne peut prendre qu'un seul attribut, `android:title`, pour préciser le texte qu'il affichera.

Ainsi le code suivant :

```
<?xml version="1.0" encoding="utf-8"?>
<PreferenceScreen xmlns:android="http://schemas.android.com/apk/res/android" >
<PreferenceCategory android:title="Réseau">

</PreferenceCategory>
<PreferenceCategory android:title="Luminosité">

</PreferenceCategory>
<PreferenceCategory android:title="Couleurs">

</PreferenceCategory>
</PreferenceScreen>
```



Nous avons nos catégories, il nous faut maintenant insérer des préférences ! Ces trois vues ont cinq attributs en commun :

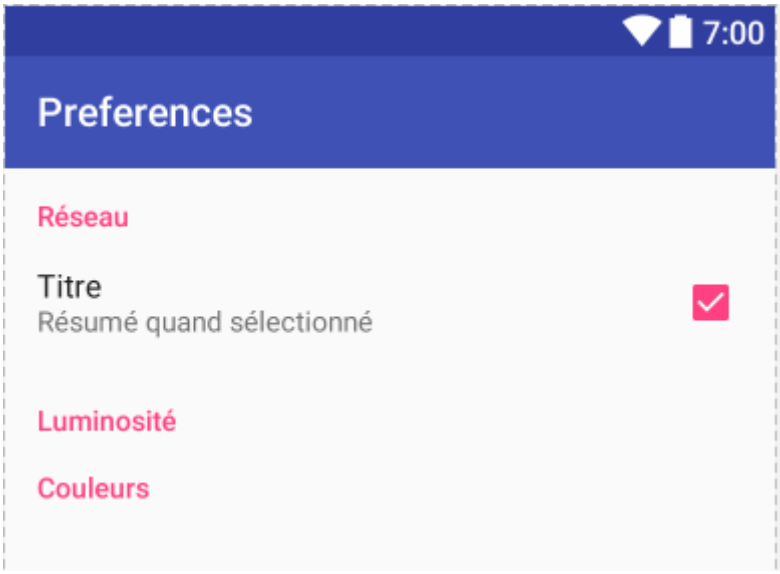
- android:key est l'identifiant de la préférence partagée. C'est un attribut *indispensable*, ne l'oubliez *jamais*.
- android:title est le titre principal de la préférence.
- android:summary est un texte secondaire qui peut être plus long et qui explique mieux ce que veut dire cette préférence.
- Utilisez android:dependency, si vous voulez lier votre préférence à une autre activité. Il faut y insérer l'identifiant android:key de la préférence dont on dépend.
- android:defaultValue est la valeur par défaut de cette préférence.

Il existe au moins trois types de préférences, la première étant une case à cocher avec CheckBoxPreference, avec true ou false comme valeur (soit la case est cochée, soit elle ne l'est pas).

À la place du résumé standard, vous pouvez déclarer un résumé qui ne s'affiche que quand la case est cochée, android:summaryOn, ou uniquement quand la case est décochée, android:summaryOff.

```
<CheckBoxPreference
    android:key="checkBoxPref"
    android:title="Titre"
    android:summaryOn="Résumé quand sélectionné"
    android:summaryOff="Résumé quand pas sélectionné"
    android:defaultValue="false"/>
```

Ce qui donne la figure suivante.

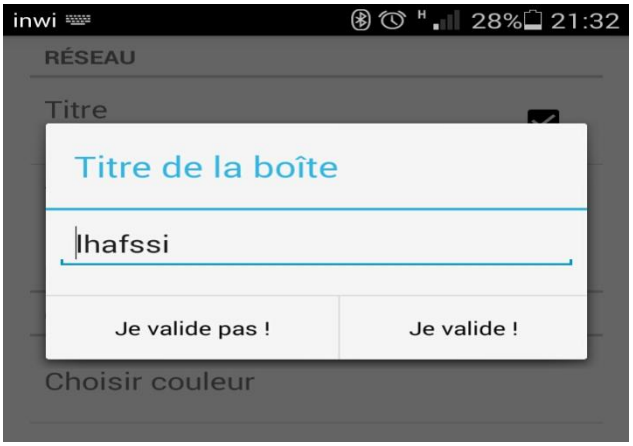


Le deuxième type de préférences consiste à permettre à l'utilisateur d'insérer du texte avec EditTextPreference, qui ouvre une boîte de dialogue contenant un EditText permettant à l'utilisateur d'insérer du texte. On retrouve des Par exemple, android:dialogTitle permet de définir le texte de la boîte de dialogue, alors que android:negativeButtonText et android:positiveButtonText permettent respectivement de définir le texte du bouton à droite et celui du bouton à gauche dans la boîte de dialogue.

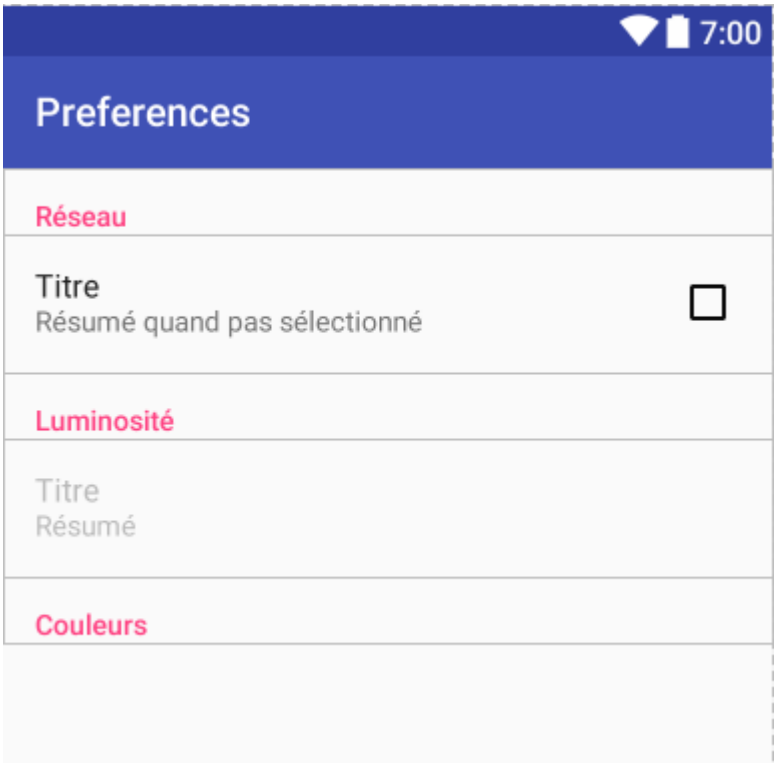
```
<EditTextPreference
    android:key="editTextPref"
    android:dialogTitle="Titre de la boîte"
```

```
    android:positiveButtonText="Je valide !"
    android:negativeButtonText="Je valide pas !"
    android:title="Titre"
    android:summary="Résumé"
    android:dependency="checkBoxPref" />
```

Ce qui donne la figure suivante.



De plus, comme vous avez pu le voir, ce paramètre est lié à la CheckBoxPreference précédente par l'attribut android:dependency="checkBoxPref", ce qui fait qu'il ne sera accessible que si la case à cocher de checkBoxPref est activée, comme à la figure suivante.



De plus, comme nous l'avons fait avec les autres boîtes de dialogue, il est possible d'imposer un layout à l'aide de l'attribut android:dialogLayout.

Le troisième type de préférences est un choix dans une liste d'options avec ListPreference. Dans cette préférence, on différencie ce qui est affiché de ce qui est réel. Pratique pour traduire son application en plusieurs langues ! Encore une fois, il est possible d'utiliser les attributs android:dialogTitle, android:negativeButtonText et android:positiveButtonText. Les données de la liste que lira l'utilisateur sont à présenter dans l'attribut android:entries, alors que les données qui seront

enregistrées sont à indiquer dans l'attribut android:entryValues. La manière la plus facile de remplir ces attributs se fait à l'aide d'une ressource de type array, par exemple pour la liste des couleurs :

```
<resources>
    <string name="app_name">Preferences</string>
    <array name="liste_couleurs_fr">
        <item>Bleu</item>
        <item>Rouge</item>
        <item>Vert</item>
    </array>
    <array name="liste_couleurs">
        <item>blue</item>
        <item>red</item>
        <item>green</item>
    </array>
</resources>
```

Qu'on peut ensuite fournir aux attributs susnommés :

```
<ListPreference
    android:key="listPref"
    android:dialogTitle="Choisissez une couleur"
    android:entries="@array/liste_couleurs_fr"
    android:entryValues="@array/liste_couleurs"
    android:title="Choisir couleur" />
```

Ce qui donne la figure suivante.

Étape 2 : dans le Manifest

Pour recevoir cette nouvelle interface graphique, nous avons besoin d'une activité. Il nous faut donc la déclarer dans le Manifest si on veut pouvoir y accéder avec les intents. Cette activité sera déclarée comme n'importe quelle activité :

```
<activity
    android:name=".PreferenceActivityExample"
    android:label="@string/app_name" >
</activity>
```

.PreferenceActivityExample : Classe java que vous devez le créé.

Étape 3 : en Java

Notre activité sera en fait de type PreferenceActivity. On peut la traiter comme une activité classique, sauf qu'au lieu de lui assigner une interface graphique avec setContentView, on utilise void addPreferencesFromResource(int preferencesResId) en lui assignant notre layout :

```
public class PreferenceActivityExample extends PreferenceActivity {
    @Override

    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        addPreferencesFromResource(R.xml.preference);

}}
```

Il ne reste maintenant qu'ajouter le code pour lancer cette Activity de configuration :

Code à ajouter dans la classe java de l'activity main

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

    Button button =(Button) findViewById(R.id.prefs);
    button.setOnClickListener( new View.OnClickListener() {

        @Override
        public void onClick(View v) {
            Intent go_to_prefs = new Intent(getApplicationContext(),
                    PreferenceActivityExample.class);
            startActivity(go_to_prefs);

        }
    });
}
```

Lire et Ecrire dans des fichiers

Tous comme les autres langages de programmation, on peut sous android lire et écrire dans fichier « txt » par exemples.

Cette partis est en java donc je vais vous montrer comment faire sans expliquer puisque c'est pas du Android.

LIRE

```
public void ReadSettings(Context context)
{
    try {
        FileInputStream input =
getApplicationContext().openFileInput("settings.txt");
        InputStreamReader isr2 = new InputStreamReader(input);
        BufferedReader reader = new BufferedReader(isr2);
        String ligne;
        ligne="";

        while(reader.ready())
        {

                ligne += reader.readLine();;
                ligne +="\n";



        }
        input.close();
        isr2.close();
        TextView lbl=(TextView) findViewById(R.id.textView);

        lbl.setText(ligne);
    }catch (Exception e){
        Toast.makeText(getApplicationContext(),e.getMessage()
, Toast.LENGTH_SHORT).show();
    }
}
```

Cette procédure lit un fichier texte nommé **settings.txt** ligne par ligne et affiche le contenu de fichier dans une TextView.

Vous pouvez faire un filtre dans la boucle while si vous voulez récupérer des lignes précises.

ECRIRE

```
public void WriteSettings(Context context, String data){
    FileOutputStream fOut = null;
    OutputStreamWriter osw = null;

    try{
        fOut = context.openFileOutput("settings.txt",MODE_APPEND);
        osw = new OutputStreamWriter(fOut);
        osw.write(data);
        osw.write("\n");
        osw.flush();
        //popup surgissant pour le résultat
        Toast.makeText(context, "Contenu ajouté",Toast.LENGTH_SHORT).show();
```

```
    }
    catch (Exception e) {
        Toast.makeText(context, e.getMessage(),Toast.LENGTH_SHORT).show();
    }
    finally {
        try {
            osw.close();
            fOut.close();
        } catch (IOException e) {
            Toast.makeText(context, e.getMessage(),Toast.LENGTH_SHORT).show();
        }
    }
}
```

cette procédure ouvre un fichier en mode ajout. Et écris ce qui est dans l' EditText dans le fichier **settings.txt .**

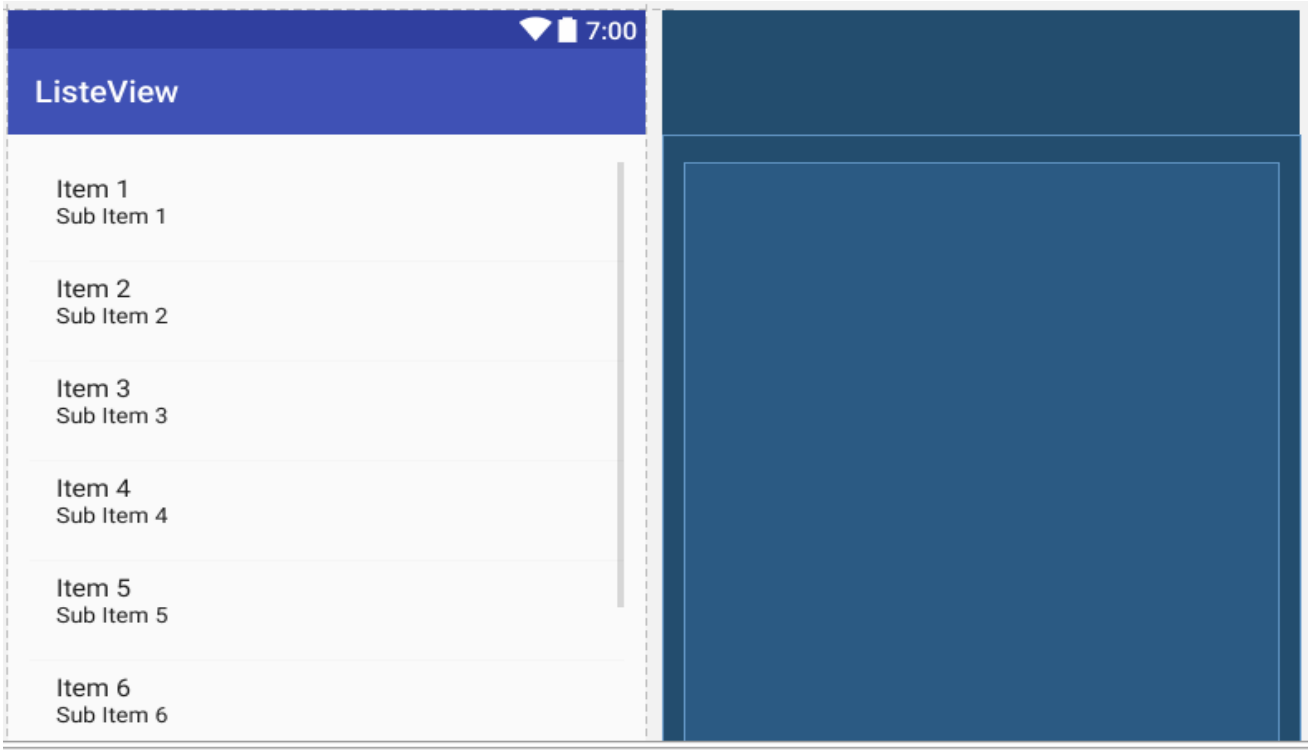
ListView

AFFICHAGE SIMPLE

Sous Android, le composant principal pour afficher et gérer une liste de données est le composant ListView. Par défaut ce composant affiche une liste de chaines de caractères. Selon les besoins, il peut être nécessaire d'afficher plus d'informations sur chaque ligne de la liste, ou même de venir effectuer des opérations particulières suite à des actions sur les lignes.

Dans un premier temps nous allons voir comment afficher une liste toute simple ne contenant que des chaines de caractères, puis nous verrons comment personnaliser l'affichage des items de la liste et enfin comment ajouter des actions sur les items.

Pour cela crée un projet et insérer dans votre layout une ListView et donner un Id votre ListView dans noter cas id=List1



Maintenant que notre layout contient une liste et que notre activité sait la gérer, on peut donc remplir celle-ci. Pour ce faire, nous allons tout d'abord initialiser un tableau de chaines de caractères de la manière suivante :

```
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main2);
     String[] mStrings = {
            "AAAAAAAA", "BBBBBBBB", "CCCCCCCC", "DDDDDDDD", "EEEEEEEE",
            "FFFFFFFF", "GGGGGGGG", "HHHHHHHH", "IIIIIIII", "JJJJJJJJ",
            "KKKKKKKK", "LLLLLLLL", "MMMMMMMM", "NNNNNNNN", "OOOOOOOO",
            "PPPPPPPP", "QQQQQQQQ", "RRRRRRRR", "SSSSSSSS", "TTTTTTTT",
            "UUUUUUUU", "VVVVVVVV", "WWWWWWWW", "XXXXXXXX", "YYYYYYYY",
            "ZZZZZZZZ"};

}
```

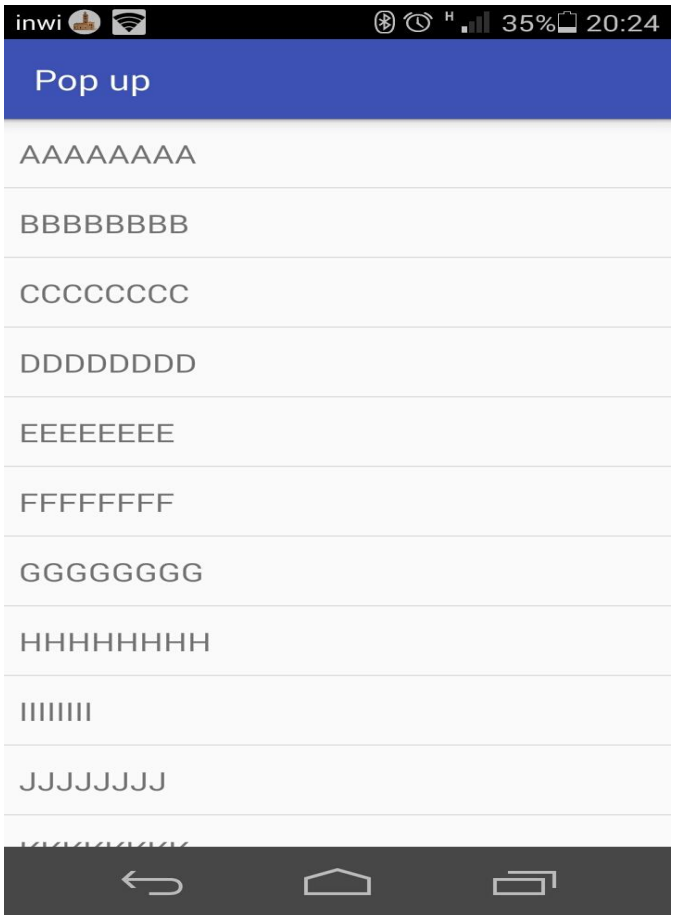
Ici l'initialisation du tableau est statique, mais on peut très bien imaginer que les informations contenues dans ce tableau proviennent par exemple d'un webservice, d'un fichier XML ou d'une base de données, etc.

Il ne nous reste plus qu'à afficher ces données dans notre liste. Pour ce faire, nous allons utiliser les fonctionnalités proposées par ListActivity :

```
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main2);
     String[] mStrings = {
            "AAAAAAAA", "BBBBBBBB", "CCCCCCCC", "DDDDDDDD", "EEEEEEEE",
            "FFFFFFFF", "GGGGGGGG", "HHHHHHHH", "IIIIIIII", "JJJJJJJJ",
            "KKKKKKKK", "LLLLLLLL", "MMMMMMMM", "NNNNNNNN", "OOOOOOOO",
            "PPPPPPPP", "QQQQQQQQ", "RRRRRRRR", "SSSSSSSS", "TTTTTTTT",
            "UUUUUUUU", "VVVVVVVV", "WWWWWWWW", "XXXXXXXX", "YYYYYYYY",
            "ZZZZZZZZ"};
    ListView lst=(ListView) findViewById(R.id.List1);
    ArrayAdapter<String> adapter = new ArrayAdapter<String>(this,
        android.R.layout.simple_list_item_1, mStrings);
    lst.setAdapter(adapter);

}
```

L'adapter permet de gérer les données et de réaliser le mapping relationnel entre les données et l'IHM(interface homme-machine). Le premier paramètre (this) permet d'identifier le contexte dans lequel notre adapter va fonctionner. Le deuxième paramètre "android.R.layout.simple_list_item_1" permet d'indiquer la présentation utilisée pour les items de notre liste. Ici il s'agit d'une présentation simple pour les chaines de caractères incluse dans le SDK.

L'appel de cette méthode permet donc d'afficher notre liste. Ainsi si nous exécutons notre projet, nous obtenons ceci :



PERSONNALISATION DES ITEMS

Dans cette partie nous allons voir comment personnaliser notre liste, et surtout comment personnaliser la présentation des items de la liste. Le thème de cette exercice sera d'afficher une liste de personne. Une personne aura un nom, un prénom, un genre (Masculin / Féminin).

Apres la création de l'activity et l'insertion de la LsitView , nous allons créer une classe pour représenter les personnes. Cette classe contient trois propriétés : "nom", "prenom" et "genre". Afin de simplifier les développements et l'exemple, on rajoute une méthode static dans cette classe pour remplir et renvoyer une liste de personnes. Le code de la classe donne donc :

```
public class Personne {
    public final static int MASCULIN = 1;
    public final static int FEMININ = 2;

    public String nom;
    public String prenom;
    public int genre;

    public Personne(String aNom, String aPrenom, int aGenre) {
        nom = aNom;
        prenom = aPrenom;
        genre = aGenre;
    }

    /**
     * Initialise une liste de personnes
     * @return une liste de "Personne"
```

```
     */
    public static ArrayList<Personne> getAListOfPersonne() {
        ArrayList<Personne> listPers = new ArrayList<Personne>();

        listPers.add(new Personne("Nom1", "Prenom1", FEMININ));
        listPers.add(new Personne("Nom2", "Prenom2", MASCULIN));
        listPers.add(new Personne("Nom3", "Prenom3", MASCULIN));
        listPers.add(new Personne("Nom4", "Prenom4", FEMININ));
        listPers.add(new Personne("Nom5", "Prenom5", FEMININ));
        listPers.add(new Personne("Nom6", "Prenom6", MASCULIN));
        listPers.add(new Personne("Nom7", "Prenom7", FEMININ));
        listPers.add(new Personne("Nom8", "Prenom8", MASCULIN));
        listPers.add(new Personne("Nom9", "Prenom9", MASCULIN));
        listPers.add(new Personne("Nom10", "Prenom10", FEMININ));
        listPers.add(new Personne("Nom11", "Prenom11", MASCULIN));
        listPers.add(new Personne("Nom12", "Prenom12", MASCULIN));
        listPers.add(new Personne("Nom13", "Prenom13", FEMININ));
        listPers.add(new Personne("Nom14", "Prenom14", MASCULIN));

        return listPers;
    }
}
```

Les personnes possèdent trois propriétés. Il n'est donc plus possible de passer par le layout standard pour les afficher. Nous allons donc créer un nouveau layout, et l'utiliser pour afficher les items de la liste. Ajoutez un nouveau fichier XML dans le répertoire layout de votre application (Clique droite sur le dossier layout puis New puis Layout resource file. Donner un nom au fichier et cliquer sur ok. Dans notre cas le nom sera : listepersonne)

Dans ce nouveau layout, nous allons donc ajouter deux TextView pour décrire les nom et prénom de chaque personne. Le genre de la personne sera décrit par la couleur de fond de notre item : bleu pour les garçons, rose pour les filles :). Le XML de notre item doit donc ressembler à ça :

```
<LinearLayout
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="horizontal">

    <TextView
        android:id="@+id/txtNom"
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:layout_weight="1"
        android:text="TextView" />

    <TextView
        android:id="@+id/txtPrenom"
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:layout_weight="1"
        android:text="TextView" />
</LinearLayout>
```

Il ne nous reste plus qu'à afficher la liste des personnes en utilisant notre layout. Pour ce faire, nous allons créer un objet qui se chargera de gérer le mapping entre nos données et le layout des items. Ce composant sera basé sur un Adapter. Créez une nouvelle classe : "PersonneAdapter". Faites-la hériter de "BaseAdapter". (Ne pas créer un nouveau fichier pour la classe mettez la juste avec la classe Main activity.java)

```
public class PersonneAdapter extends BaseAdapter {


    @Override
    public int getCount() {
        return 0;
    }

    @Override
    public Object getItem(int i) {
        return null;
    }

    @Override
    public long getItemId(int i) {
        return 0;
    }

    @Override
    public View getView(int i, View view, ViewGroup viewGroup) {
        return null;
    }
}
```

L'Adapter va gérer une liste de personnes et va s'occuper également de l'affichage. On va donc lui ajouter une propriété qui est la table des personnes

Avec un constructeur Notre classe devient:

```
public class PersonneAdapter extends BaseAdapter {
 ArrayList<Personne> arrayItems=new ArrayList<Personne>();
PersonneAdapter(ArrayList<Personne> Items ) {
    this.arrayItems=Items;

}
    @Override
    public int getCount() {
        return 0;
    }

    @Override
    public Object getItem(int i) {
        return null;
    }

    @Override
    public long getItemId(int i) {
        return 0;
    }

    @Override
    public View getView(int i, View view, ViewGroup viewGroup) {
        return null;
    }
}
```

Le LayoutInflater permet de parser un layout XML et de te transcoder en IHM Android. Pour respecter l'interface BaseAdapter, il nous faut spécifier la méthode "count()". Cette méthode permet de connaître le nombre d'items présent dans la liste. Dans notre cas, il faut donc renvoyer le nombre de personnes contenus dans "mListP".

```
@Override
public int getCount() {
```

```
    return arrayItems.size();
}
```

Ensuite, il y a deux méthodes pour identifier les items de la liste. Une pour connaitre l'item situé à une certaine position et l'autre pour connaitre l'identifiant d'un item en fonction de sa position.

```
@Override
public Object getItem(int position) {
    return arrayItems.get(position);
}
@Override

public long getItemId(int position) {
    return position;
}
```

Maintenant il faut surcharger la méthode pour renvoyer une "View" en fonction d'une position donnée. Cette view contiendra donc une occurrence du layout " listepersonne.xml" convenablement renseignée (avec le nom et prénom au bon endroit).

```
@Override
public View getView(int i, View view, ViewGroup viewGroup) {
    LayoutInflater linflater =getLayoutInflater();
    View view1=linflater.inflate(R.layout.listepersonne, null);

    //(2) : Récupération des TextView de notre layout
    TextView tv_Nom = (TextView)view1.findViewById(R.id.txtNom);
    TextView tv_Prenom = (TextView)view1.findViewById(R.id.txtPrenom);

    //(3) : Renseignement des valeurs
    tv_Nom.setText(arrayItems.get(i).nom);// i=position
    tv_Prenom.setText(arrayItems.get(i).prenom);

    //(4) Changement de la couleur du fond de notre item
    if (arrayItems.get(i).genre == Personne.MASCULIN) {
        view1.setBackgroundColor(Color.BLUE);
    } else {
        view1.setBackgroundColor(Color.MAGENTA);
    }

    //On retourne l'item créé.
    return view1;
}
```

Lorsque notre layout est initialisé, on peut récupérer les deux champs texte qui y sont présent (2) afin de modifier leur valeur (3). Afin de respecter notre contrat, les garçons doivent être en bleu et les filles en rose. Il nous suffit donc de changer la couleur du fond (4).

Pour résumer, nous avons une vue contenant une liste (activity_main.xml), une description de la présentation d'un item de la liste (listepersonne.xml) et un composant pour gérer le mapping donnée - IHM (PersonneAdapter.java). Il ne nous reste donc plus qu'à afficher notre liste. Pour ce faire, modifiez l'Activity principale.

```
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main2);
    //Récupération de la liste des personnes
    ArrayList<Personne> listP = Personne.getAListOfPersonne();

    //Création et initialisation de l'Adapter pour les personnes
    PersonneAdapter adapter = new PersonneAdapter( listP);
```

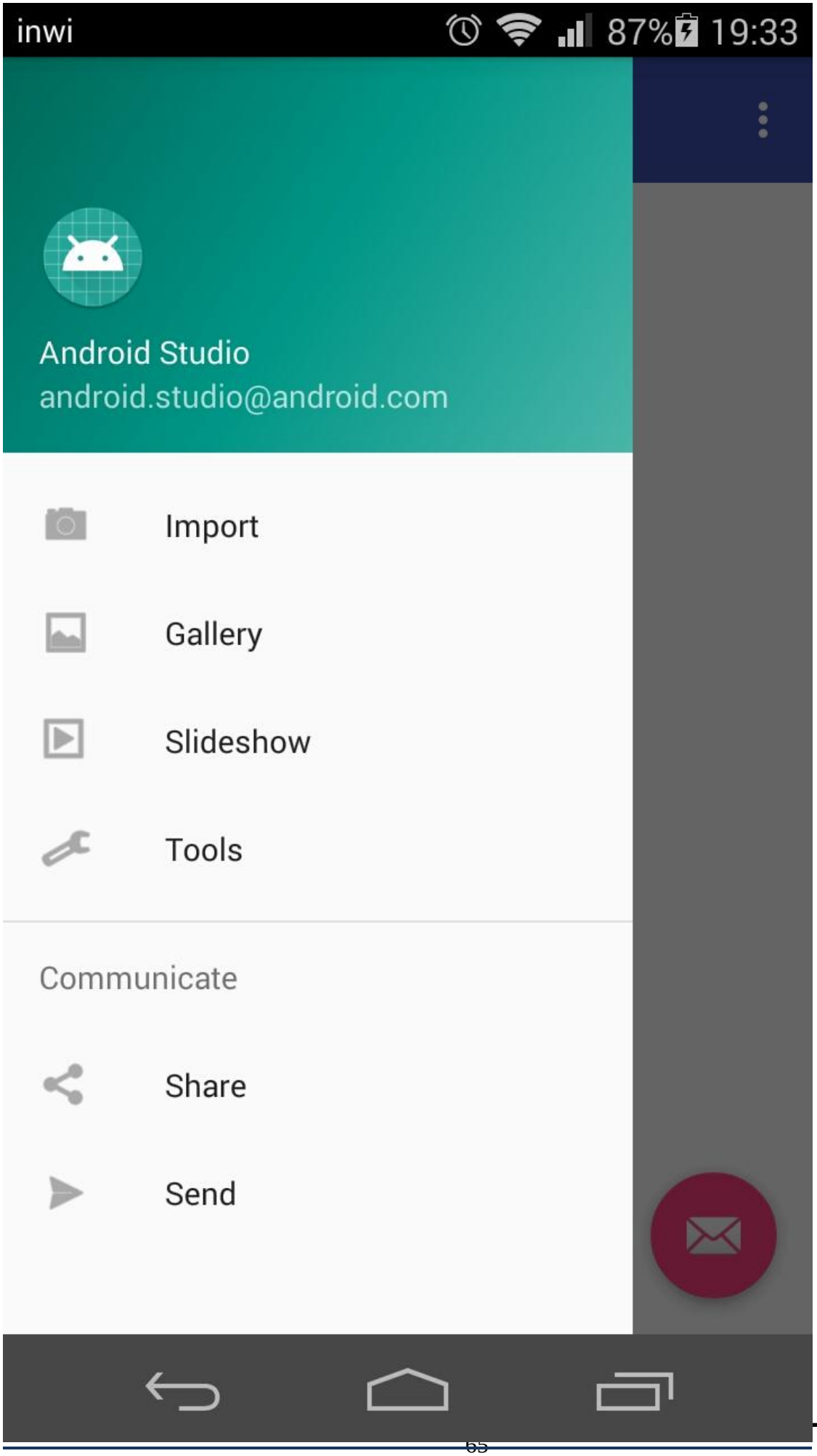
```
    //Récupération du composant ListView
    ListView list = (ListView)findViewById(R.id.List1);

    //Initialisation de la liste avec les données
    list.setAdapter(adapter);

}
```

Si vous voulez faire un traitement lorsqu'on clique sur un élément de la liste on utilise l'événement : `setOnItemClickListener` . (Exemple : afficher la position de l'élément sélectionné)

```
  ls.setOnItemClickListener(new AdapterView.OnItemClickListener() {
                @Override
                public void onItemClick(AdapterView<?> parent, View view, int
position, long id) {


Toast.makeText(getApplicationContext(),String.valueOf(position),Toast.LENGTH_LONG
).show();



        }
    });
```

Vous pouvez aussi ajouter un bouton à la view et dans la méthode `getView` de la classe `MyCustomAdapter` on utlise `setOnItemClickListener` du bouton :

```
Button btn=(Button) view1.findViewById(R.id.btnGetNom);
btn.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View view) {
        Toast.makeText(MainActivity.this, txtname.getText().toString(),
Toast.LENGTH_SHORT).show();
    }
});
```


inwi
87%
19:33
Android Studio
android.studio@android.com
Import
Gallery
Slideshow
Tools
Communicate
Share
Send

WebView

WebView est une vue qui affiche les pages Web dans votre application. Vous pouvez également spécifier chaîne HTML et peut montrer à l'intérieur de votre application en utilisant WebView. WebView fait transforme votre application à une application Web.

Pour ajouter WebView à votre application, vous devez ajouter **<WebView> élément à votre fichier de configuration de xml.** Sa syntaxe est la suivante :

```
<WebView
    android:id="@+id/web1"
    android:layout_width="match_parent"
    android:layout_height="match_parent" />
```

Pour l'utiliser, vous devez obtenir une référence de ce point de vue dans le fichier Java. Pour obtenir une référence, créer un objet de la WebView de classe. Sa syntaxe est :

```
WebView web=(WebView)findViewById(R.id.web1);
```

web1 : id du contrôle WebView dans mon fichier XML.

Pour charger une URL Web dans le WebView, vous avez besoin d'appeler une méthode **loadURL (String url) de la classe WebView, en spécifiant l'URL requise.** Sa syntaxe est:

```
web.loadUrl("http://wwww.google.com");
```

Pour que l'application puisse fonctionner correctement, elle devra avoir accès à Internet. Il faut donc donner l'autorisation d'accès à Internet à l'application. Pour cela, ouvrez le fichier **AndroidManifest.xml** et rajoutez la ligne suivante :

```
<uses-permission android:name="android.permission.INTERNET"></uses-permission>
```

Jusqu'à présent vous pouvez afficher la page demander dans votre WebView mais lorsque vous cliquer sur un lien il va vous demander de choisir le navigateur pour ouvrir ce lien. Pour résoudre ce problème on ajoute ce code avant de charger la page :

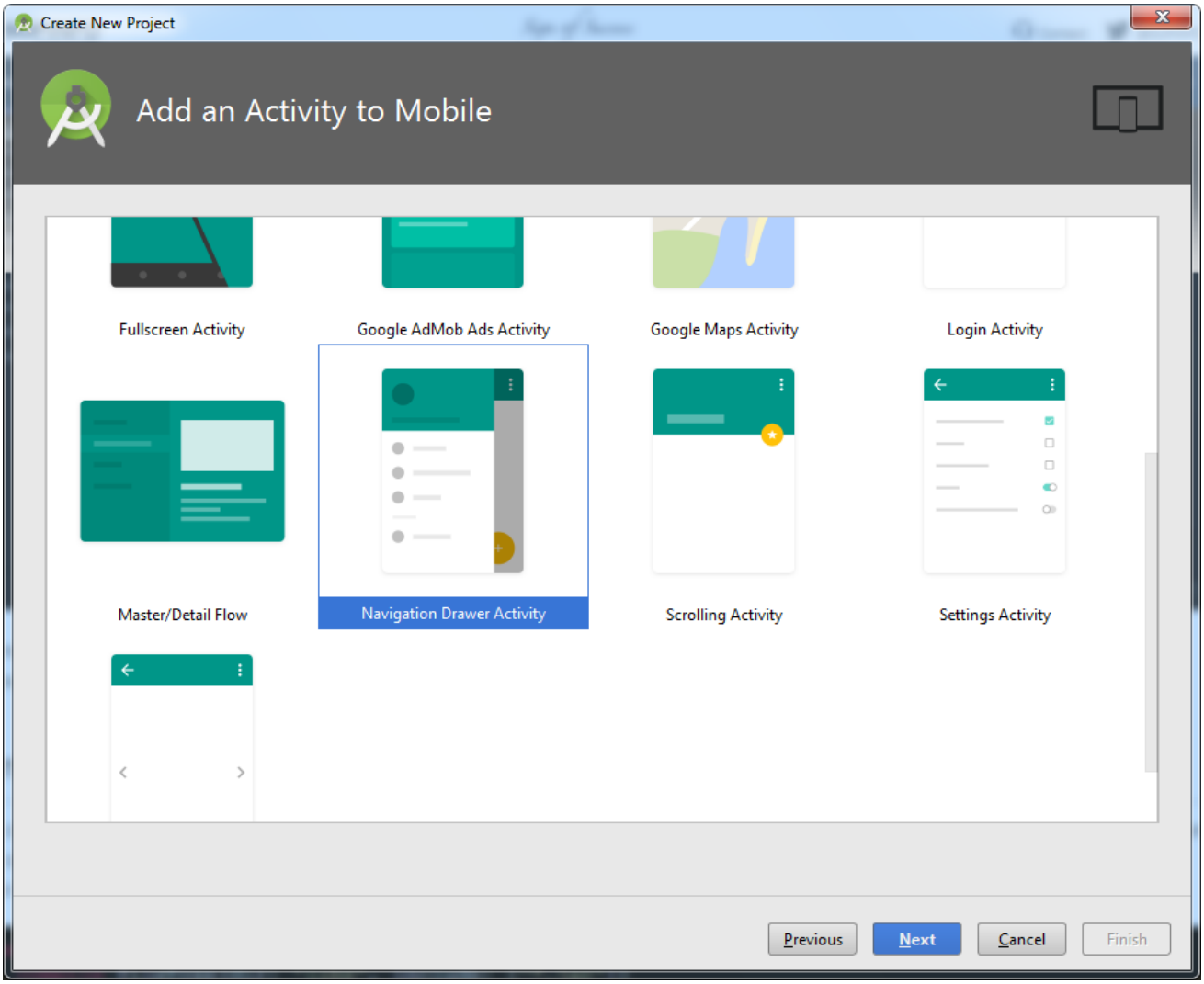
```
web.setWebViewClient(new WebViewClient());
```

En dehors de tout chargement url, vous pouvez avoir plus de contrôle sur votre WebView en utilisant les méthodes définies dans la classe WebView. Exemples :

- **getProgress () :** Cette méthode obtient la progression de la page en cours.
- **getTitle () :** Cette méthode renvoie le titre de la page en cours.
- **getUrl () :** Cette méthode retourne l'url de la page courante.

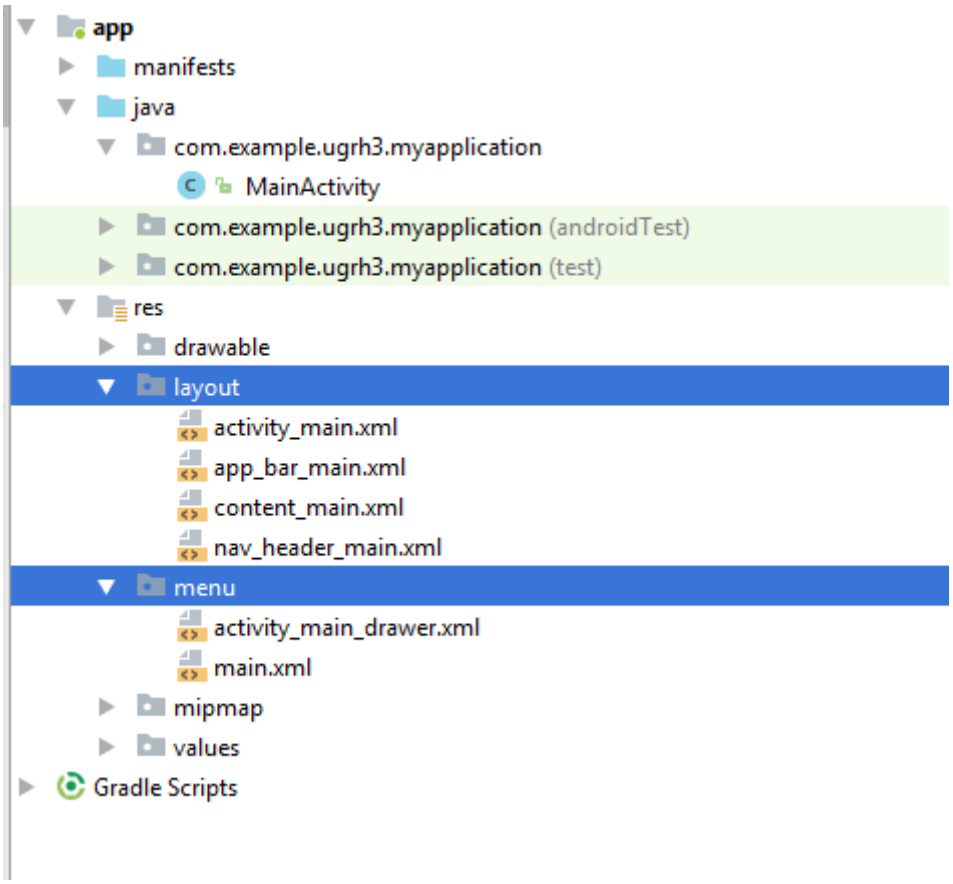
Navigation Drawer

Le **NavigationDrawer** est un élément permettant de naviguer dans une application Android. Cet élément a la particularité d'être caché la plupart du temps et d'apparaitre en fonction des besoins de l'utilisateur.

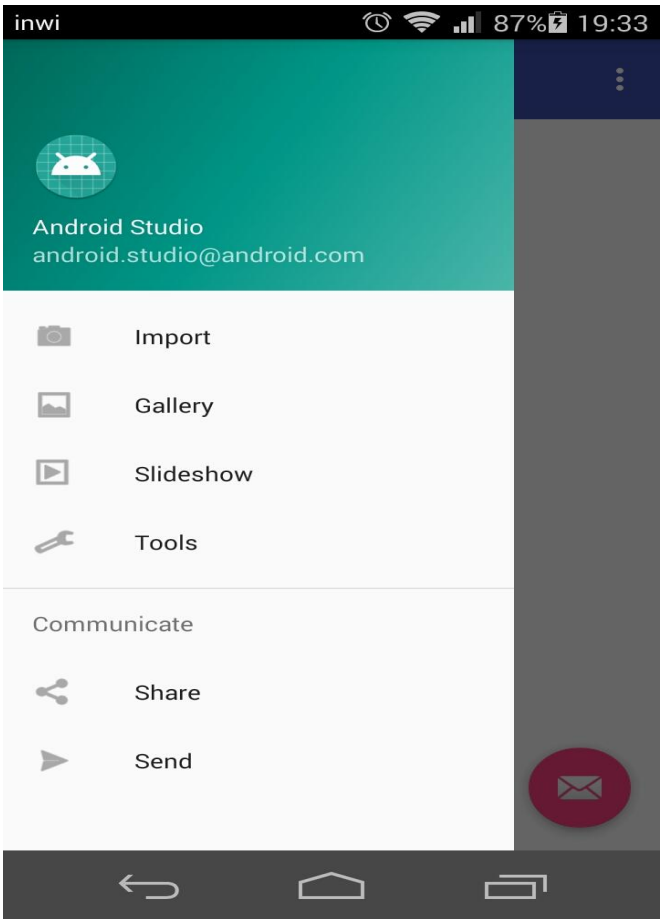


Après la création du projet nous allons remarquer que android studio a crée plusieurs fichiers diviser par deux dossier :

1. layout
2. menu



Pour modifier les composants du menu :

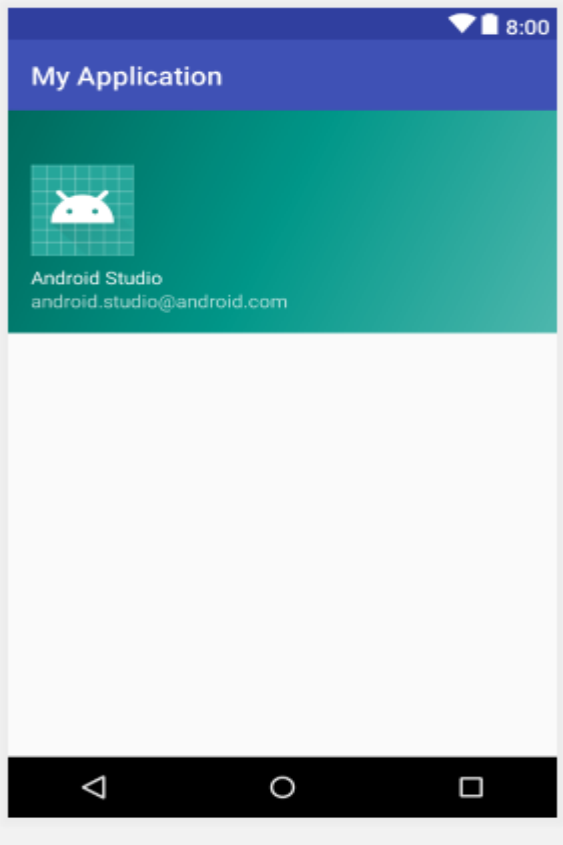


On modifier le fichier activity_main_drawer.xml

```
<menu xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    tools:showIn="navigation_view">

    <group android:checkableBehavior="single">
        <item
            android:id="@+id/nav_camera"
            android:icon="@drawable/ic_menu_camera"
            android:title="Import" />
        <item
            android:id="@+id/nav_gallery"
            android:icon="@drawable/ic_menu_gallery"
            android:title="Gallery" />
        <item
            android:id="@+id/nav_slideshow"
            android:icon="@drawable/ic_menu_slideshow"
            android:title="Slideshow" />
        <item
            android:id="@+id/nav_manage"
            android:icon="@drawable/ic_menu_manage"
            android:title="Tools" />
    </group>

    <item android:title="Communicate">
        <menu>
            <item
                android:id="@+id/nav_share"
                android:icon="@drawable/ic_menu_share"
                android:title="Share" />
            <item
                android:id="@+id/nav_send"
                android:icon="@drawable/ic_menu_send"
                android:title="Send" />
        </menu>
    </item>

</menu>
```

Vous remarquer que c'est très simple il suffit de modifier le titre ou l'icone souhaiter ou bien de supprimer l'élément.

Pour modifier l'entête du menu :



Ouvrez le fichier nav_header_main.xml vous allez trouver une layout avec une simple LinerLayout. La modification est très simple, vous pouvez modifier tout ce que vous voulez.

Il nous reste à savoir sur quel élément l'utilisateur à cliquer, pour cela : ouvrer la classe MainActivity.class et vous trouvez tous les codes du menu déjà intégrer il vous reste seulement à écrire le traitement que vous voulez lorsque l'utilisateur clique sur n'importe quel élément :

```
@SuppressWarnings("StatementWithEmptyBody")
@Override
public boolean onNavigationItemSelected(MenuItem item) {
    // Handle navigation view item clicks here.
    int id = item.getItemId();

    if (id == R.id.nav_camera) {
        // Handle the camera action
    } else if (id == R.id.nav_gallery) {

    } else if (id == R.id.nav_slideshow) {

    } else if (id == R.id.nav_manage) {

    } else if (id == R.id.nav_share) {

    } else if (id == R.id.nav_send) {

    }

    DrawerLayout drawer = (DrawerLayout) findViewById(R.id.drawer_layout);
    drawer.closeDrawer(GravityCompat.START);
    return true;
}
```

TabHost

Pour la mise en place d'onglets, il faut s'appuyer sur les widgets suivants :

- TabHost est le conteneur général ;
- TabWidget implémente la ligne des boutons d'onglets (qui contiennent labels, icônes ou une vue) ;
- FrameLayout est le conteneur des contenus des onglets : chaque onglet est un fils de FrameLayout.

Exemple de la définition statique d'onglets:

```
<TabHost
    android:id="@+id/tabtest"
    android:layout_width="match_parent"
    android:layout_height="match_parent">
    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:orientation="vertical">
        <TabWidget
            android:id="@android:id/tabs"
            android:background="@color/colorAccent"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"/>
        <FrameLayout
            android:id="@android:id/tabcontent"
            android:layout_width="match_parent"
            android:layout_height="match_parent">
            <LinearLayout
                android:id="@+id/disc"
                android:layout_width="match_parent"
                android:layout_height="match_parent"
                android:orientation="vertical">
                <LinearLayout
                    android:layout_width="match_parent"
                    android:layout_height="match_parent"
                    android:orientation="horizontal">
                    <TextView
                        android:id="@+id/textView3"
                        android:layout_width="wrap_content"
                        android:layout_height="wrap_content"
                        android:layout_weight="1"
                        android:text="Tab numero 1" />
                </LinearLayout>
            </LinearLayout>
            <LinearLayout
                android:id="@+id/statut"
                android:layout_width="match_parent"
                android:layout_height="match_parent"
                android:orientation="vertical">
                <LinearLayout
                    android:layout_width="match_parent"
                    android:layout_height="match_parent"
                    android:orientation="horizontal">
                    <TextView
                        android:id="@+id/textView4"
                        android:layout_width="wrap_content"
                        android:layout_height="wrap_content"
                        android:layout_weight="1"
```

```
                    android:text="Tab numero 2" />
            </LinearLayout>
        </LinearLayout>
        <LinearLayout
            android:id="@+id/appels"
            android:layout_width="match_parent"
            android:layout_height="match_parent"
            android:orientation="vertical">
            <LinearLayout
                android:layout_width="match_parent"
                android:layout_height="match_parent"
                android:orientation="horizontal">
                <TextView
                    android:id="@+id/textView5"
                    android:layout_width="wrap_content"
                    android:layout_height="wrap_content"
                    android:layout_weight="1"
                    android:text="Tab numero 3" />
            </LinearLayout>
        </LinearLayout>
    </FrameLayout>
</LinearLayout>
</TabHost>
```

On remarque que TabHost contient deux objets distincts, le TabWidget qui est la ligne des boutons d'onglets et le FrameLayout qui lui contient les onglets. Ainsi, c'est le LinearLayout qui contient les widgets visualisés dans l'onglet.

Exemple du code JAVA :

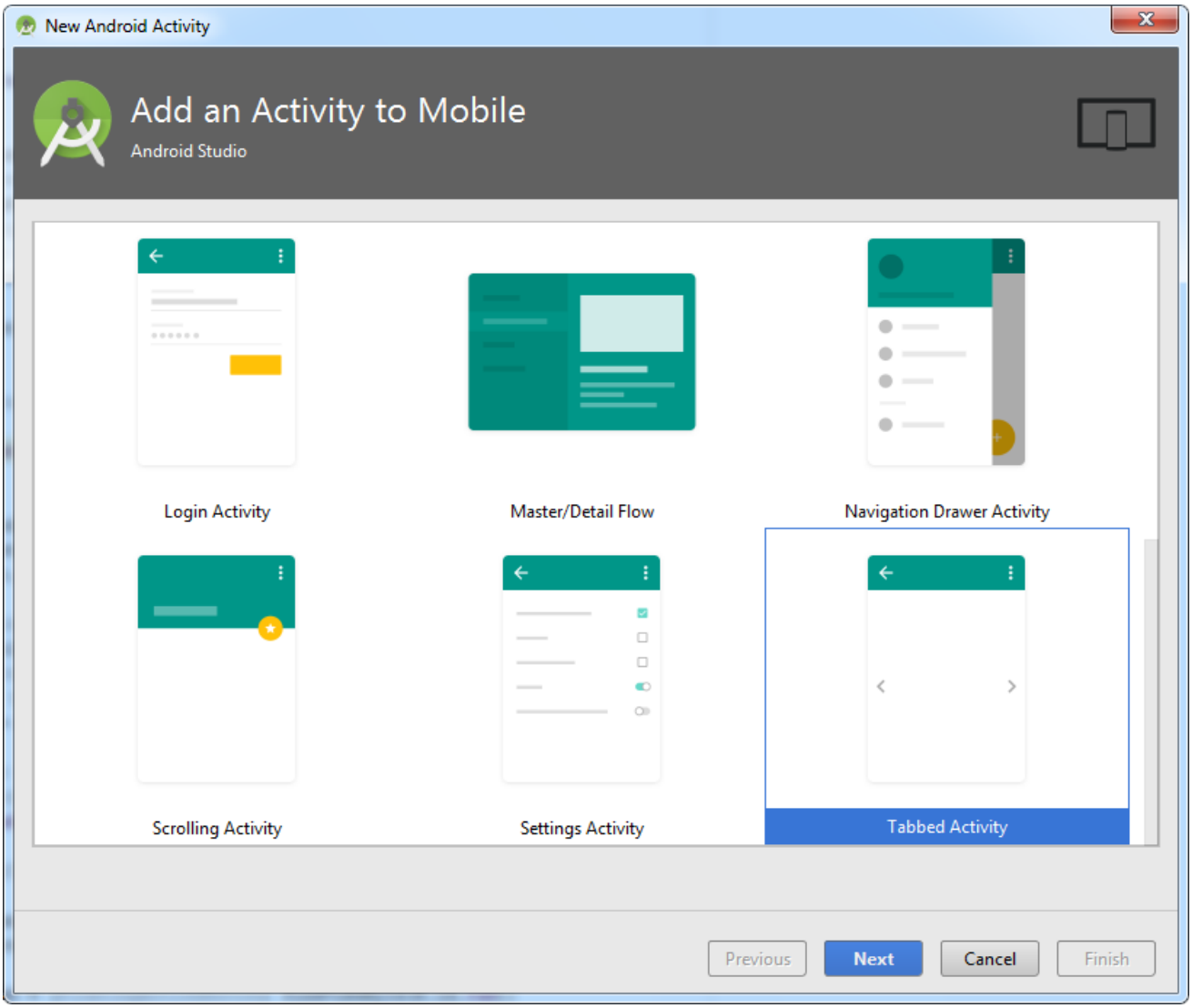
```
TabHost host = (TabHost)findViewById(R.id.tabtest);
host.setup();

//Tab 1
TabHost.TabSpec spec = host.newTabSpec("DISC.");
spec.setContent(R.id.disc);
spec.setIndicator("DISC.");
host.addTab(spec);

//Tab 2
spec = host.newTabSpec("STATUT");
spec.setContent(R.id.statut);
spec.setIndicator("STATUT");
host.addTab(spec);

//Tab 3
spec = host.newTabSpec("APPELS");
spec.setContent(R.id.appels);
spec.setIndicator("APPELS");
host.addTab(spec);

host.setCurrentTab(1);
```

On récupère dans un premier temps le TabHost puis on le démarre par .setup().
Puis on récupère les onglets et on l'ajoute au TabHost qu'on a récupérer.
On utilise la méthode setIndicateur pour affecter le titre à afficher dans l'onglet.
Par default le premier onglet va etre afficher mais si vous voulez afficher un autre onglet par default, utiliser la méthode setCurrentTab qui prend le numéro d'onglet comme paramètre.

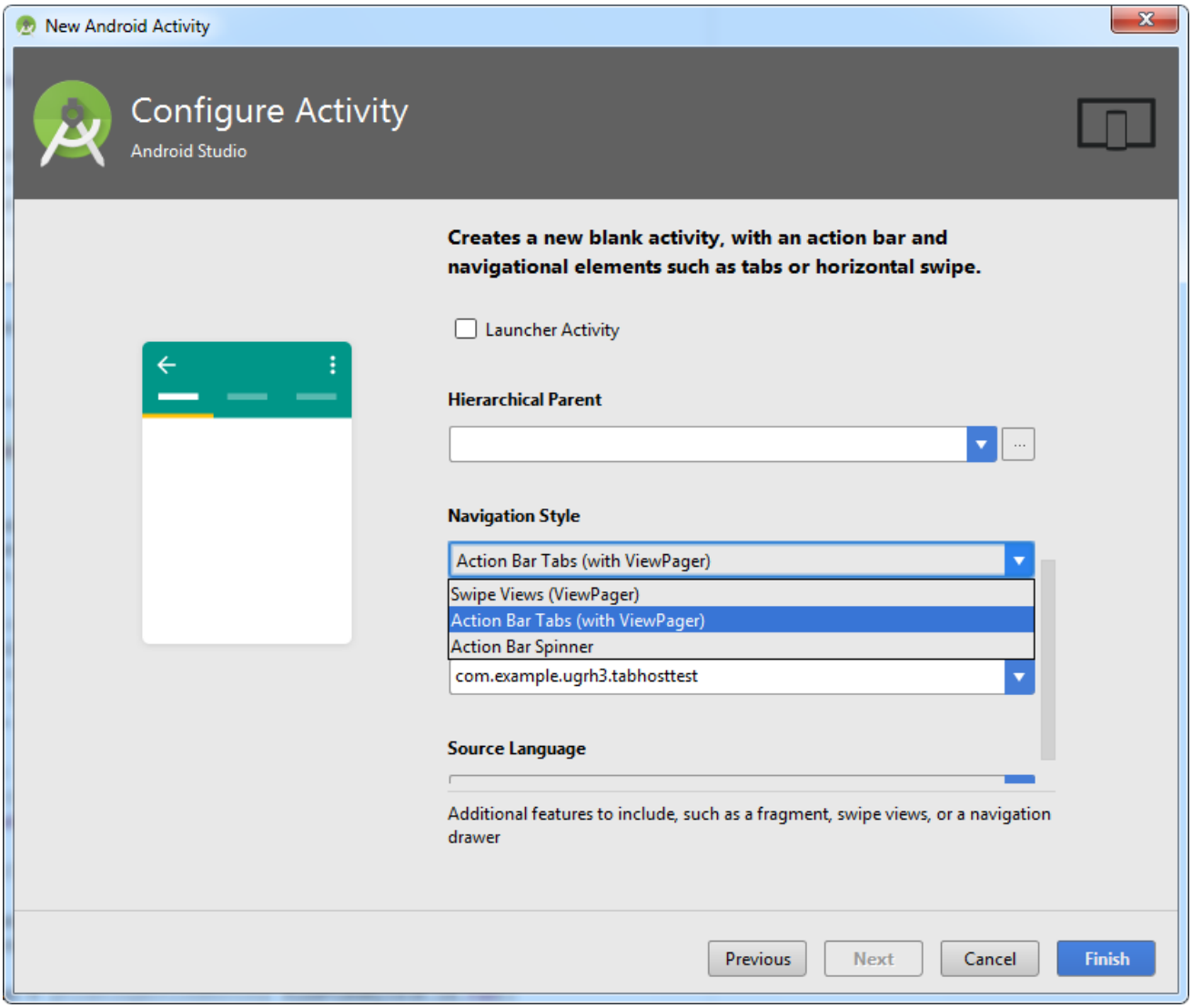
ViewPager

Nous allons aborder la notion de **Fragment**. Les Fragments ont été introduits dans la version Android 3.0 pour objectif de permettre une plus grande flexibilité pour les écrans larges tel que les tablettes tactiles. Notre objectif est simplement de créer un effet “slide” entre des pages.

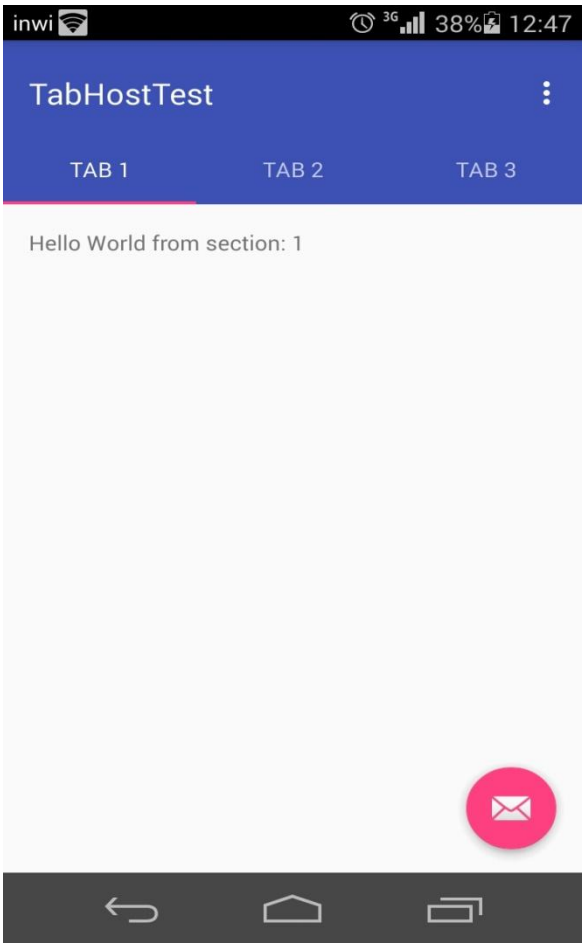
Si vous voulez crée un projet avec des effets slide : choisir le type des activity suivant :



Puis si vous voulez que ce slide comporte des onglets pour accès rapide alors dans l’étape suivante modifier la navigation style :



Apres la validation vous obtenez le résultat suivant :



Maintenant vous pouvez naviguer entre les onglets soit par un click sur l'onglet souhaité soit par une glisse sur l'écran.

CRÉATION DES TABS

Pour personnaliser le contenu de chaque onglet nous devons crée des fragments et l'affecter à l'onglet souhaité.

Dans le package **res => layout**, clic droit sur le répertoire, **new => layout resource file**, on les nommera réciproquement **fragment1.xml**, **fragment2.xml**, et **fragment3.xml**.

Puis on ajoute une TextView dans chaque Fragment et on écrit un texte différent pour différencier entre les onglets. Exemple :

```
<?xml version="1.0" encoding="utf-8"?>
<android.support.constraint.ConstraintLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
android:layout_width="match_parent"
    android:layout_height="match_parent">

    <TextView
        android:id="@+id/textView"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Onglet numéro 1" />
</android.support.constraint.ConstraintLayout>
```

Après la création des Fragment nous devons crée un class java pour chaque Fragment pour l'associé :

Ces class doivent hériter de la class Fragment (import android.support.v4.app.Fragment;) et doivent redéfinir la méthode onCreateView :

```
 public class tab1 extends Fragment
{
    @Nullable
    @Override
    public View onCreateView(LayoutInflater inflater, @Nullable ViewGroup
container, Bundle savedInstanceState) {
        return inflater.inflate(R.layout.fragment1, container, false);
    }
}
```

Après la création de tous le class java on passe à la modification de la classe main et on modifier la méthode :

```
@Override
public Fragment getItem(int position) {
    // getItem is called to instantiate the fragment for the given page.
    // Return a PlaceholderFragment (defined as a static inner class below).
    return PlaceholderFragment.newInstance(position + 1);
}
```

Par le code suivant :

```
@Override
public Fragment getItem(int position) {
switch (position)
    {
        case 0:
            Tab1 tab1=new Tab1();
            return tab1;
        case 1:
            Tab2 tab2=new Tab2();
            return tab2;
        case 2:
            Tab3 tab3=new Tab3();
            return tab3;

    }

    return null;
}
```

Pour modifier le titre de chaque onglet : modifier ces trois lignes dans le fichier string.xml :

SQLite

Ce que nous avons vu précédemment est certes utile, mais ne répondra pas à tous nos besoins. Nous avons besoin d'un moyen efficace de stocker des données complexes et d'y accéder. Or, il nous faudrait des années pour concevoir un système de ce style. Imaginez le travail s'il vous fallait développer de A à Z une bibliothèque multimédia qui puisse chercher en moins d'une seconde parmi

plus de 100 000 titres une chanson bien précise ! C'est pour cela que nous avons besoin des bases de données, qui sont optimisées pour ce type de traitements.

Les bases de données pour Android sont fournies à l'aide de SQLite. L'avantage de SQLite est qu'il s'agit d'un SGBD très compact et par conséquent très efficace pour les applications embarquées, mais pas uniquement puisqu'on le trouve dans Skype, Adobe Reader, Firefox, etc.

QU'EST-CE QUE SQLITE

SQLite est une base de données open source, qui supporte les fonctionnalités standards des bases de données relationnelles comme la syntaxe SQL, les transactions et les prepared statement. La base de données nécessite peu de mémoire lors de l'exécution (env. 250 ko), ce qui en fait un bon candidat pour être intégré dans d'autres environnements d'exécution.

SQLite prend en charge les types de données TEXT (similaire à String en Java), INTEGER (similaire à long en Java) et REAL (similaire à double en Java). Tous les autres types doivent être convertis en l'un de ces types avant d'être enregistrés dans la base de données. SQLite ne vérifie pas si les types des données insérées dans les colonnes correspondent au type défini, par exemple, vous pouvez écrire un nombre entier dans une colonne de type chaîne de caractères et vice versa.

Contrairement à MySQL par exemple, SQLite ne nécessite pas de serveur pour fonctionner, ce qui signifie que son exécution se fait dans le même processus que celui de l'application. Par conséquent, une opération massive lancée dans la base de données aura des conséquences visibles sur les performances de votre application. Ainsi, il `onCreate(SQLiteDatabase db)` vous faudra savoir maîtriser son implémentation afin de ne pas pénaliser le restant de votre exécution

L'accès à une base de données SQLite implique l'accès au système de fichiers. Cela peut être lent. Par conséquent, il est recommandé d'effectuer les opérations de base de données de manière asynchrone.

SQLITE POUR ANDROID

SQLite a été inclus dans le cœur même d'Android, c'est pourquoi chaque application peut avoir sa propre base. De manière générale, les bases de données sont stockées dans les répertoires de la forme /data/data/<package>/databases. Il est possible d'avoir plusieurs bases de données par application, cependant chaque fichier créé l'est selon le mode MODE_PRIVATE, par conséquent les bases ne sont accessibles qu'au sein de l'application elle-même. Notez que ce n'est pas pour autant qu'une base de données ne peut pas être partagée avec d'autres applications.

Enfin, il est préférable de faire en sorte que la clé de chaque table soit un identifiant qui s'incrémente automatiquement.

CRÉATION ET MISE À JOUR DE LA BASE DE DONNÉES AVEC SQLITEOPENHELPER

Pour créer et mettre à jour une base de données dans votre application Android, vous créez une classe qui hérite de SQLiteOpenHelper. Dans le constructeur de votre sous-classe, vous appelez la méthode super() de SQLiteOpenHelper, en précisant le nom de la base de données et sa version actuelle.

Dans cette classe, vous devez redéfinir les méthodes suivantes pour créer et mettre à jour votre base de données.
• onCreate() - est appelée par le framework pour accéder à une base de données qui n'est pas encore créée.
• onUpgrade() - est appelée si la version de la base de données est augmentée dans le code de votre application. Cette méthode vous permet de mettre à jour un schéma de base de données existant ou de supprimer la base de données existante et la recréer par la méthode onCreate().
void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) oldVersion est le numéro de l'ancienne version de la base de données que l'application utilisait, alors que newVersion est le numéro de la nouvelle version. En fait, Android rajoute automatiquement dans la base une table qui contient la dernière valeur connue de la base. À chaque lancement, Android vérifiera la dernière version de la base par rapport à la version actuelle dans le code. Si le numéro de la version actuelle est supérieur à celui de la dernière version, alors cette méthode est lancée.

Les deux méthodes reçoivent en paramètre un objet SQLiteDatabase qui est la représentation Java de la base de données.
La classe SQLiteOpenHelper fournit les méthodes getReadableDatabase() et getWriteableDatabase() pour accéder à un objet SQLiteDatabase en lecture, respectif en écriture.

C'est une bonne pratique de créer une classe par table. Cette classe définit des méthodes statiques onCreate() et onUpgrade(), qui sont appelées dans les méthodes correspondantes de la superclasse SQLiteOpenHelper. De cette façon, votre implémentation de SQLiteOpenHelper reste lisible, même si vous avez plusieurs tables

Pour créer une table, il vous faudra réfléchir à son nom et à ses attributs. Chaque attribut sera défini à l'aide d'un type de données. Ainsi, dans une table Agent comme exemple nous allons définir trois attributs :

- ID, qui est un entier auto-incrémental pour représenter la clé ;
- Nom, qui est une chaîne de caractères ;
- Prenom, qui est une chaîne de caractères.

Pour SQLite, c'est simple, il n'existe que cinq types de données :

- NULL pour les données NULL.
- INTEGER pour les entiers (sans virgule).
- REAL pour les nombres réels (avec virgule).
- TEXT pour les chaînes de caractères.
- BLOB pour les données brutes, par exemple si vous voulez mettre une image dans votre base de données.

La création de table se fait avec une syntaxe très naturelle :

```
CREATE TABLE nom_de_la_table (
  nom_du_champ_1 type {contraintes},
  nom_du_champ_2 type {contraintes},
  …);
```

Pour chaque attribut, on doit déclarer au moins deux informations :

- Son nom, afin de pouvoir l'identifier ;
- Son type de donnée.

Mais il est aussi possible de déclarer des contraintes pour chaque attribut à l'emplacement de {contraintes}. On trouve comme contraintes :

- PRIMARY KEY pour désigner la clé primaire sur un attribut ;
- NOT NULL pour indiquer que cet attribut ne peut valoir NULL ;
- CHECK afin de vérifier que la valeur de cet attribut est cohérente ;
- DEFAULT sert à préciser une valeur par défaut.

Ce qui peut donner par exemple :

```
CREATE TABLE nom_de_la_table (
 nom_du_champ_1 INTEGER PRIMARY KEY,
 nom_du_champ_2 TEXT NOT NULL,
 nom_du_champ_3 REAL NOT NULL CHECK (nom_du_champ_3 > 0),
 nom_du_champ_4 INTEGER DEFAULT 10);
```

Il existe deux types de requêtes SQL. Celles qui appellent une réponse, comme la sélection, et celles qui n'appellent pas de réponse. Afin d'exécuter une requête SQL pour laquelle on ne souhaite pas de réponse ou on ignore la réponse, il suffit d'utiliser la méthode void execSQL(String sql). De manière générale, on utilisera execSQL(String) dès qu'il ne s'agira pas de faire un SELECT, UPDATE, INSERT ou DELETE.

On va donc créer la classe `SqlTest` qui hérite donc de `SQLiteOpenHelper`. Cette classe va permettre de définir la table qui sera produite lors de l'instanciation. On va prendre comme exemple, une table table Agent dont les champs sont (id,nom,prenom).

```
public class SqlTest extends SQLiteOpenHelper {
    public static final String AGENT_KEY = "id";
    public static final String AGENT_NOM = "Nom";
    public static final String AGENT_PRENOM = "Prenom";
    public static final String AGENT_TABLE_NAME = "Agent";
    public static final String AGENT_TABLE_CREATE =
            "CREATE TABLE " + AGENT_TABLE_NAME + " (" +
    AGENT_KEY + " INTEGER PRIMARY KEY AUTOINCREMENT, " +
    AGENT_NOM + " TEXT, " +
    AGENT_PRENOM + " TEXT);";

    public SqlTest (Context context, String name, SQLiteDatabase.CursorFactory
factory, int version)
    {
        super(context, name, factory, version);
    }
    @Override
    public void onCreate(SQLiteDatabase db)
    {
        db.execSQL(AGENT_TABLE_CREATE);
    }
    @Override
    public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion)
    {
        db.execSQL("drop table IF  EXISTS "+ AGENT_TABLE_NAME);
```

```
        onCreate(db);
    }

}
```

Comme vous l'aurez remarqué, une pratique courante avec la manipulation des bases de données est d'enregistrer les attributs, tables et requêtes dans des constantes de façon à les retrouver et les modifier plus facilement. Tous ces attributs sont public puisqu'il est possible qu'on manipule la base en dehors de cette classe.

LIRE ET ÉCRIRE DANS LA BASE DE DONNÉES (LA CLASSE SQLITEDATABASE)

SQLiteDatabase est la classe de base pour travailler avec une base de données SQLite sous Android et fournit des méthodes pour ouvrir, effectuer des requêtes, mettre à jour et fermer la base de données.
Plus précisément, SQLiteDatabase fournit les méthodes insert(), update() et delete().
En outre, elle fournit la méthode execSQL(), qui permet d'exécuter une instruction SQL directement.
Pour réaliser des écritures ou lectures, on utilise les méthodes **getWritableDatabase()** et **getReadableDatabase()** qui renvoient une instance de **SQLiteDatabase**.

La sélection

Les requêtes peuvent être créées via les méthodes rawQuery() et query(), ou par la classe SQLiteQueryBuilder.

- rawQuery() accepte directement une requête SQL SELECT en entrée.

  ```
  db.rawQuery( "select * from Agent where Nom like '%za%'",null);
  ```
- query() fournit une interface structurée pour spécifier la requête SQL.

  query (String table, String[] columns, String whereClause, String[] selectionArgs, String groupBy, String having, String orderBy)

- columns est la liste des colonnes à retourner. Mettre null si on veut toutes les colonnes
- whereClause est la clause WHERE d'un SELECT (sans le mot WHERE). Mettre null si on veut toutes les lignes
- selectionArgs est utile si dans whereClause (~ WHERE), il y a des paramêtres notés ?. Les valeurs de ces paramêtres sont indiqués par selectionArgs. Bref en général on met null
- groupBy est la clause GROUP BY d'un SELECT (sans les mots GROUP BY). Utile pour des SELECT COUT(*). Bref on général on met null
- having indique les groupes de lignes à retourner (comme HAVING de SQL = un WHERE sur résultat d'un calcul, pas sur les données)
- orderBy est la clause ORDER BY d'un SELECT (sans les mots ORDER BY). Mettre null si on ne tient pas compte de l'ordre.

Exemple :

```
db.query(Agent, new String[] { ID,
        Nom, Prenom }, ID + "=?", new String[]
        { String.valueOf(id) }, null, null, null, null);
 est l'équivalent de "SELECT ID, Nom, Prenom FROM Agent WHERE ID='" + id + "'"
```

L'objet Cursor

- La méthode query() retourne un android.database.Cursor
- Le Cursor représente un ensemble de "lignes" contenant le résultat de la requête SELECT
- public int getCount() retourne le nombre de lignes contenues dans le Cursor
- On se positionne au début du Cursor (= avant la première ligne) par la méthode public boolean moveToFirst() (qui retourne false si le Cursor est vide)
- On teste si on a une nouvelle ligne à lire par la méthode public boolean moveToNext() (qui retourne false si on était positionné après la dernière ligne)
- On récupère la columnIndex cellule de la ligne par la méthode :

public XXX getXXX(int columnIndex). columnIndex est (Évidemment) le numéro de la cellule dans la requête. XXX est le type retourné (String, short, int, long, float, double)

- On referme le Cursor (et libère ainsi les ressources) par public void close ()
- On peut avoir des renseignements sur le résultat de la requête

SELECT (* FROM ...) (Méta données) à l'aide du Cursor comme :

- public int getColumnCount() qui retourne le nombre de colonnes contenues dans le Cursor
- public String getColumnName(int columnIndex) qui retourne le nom de la columnIndex ième colonne

Exemple Nombre de lignes

```
public int nbLigne()
{
    ArrayList al =new ArrayList();
    // pour lire  une base de donnee il faut utiliser getReadableDatabase
    SQLiteDatabase db=this.getReadableDatabase();
    Cursor cur=db.rawQuery("select * from Agent",null);

    return  cur.getCount();
        }
```

Exemple récupérer tous les lignes et les mettre dans un ArrayList

```
public ArrayList LireTousLignes()
{
    ArrayList al =new ArrayList();
    // pour lire  une base de donnee il faut utiliser getReadableDatabase
    SQLiteDatabase db=this.getReadableDatabase();
    Cursor cur=db.rawQuery( "select * from Agent where Nom like '%za%'",null);

    cur.moveToFirst();
    while (cur.isAfterLast()==false)
    {
        al.add(cur.getString(cur.getColumnIndex("Nom")) + " " +
cur.getString(cur.getColumnIndex("Prenom")));
        cur.moveToNext();
    }
    return  al;
}
```

L'insertion

Ayant obtenu une SQLiteDatabase, on utilise les méthodes de cette classe pour faire des opérations sur la base de données

- public long insert (String table, String nullColumnHack, ContentValues values) insert dans la table table les valeurs indiquées par values.
- values, de classe ContentValues, est une suite de couples (clé,valeur) où la clé, de classe String, est le nom de la colonne et valeur, sa valeur.
- Bref on prépare tout, la ligne à insérer en remplissant values par des put() successifs puis on lance insert()
- Le second argument nullColumnHack est le nom de colonne qui aura la valeur NULL si values est vide. Cela est du au fait que SQLite ne permet pas de lignes vides. Ainsi avec cette colonne, au moins un champ dans la ligne aura une valeur (= NULL). Bref cela sert seulement lorsque values est vide !
- Cette méthode insert() retourne le numéro de la ligne insérée ou -1 en cas d'erreur

Exemple : **Insérer** un Agent dans la table des agents.

```
public void insererAgent (String nom,String prenom)
{
    SQLiteDatabase db=this.getWritableDatabase();
    ContentValues cv=new ContentValues();
    cv.put("Nom",nom);
    cv.put("Prenom",prenom);
    db.insert("Agent",null,cv);
}
```

L'objet ContentValues est utilisé pour insérer des données dans la base. Ainsi, on peut dire qu'ils fonctionnent un peu comme les Bundle par exemple, puisqu'on peut y insérer des couples identifiant-valeur, qui représenteront les attributs des objets à insérer dans la base. L'identifiant du couple doit être une chaîne de caractères qui représente une des colonnes de la table visée.

Modification

Pour mettre à jour des lignes dans une table, la méthode utilisée (de la classe SQLiteDatabase) est public int update (String table, ContentValues values, String whereClause, String[] whereArgs) où :

- table est la table qui doit être mise à jour
- values est une suite de couples (clé, valeur) où la clé, de classe String, est le nom de la colonne et valeur, sa valeur
- whereClause est la clause WHERE filtrant les lignes à mettre à jour. Si la valeur est null, toutes les lignes sont mises à jour
- whereArgs indiquent les valeurs à passer aux différents arguments de la clause WHERE qui sont notés ? dans whereClause.
- Cette méthode retourne le nombre de ligne qui ont été affectées.

Exemple : Modifier une ligne dans la table Agent

```
public void ModifierAgent(int id,String Nom,String Prenom)
{
    SQLiteDatabase db = this.getWritableDatabase();
    ContentValues values = new ContentValues();
    values.put("Nom", Nom);
    values.put("Prenom", Prenom);
    db.update("Agent", values, "id" + " = ?",new String[] { String.valueOf(id)});
}
```

Suppression

Pour supprimer des lignes dans une table, la méthode utilisée (de la classe SQLiteDatabase) est public int delete (String table, String whereClause, String[] whereArgs)

- table est la table à manipuler
- whereClause est la clause WHERE filtrant les lignes à supprimer. Si la valeur est null, toutes les lignes sont détruites
- whereArgs indiquent les valeurs à passer aux différents arguments de la clause WHERE qui sont notés ? dans whereClause
- Cette méthode retourne le nombre de ligne qui ont été supprimées

Exemple : Supprimer une ligne dans la table Agent

```
public void SupprimerAgent(int id)
{
    SQLiteDatabase db = this.getWritableDatabase();
    db.delete("Agent", "Id" + " = ?", new String[] { String.valueOf(id) });
    db.close();
}
```

NB: on peut faire la modification et la suppression à l'aide de la méthode execSQL.

Exemple de la suppression :

```
public void SupprimerAgent(int id)
{
    db.execSQL("delete from agent where id ='"+ id +"'");
}
```

Pour appeler ces méthodes dans un événement (Click par exemple)

```
public void AjouterAgent(View view)
{
   try
   {
EditText txtNom=(EditText)findViewById(R.id.txtNOM);
EditText txtPrenom=(EditText)findViewById(R.id.txtPRENOM);
ListView lv=(ListView)findViewById(R.id.lstAgent);

// on crée une instance de la classe(Notre classe s'appel SqlTest
SqlTest db= new SqlTest(this,"BDTEST",null,1);
// BDTEST est le nom de la base de données et 1 est la version
// Puis on utilise cette instance pour appeler les méthodes
db.insererAgent(txtNom.getText().toString(),txtPrenom.getText().toString());
//Remplissage de la Listview
ArrayList<String> al =db.LireTousLignes();
lv.setAdapter(new ArrayAdapter <>(this,android.R.layout.simple_list_item_1,al));
   }
   catch (Exception e)
   {
Toast.makeText(this, e.getMessage().toString(), Toast.LENGTH_SHORT).show();
   }
}
```

Les services

QU'EST-CE QU'UN SERVICE ?

Un **service** est une composante essentielle d'une application. Il est nécessaire lorsque votre application souhaite effectuer des opérations ou des calculs en dehors de l'interaction utilisateur.

Un service ne possède pas d'interface graphique, mais permet de dérouler un algorithme sur un temps indéfini. Il s'arrêtera lorsque sa tâche sera finie ou lorsqu'il sera arrêté. Il peut être soit exécuté lors du lancement du téléphone (ou tout autre mécanisme interceptable : arrivée d'un appel, d'un SMS, etc.), soit au cours d'une action particulière dans votre application via un broadcast receivers.

Exemples concret d'utilisation ?

Exemple 1 : Envoyer une notification pour un rappel d'événement à tel date.

Exemple 2 : Lire une séquence audio en arrière plan.

Différents types de service

- Services locaux
- Services distants

- Les plus courants sont les services *locaux* (on trouve aussi le terme *started* ou *unbound service*), où l'activité qui lance le service et le service en lui-même appartiennent à la même application.
- Il est aussi possible qu'un service soit lancé par un composant qui appartient à une autre application, auquel cas il s'agit d'un service *distant* (on trouve aussi le terme bound*service*). Dans ce cas de figure, il existe toujours une interface qui permet la communication entre le processus qui a appelé le service et le processus dans lequel s'exécute le service. Cette communication permet d'envoyer des requêtes ou récupérer des résultats par exemple.

- Un service **local** est réservé à une application. L'accès au service à partir d'autres applications est bloqué en ajoutant une ligne au fichier manifeste Android :

```
<service android:exported="false" android:name=".MonPremierService"/>
```

- Un service **distant** est public, et d'autres applications peuvent y accéder. L'accès au service est accordé en ajoutant une ligne au fichier manifeste Android :

```
<service android:exported="true" android:name=".MonPremierService"/>
```

Attention : Un service peut être les deux à la fois.

CYCLE DE VIE D'UN SERVICE

Un **service** n'a pas de durée définie, il est là pour exécuter sa tâche et il fonctionnera tant que c'est nécessaire.
On va présenter les différentes méthodes qui correspondent au **cycle de vie d'un service**. Ces méthodes seront à surcharger quand vous créerez votre service :

- **OnCreate()** : Cette méthode est appelée à la création du service et est en général utilisée pour initialiser ce qui sera nécessaire à votre service.
- **OnStart(Intent i)** : Le service démarre. Valable uniquement pour les versions du **SDK inférieur à 2.0**.
- **OnStartCommand(Intent i, int flags, int startId)** : Le service démarre. Valable uniquement pour les versions du **SDK supérieur à 2.0**.
- **OnDestroy()** : Appelé à la fermeture du **service**.

La valeur retournée par la fonction onStartCommand

START_NOT_STICKY

Si le système tue le service, alors ce dernier ne sera pas recréé. Il faudra donc effectuer un nouvel appel à startService() pour relancer le service.

Ce mode vaut le coup dès qu'on veut faire un travail qui peut être interrompu si le système manque de mémoire et que vous pouvez le redémarrer explicitement par la suite pour recommencer le travail. Si vous voulez par exemple mettre en ligne des statistiques sur un serveur distant. Le processus qui lancera la mise en ligne peut se dérouler toutes les 30 minutes, mais, si le service est tué avant que la mise en ligne soit effectuée, ce n'est pas grave, on le fera dans 30 minutes.

START_STICKY

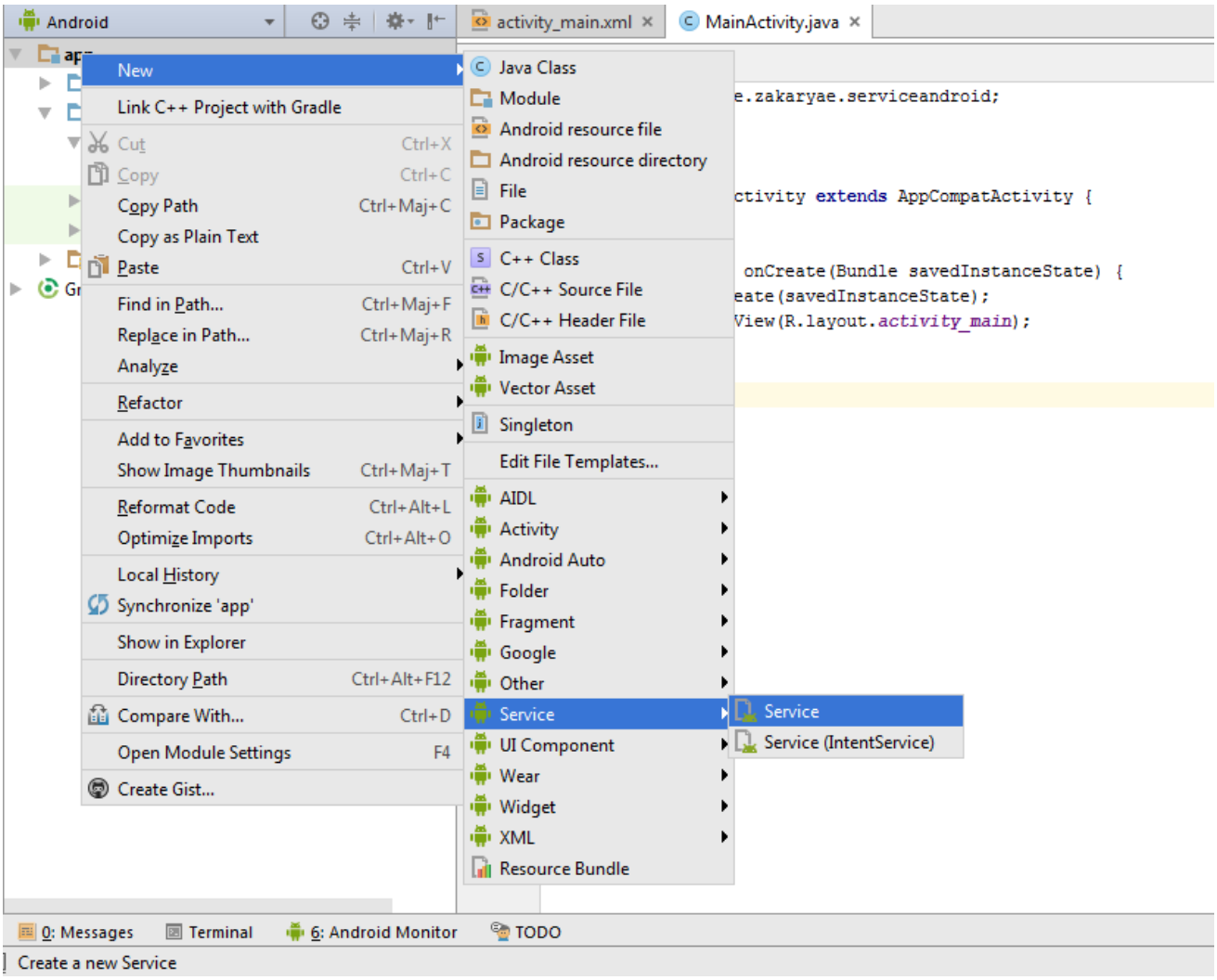
Cette fois, si le système doit tuer le service, alors il sera recréé mais sans lui fournir le dernier Intentqui l'avait lancé. Ainsi, le paramètre intent vaudra null. Ce mode de fonctionnement est utile pour les services qui fonctionnent par intermittence, comme par exemple quand on joue de la musique.

START_REDELIVER_INTENT

Si le système tue le service, il sera recréé et dans onStartCommand() le paramètre intent sera identique au dernier intent qui a été fourni au service. START_REDELIVER_INTENT est indispensable si vous voulez être certains qu'un service effectuera un travail complètement.

Création d'un service Android

Pour créer un nouveau Service Android, choisissez **app > new > Service > Service**



LES SERVICES LOCAUX

L'objectif est de créer un service ou l'on va pouvoir visualiser le cycle de vie d'un service.

Pour cela je propose de commencer par implémenter des toasts sur chaque fonction du service en question. Je vous suggère également d'implémenter les return de la façon suivante.

```
@Override
    public void onCreate()
        {
// TODO Auto-generated method stub
            }
    @Override
    public IBinder onBind(Intent intent) {
        return null;
    }
    @Override
    public void onDestroy() {
        super.onDestroy();
            }
    @Override
    public int onStartCommand(Intent intent, int flag, int startId){

//on retourne le flag comme évoqué dans l'introduction
        return START_NOT_STICKY;
    }
```

Arrivé à ce stade vous pouvez exécuter l'application et la tester sur votre smartphone android. Vous devriez apercevoir un toast pour chaque action du cycle de vie du service.

N'oubliez pas que dans les paramètres du service vous pouvez "forcer l'arrêt" du service étant donné que nous n'avons pas implémenté d'arrêt automatique de celui-ci. Je vous invite par ailleurs à modifier le paramètre return de la méthode onStartCommande() pour observer les différents comportements évoqué dans l'introduction.

Premier Exemple

```
    @Override
    public void onCreate() {
// TODO Auto-generated method stub
        Toast.makeText(this, "Service.onCreate()", Toast.LENGTH_LONG).show();
    }
    @Override
    public IBinder onBind(Intent intent) {

        return null;
    }
    @Override
    public void onDestroy() {
        super.onDestroy();
        Toast.makeText(this, "Service.onDestroy()", Toast.LENGTH_LONG).show();
    }
    @Override
    public int onStartCommand(Intent intent, int flag, int startId){
        Toast.makeText(this, "Service.onStartCommand()",
Toast.LENGTH_LONG).show();

//on retourne le flag comme évoqué dans l'introduction
        return START_NOT_STICKY;
    }
```

Listner pour lancer et Arrêter le service

lancer

```
buttonStop.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View view) {
        startService(new Intent(MainActivity.this, MonPremierService.class));
    }
});
```

Arrêter

```
buttonStop.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View view) {
        stopService(new Intent(MainActivity.this, MonPremierService.class));
    }
});
```

Deuxième Exemple

Dans ce deuxième exemple nous allons créer un service qui sert à lancer un compteur et afficher son résultat dans un Toast.

Le code qui sera exécuté au lancement du service doit être écrit dans la fonction onStartCommand

```
    @Override
    public void onCreate() {
// TODO Auto-generated method stub
```

```
        Toast.makeText(this, "Service.onCreate()", Toast.LENGTH_LONG).show();
    }
    @Override
    public IBinder onBind(Intent intent) {

        return null;
    }
    @Override
    public void onDestroy() {
        super.onDestroy();
        Toast.makeText(this, "Service.onDestroy()", Toast.LENGTH_LONG).show();
    }
    @Override
    public int onStartCommand(Intent intent, int flag, int startId){
        for (int i = 1; i <11 ; i++)
        {
            Toast.makeText(this, i, Toast.LENGTH_SHORT).show();
            try {Thread.sleep(1000);}
            catch(Exception e){}
        }
        return START_STICKY;

//on retourne le flag comme évoqué dans l'introduction

    }
```

Ce service va nous afficher de 1 à 10 et s'arrête une seconde après chaque affichage.

Le code pour lancer ce service et le même que l'exemple précédent.

Troisième exemple : créer une alarme

Le code xml de notre activity :

```
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context="com.example.zakaryae.serviceandroid.MainActivity"

    android:orientation="vertical"
    tools:layout_editor_absoluteY="8dp"
    tools:layout_editor_absoluteX="8dp">

    <TimePicker
        android:id="@+id/timePicker"
        android:layout_width="match_parent"
        android:layout_height="wrap_content" />

<Button
    android:id="@+id/btnLancer"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:text="@string/Lancer"
    android:textColor="@color/ColorBotona" />

<Button
    android:id="@+id/btnArreter"
    android:layout_width="match_parent"
```

```
    android:layout_height="wrap_content"
    android:text="@string/Arreter" />
```

Fichier string.xml:

```
<resources>
    <string name="app_name">ServiceAndroid</string>
    <string name="Lancer">Lancer</string>
    <string name="Arreter">Arrêter</string>
</resources>
```

Le service:

```
public class MonPremierService extends Service {
    public MonPremierService() {
    }

    @Override
    public void onCreate() {
// TODO Auto-generated method stub
        Toast.makeText(this, "Service.onCreate()", Toast.LENGTH_LONG).show();
    }
    @Override
    public IBinder onBind(Intent intent) {

        return null;
    }
    @Override
    public void onDestroy() {
        super.onDestroy();
        Toast.makeText(this, "Service.onDestroy()", Toast.LENGTH_LONG).show();
    }
    @Override
    public int onStartCommand(Intent intent, int flag, int startId){
        Toast.makeText(this, "Alert", Toast.LENGTH_LONG).show();
        try
        {

Uri notification= RingtoneManager.getDefaultUri(RingtoneManager.TYPE_ALARM);
Ringtone r=RingtoneManager.getRingtone(getApplicationContext(),notification);
            r.play();

        }
        catch(Exception e){e.printStackTrace();}
        return START_STICKY;

//on retourne le flag comme évoqué dans l'introduction

    }

}
```

Le code de onStartCommand : afficher message Alert puis choix de sonnerie à utiliser et le lancer.

MainActivity.java :

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    final  Button buttonStart = (Button) findViewById(R.id.btnLancer);
    final Button buttonStop = (Button) findViewById(R.id.btnArreter);
    final TimePicker alarm=(TimePicker)findViewById(R.id.timePicker) ;


    buttonStart.setOnClickListener(new View.OnClickListener() {
```

```
        @Override
        public void onClick(View view) {

            Intent in=new Intent(MainActivity.this,MonPremierService.class);
            AlarmManager am=(AlarmManager)getSystemService(ALARM_SERVICE);
            PendingIntent pi=PendingIntent.getService(MainActivity.this,0,in,0);
            Calendar c=Calendar.getInstance();
            c.set(Calendar.HOUR_OF_DAY,alarm.getCurrentHour());
            c.set(Calendar.MINUTE,alarm.getCurrentMinute());
            c.set(Calendar.SECOND,00);
            am.setRepeating(AlarmManager.RTC_WAKEUP,c.getTimeInMillis(),0,pi);
        }
    });
    buttonStop.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View view) {
            stopService(new Intent(MainActivity.this, MonPremierService.class));
        }
    });
}
```

Lorsque on click sur le bouton lancer on crée une alarme et on attend l'heure qu'on a choisi pour lancer le service.

SERVICES DISTANTS

Dans les exemples précédents on a seulement lancé un service mais on n'a pas pu récupérer des résultats à partir de notre service (pas d'interaction entre le service et l'utilisateur).

On utilisera cette fois boolean bindService(Intent service, ServiceConnection conn, int flags) afin d'assurer une connexion persistante avec le service. Le seul paramètre que vous ne connaissez pas est conn qui permet de recevoir le service quand celui-ci démarrera et permet de savoir s'il meurt ou s'il redémarre.

Un ServiceConnection est une interface pour surveiller l'exécution du service distant. Il existe deux méthodes de *callback* que vous devrez redéfinir :

1. void onServiceConnected(ComponentName name, IBinder service) qui est appelée quand la connexion au service est établie, avec un IBinder qui correspond à un canal de connexion avec le service.
2. void onServiceDisconnected(ComponentName name) qui est appelée quand la connexion au service est perdue, en général parce que le processus qui accueille le service a planté ou a été tué.

Mais qu'est-ce qu'un IBinder ? Comme je l'ai déjà dit, il s'agit d'un pont entre votre service et l'activité, mais au niveau du service. Les IBinder permettent au client de demander des choses au service. Alors, comment créer cette interface ? Tout d'abord, il faut savoir que le IBinder qui sera donné à onServiceConnected(ComponentName, IBinder) est envoyé par la méthode de *callback* IBinder onBind(Intent intent) dans Service. Maintenant, il suffit de créer un IBinder. Nous allons voir la méthode la plus simple, qui consiste à permettre à l'IBinder de renvoyer directement le Service de manière à pouvoir effectuer des commandes dessus.

```
public class MonService extends Service {
    // Attribut de type IBinder
    private final IBinder mBinder = new MonBinder();
    // Le Binder est représenté par une classe interne
    public class MonBinder extends Binder {
        // Le Binder possède une méthode pour renvoyer le Service
        MonService getService() {
            return MonService.this;
        }
    }

    @Override
    public IBinder onBind(Intent intent)
    {
        return mBinder;
    }


}
```

Le service sera créé s'il n'était pas déjà lancé (appel à onCreate() donc), mais ne passera pas par onStartCommand().

Pour qu'un client puisse se détacher d'un service, il peut utiliser la méthode void unbindService(ServiceConnection conn) de Context, avec conn l'interface de connexion fournie précédemment à bindService().

Ainsi, voici une implémentation typique d'un service distant :

```
public class MonService extends Service {
    // Retient l'état de la connexion avec le service
    public boolean mBound = false;
    // Attribut de type IBinder
    private final IBinder mBinder = new MonBinder();

      @Override
    public void onCreate()
    {
                }

    @Override
    public IBinder onBind(Intent intent)
    {
        return mBinder;
    }

    @Override
    public boolean onUnbind(Intent intent)
    {
        return false;
    }

    @Override
    public void onRebind(Intent intent)
    {
            }

    @Override
    public void onDestroy()
    {
```

```
                super.onDestroy();
    }
    // Le Binder est représenté par une classe interne
    public class MonBinder extends Binder {
        // Le Binder possède une méthode pour renvoyer le Service
        MonService getService() {
            return MonService.this;
        }
    }
}
```

Exemple

L'exemple qu'on va créer est de lancer un service puis lorsque click sur un bouton on va récupérer un numéro aléatoire à partir du service avec l'affichage d'un message dans chaque événement du service.

Notre activity comporte trois boutons : un pour lancer le service, l'autre pour l'arrêter et le dernier boiton pour récupérer le numéro aléatoire.

Activity :

```
<LinearLayout
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    android:padding="15dp">

    <Button
        android:id="@+id/btnLancer"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:onClick="lancer"
        android:text="Lancer" />

    <Button
        android:id="@+id/btnArreter"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:onClick="stop"
        android:text="Stop" />

    <Button
        android:id="@+id/button3"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:onClick="interaction"
        android:text="Interaction" />
</LinearLayout>
```

Service:

```
public class MonService extends Service {
    // Retient l'état de la connexion avec le service
    public boolean mBound = false;
    // Attribut de type IBinder
    private final IBinder mBinder = new MonBinder();

private final Random mrand=new Random();
```

```
      @Override
    public void onCreate()
    {
        Toast.makeText(this, "Service.onCreate", Toast.LENGTH_SHORT).show();
        }

    @Override
    public IBinder onBind(Intent intent)
    {
        Toast.makeText(this, "Service.onBind", Toast.LENGTH_SHORT).show();
        return mBinder;
    }

    @Override
    public boolean onUnbind(Intent intent)
    {
        Toast.makeText(this, "Service.onUnbind", Toast.LENGTH_SHORT).show();
        return false;
    }

    @Override
    public void onRebind(Intent intent)
    {
        Toast.makeText(this, "Service.onRebind", Toast.LENGTH_SHORT).show();
    }

    @Override
    public void onDestroy()
    {
        Toast.makeText(this, "Service.onDestroy", Toast.LENGTH_SHORT).show();
        super.onDestroy();
    }



    // Le Binder est représenté par une classe interne
    public class MonBinder extends Binder {
        // Le Binder possède une méthode pour renvoyer le Service
        MonService getService() {
            return MonService.this;
        }
    }

    public int getRandNumber()
    {
        return mrand.nextInt();
    }
}
```

Code java:

Déclaration Instance du service et un variable pour tester si on est connectée au service ou pas.

```
 MonService mservice;
boolean mbound=false;
```

Méthodes de lancement du service. Ici on utilise bindService à la place du startService

```
public void lancer(View view)
{
    Intent mIntent = new Intent(this, MonService.class);
    bindService(mIntent, mConnexion, BIND_AUTO_CREATE);

}
```

Méthodes pour stopper le service. Ici on utilise unbindService à la place du stopService

```
    public void stop(View view)
    {
if (mbound)
{
    unbindService(mConnexion);
    mbound=false;
}
    }
```

Récupérer le numéro aléatoire à partir du service.

```
public void interaction(View view)
{
    if (mbound)
    {
        int a;
        a=mservice.getRandNumber();
        Toast.makeText(mservice, String.valueOf(a), Toast.LENGTH_SHORT).show();
    }
}
```

Interface de connexion au service

```
private ServiceConnection mConnexion = new ServiceConnection() {
    // Se déclenche quand l'activité se connecte au service
    @Override
    public void onServiceConnected(ComponentName className, IBinder service) {
        MonService.MonBinder binder=(MonService.MonBinder)service;
        mservice=binder.getService();
        mbound=true;
    }
    // Se déclenche dès que le service est déconnecté
    @Override
    public void onServiceDisconnected(ComponentName className) {
        mbound = false;
    }
};
```

Les receivers

Au cours du développement de nos applications, on a besoin dans plusieurs cas qu'on soit notifié d'un événement qui se produit dans le téléphone mobile pour qu'on puisse adapter les applications face è ces événements comme il faut : Appel/SMS entrant , Connexion internet devenu disponible, changement d'un paramètre dans la configuration du téléphone, batterie faible, Ecouteur branché,… etc.

Pour cela, le système Android nous prépare le «sol» pour bien gérer ce besoin à travers les '**Broadcast Receivers**'

UNE DÉFINITION GÉNÉRALE

Quand un événement se produit dans le téléphone mobile, le Système fait une sorte d'émission du déroulement de cet événement. Ce mécanisme peut se généraliser entre les applications entre eux par des émissions propres à eux qu'ils définissent.

A ce temps là intervient le Broadcast Receiver : C'est un composant Android qui peut être illustré comme une sorte d'antenne (Receiver) qui est destinée à capter une émission d'un événement bien particulier.

Donc pour utiliser ce mécanisme, il suffit de :

1. Créer le Receiver
2. Mettre le Receiver sur la bonne voie (le diriger à capter un événement spécifique)
3. Décider quoi faire face à cet événement

TYPE DE BROADCASTS

A. Broadcasts normales : Ce type d'émission est asynchrone où le système (ou resp. l'application qui émet) n'attend aucun résultat de cette émission et l'ordre des Reveivers qui interceptent cette émission n'est pat définit à l'avance

B. Broadcasts ordonnées : Ce type d'émission est intercepté par un seul Receiver en un temps donné. Dans ce cas, chaque Receiver qui capte cette émission exécute sa tâche et laisse la main au Receiver suivant (l'ordre est indiqué par la priorité indiqué lors de la déclaration du Receiver). Dans certains cas, il y a une possibilité qu'un Receiver consomme cette émission et l'annule d'où elle ne va pas être intercepté par les Receivers qui suivent.

Ce type est utilisé par exemple pour le Broadcast de l'événement d'un appel sortant ou on a la possibilité de réécrire/modifier le numéro à appeler ou même refuser l'appel (Cela va être plus clair avec l'exemple pratique)

C. Sticky Broadcasts : Dans les 2 types qui précèdent, quand les Receivers consomment l'émission, cette dernière ne serait plus disponible. Pour ce type de Broadcast, elle reste disponible à l'écoute à tout moment. L'exemple le plus pertinent de ce type est le Broadcast du niveau de la batterie car c'est une information disponible à tout moment.

IMPLÉMENTER ET UTILISER LES BROADCAST RECEIVER

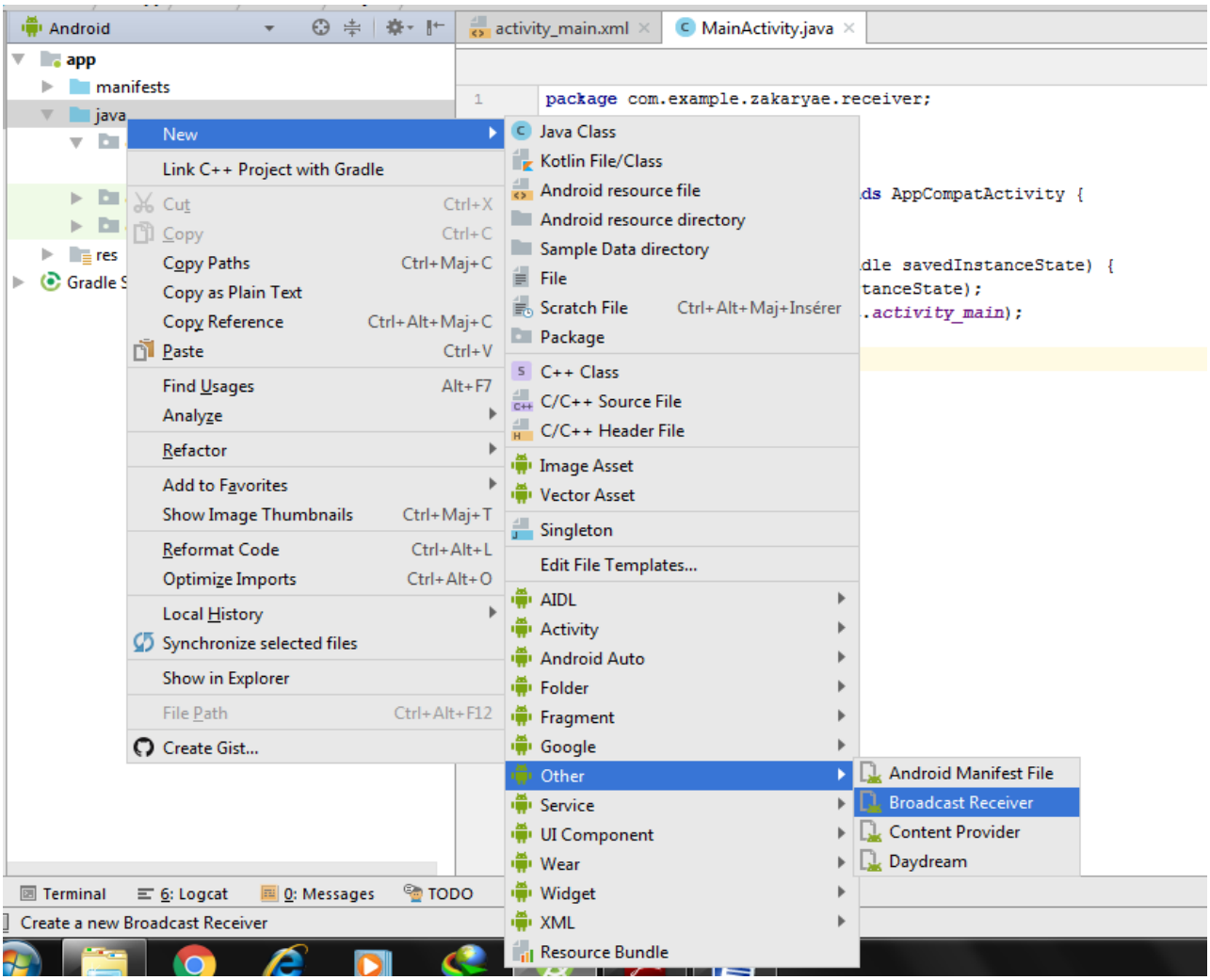
Pour configurer un Broadcast Receiver on 2 méthodes :

1. Méthode dynamique où toute l'implémentation se fait dans le code Java de l'application. L'apport de cette méthode est dans sa flexibilité dans le fait où on peut activer/désactiver le Receiver pour optimiser les ressources.

2. Méthode statique où la définition du Receiver et sa configuration se fait au niveau du fichier AndroidManifest.xml et la partie ou on définit « Que fait l'application à ce moment » est au niveau du code Java de l'application. Dans ce qui suit, la méthode statique serait utilisée.

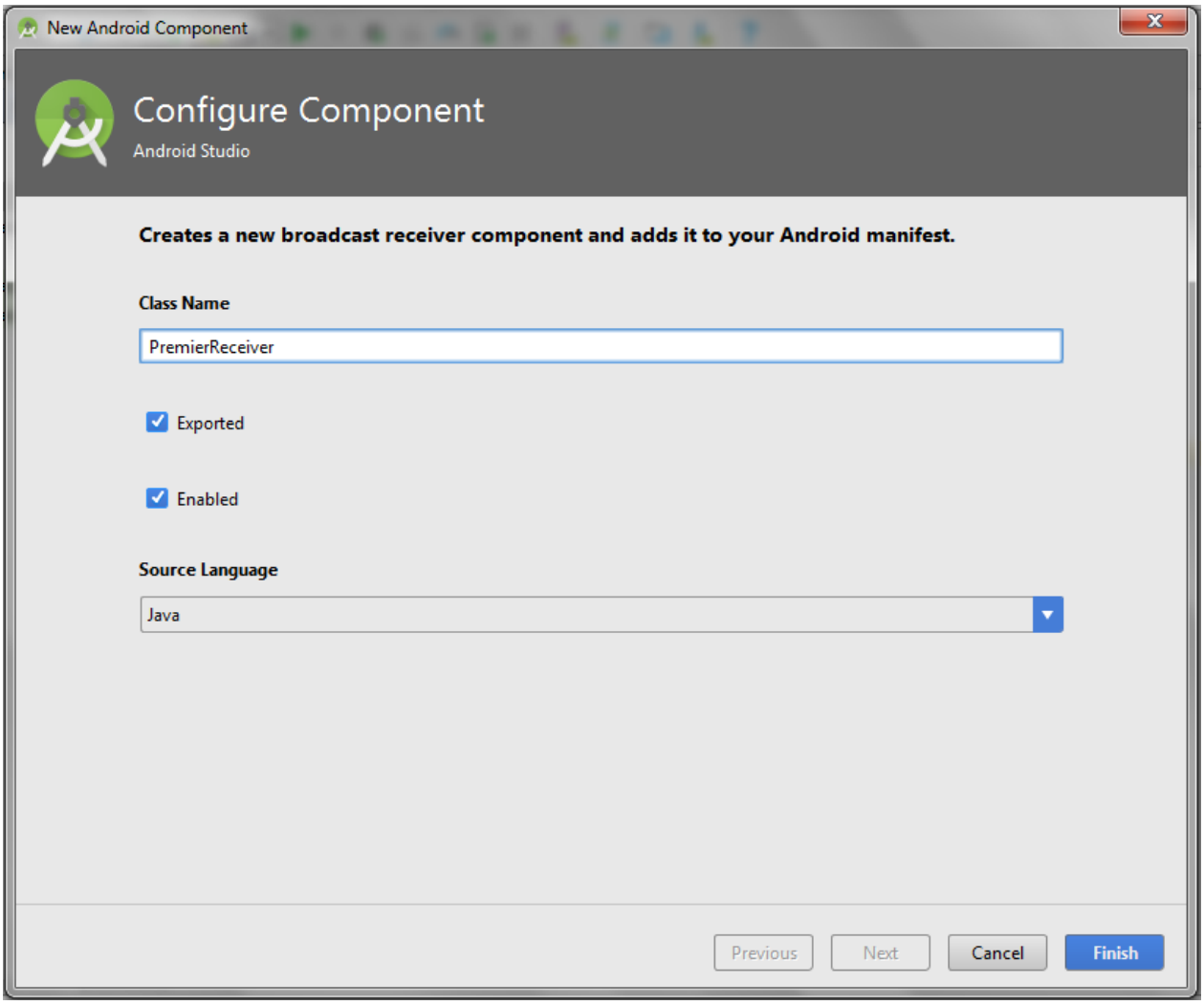
Ajouter Broadcast Receiver

Pour ajouter un Broadcast Receiver cliquez droite sur le dossier java puis New puis Other et choisissez Broadcast Receiver.

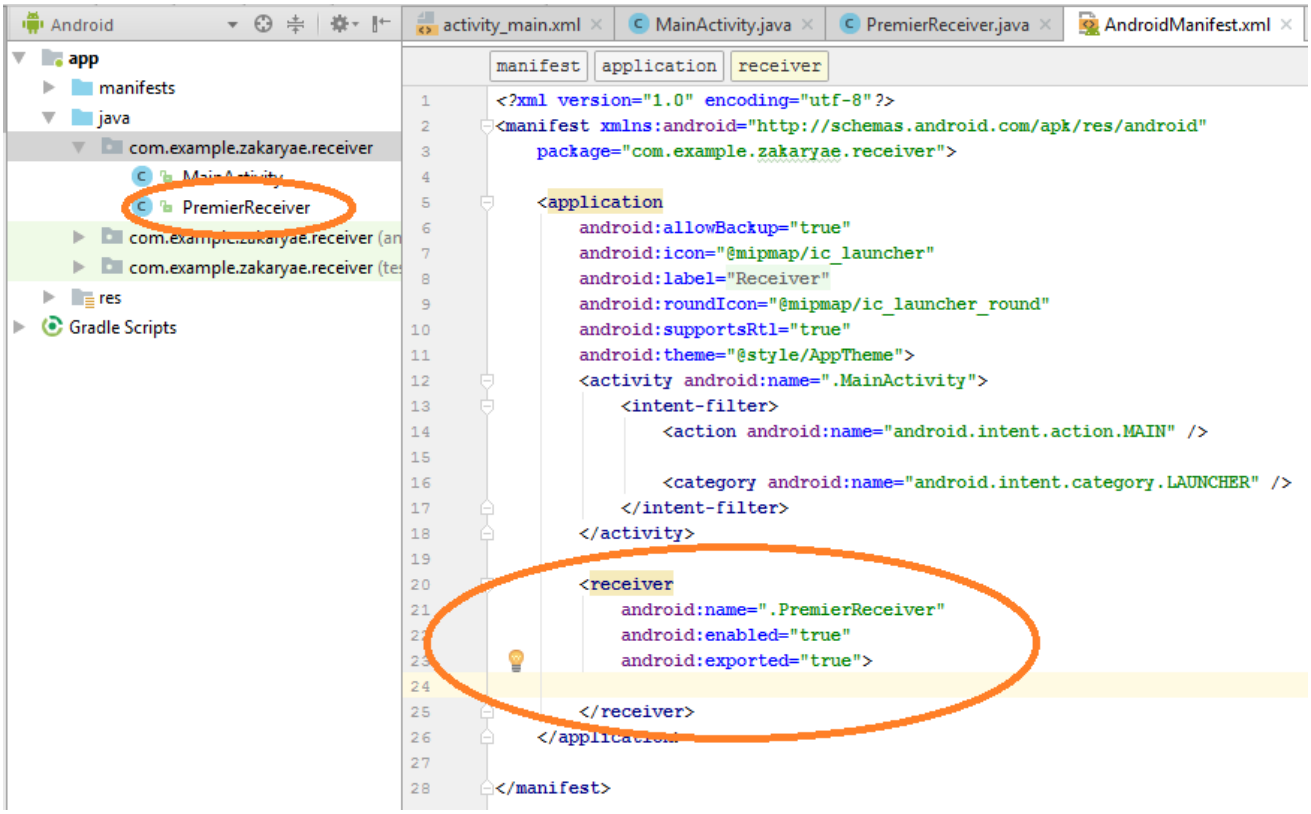


Après le choix de Broadcast Receiver une fenêtre va apparaitre pour mentionner le nom de Receiver.

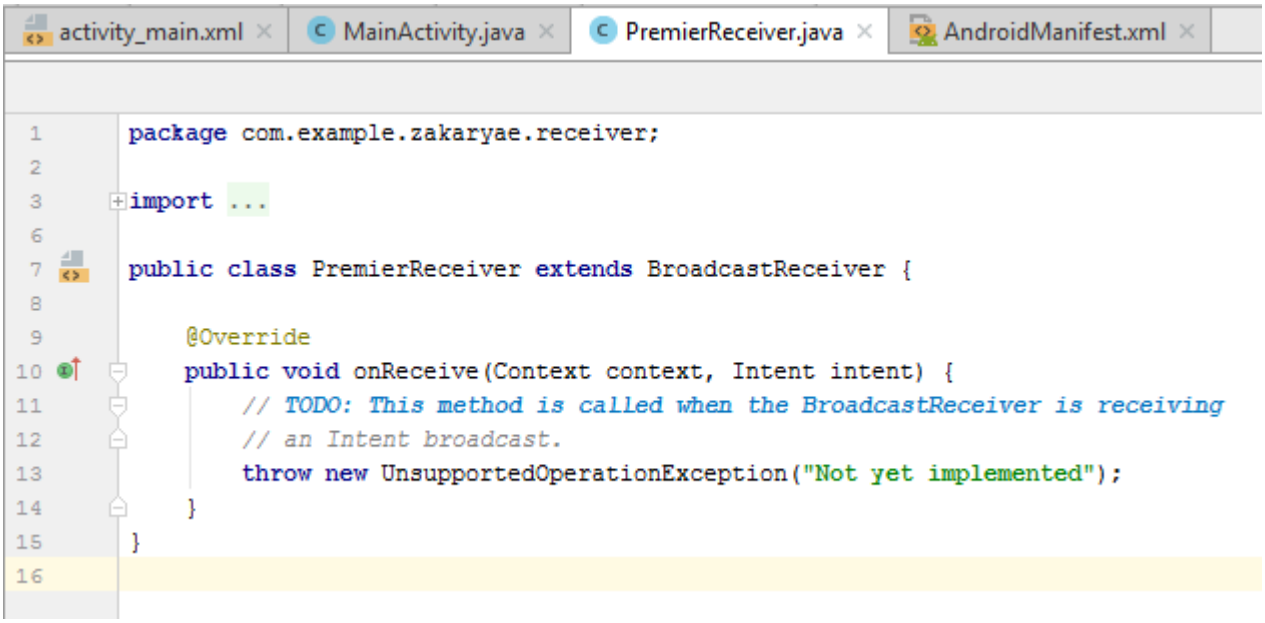
Ajouter le nom et Cliquer sur Finish (dans notre exemple le nom est « PremierReceiver »)



Lorsqu'on clique sur le bouton finish une classe java nommé PremierReceiver va être créé et le code du fichier AndroidManifest.xml sera modifier.



Class PremierReceiver



Dans la déclaration au niveau du fichier AndroidManifest.xml, un nom de classe est indiqué comme nom de ce Receiver. Cette classe doit obligatoirement hériter de la classe abstraite BroadcastReceiver où il y a une seule méthode à implémenter.

Un **BroadcastReceiver** ne vit que le temps de traiter votre **onReceive()**. L'instance peut être supprimée par le Garbage Collector.

Les receivers sont limités : ils ne peuvent pas ouvrir de boite de dialogue par exemple. Le système Android envoie l'intention à tous les **BroadCast Receiver** abonnées par ordre de priorité (priorité de votre BroadCast dans le fichier **AndroidManifest.xml**).

Exemple 1 :

Nous allons créer une classe "**SMSReceiver**", qui hérite de "**BroadcastReceiver**". Il vous demandera de rajouter les méthodes à implémenter **(OnReceive** dans notre cas).

Votre classe doit ressembler à ça :

```
import android.content.BroadcastReceiver;
import android.content.Context;
import android.content.Intent;

public class SMSReceiver extends BroadcastReceiver {

    @Override
    public void onReceive(Context context, Intent intent) {
        // TODO: This method is called when the BroadcastReceiver is receiving
        // an Intent broadcast.
        throw new UnsupportedOperationException("Not yet implemented");
    }
}
```

Donc pour recevoir les **messages** vous devez définir la permission ainsi que le Broadcast Receiver dans votre **AndroidManifest.xml**

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.zakaryae.receiver">
    <uses-permission android:name="android.permission.RECEIVE_SMS" />
    <uses-permission android:name="android.permission.READ_SMS" />
    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportsRtl="true"
        android:theme="@style/AppTheme">
        <activity android:name=".MainActivity">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
        <receiver
            android:name=".SMSReceiver"
            android:enabled="true"
            android:exported="true">
            <intent-filter android:priority="100">
                <action android:name="android.provider.Telephony.SMS_RECEIVED"/>
            </intent-filter>
        </receiver>
    </application>
</manifest>
```

Donc on déclare notre receiver, on peut lui définir une priorité dans la réception d'évènement (100 dans notre exemple) et dans la partie intent filtres, on définie que notre receiver est appelé quand le téléphone reçoit un SMS.

Revenons à notre classe, nous allons gérer la réception de notre évènement, donc dans la méthode OnReceive.

```
public class SMSReceiver extends BroadcastReceiver {
    private final String ACTION_RECEIVE_SMS=
"android.provider.Telephony.SMS_RECEIVED";

    @Override
    public void onReceive(Context context, Intent intent) {
        if (intent.getAction().equals(ACTION_RECEIVE_SMS))
        {

        }
    }
}
```

On déclare une chaîne de caractères qui représente l'action correspondante à l'évènement que l'on va recevoir et on teste si l'évènement reçu correspond bien à celui là.

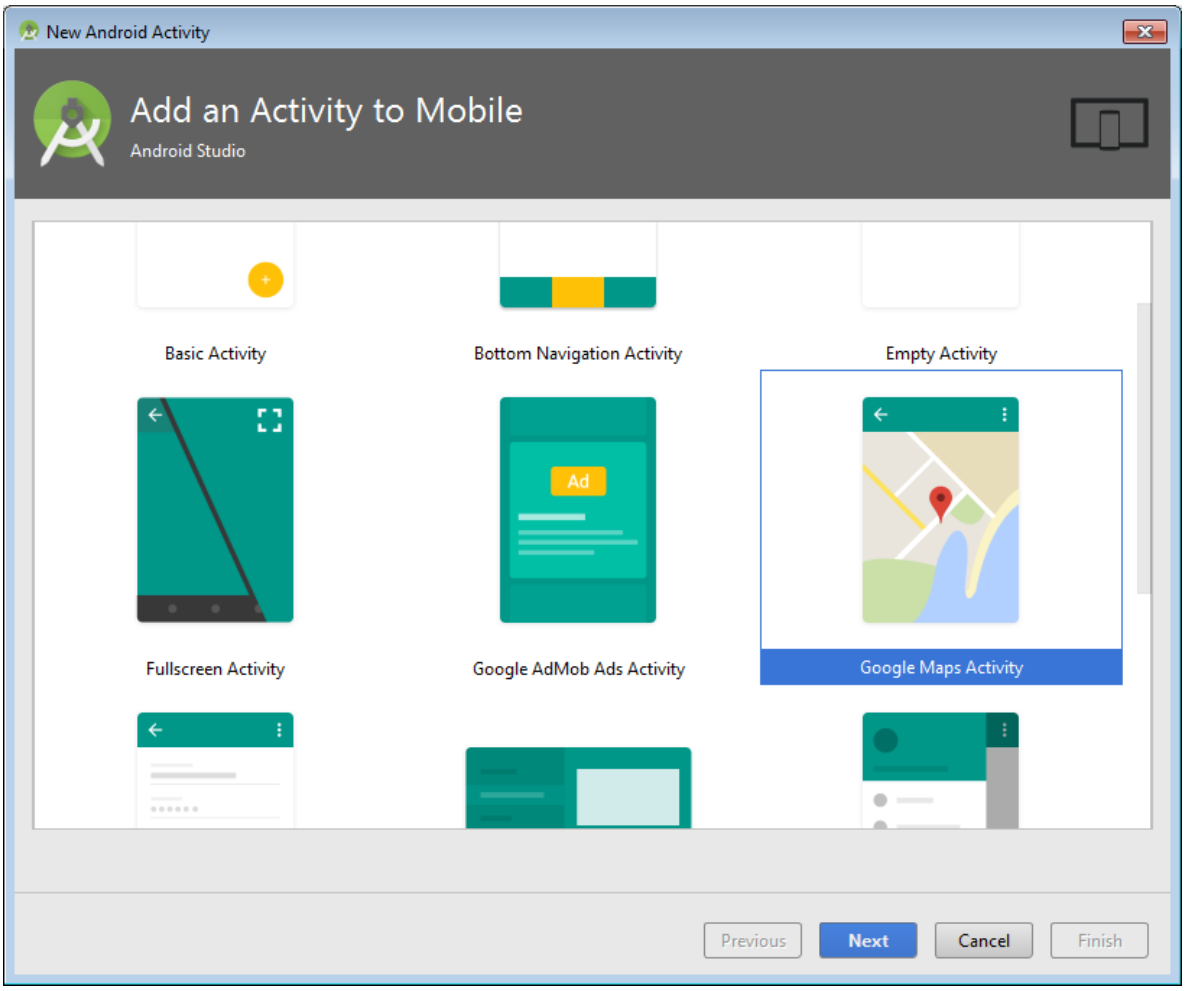
Maintenant mettons en place la lecture du dernier message reçu et son affichage sous forme de toast.

```
public class SMSReceiver extends BroadcastReceiver {
    private final String ACTION_RECEIVE_SMS=
"android.provider.Telephony.SMS_RECEIVED";

    @Override
    public void onReceive(Context context, Intent intent) {
        if (intent.getAction().equals(ACTION_RECEIVE_SMS))
        {
            Bundle bundle = intent.getExtras();
            if (bundle != null)
            {
                Object[] pdus = (Object[]) bundle.get("pdus");

                final SmsMessage[] messages = new SmsMessage[pdus.length];
                for (int i = 0; i < pdus.length; i++)
                {
                    messages[i] = SmsMessage.createFromPdu((byte[]) pdus[i]);
                }
                    if (messages.length > -1)
            {
                final String messageBody = messages[0].getMessageBody();
                final String phoneNumber =
messages[0].getDisplayOriginatingAddress();

Toast.makeText(context,"Expediteur : " + phoneNumber, Toast.LENGTH_LONG).show();
Toast.makeText(context, "Message : " + messageBody, Toast.LENGTH_LONG).show();

            }
            }
        }
    }
}
```

- On récupère le message grâce aux extras de l'i**ntent**.
- Puis on récupère dans le **bundle**, l'extra correspondant à l'identifiant "**pdus**"
- On récupère et on parcourt tous les messages pour obtenir le dernier.
- Puis on récupère les infos souhaitées dans le dernier message.

Intégration d'une Google Map sous Android

Si vous voulez crée une activity qui affiche une carte MAPS, ajouter une nouvelle activity de type **Google Maps Activity.**



Une fois le projet créé, Android Studio ouvre les fichiers google_maps_api.xml et MapsActivity.java dans l'éditeur. (Notez que votre activity peut avoir un nom différent, mais il s'agira du nom défini durant la configuration.) Notez que le fichier google_maps_api.xml contient des instructions sur comment obtenir une clé d'API Google Maps afin de pouvoir exécuter l'application. La section suivante explique plus en détail comment obtenir une clé d'API.

OBTENIR UNE CLÉ D'API GOOGLE MAPS

Votre application doit disposer d'une clé d'API pour pouvoir accéder aux serveurs Google Maps. Le type de clé dont vous avez besoin est une clé d'API avec une restriction pour les **applications Android**. Cette clé est gratuite. Vous pouvez l'utiliser avec n'importe quelle application qui appelle Google Maps Android API ; elle prend en charge un nombre illimité d'utilisateurs.

Choisissez **l'une des méthodes suivantes** pour obtenir votre clé d'API depuis Android Studio :

- **Méthode la plus simple et la plus rapide :** Utilisez le lien fourni dans le fichier `google_maps_api.xml` qu'Android Studio a créé pour vous :
    1. Copiez le lien fourni dans le fichier `google_maps_api.xml` et collez-le dans votre navigateur. Ce lien vous dirige vers la Google API Console et fournit les informations

requises pour la Google API Console via des paramètres d'URL, réduisant ainsi la saisie manuelle nécessaire de votre part.
2. Suivez les instructions pour créer un nouveau projet sur la Google API Console ou sélectionnez un projet existant.
3. Créez une clé d'API restreinte à Android pour votre projet.
4. Copiez la clé d'API obtenue, retournez dans Android Studio et collez la clé d'API dans l'élément <string> du fichier `google_maps_api.xml`.

- **Méthode un peu moins rapide :** Utilisez les informations d'identification fournies dans le fichier `google_maps_api.xml` qu'Android Studio a créé pour vous :
1. Copiez les informations d'identification fournies dans le fichier `google_maps_api.xml`.
2. Allez à la Google API Console dans votre navigateur.
3. Utilisez les informations d'identification copiées pour ajouter votre application à une clé d'API existante ou pour en créer une nouvelle.

EXAMINER LE CODE

Examinez le code fourni par le modèle. Observez plus particulièrement les fichiers suivants dans votre projet Android Studio.

Fichier XML de disposition (layout)

Par défaut, le fichier XML qui définit la disposition de l'application est `res/layout/activity_maps.xml`. Il contient le code suivant :

```
<fragment xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:map="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:id="@+id/map"
    android:name="com.google.android.gms.maps.SupportMapFragment"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context="com.example.zakaryae.mapstest.MapsActivity" />
```

Fichier Java d'activité des cartes

Par défaut, le fichier Java qui définit l'activité des cartes est appelé `MapsActivity.java`. Il doit contenir le code suivant après le nom du package :

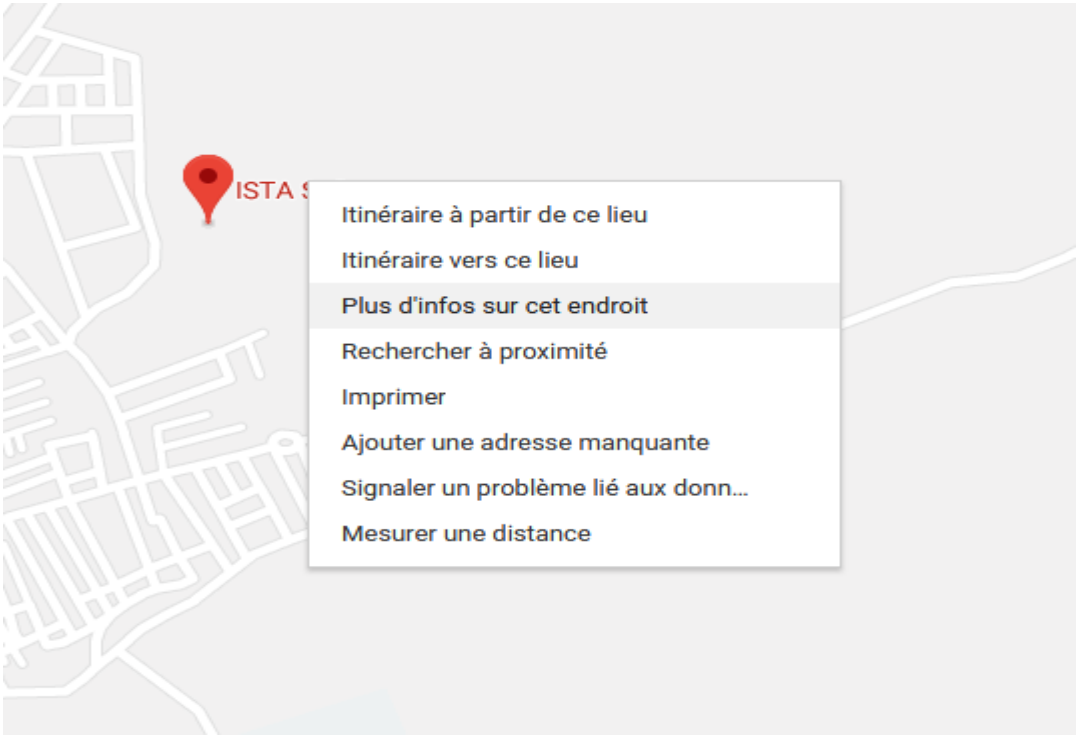
```
import android.support.v4.app.FragmentActivity;
import android.os.Bundle;

import com.google.android.gms.maps.CameraUpdateFactory;
import com.google.android.gms.maps.GoogleMap;
import com.google.android.gms.maps.OnMapReadyCallback;
import com.google.android.gms.maps.SupportMapFragment;
import com.google.android.gms.maps.model.LatLng;
import com.google.android.gms.maps.model.MarkerOptions;

public class MapsActivity extends FragmentActivity implements OnMapReadyCallback
{

    private GoogleMap mMap;

    @Override
```

```
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_maps);
        // Obtain the SupportMapFragment and get notified when the map is ready
to be used.
        SupportMapFragment mapFragment = (SupportMapFragment)
getSupportFragmentManager()
                .findFragmentById(R.id.map);
        mapFragment.getMapAsync(this);
    }


    /**
     * Manipulates the map once available.
     * This callback is triggered when the map is ready to be used.
     * This is where we can add markers or lines, add listeners or move the
camera. In this case,
     * we just add a marker near Sydney, Australia.
     * If Google Play services is not installed on the device, the user will be
prompted to install
     * it inside the SupportMapFragment. This method will only be triggered once
the user has
     * installed Google Play services and returned to the app.
     */
    @Override
    public void onMapReady(GoogleMap googleMap) {
        mMap = googleMap;
        // Add a marker in Sydney and move the camera
        LatLng sydney = new LatLng(-34, 151);
        mMap.addMarker(new MarkerOptions().position(sydney).title("Marker in
Sydney"));
        mMap.moveCamera(CameraUpdateFactory.newLatLng(sydney));
    }
}
```

modifier le point d'affichage par défaut

Si vous avez remarqué le point par défaut affiché par notre MAPS est un point de Sydney, mais je peux mettre le point que je voulais, pour cela ouvrer Google Maps et chercher le point que vous voulez : par exemple je vais choisir ISTA SEFROU.

Cliquer droite sur le point et choisir : plus d'infos sur cet endroit



Copier les coordonnées qui vont s'afficher et remplacer :

```
LatLng sydney = new LatLng(-34, 151);
```

Par les nouveaux cordonnés :

```
LatLng sydney = new LatLng(33.836264, -4.820212);
```

Vous pouvez aussi changr le mot explicatif qui s'affiche avec le point on modifiant :

```
mMap.addMarker(new MarkerOptions().position(sydney).title("Marker in Sydney"));
```

"Marker in Sydney" est le mot qui nous devons changer, par exemple :

```
mMap.addMarker(new MarkerOptions().position(sydney).title("CFP DEFROU"));
```

Ce qui donne :

```
@Override
public void onMapReady(GoogleMap googleMap) {
    mMap = googleMap;
    // Add a marker in Sydney and move the camera
    LatLng sydney = new LatLng(33.836264, -4.820212);
    mMap.addMarker(new MarkerOptions().position(sydney).title("CFP SEFROU"));
    mMap.moveCamera(CameraUpdateFactory.newLatLng(sydney));
}
```

Niveau de zoom

Maintenant que tout est prés on peut faire un zoom pour que notre carte Maps s'affiche avec l'emplacement de notre point et pas toute le continent, pour cela veuillez modifier cette ligne :

```
mMap.moveCamera(CameraUpdateFactory.newLatLng(sydney));
```

Par celle ci :

```
mMap.moveCamera(CameraUpdateFactory.newLatLngZoom(sydney,15));
```

La valeur 15 est la valeur de zoom, vous pouvez la modifier afin de trouvez la vision qui vous plait.

Modifier le type de la carte

Comme vous savez google propose deux type de carte normal et satellite, jusqu'à ici la carte que nous affichons et la carte normal, si nous voulons modifier ce type alors on ajoute :

```
mMap.setMapType(GoogleMap.MAP_TYPE_HYBRID);
```

Et pour la modifier au type normal :

```
mMap.setMapType(GoogleMap.MAP_TYPE_NORMAL);
```

LE CALQUE MA POSITION

Vous pouvez utiliser le calque et le bouton Ma position pour permettre à l'utilisateur de connaître sa position actuelle sur la carte.

Remarque : Avant d'activer le calque Ma position, vous devez vous assurer que vous disposez de l'autorisation de géolocalisation à l'exécution nécessaire.

Pour activer le calque Ma position sur la carte, procédez comme suit :

```
mMap.setMyLocationEnabled(true);
```

Une fois que le calque Ma position est activée, le bouton Ma position apparaît dans l'angle supérieur droit de la carte. Dès qu'un utilisateur clique sur ce bouton, l'appareil photo centre la carte sur la position actuelle de l'appareil, si celle-ci est connue. Cette position est indiquée sur la carte par un petit point bleu si l'appareil est fixe, ou sous forme de chevron si l'appareil est en mouvement.

Vous pouvez empêcher l'apparition du bouton en appelant

```
mMap.setMyLocationEnabled(false);
```

Notez que le calque Ma position ne renvoie aucune donnée. Si vous souhaitez accéder aux données de géolocalisation par programmation, utilisez l'API de géolocalisation.

Echantillon du code avec permission

```
if
(ContextCompat.checkSelfPermission(this,Manifest.permission.ACCESS_FINE_LOCATION)
        == PackageManager.PERMISSION_GRANTED)
{
    mMap.setMyLocationEnabled(true);
}
```

API DE GÉOLOCALISATION

Pour pouvoir **géocaliser** un utilisateur, il faut que vous possédiez une instance de la classe **LocationListener** ou que l'activité incluant votre carte implémente l'interface **LocationListener**. Nous allons opter pour la seconde solution et faire en sorte que la classe **HelloGoogleMapActivity** implémenter cette interface.

Ainsi afin de pouvoir s'abonner à la mise à jour des coordonnées de l'utilisateur, il faut utiliser la méthode **requestLocationUpdates(String, long, float, LocationListener)**.
Cette méthode posséde 4 arguments :

- Le provider utiliser pour recevoir les mises à jour des coordonnées utilisateurs (GPS / NETWORK …)
- Un interval minimum entre deux notifications (en millisecondes)
- Un interval minimum entre deux notifications (en metre)
- L'instance de votre LocationListener

Il est conseillé de s'abonner aux mises à jour des coordonnées de l'utilisateur dans la méthode **onResume** et de se désabonner dans la méthode **onStop** afin de stopper l'utilisateur des ressources de localisation alors que l'application n'en a plus l'utilité.

```
private LocationManager lm;
private double latitude;
private double longitude;
private double altitude;
private float accuracy;

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

@Override
protected void onResume() {
    super.onResume();
    lm = (LocationManager) this.getSystemService(LOCATION_SERVICE);
    if (lm.isProviderEnabled(LocationManager.GPS_PROVIDER))
    lm.requestLocationUpdates(LocationManager.GPS_PROVIDER, 10000, 0, this);
    lm.requestLocationUpdates(LocationManager.NETWORK_PROVIDER, 10000, 0,this);
}

@Override
protected void onPause() {
    super.onPause();
    lm.removeUpdates(this);
}
```

Implémenter l'interface **LocationListener** équivaut à surcharger les 4 méthodes suivantes :

- **OnProviderEnabled(String provider)** : Cette méthode est appelée quand une source de localisation est activée (GPS, 3G..etc). L'argument représente le nom de la source activée. Vous pouvez par exemple vous abonner à la mise à jour de localisation via cette source.
- **OnProviderDisabled(String provider)** : Cette méthode est appelée quand une source de localisation est désactivée(GPS, 3G..etc). L'argument est le nom de la source désactivée. Vous pouvez par exemple vous désabonner à la mise à jour de localisation via cette source.
- **OnStatusChanged(String provider, int status, Bundle extras)** : Appeler quand le status d'une source change. Il existe 3 statuts (OUT_OF_SERVICE, TEMPORARILY_UNAVAILABLE , AVAILABLE). Ce qui donnera par exemple :

```
@Override
public void onStatusChanged(String provider, int status, Bundle extras) {
    String newStatus = "";
    switch (status) {
        case LocationProvider.OUT_OF_SERVICE:
            newStatus = "OUT_OF_SERVICE";
            break;
        case LocationProvider.TEMPORARILY_UNAVAILABLE:
            newStatus = "TEMPORARILY_UNAVAILABLE";
            break;
        case LocationProvider.AVAILABLE:
            newStatus = "AVAILABLE";
            break;
    }
 Toast.makeText(this, "Le nouveau status de "+ provider + " est : "+newStatus,
Toast.LENGTH_SHORT).show();}
```

- **OnLocationChanged(Location location)** : Cette méthode est appelée quand la **localisation** de l'utilisateur est mise à jour. L'argument représente la nouvelle position. Vous pouvez récupérer plusieurs informations comme la latitude, longitude, altitude et précision (en mètre). Ce qui donnera :

```
@Override
public void onLocationChanged(Location location) {
    latitude = location.getLatitude();
    longitude= location.getLongitude();
    altitude = location.getAltitude();
    accuracy = location.getAccuracy();
    String msg = "Lattude = " + latitude + " Longitude =" +  longitude + "
       Altitude =" + altitude +"Précision =" + accuracy;
    TextView txt=(TextView) findViewById(R.id.txt1);
    txt.setText(msg);       }
```

Petite explications :

- Nous avons besoin d'une instance de la classe **LocationManager**. Cette dernière donne accès au service de localisation. Cela permet aux applications d'obtenir des mises à jour périodiques de la **position** de l'utilisateur.
- On appele la méthode **requestLocationUpdate** pour la mise à jour via le Réseau (NetWork) et le GPS.
- On implémente les méthodes obligatoires de **LocationListener**.

Sans oublier les permissions suivantes :

```
<uses-permission android:name="android.permission.INTERNET" />
<uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />
<uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION" />
```

- **INTERNET** : Pour télécharger et afficher les cartes
- **ACCESS_FINE_LOCATION** : Pour pouvoir localiser précisément l'utilisateur (GPS)
- **ACCESS_COARSE_LOCATION** : Pour pouvoir localiser approximativement l'utilisateur (WIFI, 3G..)