La ToolBar

Table des matières

Alors aujourd'hui on migre l'ActionBar et on la remplace par la ToolBar en quelques lignes de code.

Bien sûr, vous serez compatible du level Gingerbread au level M.

1 Migrer son ActionBar vers la ToolBar

1.1 Mise en place du style

Tout d'abord, il suffit de définir votre style dans le fichier res/values/styles.xml (ci dessous). Il hérite de Theme.AppCompat et affecte la valeur true à la balise windowNoTitle et false à la valeur windowActionBar, ce qui signifie je ne souhaite ni la barre de titre ni l'actionBar pour mon activité.

```
<resources>
   <style name="AppBlankTheme" parent="Theme.AppCompat.Light.DarkActionBar">
      <!-- Main theme colors -->
      <!--   your app branding color for the app bar -->
      <item name="colorPrimary">@color/primary</item>
      <!--   darker variant for the status bar and contextual app bars -->
      <item name="colorPrimaryDark">@color/primary_dark</item>
      <!--   theme UI controls like checkboxes and text fields -->
      <item name="colorAccent">@color/accent</item>
   </style>
   <!--Both themes below are those accepted to make the ToolBar works-->
   <!-- Base application theme. -->
   <style name="AppTheme" parent="AppBlankTheme">
      <!-- Customize your theme here. -->
      <!-- Base application theme. -->
         <item name="windowNoTitle">true</item>
         <item name="windowActionBar">false</item>
      <item name="spinnerDropDownItemStyle">@style/mydropdown</item>
   </style>

</resources>
```

J'ai défini aussi les couleurs de mon activité. Cela vient de la release Lollipop de la support Library qui (grace à Chris Banes) nous permet simplement de définir un jeu de couleur uni pour notre activité. Ces couleurs seront automatiquement utilisées par le Theme.AppCompat au niveau de votre application de manière à avoir une charte colorimétrique uniforme. Vous pouvez allez vous faire plaisir ici pour choisir vos couleurs et voir l'effet que cela produit: http://www.materialpalette.com/

Maintenant que nous avons défini notre style, appliquons le. Comme d'habitude cela s'effectue au niveau du fichier Manifest :

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
   package="com.android2ee.formation.lollipop.toolbar" >
   <application
      android:allowBackup="true"
      android:icon="@mipmap/ic_launcher"
      android:label="@string/app_name"
      android:theme="@style/AppTheme" >
```

1.2 Modification du Layout de l'activité

C'est ici que tout se joue (oui, tout, la survie de l'univers, l'existence de dieu, tout:) dans votre layout. L'actionBar n'est plus ajouté par défaut par le framework (on a demandé explicitement d'utiliser un theme pour notre application qui n'a pas l'actionBar), c'est nous qui la positionnons dans notre layout. C'est un changement profond dans la philosophie de l'ActionBar, sommes nous suffisament mature pour la positionner où il se doit, en respect des nouvelles règles de Design ?

Pour les règles de design, c'est ici : http://www.google.com/design/spec/material-design/introduction.html et j'ajouterai, qu'en tant que développeur Android, il est de votre devoir d'aller y perdre une ou deux de vos soirées. Et puis le site est tellement bien fait, tellement riche que ce serait du gachis de ne pas l'utiliser.

Donc tout ça pour vous dire de mettre à jour votre layout comme suit:

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    tools:context=".MainActivity" >
    <android.support.v7.widget.Toolbar
        android:id="@+id/toolbar"
        android:minHeight="?attr/actionBarSize"
        android:background="#2196F3"
        android:layout_width="match_parent"
        android:layout_height="wrap_content">
    </android.support.v7.widget.Toolbar>
    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/hello_world" />
    <ListView android:id="@+id/sampleList"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_margin="5dp"
        android:animateLayoutChanges="true"
        android:drawSelectorOnTop="true"></ListView>

</LinearLayout>
```

Alors, clairement, je vous montre le layout du tutorial, la chose importante ici est le linearLayout de premier niveau suivit directement par la ToolBar puis le contenu réel de votre activité.

La seule chose qui diffère d'un composant normal est le minHeight qui récupère la taille de l'ActionBar du thème habituel.

1.3 Mise à jour du fichier Java de l'activité (ou du fragment)

Vous avez quasiment fini de migrer votre application, il ne vous reste que deux petites choses à faire :

1. Votre activité hérite d'AppCompatActivity
2. Définir la ToolBar comme étant l'actionBar de l'activité

Et cela s'effectue tout simplement :

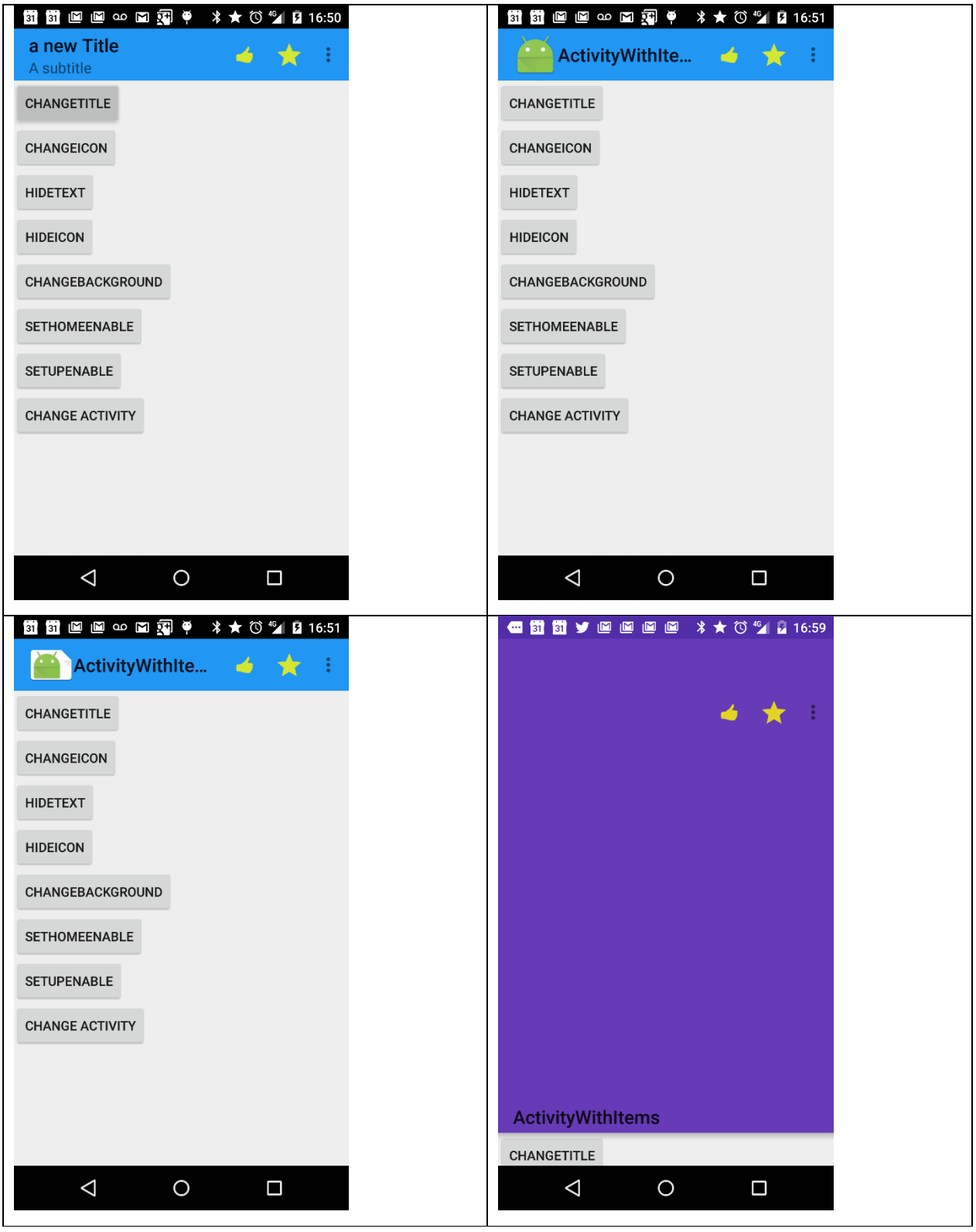
```
public class MainActivityextends AppCompatActivity {
   ListView sampleList;


   @Override
   protected void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.activity_main);
      Toolbar toolbar = (Toolbar) findViewById(R.id.toolbar);
      setSupportActionBar(toolbar);
      getSupportActionBar().setSubtitle("Using ToolBar");
```

1.4 Vérification de la version de la support library

Pour que votre code s'exécute proprement, vous devez utiliser la support library de niveau (au moins) 22.2.0 :

```
compile 'com.android.support:appcompat-v7:22.2.0'
```



1.5 Rappel sur votre fichier Xml de menu

Depuis que vous utilisez l'appCompat (et avant cela la SherlockBar), vos fichiers de menu définnissent un nouveau name space pour être interprettable par la support library, sinon la baslie showAsAction est juste ignorée et vos icônes absents.

Il vous faut donc définir ce namespace et l'utiliser pour toutes les baslies qui n'appartiennent pas aux vieilles versions:

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

<item
android:id="@+id/action_one"
android:icon="@drawable/ic_action_one"
android:orderInCategory="0"
actionbarcompat_mse:showAsAction="always"
android:title="One"/>
<item
android:id="@+id/action_two"
android:icon="@drawable/ic_action_two"
android:orderInCategory="1"
actionbarcompat_mse:showAsAction="always"
android:title="Two"/>
<item
android:id="@+id/action_one_duplicate"
android:icon="@drawable/ic_action_one"
android:orderInCategory="0"
actionbarcompat_mse:showAsAction="always"
android:title="One2"/>
<item
android:id="@+id/action_two_duplicate"
android:icon="@drawable/ic_action_two"
android:orderInCategory="0"
actionbarcompat_mse:showAsAction="ifRoom"
android:title="Two2"/>
<item
android:id="@+id/action_settings"
android:orderInCategory="100"
actionbarcompat_mse:showAsAction="never"
android:title="@string/action_settings"/>

</menu>
```

2 Action Mode

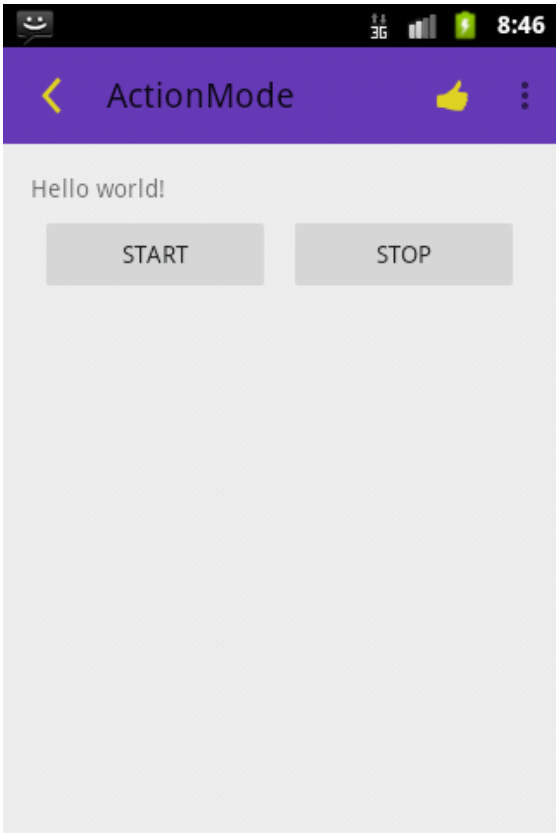
Continuons notre découverte de la ToolBar et en particulier regardons comment mettre l'ActionMode en place.

Tout d'abord qu'est-ce que l'ActionMode ? C'est la capacité de changer notre ToolBar pour n'afficher que des actions lors d'un évènement utilisateur particulier. L'exemple compréhensible est quand je sélectionne un mail, la ToolBar affiche de nouvelles actions.

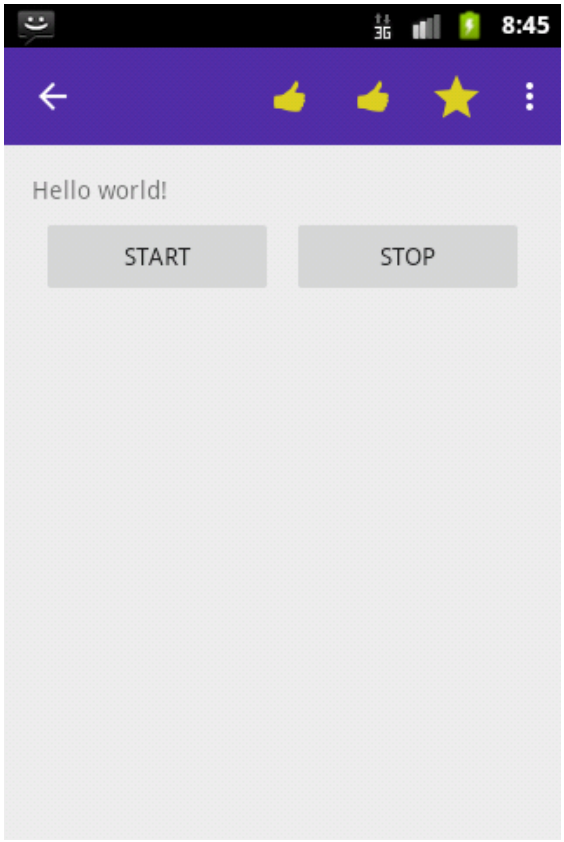
C'est ce changement que l'on nomme l'ActionMode et grosso modo, cela correspond juste à demander à notre ToolBar d'afficher un nouveau fichier xml de Menu. Oui, en fait c'est trivial. Avant ça l'était

vraiment, maintenant à deux trois petits détails, ça l'est toujours. Regardons comment le mettre en place:

ActionMode Off



ActionMode On



Le code Java et la définition des menus n'a pas changé (non, toi non plus tout n'a pas changé - Julio Iglesias)

2.1 Fichiers Xml des menus

Comme vous l'avez compris, il nous faut deux menus xml pour mettre en place l'ActionMode. Le premier est l'OptionMenu naturel de votre activité (fragment) et le second celui que vous souhaitez afficher lorsque l'ActionMode est activé.

Il est évident que c'est vos menus, vous mettez les items que vous souhaitez afficher, mes fichiers sont des fichiers d'exemples :)=

res\menu\main_menu.xml (Votre menu normal)

```
<menu xmlns:android="http://schemas.android.com/apk/res/android" >
<item
android:id="@+id/action_settings"
```

```
android:orderInCategory="100"
actionbarcompat:showAsAction="never"
android:title="@string/action_settings"/>
</menu>
```

res\menu\action_mode.xml (Votre menu lorsque l'ActionMode est activé)

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

<item
android:id="@+id/action_one"
android:icon="@drawable/ic_action_one"
android:orderInCategory="0"
actionbarcompat_mse:showAsAction="always"
android:title="One"/>
<item
android:id="@+id/action_two"
android:icon="@drawable/ic_action_two"
android:orderInCategory="1"
actionbarcompat_mse:showAsAction="always"
android:title="Two"/>
<item
android:id="@+id/action_one_duplicate"
android:icon="@drawable/ic_action_one"
android:orderInCategory="0"
actionbarcompat_mse:showAsAction="always"
android:title="One2"/>
<item
android:id="@+id/action_two_duplicate"
android:icon="@drawable/ic_action_two"
android:orderInCategory="0"
actionbarcompat_mse:showAsAction="ifRoom"
android:title="Two2"/>
<item
android:id="@+id/action_settings"
android:orderInCategory="100"
actionbarcompat_mse:showAsAction="never"
android:title="@string/action_settings"/>

</menu>
```

Comme vous le constatez, il n'y a aucun changement dans ces fichiers xml par rapport aux fichiers xml que nous utilisions pour l'ActionBarCompat.

2.2 Fichier du Layout

Comme expliqué dans le chapitre précédent, il faut que vous déclariez votre ToolBar dans le fichier de Layout de votre activité pour que celle ci apparraissent:

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

```
    android:layout_height="match_parent"
    android:orientation="vertical"
    tools:context=".ActionModeActivity" >


    <include layout="@layout/my_toolbar"/>


    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:orientation="vertical"
        android:paddingBottom="@dimen/activity_vertical_margin"
        android:paddingLeft="@dimen/activity_horizontal_margin"
        android:paddingRight="@dimen/activity_horizontal_margin"
        android:paddingTop="@dimen/activity_vertical_margin">
```

où le fichier res\layout\my_toolbar.xml est

```
<android.support.v7.widget.Toolbar xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/toolbar"
    android:minHeight="?attr/actionBarSize"
    android:background="?attr/colorPrimary"
    android:layout_width="match_parent"
    android:layout_height="wrap_content">
</android.support.v7.widget.Toolbar>
```

2.3 Code Java

Comme toujours avec la ToolBar, vous devez étendre l'AppCompatActivity et définir votre ToolBar comme étant l'ActionBar de votre activité:

```
public class ActionModeActivity extends AppCompatActivity {
    /**
     * The actionMode
     */
    ActionMode mMode;
   /**
    * The action Bar
    */
   private ActionBar actionBar;
    /**
     * The ToolBar
     */
    private Toolbar toolbar;
    /***
     * The ActionMode callBack
     */
    Callback actionModeCallBack;
    /***
     * Boolean to know which version is running
     */
    private boolean postICS,postLollipop;
```

```
  @Override
  protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_action_mode);
     //find the toolbar
    toolbar = (Toolbar) findViewById(R.id.toolbar);
     //customize the toolbar
     toolbar.setNavigationIcon(R.drawable.ic_action_custom_up);
    postICS =getResources().getBoolean(R.bool.postICS);
    postLollipop =getResources().getBoolean(R.bool.postLollipop);
    if(postLollipop){
      toolbar.setElevation(15);
    }
     //define the toolbar as the ActionBar
    setSupportActionBar(toolbar);
    actionBar=getSupportActionBar();
     //You could also hide the action Bar
     //getSupportActionBar().hide();

     // Show the Up button in the action bar.
     actionBar.setDisplayUseLogoEnabled(false);
     actionBar.setDisplayHomeAsUpEnabled(true);
     setListeners();
     //initialize the actionMode callback
    initializeActionModeCallBack();
}
```

Ensuite il vous faut initialiser votre ActionModeCallBack qui est le callback du cycle de vie de l'ActionMode:

```
private void initializeActionModeCallBack() {
   actionModeCallBack=new  android.support.v7.view.ActionMode.Callback() {
      @Override
      public boolean onPrepareActionMode(ActionMode mode, Menu menu) {
         return false;
      }

@Override
public void onDestroyActionMode(ActionMode mode) {
}

@Override
public boolean onCreateActionMode(ActionMode mode, Menu menu) {
getMenuInflater().inflate(R.menu.action_mode, menu);
return true;
}

@Override
public boolean onActionItemClicked(ActionMode mode, MenuItem item) {
Toast.makeText(ActionModeActivity.this, "Got click: " + item, Toast.LENGTH_SHORT).show();
mode.finish();
return true;
}
};
}
```

Ce callback possède 4 méthodes:

- La méthode onCreateActionMode qui permet de créer le menu de l'ActionMode. Il suffit ainsi d'inflater le fichier xml que l'on souhaite;
- La méthode onPrepareActionMode qui nous permet de mettre à jour le menu avant que celui soit affiché;
- La méthode onDestroyActionMode qui nous permet de faire le ménage si besoin est;
- La méthode onActionItemClicked qui nous permet de savoir qu'un menu item du menu de l'ActionMode à été cliqué et lequel.

Il ne nous reste plus qu'à savoir activer ou désactiver ce mode:

Pour l'activation (remarquez que mMode est une variable de classe) :

```
mMode = startSupportActionMode(actionModeCallBack);
```

Pour la désactivation:

```
mMode.finish();
```

Et voilà, c'est tout, vous pouvez activer ou désactiver l'ActionMode quand bon vous semble maintenant.

3 Gestion du styles

Il faut spécifiquement rajouter au style de la ToolBar les valeurs suivantes:

res\values\myStyles.xml

```
<!--The Theme for the Actvity that have actionMode-->
   <style name="ActionModeAppTheme" parent="AppTheme">
      <item name="windowActionModeOverlay">true</item>
      <item name="actionModeBackground">@color/primary_dark</item>
   </style>
```

C'est à dire, je ne souhaite pas voir l'ActionMode usuel (parce que sinon il apparait au-dessus de votre ToolBar et c'est très laid), je souhaite avoir une couleur particulière de fond pour ma ToolBar quand l'ActionMode est activé et j'utilise le Theme.Light pour surcharger mon thème actuel.

Où le style de la ToolBar ressemble à:

res\values\myStyles.xml

```
<style name="AppBlankTheme" parent="Theme.AppCompat.Light.DarkActionBar">
   <!-- Main theme colors -->
   <!--   your app branding color for the app bar -->
   <item name="colorPrimary">@color/primary</item>
   <!--   darker variant for the status bar and contextual app bars -->
   <item name="colorPrimaryDark">@color/primary_dark</item>
   <!--   theme UI controls like checkboxes and text fields -->
   <item name="colorAccent">@color/accent</item>
```

```
</style>
<!--Both themes below are those accepted to make the ToolBar works-->
<!-- Base application theme. -->
<style name="AppTheme" parent="AppBlankTheme">
    <!-- Customize your theme here. -->
    <!-- Base application theme. -->
        <item name="windowNoTitle">true</item>
        <item name="windowActionBar">false</item>
    <item name="spinnerDropDownItemStyle">@style/mydropdown</item>
</style>
```

Et l'appliquer à votre activité (dans votre Manifest)

manifest.xml

```
<activity
    android:name=".sample.ActionModeActivity"
    android:label="@string/title_activity_actionmode"
    android:parentActivityName=".MainActivity"
    android:theme="@style/ActionModeAppTheme" >
    <meta-data
        android:name="android.support.PARENT_ACTIVITY"
        android:value="com.android2ee.formation.lollipop.toolbar.MainActivity" />
    <intent-filter>
      <action android:name="android.intent.action.MAIN" />
      <category android:name="com.android2ee.formation.lollipop.toolbar.EXAMPLE" />
    </intent-filter>
</activity>
```

4 Remarques concernant la Support library et l'ActionBarCompat

Si vous penser que vous pouvez encore utiliser l'ActionBarCompat, vous faîtes une erreur, il vous faut migrer.

Pourquoi ? Pour ça:

```
 compileSdkVersion 21
...
    defaultConfig {
        applicationId "com.android2ee.formation.librairies.actionbar.compat"
        minSdkVersion 9
        targetSdkVersion 21
        ...
    }
   dependencies {
compile 'com.android.support:appcompat-v7:20.0.0'
}
```

Si vous souhaitez que l'ActionBar marche, il vous faut figer le compileSdk, le TargetSdk et la SupportLib à 21.

Mais si vous souhaitez que l'Overlay (votre contenu passe sous l'Action), que le SplitWhenNarrow ou que les ProgressBar marche encore, vous devez figer tout ce petit monde à 20.

Ce qui veut dire qu'en fait, vous devez migrer :)

5 ActionView

Alors pour continuer à visiter l'ActionBar... euh non, la ToolBar, je vous propose d'apprendre à afficher une vue dans votre ToolBar. L'exemple que j'utilise souvent pour ça est de mettre une zone de recherche, même si ce n'est pas très pertinent car la SearchWidget est un peu là pour ça.

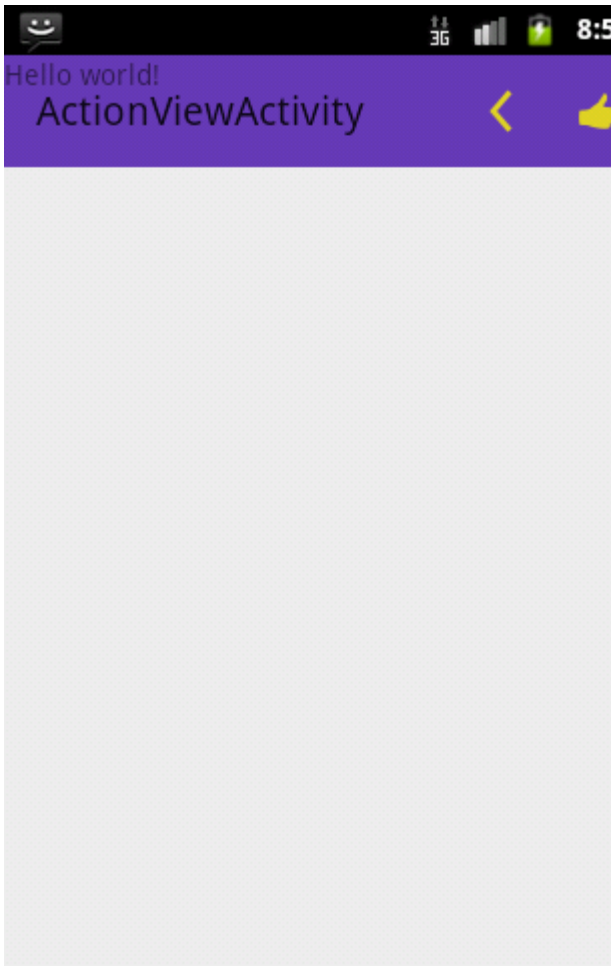
Mais, ce n'est pas grave, je le fais quand même pour vous expliquer le comportement.

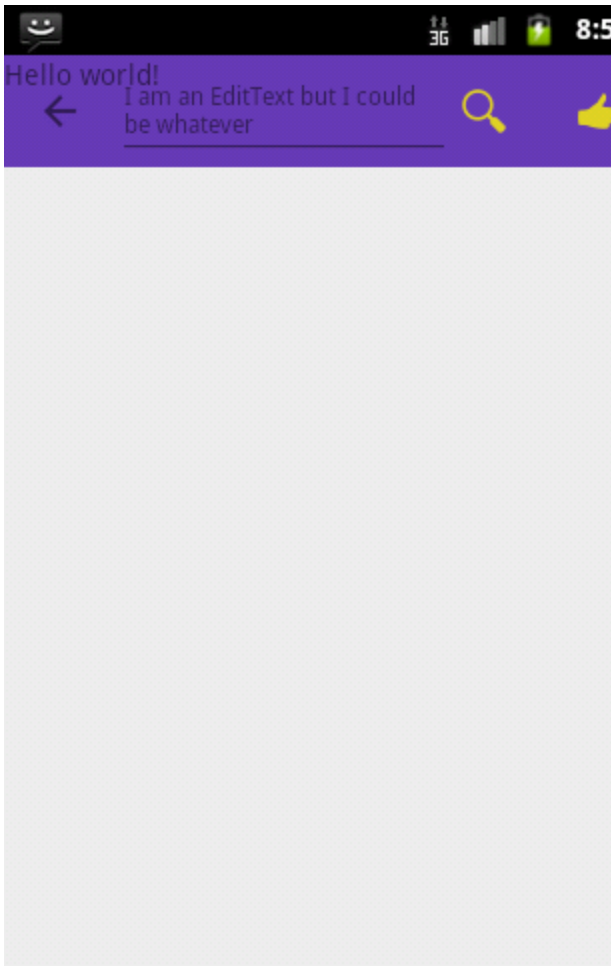
Tout d'abord, à quoi cela ressemble-t-il ?

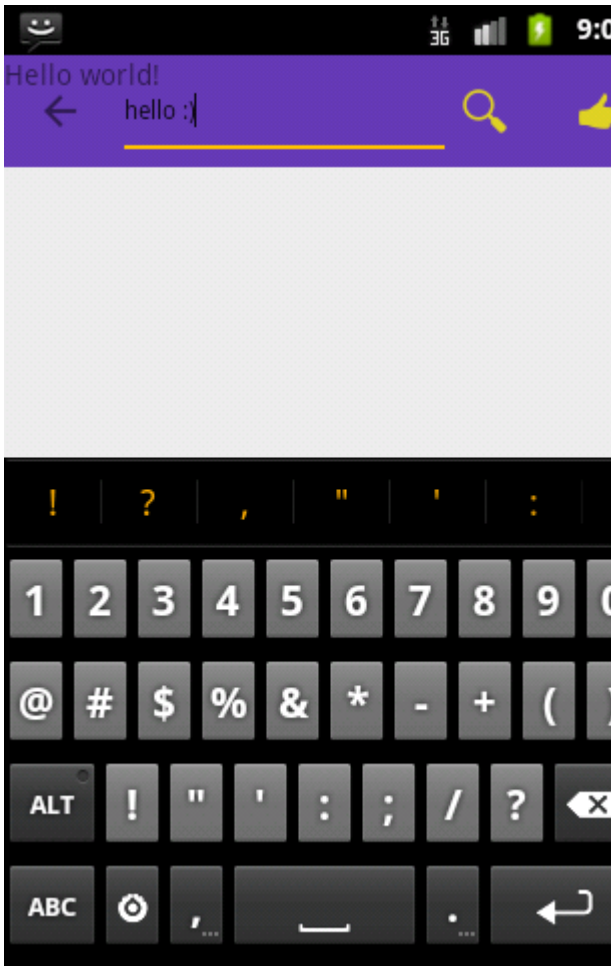
Lorsque j'appuie sur l'icone (le MenuItem) "<" dans la ToolBar, celui-ci se déplie et affiche une vue avec une fléche <-, une EditText et une ImageButton qui montre une loupe. Lorsque j'appuie sur le bouton loupe, cela replie la vue et récupère la valeur contenue dans l'EditText. Dans mon exemple, je l'affiche dans un Toast.

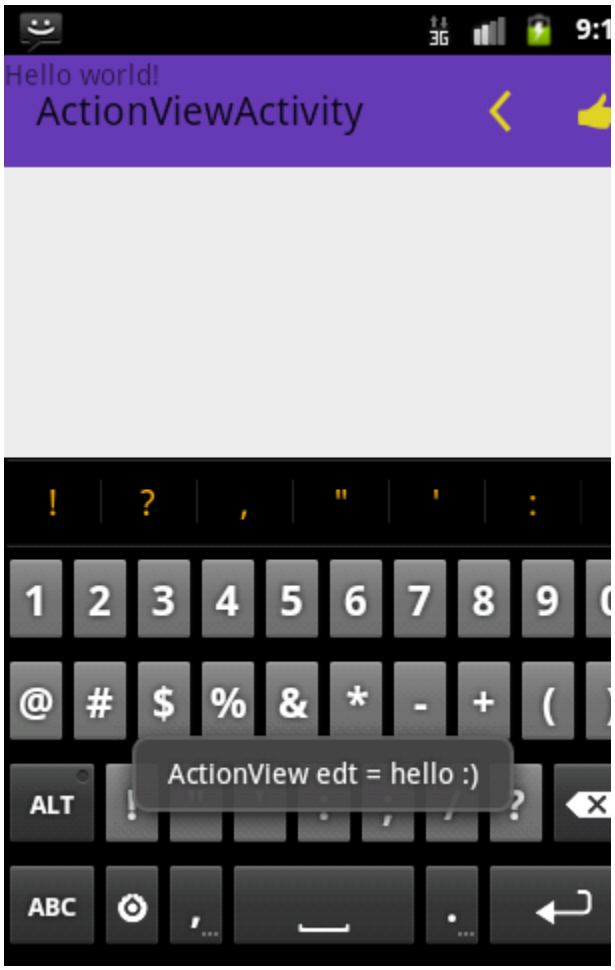
La flèche "<-" est nativement ajoutée par le système, elle ne dépend pas de vous.

Dans les copies d'écrans ci-dessous, je vous montre le résultat sur un émulateur avec GingerBread.


Hello world!
ActionViewActivity


Hello world!
I am an EditText but I could be whatever





Ce principe est le principe de base de l'ActionView: Il remplace un icone de la ToolBar par un layout.

5.1 Le fichier de Menu

Commençons par examiner le fichier de menu (car c'est lui qui porte l'information principale):

res\menu\menu_action_view.xml

```
<menu xmlns:android="http://schemas.android.com/apk/res/android"
   xmlns:actionbarcompat_mse="http://schemas.android.com/apk/res-auto" >
<item
android:id="@+id/menu_item_actionview"
actionbarcompat_mse:actionLayout="@layout/actionview_view"
actionbarcompat_mse:showAsAction="always|collapseActionView"
android:icon="@drawable/ic_action_provider_extends"
android:title="ActionView"/>
<item
android:id="@+id/action_one"
actionbarcompat_mse:showAsAction="always"
android:icon="@drawable/ic_action_one"
android:orderInCategory="0"
```

```
android:title="One"/>
</menu>
```

Comme d'habitude, je définis mon propre namespace xml pour que le système ne zappe pas purement et simplement les balises de la support library et je l'utilise pour ajouter deux propriétés à mon MenuItem menu_item_actionview qui sont:

actionLayout qui me permet de définir quel est le layout qui sera affiché à la place de l'item lorsque celui-ci sera déplié;

showAsAction qui me permet de définir le comportement de mon MenuItem, à savoir, toujours visible et au démarrage affiche l'item et non pas le layout.

5.2 Les fichiers de layout

Le layout qui remplacera le MenuItem est un layout rien de plus normal. Il faut quand même faire attention à un truc important, ce n'est pas lui qui définit la taille de la ToolBar. C'est à dire que s'il doit s'afficher avec 128dp de haut, il vous faudra changer la hauteur de la ToolBar vous-même lors de l'inflation du menu (quand on affichera la layout) dans le code Java (n'oubliez pas de rétablir cette taille lors du collapse du layout).

res\layout\actionview_view.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
   android:layout_width="match_parent"
   android:layout_height="match_parent"
   android:layout_gravity="center"
   android:focusable="true">
   <EditText
      android:id="@+id/edtActionView"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:ems="12"
      android:hint="I am an EditText but I could be whatever"
      android:textSize="12sp" />

   <ImageButton
      android:id="@+id/btnActionView"
      android:layout_width="32dip"
      android:layout_height="32dip"
      android:layout_gravity="center"
      android:adjustViewBounds="true"
      android:background="@drawable/ic_action_search"
      android:scaleType="fitCenter" />
</LinearLayout>
```

Le fichier de layout de l'activité est toujours le même, en particulier, il n'oublie pas de déclarer la ToolBar :

res\layout\activity_action_view.xml

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
   xmlns:tools="http://schemas.android.com/tools" android:layout_width="match_parent"
   android:layout_height="match_parent"
   tools:context="com.android2ee.formation.lollipop.toolbar.sample.ActionViewActivity">

<include layout="@layout/my_toolbar"/>

<TextView android:text="@string/hello_world"
android:layout_width="wrap_content"
android:layout_height="wrap_content" />

</RelativeLayout>
```

res\layout\my_toolbar.xml

```
<android.support.v7.widget.Toolbar xmlns:android="http://schemas.android.com/apk/res/android"
   android:id="@+id/toolbar"
   android:minHeight="?attr/actionBarSize"
   android:background="?attr/colorPrimary"
   android:layout_width="match_parent"
   android:layout_height="wrap_content">
</android.support.v7.widget.Toolbar>
```

5.3 Le code Java

Le code Java de l'activité est assez simple. Il y a trois étapes aux quelles il faut faire attention :

- instancier la ToolBar dans le onCreate
- récupérer les pointeurs vers les composants graphiques qui seront affichés dans la ToolBar quand l'ActionView s'affiche (pour les mettre à jour, rajouter des listeners...)
- écouter dans le onOptionItemSelected, le menuItem de votre ActionView et le laisser être traité par le système.

Ainsi, comme d'habitude, dans la méthode onCreate, on instancie la ToolBar:

```
protected void onCreate(Bundle savedInstanceState) {
   super.onCreate(savedInstanceState);
   setContentView(R.layout.activity_action_view);
   //find the toolbar
   toolbar = (Toolbar) findViewById(R.id.toolbar);
   postICS =getResources().getBoolean(R.bool.postICS);
   postLollipop =getResources().getBoolean(R.bool.postLollipop);
   if(postLollipop){
      toolbar.setElevation(15);
   }
   //define the toolbar as the ActionBar
   setSupportActionBar(toolbar);
   actionBar=getSupportActionBar();
}
```

Ensuite dans la onCreateOptionsMenu, on instancie le menu et on met en place l'actionView, en particulier on récupère les pointeurs vers les composants graphiques qui nous intéresse, on leur rajoute des listeners au besoin :

```
public boolean onCreateOptionsMenu(Menu menu) {
   // Inflate the menu; this adds items to the action bar if it is present.
   getMenuInflater().inflate(R.menu.menu_action_view, menu);
   //Find your menuItem that handles your actionView
   menuItemActionView = menu.findItem(R.id.menu_item_actionview);
   //Find the root layout encapsulated by the MenuItem
   lilActionView = (LinearLayout) MenuItemCompat.getActionView(menuItemActionView);
   //then find your graphical elements
   edtActionView = (EditText) lilActionView.findViewById(R.id.edtActionView);
   btnActionView = (ImageButton) lilActionView.findViewById(R.id.btnActionView);
   btnActionView.setOnClickListener(new View.OnClickListener() {
      public void onClick(View v) {
         actionOfTheActionView();
      }
   });
```

Où la méthode actionOfTheActionView est assez triviale, elle affiche un Toast et ferme l'ActionView:

```
/**
 * Handling Action from the btnActionView contained by the ActionView
 */
private void actionOfTheActionView() {
   Log.e("ActionViewActivity", "ActionView edt " + edtActionView.getText().toString());
   Toast.makeText(this, "ActionView edt = " + edtActionView.getText().toString(), Toast.LENGTH_SHORT).sho
w();
   MenuItemCompat.collapseActionView(menuItemActionView);
   //what ever is the version, hide the keyboard:
  InputMethodManager imm = (InputMethodManager)getSystemService(Context.INPUT_METHOD_SERVICE);
  imm.hideSoftInputFromWindow(edtActionView.getWindowToken(), 0);
}
```

Et enfin, dans la méthode onOptionItemSelected, on ne gère pas le MenuItem qui porte l'ActionView (mais on gère le up, ce qui n'a rien à voir avec notre exemple):

```
public boolean onOptionsItemSelected(MenuItem item) {
   switch (item.getItemId()) {
      case android.R.id.home:
         // This ID represents the Home or Up button. In the case of this
         // activity, the Up button is shown. Use NavUtils to allow users
         // to navigate up one level in the application structure. For
         // more details, see the Navigation pattern on Android Design:
         //
         // http://developer.android.com/design/patterns/navigation.html#up-vs-back
         //
         NavUtils.navigateUpFromSameTask(this);
         return true;
      case R.id.menu_item_actionview:
         // Because the system expands the action view when the user selects the action, you do
         // not need to respond to the item in the onOptionsItemSelected() callback. The system
```

```
            // still calls onOptionsItemSelected(), but if you return true (indicating you've
            // handled the event instead), then the action view will not expand.
            Log.e("ActionViewActivity", "menu_item_actionview get");
            return false;
        }
        return super.onOptionsItemSelected(item);
    }
```

Et voilà, c'est tout, en fait, c'était pas compliqué.

Ah oui, j'oubliais, dans les tutos, les gars qui oublient de déclarer leur attributs de classes, c'est super chi**t, tu sais jamais ce qu'il a déclaré (je vous mets aussi les imports qui utilise la support-library, pas les autres):

```
import android.support.v4.app.NavUtils;
import android.support.v4.view.MenuItemCompat;
import android.support.v7.app.ActionBar;
import android.support.v7.app.AppCompatActivity;
import android.support.v7.widget.Toolbar;

public class ActionViewActivity extends AppCompatActivity {
    /**
     * The action Bar
     */
    private ActionBar actionBar;
    /**
     * The ToolBar
     */
    private Toolbar toolbar;
    /**
     * The menuItem that handles the ActionView
     */
    MenuItem menuItemActionView;
    /**
     * The root Layout of the View of the ActionView
     */
    LinearLayout lilActionView;
    /**
     * The EditText to listen for the user input
     */
    EditText edtActionView;
    /**
     * The searchButton
     */
    ImageButton btnActionView;
    /***
     * Boolean to know which version is running
     */
    private boolean postICS,postLollipop;
```

5.4 Ajouter des animations

J'avais envie d'ajouter quelques animations quand l'ActionView s'affiche pour les version post HoneyComb. J'aurais pu le faire pour les versions préHoneyComb, mais quand je l'ai fait, j'ai trouvé ça trop laid. Parfois, il faut savoir s'abstenir.

Donc, pour ça, la première chose à faire est de savoir sur quelle version on se trouve. Et comme j'aime pas le BuildConfig et que je préferre les booléans, je me fais mon petit fichier de versions.

values\versions.xml

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
   <bool name="postICS">false</bool>
   <bool name="postLollipop">false</bool>
</resources>
```

values-v14\versions.xml

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
   <bool name="postICS">true</bool>
   <bool name="postLollipop">false</bool>
</resources>
```

values-v21\versions.xml

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
   <bool name="postICS">true</bool>
   <bool name="postLollipop">true</bool>
</resources>
```

Et donc dans mon code Java, je fais :

```
postICS =getResources().getBoolean(R.bool.postICS);
postLollipop =getResources().getBoolean(R.bool.postLollipop);
```

Et voilà, j'ai mes beaux booleans. Je n'ai pas inventé cette technique, j'ai regardé la conférence suivante: https://www.youtube.com/watch?v=amZM8oZBgfk (GoogleIo 2012, MultiVersionning par Bruno Oliveira, Adam Powell). ***[Alors surtout dans cette conférence ils vous disent d'utiliser le parrallel activity pattern, NE LE FAITES PAS, avec les fragments, utilisez toujours la support library si vous codez pour Gingerbread. Pas de duplication de code, surtout pas de duplication de code.]***

Et maintenant, à moi les animations. Pour ce faire, je souhaite les lancer :

- Quand l'ActionView s'affiche
- Quand j'appuie sur l'ImageButton search (la loupe)

Ainsi, je rajoute un listener pour écouter l'ouverture de l'ActionView au moment de la création du menu:

```
@Override
public boolean onCreateOptionsMenu(Menu menu) {
   // Inflate the menu; this adds items to the action bar if it is present.
   getMenuInflater().inflate(R.menu.menu_action_view, menu);
   //Find your menuItem that handles your actionView
   menuItemActionView = menu.findItem(R.id.menu_item_actionview);
   //Find the root layout encapsulated by the MenuItem
   lilActionView = (LinearLayout) MenuItemCompat.getActionView(menuItemActionView);
   //then find your graphical elements
   edtActionView = (EditText) lilActionView.findViewById(R.id.edtActionView);
   btnActionView = (ImageButton) lilActionView.findViewById(R.id.btnActionView);
   btnActionView.setOnClickListener(new View.OnClickListener() {
      public void onClick(View v) {
         actionOfTheActionView();
      }
   });
   // When using the support library, the setOnActionExpandListener() method is
   // static and accepts the MenuItem object as an argument
   MenuItemCompat.setOnActionExpandListener(menuItemActionView, new MenuItemCompat.OnActionExpand
Listener() {
      @Override
      public boolean onMenuItemActionCollapse(MenuItem item) {
         // Do something when collapsed, it's too late for animation
         Log.e("ActionViewActivity", "menu_item_actionview collapsing");
         return true; // Return true to collapse action view
       }
       @SuppressLint("NewApi")
       @Override
       public boolean onMenuItemActionExpand(MenuItem item) {
          //the first time, elements are not expanded, It's too soon to have their size
         if(postICS) {
            btnActionView.animate().rotationBy(360f).alpha(1f).setDuration(300);
            edtActionView.animate().rotationBy(360f).alpha(1f).y(0).setDuration(300);
         }
         // Do something when expanded
         return true; // Return true to expand action view
       }
     });
   return super.onCreateOptionsMenu(menu);
}
```

Et quand l'ActionView est affichée, j'anime mon ImageButton de loupe et mon EditText en les faisant tourner et bouger.

L'autre lancement s'effectue sur le clic de l'ImageButton:

```
/**
 * Handling Action from the btnActionView contained by the ActionView
 */@SuppressLint("NewApi")
private void actionOfTheActionView() {
   Log.e("ActionViewActivity", "ActionView edt " + edtActionView.getText().toString());
   Toast.makeText(this, "ActionView edt = " + edtActionView.getText().toString(), Toast.LENGTH_SHORT).sho
w();
   collapsing=true;
```

```
    if(postICS) {
        //this is called when the user press the search icon, so the
        edtActionView.animate().alpha(0).rotationBy(-360f).setDuration(300);
        btnActionView.animate().alpha(0).rotationBy(-360f).setDuration(300).setListener(new Animator.AnimatorListener() {
            //the listener is set for all the animations (further animations)
            @Override
            public void onAnimationStart(Animator animation) {
            }
            @Override
            public void onAnimationEnd(Animator animation) {
              if (collapsing) {
                MenuItemCompat.collapseActionView(menuItemActionView);
                collapsing = false;
                }
            }
            @Override
            public void onAnimationCancel(Animator animation) {
              if (collapsing) {
                 MenuItemCompat.collapseActionView(menuItemActionView);
                 collapsing = false;
               }
            }
            @Override
            public void onAnimationRepeat(Animator animation) {
            }
        });
    }else{
        MenuItemCompat.collapseActionView(menuItemActionView);
        collapsing = false;
    }
    //what ever is the version, hide the keyboard:
    InputMethodManager imm = (InputMethodManager)getSystemService(
    Context.INPUT_METHOD_SERVICE);
    imm.hideSoftInputFromWindow(edtActionView.getWindowToken(), 0);
}
```

Et là, je fais pareil, j'anime mon ImageButton et mon EditText. Mais surtout, ce qui est important ici, c'est que j'écoute l'animation pour quand elle se termine, mettre à jour mon IHM, à savoir fermer l'ActionView.

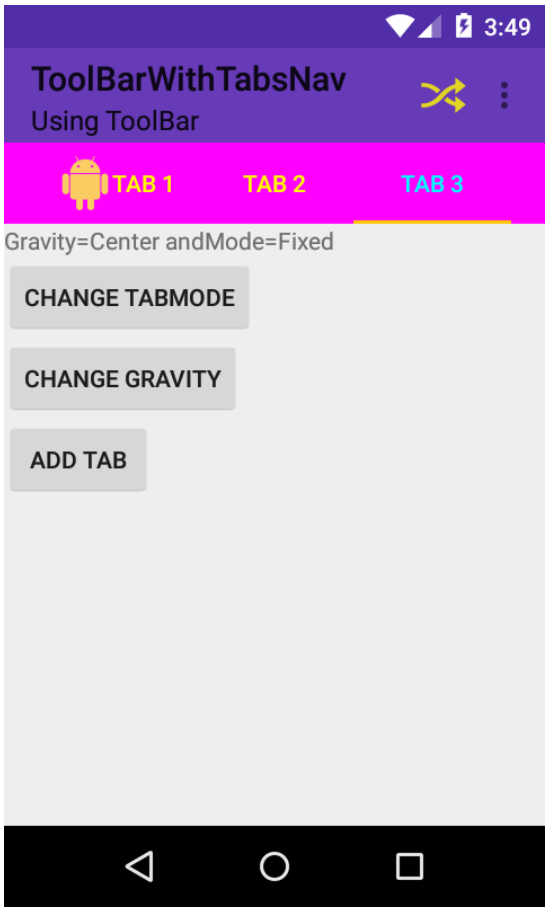
Ce qui complexifie un peu quand même ma méthode, mais est nettement plus agréable pour mon utilisateur.

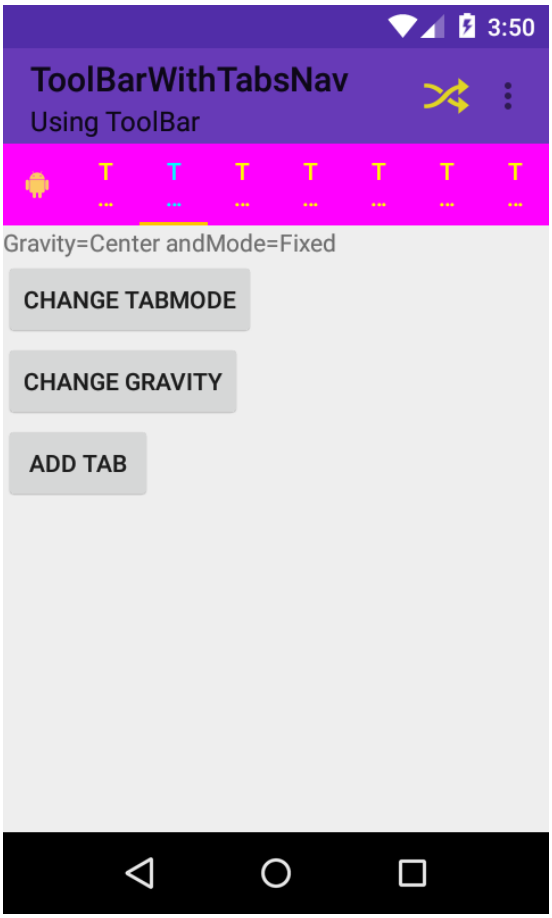
6 ToolBar et TabLayout

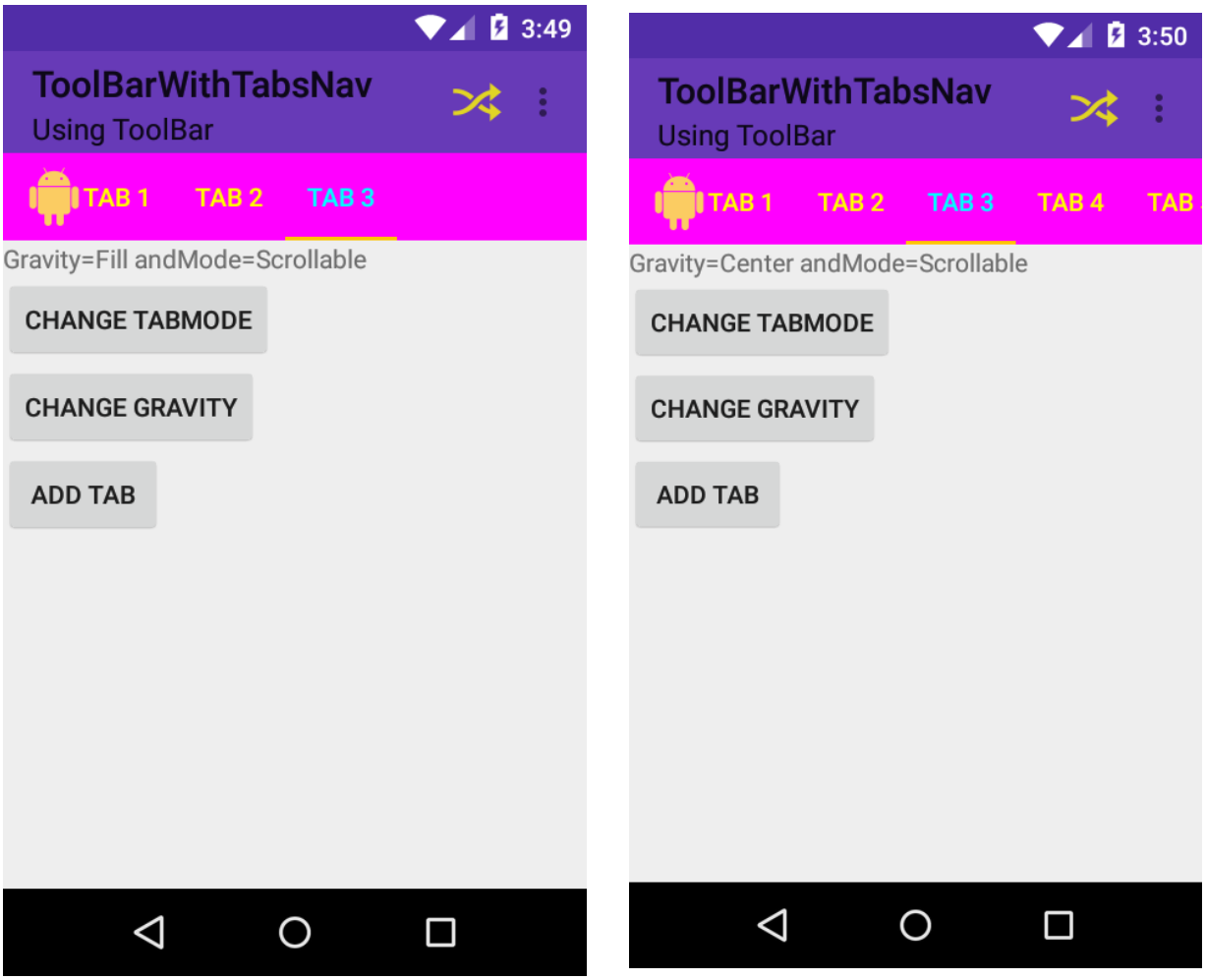
Pour finir cette série sur la ToolBar, il ne nous reste plus qu'à mettre en place la ToolBar avec une ligne d'onglets et/ou avec un ViewPager.

Commençons par la ligne d'onglets.

Nous voulons simplement avoir une activité avec des onglets et la ToolBar :


3:49
ToolBarWithTabsNav
Using ToolBar
TAB 1
TAB 2
TAB 3
Gravity=Center andMode=Fixed
CHANGE TABMODE
CHANGE GRAVITY
ADD TAB


3:50
ToolBarWithTabsNav
Using ToolBar
Gravity=Center andMode=Fixed
CHANGE TABMODE
CHANGE GRAVITY
ADD TAB



Pour ajouter cette ligne d'onglets, rien de plus facile, il faut tout d'abord définir en xml sa ligne d'onglets et les construire coté Java. Aucune surprise ne vous attend. Nous utilisons uniquement les composants de la DesignLibrary pour la mise en place des onglets et tout se déroule alors facilement.

Il y a deux paramètres que nous pouvons modifier pour le comportement de la ToolBar: La notion de Gravité et la notion de Mode.

La gravité possède deux valeurs Center ou Fill. Center rassemble les onglets au centre de l'écran en mode wrapContent alors que Fill répartit uniformément l'espace disponible pour chaque onglet.

Le mode permet simplement de dire si vous souhaitez que vos onglets soient scrollables ou pas. S'ils ne sont pas scrollables, ils s'affichent tous dans l'espace disponible, ce qui veut dire qu'ils sont compressés et peuvent ne pas avoir la place de s'afficher complètement.

6.1 Le fichier de Layout

Commençons par examiner le fichier de layout:

res\layout\activity_activity_with_tabs_nav.xml

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    tools:context=".ActivityWithTabsNav">
<android.support.v7.widget.Toolbar
   android:id="@+id/toolbar"
   android:layout_width="match_parent"
   android:layout_height="wrap_content"
   android:background="?attr/colorPrimary"
   android:minHeight="?attr/actionBarSize"/>
<android.support.design.widget.TabLayout
   android:id="@+id/tabLayout"
   android:layout_width="match_parent"
   android:layout_height="wrap_content"
   android:background="#FF00FF"
   android:fillViewport="true" />
<TextView
   android:id="@+id/txvState"
   android:layout_width="wrap_content"
   android:layout_height="wrap_content"
   android:text="@string/hello_world" />
<Button
   android:id="@+id/btnChangeMode"
   android:layout_width="wrap_content"
   android:layout_height="wrap_content"
   android:text="@string/btnChangeTabMode"/>
<Button
   android:id="@+id/btnChangeGravity"
   android:layout_width="wrap_content"
   android:layout_height="wrap_content"
   android:text="@string/btnChangeGravity"/>
<Button
  android:id="@+id/btnAddTab"
  android:layout_width="wrap_content"
  android:layout_height="wrap_content"
  android:text="@string/btnAddTab"/>
LinearLayout>
```

Il n'y a pas grand-chose à dire, si ce n'est que TabLayout appartient à la DesignLibrary et la ToolBar à la support.v7.

6.2 Le code Java

Le code Java de l'activité est assez simple. Il y a trois étapes auxquelles il faut faire attention :

- Instancier la ToolBar et le TabLayout
- Construire les onglets du TabLayout
- Ajouter un listener au TabLayout pour être avertit des changements d'onglets.

Tout se fait dans le onCreate de votre activité (ou dans le onCreateView de votre fragment).

Ainsi, comme d'habitude, dans la méthode onCreate, on instancie la ToolBar:

```
protected void onCreate(Bundle savedInstanceState) {
   super.onCreate(savedInstanceState);
   //set your view
   setContentView(R.layout.activity_activity_with_tabs_nav);
   //find the Toolbar
   Toolbar toolbar = (Toolbar) findViewById(R.id.toolbar);
   //use it as your action bar
   setSupportActionBar(toolbar);
   //Add subtitle
   getSupportActionBar().setSubtitle("Using ToolBar");
   //Find the Tab Layout
   tabLayout = (TabLayout) findViewById(R.id.tabLayout);
   //Define its gravity and its mode
   tabLayout.setTabGravity(TabLayout.GRAVITY_FILL);
   tabLayout.setTabMode(TabLayout.MODE_FIXED);
   //Define the color to use (depending on the state a different color should be disaplyed)
   //Works only if done before adding tabs
   tabLayout.setTabTextColors(getResources().getColorStateList(R.color.tab_selector_color));
   //populate your TabLayout
   tabLayout.addTab(tabLayout.newTab().setText("Tab 1").setIcon(R.drawable.ic_tab));
   tabLayout.addTab(tabLayout.newTab().setText("Tab 2"));
   tabLayout.addTab(tabLayout.newTab().setText("Tab 3"));
   //add a listener
   tabLayout.setOnTabSelectedListener(new TabLayout.OnTabSelectedListener() {
      @Override
      public void onTabSelected(TabLayout.Tab tab) {/*do your navigation*/}
      @Override
      public void onTabUnselected(TabLayout.Tab tab) {/*do nothing*/}
          @Override
      public void onTabReselected(TabLayout.Tab tab) {/*do nothing*/}
   });
   ...
}
```

Comme d'habitude, on récupère la ToolBar et on l'affecte à l'activité comme étant son ActionBar.

Puis on s'occupe du TabLayout. Ici, il n'y a plus besoin d'appeler la méthode setup sur le TabLayout, l'initialisation est automatique. Il vous suffit de le récupérer (findViewById), de lui affecter son Mode et sa Gravité puis de lui ajouter ses onglets via la méthode addTab en fournissant pour chaque onglet, un texte et un icone optionnel.

Enfin, il ne vous reste plus qu'à ajouter le TabListener pour interagir avec l'utilisateur et changer le fragment affiché dans l'onglet. Vous pouvez aussi mettre des vues en utilisant la méthode addView(View view, int index, LayoutParameter layoutParam).

Mais attention, la philosophie à changer, il n'y a plus besoin que votre TabLayout contienne les vues pour chaque onglet. Il n'est qu'un conteneur d'onglets; uniquement la ligne qui affiche les boutons, à vous de gérer la mise à jour de la vue / du fragment sous cette ligne. Je pense que c'est la bonne manière d'aborder ce composant avec les fragments en utilisant ces derniers de manière dynamique.

Et voilà, c'est tout, en fait, ce n'était pas compliqué.

Ah oui, j'oubliais, les imports utilisés pour cette activité sont:

```
importandroid.support.design.widget.TabLayout;
import android.support.v7.app.AppCompatActivity;
import android.support.v7.widget.Toolbar;
```

6.3 Gestion des couleurs des onglets

Une dernière précision sur l'affectation des couleurs à vos boutons d'onglets. Pour effectuer cela, on s'appuie sur un fichier de couleurs un peu particulier, un SelectorColor (identique à un SelectorDrawable dans l'idée). Un SelectorColor est une couleur qui en fonction de l'état du composant (enabled/selected/hover...) possède une valeur spécifique:

```
<selector xmlns:android="http://schemas.android.com/apk/res/android">
   <item android:state_enabled="false" android:color="@color/testcolor2" />
   <item android:state_selected="true" android:color="@color/testcolor3" />
   <item android:color="@color/testcolor5"/>
<selector>
```

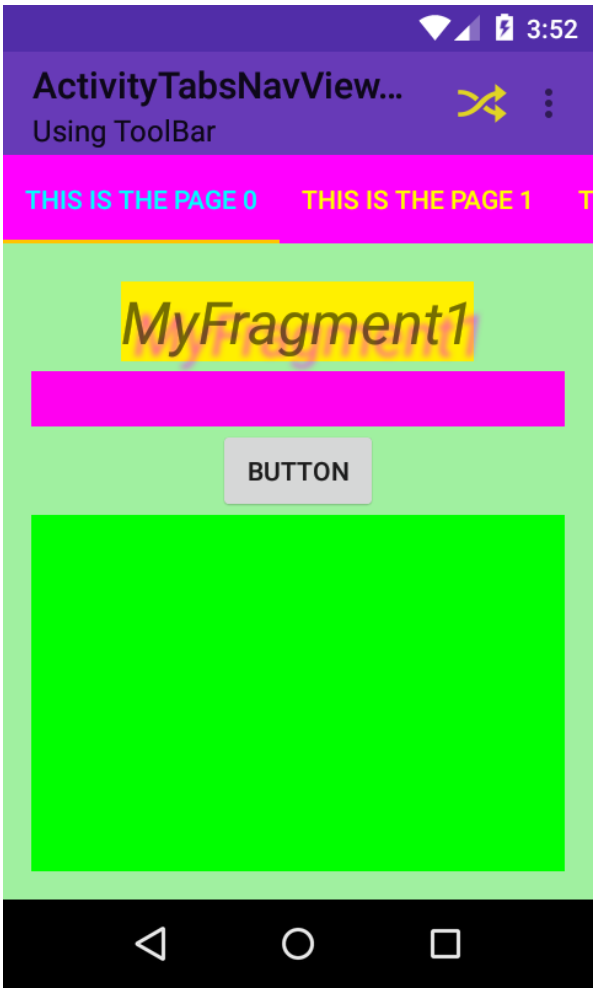
Il ne reste plus qu'à affecter cette couleur à nos onglets et automatiquement (s'ils sont sélectionnés ou pas) s'afficheront de manière différente (avec une couleur différente).

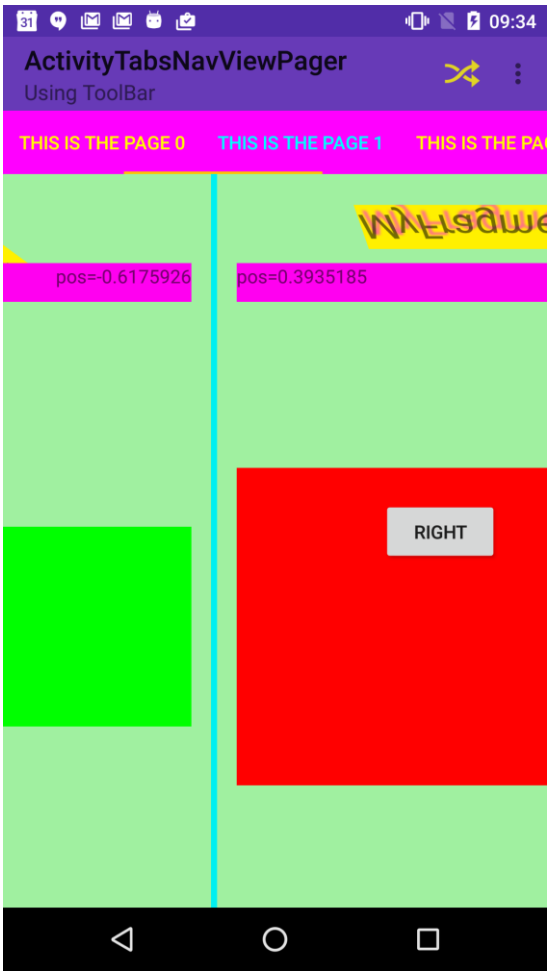
```
tabLayout.setTabTextColors(getResources().getColorStateList(R.color.tab_selector_color));
```

Vous devez affecter les couleurs à vos onglets AVANT d'ajouter des boutons à votre ligne d'onglet (bref avant d'appeler la méthode addTab).

7 ToolBar et ViewPager

Ces composants sont faits pour marcher ensemble simplement. Il suffit de déclarer la TabLayout, le ViewPager et la ToolBar dans votre fichier de layout, puis côté Java, d'ajouter la TabLayout comme précedement et de lier le ViewPager au TabLayout.





7.1 Le fichier de Layout

Commençons par examiner le fichier de layout:

res\layout\activity_tabs_view_pager.xml

```
!--xml version="1.0" encoding="utf-8"-->
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    tools:context="com.android2ee.formation.lollipop.toolbar.tabsnav.ActivityTabsNavViewPager">
<android.support.v7.widget.Toolbar
   android:id="@+id/toolbar"
   android:layout_width="match_parent"
   android:layout_height="wrap_content"
   android:background="?attr/colorPrimary"
   android:minHeight="?attr/actionBarSize"/>
```

```
<!--For more information: http://stackoverflow.com/questions/30904138/how-to-change-the-new-tablayout-indicator-color-and-height-->
 <android.support.design.widget.TabLayout
   android:id="@+id/tabLayout"
   android:layout_width="match_parent"
   android:layout_height="wrap_content"
   android:background="#FF00FF"
   android:fillViewport="true" />
<android.support.v4.view.ViewPager
   android:layout_width="fill_parent"
   android:layout_height="fill_parent"
   android:id="@+id/viewpager"
   android:background="#FF00F0F0"
    android:fillViewport="true">
</android.support.v4.view.ViewPager>
</LinearLayout>
```

Il n'y a pas grand-chose à dire, si ce n'est que TabLayout appartient à la DesignLibrary et la ToolBar à la support.v7 et le ViewPager à la support v4 (inclus dans la support v7).

7.2 Le code Java

Le code Java de l'activité est assez simple. Il y a quatre étapes auxquelles il faut faire attention :

Instancier la ToolBar, le ViewPager et le TabLayout,

Construire les onglets du TabLayout,

Ajouter un listener au TabLayout pour être avertit des changements d'onglets (ce n'est ici pas obligatoire, je le conseille pour la restauration de l'onglet courant quand l'utilisateur revient dans votre activité),

Lier le ViewPager au TabLayout.

Tout se fait dans le onCreate de votre activité (ou dans le onCreateView de votre fragment).

Ainsi, comme d'habitude, dans la méthode onCreate, on instancie la ToolBar:

```
@Override
protected void onCreate(Bundle savedInstanceState) {
   super.onCreate(savedInstanceState);
   setContentView(R.layout.activity_tabs_nav_view_pager);
   //find the Toolbar
   Toolbar toolbar = (Toolbar) findViewById(R.id.toolbar);
   //use it as your action bar
   setSupportActionBar(toolbar);
   //Add subtitle
   getSupportActionBar().setSubtitle("Using ToolBar");
   //Find the Tab Layout
   tabLayout = (TabLayout) findViewById(R.id.tabLayout);
   //Define its gravity and its mode
```

```
    tabLayout.setTabGravity(TabLayout.GRAVITY_CENTER);
    tabLayout.setTabMode(TabLayout.MODE_SCROLLABLE);
    //instanciate the PageAdapter
    pagerAdapter=new MyPagerAdapter(this);
    //Define the color to use (depending on the state a different color should be disaplyed)
    //Works only if done before adding tabs
    tabLayout.setTabTextColors(getResources().getColorStateList(R.color.tab_selector_color));
    //Find the viewPager
    viewPager = (ViewPager) super.findViewById(R.id.viewpager);
    // Affectation de l'adapter au ViewPager
    viewPager.setAdapter(pagerAdapter);
    viewPager.setClipToPadding(false);
    viewPager.setPageMargin(12);
    //Add animation when the page are swiped
    //this instanciation only works with honeyComb and more
    //if you want it all version use AnimatorProxy of the nineoldAndroid lib
    //@see:http://stackoverflow.com/questions/15767729/backwards-compatible-pagetransformer
    if(Build.VERSION.SDK_INT>=Build.VERSION_CODES.HONEYCOMB){
       viewPager.setPageTransformer(true, new MyPageTransformer(this));
    }
    //AND CLUE TABLAYOUT AND VIEWPAGER
    tabLayout.setupWithViewPager(viewPager);
}
```

Comme d'habitude, on récupère la ToolBar et on l'affecte à l'activité comme étant son ActionBar.

Puis on s'occupe du TabLayout (paragraphe précédent).

Enfin, il ne vous reste plus qu'à ajouter le TabListener pour se souvenir de l'onglet dans le quel est l'utilisateur. Ainsi, quand l'utilisateur revient dans votre activité, si elle est encore dans la liste des applications récentes (dans votre LRUCach pour être exact), vous pouvez alors le replacer dans l'onglet qu'il vient de quitter.

Enfin, pour rappel, le code du ViewPager est le suivant:

```
public class MyPagerAdapter extends FragmentPagerAdapter {

/**
 * The list of ordered fragments
 */
 private final ArrayList fragments;

//On fournit à l'adapter la liste des fragments à afficher

 /***
 * The constructor
 * @param ctx The Context
 */
 public MyPagerAdapter(ActivityTabsNavViewPager ctx) {
    super(ctx.getSupportFragmentManager());
   fragments = new ArrayList();
    //A stuff I never did before, instanciate my fragment
   Fragment frag = new MyFragment1();
```

```
  fragments.add(frag);
  frag = new MyFragment2();
  fragments.add(frag);
  frag = new MyFragment3();
  fragments.add(frag);
  frag = new MyFragment4();
  fragments.add(frag);
  frag = new MyFragment5();
  fragments.add(frag);
 }

 /**
 * This method may be called by the ViewPager to obtain a title string
 * to describe the specified page. This method may return null
 * indicating no title for this page. The default implementation returns
 * null.
 *
 * @param position The position of the title requested
 * @return A title for the requested page
 */
 @Override
 public CharSequence getPageTitle(int position) {
  return "This is the Page "+position;
 }

 @Override
 public Fragment getItem(int position) {
  return fragments.get(position);
 }

 @Override
 public int getCount() {
  return 5;
 } }
```

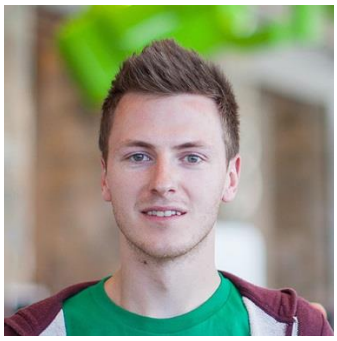
Ah oui, j'oubliais, les imports utilisés pour cette activité sont:

```
import android.support.design.widget.TabLayout;
import android.support.v4.view.ViewPager;
import android.support.v7.app.AppCompatActivity;
import android.support.v7.widget.Toolbar;
```

8 Conclusion

Je finis cet article, concernant la ToolBar, en remerciant Chris Banes (@chrisbanes) car c'est à lui que l'on doit cette simplification et surtout cette libération de l'ActionBar,

so Thanks a billion Mr Banes.



9 Le tutorial

Comme tous mes articles, celui-ci est associé à un tutorial, vous le trouverez ici :
https://github.com/MathiasSeguy-Android2EE/ToolBar4Github