Les Fragments

Cet article est rédigé par Android2EE, expert en formation Android.

Il est associé à deux tutoriaux vous montrant comment mettre en place :

- Une application avec des fragments statiques
- Une application avec des fragments dynamiques

Pour plus d'information (tutoriels, ebooks, formations), une seule adresse :

Android2EE : http://www.android2ee.com.

1.1 Définition, objectifs et philosophie des Fragments

Les fragments sont apparus le 19 Novembre 2011 sur le système d'exploitation HoneyComb dédié aux tablettes.

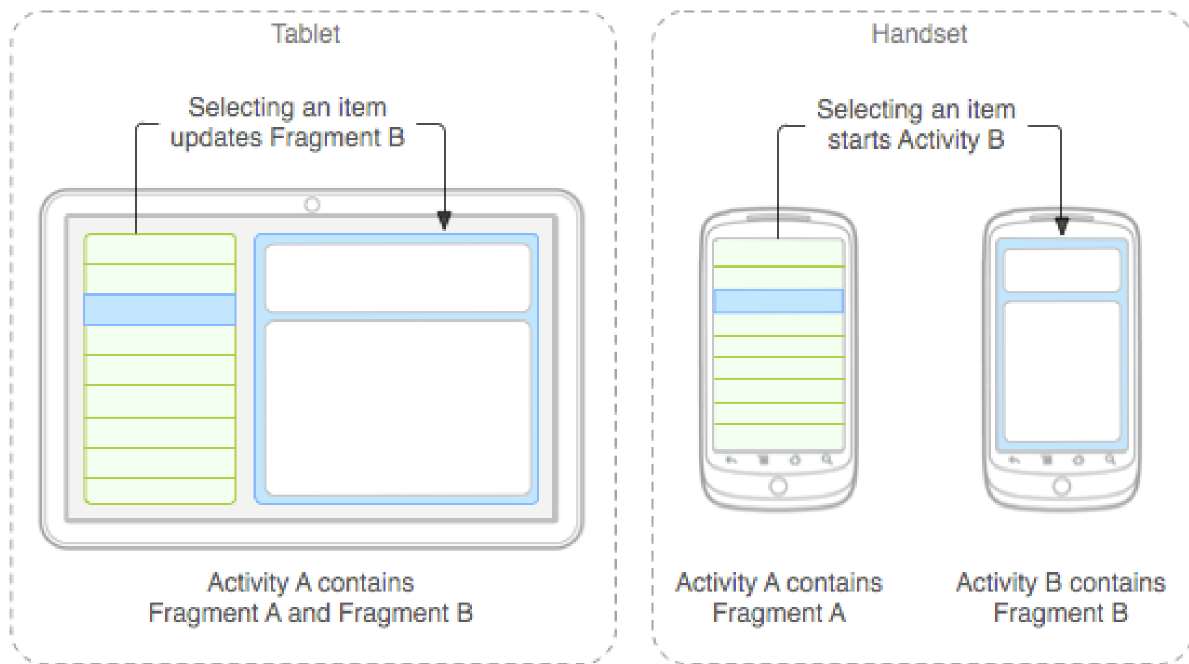
1.1.1 Philosophie

La philosophie découle d'un problème simple qui est l'adaptation d'une application Android à toutes les tailles d'appareils existant.

Tout d'abord, il faut bien comprendre que sans les fragments il était déjà possible de s'adapter à toutes les tailles d'écrans.Il suffisait de mettre un layout particulier dans layout-small, layout-normal, layout-large, layout-xlarge, pour obtenir l'I.H.M. souhaitée en fonction des caractéristiques de l'écran. Par contre, l'activité qui contrôlait tous ces cas devenait très complexe ; son nombre cyclotimique explosait, la capacité d'un esprit humain à comprendre, modifier, corriger ou faire évoluer l'activité devenait réduite, les méthodes du cycle de vie faisait 50 à 100 lignes. Et finalement, l'activité devenait ingérable.

C'est ce qui a motivé à créer les fragments; les fragments permettent de scinder vos activités en composants encapsulés et réutilisables qui possèdent leur propre cycle de vie et leur propre interfaces graphiques. Cela permet de mettre en place des IHM évoluées qui s'adaptent aux différents écrans et à leur orientation tout en maintenant le code de l'activité « human readable ».

Ainsi dans l'exemple ci-dessous (celui de Google), nous voyons comment l'activité A s'adapte en fonction de la taille de l'écran.



Nous voyons aussi qu'il faut une seconde activité qui sera utilisée uniquement pour les smartphones. Ainsi l'utilisation des fragments change une activité complexe en une application complexe d'activités simplifiées. Ce qui est un objectif fondamental d'architecture.

1.1.2 Définition

Les fragments permettent de scinder vos activités en composants encapsulés et réutilisables qui possèdent leur propre cycle de vie et leur propre interfaces graphiques. Cela permet de mettre en place des IHM évoluées qui s'adaptent aux différents écrans et à leur orientation.

Quelques notions élémentaires concernant les fragments:

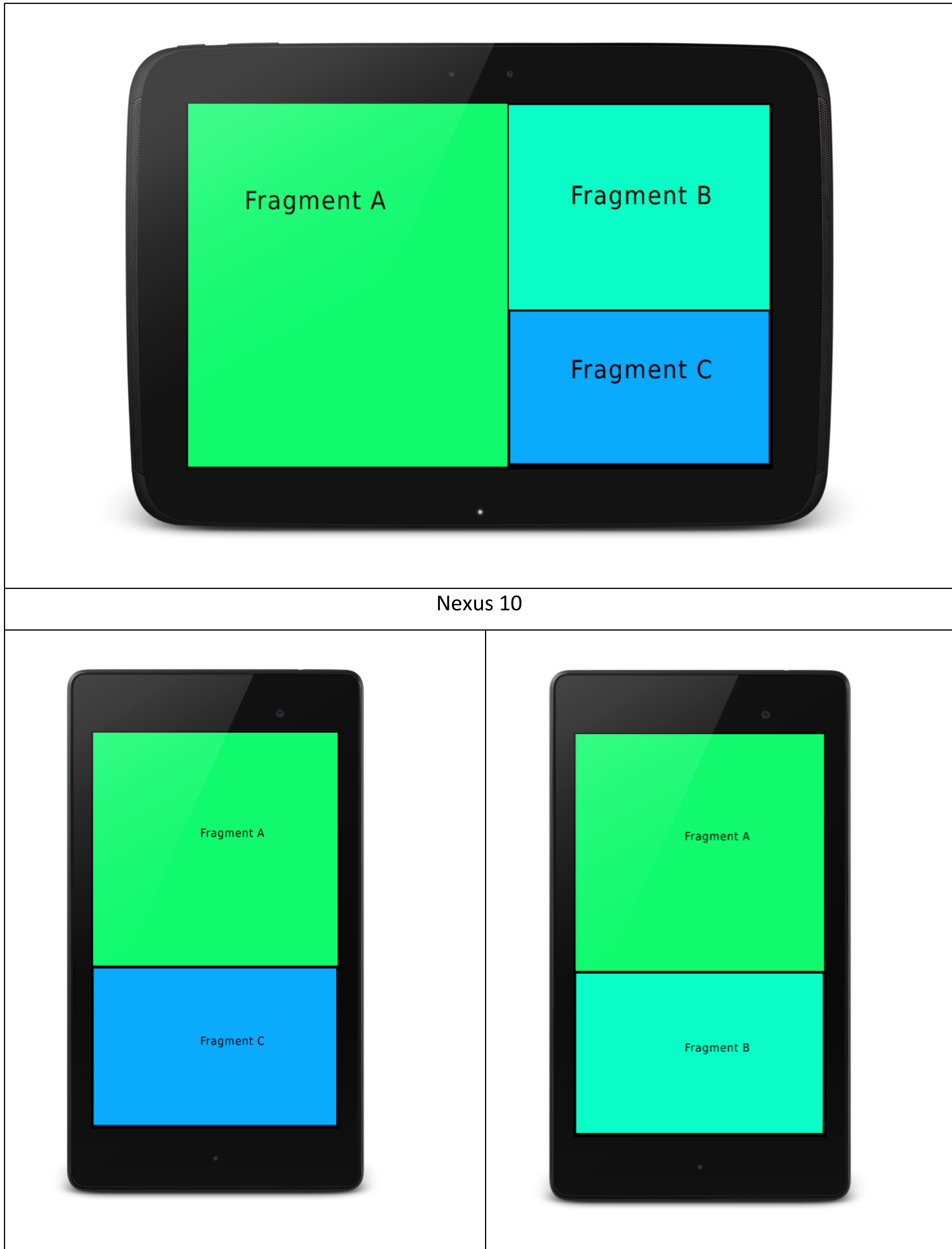
- Ils déportent une partie du traitement de l'activité en leur sein;
- Ils sont liés à une activité (ils n'existent pas sans elle);
- Ils définissent la plupart du temps une interface graphique mais peuvent aussi être utilisés pour retenir un état lors de la destruction/reconstruction de leur activité parente (le bon vieux onRetainNonConfigurationInstance);
- Ils peuvent être statiques (définis une fois pour toute dans le fichier de layout) ou dynamiques (créés, supprimés, ajoutés dynamiquement);
- Ils sont apparus à partir de HoneyComb (level 11) ainsi pour les mettre en place avant HoneyComb, il faut utiliser la support-librairy.
- Pour les utiliser, il faut un BuildSDK et un TragetSDK supérieur à 11 (cela tombe bien vous devriez être à 19 à l'heure où j'écris ces lignes).

Les classes fondamentales pour la gestion des fragments sont : *Fragment, FragmentManager et FragmentTransaction.*

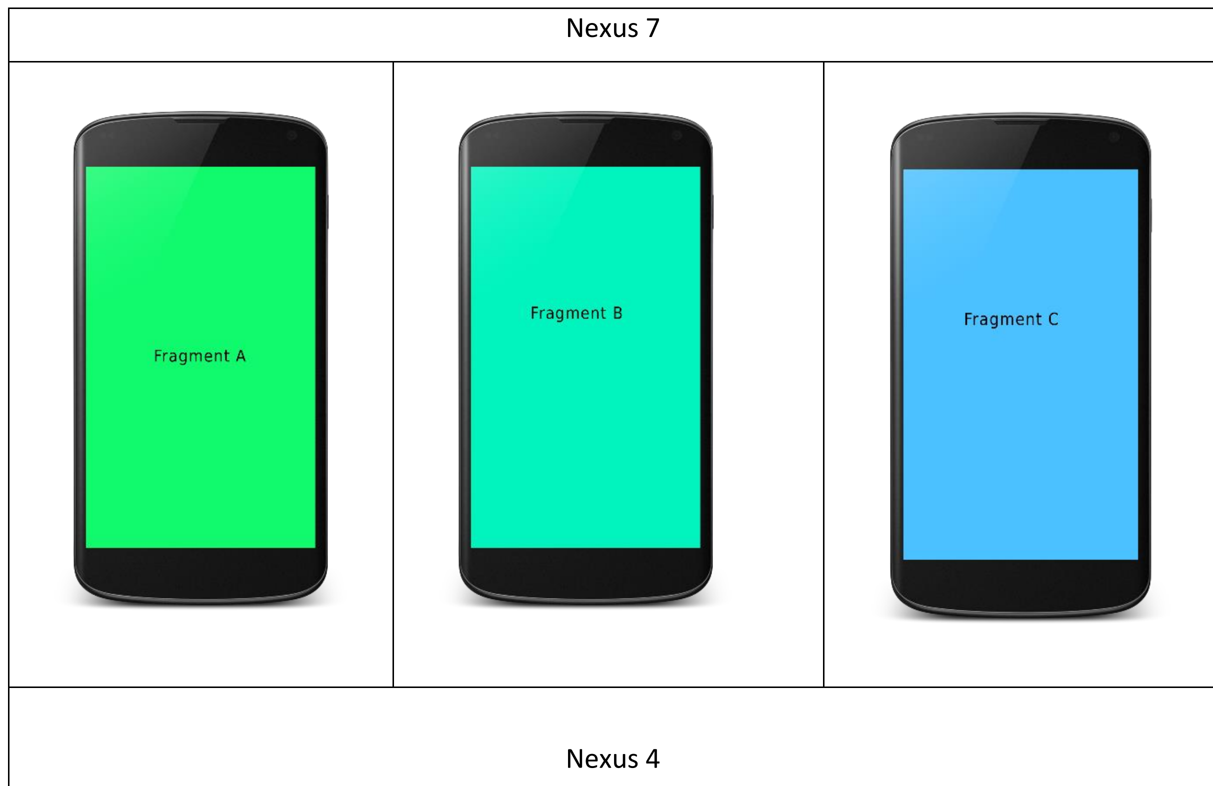
1.1.3 Objectifs

Vos objectifs sont simples ; il vous faut utiliser les fragments pour vous adapter le plus facilement possible aux configurations des écrans. Ce qui veut dire que je vous parle de Design.

Ainsi vous allez commencer par définir vos I.H.M. pour les tablettes 10 pouces, puis pour les 7 pouces et finir par celles du smartphone. Il faut réutiliser les fragments que vous avez définis pour les 10' et peut-être définir une navigation pour chaque taille d'écran.



@ copyright2012@android2ee



Votre objectif est de construire une application s'adaptant à ces tailles d'écrans. Remarquez que ce ne sont que des jalons, il faut aussi que votre application se redimensionne entre ces jalons; donc toujours pas de tailles en dur.

1.2 Fragments et cycles de vie

Avant de commencer à expliquer l'utilisation des fragments, il faut comprendre quel est le cycle de vie d'un fragment, comment il se lie à celui de l'activité qui le contient.

Comme pour les activités, en fonction du cycle de vie du fragment, il faut savoir sauver, restaurer, instancier, détruire les données lors des différentes étapes de ce cycle.

Pour rappel, quand on est dans une activité :

- Dans onCreate, on instancie les objets ;
- Dans onStart, on lance les traitements ;
- Dans onResume, on s'abonne et on remet le contexte utilisateur ;
- Dans onPause, on se désabonne et on enregistre le contexte utilisateur ;
- Dans onStop on arrête les traitements et on désalloue les objets ;
- Dans onDestroy on ne fait rien (elle n'est pas appelée systématiquement), on préfère utiliser les méthodes de type onTrimMemory pour connaître l'état de l'application dans le LRU cache.

Ces principes s'appliquent aux fragments de la même manière. Ainsi quand on est dans un fragment :

- Dans onAttach, on récupère un pointeur vers l'activité contenante (attention aux NullPointerException, celle-ci n'a pas finalisée sa construction) ;
- Dans onCreate, on instancie les objets non graphique;
- Dans onCreateView, on instancie la vue et les composants graphiques ;

- Dans onActivityCreate, on récupère un pointeur sur l'activité (qui est construite), on lance les initialisations qui ont besoin de cette activité, on restaure le contexte des fragments et utilisateur.
- Dans onStart, on lance les traitements ;
- Dans onResume, on s'abonne et on remet le contexte utilisateur ;
- Dans onPause, on se désabonne et on enregistre le contexte utilisateur ;
- Dans onStop, on arrête les traitements et on désalloue les objets ;
- Dans onDestroyView, la vue est détachée de celle de l'activité, on peut délivrer la mémoire des objets graphiques ;
- Dans onDestroy, je ne préconise pas grand-chose ;
- Dans onDetach, non plus.

Personnellement, je surcharge systématiquement les méthodes suivantes:

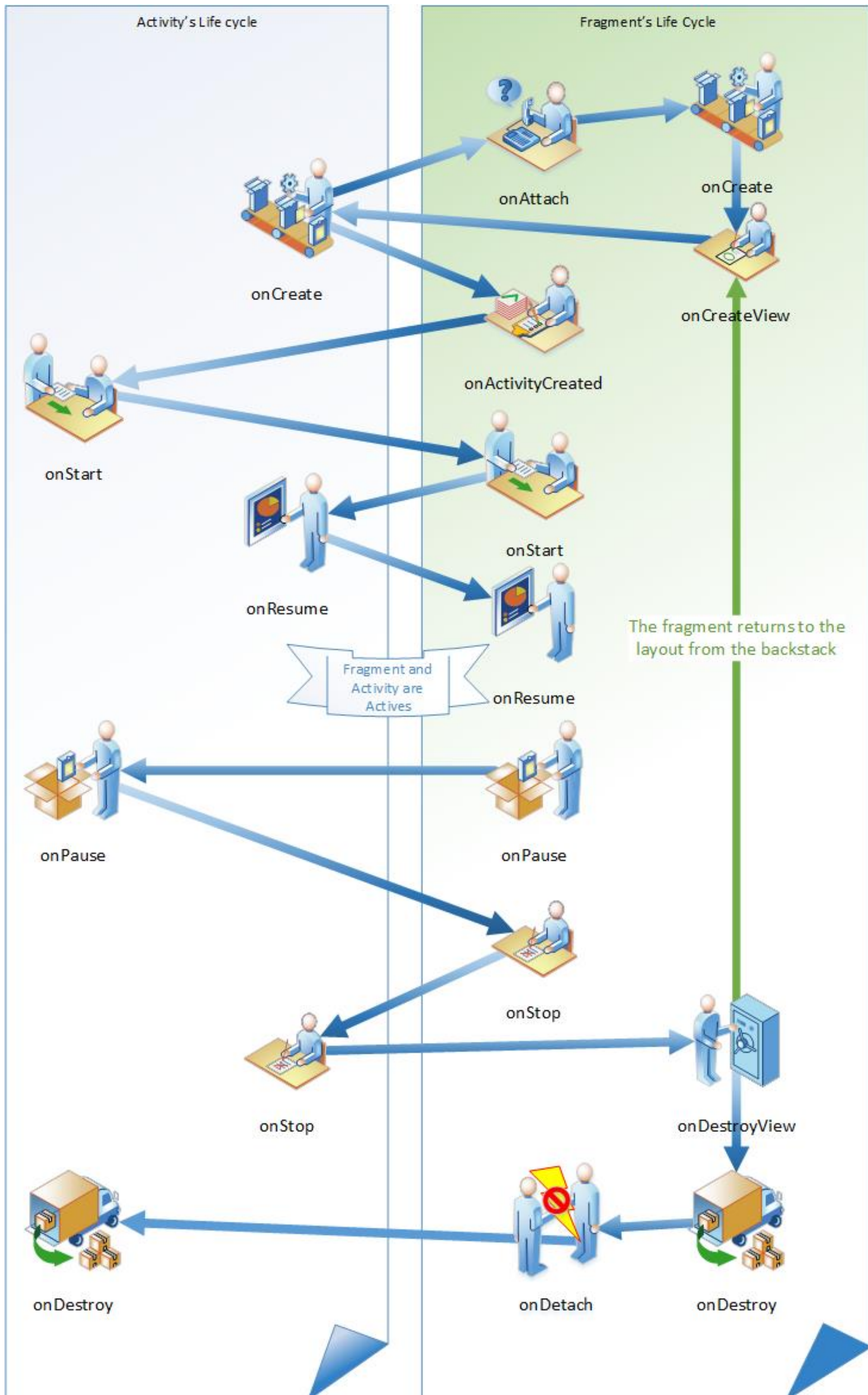
- Dans le cadre d'une activité onCreate, onResume, onPause, onSaveInstance, onRestoreInstance et onStop ;
- Dans le cadre d'un fragment onCreate, onCreateView, onActivityCreated, onResume, onPause, onSaveInstance et onStop.

L'intrication du cycle de vie d'une activité et d'un fragment se lit dans les logs suivants :

12-11 16:41:19.188: E/MainActivityHC(9879): **onCreate** called
12-11 16:41:19.258: W/MainFragmentHC(9879): **onAttach** called
12-11 16:41:19.258: W/MainFragmentHC(9879): **onCreate** called
12-11 16:41:19.268: W/MainFragmentHC(9879): **onCreateView** called
12-11 16:41:19.318: E/MainActivityHC(9879): **onCreate** finished
12-11 16:41:19.318: W/MainFragmentHC(9879): **onActivityCreated** called
12-11 16:41:19.318: E/MainActivityHC(9879): **onStart** called
12-11 16:41:19.318: E/MainActivityHC(9879): **onStart** finished
12-11 16:41:19.318: W/MainFragmentHC(9879): **onStart** called
12-11 16:41:19.318: E/MainActivityHC(9879): **onResume** called
12-11 16:41:19.318: E/MainActivityHC(9879): **onResume** finished
12-11 16:41:19.318: W/MainFragmentHC(9879): **onResume** called
12-11 16:41:35.528: W/MainFragmentHC(9879): **onPause** called
12-11 16:41:35.528: E/MainActivityHC(9879): **onPause** called
12-11 16:41:35.528: E/MainActivityHC(9879): **onPause** finished
12-11 16:41:36.088: W/MainFragmentHC(9879): **onStop** called
12-11 16:41:36.108: E/MainActivityHC(9879): **onStop** called
12-11 16:41:36.108: E/MainActivityHC(9879): **onStop** finished
12-11 16:41:36.128: W/MainFragmentHC(9879): **onDestroyView** called
12-11 16:41:36.168: W/MainFragmentHC(9879): **onDestroy** called
12-11 16:41:36.168: W/MainFragmentHC(9879): **onDetach** called
12-11 16:41:36.178: E/MainActivityHC(9879): **onDestroy** called
12-11 16:41:36.178: E/MainActivityHC(9879): **onDestroy** finished

Ce qui se visualise avec le schéma suivant :

@ copyright2012@android2ee


Activity's Life cycle
Fragment's Life Cycle
onAttach
onCreate
onCreate
onCreateView
onActivityCreated
onStart
onStart
onResume
onResume
The fragment returns to the layout from the backstack
Fragment and Activity are Actives
onResume
onPause
onPause
onStop
onStop
onDestroyView
onDestroy
onDetach
onDestroy

Les fragments possèdent leur propre cycle de vie et celui-ci est intimement lié à celui de l'activité qui le contient.

Fondamental :

Toute méthode surchargée de ce cycle de vie doit impérativement appeler son super sinon une exception sera levée au RunTime (exception faite de onCreateView).

onCreateView

Dans la pratique, la méthode onCreateView est la méthode que l'on utilise pour créer la vue qui sera affichée par le fragment. C'est la méthode qui est systématiquement surchargée.

public View onCreateView (LayoutInflater inflater, ViewGroup container, Bundle savedInstanceState) possède les arguments suivants:

- *inflater* : Il permet de gonfler le fichier xml de layout
- *container*: Si non-null, correspond à la vue parente qui contient le fragment et à laquelle la vue du fragment va s'attacher.
- *savedInstanceState* : si non-null, le fragment est reconstruit à partir d'un état précédent sauvegardé et transmis par ce paramètre.

Le code typique de la méthode onCreateView gonfle le fichier xml pour en construire une vue et la renvoyer :

```
@Override
public View onCreateView(LayoutInflater inflater, ViewGroup container, Bundle savedInstanceState)
{
        View view = inflater.inflate(R.layout.speaker_detail, container, false);
        return view;
}
```

Autres méthodes

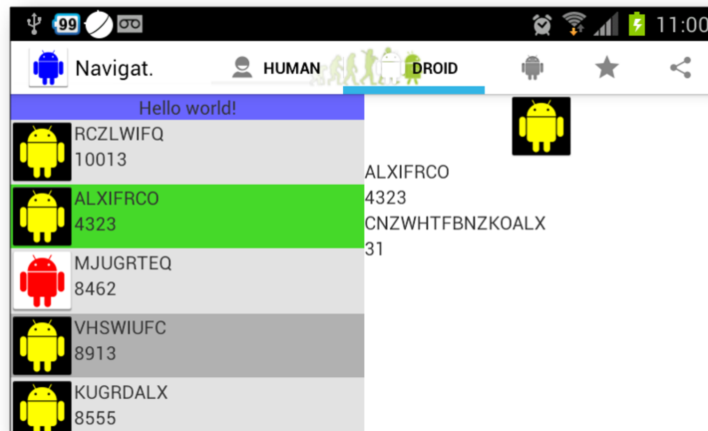
Comme pour les activités les bonnes pratiques sont:

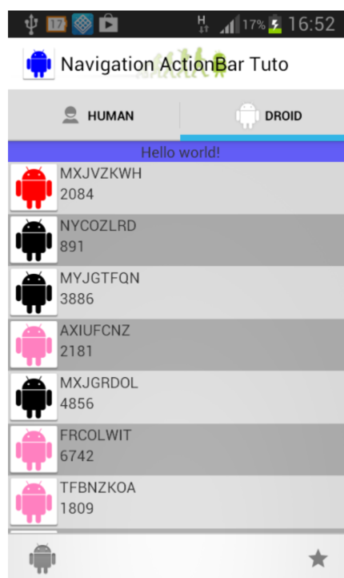
- **onCreate** permet de créer les objets de la classe pour qu'ils soient instanciés une seule fois dans le cycle de vie du fragment;
- **onCreate** ne créé pas l'interface graphique;
- **onCreateView** créé l'interface graphique;
- **onActivityCreated** permet de récupérer un pointeur vers l'activité;
- **onDestroy** n'est pas toujours appelée (que ce soit pour l'activité comme pour le fragment);
- **onAttach** permet de récupérer une instance de l'activité parente, mais attention, elle n'a pas fini son initialisation (il vaut mieux attendre onActivityCreated).

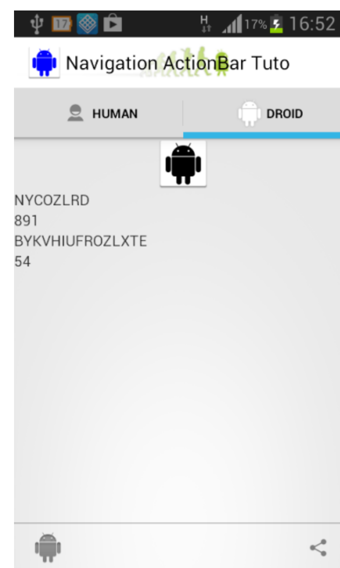
Un fragment peut finir son cycle de vie sans que l'activité ne modifie le sien (reste à l'état actif).

@ copyright2012@android2ee

2 Fragments statiques







2.1 Mise en place

Nous allons voir dans ce chapitre comment mettre en place une activité qui contient un fragment de manière statique.

Un fragment statique ne peut ni être supprimé ni être remplacé, il est statique.

2.1.1 Création du fragment

L'interface graphique du fragment se définit dans un fichier xml. Il n'y a aucune différence entre un fichier xml de layout (avant HoneyComb) pour une activité et un fichier xml de layout pour un fragment. Vous pouvez utiliser les mêmes composants graphiques, les mêmes layouts, la syntaxe est identique. Il n'y a pas de différence.

```
<LinearLayout
        android:layout_width="wrap_content"
        android:layout_height=" fill_parent">

        <TextView
        android:id="@+id/name"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
```

```
android:text="@string/hello" />

<ListView
android:id="@+id/myListView"
android:layout_width="fill_parent"
android:layout_height="wrap_content"/>

</LinearLayout>
```

Votre classe Fragment (MyFragment par exemple) étend simplement Fragment et la seule chose obligatoire est de surcharger la méthode onCreateView

```
@Override public View onCreateView(LayoutInflater inflater, ViewGroup container, Bundle
savedInstanceState) { View view = inflater.inflate(R.layout.speaker_detail, container, false);
//Instancier vos composants graphique ici (faîtes vos findViewById) return view; }
```

Le point important se produit lors du gonflage de lavue **inflater.inflate(R.layout.*speaker_detail, container, false*).** En effet le dernier paramètre est à **false**. Ce paramètre spécifie si la vue doit être automatiquement attachée à son container parent. Dans le cas d'un fragment, cette liaison doit être laissée au système qui se charge de la rattacher comme il se doit à l'activité.

C'est dans cette classe que vous allez coder votre comportement, faire vos findViewById et tout le code que vous auriez mis dans votre activité. Il suffit de coder « comme avant » directement dans votre fragment.

2.1.2 Création de l'activité qui va accueillir le Fragment

Pour qu'une activité utilise un fragment statique, il suffit de déclarer le fragment en tant qu'objet graphique au sein du fichier de layout de votre activité.

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

    <fragment
        android:id="@+id/list"
        android:layout_width="fill_parent"
        android:layout_height="fill_parent"
        android:name="com.android2ee.tuto.fragment.sample1.ListFragment"
        android:orientation="horizontal" >
    </fragment>

</LinearLayout>
```

Les points importants sont :

- La balise **fragment** (en minuscule je vous prie)
- la balise **android:name** qui permet de spécifier quelle est la classe qui prend en charge l'implémentation du fragment
- La balise **android:id** (resp. **android:tag**) qui permet de spécifier un identifiant entier (resp. de type String) unique au fragment, ce qui est très important lors de la manipulation dynamique des fragments;
- Et bien sûr, la manière dont le fragment rempli l'espace via les balises layout_width et layout_height.

Et voilà, c'est terminé, la méthode onCreate de votre activité ressemble à ça :

@ copyright2012@android2ee

```
@Override
public void onCreate(Bundle savedInstanceState) {
	super.onCreate(savedInstanceState);
	//Définissez votre vue, rien de plus. Tout sera pris en charge par le
	fragment qui affiche les données
	setContentView(R.layout.activity_main);
	//Retrouver votre fragment en utilisant son identifiant (si besoin)
	MainFragmentHC
	mainFragment=(MainFragmentHC)findViewById(R.id.list_fragment);
}
```

Vous pouvez lancer votre application, elle s'exécutera.

2.1.3 Positionner les fragments

Le plus efficace pour positionner les fragments est d'utiliser la balise weight qui permet de répartir l'espace libre entre les composants graphiques affichés au sein d'un LinearLayout. Cela va vous permettre de les positionner où vous le souhaitez dans l'espace tout en leur donnant leur proportion les uns par rapport aux autres.

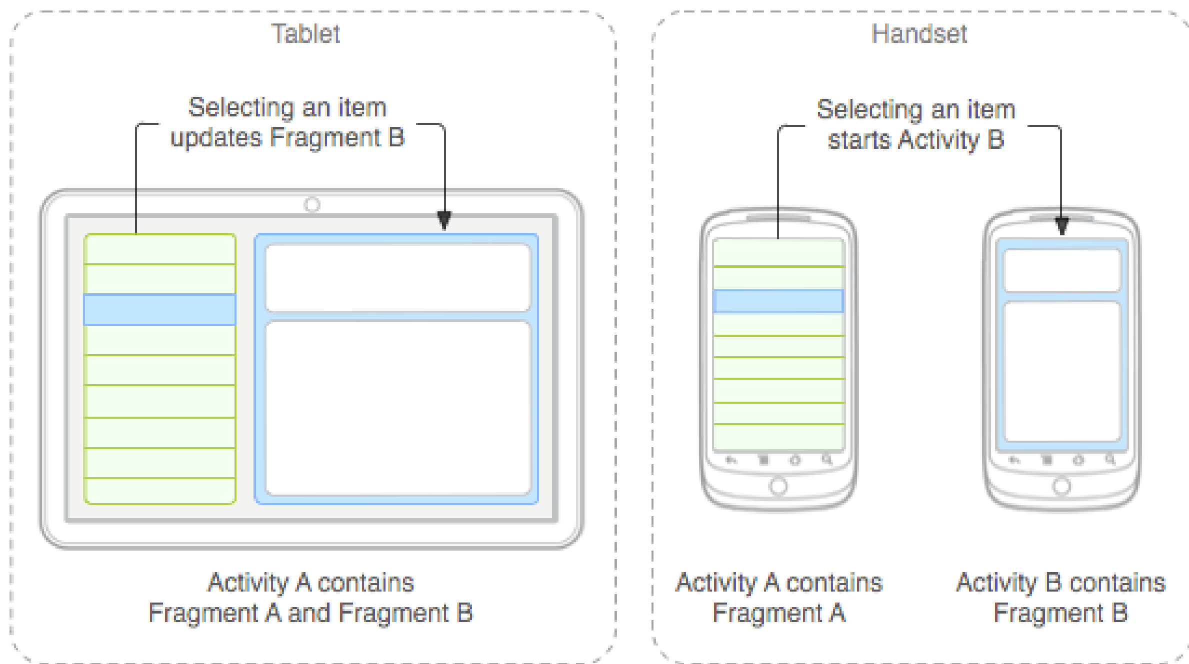
Ainsi vous souhaitez que le fragment A prenne 1/3 de l'espace en largeur et le fragment B 2/3, il vous suffit de les déclarer de la manière suivante :

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

	<fragment android:name="com.example.news.FragmentA"
	android:id="@+id/list"
	android:layout_weight="1"
	android:layout_width="0dp"
	android:layout_height="match_parent" />

	<fragment android:name="com.example.news.FragmentB"
	android:id="@+id/viewer"
	android:layout_weight="2"
	android:layout_width="0dp"
	android:layout_height="match_parent" />

</LinearLayout>
```

L'astuce ici est de demander une taille de 0dp aux fragments. L'espace libre restant correspond à la largeur complète du layout. La balise weight redistribue cet espace libre à chacun des composants 1/3 pour le fragment A qui pèse 1(car 1+2=3 et vaut le poids total) et 2/3 pour le fragment B qui pèse 2.

2.2 Structurer une application possédant des fragments

Revenons concrètement à l'exemple de Google sur les fragments :



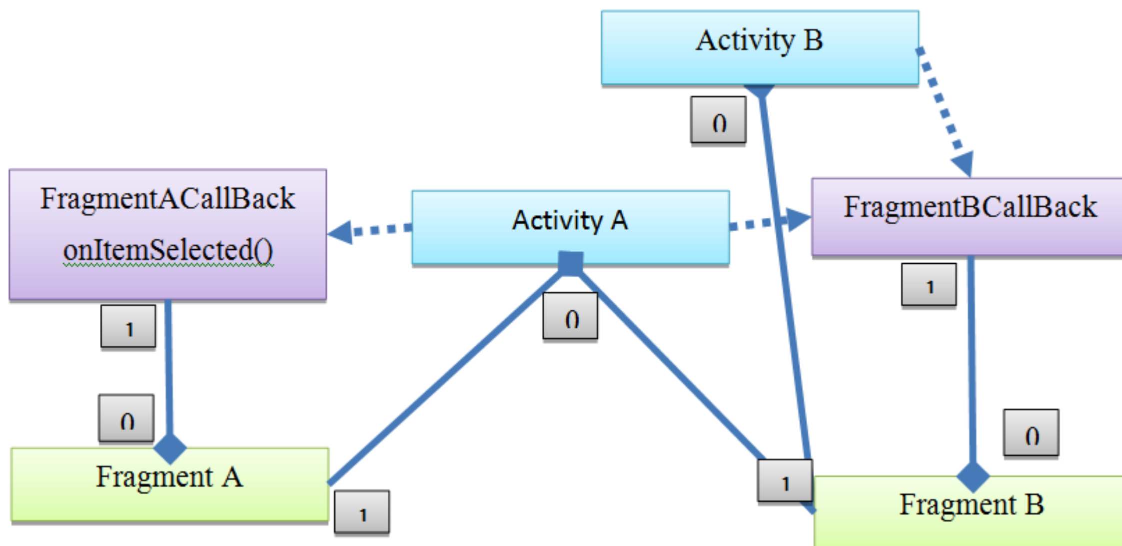
Il y a l'activité A qui possède un ou deux fragments en fonction de son contexte d'exécution. Le fragment A affiche une liste, le fragment B affiche l'item de la liste sélectionné.

Mais alors que fait la méthode onListItemSelected du fragment A, connait-elle fragment B ? Et comment communique-t-elle avec lui ?

2.2.1 Design Pattern

En fait le principe est le suivant :

@ copyright2012@android2ee



L'activité A connait les fragments qu'elle contient. Les fragments ne connaissent pas leur conteneur, mais uniquement les CallBack que l'activité A implémente.

L'activité a à charge l'orchestration de ses fragments, elle les écoute et les met à jour. C'est à elle de passer l'information de l'un à l'autre, de lancer une activité paire pour afficher le second (cas statique) ou de changer les fragments (cas dynamique).

2.2.2 La méthode onItemSelectedde l'activité A

Ainsi dans l'exemple de Google, le fragment A appelle l'activité A (via le onItemSelected de l'interface FragmentACallBack). L'activité A doit alors connaitre son contexte, si elle affiche deux fragments, elle doit mettre le second fragment à jour, si elle n'affiche qu'un fragment, elle doit lancer l'activité B.

En code cela donne :

```
//Quelque part dans ActivityA extends Activity
//Cas statique
public void onItemSelected(String id) {
        // Si deux panels sont visibles,
        if (mTwoPane) {
                ItemDetailFragment fragment =
                getSupportFragmentManager().getFragmentByTag(itemDetailTag);
                fragment.setItem(id)
        } else {
                // Lancement de l'activité B : Définition de l'Intent
                Intent detailIntent = new Intent(this, ItemDetailActivity.class);
                // Ajout des paramètres d'initialisation de l'activité
                detailIntent.putExtra(ItemDetailFragment.ARG_ITEM_ID, id);
                // Lancement de l'activité
                startActivity(detailIntent);
        }
  }
```

2.2.3 Structure du projet

Le projet ci-contre implique la structure suivante:

- src
  - ActivityA (extends Activity implements FragmentACallBack, FragmentBCallBack)
  - ActivityB (extends Activity implements FragmentBCallBack)
  - FragmentA (extends Fragment)
  - FragmentACallBack
  - FragmentB (extends Fragment)
  - FragmentBCallBack
- res\layout
  - activity_a_layout
  - activity_b_layout
  - fragment_a_layout
  - fragment_b_layout
- res\layout-port||res\layout-large||res\layout-xlarge
  - activity_a_layout

2.2.4 Sauvegarder et restaurer le contexte utilisateur

L'un des éléments les plus importants lorsque l'on développe une application Android est de ne pas perdre le contexte utilisateur quand celui-ci tourne son écran (entre autres). Ainsi quand l'application est détruite pour être immédiatement recréée, il nous faut sauvegarder puis restaurer ce contexte ; cela peut être les valeurs contenues dans les EditText, les choix des cases à cocher, l'item de la liste sélectionné, la valeur de la scrollView …

Pour mettre en place cette restauration, l'activité utilise les méthodes onRestoreInstanceState et onSaveInstanceState qui permettent de stocker dans un Bundle (une HashMap typée) les différents champs primitifs ou parcelables que l'on souhaite passer de l'activité mourante à l'activité recréée.

Un exemple de l'utilisation de ces méthodes dans une activité :

```
/******************************************************************************/
/** Managing Rotation change *****************************************/
/******************************************************************************/
/* * (non-Javadoc)
 *  * @see android.app.Activity#onRestoreInstanceState(android.os.Bundle)
 */
@Override
protected void onRestoreInstanceState(Bundle savedInstanceState) {
	super.onRestoreInstanceState(savedInstanceState);
	// Restauration des données du contexte utilisateur
	edtMessage.setText(savedInstanceState.getString(EDT));
	for (Parcelable human : savedInstanceState.getParcelableArrayList(LIST)) {
		messages.add((Human) human);
	}
}

/*
 * (non-Javadoc)
```


@ copyright2012@android2ee

```
 *
 * @see android.app.Activity#onSaveInstanceState(android.os.Bundle)
 */
@Override
protected void onSaveInstanceState(Bundle outState) {
	super.onSaveInstanceState(outState);
	// Sauvegarde des données du contexte utilisateur
	outState.putString(EDT, edtMessage.getText().toString());
	outState.putParcelableArrayList(LIST, messages);
}
```

Dans cet exemple, nous stockons et restaurons la chaîne de caractères contenue dans l'EditText edtMessage ainsi que la liste des objets (ici des humains) affichés par la listView.

La méthode onSaveInstanceState possède pour paramètre le bundle (outState) qui sera renvoyé en paramètre de la méthode onRestoreInstanceState (savedInstanceState). Ainsi, dans la méthode onSaveInstanceState on stocke les données à sauvegarder dans ce bundle et dans la méthode onRestoreInstanceState on les restaure à partir de ce bundle.

Ces méthodes sont appelées dans le cycle de vie de l'activité de la manière suivante :

```
E/MainActivity(10447): MainActivityHC Launched
E/MainActivityHC(10447): onCreate called
E/MainActivityHC(10447): onStart called
E/MainActivityHC(10447): onResume called
<-- Rotation de l'écran-->
E/MainActivityHC(10447): onPause called
E/MainActivityHC(10447): onSaveInstanceState called
E/MainActivityHC(10447): onStop called
E/MainActivityHC(10447): onDestroy called
E/MainActivityHC(10447): onCreate called
E/MainActivityHC(10447): onStart called
E/MainActivityHC(10447): onRestoreInstanceState called
E/MainActivityHC(10447): onResume called
```

2.2.4.1 Mise en place de la restauration avec les fragments

Pour mettre en place cette technique avec des fragments, il suffit de surcharger la méthode onSaveInstanceState, comme pour les activités. Par contre, la méthode onRestoreInstanceState n'existe plus dans les fragments, le bundle est renvoyé directement aux méthodes du cycle de vie du fragment. Ainsi, le Bundle outState est disponible en paramètre des méthodes:

- onCreate(**Bundle**),
- onCreateView(LayoutInflater, ViewGroup, **Bundle**), et
- onActivityCreated(**Bundle**).

Il vous suffit de restaurer vos données dans l'une de ces trois méthodes, au moment qui vous semble le plus approprié :

```
public void onActivityCreated(Bundle savedInstanceState) {
	super.onActivityCreated(savedInstanceState);
	if (savedInstanceState != null) {
		// Restauration des données du contexte utilisateur
		mCurCheckPosition = savedInstanceState.getInt("curChoice", 0);
	}
}
```

```
public void onSaveInstanceState(Bundle outState) {
	super.onSaveInstanceState(outState);
	// Sauvegarde des données du contexte utilisateur
	outState.putInt("curChoice", mCurCheckPosition);
}
```

Faîtes bien attention à vérifier que savedInstanceState est non-null (et donc que vous êtes bien dans le pattern destruction-recréation avant de restaurer les données).

Un dernier petit détail, mais on y reviendra, n'utilisez pas la méthode setRetainInstance sur ce type de fragment car alors le bundle SavedInstanceState est toujours null.

2.2.5 Fragment sans IHM : les working fragments

Tant qu'à être dans la problématique de la sauvegarde et de la restauration des données lors d'un changement de configuration de l'appareil (cas de la rotation de l'écran), penchons-nous sur le problème de la rétention d'objet qui ne peuvent être ni parcelables ni primitifs ni serializables.

L'exemple typique est la gestion des Threads (qui si vous avez une bonne architecture ne devrait jamais arriver). Vous souhaitez retenir l'instance de la Thread si votre activité est détruite pour être immédiatement reconstruite. Avant, vous utilisiez la méthode onRetainConfigurationInstance de la classe activité. Cette méthode est dépréciée depuis HoneyComb.

En effet, depuis HoneyComb, il est préconisé d'utiliser des fragments sans IHM à qui on demande de ne pas mourir lors du processus de destruction/recréation de l'activité qui le contient. Ainsi vous créez un fragment sans IHM (une vraie classe par exemple, MyWorkingFragment extends Fragment).

Pour qu'un fragment ne soit pas détruit lorsque l'activité est détruite pour être recréée, il suffit d'appeler la méthode **setRetainInstance(true)** sur ce fragment.

2.2.5.1 setRetainInstance

La méthode setRetainInstance contrôle si l'instance du fragment est retenue lors de la recréation de l'Activité (comme lors d'un changement de configuration). ***Elle peut être utilisée uniquement avec des fragments qui n'appartiennent pas à la BackStack*** *(car ils sont persistés quoi qu'il arrive). S'il est défini à true, le cycle de vie du fragment sera légèrement différent quand l'activité sera recréée:*

- *OnDestroy () ne sera pas appelée (mais onDetach () le sera toujours, parce que le fragment est détaché de son activité actuelle).*
- *onCreate (Bundle) ne sera pas appelée car le fragment n'est pas recréé.*
- *onAttach (Activity) et onActivityCreated (Bundle) seront encore appelées. (JavaDoc)*

Cette méthode ne doit pas être appelée sur les fragments normaux, sinon dîtes adieu à votre BackStack de fragments.

2.2.5.2 Exemple

La mise en place de ce principe est assez simple: quelque part dans votre code vous appelez sur le dit fragment (en interne ou pas) **setRetainInstance(true)**:

```
private static final String WORKER_FRAGMENT_TAG="WORKER_FRAGMENT_TAG";

// Quelque part dans votre code
workerFragment.setRetainInstance(true);

//***************/
// Ailleurs dans votre code (dans onCreate, onResume ou onStart)
FragmentManager fm = getSupportFragmentManager();
FragmentTransaction ft=fm.beginTransaction();
ft.add(workerFragment, WORKER_FRAGMENT_TAG);
```


@ copyright2012@android2ee

```
ft.commit();
//****************/

//Ailleurs dans le code
//Pour récupérer votre working fragment
WorkerFragment
workerFragment=(workerFragment)fm.findFragmentByTag(WORKER_FRAGMENT_TAG);
```

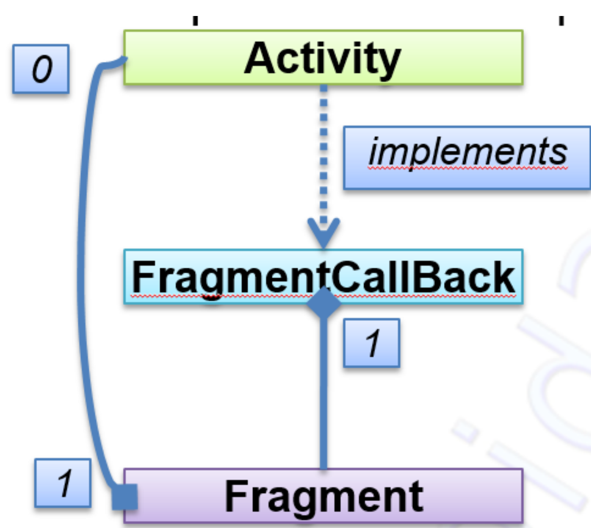
Il faut ainsi utiliser le tag pour identifier le fragment (en effet n'ayant pas d'IHM, il ne possède pas d'identifiant).

2.2.6 Design Pattern de communication fragment-activité

Un élément à bien comprendre dans l'utilisation des fragments : L'objectif d'un fragment est d'être utilisé par plusieurs activités. De ce fait, le fragment ne peut pas connaître l'activité qui le contient. Mais alors comment communique-t-il avec elle ? L'idée est simple, toute activité qui souhaite contenir un fragment implémente une interface qui sera utilisée par le fragment pour communiquer avec son conteneur.

Inversement, une activité connait les fragments qu'elle contient, elle peut donc pointer directement sur eux.

Ce principe se résume ainsi :



Un exemple :

L'interface :

```
public interface MainFragmentCallBack {
	/**
	* Un item a été sélectionné dans le fragment
	* En tant que CallBack du fragment, vous devriez faire quelque chose avec cette information
	* @param itemId
	*/
	public void onItemSelected(int itemId);
}
```

La classe de l'activité :

```
public class MainActivityHC extends Activity implements MainFragmentCallBack {
/*Some code*/
```

```
        /* * (non-Javadoc)
         *
         * @see
        com.android2ee.tuto.fragment.fragmentstatic.tuto.view.honeycomb.main.MainFra
        gmentCallBack#onItemSelected(int)
         */
        @Override
        public void onItemSelected(int itemId) {
        //Faîtes votre traitement ici
        }
}
```

La classe du fragment :

```
public class MainFragmentHC extends Fragment {
        /**
         * La parent du fragment (son callback/sonactivité contenante)
         */
        private MainFragmentCallBack parent;

        /*
         * (non-Javadoc)
         *
         * @see android.app.Fragment#onAttach(android.app.Activity)
         */
        @Override
        public void onAttach(Activity activity) {
                Log.w("MainFragmentHC", "onAttach called");
                super.onAttach(activity);
                //Utiliser cette méthode pour lier votre fragment avec son callback
                parent = (MainFragmentCallBack) activity;
        }

        /**
         * Un item a été sélectionné, notifier le changement
         * @param position of the item
         */
        public void onItemSelected(int position) {
                //Notifiez le parent qu'un item a été sélectionné
                parent.onItemSelected(position);
                //Faîtes d'autres traitements ici au besoin
        }
}
```

Ce pattern est obligatoire dès lorsqu'un fragment est contenu par plus d'une activité, il est recommandé dans tous les cas.

Inversement, vous pouvez, dans vos fragments, appeler la méthode getActivity pour récupérer le contexte si vous en avez besoin. Il vous faut alors n'utiliser que les méthodes de la classe activité (pas de méthode spécifique).

2.2.7 Gestion des menus

Il est naturel que les fragments souhaitent rajouter des MenuItem quand ils sont actifs. En effet, ils doivent porter les actions associées aux éléments qu'ils affichent.

Pour cela vous devez **dans le fragment**:

- invoquer **setHasOptionMenu(true)** dans la méthode onCreate du fragment;
- Surcharger les méthodes **onCreateOptionsMenu** et **onOptionsItemSelected**, comme pour les activités;

@ copyright2012@android2ee

Dans l'activité:

- Surcharger les méthodes **onCreateOptionsMenu** et **onOptionsItemSelected**, comme d'habitude et rajouter dans le default un appel au super pour que les menus des fragments soient pris en compte;

Ainsi :

Le code de l'activité:

```
 public boolean onCreateOptionsMenu(Menu menu) {
	//Appelez le super pour collecter les menu items de vos fragments
	super.onCreateOptionsMenu(menu);
	//Construisez le menu de l'activité
	getMenuInflater().inflate(R.menu.activity_menu, menu);
	//et voilà
	return true;
}
@Override
public boolean onOptionsItemSelected(MenuItem item) {
	switch (item.getItemId()) {
	case R.id.menu_activity_item1:
		// Faire un truc, ce menu item appartient à l'activité, c'est à elle de le traiter
		return true;
	default:
		//Appelez vos fragments pour qu'il vérifie si ce menu item n'est pas sous leur
		responsabilité
		return super.onOptionsItemSelected(item);
}}
```

Le code du fragment:

```
public void onCreate(Bundle savedInstanceState) {
	super.onCreate(savedInstanceState);
	setHasOptionsMenu(true);
}
public void onCreateOptionsMenu(Menu menu, MenuInflater inflater) {
	inflater.inflate(R.menu.fragment_menu, menu);
	super.onCreateOptionsMenu(menu);
}
public boolean onOptionsItemSelected(MenuItem item) {
	switch (item.getItemId()) {
	case R.id.fragment_menu_item1:
		// Faîtes un truc, ce menu item appartient au fragment, c'est à lui de le traiter
		returntrue;
	default:
		//Continuez à chercher quelqu'un qui va gérer ce menu
		returnsuper.onOptionsItemSelected(item);
}}
```

2.2.8 Nombre de fragments affichés et structuration des layouts

Vous verrez qu'un élément fondamental dans vos gestions des fragments est de répondre à la question suivante « Combien de fragment suis-je en train d'afficher ?». Certains font des calculs complexes ou demande aux fragments de dire à l'activité qu'ils sont visibles ou je ne sais quel algorithme tordu.

Restez simple et efficace, utilisez les ressources !

Tout d'abord, vous allez définir vos fichier de layouts dans le dossier res\layout (et pas de suffixe). Là vous allez définir vos fichiers**main_activity_two_panes.xml**, **main_activity_one_pane.xml** (et d'autres à 3,4 ou plus de fragments, si vous en avez besoin). Ainsi tous vos fichiers de description des IHM sont au même endroit.

Maintenant, votre activité va charger le fichier main_activity.xml (qui n'existe pas, on est bien d'accord) :

```
public void onCreate(Bundle savedInstanceState) {
	super.onCreate(savedInstanceState);
	//Construisez la vue de l'activité, rien de plus. Le fragment prend en
	charge le reste
	setContentView(R.layout.main_activity);
}
```

Et là vous allez utiliser les redirections dans vos ressources :

Pour cela, vous allez créer les dossiers values-large, values-land, values-large-land et dans chacun de ces dossiers vous allez créer le fichier refs.xml (vous auriez pu l'appeler toto.xml mais refs possède une sémantique plus claire). Ce fichier refs.xml est le suivant :

```
<resources>
<item name="main_activity" type="layout">@layout/activity_two_panes</item>
<bool name="twoPane">true</bool>
</resources>
```

D'une part il redirige le layout main_activity.xml vers le layout activity_two_pane et d'autre part il spécifie le booléen twoPane à true, car le layout activity_two_panes possède bien deux fragments (two panes signifiant deux panneaux).

Si vous souhaitiez que dans cette configuration, le layout ne possède qu'un seul fragment, il suffisait de rediriger vers activity_one_pane :

```
<resources>
<item name="activity_main" type="layout">@layout/main_activity_one_pane</item>
<bool name="twoPane">false</bool>
</resources>
```

Enfin, vous auriez pu remplacer le booléens twoPane par un entier :

```
<resources>
<item name="activity_main" type="layout">@layout/main_activity_one_pane</item>
<int name="fragmentNumber">3</int>
</resources>
```

Dans votre activité, ou n'importe où dans votre code, pour connaitre ce nombre de fragments, ou le booléen twoPane, il vous suffit de faire l'appel suivant :

```
//Récupérez le nombre de fragments affichés
boolean twoPane = getResources().getBoolean(R.bool.twoPane);
```

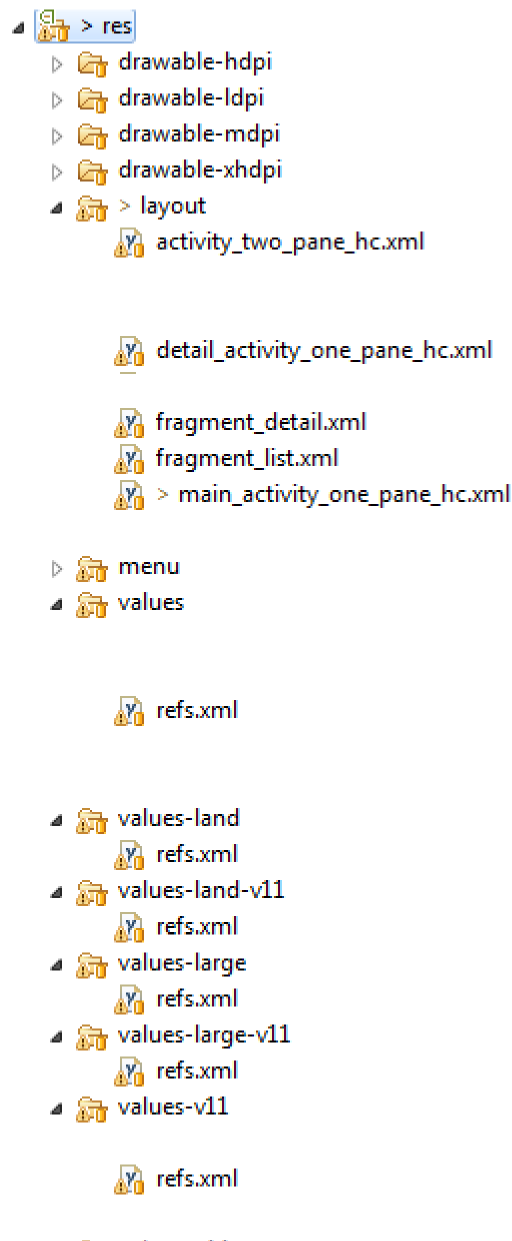
Vous avez ainsi accompli deux choses :

- Pas de duplication inutile des fichiers de layout (factorisation et redirection) ;
- Parfaite connaissance du nombre de fragments affichés (contexte des fragments).

Et le projet, votre équipe et l'équipe de maintenance vous en remercie.

Votre projet est structuré de la manière suivante :

@ copyright2012@android2ee



3 Exercice : Migration d'une application existante

Je vous propose un exercice à cette étape. Cela vous rassurera et vous verrez que même si ce que je dis paraît compliqué, cela ne l'est plus quand on l'a fait une fois.

Vous allez prendre l'un de vos projets qui n'a pas de fragment (commencez par un petit projet :) et nous allons le migrer l'activité principale ensemble.

0. Ajoutez la support librairy à votre projet.

Clic droit sur le projet Android->AddSupportLibrairy

1. Créez le fragment :

```
public class MyFragment extends android.support.v4.app.Fragment {
```

2. Créez le fichier myfragment_layout.xml de layout de votre Fragment dans res\layout. Copiez collez le code xml du layout de votre activité dans ce nouveau fichier.

3. Modifiez le fichier de layout de votre activité pour qu'il affiche le fragment. Vous allez ainsi remplacer le contenu du fichier par un truc qui ressemble à ça :

```
<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" >
                <fragment
                android:id="@+id/myfragment"
                android:name="yourPackage.MyFragment"
                android:layout_width="match_parent"
                android:layout_height="match_parent" />
</LinearLayout>
```

Il faut remplacer ***yourPackage*** par le package qui contient votre classe MyFragment.

4. Ouvrez la classe de votre activité et modifiez là ainsi :

```
public class MainActivity extends FragmentActivity{
        /**
         * Le fragment
         */
        MyFragment fragment;

        @Override
        protected void onCreate(Bundle savedInstanceState) {
                super.onCreate(savedInstanceState);
                setContentView(R.layout.activity_main);
                fragment = (MyFragment)
                getSupportFragmentManager().findFragmentById(R.id.myfragment);

        }
```

Notez que l'identifiant du fragment est celui que vous lui avez donné dans sa déclaration dans le fichier de layout de l'activité.

Notez aussi que votre activité étend maintenant FragmentActivity

5. Ouvrez les classes MyFragment et celle de votre activité côte à côte (ça va aider).

6. Il va falloir maintenant migrer le code de l'activité vers le fragment. La première étape est la ventilation de la méthode onCreate de l'activité dans les méthodes onActivityCreate et onCreateView du fragment.

Commencez par coupez/collez tous les champs graphiques (TextView, EditText,…) de l'activité dans le fragment.

Dans onCreateView, vous construisez votre vue puis vous instancier vos éléments graphiques. Ainsi vous coupez/collez toutes les lignes contenant un findViewById du onCreate de l'activité vers le onCreateView du fragment.La méthode onCreateView ressemble à ça :

```
// Dans MyFragment
public View onCreateView(LayoutInflater inflater, ViewGroup container, Bundle
savedInstanceState) {
        View view = inflater.inflate(R.layout.fragment_layout, container, false);
        //Exemple d'instanciation d'éléments graphiques. Ils devraient être ceux de votre activité.
        btnAdd = (Button) view.findViewById(R.id.btnAdd);
        lsvWordsList = (ListView) view.findViewById(R.id.lsvWordsList);
```


@ copyright2012@android2ee

```
        edtMessage = (EditText) view.findViewById(R.id.edtMessage);
        return view;
}
```

Dans la méthode onActivityCreate vous coupez/collez toutes les lignes qui restent dans le onCreate de l'activité. Du coup, elle ressemble à quelque chose comme ça :

```
// Dans MyFragment
private Context ctx ;
public void onActivityCreated(Bundle savedInstanceState) {
        ctx = getActivity();
        if (savedInstanceState != null) {
                edtMessage.setText(savedInstanceState.getString(EdtStorageKey));
        }
        //Instanciation des éléments de la ListView
        humans = new ArrayList<Human>();
        humanAdapter = new HumanAdapter(ctx, humans);
        lsvWordsList.setAdapter(humanAdapter);
        // Ajout des listeners
        btnAdd.setOnClickListener(new OnClickListener() {
                public void onClick(View v) {
                        add();}
                });
        lsvWordsList.setOnItemClickListener(new OnItemClickListener() {
                public void onItemClick(AdapterView<?> parent, View view, int position, long id) {
                        listItemSelected(position);}
                });
        super.onActivityCreated(savedInstanceState);
}
```

Si un champ de cette méthode n'appartient pas au fragment, coupez/collez sa déclaration de l'activité dans le fragment.

La méthode onCreate de l'activité devrait ressembler à cela :

```
// Dans l'activité
@Override
protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        // Construire la vue
        setContentView(R.layout.activity_main);
        myFragment=(MyFragment)
        getSupportFragmentManager().findFragmentById(R.id.myfragment);
}
```

7.Il ne vous reste plus qu'à migrer toutes les méthodes de l'activité qui sont associées à la gestion de l'I.H.M. dans le fragment.

Pour les migrer, vous les coupez/collez et vous faîtes pareils avec les attributs de classe qu'elles utilisaient dans l'activité.

Votre activité ne doit alors plus contenir que les méthodes qui affichent les fenêtres de dialogues, gère le cycle de vie, les menus. Votre fragment doit gérer tous les évènements de l'IHM.

Si dans votre fragment le code ne marche pas,c'est souvent qu'il leur manque le contexte. Ne vous inquiétez pas, vous avez le contexte. Vous l'avez instancié dans la méthode onActivityCreated, il vous suffit de rajouter ctx devant les méthodes qui ne marchent pas.

Par exemple vous avez coupez/collez :

```
humanAdapter = new HumanAdapter(this, humans);
```

Remplacez this par ctx :

```
humanAdapter = new HumanAdapter(ctx, humans);
```

Et votre code compile de nouveau.

Si vous avez besoin d'appeler une méthode de l'activité à partir du fragment, cela signifie que vous devez créer la classe MyFragmentCallBack et y définir cette méthode. Puis vous allez implémenter l'interface MyFragmentCallBack par l'activité et vous pourrez ainsi appeler cette méthode via myFragmentCallBack.maMethode() ; Dans ce cas, n'oubliez pas de rajouter le callBack à votre fragment :

```
// Dans MyFragment
public void onAttach(Activity activity) {
	super.onAttach(activity);
	myCallBack=(MyFragmentCallBack)activity;
}
```

Normalement, vous êtes capable de terminer la migration en corrigeant les erreurs de compilation assez facilement.

8. A ce stade vous avez fini. Votre activité est quasi-vide et votre fragment gère l'interface graphique.

4 Multi versionning avec les fragments

Dans la conférence « 106 - Multiversioning Android User Interfaces - I_O 2012 » de Bruno Oliveira et Adam Powell des GoogleI/O 2012, ils nous ont expliqué comment gérer le multi-versionning avec les fragments. C'est-à-dire comment mettre en place les fragments quels que soit la version du système.

Je vais aussi vous l'expliquer. Mais surtout je vais vous expliquer pourquoi il ne faut pas mettre en place leur solution.

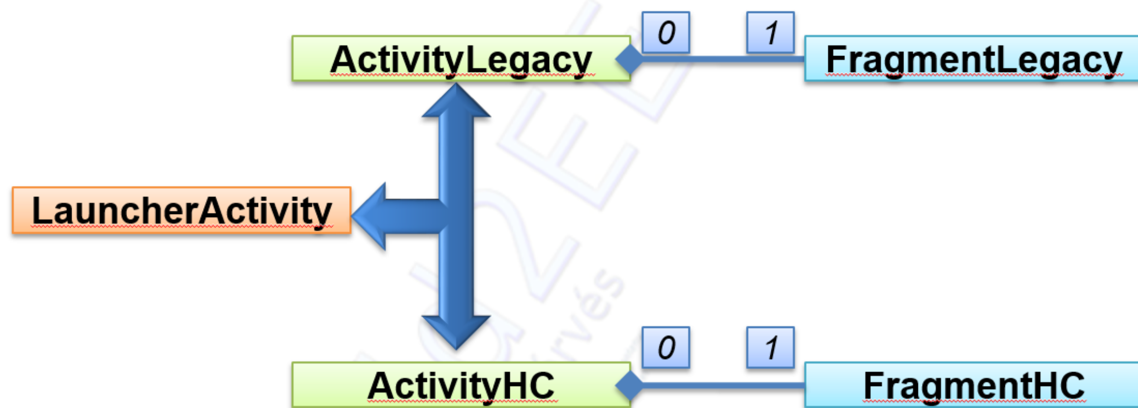
Sachez que si vous utilisez la support-library pour définir tous les fragments de votre application, même si votre application est exécutée sur HoneyComb ou plus, le système ne switchera pas sur les fragments natifs.

4.1.1 Fragments natifs et Fragments de la support-librairy : Le parrallel activity pattern

C'est la solution expliquée par A. Powell et B. Oliveira.

Si vous souhaitez utiliser les fragments natifs quand vous êtes au-dessus d'HoneyComb et les fragments de la support-librairy quand vous êtes au-dessous, il vous faut mettre en place le Parrallel activity pattern :

@ copyright2012@android2ee



Pour cela, vous allez déclarer dans votre manifest.xml, l'activité LauncherActivity comme étant votre activité de lancement :

```
<manifestxmlns:android="http://schemas.android.com/apk/res/android"
        package="com.android2ee.tuto.fragment.fragmentstatic.tuto"
        android:versionCode="1"
        android:versionName="1.0">

<uses-sdk
        android:minSdkVersion="8"
        android:targetSdkVersion="15"/>

<application
        android:name="MApplication"
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name"
        android:theme="@style/AppTheme">

        <activity
        android:name=".LauncherActivity"
        android:label="@string/title_activity_main">
        <intent-filter>
                <action android:name="android.intent.action.MAIN"/>
                <category android:name="android.intent.category.LAUNCHER"/>
        </intent-filter>
        </activity>

        <activity android:name=".view.honeycomb.main.MainActivityHC">
        </activity>

        <activity android:name=".view.legacy.main.MainActivityLegacy">
        </activity>

        <activity android:name=".view.honeycomb.detail.DetailActivityHC">
        </activity>

        <activity android:name=".view.legacy.detail.DetailActivityLegacy">
        </activity>
</application>

</manifest>
```

Vous déclarez aussi les autres activités qui peuplent votre application, comme d'habitude.

Dans l'activité LauncherActivity vous n'effectuez qu'une seule chose, vous regarder la version de l'appareil sur laquelle s'exécute votre application et en fonction de cette version, vous lancez soit l'activité spécifique à HoneyComb (avec les fragments natifs) soit la version legacy (avec les fragments de la support-librairy).

Le code de l'activité est le suivant :

```
public class LauncherActivity extends Activity {
@Override
publicvoid onCreate(Bundle savedInstanceState) {
	super.onCreate(savedInstanceState);
	// Définissez l'Intent
	Intent startActivityIntent = null;
	// Trouvez la version (post ou pre HoneyComb)
	boolean postHC = getResources().getBoolean(R.bool.postHC);

	// Instanciez l'Intent en fonction de cette version
	if (postHC) {
		startActivityIntent = new Intent(this, MainActivityHC.class);
	} else {
		startActivityIntent = new Intent(this, MainActivityLegacy.class);
	}

	// Lancez l'activité :
	startActivity(startActivityIntent);
	// Et suicidez vous : soyez sûr de ne pas appartenir à la backstack
	finish();
	}
}
```

Pour connaitre la version de l'appareil, les ressources sont une fois de plus utilisées et déclarent le booléen postHC de la manière suivante :

Dans le dossier res\values, vous avez définis le fichier version.xml :

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

Et dans le dossier values-v11, vous avez défini le fichier version.xml :

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

Ainsi l'appel à `getResources().getBoolean(R.bool.postHC)` vous renvoie auto-magiquement la version du système sur laquelle s'exécute l'application. Et en fonction de cette version vous lancez l'activité adaptée.

Maintenant que vous avez fait ça, il ne vous reste plus qu'à définir votre activité post HoneyComb et votre activité preHoneyComb. L'activité postHoneyComb est celle expliquée depuis le début de cet article, apprenons à mettre en place l'activité preHoneyComb avec la support-librairy.

@ copyright2012@android2ee

4.1.1.1 *Utilisation de finish*

Remarque importante : La LauncherActivity finit son code par un appel à finish(), c'est essentiel. Cela signifie à l'activité qu'elle doit mourir et surtout ne pas se placer dans la backstack. En effet, si on oublie cette ligne, quand la nouvelle activité est lancée, LauncherActivity se place dans la backstack. Quand l'utilisateur quitte l'activité Main (post ou pre HoneyComb, cela n'a pas d'importance), cela relancera alors automatiquement LauncherActivity (car on dépile la backstack). Et comme LauncherActivity relance MainActivity, votre utilisateur ne pourra pas sortir de MainActivity…

4.1.2 Mise en place des Fragments de la support-library

Tout d'abord, ajoutez la support-library à votre projet : Clic droit sur le projet Android->Add Support Librairy (si une erreur survient, mettre à jour la support-library via le SDK manager).

4.1.2.1 *Migration de l'activité*

1. Commencez par coder l'activité post-HoneyComb.
2. Créez votre activité preHoneyComb, que nous appellerons ActivityLegacy. Copiez-collez le code de l'activité postHoneyComb dans l'activité que vous venez de créer.
3. Modifier ActivityLegacy pour qu'elle hérite de FragmentActivity (et non plus d'Activity). Vous pouvez aussi hériter de ActionBarActivity si vous souhaitez faire apparaître l'action bar quelle que soit la version de l'appareil (mais c'est encore une autre histoire que je ne vous conterai pas dans cet article).
4. Puis supprimez tous les imports de la classe ActivityLegacy et remplacez-les par ceux de la support-librairy.

Voilà, vous avez fini.

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

public class ItemListActivity extends FragmentActivity {
@Override
public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_item_list);
        FragmentManager fm=getSupportFragmentManager() ;
        .... blabla ....}
  }
```

Un dernier détail, la récupération du FragmentManager s'effectue via la méthode getSupportFragmentManager (et non plus par getFragmentManager).

4.1.2.2 *Migration des fragments*

Pour migrer un fragment natif en fragment de la support-library, rien de plus simple, vous copier-coller la classe du fragment natif (celui codé pour les versions post-HoneyComb). Vous supprimez ces imports et vous les remplacez par ceux de la support-library.

4.1.3 Migration des fichiers de layout

Dans le cas des fragments statiques, le fichier de layout de l'activité déclare quel est la classe qui instancie le fragment. Ainsi, vous devez dupliquer tous vos fichiers de layout de vos activités et pour ceux qui implémentent les activités legacy, modifier le chemin de la classe qui implémente le fragment pour le faire pointer vers la classe monFragmentLegacy.

Le fichier de layout main_ActivityHC.xml ressemble à cela :

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

```
    android:layout_height="fill_parent"
    android:orientation="vertical">

    <fragment
        android:id="@+id/list_fragment"
        android:name="com.android2ee.tuto.fragment.fragmentstatic.tuto.view.honeycomb
.main.MainFragmentHC"
        android:layout_width="fill_parent"
        android:layout_height="fill_parent"
        android:layout_weight="1"
        android:orientation="horizontal">
    </fragment>

</LinearLayout>
```

Et le fichier de layout main_Activity_Legacy ressemble à cela :

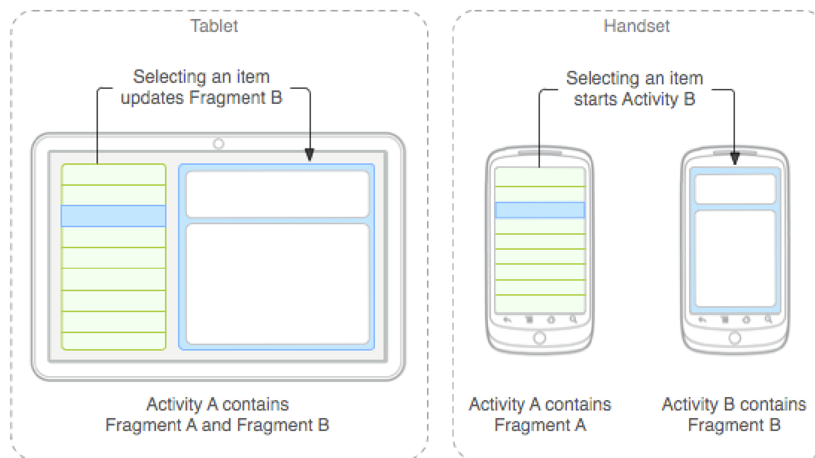
```
<?xml version="1.0"encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:orientation="vertical">

    <fragment
        android:id="@+id/list_fragment"
        android:name="com.android2ee.tuto.fragment.fragmentstatic.tuto.view.legacy.mai
n.MainFragmentLegacy"
        android:layout_width="fill_parent"
        android:layout_height="fill_parent"
        android:layout_weight="1"
        android:orientation="horizontal">
    </fragment>

</LinearLayout>
```

4.1.4 Duplication du code

Vous venez de mettre en place le parrallel activity pattern, bravo. Maintenant regardez votre code, vous avez dupliqué toute vos classes d'activité et de fragment… En fait, vous avez fait pire, vu que dans le fichier de layout de l'activité, la classe qui implémente le fragment est mentionnée, vous avez aussi été obligé de dupliquer votre fichier de layout de vos activités.

Revenons à l'exemple de Google :

@ copyright2012@android2ee



Ainsi si vous n'aviez pas mis en place le multi-versionning votre projet ressemblait à cela :

- src
  - ActivityA_HC
  - ActivityB_HC
  - FragmentA_HC
  - FragmentB_HC
- res\layout-port-v11||res\layout-large-v11||res\layout-xlarge-v11
  - activity_a_layout_hc
- res\layout
  - activity_a_layout_hc
  - activity_b_layout_hc
  - fragment_a_layout
  - fragment_b_layout

Et maintenant, avec le Parrallel Activity Pattern, votre projet ressemble à cela (en rouge les duplications):

- src
  - ActivityA_HC
  - ActivityB_HC
  - FragmentA_HC
  - FragmentB_HC
  - ActivityA_Leg
  - ActivityB_Leg
  - FragmentA_Leg
  - FragmentB_Leg
- res\layout-port||res\layout-large||res\layout-xlarge
  - activity_a_layout_leg
- res\layout-port-v11||res\layout-large-v11||res\layout-xlarge-v11

- activity_a_layout_hc
- res\layout
    - activity_a_layout_hc
    - activity_b_layout_hc
    - activity_a_layout_leg
    - activity_b_layout_leg
    - fragment_a_layout
    - fragment_b_layout

4.1.5 Conclusion concernant le multi-versionning : Use only the support-librairy

Le choix est donc soit d'utiliser uniquement la support-librairy soit de dupliquer votre code (Activité, layout et Fragment) en ne changeant que les imports et le type d'animation puis d'utiliser le parrallel activity pattern pour être le plus adapté à toutes les versions du système. Donc :

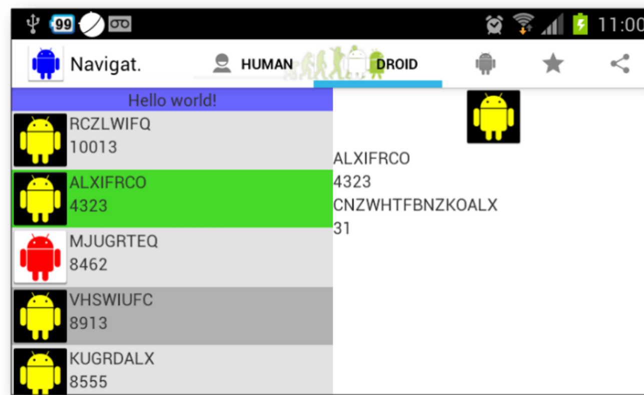
N'utilisez pas les fragments natifs, utilisez toujours la support-librairy !

Concrètement, ajoutez la support-librairy à votre projet et n'utilisez que les imports de la support-librairy quand vous codez votre projet. Pour le reste, vous ne verrez pas de différence.

5 Fragment dynamique

Une seconde manière d'utiliser les fragments est de mettre en place une activité qui va les ajouter, les remplacer, les supprimer dynamiquement. Ainsi, votre application ne possède qu'une activité et elle a à charge la gestion de l'affichage des fragments.

@ copyright2012@android2ee



L'utilisateur final ne verra aucune différence.

5.1 Mise en place

La mise en place est quasi-identique à celle des fragments statiques :

- Vos classes de type fragment ne changent pas ;
- Vos interfaces de callback ne changent pas.

Les fichiers de layout de vos activités eux remplacent la balise fragment par un layout (LinearLayout souvent).

Cela signifie simplement qu'avant vous aviez un layout pour votre activité qui ressemblant à cela :

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

        <fragment
                android:id="@+id/list"
                android:layout_width="0dp"
                android:layout_height="fill_parent"
                android:layout_weight="1"
                android:name="com.android2ee.tuto.fragment.sample1.ListFragment"
                android:orientation="horizontal" >
        </fragment>
        <fragment
                android:id="@+id/detail"
                android:layout_width="0dp"
                android:layout_height="fill_parent"
                android:layout_weight="1"
                android:name="com.android2ee.tuto.fragment.sample1.DetailFragment"
                android:orientation="horizontal" >
        </fragment>
</LinearLayout>
```

Avec les fragments dynamiques, votre fichier de layout ressemble à ça :

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayoutxmlns:android="http://schemas.android.com/apk/res/android"
        android:layout_width="fill_parent"
        android:layout_height="fill_parent"
        android:orientation="horizontal">
```

```
    <LinearLayout
        android:id="@+id/firstpane"
        android:layout_width="0dp"
        android:layout_height="fill_parent"
        android:layout_weight="1"
        android:orientation="horizontal">
    </LinearLayout>

    <LinearLayout
        android:id="@+id/secondpane"
        android:layout_width="0dp"
        android:layout_height="fill_parent"
        android:layout_weight="1"
        android:orientation="horizontal">
    </LinearLayout>
</LinearLayout>
```

C'est-à-dire qu'ils ressemblent à un bon vieux fichier de layout.

5.2 Gestion des Fragments : FragmentManager et FragmentTransition

Maintenant, il faut pouvoir savoir remplacer/ajouter ou supprimer des fragments. Cela se fait toujours de la même manière :

- Instanciation du FragmentManager et d'une FragmentTransaction ;
- Ajout des animations entrantes et sortantes ;
- Vérification que le fragment à créer appartient (ou pas) à la backstack ;
    - S'il n'appartient pas à la backstack on le créé et on ajoute/remplace le fragment existant par celui que l'on vient de créer ;
    - S'il appartient à la backstack,
        - soit onconsidère que l'on dépile la backstack (en d'autres termes dans votre tête, l'utilisateur a appuyé sur le bouton back), donc on « pop » la backstack pour afficher le fragment ;
        - soit onconsidère que l'on avance dans la navigation (même si le fragment a déjà été présenté à l'utilisateur), dans ce cas on ré-affiche le fragment ;
- Ajout de la transaction à la backstack ;
- Commit de la transaction.

```
public void showAnotherFragment() {
// Récupérez le FragmentManager
FragmentManager fm = mainActivity.getFragmentManager();
// Débutez la transaction des fragments
FragmentTransaction fTransaction = fm.beginTransaction();
// Définissez les animations entrantes et sortantes
fTransaction.setCustomAnimations(R.anim.anim_push_left_in,
        R.anim.anim_push_left_out,
        R.anim.anim_push_left_in,
        R.anim.anim_push_left_out);
// Trouvez si le fragment à afficher existe déjà
AnotherFragmentHC eventsListFragment = (AnotherFragmentHC)
fm.findFragmentByTag(anotherFragmentTag);
 // Si le fragment n'existe pas, il faut le créer
if (eventsListFragment == null) {
        eventsListFragment = new EventsListFragmentHC();
        // Ajoutez le nouveau fragment (Dans ce cas précis, un fragment est déjà affiché à cet
        emplacement, il faut donc le remplacer et non pas l'ajouter)
```


@ copyright2012@android2ee

```
        fTransaction.replace(R.id.mainfragment, eventsListFragment, anotherFragmentTag);
}else{
        // Le fragment existe déjà, il vous suffit de l'afficher
        fTransaction.show (eventsListFragment);
}
// Ajoutez la transaction à la backstack pour la dépiler quand l'utilisateur appuiera sur back
fTransaction.addToBackStack(mainActivity.getString(R.string.main_htitle));
// Faîtes le commit
fTransaction.commit();
}
```

Remarquez que nous n'utilisons plus les identifiants des fragments mais leur tag, en effet, vous n'avez plus la possibilité de leur donner un identifiant.

5.3 Initialisation

Lors de l'initialisation, il faut être attentif au cas de recréation de l'activité pour ne pas instancier plusieurs fois le fragment. En effet, les fragments sont automatiquement restaurés (bien que détruit entre temps, cela n'a rien à voir avec le RetainInstance qui lui ne détruit pas le fragment et ne s'applique qu'aux fragments sans IHM, rappelons-le).

```
public class MainActivity extendsActivity {

private String ListFragmentTag = "listfragmenttag";

@Override
public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        //Cependant, si le fragment est restauré à partir d'un état précédent, alors il n'y a pas besoin
        d'effectuer quoi que ce soit et il faudrait juste effectuer le return, sinon nous pourrions nous
        retrouver avec des fragments qui se chevauchent.
        if (savedInstanceState == null) {
                // Récupérez le FragmentManager
                FragmentManager fm = getFragmentManager();
                // Trouver si le fragment que nous souhaitons afficher appartient à la backstack
                ListFragment firstFragment = (ListFragment)
                fm.findFragmentByTag(ListFragmentTag);
                if (null == firstFragment) {
                        // Créez le fragment
                        firstFragment = new ListFragment();
                }
                //Dans notre cas, l'activité a été démarrée avec des paramètres spécifiques contenus
                dans l'Intent. Passez ces paramètres à votre fragment
                firstFragment.setArguments(getIntent().getExtras());
                // Ajoutez le fragment à son layout et effectuez le commit
                fm.beginTransaction().add(R.id.myListView, firstFragment).commit();
        }
        }
}
```

5.3.1 Passage de paramètres lors de la création du fragment

Pour instancier un fragment en lui passant des paramètres, il suffit d'utiliser la méthode setArguments.

```
Bundle arguments = new Bundle();
arguments.putString(ItemDetailFragment.ARG_ITEM_ID, id);
ItemDetailFragment fragment = new ItemDetailFragment();
fragment.setArguments(arguments);
```

Le fragment n'a plu qu'à appeler la méthode getArguments (n'importe où dans son code) pour récupérer le bundle associé à sa création.

```
if (getArguments().containsKey(ARG_ITEM_ID)) {
mItem = DummyContent.ITEM_MAP.get(getArguments().getString(ARG_ITEM_ID));
        }
```

Pour information, avec une activité on utilise le *setExtras* et le *getIntent().getExtras*.

5.3.2 Bonne pratique du cas particulier de la rotation de l'appareil

Le corolaire de cette restauration automatique des fragments lors de la destruction/recréation de l'activité est la bonne pratique suivante :

Tous vos layouts déclinez en mode landscape-portrait doivent avoir les mêmes layouts container même s'ils ne sont pas utilisés.

En d'autres termes, votre activité affiche 2 fragments dans le mode landscape et un seul en mode portrait (ou vis-versa), vous devez déclarer les 2 LinearLayouts pour recevoir les deux fragments dans les deux fichiers xml de layout de l'activité. C'est à dire :

res\layout\main_layout.xml

(Un seul fragment)

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
... blabla ...>
        <FrameLayout android:id="@+id/item_list_container"
                android:layout_width="0dp"
                android:layout_height="match_parent"/>

        <FrameLayout android:id="@+id/item_detail_container"
                android:layout_width="0dp"
                android:layout_height="match_parent"
                android:visibility="gone" />
</LinearLayout>
```

res\layout-land\main_layout.xml

(Deux fragments)

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
... blabla ...>
        <FrameLayout android:id="@+id/item_list_container"
                android:layout_width="0dp"
                android:layout_height="match_parent"/>

        <FrameLayout android:id="@+id/item_detail_container"
                android:layout_width="0dp"
                android:layout_height="match_parent"
/>
</LinearLayout>
```

Si vous ne faîtes pas ça, vous ne restaurerez pas le second fragment dans le cas de la recréation/destruction de votre activité deux fois de suite :

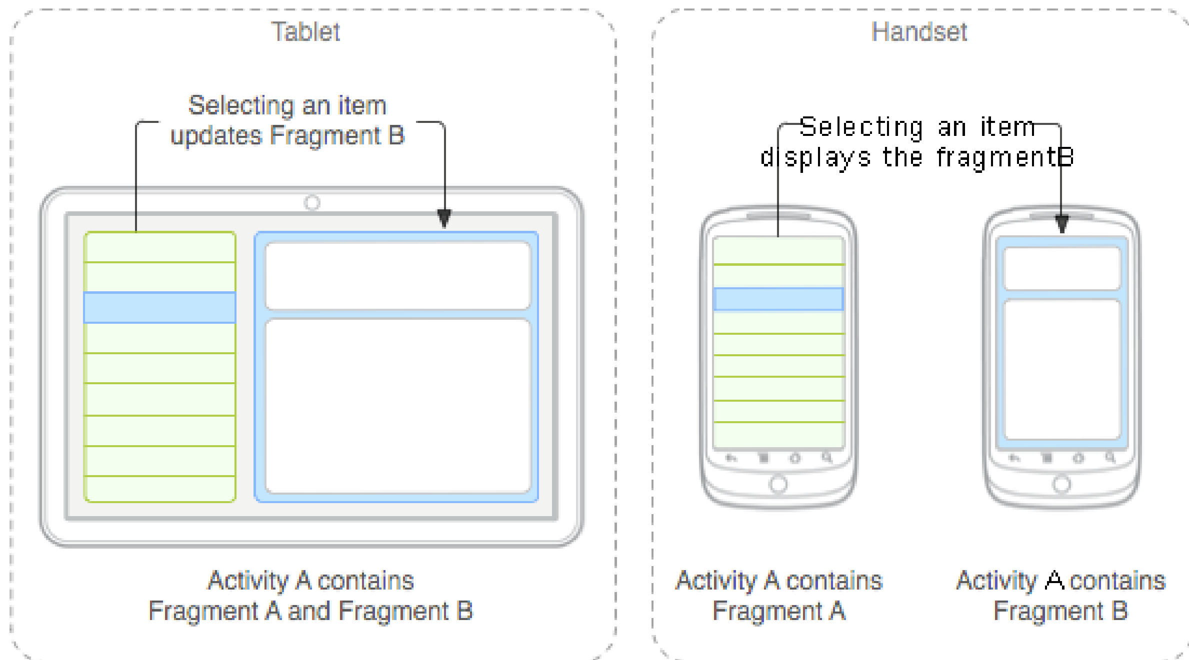
@ copyright2012@android2ee

Deux rotations de l'appareil : land -> port -> land (2° fragment non restauré)

Notez que cette bonne pratique s'applique quasiment uniquement au cas landscape/portrait. En effet il est rare de changer la taille ou la densité d'un appareil au run-time.

5.4 Structurer une application possédant des fragments

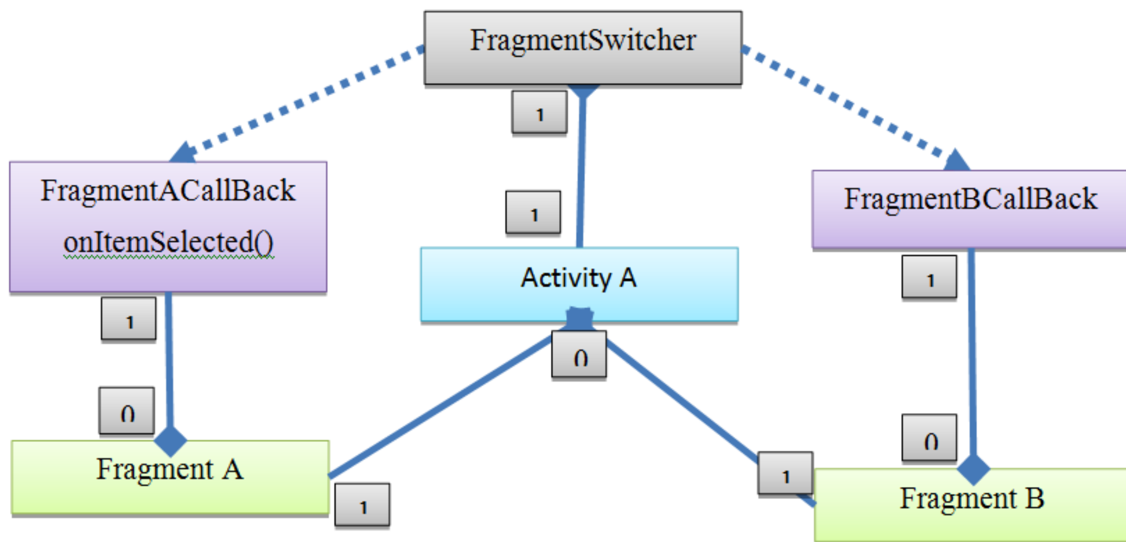
Revenons concrètement à l'exemple de Google sur les fragments :



Il y a l'activité A qui possède un ou deux fragments en fonction de son contexte d'exécution. Le fragment A affiche une liste, le fragment B affiche l'item de la liste sélectionné.

5.4.1 Design Pattern

En fait le principe est le suivant :



L'activité A connait les fragments qu'elle contient. Les fragments ne connaissent pas leur conteneur, mais uniquement leur CallBack. Ces callbacks ne sont plus implémentés par l'activité elle-même mais par une classe FragmentSwitcher dans laquelle on a déporté le code de gestion des fragments.

Le FragmentSwitcher a à charge l'orchestration de ses fragments, elle les écoute, les met à jour, les supprime et les créé. C'est à elle de passer l'information de l'un à l'autre ou de changer les fragments (cas dynamique).

5.4.2 La méthode onItemSelected de l'activité A

Ainsi dans l'exemple de Google, le fragment A appelle l'activité A (via le onItemSelected de l'interface FragmentACallBack). L'activité A doit alors connaitre son contexte, si elle affiche deux fragments, elle doit mettre le second fragment à jour, si elle n'affiche qu'un fragment, elle doit afficher le fragment B (et donc le créer).

En code cela donne :

```
/******************************************************************************/
/** OnItemSelection Management ***********************************************/
/******************************************************************************/
/* * (non-Javadoc) *
 * @see com.android2ee.tuto.fragment.fragment.dynamic.tuto.view.callback.MainFragmentCallBack#
 * onItemSelected(int) */
@Override
public void onItemSelected(int itemId) {
	// Ok le moment du choix
	//En fonction du nombre de fragments affichés, soit detailFragment doit être mis à jour, soit il
	faut changer le fragment affiché dans le premier layout par detailFragment
	if (twoPanes) {
		onItemSelectedTwoPanes(itemId);
	} else {
		onItemSelectedOnePane(itemId);
```


@ copyright2012@android2ee

```
		}
	}
	/**
	 * OnPane Management
	 * Gestion du cas où il n'y a qu'un layout
	 * @param itemIdthe item to display
	 */
	public void onItemSelectedOnePane(int itemId) {
		// Ne conservez jamais une référence vers un fragment en vue de le ré-instancier
		// (DetailFragmentHC) fm.findFragmentByTag (detailFragmentFPTag) devrait être nul
		// Nous avons besoin de le créer puis de l'ajouter au layout
		// En effet Soit il n'a pas été créé (première création)
		// Soit il a été détruit (créé puis backstack.popup appelé)
		// Dans tous les cas, il n'existe plus

		// Trouver le gestionnaire de fragment
		FragmentManager fm = activity.getFragmentManager();
		DetailFragmentHC detailFragmentFirstPane = (DetailFragmentHC)
		fm.findFragmentByTag(detailFragmentFPTag);
		if (null == detailFragmentFirstPane) {
			//Comportement normal
			detailFragmentFirstPane = new DetailFragmentHC();
			//Ajoutez la position de l'item en argument de création du fragment
			//Utilisez le pattern de l'argument
			Bundle bundle = new Bundle();
			bundle.putInt(DetailFragmentHC.ITEM_ID, itemId);
			detailFragmentFirstPane.setArguments(bundle);
			//Remplacez mainFragment par detailFragment
			FragmentTransaction fTransaction = fm.beginTransaction();
			// Définissez les animations
			fTransaction.setCustomAnimations(R.anim.anim_push_left_in,
				R.anim.anim_push_left_out,
				R.anim.anim_push_right_in,
				R.anim.anim_push_right_out);
			// Remplacez, ajoutez à la backstack et commit
			fTransaction.replace(R.id.firstpane, detailFragmentFirstPane, detailFragmentFPTag);
			fTransaction.addToBackStack(activity.getString(R.string.main_fragment));
			fTransaction.commit();
		}
	}
	/**
	 * TwoPane Management
	* Gestion du cas où il y a deux layouts
	 * @param itemIdthe item to display
	 */
	public void onItemSelectedTwoPanes(int itemId) {
		// Trouvez le FragmentManager
		FragmentManager fm = activity.getFragmentManager();
		// Le fragment existe et a déjà été ajouté au layout
		// Ainsi, retrouvez le et mettez le juste à jour
		DetailFragmentHC detailFragmentSecondPane = (DetailFragmentHC)
		fm.findFragmentByTag(detailFragmentSPTag);
		// Mise à jour donc :
		detailFragmentSecondPane.updateData(itemId);
	}
```

5.4.3 La classe FragmentsSwitcher

Pour une activité qui met en place des fragments dynamiquement, il est une bonne pratique qui consiste à extraire de l'activité tout le code correspondant à la gestion des fragments dans une classe à part dédiée à cette problématique. Cette classe n'est autre que le FragmentSwitcher (vous pouvez l'appeler comme bon vous semble).

Ainsi, vous centraliser dans le FragmentSwitcher toute votre gestion des fragments. La classe FragmentSwitcher implémente tous les callback des fragments qu'elle gère.

Vous pouvez avoir plusieurs FragmentSwitchers pour une même activité. Par exemple, FragmentSwitcherHuman pour gérer les fragments affichant les humains, FragmentSwitcherDroid pour les fragments affichant les droids. Cela permet à la navigation de changer de contexte de fragments plutôt que de lancer une nouvelle activité pour afficher les droids.

Enfin, cela permet à l'activité de se concentrer sur ce dont elle est responsable : cycle de vie, écoute des Intents …

La liaison Activity-FragmentSwitcher est de type 1 - [0 ,1]. L'activité possède un ou 0 FragmentSwitcher, le FragmentSwitcher possède toujours une activité.

Les fragments ne connaissent plus l'activité mais uniquement le FragmentSwitcher.

5.4.4 Fragments et BackStack

Il est naturel que certains fragments soient considérés comme des "activités" d'un point de vue utilisateur et que l'appuie sur le bouton back réaffiche le fragment précédemment affiché. En effet pour l'utilisateur lancer une nouvelle activité ou afficher un nouveau fragment est pour lui transparent. Il doit donc pouvoir dépiler sa navigation au sein des fragments comme il le faisait avec les activités.

Pour mettre en place ce mécanisme, il suffit d'ajouter le fragment à la backStack.

```
// Ajoutez le fragment
fTransaction.replace(R.id.mainfragment, eventsListFragment, eventsListFragmentTag);
// Ajoutez la transaction à la backstack
fTransaction.addToBackStack(mainActivity.getString(R.string.main_htitle));
// Faîtes le commit
fTransaction.commit();
```

En fait, ce n'est pas le fragment qui est ajouté à la backStack, mais la transaction. Quand la backstack sera dépilée, la transaction sera inversée. Plus subtilement, cela signifie aussi que le fragment est conservé dans la backstack et dépiler la backstack le restaurera. Cela signifie au niveau du cycle de vie du fragment un petit changement.

Quand on supprime ou l'on remplace un fragment et qu'on le dépose dans la backstack :

- Les méthodes **onPause, onStop et onDestroyView** sont appelées sur le fragment quand il est placé dans la backStack.
- Et les méthodes **onCreateView, onActivityCreated, onStart et onResume** quand il en est dépilé.

De placer un fragment dans la backstack impacte le cycle de vie du fragment.

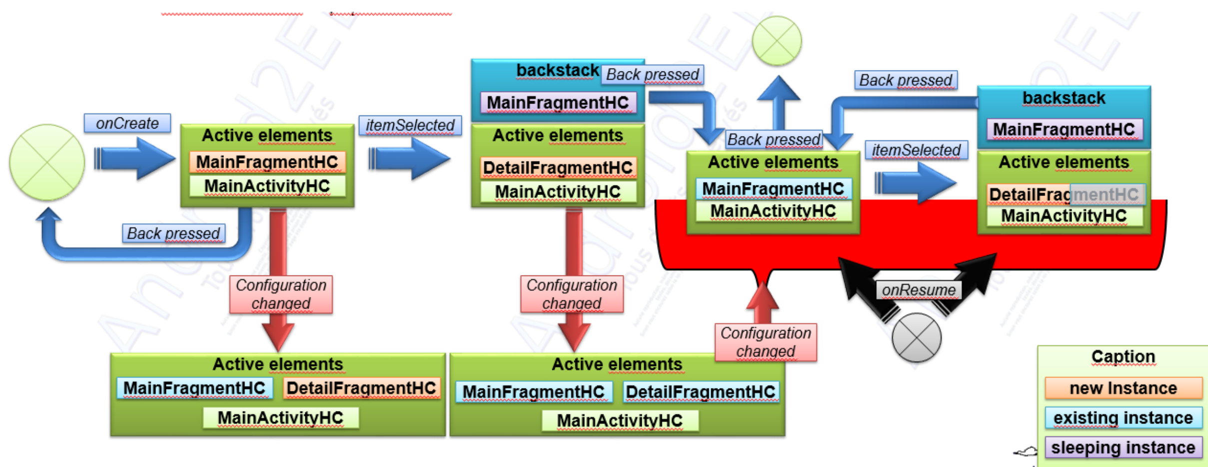
5.4.4.1 Backstack : ne la supposez pas

Quand on manipule les fragments dynamiquement, il est toujours nécessaire de savoir où on en est; quel est l'état de la BackStack, quels sont les fragments instanciés, détruits.

En effet quand on affiche ou réaffiche un fragment, le choix de la méthode show, add ou replace est primordial, de même que l'appel aux méthodes addToBackstack et popBackstack.

@ copyright2012@android2ee

Pourtant, sur un simple exemple, s'il l'on considère le diagramme d'état de la backstack, on voit qu'il nous est impossible de prévoir l'état de la backstack :



Ainsi, au lieu d'imaginer que l'on sait quel est l'état de la backstack, il nous faut la requêter et en fonction du résultat faire un popBackStack, un ajout, un replace ou un show du fragment. C'est la bonne pratique « Toujours interroger la backstack avant d'instancier un fragment » expliquée plus bas dans l'article.

5.4.5 Nombre de fragments affichés et structuration des layouts.

De la même manière que pour la gestion statique des fragments, vous allez mettre en place le système des redirections de vos layouts avec les fichiers du type :

```
<resources>
<itemname="main_activity"type="layout">@layout/activity_two_panes</item>
<boolname="twoPane">true</bool>
</resources>
```

L'objectif est de toujours connaître le nombre de fragments affichés à l'écran et de centraliser vos fichiers de layout dans res\layout.

5.5 Navigation

Il y a quatre manières de mettre en place une navigation dans une application Android.

Les six packs	Navigation par onglets	Navigation avec un spinner	Navigation avec le Navigation Drawer

Ce qu'il vous faut comprendre, d'un point de vue développement, c'est que c'est de la cosmétique pour les utilisateurs finaux. Pour vous la navigation se résume à mettre un composant graphique en place avec un listener permettant à l'utilisateur de faire un choix.

En fonction de ce choix, vous pouvez :

- Filtrer vos données ;
- Changer de fragment ;
- Changer de contexte de fragments ;
- Lancer une nouvelle activité ;
- Effectuez n'importe quelle action qui affiche de nouvelles données à l'utilisateur.

Si vous voulez en savoir plus sur le sujet, vous pouvez venir aux formations Android2EE, je vous explique beaucoup, beaucoup plus de choses concernant la mise en place d'applications Android, les bonnes pratiques à respecter et tout ce qu'il faut savoir pour réussir le développement d'une application Android. En particulier la mise en place de la navigation via l'ActionBar (l'ActionBarCompat pour être exact).

5.6 Animations

Les animations sont importantes pour permettre à l'utilisateur de comprendre la dynamique de votre application. L'esprit humain comprend beaucoup mieux les changements de contexte avec une animation de transition que sans. C'est l'objectif principal des animations.

Les animations ajoutent aussi une touche de « beauté » à votre application, ce qui améliore l'expérience utilisateur.

Je ne rentrerai pas dans les détails, si vous souhaitez un approfondissement, comme je viens de vous le dire, venez à mes formations je vous l'expliquerai plus longuement.

Concernant les animations, les points cruciaux sont :

MinSDK=11 ?o? Utiliser le parrallel pattern.

Deux types d'animation incompatibles : Avant (TweenAnimation) Après (ObjectAnimator, ViewProperty) HoneyComb

Soyez cours.

Le temps d'animation par défaut est de 300 ms.

Cela dépend de l'objet à animer (une vue, une activité ce n'est pas un bouton).

Soyez époustouflant… mais une fois.

Les animations longues, belles et délirantes deviennent ennuyeuses au bout de 3.

Remplacer les par des animations efficaces au bout de trois.

Si en développement votre animation ne vous pourrit pas la vie alors elle ne pourrira pas celle de vos utilisateurs.

Thanks Chet.

Depuis HoneyComb, les animations sont améliorées, simplifiées et super flexibles.

Dev.Bytes nous poste des tutoriaux de 3 minutes pour nous les apprendre.

Merci à Chet pour tout ça.

@ copyright2012@android2ee

5.6.1 Mise en place des animations

Pour mettre en place les animations lors des transitions entre les fragments, vous êtes obligés d'utiliser le parrallel activity pattern étendu.

Il vous faut vérifier quelle est la version de l'appareil sur laquelle s'exécute votre application. Si cette version est inférieure à HoneyComb vous devez utiliser les TweenAnimations si il est supérieur à HoneyComb vous mettrez en place les ObjectAnimators.

Vous allez donc avoir deux dossiers anim et anim-v11 possédant les mêmes fichiers (par exemple monAnim.xml) mais dans anim, vous utiliserez les TweenAnimation et dans anim-v11 vous utiliserez les ObjectAnimator.

Ainsi vous aurez les fichiers suivants :

res\anim\anim_push_left.xml

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

<translate
        android:duration="500"
        android:fromXDelta="100%p"
        android:toXDelta="0"/>

<alpha
        android:duration="500"
        android:fromAlpha="0.0"
        android:toAlpha="1.0"/>

<scale
        android:duration="500"
        android:fromXScale="0.0"
        android:fromYScale="0.0"
        android:pivotX="50%"
        android:pivotY="50%"
        android:toXScale="1.0"
        android:toYScale="1.0"/>

</set>
```

res\anim-v11\anim_push_left.xml

```
<?xmlversion="1.0"encoding="utf-8"?>
<setxmlns:android="http://schemas.android.com/apk/res/android">

<objectAnimator
        android:duration="1000"
        android:propertyName="translationX"
        android:valueFrom="-250"
        android:valueTo="0"
        android:valueType="floatType"/>

<objectAnimator
        android:duration="1000"
        android:propertyName="scaleX"
        android:valueFrom="0.0"
        android:valueTo="1.0"
        android:valueType="floatType"/>
```

```
<objectAnimator
        android:duration="1000"
        android:propertyName="scaleY"
        android:valueFrom="0.0"
        android:valueTo="1.0"
        android:valueType="floatType"/>

<objectAnimator
        android:duration="1000"
        android:propertyName="rotationY"
        android:valueFrom="0"
        android:valueTo="360"
        android:valueType="floatType"/>

</set>
```

5.7 Bonnes pratiques

5.7.1 Toujours interroger la backstack avant d'instancier un fragment avec findByTag

```
//La méthode findByTag (resp. findById) trouve un fragment qui a été identifié par la balise
//donnée (lorsqu'il est gonflé à partir de XML ou fourni lorsqu'il est ajouté à une transaction).
//La recherche s'effectue d'abord dans les fragments qui sont actuellement gérer par l'activité; si
//aucun fragment ne s'y se trouve, alors la recherche continue parmis tous les fragments
//actuellement dans la backstack.
//Ainsi, s'il n'est pas trouvé, il faut le créer.
MainFragmentHC mainFragment;
mainFragment = (MainFragmentHC) fm.findFragmentByTag(mainFragmentTag);
if (null == mainFragment) {
        // Créez le fragment
        mainFragment = new MainFragmentHC();
        // Ajoutez le ou remplacer un fragment existant par celui-ci.
        fTransaction.add(R.id.firstpane, mainFragment, mainFragmentTag);
} else {
        // MainFragment appartient soit à la vue actuelle, soit à la backStack
        // Si elle appartient à la vue actuelle: ne rien faire est la chose à faire
        // Si elle appartient à la backStack: faire un pop de la backstack pour l'afficher est la
        //chose à faire.
        // Donc, on fait un pop de la backstack si Pop_BackStackInclusive vous faîtes un pop de
        la transaction avec le tag inclus (elle n'est donc plus là) sinon vous faîtes avec 0.
        fm.popBackStackImmediate(activity.getResources().getString(R.string.main_fragment),Fragm
        entManager.POP_BACK_STACK_INCLUSIVE);
        // Maintenant nous sommes sûrs que s'il était dans la backstack, il a été ramené au
        // devant de la scène
        // Donc l'afficher est suffisant
        fTransaction.show(mainFragment);
}
```

5.7.2 Autres bonnes pratiques

5.7.2.1 Le fragment est lié au layout qui le contient

Lorsque vous placez dynamiquement un fragment dans un layout, vous ne pourrez plus le placer dans un autre layout. Concrètement, vous avez ajouté le fragment myFragment avec le tag myFrag au layout dont l'identifiant est layout_panel1. Si vous souhaitez ajouter myFragment (la même

@ copyright2012@android2ee

instance) avec le tag myFrag au sein d'un autre layout (par exemple celui avec l'identifiant layout_panel2), cela génèrera une exception au RunTime. Vous serez obligé de créer une nouvelle instance de votre fragment et de l'ajouter au layout layout_panel2 en utilisant un autre tag, par exemple myFrag2.

5.7.2.2 Ne jamais ré-instancier un fragment avec une référence obsolète

C'est une erreur grave que de garder un pointeur vers un fragment et d'essayer de le réutiliser au lieu d'instancier de nouveau le fragment.

Vous pouvez garder un pointeur vers un fragment, mais assurez-vous que celui-ci suit bien le cycle de vie du fragment, quand le fragment est détruit votre pointeur doit être null.

5.7.2.3 Toujours utiliser les tags des fragments pour les identifier et les retrouver

Je n'ai rien à ajouter au titre.

6 Autres fragments

Je ne vous présenterai que succinctement DialogFragment et PreferenceFragment.

6.1 DialogFragment

Vous savez mettre en place des fenêtres de dialogue et bien vous savez aussi mettre en place des DialogFragment car c'est identique.

Pour afficher une fenêtre de dialogue vous faîtes :

```
new MyDialogFragment().show(getSupportFragmentManager(),MY_DIALOG_FRAGMENT_TAG);
```

Et pour définir la fenêtre de dialogue, il vous suffit de créer une classe qui étend DialogFragment et de surcherger la méthode onCreateDialog. Dans la méthode onCreateDialog vous faîtes exactement comme vous faisiez avant.

```
public classMyDialogFragment extends DialogFragment {
	@Override
	public Dialog onCreateDialog(Bundle savedInstanceState) {
		// Création de l'AlertDialog Builder
		AlertDialog.Builder builder = new AlertDialog.Builder(getActivity());
		// Titre et message
		builder.setMessage(R.string.dialog_google_att_message);
		builder.setTitle(R.string.dialog_google_att_title);
		// Pas de bouton cancel
		builder.setCancelable(false);
		// Définition du bouton OK
		builder.setPositiveButton(getString(R.string.dialog_google_att_btn), null);
		// Création et renvoie de l'AlertDialog
		return builder.create();
	}
}
```

6.2 PreferenceFragment

Comme pour les DialogFragments, si vous saviez le faire avant (avec les PreferenceActivity), vous savez le faire avec les fragments. La seule différence est que vous allez créer une classe MyPrefsFragment qui hérite de PreferenceFragment et vous surchargerez sa méthode onCreate ainsi :

```
public static class PrefsFragment extends PreferenceFragment {
```

@ copyright2012@android2ee

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

		//Charger les préférences à partir du XML
		addPreferencesFromResource(R.xml.preferences);
	}
}
```

Où R.xml.preferences pointe vers votre fichier xml de préférences.

7 Conclusion

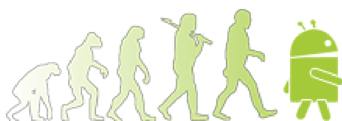
A l'heure actuelle, vous ne pouvez plus effectuer de développement Android sans utiliser les fragments. J'espère que cette article vous a plu et vous a permis de comprendre le fonctionnement des fragments et leur mise en place.

J'espère aussi vous retrouver en formation pour pouvoir vous expliquer durant 5 jours quels sont les bonnes pratiques de développement Android, les architectures à mettre en place, la gestion des ressources et bien plus encore. En formation, c'est un classeur de 500 pages que je vous explique et avec lequel vous repartez, ainsi que plus de 70 tutoriaux que vous pourrez utiliser librement.

Alors à vos Fragments et ActionBars and may the force be with you !

8 Contact Android2EE

Mathias Séguy
mathias.seguy.it@gmail.com
Fondateur Android2EE
Architecte – Formateur – Consultant Android
Auteur Android2ee.com
Docteur en Mathématiques Fondamentales
Expert Technique de l'Agence Nationale de la Recherche
Rédacteur sur Developpez.com



Android 2EE la programmation sous Android.

9 Android2ee spécialiste de la formation Android

Android2EE vous propose des formations Android adaptées à vos besoins.

- Formation Initiale : Devenir autonome (3j).
- Formation Complète (5j).
- **Mais aussi la possibilité d'effectuer des formations d'approfondissement sur mesure (2-3j).**

Calendrier.

Tarifs et considérations contractuelles.

@ copyright2012@android2ee

Comme tant d'autres stagiaires, d'autres entreprises, avant vous, offrez-vous une formation Android d'excellence ; un support de cours de 500 pages, un formateur expert et passioné par Android, (speaker lors de grands rassemblement I.T.), plus de 50 tutoriaux en utilisation libre, une pédagogie éprouvée.

N'hésitez pas, mon seul objectif sur ces 3 dernières années, c'est de vous offrir la meilleure formation Android et je pense l'avoir attend.

Pour me contacter :

contact@android2ee.com

10 Retrouvez les tutoriaux de cet article sur le site Android2EE

Pour télécharger les tutoriaux associés à cet article, il vous suffit de vous rendre sur la page suivante (ils se trouvent en bas de page) : Téléchargement des tutoriaux.

11 Le site Android2ee, une référence Android.

Le site Android2EE vous propose des tutoriaux, des articles, des vidéos, des conférences, des EBooks en libre consultation pour monter en compétence sur la technologie Android.

Vous trouverez tout cela dans la partie « Open Resources ».

N'hésitez, plus visitez le !

12 Android2ee vous présente l'Ebook de programmation Android

L'objectif de ces livres est très clair : vous permettre en un temps record d'être autonome en programmation Android. Si vous êtes un programmeur Java (débutant ou confirmé), le but est que vous soyez autonome en moins de dix jours. C'est cet objectif qui est à l'origine de ce livre, permettre aux collaborateurs de mon entreprise de monter en compétence sur cette technologie avec rapidité et efficience. Vous serez alors à même de concevoir une application, de l'implémenter, de la tester, de l'internationaliser et de la livrer à votre client.

Lancez-vous dans la programmation Android et faites-vous plaisir !

Vous serez aussi capable de connaître et comprendre quelles sont les considérations à avoir lorsque l'on a à charge une application Android en tant que professionnel de l'informatique. Quelle est la stratégie de tests à utiliser ? Comment signer son application ? Comment la déployer ? Comment mettre en place la gestion du cycle de vie de l'application ? Comment implémenter l'intégration continue ?

Soyez efficient dans l'encadrement de vos projets Android d'entreprise.

13 Remerciements

J'adresse ici tous mes remerciements à Feanorin pour son implication, son aide et sa sympathie et à jacques_jean pour l'excellence de ses corrections orthographiques.

Je remercie les correcteurs techniques Feanorin et MrDuChnok pour la pertinence de leurs remarques.

Je remercie spécialement Monsieur Adam Daniel, DRH de développez.com, pour ses mails nocturnes, ses encouragements et son aide qui m'a été précieuse.

@ copyright2012@android2ee