Android TD 5
Fragments dynamiques

Les fragments peuvent être manipulés (ajoutés/supprimés) dynamiquement lors de l'exécution de l'application et cette gestion est effectuée dans le code Java. Ce TD se découpe en plusieurs phases :

1. ajout d'un fragment dynamiquement lors de la création de l'application,
2. ajout d'une transition lors du chargement du fragment,
3. empilement/dépilement de fragments,
4. ajout de transitions entre passages de fragments.

1 Ajout dynamique d'un fragment

Des fragments peuvent être ajoutés à un layout du type `FrameLayout` qui par défaut ne contient pas de fragment (`main.xml`) :

```
<?xml version="1.0" encoding="utf-8"?>
<FrameLayout xmlns:android="http://schemas.android.com/apk/res/android"
             android:id="@+id/framelayout"
             android:layout_width="match_parent"
             android:layout_height="match_parent"
             >
</FrameLayout>
```

On souhaite ajouter le fragment suivant (`fragment1.xml`) dynamiquement au début du programme :

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    >
<TextView
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:text="Fragment1"
    />
</LinearLayout>
```

La classe du fragment (`Fragment1.java`) est définie de la manière suivante :

```
public class Fragment1 extends Fragment
{
    @Override
    public View onCreateView(LayoutInflater inflater,
                             ViewGroup container,
                             Bundle savedInstanceState)
    {
        return inflater.inflate(R.layout.fragment1, container, false);
    }
}
```

Il suffit ensuite de procéder explicitement au chargement du fragment

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

```
        fragment1 = new Fragment1();
        getFragmentManager().beginTransaction()
            .add(R.id.framelayout, fragment1).commit();
    }
```

Les transactions disponibles sont `add`, `remove` et `replace` respectivement pour ajouter, supprimer ou remplacer des fragments dans un `FrameLayout`.

2 Effets de transition

2.1 Effet fondu

On commence par ajouter un effet de fondu lors du chargement du fragment. Il suffit pour cela de modifier le code dans la classe `Fragment1` pour ajouter les étapes suivantes :

1. définir l'opacité du fragment à 0 à l'aide de la méthode `setAlpha( float )` de manière à ce que le contenu ne soit pas visible au chargement : le fragment est visible mais transparent !
2. récupérer les propriétés d'animation du fragment avec la méthode `animate()`,
3. définir la valeur finale d'opacité après la transition,
4. définir la durée de la transition.

Cela se traduit par l'ajout du code suivant dans la classe du fragment :

```
    @Override
    public View onCreateView(LayoutInflater inflater,
                             ViewGroup container,
                             Bundle savedInstanceState)
    {
        View view = inflater.inflate(R.layout.fragment1, container, false);

        view.setAlpha(0.0f);
        view.animate().alpha(1.0f).setDuration(1000);

        return view;
    }
```

Normalement, le fragment devrait s'afficher progressivement au lancement de l'application. L'effet est plus visible si on ajoute une couleur de fond pour le layout du fragment à l'aide de l'attribut `android:background="#ff0"`.

2.2 Effet rotation

De la même manière on peut faire tourner un fragment en utilisant le code suivant :

```
view.setRotation(180);
view.animate().rotation(0).setDuration(10000);
```

3 Empilement de Fragments

Les fragments peuvent être empilés de manière à pouvoir revenir au fragment précédent, par exemple lorsqu'un fragment contient une `ListView` et qu'un clic entraîne l'ouverture d'un nouveau fragment, on souhaite revenir au fragment précédent par la touche de retour.
Cette fonctionnalité est effectuée par la méthode `addToBackStack(fragment)` appliquée à la transaction.

1. Créer un deuxième fragment (copie du premier)
2. Construire ce fragment au lancement de l'application :

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

    fragment1 = new Fragment1();
    fragment2 = new Fragment2();
    getFragmentManager().beginTransaction()
      .add( R.id.fragmentcontainer, fragment1 ).commit();
}
```

3. Ajouter un bouton au premier fragment et une méthode `onClick` contenant le code suivant :

```
public void onClick(View view) {
  getFragmentManager().beginTransaction()
    .replace( R.id.fragmentcontainer, fragment2 )
    .addToBackStack( null )
    .commit();
}
```

4 Effets personnalisés de transition entre Fragments

Cette exercice est une version simplifiée de l'exemple suivant : `http://developer.android.com/training/animation/cardflip.html`. Télécharger l'archive de cet exemple dont nous reprendrons certains fichiers.
Le principe est de définir les transitions dans des fichiers XML placés dans le dossier `res/animator` de votre projet. Le fichier `card_flip_right_out.xml` de l'exemple contient le code suivant :

```
<set xmlns:android="http://schemas.android.com/apk/res/android">

    <objectAnimator
        android:valueFrom="0"
        android:valueTo="-180"
        android:propertyName="rotationY"
        android:interpolator="@android:interpolator/accelerate_decelerate"
        android:duration="300" />

    <objectAnimator
        android:valueFrom="1.0"
        android:valueTo="0.0"
        android:propertyName="alpha"
        android:startOffset="150"
        android:duration="1" />
</set>
```

Ce code contient les effets de transition à appliquer sur le fragment qui va disparaitre pendant la transition. La transition va durer 300ms, pendant ce temps, le fragment sortant va effectuer une rotation de $-180°$ sur l'axe Y, la valeur de l'angle va passer de 0 à $-180°$ en étant interpolée suivant un interpolateur par défaut. A la moitié de la transition, 150ms, le fragment va devenir transparent.
Les 3 autres fichiers décrivent de la même manière la transition du fragment qui va apparaître, et les 2 transitions inverses lorsqu'on reviendra au fragment précédent.
Afin d'ajouter ces effets, il suffit d'utiliser la méthode `setCustomAnimations` avec les transitions correspondantes comme paramètres :

```
FragmentTransaction frtr = getFragmentManager().beginTransaction();
frtr.setCustomAnimations(
```

```
R.animator.card_flip_right_in, R.animator.card_flip_right_out,
R.animator.card_flip_left_in, R.animator.card_flip_left_out);
frtr.replace( R.id.fragmentcontainer, currentfragment );
frtr.addToBackStack(null);
frtr.commit();
```