Polycopié de cours pour le module Applications mobiles

Chapitre II

Les fragments

Un fragment représente un comportement ou une partie de l'interface utilisateur dans une activité. Plusieurs fragments peuvent être groupés dans une seule activité pour créer une interface utilisateur à plusieurs volets de vues et un seul fragment peut aussi être réutilisé dans plusieurs activités. On peut considérer un fragment comme étant un module faisant partie d'une activité, possédant un cycle de vie propre, recevant en entré des événements propres et pouvant aussi être toujours ajouté ou supprimé pendant l'exécution.

II.1 Création de fragments

Pour créer un fragment, on crée une sous-classe de Fragment (faire un extends de la classe Fragment). La classe `Fragment` a un code qui ressemble beaucoup à une activité. Il contient des méthodes callback similaires comme : `onCreate()`, `onStart()`, `onPause()` et `onStop()` .

◦ exemple

La sous-classe **"My_Fragmt"** dérivée de **Fragment** ainsi définie dans le code suivant fait charger le layout déterminé par le fichier my_sample_fragmt.xml:.

```
public static class My_Fragmt extends Fragment {
    @Override
    public View onCreateView(LayoutInflater inflater,
              ViewGroup container, Bundle savedInstanceState) {

        return inflater.inflate(R.layout.my_sample_fragmt,
               container, false);
     }
}
```

◦ Ajout et remplacement de fragments

On peut ajouter un fragment à l'activité pendant l'exécution. Cela est nécessaire si on envisage de changer (remplacer) des fragments, Au cours de la vie de l'activité. Pour effectuer des transactions (ajout, remplacement ou suppression d'un fragment), on a à utiliser un **FragmentManager** pour créer une **FragmentTransaction** qui fournit des API permettant d'ajouter, de supprimer, de remplacer et d'exécuter d'autres transactions de fragment.

```
FragmentManager frgmt_mngr = getSupportFragmentManager() ;
FragmentTransaction frgmt_trsctn = frgmt_mngr.beginTransaction();

Another_Fragment frgmt_2 = new Another_Fragment();
frgmt_trsctn.add(R.id.fragment_container_id, frgmt_2);
frgmt_trsctn.commit() ;
```

II.2 Implémentation du swip

Le swip permet de faire de la pagination par glissement. Son implémentation nécessite l'utilisation des fragments, fait intervenir le widget **ViewPager** et la classe **FragmentStatePagerAdapter** comme adaptateur.

Les trois parties de code qui suivent illustrent une alternative pour l'implémentation du swip. La première partie montre les appels nécessaires à insérer au niveau de l'activité, la deuxième décrit une façon de peupler la classe **FragmentStatePagerAdapter** et la dernière partie montre une allure du code au niveau du fragment.

```
public class Actvty_with_Swp extends Activity {
Swip___adpter My__adptr_swp;
ViewPager Vw_pagr;

    public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_layt);
    My__adptr_swp = new Swip___adpter(getSupportFragmentManager());
    Vw_pagr = (ViewPager)findViewById(R.id.pager_id);
    Vw_pagr.setAdapter(My__adptr_swp);
    }
}
```

```
public class Page_Fragment extends Fragment {
    public static final String ARGMTS_OBJECT = "object";

    @Override
    public View onCreateView(LayoutInflater inflater,
              ViewGroupcontainer, Bundle savedInstanceState) {
        View Layt_vw = inflater.inflate(R.layout.page_fragmt_lyt,
                                    container, false);
        Bundle argmts = getArguments();
        ((TextView)Layt_vw.findViewById(R.id.text_id)).
        setText(Integer.toString(argmts.getInt(ARGMTS_OBJECT)));
        return Layt_vw;
    }
}
```

```
public class Swip__adpter extends FragmentStatePagerAdapter {
    public Swip__adpter(FragmentManager fm) {
        super(fm);
     }

    @Override
    public Fragment getItem(int page_pos) {
        Page_Fragment my_page_fragmt = new Page_Fragment();
        Bundle argmts = new Bundle();
        argmts.putInt(Page_Fragment.ARGMTS_OBJECT, page_pos);
        my_page_fragmt.setArguments(argmts);
    }

    @Override
    public int getCount() {
        return 100;
    }

    @Override
    public CharSequence getPageTitle(int position) {
        return "OBJECT " + Integer.toString((position + 1));
    }
}
```

II.3 Partager un listener avec l'activity

Un listener est rendu "partagé" en définissant une interface (comme "callback") à l'intérieur du fragment pour que l'activité hôte l'implémente avec la clause **implements** .
On se permet ainsi de faire manifester nos handlers non pas au niveau du fragment, mais au niveau de l'activitié ou au niveau d'un autre fragment.
On aura besoin d'avoir ce fait dans la cas où l'on désire par exemple montrer les détails d'un article ou un élément d'une liste figurant dans un premier fragment pour afficher ces détails dans un deuxième fragment à part.

le code ci_dessous illustre la logique du code à implémenter.

```
public class Articles__Fragment extends Fragment {
    Interfc__Callbck the___callbck;

    public void  set__listenner(Activity the__actvty) {
        the___callbck = the__actvty;
    }

     ...

    //Container Activity must implement this interface
    @Override
    public Interface Interfc__Callbck {
        public void  onItem_selcted(int position);
     }

     ...

}
```

Pour que l'activité hôte implémente cette interface, cette activité est transmise comme argument à travers la méthode de rappel (le callback) **onAttachFragment().** (Le système appelle ce callback lors de l'ajout et l'attachement du fragment à l'activité). l'activité est transmise pour subir un cast dans la méthode **set_listenner().**

```
public class Host_actvty extends Activity implements
          Articles_Fragment.Interfc_Callbck   {

     ...

    @Override
    public void onAttachFragment(Fragment the_fragm) {

         if(the_fragm instanceof Articles_Fragment) {
            Articles_Fragment my_fragmt =
                 (Articles_Fragment)the_fragm;
            my_fragmt.set_listenner(this);
        }

    }

     ...

}
```

Désormais, l'action peut être manifestée au niveau de l'activité en appelant la méthode **onItem_selcted()** (ou d'autres méthodes) de l'interface **Interfc_Callbck** en réponse à un événement du fragment. s'il s'agit d'un clique sur une Listview le handler peut être transmis à l'activité comme indiqué par le code suivant :

```
    @Override
    public void onListitemClick(ListView L, View v,
                                    int position, long id) {

         //Send the event to the host activity

          the__callbck.onItem_selcted(position);
     }
```

Le traitement à réaliser en réponse à l'action est défini dans l'activité.

```
public class Host__actvty extends Activity implements
          Articles__Fragment.Interfc__Callbck   {

     ...

    public void onItem_selcted(position) {

        //Do something here as an event handler

    }

     ...

}
```