Chapitre 3. Composantes de base d’une application mobile

Table des matières

3. Chapitre 3. Composantes de base d'une application mobile 3

3.1. Introduction 3

3.2. Les fragments 3

3.2.1. Caractéristiques des fragments 3

3.2.2. Importance des fragments 4

3.3. Utilisation 4

3.3.1. Définir un fragment 4

3.3.2. Intégrer un fragment dans une activité 5

3.3.3. Intégrer un fragment statiquement 6

3.3.4. Intégrer un fragment dynamiquement 6

3.4. Le composant ListView 7

3.4.1. Présentation de ListView 7

3.4.2. L'adaptateur de listview 8

3.4.3. Les items de listView 9

3.4.4. Les actions sur les listView 9

3.5. Conclusion 9

Bibliographie et Webographie 11

3. Chapitre 3. Composantes de base d’une application mobile

3.1. Introduction

Un fragment est un élément d'une activité qui permet une conception d'activité plus modulaire. un fragment est une sorte de sous-activité. Avant l'apparition des fragments, il n’été pas possible de diviser l'écran de l'appareil et contrôler les différentes parties séparément. Mais avec l'introduction du fragment, une flexibilité a été ajoutée aux activités qui peuvent contenir plusieurs fragments qui auront leur propre mise en page, événements et cycle de vie complet.

3.2. Les fragments

Un fragment est une classe réutilisable qui fonctionne sur une partie d'une activité. Il définit généralement une partie d'une interface utilisateur. Les fragments doivent être intégrés aux activités, ils ne peuvent pas fonctionner indépendamment.

3.2.1. Caractéristiques des fragments

Les principales caractéristiques des fragments sont:

- Un Fragment est une combinaison d'un fichier de mise en page XML et d'une classe Java comme le cas des activités.
- En utilisant la bibliothèque de support, les fragments sont pris en charge dans toutes les versions Android.
- Les fragments encapsulent les vues et la logique afin qu'il soit plus facile de les réutiliser dans les activités.
- Les fragments sont des composantes autonomes qui peuvent contenir des vues, des événements et une logique.

Dans une architecture orientée fragment, les activités deviennent des conteneurs de navigation qui sont principalement responsables d’accès à d'autres activités, présentant des fragments et transmettant des données.

3.2.2. Importance des fragments

Il existe de nombreux cas d'utilisation pour les fragments, mais les cas d'utilisation les plus courants sont :

- Réutilisation des composants de vue et de logique : Les fragments permettent de réutiliser des parties de l'écran, y compris les vues et la logique des événements à travers de de nombreuses activités. Par exemple, utiliser la même liste déroulante dans différentes sources de données au sein d'une application.
- Prise en charge des tablettes : Souvent dans les applications, la version tablette d'une activité a une présentation sensiblement différente de la version téléphone. Les fragments permettent aux activités spécifiques à l'appareil de réutiliser des éléments partagés tout en présentant des différences.
- Orientation de l'écran : Souvent dans les applications, la version portrait d'une activité a une disposition sensiblement différente de la version paysage. Les fragments permettent aux deux orientations de réutiliser des éléments partagés tout en présentant des différences.

3.3. Utilisation

3.3.1. Définir un fragment

Un fragment, comme une activité, possède un fichier de mise en page XML et une classe Java qui représente le contrôleur Fragment.

Le fichier de mise en page XML est comme n'importe quel autre fichier de mise en page et considéré comme une activité partielle. La figure 3.1. montre un code XML basique d'un conteneur Layout composé d'un TextView et d'un Button. Cette partie XML peut être incluse dans un fragment ou d'une simple activité.

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

    <TextView
        android:id="@+id/textView1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="TextView" />

    <Button
        android:id="@+id/button1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Button" />

</LinearLayout>
```

Figure 3.1. un exemple de code XML basique d'un Layout.

Cependant, Le contrôleur Java d'un fragment peut être écrit comme montré par la figure 3.2. et comme écrit ci-dessous :

- Déclarer la classe fragment héritant de l'interface 'Fragment'.
- Injecter le fragment dans le bloc parent récupéré par paramètres. L'injection se fait via la méthode ***'inflate'*** qui accepte comme paramètres le code XML du fragment, le parent dans lequel le fragment sera injecté, et le fait d'intégration immédiat ou non. La section qui suit détail plus l'intégration des fragments.

```java
import androidx.fragment.app.Fragment;

public class FooFragment extends Fragment {
    // The onCreateView method is called when Fragment should create its View object hierarchy,
    // either dynamically or via XML layout inflation.
    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup parent, Bundle savedInstanceState) {
        // Defines the xml file for the fragment
        return inflater.inflate(R.layout.fragment_foo, parent, false);
    }
```

Figure 3.2. déclaration d'un fragment.

3.3.2. Intégrer un fragment dans une activité

Il existe deux manières d'ajouter un fragment à une activité : dynamiquement en utilisant Java et statiquement en utilisant XML.

Avant d'intégrer un fragment dans une activité, il faut s'assurer que l'activité hérite de FragmentActivity ou AppCompatActivity pour la prise en charge du gestionnaire de fragments comme montre la figure ci-dessous.

```
import androidx.appcompat.app.AppCompatActivity;

public class MainActivity extends AppCompatActivity {
    // ...
}
```

Figure 3.3. déclaration de MainActivity pour prendre en charge les fragments.

3.3.3. Intégrer un fragment statiquement

Pour ajouter le fragment de manière statique , intégrer simplement le fragment dans le fichier de mise en page xml de l'activité. Mais il faut prendre en considération que ce n'est pas possible de remplacer un fragment défini statiquement dans le fichier de mise en page via une *FragmentTransaction*. La figure 3.4 ci-dessous montre un composant *Fragment* intégré statiquement dans un *Layout*.

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

    <fragment
        android:name="com.example.android.FooFragment"
        android:id="@+id/fooFragment"
        android:layout_width="match_parent"
        android:layout_height="match_parent" />

</LinearLayout>
```

Figure 3.4. Intégration statique d'un fragment dans une activity.

3.3.4. Intégrer un fragment dynamiquement

La deuxième méthode consiste à ajouter le fragment dynamiquement en Java à l'aide de l'extension *FragmentManager*. Cette dernière classe et la classe *FragmentTransaction* permettent d'ajouter, de supprimer et de remplacer des fragments dans la mise en page de l'activité au moment de l'exécution.

Dans ce cas, un conteneur "espace réservé" (généralement un *FrameLayout*) est ajouté à l'activité où le fragment sera inséré comme montre la figure ci-dessous.

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

  <FrameLayout
       android:id="@+id/your_placeholder"
       android:layout_width="match_parent"
       android:layout_height="match_parent">
  </FrameLayout>

</LinearLayout>
```

Figure 3.5. le conteneur FrameLayout qui sera remplacé dynamiquement par le fragment.

La classe *FragmentManager* est ensuite utilisée pour créer une *FragmentTransaction* qui permet d'ajouter des fragments au conteneur FrameLayout comme montré par la figure ci-dessous:

```java
// Begin the transaction
FragmentTransaction ft = getSupportFragmentManager().beginTransaction();
// Replace the contents of the container with the new fragment
ft.replace(R.id.your_placeholder, new FooFragment());
// or ft.add(R.id.your_placeholder, new FooFragment());
// Complete the changes added above
ft.commit();
```

Figure 3.6. création de FragmentTransaction et remplacement du FramLayout par le fragment créé.

3.4. Le composant ListView

3.4.1. Présentation de ListView

Le composant *listView* affiche une collection de vues à défilement vertical, où chaque vue est positionnée immédiatement en dessous de la vue précédente dans la liste.

Pour afficher une *listView*, il faut inclure une vue de liste dans le fichier XML de mise en page comme montré dans la figure ci-dessous :

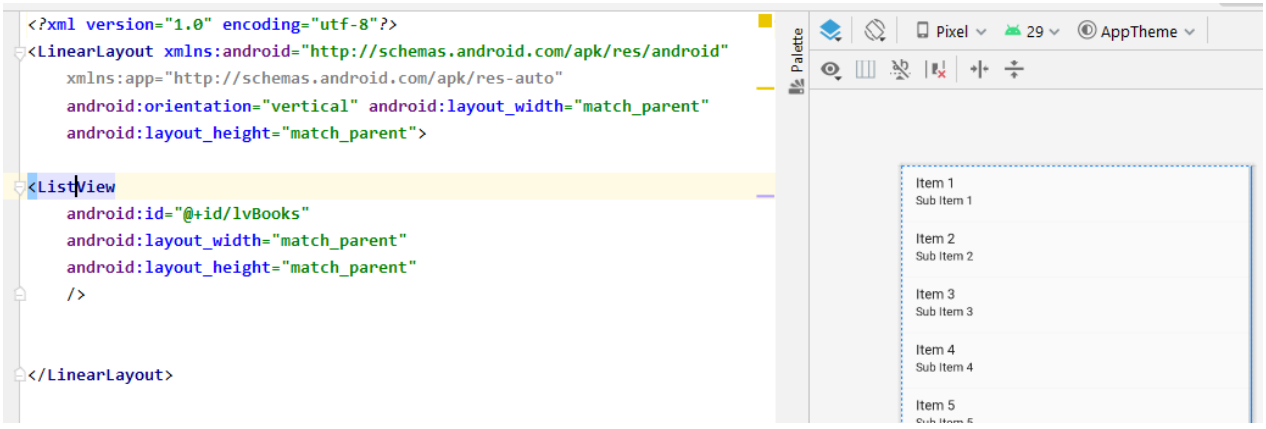

Figure 3.7. l'ajout d'une listView dans une activity.

3.4.2. L'adaptateur de listview

Une *listView* est une vue d'adaptateur qui n'a pas d'information sur les détails, tels que le type et le contenu des autres vues qu'elle contient. Cependant, la *listView* s'alimente des vues à la demande à partir d'un *ListAdapter*, par exemple pour afficher de nouvelles vues lorsque l'utilisateur fait défiler vers le haut ou vers le bas [18].

Pour afficher les éléments de la *listView*, il faut implémenter la méthode *setAdapter*(android.widget.*ListAdapter*) pour associer un adaptateur à la *listView*.

Pour afficher une vue plus personnalisée pour chaque élément de l'ensemble de données, il faut créer un *ListAdapter* personnalisé qui hérite de la classe *BaseAdapter* ou *ArrayAdapter*. L'élément *ListAdapter* créé sera personnalisé par la redéfinition de sa méthode getView qui est responsable de retourner la vue demandée par l'utilisateur. Ci-dessous la figure qui montre l'implémentation de la classe *listAdapter* de l'application *BookCollection* vu au TP.

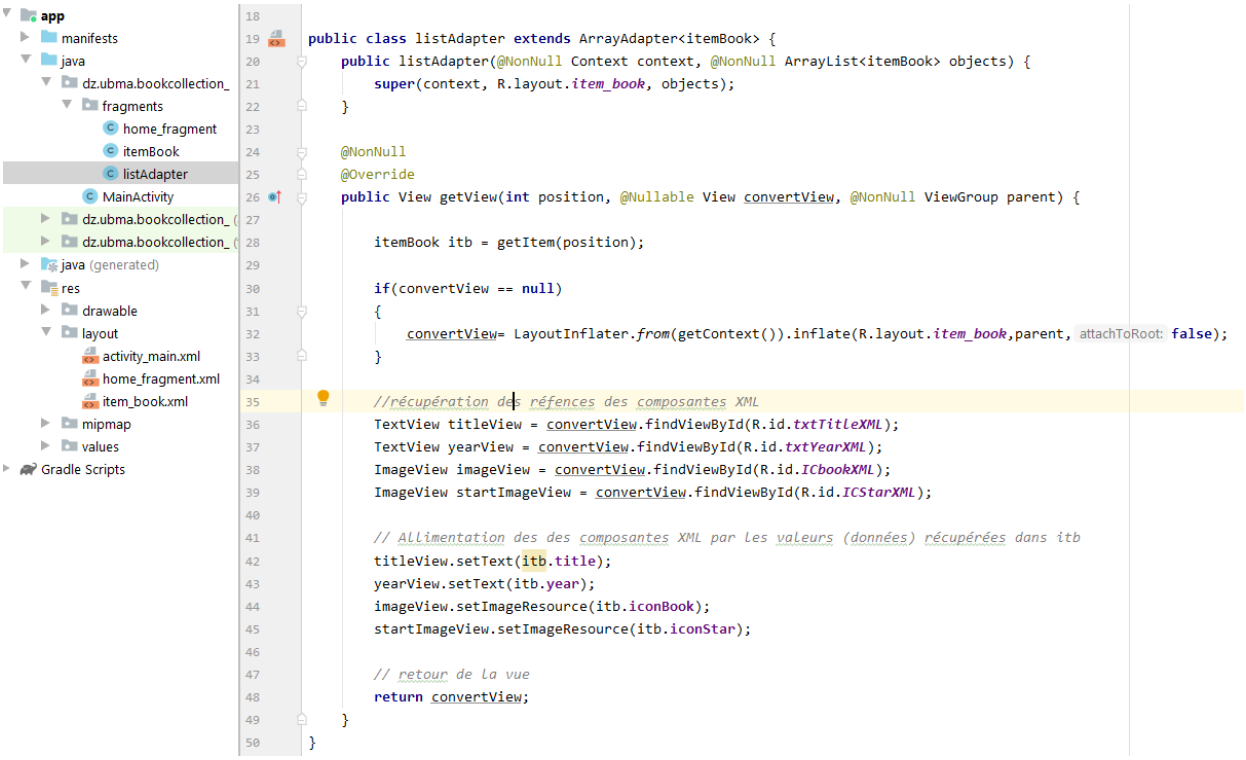

Figure 3.8. La classe ListAdapter de l'application BookCollection (vu au TP)

Une *ListView* tente de réutiliser les objets de vue afin d'améliorer les performances et d'éviter un décalage en réponse aux défilements de l'utilisateur. Pour tirer parti de cette fonctionnalité, il faut vérifier que la valeur *convertView* récupérée comme paramètre de la méthode getView(...) est nulle avant de créer un nouvel objet de vue.

3.4.3. Les items de listView

Une *listView* contient souvent une liste d'une entité composée de plusieurs attributs. Dans le cas de l'application bookCollection, la listView contient une liste des items 'Livre' où chaque item est composé des attributs : Titre du livre, année d'Edition, photo de couverture,…

Dans le cas où l'item est composé de plusieurs attributs, une classe de cet item est créé est passée au *listAdapter* pour récupérer ses attributs par la fonction *getItem*(*position*) où le paramètre position indique l'index de l'item dans la liste des items.

Ci-dessous un exemple d'item utilisé dans le cas de l'application bookCollection :

itemBook.java

```java
package dz.ubma.bookcollection_.fragments;

public class itemBook {
    String title;
    String year;
    int iconBook;
    int iconStar;

    public itemBook(String title, String year, int iconBook, int iconStar) {
        this.title = title;
        this.year = year;
        this.iconBook = iconBook;
        this.iconStar = iconStar;
    }
}
```

Figure 3.9. déclaration de la classe item Livre.

3.4.4. Les actions sur les listView

Des actions sur les listView peuvent être définis tels que : onClick, onDrag, onFocusChange, etc. Dans ce cas, il faut activer les événements et définir ses actions. Comme par exemple on peut définir l'action du clic sur un item comme suit :

```java
lvBooks.setClickable(true);
lvBooks.setOnItemClickListener(new AdapterView.OnItemClickListener() {
    @Override
    public void onItemClick(AdapterView<?> parent, View view, int position, long id) {
        // l'action Souhaitée
    }
});
```

Figure 3.10. l'ajout d'une action de click sur les items d'une listView.

3.5. Conclusion

Dans ce chapitre, on a présenté des éléments de base d'une application mobile qui sont les fragments et les listeView et une introduction à l'utilisation d'adaptateur, et d'alimentation d'une listeView par des données. A la fin de ce chapitre composé aussi d'un TP (voir l'Annexe), l'étudiant sera capable de définir les différentes ressources nécessaires à son application telles

que les ressources Images, icones, styles, dimensions, etc. et il sera prêt pour entamer des éléments plus complexes.

Bibliographie et Webographie

[18] « Exemples de ListView Android avec des collections - Exemples Android », 19 mai 2021. https://camposha.info/android-examples/fr/andorid-listview-collection/ (consulté le 27 janvier 2022).