Les Fragments

03 mai 2017

Rôle des fragments

- Un Fragment est une partie du comportement d'une application ou de son interface utilisateur.

- Est une section modulaire qui fait partie d'une activité
  - Elle possède son propre cycle de vie.
  - Reçoit ses propres évènements.
  - Peut être ajoutée, supprimée dynamiquement.

- Peut être considéré comme une sous activité.

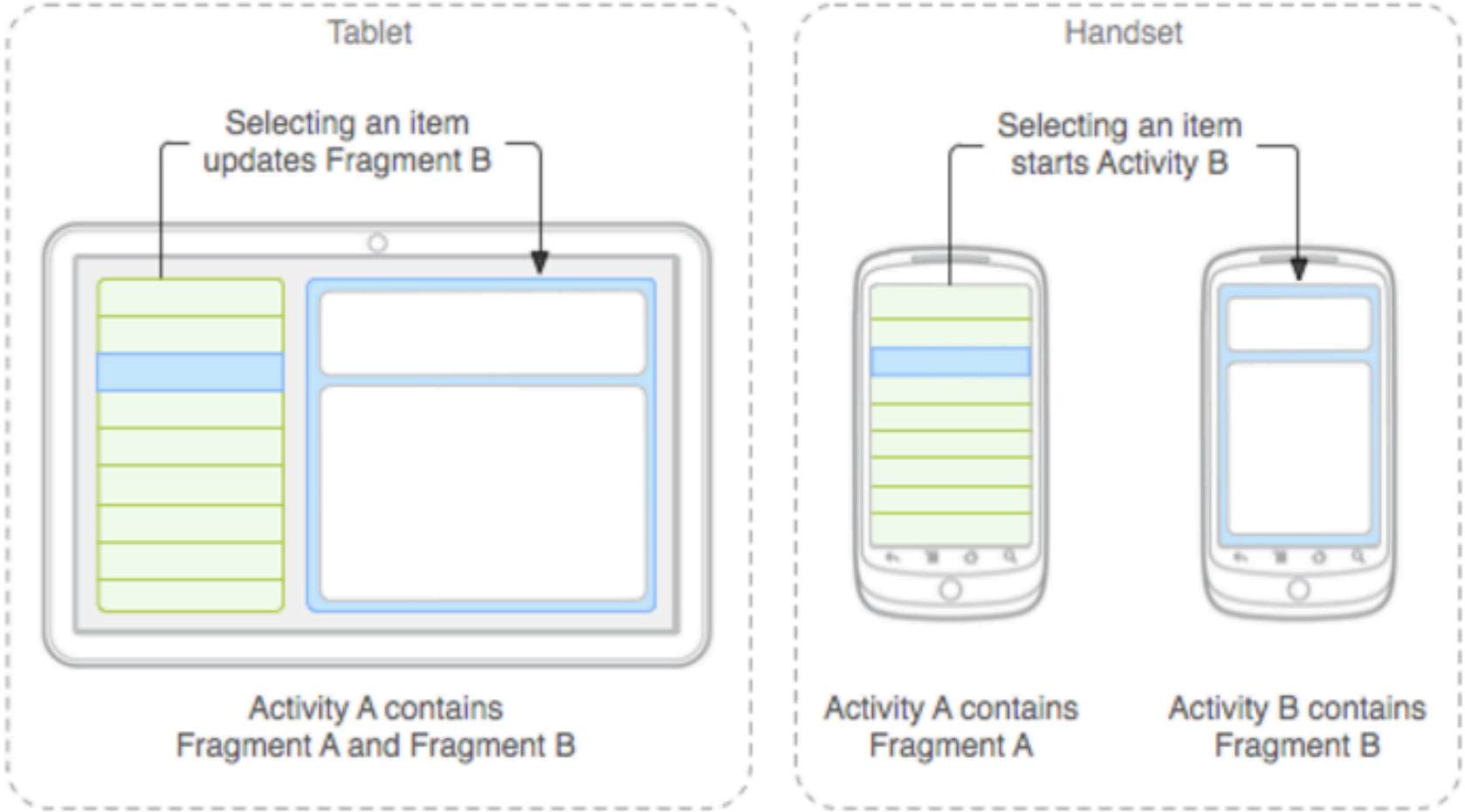
- Notion introduite à partir de Android 3.0 (API level 11).

- Permet de créer des interfaces utilisateurs dynamiques et flexibles
  - Qui s'adaptent aux différents formats d'écrans;
  - Sans une gestion complexe de la hiérarchie des vues.

Exemple

- Une application de « News »
  - Pour un affichage sur un écran large (tablette)
    - Utilise un fragment pour montrer une liste d'articles à gauche et un autre fragment à droite pour afficher les détails d'un article sélectionné.
    - Les deux fragments sont définis dans la même activité.
  - Pour un petit écran (téléphone)
    - Utilisation de deux activités qui intègre chacune un seul fragment.

Exemple


Tablet
Selecting an item
updates Fragment B
Activity A contains
Fragment A and Fragment B
Handset
Selecting an item
starts Activity B
Activity A contains
Fragment A
Activity B contains
Fragment B

Création d'un fragment

- Un fragment est créé comme une sous classe de la classe Fragment (ou d'une de ses sous-classe).

- Le code de définition d'un fragment est très similaire à celui de la définition d'une activité :
  - Contient des méthodes de rappel similaires à celles de la définition d'une activité telles que onCreate(), onStart(), onPause() et onStop().

  - Les méthodes de rappel les plus utilisées:
    - onCreate()
      - Appelée lors de la création d'un fragment.

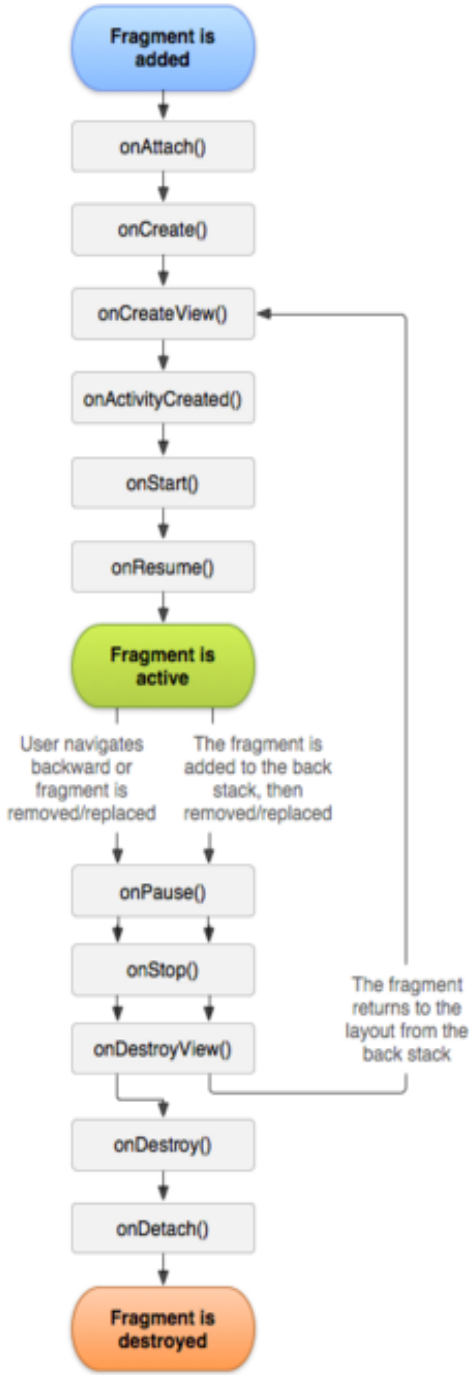
    - onCreateView()
      - Appelée lors du premier affichage de l'interface utilisateur associée au fragment.
      - Retourne une Vue (instance de View) qui est la racine du gabarit associé au fragment.
      - Retourne *null* si le fragment n'est pas associé à une interface utilisateur.

    - onPause()
      - Appelée quand l'utilisateur quitte l'interface du fragment.

Création d'un fragment

- D'autres classes peuvent être utilisées directement pour la création d'un Fragment :
  - DialogFragment
    - Utilisée pour une interface qui représente un dialogue avec l'utilisateur.
  - ListFragment
    - Utilisée pour l'affichage des items d'une liste gérée par un adaptateur.
    - Similaire à ListActivity.
    - Fournit des méthodes pour gérer la vue sous une forme de liste telle que onListItemClick().
  - PreferenceFragment
    - Utilisée pour afficher, sous forme d'une liste, une hiérarchie d'objets qui représentent des préférences.
    - Similaire à PreferenceActivity.
    - Utilisée pour créer une activité de configuration des préférences "settings ».

Création d'un fragment



Création d'un fragment: partie d'une interface utilisateur

- Un fragment est souvent utilisé comme une partie de l'interface utilisateur définie par une activité.

- Le fragment fournit son propose gabarit:
  - Besoin d'implémenter la méthode de rappel onCreateView()
    - Appelée pour l'affichage du fragment.
    - Retourne la Vue racine du gabarit.

  - Pas besoin d'implémenter la méthode de rappel onCreateView si le fragment est créé à partir d'une sous classe de Fragment telle que ListFragment
    - L'implémentation par défaut de onCreateView() retourne la Vue appropriée (par exemple ListView).

Création d'un fragment: partie d'une interface utilisateur

```
public static class ExampleFragment extends Fragment {
    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
                             Bundle savedInstanceState) {
        // Inflate the layout for this fragment
        return inflater.inflate(R.layout.example_fragment, container, false);
    }
}
```

Création d'un fragment: partie d'une interface utilisateur

- Le paramètre *container* indique le gabarit instance de ViewGroup où le fragment va être inséré.

- Le paramètre *savedInstanceState* est un bundle qui fournit les données sauvegardées de la précédente instance du fragment.
  - Utilisé quand la méthode *onResume()* est invoquée.

- La méthode *inflate()* method a trois paramètres:
  - l'*ID* de la ressource associée au gabarit
  - Le *ViewGroup* qui intègre le gabarit du fragment.
  - Un booléen qui indique si le gabarit du fragment doit être attaché au *ViewGroup* ou pas.

Création d'un fragment: ajouter un fragment à une activité

- Deux stratégies pour ajouter un fragment à une activité :
  - Déclarer le fragment à l'intérieur du gabarit de l'activité
    - Spécifier les propriétés du gabarit comme s'il était une vue

```
<?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.ArticleListFragment"
            android:id="@+id/list"
            android:layout_weight="1"
            android:layout_width="0dp"
            android:layout_height="match_parent" />
    <fragment android:name="com.example.news.ArticleReaderFragment"
            android:id="@+id/viewer"
            android:layout_weight="2"
            android:layout_width="0dp"
            android:layout_height="match_parent" />
</LinearLayout>
```

Création d'un fragment: ajouter un fragment à une activité

- L'attribut android:name dans l'élément <fragment> spécifie la class du Fragment à instancier dans le gabarit.
  - Quand le système crée le gabarit de l'activité :
    - Instancie tous les fragments spécifiés dans le gabarit.
    - Appelle la méthode onCreateView() pour chaque fragment pour récupérer leurs gabarits respectifs.
    - Insert la vue retournée par le fragment directement à la place de l'élément <fragment> .

Création d'un fragment: ajouter un fragment à une activité

- Deux stratégies pour ajouter un fragment à une activité :
  - Ajouter le fragment de manière programmatique à un *ViewGroup* existant.
  - À n'importe quel moment, des fragments peuvent être ajoutés au gabarit d'une activité.
  - Utilisation des méthodes de la classe *FragmentTransaction*.

```
FragmentManager fragmentManager = getFragmentManager();
FragmentTransaction fragmentTransaction = fragmentManager.beginTransaction();
```

- Ajouter le fragment en utilisant la méthode *add()* :

```
ExampleFragment fragment = new ExampleFragment();
fragmentTransaction.add(R.id.fragment_container, fragment);
fragmentTransaction.commit();
```

  - Le premier argument de la méthode add est une instance de ViewGroup dans la quelle le fragment doit être inséré
    - » Spécifié par un identifiant de ressource ID
  - Le deuxième paramètre est le Fragment à ajouter.

  - Utilisation de la méthode commit pour opérationnaliser les changements

Création d'un fragment: sans interface utilisateur

- Il est possible d'utiliser un fragment sans interface utilisateur : qui implémente un comportement en arrière plan.

- Ajout du fragment par utilisation de la méthode *Add().*
  - une chaîne de caractères comme valeur du paramètre ID du *ViewGroup*.

- Il n'est pas nécessaire d'implémenter la méthode de rappel *onCreateView()*.

- Utilisation de la méthode *findFragmentByTag()* pour récupérer la référence vers le fragment à partir de l'activité.

Gestion des fragments

- Utilisation de la classe *FragmentManager* pour la gestion des fragments dans les activités.
  - Utilisation de la méthode *getFragmentManager()*.

- Quelques méthodes d'un *FragmentManager*:

  - Retourner les fragments dans l'activité :
    - *findFragmentById()*
      - Pour les fragments associés à une UI.
    - *findFragmentByTag()*
      - Pour les fragments qui ne sont pas associés à une UI.

  - Récupérer des fragments de la pile de « retour en arrière »:
    - *popBackStack()*
      - Similaire à la commande retour arrière de l'utilisateur.

  - Enregistrer un écouteur (listner) pour les changements dans la pile « retour en arrière ».
    - *addOnBackStackChangedListener()*.

Réalisation des Transactions

- Tout changement à opérer sur une activité est appelé Transaction
  - Réalisable par les méthode de la classe *FragmentTransaction*.

- Les transactions peuvent être sauvegardées dans la pile « retour en arrière » gérée par l'activité
  - Permet à l'utilisateur de retourner en arrière par rapport aux changements liés aux fragments.

- Une transaction est un ensemble de changements qui doivent se réaliser en même temps.
  - Exemples *: add(), remove()* et *replace()*.

- La méthode commit permet de lancer e même temps ces changements

- Il est possible d'appeler la méthode *addToBackStack()* pour ajouter la transaction à la pile « retour en arrière »
  - Pour être capable de retourner en arrière en cliquant sur le bouton « Retour Arrière ».

Réalisation des Transactions

```
// Create new fragment and transaction
Fragment newFragment = new ExampleFragment();
FragmentTransaction transaction = getFragmentManager().beginTransaction();

// Replace whatever is in the fragment_container view with this fragment,
// and add the transaction to the back stack
transaction.replace(R.id.fragment_container, newFragment);
transaction.addToBackStack(null);

// Commit the transaction
transaction.commit();
```

- Pour retrouver le fragment par retour arrière : onBackPressed()

```
@Override
public void onBackPressed() {
    if (getFragmentManager().getBackStackEntryCount() > 0) {
        getFragmentManager().popBackStack();
    } else {
        super.onBackPressed();
    }
}
```

Communication entre activité et fragments

- La méthode *getActivity()* permet à un fragment d' accéder à l'instance de l'activité hébergeante :

```
View listView = getActivity().findViewById(R.id.list);
```

- Les méthodes *findFragmentById()* et *findFragmentByTag* invoquées sur une instance de *FragmentManager* permettent à une activité de récupérer la référence vers un fragment :

```
ExampleFragment fragment = (ExampleFragment) getFragmentManager().findFragmentById(R.id.example_fragment);
```

Communication entre activité et fragments

- Création d'évènements de rappel vers l'activité:
  - Dans certains cas un fragment a besoin de partager des évènements avec son activité hébergeante
    - Définition d'une interface de rappel dans le fragment dont l'activité doit implémenter.
    - Quand l'activité reçoit un rappel via cette interface, elle peut partager des informations avec d'autres fragments si nécessaire.

- Exemple: communication entres fragments de l'application News:

```
public static class FragmentA extends ListFragment {
    ...
    // Container Activity must implement this interface
    public interface OnArticleSelectedListener {
        public void onArticleSelected(Uri articleUri);
    }
    ...
}
```

Communication entre activité et fragments

- Pour informer le fragment B d'un évènement dans le fragment A, l'activité qui héberge le fragment :
  - Implémente l'interface *OnArticleSelectedListener*.
  - Surcharge la méthode *onArticleSelected()*.

- Pour forcer l'activité qui héberge le fragment à implémenter cette interface:
  - La méthode *onAttach()* crée une instance de *OnArticleSelectedListener* en castrant l'activité qui est passée comme paramètre de cette méthode.
    - La méthode est appelé quand le système ajoute le fragment à l'activité.

- Si l'activité n'a pas implémenté l'interface en question, le fragment lève une exception *ClassCastException.*

Communication entre activité et fragments

```
public static class FragmentA extends ListFragment {
    OnArticleSelectedListener mListener;
    ...
    @Override
    public void onAttach(Activity activity) {
        super.onAttach(activity);
        try {
            mListener = (OnArticleSelectedListener) activity;
        } catch (ClassCastException e) {
            throw new ClassCastException(activity.toString() + " must implement OnArticleSelectedListener");
        }
    }
    ...
}
```

Communication entre activité et fragments

```
public static class FragmentA extends ListFragment {
    OnArticleSelectedListener mListener;
    ...
    @Override
    public void onListItemClick(ListView l, View v, int position, long id) {
        // Append the clicked item's row ID with the content provider Uri
        Uri noteUri = ContentUris.withAppendedId(ArticleColumns.CONTENT_URI, id);
        // Send the event and Uri to the host activity
        mListener.onArticleSelected(noteUri);
    }
    ...
}
```