**HelHa**
Tournai
Bachelor en Informatique

2019 - 2020
Android

**Développement Android**

# Clarifier l'interface via des fragments





Version du document : 7 août 2019

*Ce TD est étroitement inspiré par le livre "Android Programming, the big nerd ranch guide", 3è edition, par Bill Phililips, et Al. et en particulier par les chapitres 7 à 11.*

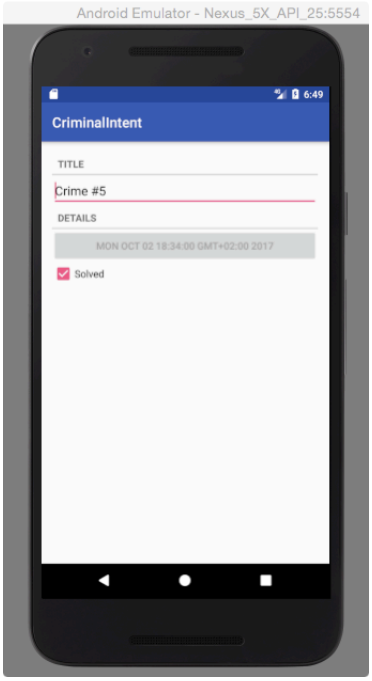
# 1 Utilité des fragments

Le concept d'activité permet de faire des applications simples, mais dès que l'on veut faire une application comportant plusieurs écrans possédant des liens entre eux, par exemple une application *master/detail*, la notion d'activité atteint rapidement ses limites.

La notion de *fragment* a été introduite avec l'introduction des premières tablettes Android, lorsque le besoin d'interfaces plus élaborées s'est fait sentir. Un fragment est un 'morceau' d'interface, un layout et son contrôleur. Un écran est défini par une activité qui elle-même est composée d'un ou plusieurs fragments. Les fragments peuvent être manipulés, retirés ou ajoutés. Un fragment peut également être composé d'autres fragments.

# 2 Nouvelle application : CriminalIntent

Pour illustrer la notion de fragment, nous allons développer une nouvelle application : CriminalIntent. Cette application permettra de visualiser des *'crimes'* et, dans un deuxième temps, de les parcourir. Le premier écran, très simple, permettra d'illustrer l'utilisation des fragments.




## 2.1 Création du projet

Créez un nouveau projet *Criminal Intent* avec une 'Empty Activity'. L'activité principale est nommée `CrimeActivity`. Le contrôleur Java associé, dans le fichier `CrimeActivity.java`, est une sous-classe de `AppCompatActivity` [1].

Définissez immédiatement les constantes textuelles du projet dans le fichier `strings.xml` :

```
<resources>
    <string name="app_name">CriminalIntent</string>
    <string name="crime_title_hint">Enter a title for the crime</string>
    <string name="crime_title_label">Title</string>
    <string name="crime_details_label">Details</string>
    <string name="crime_solved_label">Solved</string>
</resources>
```

## 2.2 Création du modèle : Crime.java

Créez une première classe du modèle, la classe `Crime`, qui a comme attributs :

1. cela permet de gérer de manière transparente les différentes versions Android via la *Support Library*, et de rester compatible avec de plus vieilles versions.

— `mId` un identifiant unique de type `java.util.UUID`

— `mTitle` un titre de type `String`

— `mDate` de type `java.util.Date`

— `mSolved` un booléen indiquant si le cas est résolu ou non.

Ajoutez un constructeur qui initialise l'id (`mId = UUID.randomUUID();`) et la date à la date du jour.

Générez les accesseurs pour tous les attributs (getters) et les setters pour les 3 derniers.

## 2.3 Création de l'activité hébergeante

Pour héberger un fragment, une activité doit faire 2 choses :

— définir un endroit de son layout où placer le fragment

— gérer le cycle de vie du fragment.

En effet tout comme une activité, un fragment est régi par un cycle de vie précis. Ce cycle de vie est très similaire au cycle de vie d'une activité et sera dirigé par l'activité qui l'héberge. On retrouve essentiellement les mêmes méthodes permettant d'intervenir lors des changements d'état du fragment : `onCreate, onPause, onResume, onStop,` etc.

Modifiez le fichier de layout `activity_crime.xml` comme suit :

```
<FrameLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/fragment_container"
    android:layout_width="match_parent"
    android:layout_height="match_parent"/>
```

Ce layout sera réservé pour l'hébergement du fragment.

## 2.4 Création d'un fragment

La création d'un fragment se fait en 3 étapes :

— création de la vue dans un fichier XML ;

— définition du contrôleur en Java ;

— lier la vue et le contrôleur dans le code Java.

### 2.4.1 Création du layout

Dans le répertoire `res/layout` de la vue projet, faites New → Layout resource file. Nommez ce fichier `fragment_crime.xml` et indiquez *LinearLayout* comme élément racine.

Modifiez ce fichier comme suit :

```xml
<?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"
    android:layout_margin="16dp">

    <TextView
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        style="?android:listSeparatorTextViewStyle"
        android:text="@string/crime_title_label"/>
    <EditText
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:id="@+id/crime_title"
        android:hint="@string/crime_title_hint"/>
    <TextView
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        style="?android:listSeparatorTextViewStyle"
        android:text="@string/crime_details_label"/>
    <Button
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:id="@+id/crime_date"/>
    <CheckBox
        android:id="@+id/crime_solved"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="@string/crime_solved_label"/>
</LinearLayout>
```

Vous pouvez visualiser le layout en mode *design*.

**Question 1**

Que signifie le '?' dans `style="?android:listSeparatorTextViewStyle"` ?

### 2.4.2 Création du contrôleur

Ajoutez une nouvelle classe `CrimeFragment` qui sera le contrôleur du fragment, cette classe est une sous-classe de la classe `android.support.v4.app.Fragment`.

Ajoutez un attribut `mCrime` de type `Crime` et redéfinissez la méthode `onCreate` afin d'initialiser le modèle :

```
    super.onCreate(savedInstanceState);
    mCrime = new Crime();
```

### 2.4.3 Lier contrôleur et vue

Finalement, vous pouvez lier le contrôleur et la vue, c'est-à-dire, récupérer une référence aux widgets de la vue et initialiser les listeners. Cela se fait par redéfinition de la méthode `onCreateview` :

```
1  public View onCreateView(LayoutInflater inflater, ViewGroup container,
2                           Bundle savedInstanceState) {
3      // inflate the fragment_crime view
4      View v = inflater.inflate(R.layout.fragment_crime,container,false);
5
6      // configure the view
7      mTitleField = (EditText) v.findViewById(R.id.crime_title);
8      mTitleField.setText(mCrime.getTitle());
9      mTitleField.addTextChangedListener(new TextWatcher() {
10         @Override
11         public void beforeTextChanged(CharSequence s, int start, int count, int after) {
12             // do nothing
13         }
14         @Override
15         public void onTextChanged(CharSequence s, int start, int before, int count) {
16             mCrime.setTitle(s.toString());
17         }
18         @Override
19         public void afterTextChanged(Editable s) {
20             // do nothing, everything's done in onTextChanged
21         }
22     });
23
24     mTitleField.setText(mCrime.getTitle());
25
26     mDatebutton = (Button) v.findViewById(R.id.crime_date);
27     mDatebutton.setText(mCrime.getDate().toString());
28     mDatebutton.setEnabled(false); //readonly
29
30     mSolvedCheckBox = (CheckBox) v.findViewById(R.id.crime_solved);
31     mSolvedCheckBox.setChecked(mCrime.isSolved());
32     mSolvedCheckBox.setOnCheckedChangeListener(
33         new CompoundButton.OnCheckedChangeListener() {
34           @Override
35           public void onCheckedChanged(CompoundButton buttonView, boolean isChecked) {
36               mCrime.setSolved(isChecked);
37           }
38         }
39      );
40
41     return v;
42 }
```

On voit que le contrôleur met à jour le modèle lorsque la checkbox est cliquée ou lorsque le titre est modifié.

**Question 2**

1. Quelle est la classe qui permet d'écouter un `TextView` ?
2. Quelle est la classe qui permet d'écouter un `Checkbox` ?

### 2.5 Fragment Manager

Votre fragment est défini et est prêt à l'emploi. Nous allons maintenant l'ajouter au `CrimeActivity`. La gestion des fragments, de leurs ajouts ou de leurs retraits de la vue se fait via le `FragmentManager`.

Ajoutez à la méthode `onCreate` de la classe `CrimeActivity` le code suivant :

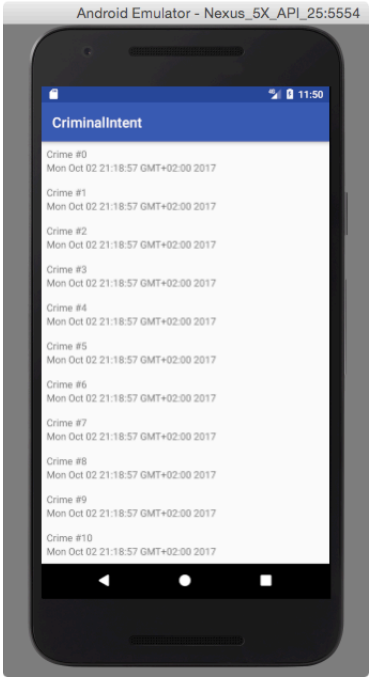
```
1 FragmentManager fm = getSupportFragmentManager();
2 Fragment fragment = fm.findFragmentById(R.id.fragment_container);
3 if(fragment==null) {
4     fragment = new CrimeFragment();
5     fm.beginTransaction()
6         .add(R.id.fragment_container, fragment)
7         .commit();
8     }
```

Nous voyons que le `FragmentManager` utilise des transactions afin de mettre à jour la vue. L'appel à `commit` va rendre effectif les changements.

Votre écran est maintenant complet.

## 3 Afficher une liste

Nous allons maintenant implémenter un écran plus complexe permettant d'afficher une liste de *crimes*. Cet écran est composé d'un container, un `RecyclerView`, qui contient une liste de vue. Chacune de ces vues est un fragment et présente un *crime*. L'interêt de l'objet `RecyclerView` est qu'il permet d'afficher et de parcourir (*scroll*) une liste d'élément qui peut être très très longue, tout en utilisant, grâce à un mécanisme de recyclage, un minimum de ressources.




## 3.1 Mise à jour du modèle

Nous commençons par mettre à jour le modèle en ajoutant la classe `CrimeLab` qui est essentiellement une liste de *crimes*. Cette classe utilise le patron de conception *singleton* :

```
1  public class CrimeLab {
2      private static CrimeLab sCrimeLab;
3
4      private List<Crime> mCrimes;
5
6      public static CrimeLab get(Context context) {
7          if(sCrimeLab == null) {
8              sCrimeLab = new CrimeLab(context);
9          }
10         return sCrimeLab;
11     }
12
13     private CrimeLab(Context context) {
14         mCrimes = new ArrayList<>();
15
16         // initialisation avec des crimes bidons.
17         for(int i = 0; i < 100; i++) {
18             Crime crime = new Crime();
19             crime.setTitle("Crime #" + i);
```

```
            crime.setSolved(i%2==0); // un sur deux résolu
            mCrimes.add(crime);
        }
    }

    public Crime getCrime(UUID id) {
        for (Crime crime : mCrimes) {
            if(crime.getId().equals(id))
                return crime;
        }
        return null;
    }

    public List<Crime> getCrimes() {
        return mCrimes;
    }
}
```

## 3.2 Refactoring : hébergeur de fragments générique

Beaucoup d'activités sont très simples et ne servent qu'à héberger des fragments, dans ce dernier se trouve souvent la complexité. Nous allons *refactoriser* le code afin de pouvoir créer plus facilement des (activités) hébergeurs de fragments.

Le layout associé à la classe `CrimeActivity`, `activity_crime.xml` est déjà entièrement générique. Renommez-le `activity_fragment.xml` à l'aide de la fonction *refactor*.

**Question 3**

Outre le nom du fichier, quels sont le/les changements effectués par le refactoring (renommage du fichier XML) que vous venez de faire ?

Créez la classe abstraite `SingleFragmentActivity` comme suit :

```
public abstract class SingleFragmentActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_fragment);

        FragmentManager fm = getSupportFragmentManager();
        Fragment fragment = fm.findFragmentById(R.id.fragment_container);
        if(fragment==null) {
            fragment = createFragment();
            fm.beginTransaction()
                    .add(R.id.fragment_container, fragment)
                    .commit();
        }
    }
    protected abstract Fragment createFragment() ;
}
```

Cette classe reprend la logique générique présente dans la classe `CrimeActivity` :

— appeler la méthode `onCreate` de la super-classe ;

— définir le vue ;

— placer le fragment.

La seule partie non générique est l'instanciation du fragment qui est déléguée à la méthode abstraite : `createFragment`.

Vous pouvez maintenant grandement simplifier la classe `CrimeActivity` en la déclarant comme sous-classe de votre nouvelle classe abstraite :

```
1 public class CrimeActivity extends SingleFragmentActivity {
2     @Override
3     protected Fragment createFragment() {
4         return new CrimeFragment();
5     }
6 }
```

Vérifiez que votre application est toujours fonctionnelle.

## 3.3 Ajout de l'activité principale : `CrimeListActivity`

Créez le fragment qui contiendra la liste : `CrimeListFragment`. C'est une sous-classe de `Fragment` et pour le moment cette classe ne contient aucun code, mais elle est nécessaire pour pouvoir écrire l'activité.

Créez une nouvelle classe `CrimeListActivity`, l'activité qui va héberger le fragment que vous venez de déclarer. Grâce au refactoring, cette activité est extrêmement simple :

```
1 public class CrimeListActivity extends SingleFragmentActivity {
2     @Override
3     protected Fragment createFragment() {
4         return new CrimeListFragment();
5     }
6 }
```

Il faut également mettre à jour le *manifest* de l'application (`manifests/AndroidManifest.xml`) afin de déclarer cette nouvelle activité comme le point d'entrée de votre application à la place de `CrimeActivity` :

```
1  (...)
2     <activity android:name=".CrimeListActivity">
3         <intent-filter>
4             <action android:name="android.intent.action.MAIN" />
5             <category android:name="android.intent.category.LAUNCHER" />
6         </intent-filter>
7     </activity>
8     <activity android:name=".CrimeActivity">
9     </activity>
10  (...)
```

Vous pouvez lancer votre application, mis à part le titre, l'écran est vide.

## 3.4 RecyclerView

Vous allez maintenant mettre en place le fragment qui gère la liste. Pour cela, il faut d'abord ajouter une dépendance à votre projet. Allez dans File -> Project Structure et sélectionnez le module `app` (à gauche). Dans l'onglet 'dependencies' utilisez le bouton '+' pour ajouter la dépendance :

`com.android.support.recyclerview.v7.26.0.0-alpha1`.

La version est peut-être légèrement différente chez vous.

Vous pouvez maintenant ajouter un layout pour la liste. Dans `res/layout`, ajoutez un nouveau layout (New -> Layout resource file) que vous nommez `crime_list_fragment.xml` et mettez à jour son contenu afin de déclarer un RecyclerView :

```
<?xml version="1.0" encoding="utf-8"?>
<android.support.v7.widget.RecyclerView
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/crime_recycler_view"
    android:layout_width="match_parent"
    android:layout_height="match_parent"/>
```

Complétez le contrôleur `CrimeListFragment` (qui est vide pour le moment) afin de le lier à la vue et en particulier afin de le lier à la RecyclerView :

```
private RecyclerView mCrimeRecyclerView;

@Override
public View onCreateView(LayoutInflater inflater, ViewGroup container, Bundle savedInstanceState) {
    View view = inflater. inflate (R.layout.crime_list_fragment, container, false);
    mCrimeRecyclerView = (RecyclerView) view.findViewById(R.id.crime_recycler_view);
    mCrimeRecyclerView.setLayoutManager(new LinearLayoutManager(getActivity()));
    return view;
}
```

Pour le moment votre vue est toujours vide. Il y a bien un fragment qui peut gérer une liste mais aucun élément ne se trouve dans cette liste.

## 3.5 Éléments de la liste

Il reste à définir les éléments de la liste (au niveau de la vue et du contrôleur).

Ajoutez d'abord un nouveau layout : `list_item_crime.xml`. Il s'agit d'un layout pour visualiser un élément de la vue. C'est ce layout qui sera répété autant de fois qu'il y a d'éléments visibles à l'écran.

```
<?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="wrap_content"
    android:padding="8dp">

    <TextView
        android:id="@+id/crime_title"
```

```
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Crime Title"/>

    <TextView
        android:id="@+id/crime_date"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Crime Date"/>
</LinearLayout>
```

Le contrôleur des éléments de la liste et le lien entre le fragment contenant la liste et chacun des éléments de cette liste se font via 2 classes :

- Le `ViewHolder` agit comme le contrôleur d'une ligne de cette liste
- L' `Adapter` qui lie la liste et chacun des éléments.

## 3.6 ViewHolder

Créez la classe `CrimeHolder` comme classe interne à la classe `CrimeListFragment`. C'est une sous-classe de la classe `RecyclerView.ViewHolder` et agit comme le contrôleur d'un élément de la vue à recyclage.

```
private class CrimeHolder extends RecyclerView.ViewHolder {

    private Crime mCrime;
    private TextView mTitleTextView;
    private TextView mDateTextView;

    public CrimeHolder(LayoutInflater inflater, ViewGroup parent) {
        super(inflater. inflate (R.layout.list_item_crime, parent, false ));
        mTitleTextView = (TextView) itemView.findViewById(R.id.crime_title);
        mDateTextView = (TextView) itemView.findViewById(R.id.crime_date);
}
```

Cette classe ne fait encore rien, hormis 'déplier' (*inflate*) le layout.

## 3.7 Lier la liste et ses éléments : `Adapter`

Il faut encore créer autant d'éléments de la liste que nécessaire, par exemple en fonction de la taille de l'écran. Et gérer la mise à jour des informations. C'est la responsabilité de l'`Adapter`.

Créez la classe `CrimeAdapter` interne à la classe `CrimeListFragment` :

```
private class CrimeAdapter extends RecyclerView.Adapter<CrimeHolder> {

    private List<Crime> mCrimes;

    public CrimeAdapter(List<Crime> crimes) {
        mCrimes = crimes;
    }
    @Override
    public CrimeHolder onCreateViewHolder(ViewGroup parent, int viewType) {
        LayoutInflater layoutInflater = LayoutInflater.from(getActivity ());
        return new CrimeHolder(layoutInflater, parent);
    }
    @Override
```

```
14      public void onBindViewHolder(CrimeHolder holder, int position) {
15      }
16      @Override
17      public int getItemCount() {
18          return mCrimes.size();
19      }
20  }
```

La méthode `onCreateViewHolder` définit ce qu'il faut faire pour initialiser chacun des éléments de la liste (vue). La méthode `getItemCount` donne le nombre d'éléments à afficher. La dernière méthode sera complétée dans un instant.

La classe adapteur utilise et instancie les ViewHolder. Une instance de l'adapteur doit être utilisée dans la liste. Ajoutez donc un adapteur comme attribut de la classe `CrimeListFragment` et ajoutez la méthode de mise à jour `updateUI` que vous appelez dans `onCreateView` :

```
1   public class CrimeListFragment extends Fragment {
2
3       private RecyclerView mCrimeRecyclerView;
4       private CrimeAdapter mCrimeAdapter;
5
6       public View onCreateView(...) {
7           (...);
8           updateUI();
9           return view;
10      }
11      private void updateUI() {
12          CrimeLab crimeLab = CrimeLab.get(getActivity());
13          List<Crime> crimes = crimeLab.getCrimes();
14
15          if(mCrimeAdapter == null) {
16              mCrimeAdapter = new CrimeAdapter(crimes);
17              mCrimeRecyclerView.setAdapter(mCrimeAdapter);
18          } else {
19              mCrimeAdapter.notifyDataSetChanged();
20          }
21      }
22  }
```

Afin d'actualiser l'affichage lorsque l'on vient de modifier un crime, il suffit d'implémenter la méthode `onResume` de la classe `CrimeListFragment` comme suit :

```
1   public void onResume(){
2       super.onResume();
3       updateUI()
4   }
```

Votre application devrait maintenant afficher une liste.

Mais le lien entre le modèle et la vue n'est pas encore effectif. Il faut encore lier le titre et la date de chaque élément à la vue. Pour cela on ajoute la méthode suivante au ViewHolder :

```
1       public void bind(Crime crime) {
2           mCrime = crime;
3           mTitleTextView.setText(crime.getTitle());
4           mDateTextView.setText(crime.getDate().toString());
5       }
```

et on l'appelle dans l'adapteur :

```
    @Override
    public void onBindViewHolder(CrimeHolder holder, int position) {
        Crime crime = mCrimes.get(position);
        holder.bind(crime);
    }
```

La liste en maintenant complète !

# 4 Passage de paramètre entre fragments

Dans cette dernière section, nous allons lancer une vue détaillée concernant un *crime* lorsque celui-ci est cliqué dans la liste.

Pour cela, il suffit d'ajouter un listener pour l'événement correspondant sur le ViewHolder :

```
public CrimeHolder(LayoutInflater inflater, ViewGroup parent) {
 (...)

    itemView.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View v) {
            Intent intent = new Intent(getActivity(), CrimeActivity.class);
            intent.putExtra("crime_id", mCrime.getId());
            startActivity(intent);
        }
    });
}
```

On lance une nouvelle activité de type `CrimeActivity` lorsqu'un élément de la liste est cliqué. On passe à cette activité un extra : l'id du crime en question.

Il reste à mettre à jour `CrimeActivity` afin d'utiliser cet extra pour afficher le crime en question. Dans la méthode `onCreate` de la classe `CrimeFragment` :

```
@Override
  public void onCreate(@Nullable Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      //mCrime = new Crime();
      UUID crime_id = (UUID) getActivity().getIntent().getSerializableExtra("crime_id");
      mCrime = CrimeLab.get(getActivity()).getCrime(crime_id);
  }
```

Votre application est fonctionnelle (enfin, on l'espère).

Vérifiez que la liste est mise à jour lorsque vous éditez le titre dans la vue détail et que vous revenez (back button) à la liste.

# 5 Exercices

1. Faites apparaître dans la liste le fait qu'un crime soit résolu ou non. Par exemple avec une couleur de fond vert lorsque c'est résolu et rouge lorsque cela ne l'est pas, ou avec une image.
2. Ajoutez *une gravité* comme attribut à la classe `Crime` : un crime peut être Mild (léger), Moderate (moyen), Severe (sérieux). Faîtes apparaître cela dans la liste (`CrimeListFragment`) et dans le détail `CrimeFragment`. Faîtes en sorte que la gravité soit éditable dans la vue détail.
3. Améliorez l'affichage de toute l'application.