Android: Fragments

http://developer.android.com/guide/components/fragments.html

http://developer.android.com/training/basics/fragments/index.html

Ferruccio Damiani

Università di Torino
www.di.unito.it/~damiani

Mobile Device Programming
(Laurea Magistrale in Informatica, a.a. 2017-2018)

Outline

Outline

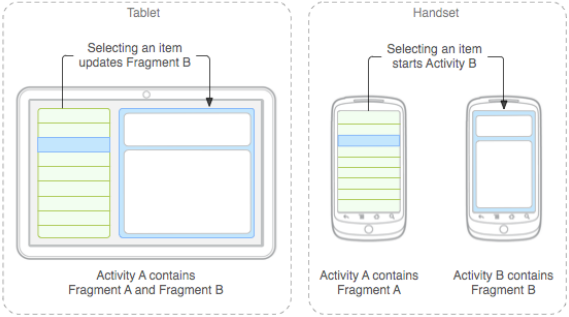
Fragments

- Introduced in Android 3.0 (API level 11) to support more dynamic and flexible UI designs on large screens
- One can combine multiple fragments in a single activity to build a multi-pane UI and reuse the same fragments in multiple activities
- Represent behaviors or portions of user interface in Activities
  - Must always be embedded in an activity and their lifecycle is directly affected by the one of the host activity
  - Can be manipulated independently
- You may also use:
  - A fragment without its own UI as an invisible worker for the activity
  - Special-purpose fragments: DialogFragment, ListFragment, PreferenceFragment

Example

A news application can use one fragment to show a list of articles on the left and another fragment to display an article on the right—both fragments appear in one activity, side by side, and each fragment has its own set of lifecycle callback methods and handle their own user input events. Thus, instead of using one activity to select an article and another activity to read the article, the user can select an article and read it all within the same activity, as illustrated in the tablet layout in figure below.

The application can embed two fragments in Activity A, when running on a tablet-sized device. However, on a handset-sized screen, there's not enough room for both fragments, so Activity A includes only the fragment for the list of articles, and when the user selects an article, it starts Activity B, which includes the second fragment to read the article. Thus, the application supports both tablets and handsets by reusing fragments in different combinations, as illustrated in figure below.



Reuse

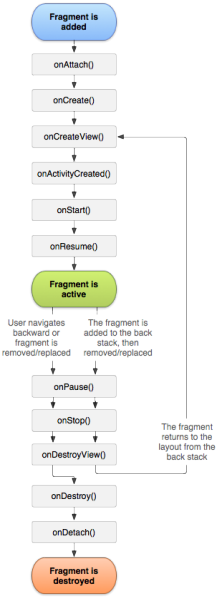
- You should design each fragment as a modular and reusable activity component
  - Each fragment defines its own layout and its own behavior
- You can include one fragment in multiple activities, so you should design for reuse
  - Avoid directly manipulating one fragment from another fragment
  - A modular fragment allows you to change your fragment combinations for different screen sizes

A fragment is a modular section of an activity, which has its own lifecycle, receives its own input events, and which you can add or remove while the activity is running (sort of like a "sub activity" that you can reuse in different activities).

Outline

To create a fragment, you must create a subclass of Fragment (or an existing subclass of it). The Fragment class has code that looks a lot like an Activity. You should implement at least:

- onCreate() initializes essential components of the fragment that you want to retain when the fragment is paused or stopped, then resumed
- onCreateView() called when it's time for the fragment to draw its user interface for the first time
  - It returns a View that is the root of the fragment's layout
  - It can return null if the fragment does not provide a onPause() called when the user is leaving the fragment
- onPause() called when the user is leaving the fragment
  - This is usually where you should commit any changes that should be persisted



Adding a user interface

A fragment is usually used as part of an activity's user interface and contributes its own layout to the activity.

Example

A subclass of Fragment that loads a layout from the example_fragment.xml file (R.layout.example_fragment is a reference to a layout resource named example_fragment.xml saved in the application resources):

```
1 class ExampleFragment : Fragment() {
2     override fun onCreateView(inflater: LayoutInflater, container: ViewGroup?, savedInstanceState: Bundle?): View? {
3         // Inflate the layout for this fragment
4         return inflater.inflate(R.layout.example_fragment, container, false)
5     }
6 }
```

The container parameter passed to onCreateView() is the parent ViewGroup (from the activity's layout) in which your fragment layout will be inserted.
The inflate() method takes three arguments:

- The resource ID of the layout you want to inflate.
- The ViewGroup to be the parent of the inflated layout.
- A boolean indicating whether the inflated layout should be attached to the ViewGroup (the second parameter) during inflation. (In this case, this is false because the system is already inserting the inflated layout into the container—passing true would create a redundant view group in the final layout.)

Add a Fragment to an Activity

You've seen how to create a fragment that provides a layout. Next, you need to add the fragment to your activity.

Usually, a fragment contributes a portion of UI to the host activity, which is embedded as a part of the activity's overall view hierarchy.
There are two ways you can add a fragment to the activity layout:

1. **Declare the fragment inside the activity's layout file.** In this case, you can specify layout properties for the fragment as if it were a view.
2. **Or, programmatically add the fragment to an existing ViewGroup.** At any time while your activity is running, you can add fragments to your activity layout. You simply need to specify a ViewGroup in which to place the fragment.
   - To make fragment transactions in your activity (such as add, remove, or replace a fragment), you must use APIs from FragmentTransaction.

1. Declare the fragment inside the activity's layout file

Example (The layout file for an activity with two fragments)

```
<?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>
```

Note: Each fragment requires a unique identifier. There are three ways to provide an ID for a fragment:

- Supply the android:id attribute with a unique ID.
- Supply the android:tag attribute with a unique string.
- If you provide neither of the previous two, the system uses the ID of the container view.

- The android:name attribute in the <fragment> specifies the Fragment class to instantiate in the layout.
- When the system creates this activity layout, it instantiates each fragment specified in the layout and calls the onCreateView() method for each one, to retrieve each fragment's layout. The system inserts the View returned by the fragment directly in place of the <fragment> element.

2. Or, programmatically add the fragment to an existing ViewGroup

Example

You can get an instance of FragmentTransaction from your Activity like this:

```
1 val fragmentTransaction = supportFragmentManager.beginTransaction()
```

You can then add a fragment using the add() method, specifying the fragment to add and the view in which to insert it":

```
1 val fragment = ExampleFragment()
2 fragmentTransaction.add(R.id.fragment_container, fragment)
3 fragmentTransaction.commit()
```

- The first argument passed to add() is the ViewGroup in which the fragment should be placed, specified by resource ID, and the second parameter is the fragment to add.
- Once you've made your changes with FragmentTransaction, you must call commit() for the changes to take effect.

To add a fragment without a UI, add the fragment using add(Fragment, String)

- Supply a unique "tag" for the fragment, rather than a view ID.
- It does not receive a call to onCreateView()—so you don't need to implement that method.

Outline

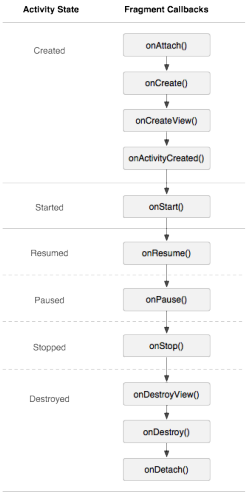
FragmentManager

- Needed to Manage fragments in an activity
- getFragmentManager() provides a manager that can then be used to
  - Get fragments that exist in the activity—with findFragmentById() (for fragments that provide a UI in the activity layout) or findFragmentByTag() (for fragments that do or don't provide a UI).
  - Pop fragments off the back stack—with popBackStack() (simulating a Back command by the user).
  - Register a listener for changes to the back stack—with addOnBackStackChangedListener().
  - Open a FragmentTransaction to add and remove fragments

Handling the Fragment Lifecycle

A fragment can exist in three states:

- **Resumed.** The fragment is visible in the running activity.
- **Paused.** Another activity is in the foreground and has focus, but the activity in which this fragment lives is still visible (the foreground activity is partially transparent or doesn't cover the entire screen).
- **Stopped.** The fragment is not visible. Either the host activity has been stopped or the fragment has been removed from the activity but added to the back stack. A stopped fragment is still alive (all state and member information is retained by the system). However, it is no longer visible to the user and will be killed if the activity is killed.



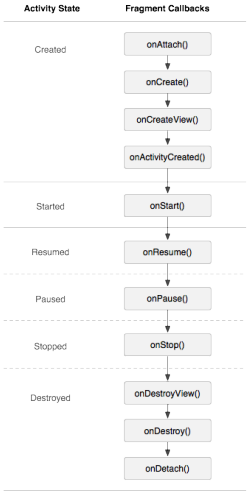
Coordinating with the activity lifecycle

Lifecycle of the activity in which the fragment lives directly affects the lifecycle of fragment: each lifecycle callback for the activity results in a similar callback for each fragment.

- For example, when the activity receives onPause(), each fragment in the activity receives onPause().

Fragments have a few extra lifecycle callbacks:

- **onAttach().** Called when the fragment has been associated with the activity (the Activity is passed in here).
- **onCreateView().** Called to create the view hierarchy associated with the fragment.
- **onActivityCreated().** Called when the activity's onCreate() method has returned.
- **onDestroyView().** Called when the view hierarchy associated with the fragment is being removed.
- **onDetach().** Called when the fragment is being disassociated from the activity.



Performing Fragment Transactions

A great feature about using fragments in your activity is the ability to add, remove, replace, and perform other actions with them, in response to user interaction.

Example

You can acquire an instance of FragmentTransaction from the FragmentManager like this:

```
1 //   supportFragmentManager
2 val fragmentTransaction = supportFragmentManager.beginTransaction()
```

- Each transaction is a set of changes that you want to perform at the same time. You can set up all the changes you want to perform for a given transaction using methods such as add(), remove(), and replace(). Then, to apply the transaction to the activity, you must call commit().
- Before you call commit(), however, you might want to call addToBackStack(), in order to add the transaction to a back stack of fragment transactions. This back stack is managed by the activity and allows the user to return to the previous fragment state, by pressing the Back button.

Example

Here's how you can replace one fragment with another, and preserve the previous state in the back stack:

```
1
2  // Create new fragment
3  val newFragment = ExampleFragment()
4
5  with(supportFragmentManager.beginTransaction()) {
6      // Replace whatever is in the fragment_container view with this fragment,
7      // and add the transaction to the back stack
8      replace(R.id.fragment_container, newFragment)
9      addToBackStack(null)
10
11     // Commit the transaction
12     commit()
13 }
```

Communication between Fragments and Activities

- A given instance of a fragment is directly tied to the activity that contains it
  - The fragment can then access the activity instance with getActivity() and easily perform tasks such as find a view in the activity
- Likewise the activity can call methods in the fragment by acquiring a reference to the fragment from the FragmentManager
- A fragment can also contribute menu items to the activity's OptionsMenu and, consequently, to the ActionBar by implementing onCreateOptionsMenu()

Outline

The example

[git clone https://<login>@gitlab2.educ.di.unito.it/ProgMob/PDM18kotlin3v1.git]

- There are two fragments
  - One to show a list of course titles
  - Another to show a summary of the course when selected from the list
- The fragments are used by two activity activity
  - One activity shows the fragment with the list of course titles in portrait, and both fragment in landscape
  - One activity to show only the fragment with course details
- It also demonstrates how to provide different configurations of the fragments, based on the screen configuration

The example: About visualization (1/5)

- In MainActivity.kt
- The main activity applies a layout in the usual way, during onCreate(), and block the orientation for large display:

```
1    override fun onCreate(savedInstanceState: Bundle?) {
2        super.onCreate(savedInstanceState)
3        setContentView(R.layout.activity_main)
4
5        if (resources.getBoolean(R.bool.large)) {
6            requestedOrientation = ActivityInfo.SCREEN_ORIENTATION_LANDSCAPE
7        }
8
9        ...
10   }
```

- Look at how and where the boolean property R.bool.large is defined

The example: About visualization (2/5)

- In activity_main.xml

- The android:name attribute in the <fragment> specifies the Fragment class to instantiate in the layout
- The tools:layout attribute is typically set in a <fragment> tag and is used to record which layout you want to see rendered at designtime (at runtime, this will be determined by the actions of the fragment class listed by the tag)

```
1  <?xml version="1.0" encoding="utf-8"?>
2  <android.support.constraint.ConstraintLayout  ...>
3
4      <fragment
5          android:id="@+id/fragment"
6          android:name="it.unito.di.educ.pdm18kotlin3v1.CourseFragment"
7          android:layout_width="0dp"
8          android:layout_height="0dp"
9          android:layout_marginBottom="8dp"
10         tools:layout="@layout/fragment_course_list"
11         ... />
12
13 </android.support.constraint.ConstraintLayout>
```

- Look at the definition of multiple activity_main.xml files

The example: About visualization (3/5)

- In fragment_course_list.xml
- app:layoutManager - The RecyclerView uses a layout manager to position the individual items on the screen and determine when to reuse item views that are no longer visible to the user. To reuse (or recycle) a view, a layout manager may ask the adapter to replace the contents of the view with a different element from the dataset.

  The Android Support Library includes three standard layout managers:
  - LinearLayoutManager: arranges the items in a one-dimensional list
  - GridLayoutManager: arranges the items in a two-dimensional grid, like the squares on a checkerboard
  - StaggeredGridLayoutManager: arranges the items in a two-dimensional grid, with each column slightly offset from the one before

```
1 <?xml version="1.0" encoding="utf-8"?>
2 <android.support.v7.widget.RecyclerView ...
3     android:id="@+id/list"
4     android:name="it.unito.di.educ.pdm18kotlin3v1.CourseFragment"
5     ...
6     app:layoutManager="android.support.v7.widget.LinearLayoutManager"
7     tools:listitem="@layout/fragment_course_list_element" />
```

The example: About visualization (4/5)

- In fragment_course_list_element.xml
- This file describes how to display each element of the list

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout ...
    android:orientation="vertical">

    <Space
        android:layout_width="match_parent"
        android:layout_height="@dimen/courseList_space" />

    <TextView
        android:id="@+id/courseList_name"
        style="@style/PDM18kotlin3.ListText"
        ... />
    <TextView
        android:id="@+id/courseList_code"
        style="@style/PDM18kotlin3.ListText.SecondLine"
        ... />
    ...
</LinearLayout>
```

The example: About visualization (5/5)

- In fragment_course_details.xml
- Defines how to display the information about every single course

```
<ScrollView ...>
    <android.support.constraint.ConstraintLayout  ... >

        <android.support.constraint.Barrier
            android:id="@+id/barrier"
            app:barrierDirection="right"
            app:constraint_referenced_ids="nomeTitolo,codiceTitolo,descrizioneTitolo,materialeTitolo"
            ... />

        <TextView
            android:id="@+id/nomeTitolo"
            android:text="@string/detail_titolo_nome"
            android:textAppearance="@style/PDM18kotlin3.TitleText"
            ... />

        <TextView
            android:id="@+id/nomeCorso"
            android:textAppearance="@style/PDM18kotlin3.BaseText"
            app:layout_constraintBaseline_toBaselineOf="@+id/nomeTitolo"
            app:layout_constraintStart_toEndOf="@+id/barrier"
            ... />
        ...
    </android.support.constraint.ConstraintLayout>
</ScrollView>
```

The example: About code (1/5)

- In CourseFragment.kt
- MyCourseRecyclerViewAdapter is used to customize the display of each element of the list
- The Model class provides the data source
- Define the OnListFragmentInteractionListener interface that allows interaction between the fragment and the activities that include it

```
...
    override fun onCreateView( inflater: LayoutInflater, container: ViewGroup?, savedInstanceState: Bundle? ): View {
        ...
            view.adapter = MyCourseRecyclerViewAdapter(context, myListener!!)
        ...
    }
    override fun onAttach(context: Context) {
        ...
        if (context is OnListFragmentInteractionListener) {
            myListener = context
        }
        ...
    }
    interface OnListFragmentInteractionListener {
        fun onListFragmentInteraction(courseCode: String)
    }
...
```

The example: About code (2/5)

- In MainActivity.kt
- Save and restore the selected course

```
1   ...
2   override fun onCreate(savedInstanceState: Bundle?) {
3       ...
4       myCurseCode = savedInstanceState?.getString(ARG_COURSE_SELECTED) ?:
5                     intent.extras?.getString(ARG_COURSE_SELECTED) ?:
6                     ""
7
8       if (myCurseCode != "") {
9           onListFragmentInteraction(myCurseCode)
10      }
11  }
12
13  public override fun onSaveInstanceState(outState: Bundle) {
14      super.onSaveInstanceState(outState)
15      outState.putString(ARG_COURSE_SELECTED, myCurseCode)
16  }
17  ...
```

The example: About code (3/5)

- In MainActivity.kt
- Manages the visualization of course details (fragment or intent)

```
...
override fun onListFragmentInteraction(courseCode: String) {
    myCurseCode = courseCode
    if (dettagli?.visibility==View.VISIBLE) {
        val courseDetails = supportFragmentManager.findFragmentById(R.id.courseDetails) as CourseDetailFragment?
        if ((myCurseCode!="") and (myCurseCode != courseDetails?.getShowCode())) {
            val newDetails = CourseDetailFragment.newInstance(myCurseCode)
            with(supportFragmentManager.beginTransaction()) {
                replace(R.id.dettagli, newDetails)
                setTransition(FragmentTransaction.TRANSIT_FRAGMENT_FADE)
                commit()
            }
        }
    } else {
        val intent = Intent(this, DetailActivity::class.java).apply {
            putExtra(CourseDetailFragment.ARG_COURSE_CODE, myCurseCode)
        }
        startActivity(intent)
    }
}
...
```

The example: About code (4/5)

- In CourseDetailFragment.kt
- Use arguments to manage the courseCode to show
- Check orientation to redirect to the MainActivity

```
1   ...
2   fun getShowCode() = arguments?.getString(ARG_COURSE_CODE) ?: ""
3
4   override fun onActivityCreated(savedInstanceState: Bundle?) {
5       super.onActivityCreated(savedInstanceState)
6
7       val config = resources.configuration
8       if (config.orientation == Configuration.ORIENTATION_LANDSCAPE) {
9           if (activity is DetailActivity) {
10              val intent = Intent(activity, MainActivity::class.java).apply {
11                  putExtra(MainActivity.ARG_COURSE_SELECTED, getShowCode())
12                  flags = Intent.FLAG_ACTIVITY_CLEAR_TOP
13              }
14              startActivity(intent)
15          }
16      }
17  }
18  ...
```

The example: About code (5/5)

- In CourseDetailFragment.kt
- Show the course deatails

```
...
override fun onCreateView( inflater: LayoutInflater, container: ViewGroup?, savedInstanceState: Bundle? ): View? {
    // Inflate the layout for this fragment
    val view = inflater.inflate(R.layout.fragment_course_detail, container, false)
    val code = getShowCode()

    if (code != "") {
        val course = Model.instance.getCourseDetail(code)
        with(view) {
            nomeCorso.text = course.nome
            codiceCorso.text = course.codice
            materialeCorso.text = course.materiale
            descrizioneCorso.text = course.descrizione
        }
    }
    return view
}
...
```

Outline

The example: Other files (1/2)

Other files in the project:

- Model.kt:
  - The class define how to access course information;
  - Use the Singleton Pattern;
  - Define three main methods:
    - getSize() : gets the number of courses to show;
    - getCourseElement(int pos) : gets information for the course in "pos" position;
    - getCourseDetail(String code) : for the detail about a single course;
  - In this file we also define the class Course used to maintain the details of each course;
- MyCourseRecyclerViewAdapter.kt:
  - Class used to manage the display of list elements;
  - Maintains the reference between the list element and the course;

The example: Other files (2/2)

Other files in the project:

- activity_detail.xml:
  - Define only the inclusion for the fragment_course_detail.xml;
- DetailActivity.kt:
  - Define only the wrapper activity for the CourseDetailFragment.kt;

Git example:

git clone https://<login>@gitlab2.educ.di.unito.it/ProgMob/PDM18kotlin3v1.git

Outline

The example: Kotlin vs. Java (1/4)

- In MainActivity

```java
Bundle param = getIntent().getExtras();
if(param!=null) {
    mCurCode = param.getString(ARG_COURSE_SELECTED, "");
}
if (savedInstanceState != null) {
    // Restore last state for checked position.
    mCurCode = savedInstanceState.getString(ARG_COURSE_SELECTED, "");
}

if (!"".equals(mCurCode)) {
    onListFragmentInteraction(mCurCode);
}
```

```kotlin
myCurseCode = savedInstanceState?.getString(ARG_COURSE_SELECTED) ?:
              intent.extras?.getString(ARG_COURSE_SELECTED) ?:
              ""

if (myCurseCode != "") {
    onListFragmentInteraction(myCurseCode)
}
```

The example: Kotlin vs. Java (2/4)

- In Model

```java
public Course getCourseDetail(String code) {
    for (Course course: courses) {
        if(code.equals(course.getCodice())) {
            return course;
        }
    }
    Course course = new Course();
    course.setCodice("empty");
    return course;
}
```

→

```
fun getCourseDetail_v1(code:String):Course {
    for (course in courses) {
        if (code == course.codice) {
            return course
        }
    }
    return Course("", "empty", "", "")
}
```

→

```
fun getCourseDetail_v2(code:String):Course {
    var course:Course? = courses.find
                         { it.codice==code }
    if (course != null) {  return course  }
    return Course("","empty","","")
}
```

→

```
fun getCourseDetail_v3(code:String):Course {
    return courses.find { it.codice==code } ?:
      Course("","empty","","")
}
```

→

```
fun getCourseDetail(code:String) = courses.find { it.codice==code } ?: Course("","empty","","")
```

The example: Kotlin vs. Java (3/4)

In Model.java

```java
    private static Model instance = null;

    private Model (Context context) {
        this.applicationContext = context;
        ...    }
    public static Model getInstance(Context context) {
        if(instance==null) {
            instance = new Model(context.getApplicationContext());
        }
        return instance;
    }
```

↓ in Model.kt

```kotlin
    companion object {
        @JvmStatic val instance = Model()
    }
```

↓ in MainActivity.kt

```kotlin
    init {        instance = this    }
    companion object {
        private var instance: MainActivity? = null
        fun applicationContext() : Context {
            return instance!!.applicationContext
        }
    }
```

The example: Kotlin vs. Java (4/4)

- In CourseDetailFragment

```java
View view = inflater.inflate(
        R.layout.fragment_course_detail,
        container, false);

TextView nome = (TextView) view.findViewById
                           (R.id.nomeCorso);
TextView codice = (TextView) view.findViewById
                           (R.id.codiceCorso);
TextView materiale = (TextView) view.findViewById
                           (R.id.materialeCorso);
TextView descrizione = (TextView) view.findViewById
                           (R.id.descrizioneCorso);

Model model = Model.getInstance(this.getContext());
String code = getShownCode();

if(!code.equals("")) {
    Course course = model.getCourseDetail(code);

    nome.setText(course.getNome());
    codice.setText(course.getCodice());
    materiale.setText(course.getMateriale());
    descrizione.setText(course.getDescrizione());
}
```

→

```kotlin
val view = inflater.inflate(R.layout.
      fragment_course_detail, container, false)
val code = getShowCode()

if (code != "") {
    val course = Model.instance.getCourseDetail(code)
    with(view) {
        nomeCorso.text = course.nome
        codiceCorso.text = course.codice
        materialeCorso.text = course.materiale
        descrizioneCorso.text = course.descrizione
    }
}
```