DA: Android Introduction: Fragments

Applications for mobile devices - Theory - Unit 2

Didac Florensa Cazorla

Any: 2021-2022
Curs: 102386
Institut: University of Lleida (Campus Igualada)
Titulació: Bachelor's degree in Digital Interaction and Computing Techniques (GTIDIC)

Agenda

Introduction

Fragments

Fragments lifecycle

Activity and Fragment communications

Introduction

Warm-up

- Dr. Jordi Mateo Fornés
- **Office**:
  - Office A.12 (Campus Igualada)
  - Office 3.08 (EPS Lleida)
- **Email**: jordi.mateo@udl.cat
- **Twitter**: https://twitter.com/MatForJordi
- **Github**: https://github/JordiMateoUdL

- Ph.D Dídac Florensa Cazorla
- **Office**:
  - Office A.12 (Campus Igualada)
  - Office 3.08 (EPS Lleida)
- **Email**: didac.florensa@udl.cat

Fragments

How to handle UI variations inside activities?

Exemple

Assume an app that responds to various screen sizes. The app should display a static navigation drawer and a list in a grid layout on larger screens. The app should show a bottom navigation bar and a list in a linear layout on smaller screens.

How can we handle this variation inside activities?

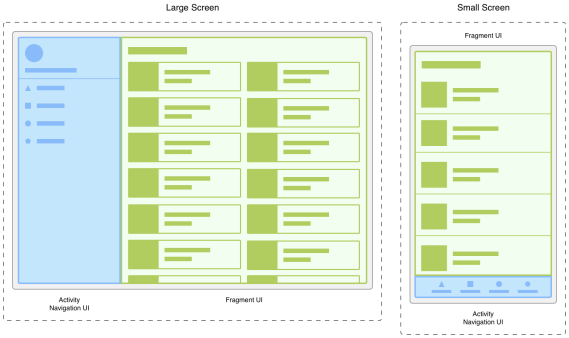

Figure 1: App that responds to various screen sizes source

What is a Fragment?

Separating the navigation elements from the content can make this process more manageable. So, Fragments allow programmers to divide the UI into pieces and modify **Activities** UI at runtime. The **Activity** is then responsible for displaying the correct navigation UI while the **fragment** shows the list with the proper layout.

Definition

A **Fragment** represents a reusable portion of your app's UI. A **fragment** defines and manages its layout, has its lifecycle and can handle its input events. **Fragment** cannot live on their own–they must be hosted by activity or another **fragment**.

Caution: In this context, a fragment only needs logic to manage its own UI. You should avoid depending on or manipulating one fragment from another.

Date picker: Example

Exemple

Imagine a form where we ask for the name, surname and birthday. This form is held by an Activity, with 2 EditText and 1 DatePicker. When the user clicks on the DataPicker component Date picker fragment appears on top of Activity.

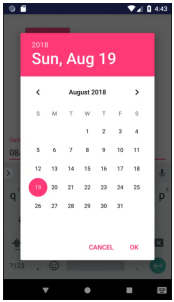

Figure 2: Date pickers: Extend DialogFragment (Fragment subclass)

Benefits of using fragments

- **UI variations**: Represent sections of a layout for different screen sizes
- **Reusability**: Reuse a Fragment in more than one Activity.

Exemple

Imagine you need a Dialog in many places of your app. Instead of copying and pasting the code (repetition), you can create a fragment, define the same dialog inside that and use that common fragment everywhere.

- Add or remove dynamically as needed
- Integrate a mini-UI within an Activity
- Retain data instances after a configuration change

How to create a fragment?

1. MyFragment needs to be a subclass of *Fragment*.
   - DialogFragment: Floating Dialogs (i.e: date and time pickers).
   - ListFragment: List of items managed by an adapter.
   - PreferenceFragment: Hierarchy of Preference objects (useful for Settings)
2. Create a layout to associate with the *Fragment*.
3. Add the *Fragment* to a host activity.
   - Static in the layout
   - Dynamic using transactions.

```
public class MyFragment extends Fragment {
    public MyFragment() {
    }
    ...
}
```

Or just, **File > New > Fragment > {X} ~Fragment (Blank)** and the check the Create layout XML option for layout.

onCreateView()

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

- *View*: Represents the root of fragment layout. ⇒ Fragment layout is inserted into container *ViewGroup* in **Activity** layout. This way *LayoutInflater* inflates layout and returns **View** that represents the layout root to the Activity.
- *Bundle* **savedInstanceState** saves previous Fragment instance.

Note: The third argument of inflate is a Boolean that represents Whether layout should be attached to parent or not. Should be false. If adding Fragment in code, don't pass true (creates a redundant *ViewGroup*).

Add a fragment to an Activity

We can add statically the fragment in Activity layout (xml), visible for entire Activity lifecycle:

```
<fragment
        android:name="cat.udl.tidic.amd.myapplication.MyFragment"
        android:id="@+id/simple_fragment"
        android:layout_weight="2"
        android:layout_width="0dp"
        android:layout_height="match_parent" />
```

The `android:name` attribute specifies the class name of the Fragment to instantiate. When the activity's layout is inflated, the specified fragment is instantiated, onInflate() is called on the newly instantiated fragment, and a FragmentTransaction is created to add the fragment to the FragmentManager.

Add a fragment to an Activity

We can **add or remove** dynamically as needed during Activity lifecycle using Fragment transactions.

```
<!-- res/layout/example_activity.xml -->
<androidx.fragment.app.FragmentContainerView
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/fragment_container_view"
    android:layout_width="match_parent"
    android:layout_height="match_parent" />
```

FragmentContainerView is a customized Layout designed specifically for Fragments. It extends FrameLayout to handle Fragment Transactions reliably, and it also has additional features to coordinate with fragment behaviour.

Unlike the XML approach, the `android:name` attribute is not used on the FragmentContainerView. Instead, a FragmentTransaction is used to instantiate a fragment and add it to the activity's layout.

Add a fragment to an Activity

Fragment operations are wrapped into a transaction:

- Start transaction with beginTransaction()
- Do all Fragment operations:
  - Add a Fragment using add()
  - Remove a Fragment using remove()
  - Replace a Fragment with another using replace()
  - Hide and show a Fragment using hide() and show()
  - Add Fragment transaction to back stack using addToBackStack(null)
- End transaction with commit()

Exemple

In your *Activity*, you can get an instance of the **FragmentManager**, which can be used to create a *FragmentTransaction*. Then, you can instantiate your fragment within your activity's onCreate() method using *FragmentTransaction.add()*, passing in the ViewGroup ID of the container.

Add a fragment to an Activity

```
public class MyActivity extends AppCompatActivity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        if (savedInstanceState == null) {
            getSupportFragmentManager().beginTransaction()
                .setReorderingAllowed(true)
                .add(R.id.fragment_container_view,
                MyFragment.class, null).commit();
        }}}
```

In Activity, get instance of FragmentManager with getSupportFragmentManager(): FragmentManager fragmentManager = getSupportFragmentManager();

Use the Support Library version—getSupportFragmentManager() rather than getFragmentManager()—for compatibility with earlier Android versions

Question 1: Analyse, justify and answer

What is doing this code?

```
FragmentTransaction fragmentTransaction =
           fragmentManager.beginTransaction();
fragmentTransaction
           .add(R.id.fragment_container_view,
                MyFragment.class, null)
           .addToBackStack(null)
           .commit();
```

Answer 1: Analyse, justify and answer

What is doing this code?

```
FragmentTransaction fragmentTransaction =
           fragmentManager.beginTransaction();
fragmentTransaction
           .add(R.id.fragment_container_view,
                MyFragment.class, null)
           .addToBackStack(null)
           .commit();
```

This code creates a fragment and adds this fragment to Fragment activity. With *addToBackStack*, users can press the Back button to return to the previous Fragment state.

Fragments lifecycle

What is the lifecycle for a fragment?

Each Fragment instance has its lifecycle. When a user navigates and interacts with your app, your fragments transition through various stages in their lifecycle as they are added, removed, and enter or exit the screen.

⇒ Similar to the Activities lifecycle

Exemple

You might display the device's location on the screen using a lifecycle-aware component. This component could automatically start listening when the fragment becomes active and stop when the fragment moves to an inactive state.

Lyfecycle

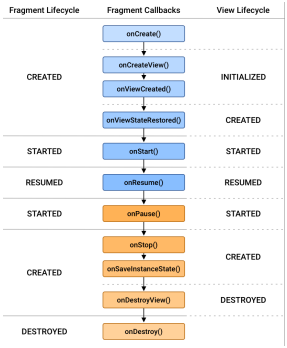

Figure 3: Fragment Lifecycle states and their relation both the fragment's lifecycle callbacks and the fragment's view Lifecycle from source

Caution BEFORE or AFTER API 28

The ordering of **onStop()** callback and **onSaveInstanceState()** differs based on API level.

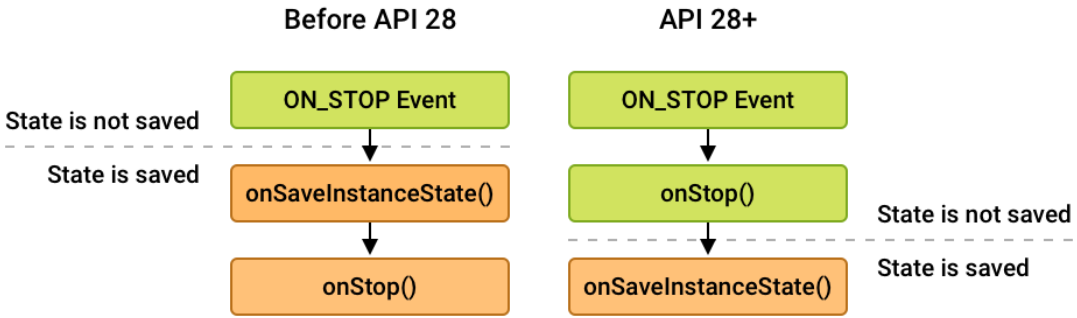

Figure 4: Calling order differences for onStop() and onSaveInstanceState()

Activity and Fragment communications

Communitacion

The Fragment library provides two options for communication: a shared *ViewModel* and the *Fragment Result API*. The recommended option depends on the use case. To share persistent data with any custom APIs, you should use a ViewModel. For a one-time result with data that can be placed in a Bundle, you should use the Fragment Result API.

Send data to a Fragment

```
public static SimpleFragment newInstance(int choice) {
SimpleFragment fragment = new SimpleFragment();
Bundle arguments = new Bundle();
arguments.putInt("choice", choice);
fragment.setArguments(arguments);
return fragment;
}
```

or

```
SimpleFragment fragment =
          SimpleFragment.newInstance(mRadioButtonChoice);
```

Receive data in the Fragment

Before drawing **Fragment View**, get the arguments from Bundle using *getArguments()*.Use it **onCreate()** or **onCreateView()** callback:

```
// onCreate() or onCreateView()
if (getArguments().containsKey(CHOICE)) {
    mRadioButtonChoice = getArguments().getInt(CHOICE);
    // ...
}
```

Retrieve data from Fragment: Fragment

Step 1: Define interface (such as a listener) with callback method(s)

```
interface OnFragmentInteractionListener {
    void onRadioButtonChoice(int choice);
}
```

Retrieve data from Fragment: Fragment

Step 2: Override onAttach() to retrieve interface implementation (check if the Activity implements the interface)

```
@Override
public void onAttach(Context context) {
    super.onAttach(context);
    if (context instanceof OnFragmentInteractionListener) {
        mListener = (OnFragmentInteractionListener) context;
    } else {
        // ...
```

Retrieve data from Fragment: Fragment

Step 3: Call interface method to pass data as parameter

```
public void onCheckedChanged(RadioGroup group,
int checkedId) {
    // ...
    switch (index) {
        case YES: // User chose "Yes."
            mListener.onRadioButtonChoice(YES);
            break;
        case NO: // User chose "No."
            mListener.onRadioButtonChoice(NO);
            break;
        // ...
    }
```

Retrieve data from Fragment: Activity

Step 4: Activity must implement the interface defined in the Fragment

```
public class MainActivity extends AppCompatActivity
 implements SimpleFragment.OnFragmentInteractionListener {
```

Step 5: Activity can then use onRadioButtonChoice() callback:

```
@Override
public void onRadioButtonChoice(int choice) {
    mRadioButtonChoice = choice;
    // Use mRadioButtonChoice in Activity
    // ...
}
```

Fragment B sends data to Fragment A

Exemple

Imagine a fragment that reads QR codes, passing the data back to a previous fragment.

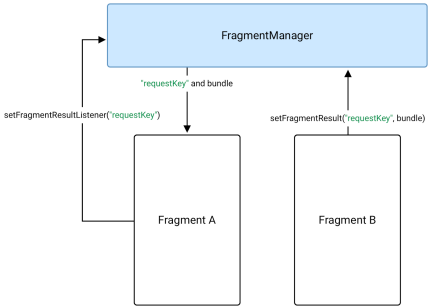

Figure 5: Fragment B sends data to fragment A using a FragmentManager from source

Fragment B sends data to Fragment A: Fragment A

```
@Override
public void onCreate(@Nullable Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    getParentFragmentManager().setFragmentResultListener(
        "requestKey", this, new FragmentResultListener() {
        @Override
        public void onFragmentResult(@NonNull String requestKey,
        @NonNull Bundle bundle) {
            String result = bundle.getString("bundleKey");t
        }
    });
}
```

Fragment B sends results to Fragment A: Fragment B

```
button.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        Bundle result = new Bundle();
        result.putString("bundleKey", "result");
        getParentFragmentManager().setFragmentResult("requestKey", result);
    }
});
```

Fragment B sends results to Host Activity

```
class MainActivity extends AppCompatActivity {
    @Override
    public void onCreate(@Nullable Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        getSupportFragmentManager().setFragmentResultListener(
            "requestKey", this, new FragmentResultListener() {
            @Override
            public void onFragmentResult(@NonNull String requestKey,
            @NonNull Bundle bundle) {
                String result = bundle.getString("bundleKey");
            }
        });
    }
}
```

That's all

QUESTIONS?

About me

www — jordimateofornes.com
github — github.com/JordiMateo
twitter — @MatForJordi
gdc — Distributed computation group