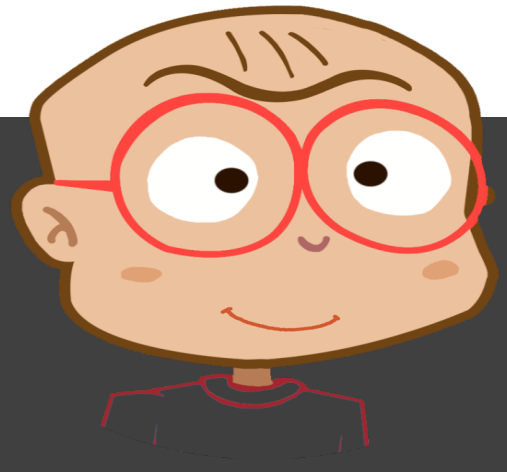
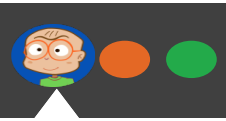


Fragments

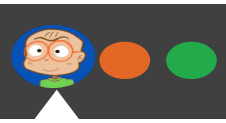
START



Learning Goals

After the course, attendees will be able to:

- Understanding about Fragments and its mechanism in Android
- Know about several practices in implementing Fragments



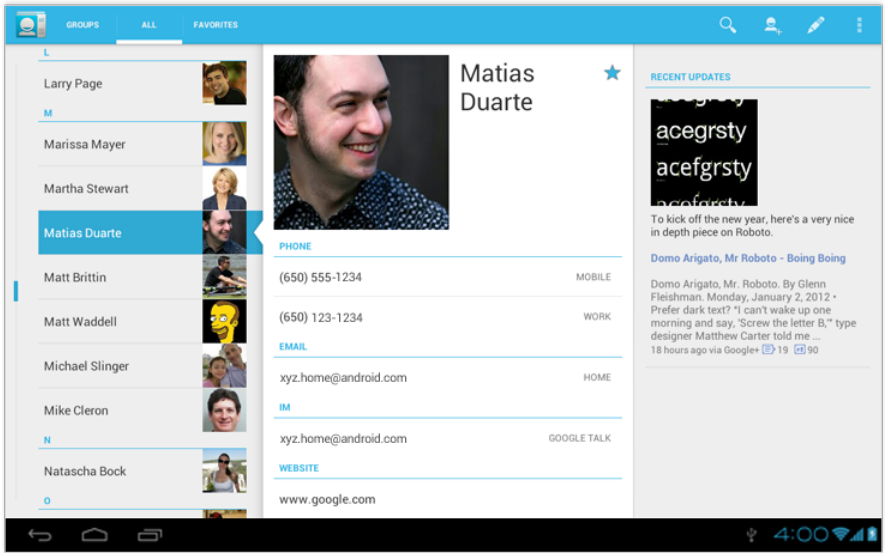
Agenda

1. Overview
2. When we need to use Fragments?
3. How to manage Fragments?
4. How to work with Fragments?
5. Practices

Overview

Overview

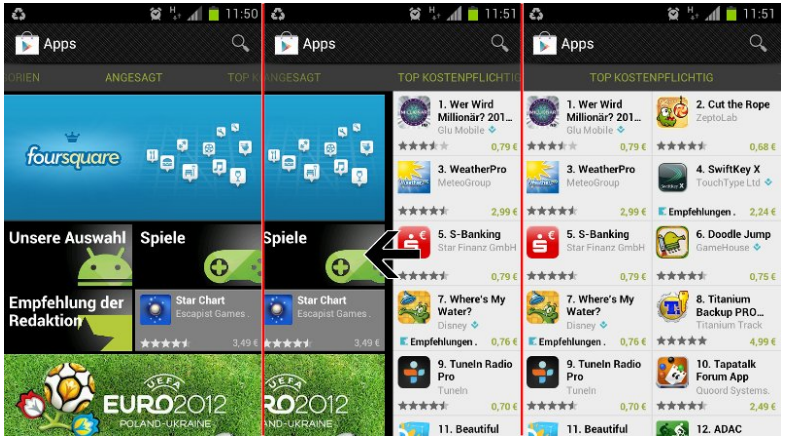
- Introduced with Honeycomb (API 11)
- Modular section of an activity / Part of a user interface*
- More dynamic and flexible UI on large screens
- Reusability across activities
- Supported via Compatibility Package
- Embedded in activities
- Lives in a ViewGroup inside the activity's view hierarchy
- Its own layout and behavior with its own life-cycle

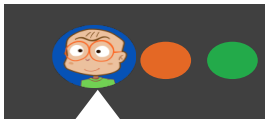


When we need to use Fragments?

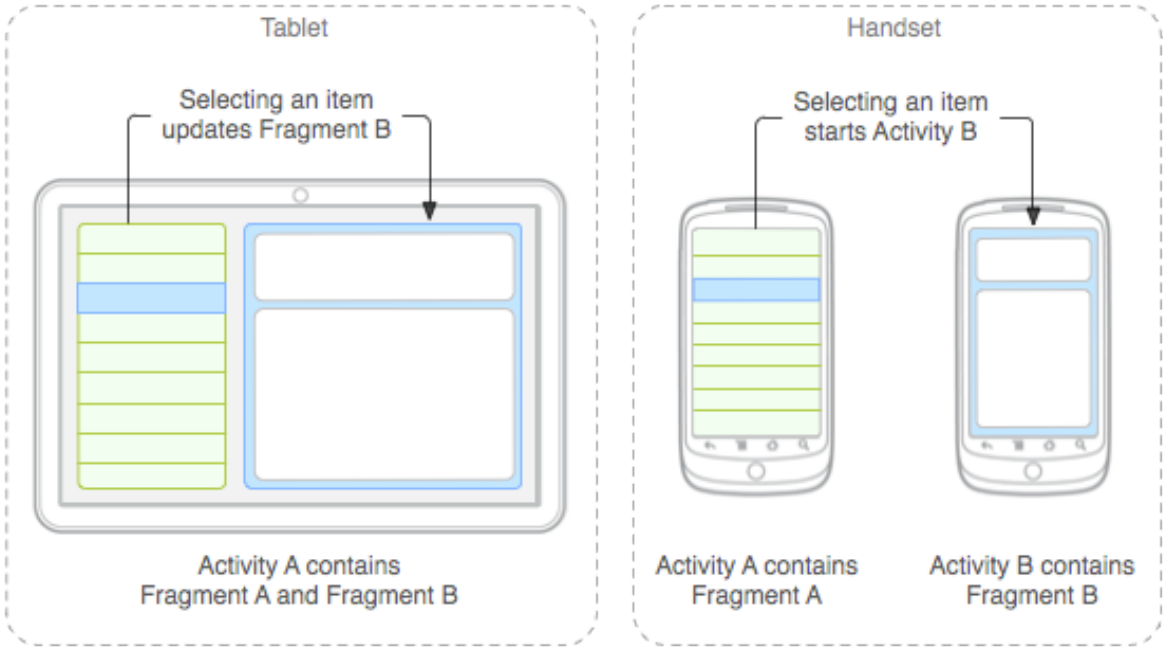
When we need to use Fragments?

- Large screens - multi pane, flexible UI
- Combined with other UI widgets, ActionBar, ViewPager, NavigationDrawer, … to create a beautiful and modern application.
- More flexible, smooth, compact app with less activities; multiple fragments in a single activity

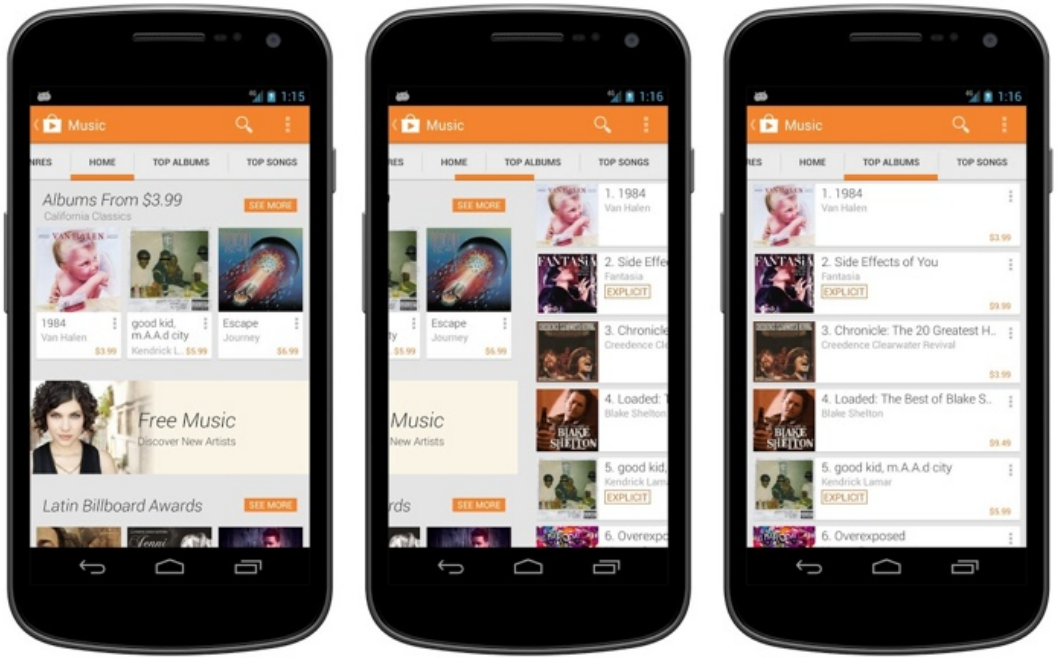


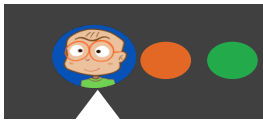


Example, Tablets vs Smartphone

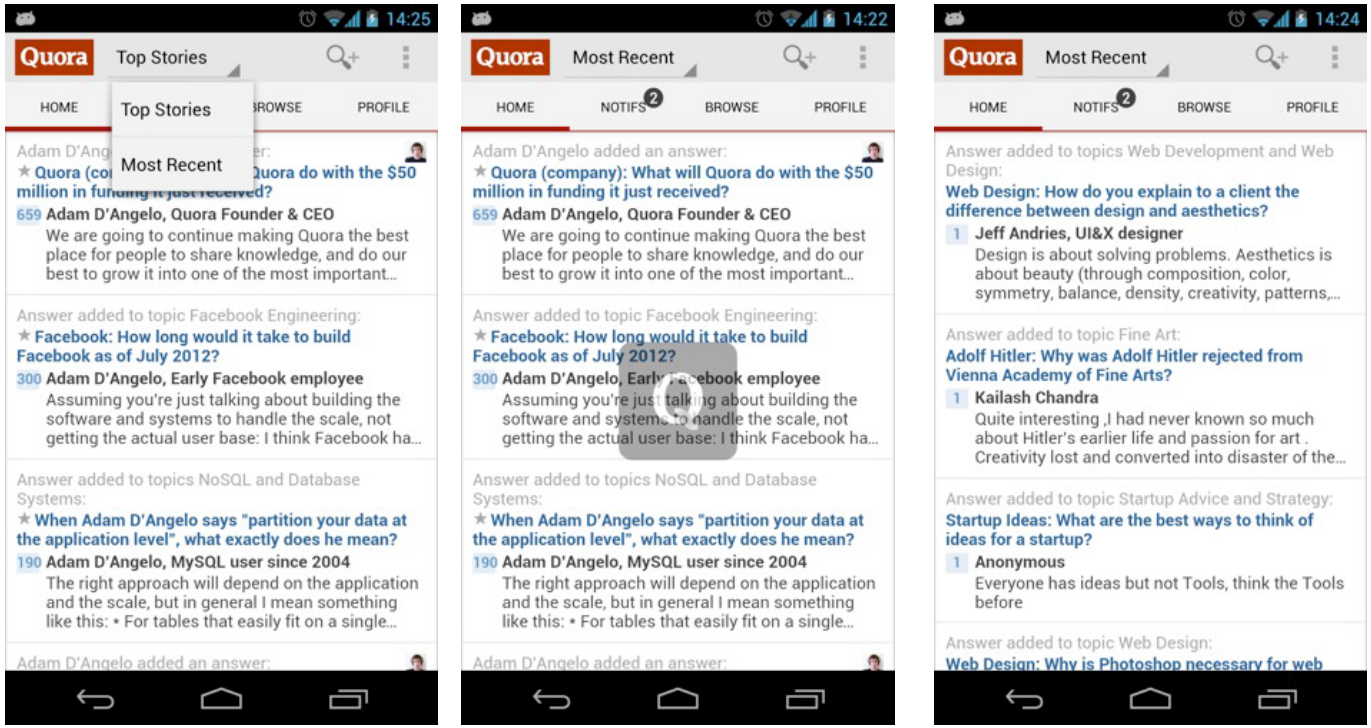


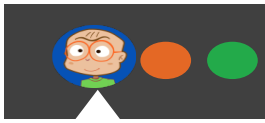
Example, Tabs and ViewPagers



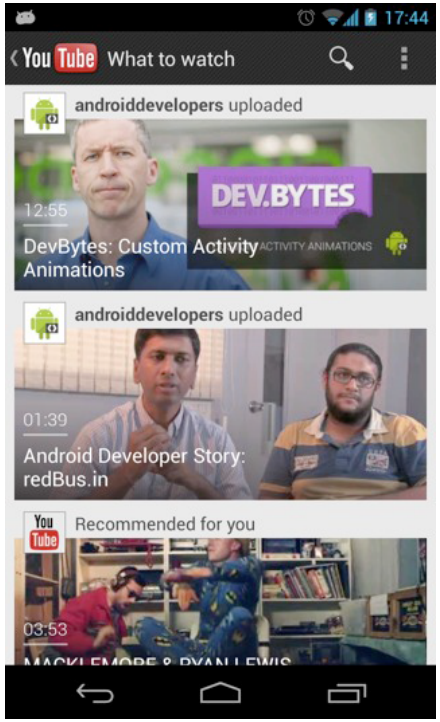


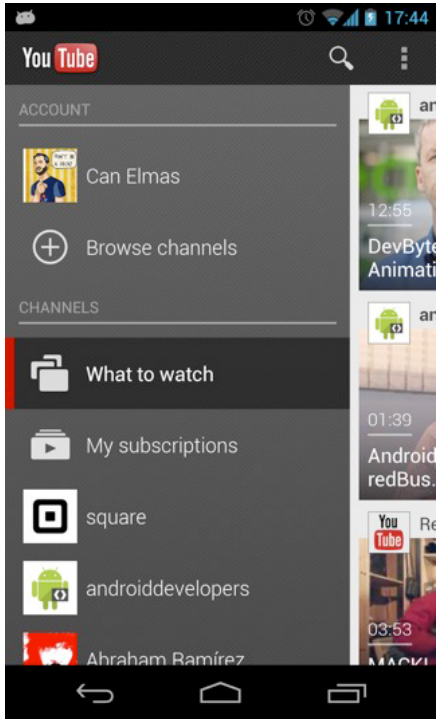
Example, ActionBar Navigation

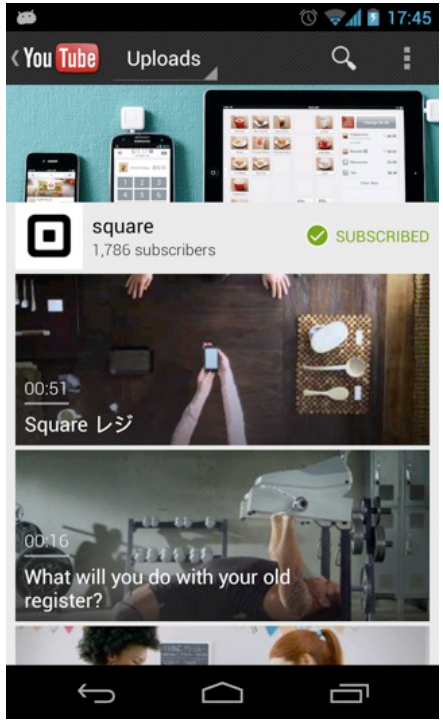




Example, Navigation Drawer



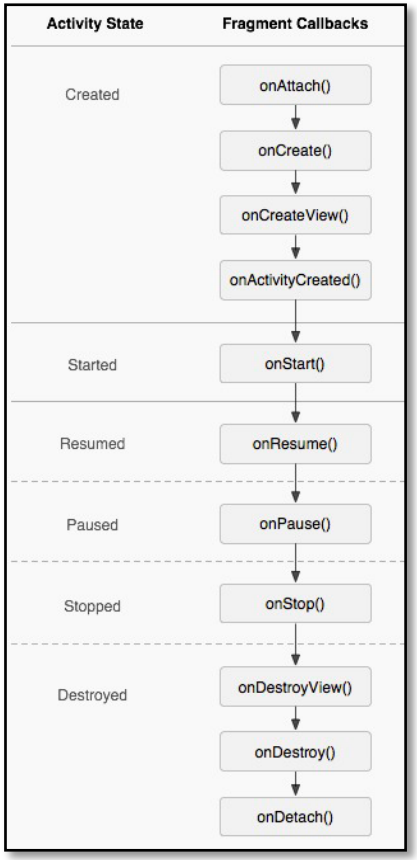




How to manage Fragments?

Fragment's Lifecycle vs Activity Lifecycle

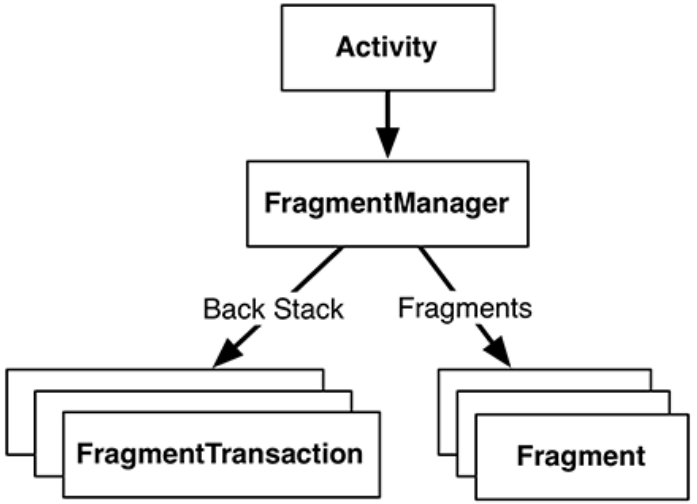
- **onCreate(Bundle)** called once the fragment is associated with its activity. Activity is still in the process of being created. Do not depend on activity's layout.
- **onCreateView(LayoutInflater, ViewGroup, Bundle)** time to instantiate for the fragment's user interface (optional, return null for non-graphical fragments)
- **onActivityCreated(Bundle)** called once the activity is created and fragment's view is instantiated. Do final initializations (context dependent instantiations, retrieving views, adding listeners etc.)
- **onDestroyView()** Called when the view has been detached from the fragment. Next time the fragment needs to be displayed, a new view will be created.



Introducing FragmentManager

- ***FragmentManager*** used to interact with all fragments which placed inside Activity:
  - *Activity.getFragmentManager()*
  - *android.support.v4.app.FragmentActivity.getSupportFragmentManager()*

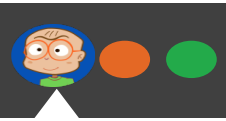
- ***FragmentTransaction*** components used to perform various fragments operation such as: add, delete, show, hide, … at run-time.



Adding Fragments in a static way

- Define a *<fragment>* element in the activity's layout XML.
- Useful if the fragment is consistent in the layout and not changed in run-time.
- Limits run-time fragment operations; can't remove

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
              android:layout_width="fill_parent"
              android:layout_height="fill_parent"
              android:orientation="vertical">
    <fragment class="com.example.unitedstatescities.ItemDescriptionFragment"
              android:id="@+id/itemDescriptionFragment"
              android:layout_width="match_parent"
              android:layout_height="0px"
              android:layout_weight="1" />
    <fragment class="com.example.unitedstatescities.ItemListFragment"
              android:id="@+id/itemListFragment"
              android:layout_width="match_parent"
              android:layout_height="0px"
              android:layout_weight="1" />
</LinearLayout>
```

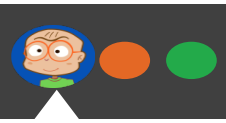


Adding Fragments dynamically

```
@Override
public void onClick(View v) { // Tap on a button to change fragment
    FragmentManager fragmentManager = getFragmentManager();
    FragmentTransaction fragmentTransaction = fragmentManager.beginTransaction();
    HelloFragment hello = new HelloFragment();
    fragmentTransaction.add(R.id.fragment_container, hello, "HELLO");
    fragmentTransaction.commit();
}
```

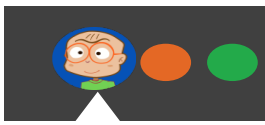
Advantages:

- Avoid tight coupling between activities and fragments.
- Switching between fragments easily.

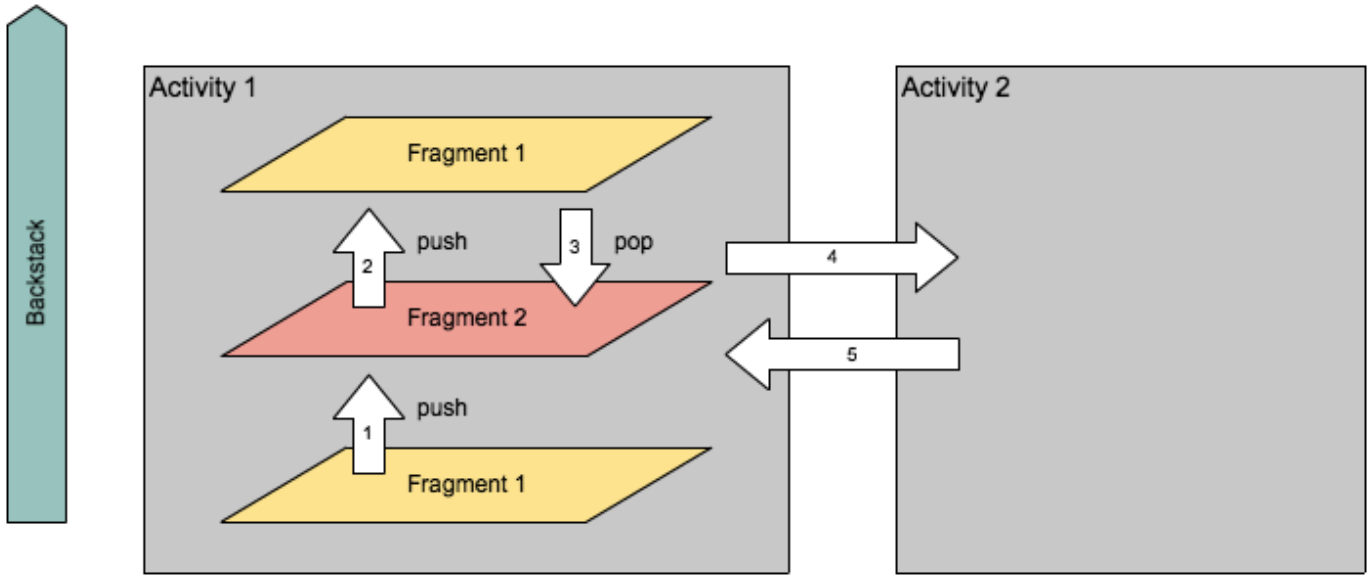


Fragments operations

- **add** - Add a fragment to the activity
- **remove** - Remove an existing fragment; its view is also removed
- **replace** - Remove one fragment from the activity and add another
- **hide** - Hides an existing fragment; its view is hidden
- **show** - Show previously hidden fragment



Fragment back stack



Fragment back stack

- Similar to activity back stack
- Allow user to navigate backward
- Ability to save fragment transaction onto back stack
- Managed by activity on back button press
- Activity destroyed once all fragment transactions removed from the back stack and back button is pressed again

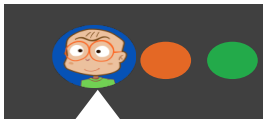
Fragment back stack

FragmentTransaction.addToBackStack(String)

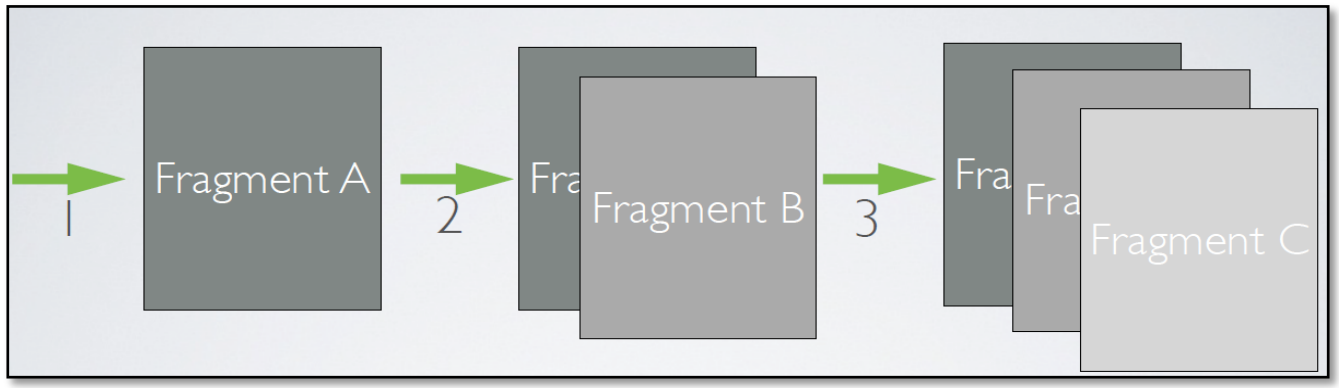
- null or optional identifier name for the back stack state (note, state is the current stack content).
- Useful for returning to a given state by calling

FragmentManager's *popBackStack* methods

- *popBackStack()* : Pop the top state off the back stack
- *popBackStack(String name, int flags)* : pop all state which have matching *name* state.
- If POP_BACK_STACK_INCLUSIVE flag is set, clear all states in back stack!

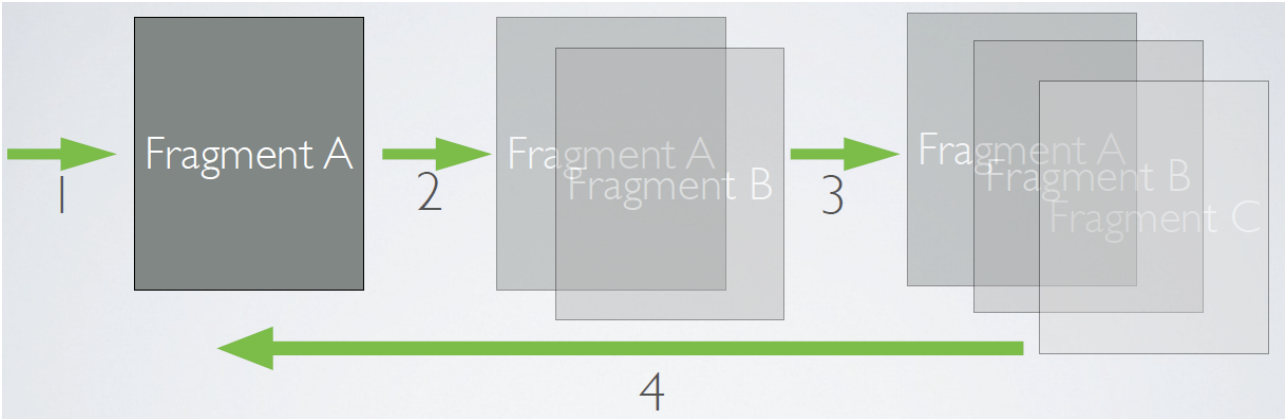


Example, Fragment back stack



- `ft.add(R.id.fragment_container, fragmentA);`
  - `ft.addToBackStack("fragStackA")`
- `ft.add(R.id.fragment_container, fragmentB);`
- `ft.add(R.id.fragment_container, fragmentC);`

Example, Fragment back stack




```
FragmentManager fm = getSupportFragmentManager();
fm.popBackStack("fragStackA", 0);
```

Exit Course

THANK YOU

You have completed “**Lesson 7" course.**
Click EXIT button to exit course and discover
the next Lecture “**Lesson 8".**

EXIT