UNIVERSITY of WASHINGTON

Android 250 - Lecture 4
Fragments

Margaret Maynard-Reid
April 20, 2015

Agenda

- Homework 1 solution
- Android Support Library
- Fragment: basics, back stack, Lifecycle
- App Navigation

SampleCode

- SampleFragment
- SampleViewPager
- SampleMasterDetail*

Homework 1

- Go over homework 1 solution

Android Stories

- Google Updates Material Design Spec With Dedicated FAB Section, Updates On Typography, Cards, And More
- The Year Of Mobile Payments
- Cyanogen and Microsoft Team Up to Integrate Microsoft's Services
- Baidu previews Android-based smartwatch OS for Chinese market

Review from Last Week

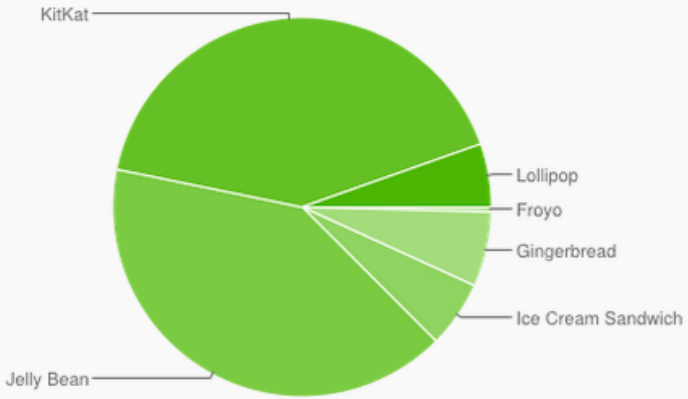
- List a few examples of resource qualifiers
- What do you specify in AndroidManifest.xml if your app uses camera?
- Why do we use Styles & Themes in Android?
- Difference between a Style & a Theme?

Break

Android Platform Versions

Version	Codename	API	Distribution
2.2	Froyo	8	0.4%
2.3.3 - 2.3.7	Gingerbread	10	6.4%
4.0.3 - 4.0.4	Ice Cream Sandwich	15	5.7%
4.1.x	Jelly Bean	16	16.5%
4.2.x		17	18.6%
4.3		18	5.6%
4.4	KitKat	19	41.4%
5.0	Lollipop	21	5.0%
5.1		22	0.4%

Data collected during a 7-day period ending on April 6, 2015.
Any versions with less than 0.1% distribution are not shown.



Source: http://developer.android.com/about/dashboards/index.html

Android Support Library

- Provides backward-compatibility, i.e.
  - Fragments
  - ActionBar
  - Loaders
  - Material Design
- Access features ***not available*** in the Framework, i.e.
  - ViewPager
  - DrawerLayout
  - NotificationCompat
  - LocalBroadcastManager

Support Library Versions

Backward Compatible up to the version in the package name

- android.support.v4.* (v7, v8, v13)
- use the versioned library and becomes compatible with that API level and on:
  - v4 = Android Donut 1.6+
  - v7 = Android Éclair 2.1+
  - v8 = Android Froyo 2.2+
  - v13 = Android Honeycomb 3.2+

Features Introduced

V4

- Fragments
- Loaders
- LocalBroadcastManager
- DrawerLayout, SlidingPaneLayout
- ViewPager, PagerTabStrip,
- PagerTitleStrip
- Notification Compatibility
- Accessibility Compatibility

V8

- Renderscript Support

V7

- ActionBar Compatibility
- GridLayout
- CardView
- RecyclerView
- Palette
- MediaRouter

V13

- Backwards compatible Fragment handling via FragmentCompat, etc.

V17

- Leanback (support for TV)

http://developer.android.com/tools/support-library/features.html

Support Library Setup

- Download the Support Library
  - SDK Manager, under *Extras*
  - “Android Support Repository” (for Android Studio)
- Android Studio: Add dependency in the app ***build.gradle*** file:

```
dependencies {
    ...
    compile "com.android.support:support-v4:22.0.+"
}
```

*** Note for Eclipse you need to import it as a library project for setting up the support library with resources (i.e. v7 appcompat)

Demo

- SDK Manager - download Android Support Repository
- Android Studio - set up the Support Library

Tip: Go One Way

- If you need to use the Support Library keep it consistent and use it for everything
- Do not mix support library classes and non-support library classes

How do I use it?

Where things get confusing...

- Need to reference the correct package
  - android.support.v4.app.Fragment ← The Support Library version of Fragment
  - android.app.Fragment ← The Framework version of Fragment
- They are not cross-compatible

Fragments

What is a fragment?

- a component of UI in an activity
- must exist as part of an activity

Why do we need it?

- introduced in Honeycomb to better support tablet and phone UI
- dynamic and flexible UI

Fragment History

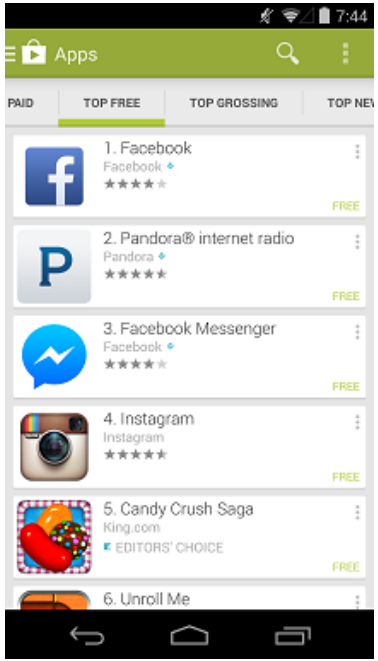
Fragments were added in Honeycomb (API 11) to allow tablets to more flexibly utilize their display space and cover the gap between “full-screen” phone applications and tablet applications

Fragments are Used Everywhere!

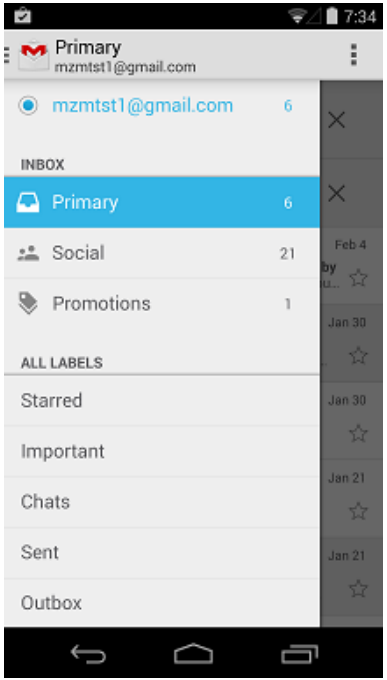
- Master/Detail list
- Swipe View w/ Tabs
- ViewPager
- Navigation Drawer

Fragment UI Examples

ViewPager



Navigation Drawer



Tasks

- Collection of active Applications
- There are flags that you can set in AndroidManifest.xml to control how the Task behaves with respect to its stack of Activities
  - alwaysRetainTaskState
  - allowTaskReparenting
  - clearTaskOnLaunch
  - finishOnTaskLaunch
  - launchMode
  - taskAffinity

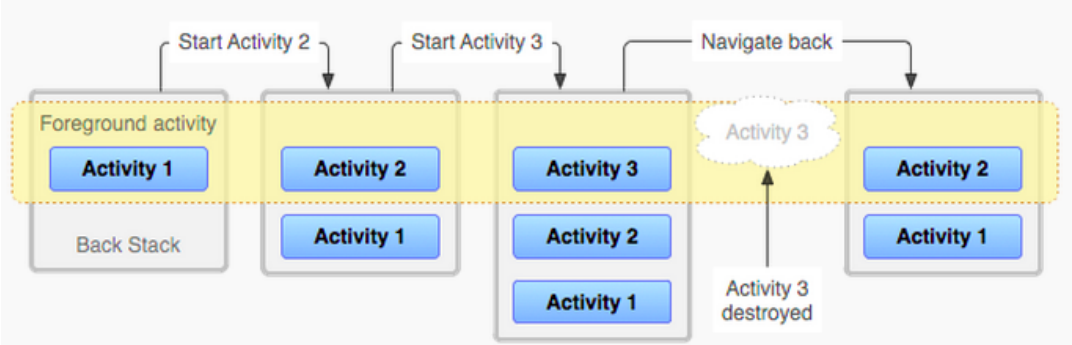
http://developer.android.com/guide/topics/manifest/activity-element.html

Activity Backstack

- A Task is a collection of Activities
- In the Task, Activities are organized onto to a Last-In/First-Out (LIFO) stack
- Each new Activity started as part of a Task, and added to the top of the stack
- Going Back removes an Activity from the stack
- Hence, this is called the "Back Stack"

Activity Back Stack

- User launches app from device Home screen
- Activity 1 is launched by default, as part of a new Task
- Navigate from Activity 1→ Activity 2 → Activity 3
- On Activity 3, press Back button
- Activity 3 is popped off the Back Stack and destroyed, Activity 2 returns
- On Activity 2, press Back button
- Activity 2 is popped off the Back Stack and destroyed, Activity 1 returns
- On Activity 1, press Back button
- Activity is popped off the Back Stack and destroyed, User leaves Task (app exists)



Fragment Back Stack

- A record of Fragment transactions
- System Back button only keeps track of Activities, not Fragments
- Unlike the Activity Back Stack, the intention to add/remove on the Fragment Back Stack is done explicitly
  - *FragmentTransaction.addToBackStack()*
  - *FragmentManager.popBackStack()*
- Similar to the Activity stack
  - *Activity.moveTaskToBack()*

Fragment Lifecycle

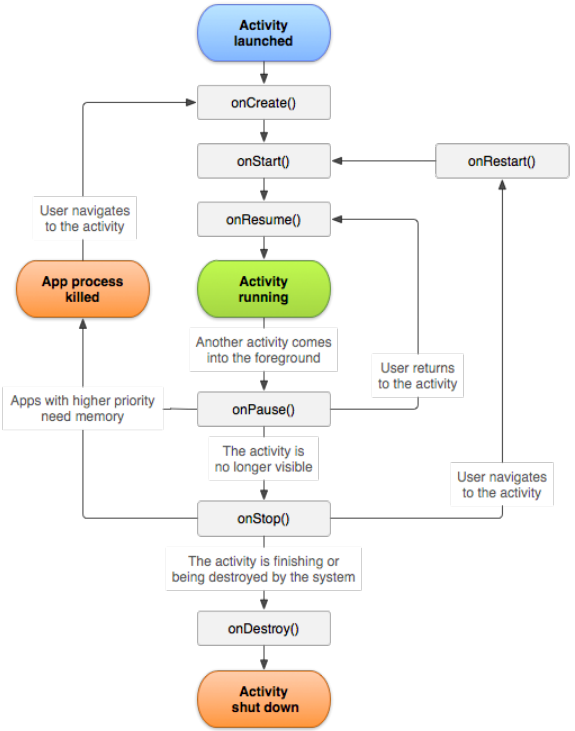
Fragment supports most of the same Activity lifecycle methods:

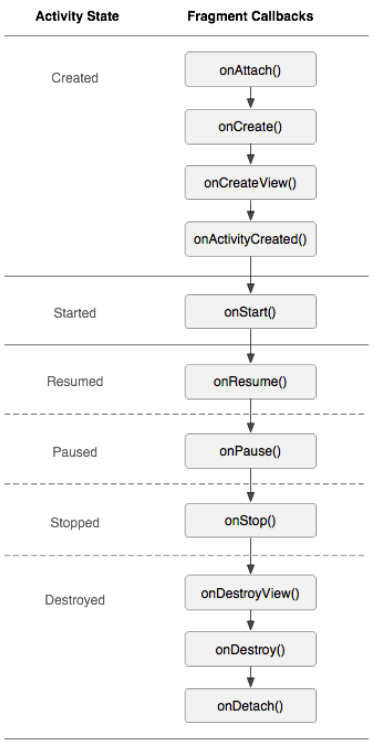
- onCreate(), onStart(), onResume(), onPause(), onStop(), onDestroy()

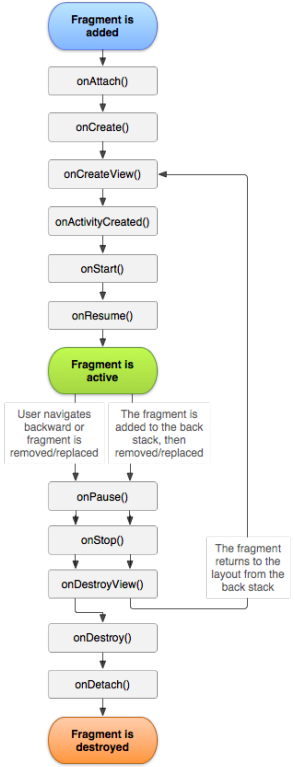
In addition, Fragment has these methods:

- onAttach() - fragment is attached to an Activity
- onCreateView() - fragment draws UI
- onActivityCreated() - hosting activity's onCreate() has completed
- onDestroyView() - clean up UI
- onDetach() - fragment is detached from hosting Activity

Activity vs. Fragment Lifecycle







Android Fragment Key Classes

- **Fragment -** base class for creating a Fragment
- **FragmentManager -** interface that allows you to interact with the Fragment
- **FragmentTransaction -** API that performs Fragment operations: add, remove and replace etc.

Creating a Fragment

- Create a Fragment by extending from Fragment class,
- or extend from a Fragment Subclass:
  - DialogFragment
  - ListFragment
  - PreferenceFragment
  - WebViewFragment
  - MapFragment

Creating a Fragment

Most apps will implement at least these methods:

- onCreate() ← initiate the fragment
- **onCreateView()** ← draws Fragment UI
- onPause()

Adding Fragment to Activity

- Fragment can’t exist on its own, and must be a part of an activity.
- Two ways to add a fragment to an activity:
  1. by XML layout statically
  2. by code programmatically

Add a Fragment Statically

Use this approach if the Fragment is fairly static

Add fragment to Activity layout using the “fragment” tag

The ***android:name*** attribute specifies the **Fragment** class to instantiate in the **activity l**ayout:

```
<?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.ArticleListFragment"
         android:id="@+id/list"
         android:layout_width="wrap_content"
         android:layout_height="match_parent" />
</LinearLayout>
```

Add a Fragment By Code

Use this approach if you would like to **dynamically add/remove/replace fragments to your activity** at runtime:

Step 1. Define container(s) in the activity layout for the fragment(s):

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
   android:orientation="horizontal"
   android:layout_width="match_parent"
   android:layout_height="match_parent">
   <Framelayout
         android:id="@+id/fragment_container"
         android:layout_weight="2"
         android:layout_width="0dp"
         android:layout_height="match_parent" />
</LinearLayout>
```

Add a Fragment By Code

Step 2. In the activity, use FragmentManager and FragmentTransaction to add a fragment

```
FragmentManager fragmentManager = getFragmentManager();
FragmentTransaction fragmentTransaction = fragmentManager.beginTransaction();
ExampleFragment fragment = new ExampleFragment();
fragmentTransaction.add(R.id.fragment_container, fragment);
fragmentTransaction.commit();
```

Demo

- Walk through SampleFragment
  - review how to create a fragment
  - review how to add a fragment

Break

FragmentManager

The managing interface for using Fragments within an Activity

- Used to transact and keep track of Fragments

FragmentTransaction

A set of Fragment actions to perform at the same time

- Get a FragmentTransaction - *FragmentManager.beginTransaction()*
- add(), remove(), replace().. FragmentTransaction
- Finalize any changes with a call to commit() FragmentTransaction

Fragment Transactions

- add() – Add a Fragment to the FragmentManager.

  onAttach() > onCreate() > onCreateView() > onStart() > onResume()
- remove() – Remove a Fragment from the FragmentManager.

  onPause() > onStop() > onDestroyView() > **onDestroy() > onDetach()**
- attach() – Attach a Fragment from FragmentManager. Put a detached view back into the Activity.

  onCreateView() > onStart() > onResume
- detach() – Detach a Fragment from FragmentManager. Put the Fragment on the back stack.

  onPause() > onStop > onDestroyView()
- show() | hide() – Show or Hide a Fragment. Basically call View.setVisibility() on the Fragment.
- replace() – Remove all Fragments from the target view and add a new Fragment.
- addToBackStack() - Add transaction to backstack so that Back button returns previous fragment.

Adding Fragments

```
Fragment newFragment = new ExampleFragment();
FragmentTransaction transaction = getFragmentManager().beginTransaction();
transaction.add(R.id.fragmentRoot, newfragment);
transaction.addToBackStack(null);
transaction.commit();
```

Removing Fragments

```
Fragment myFragment = getFragmentManager().findFragmentByTag
("MyFragment");
FragmentTransaction transaction = getFragmentManager().beginTransaction();
transaction.remove(myFragment);
transaction.commit();
```

Replacing Fragments

```
Fragment newFragment = new ExampleFragment();
FragmentTransaction transaction = getFragmentManager().beginTransaction();
transaction.remove(someOtherFragment);
transaction.replace(R.id.fragmentContainer, fragment);
transaction.commit();
```

Nested Fragments

- Scenario: a Fragment inside of another Fragment
- As of Android 4.2, you can now officially nest fragments within one another
- Use getChildFragmentManager() instead of getFragmentManager()

Android Navigation

Understanding how users can navigate your application is crucial

- A major design and development challenge for smaller form factors

http://developer.android.com/design/patterns/navigation.html

Navigation

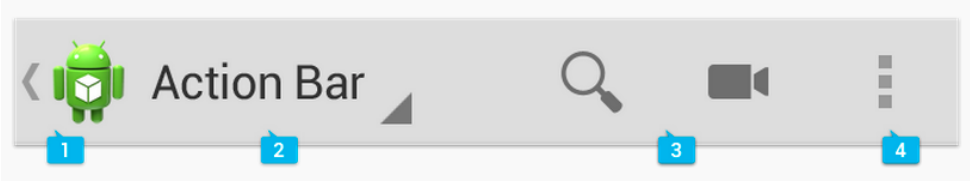
A few UI patterns that provide navigation:

- ActionBar
- ViewPager
- NavigationDrawer
- Multi-pane Layout
  - ie. ListView to a Detail screen

Action Bar

- Introduced in Android 3.0 (API 11)
- A dedicated real estate for your brand
- Provide user with navigation and screen switching
- For backwards compatibility, add support library with resources (v7 appcompat library).

Action Bar



1. App Icon (No longer visible by default starting 5.0)
2. View Control (optional)
3. Action Buttons - display most important actions
4. Action Overflow - less often used actions

Up vs. Back

Note an Android app doesn't have a single point of entry!

Up & Back can behave the same, but not always!

- **Up**: app-based hierarchical parent-child relationships
- **Back**: system Back button
  - navigate through the history of screens
  - dismiss floating windows
  - dismiss contextual ActionBar
  - hides the onscreen keyboard

ViewPager

- Use a ViewPager to implement Swipe Views or Tabs
- Provide lateral navigation between **sibling** screens horizontally
- Use a horizontal swipe gesture

Demo

- Walk through SampleViewPage

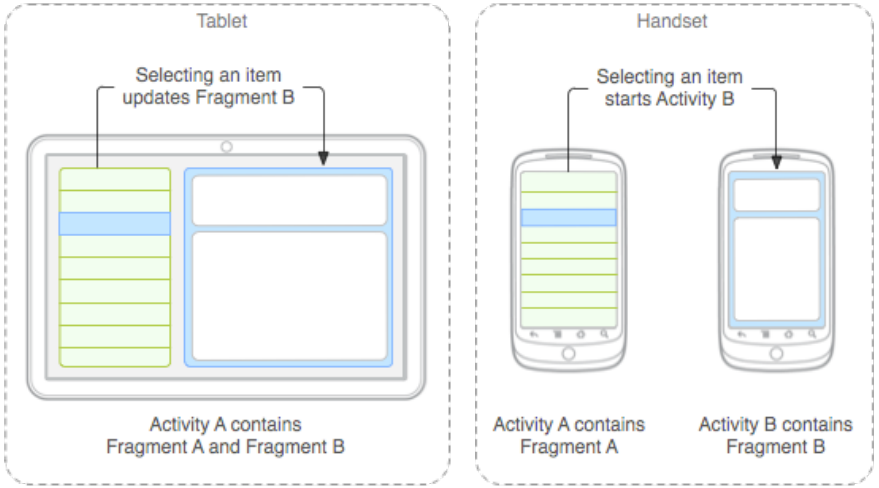
Navigation Drawer

- A panel (looks like a menu) that transitions in from the left edge of the screen
- Displays the app's main navigation options
- Reduce clutter in navigation for apps with **deep hierarchies**

Master/Detail List

A UI pattern optimized for both phones and tablets.

- Single-pane: on phone & tablet portrait mode
- Two-pane: on tablet landscape mode



Demo

- Walk through SampleMasterDetail
  - Exam how to use fragments to support multiple screens

Reminder

- Add project idea to Catalyst