Chapter 29 - Fragments and https://developer.android.com/guide/components/fragments and "ProAndroid 4"

- Introduced in 4.0, allows more dynamic UIs without having multiple activities
- Phone vs Tablet
  - screen: small vs large
  - activity: fills entire screen vs use part of the screen
  - back button use.... back normally kills an activity
    - activity may go dormant rather than quit
- Enter the "Fragment" ... a sub activity
  - Attached to an activity (kind of a sub-activity)
  - Has a View hierarchy connected to it
  - Does not necessarily use the whole screen ... in a UI element
  - Can make them respond to the back button
  - Has a much more complex lifetime than an activity
  - Can be destroyed and recreated during the run of an activity
  - Can be set to live past activity destruction/recreation
- Example app from chapter 29 has static fragments
Fragments

Life cycle:

- `void onInflate(Activity, AttributeSet, Bundle)` -- called by an inflater
  - `AttributeSet` -- from XML, array of name/value pairs
  - `attrs.getAttributeCount()` -- number
  - `attrs.getAttributeName(i)` -- string
  - `attrs.getAttributeValue(i)` -- string
  - Typically called via an activity `setContentView()`

- `onAttach(Activity)` -- called once, associated Activity
  - `getActivity()` -- returns this activity (fragment method)
  - `Arguments` -- saved `Bundle` -- reattached on reinitialization
    - `Bundle getArguments()` -- gets them
    - `setArguments(Bundle)` -- can be called before `onAttach()`
Fragments (page 2)

- **onCreate(Bundle savedInstanceState)** -- Activity may not have finished onCreate()
  - Don’t deal with views here
  - Can start background threads here
  - may call setRetainInstance() here

- **View onCreateView(LayoutInflater, ViewGroup, Bundle)**
  - Fragment creates its user interface view here
  - Many ways to create view, code and/or xml inflater
  - Can return null if not associated view/viewgroup
  - Return non-null, onDestroyView() will be called

- **void onActivityCreated (Bundle)** -- view hierarchy is instantiated, (activity ready)
  - final tweaks to UI to be done here
  - if being recreated from saved state, do work here

- **void onStart() -- UI is visible to user, no interaction**
- **void onResume() -- user interaction is now possible**

Fragment is going now! (May not be the only visible fragment in this state)
Best way to set up a Fragment -- static factory method

```java
public static MyFragment newInstance (...argumentlist...) {
    MyFragment f = new MyFragment();
    Bundle args = new Bundle();
    args.putXXX() .... build the bundle
    f.setArguments(args);
    return f;
}
```

- Can have several static factory methods
- Destroyed and reinitialized fragments use the default constructor.
Fragments (Page 4)

Fragment shutdown sequence
- onPause() -- no more user interaction
- onStop() -- fragment is not visible
- onDestroyView() -- cleanup view, don’t retain pointers
- onDestroy() -- "final cleanup"
- onDetach() -- just to be no longer associated with the activity

Tweak for "long living fragments" -- little differences (across activity destroy/recreate)
- fragment can’t be in the back stack (talk about this soon)
- void setRetainInstance (boolean) -- call with true
  - onDestroy() not called, onDetach() still called
- On "recreation of activity"
  - onAttach and onActivityCreated() still called, onCreate() not.
- activity gets new fragment manager
- activity can get fragment from the fragment manager
Fragment in XML -- very similar to views in XML.

```xml
<fragment
    class="com.androidbook.fragments.bard.TitlesFragment"
    android:id="@+id/titles" android:layout_weight="1"
    android:layout_width="0px"
    android:layout_height="match_parent"
    android:background="#00550033" />
```

class attribute used

- onInflate() called. Could have your own XML tags
  - No details here, to use it you need to find out how to do it.

- onAttachFragment (Fragment f) -- method in an activity
Code management of Fragments: Fragment management (in the activity)

- FragmentManager activity.getFragmentManager()

- Methods in FragmentManager
  - findFragmentByTag(int), findFragmentByIdbyTag() -- getting fragments
  - FragmentTransaction ft = beginTransaction() -- transactions to operate on
  - ft.commit() -- commits changes
  - ft.add(containerViewId, Fragment) // various other add() methods
  - ft.replace(containerViewId, Fragment) // put fragment in the view
  - ft.setTransition()
    - TRANSIT_NONE, TRANSIT_FRAGMENT_FADE, ....
  - ft.setCustomAnimations(enter, exit)
    - res/animator -- various ObjectAnimator XML files
  - ft.addToBackStack(TAG) -- put current fragment on "Back Stack"

- Other FragmentManager methods for the back stack:
  - popBackStack(several), getBackStackEntryCount(), popBackStackImmediate
  - addOnBackStackChangedListener, removeBackStackChangedListener

Example program ShakespeareInstrumented from ProAndroid4.zip (From Pro Android 4)
Other Fragment notes

- FragmentTransaction add(Fragment f, String tag)
  - Adds fragment to activity, does not show it
- It is not "cool" to keep fragment references
  - Fragments can be "destroyed" at any time
  - Need to always ask Activity for fragment, use string Tags.
- On a phone, an activity’s job may be to just start a fragment for the UI
  - See portrait mode for the Shakespeare app
Fragment Communication

Activity - Fragment communication
- Activities create fragments
  - Can then call fragment methods
- Fragments can get a reference to their activities
  - Can then call activity methods

Fragment to Fragment communication
- Activity can have methods that transfer data from one fragment to another
- Fragments can create other fragments
- Fragments can get a reference to another fragment from the fragment manager
  - Then directly call methods in the other fragment
- A fragment that starts another fragment can set up direct connections
  - original fragment: creates new fragment(f), f.setTargetFragment(this,0)
  - new fragment: getTargetFragment(), can directly call methods in original

Fragments can also start activities ...
- startActivity() and startActivityForResult()