**Topic: Android Activity Lifecycle, Multi-Screen Apps**   
(Previous topic: Intro)

* Review Quiz 1
* Lab 1 Peer Evaluation
  + Discussion questions
    - What does SetContentView do? Does it need to be called after all the other UI related code?
    - Is it appropriate to define this event handler using a lambda expression? Why?
    - What are the pros and cons of defining a UI in code vs. defining it in xaml?
* Review Quiz 2
* Application Fundamentals   
  <http://developer.android.com/guide/components/fundamentals.html>
  + Android Application Components
    - Activities: Single screen with a UI. Subclass of Activity  
      Each activity is independent of the others. Another application can potentially start any activity.
    - Services: Run in the background, no UI. Subclass of Service.
    - Content providers: Read / write data from a variety of sources. Subclass of ContentProvider.
    - Broadcast receivers: Respond to system-wide broadcast announcements. Subclass of BroadcastReceiver.
    - Activating Components
      * Activities, Services, and Broadcast receivers are activated by Intents.   
        An intent:
        + Is an asynchronous message.
        + Can be sent to components in a different application.
        + Bind individual components to each other at run time.
        + Define the action for an activity or service to perform.
      * Content Providers are activated when targeted by a request from a ContentResolver.
  + The Manifest File
    - Read by the system before starting an app.
    - Declares all the components in the app.
    - Identifies: permissions, minimum API level, hardware & software features used, API libraries required, etc.
  + Resources
  + Context  
    <http://www.levinotik.com/2011/10/12/demystifying-context-in-android/>
    - Activity inherits from Context.
    - Context provides access to other parts of the application and to the system.
* Activities  
  <http://developer.android.com/guide/components/activities.html>
  + Notified of change of state through callbacks.
  + UI provided by “widget” objects from the View class and layout objects from the ViewGroup class.
  + Intent filters, using the <intent-filter> element, declare how other application components may activate an activity.
  + Lifecycle: an activity can be in one of three states:
    - Resumed (running): running in the foreground and has user focus.
    - Paused: another activity is in the foreground and has focus, but this one is still visible. This activity’s object is still in memory, retains all state and member info and is attached to the window manager. But, it can be killed by the system.
    - Stopped: The activity’s UI is completely obstructed by another activity. This activity object is still in memory and retains state and member info, but is not attached to the window manager. It can be killed by the system.
  + Lifecycle callbacks:   
    <http://developer.android.com/guide/components/activities.html#ImplementingLifecycleCallbacks>
    - onCreate
    - onStart
    - onRestart
    - onResume
    - onPause
    - onStop
    - onDestroy
  + Saving activity state:  
    <http://developer.android.com/guide/components/activities.html#SavingActivityState>
    - OnSaveInstanceState is called before the system makes your activity vulnerable to destruction. The system passes a Bundle with this method call. You can save the activity’s state in the Bundle object.
    - Every View object’s state is saved by the default implementation of onSaveInstanceState. So UI visual info is saved automatically.
    - Rotation causes the system to destroy an activity.