CSCE 315: Android Lectures (2/2)

- Dr. Jaerock Kwon
- Kettering University

App Development for Mobile Devices

Jaerock Kwon, Ph.D. Assistant Professor in Computer Engineering

Announcement

Lecture 2
Application Fundamentals

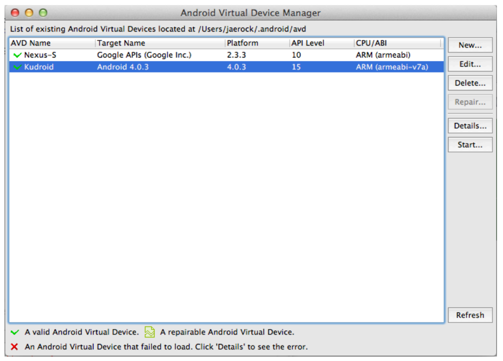
Today's Topics

- Android development tools
  - Virtual Device Manager
  - Android Emulator
  - Dalvik Debug Monitoring Service
- Application components
  - Activity
  - Service
  - Broadcast receiver
  - Content provider
  - Intent
  - App manifest
- Application resources

Android Development Tools

Virtual Device Manager

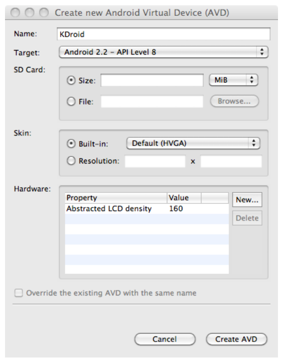
- Create and manage Android Virtual Devices
  - Window menu item in Eclipse – Android SDK and AVD Manager
- You can start Android Emulator from the AVD Manager.



List of existing Android Virtual Devices located at /Users/jaerock/.android/avd

AVD Name	Target Name	Platform	API Level	CPU/ABI
✓ Nexus-S	Google APIs (Google Inc.)	2.3.3	10	ARM (armeabi)
✓ Kudroid	Android 4.0.3	4.0.3	15	ARM (armeabi-v7a)

✓ A valid Android Virtual Device. A repairable Android Virtual Device.
✗ An Android Virtual Device that failed to load. Click 'Details' to see the error.



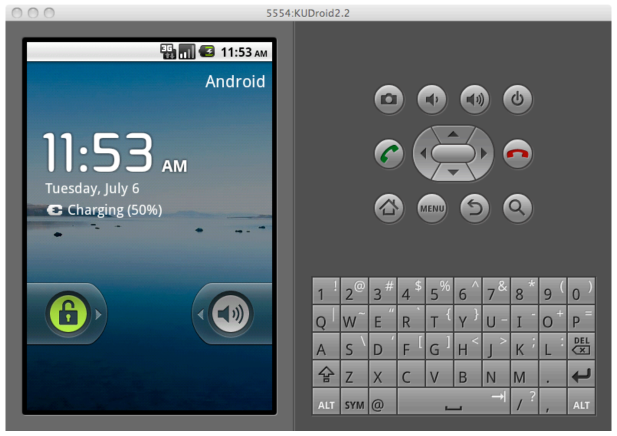
Name: KDroid
Target: Android 2.2 – API Level 8
SD Card: [✓ Size:] MiB; File:
Skin: [✓ Built-in:] Default (HVGA); Resolution: x
Hardware:

Property	Value
Abstracted LCD density	160

☐ Override the existing AVD with the same name

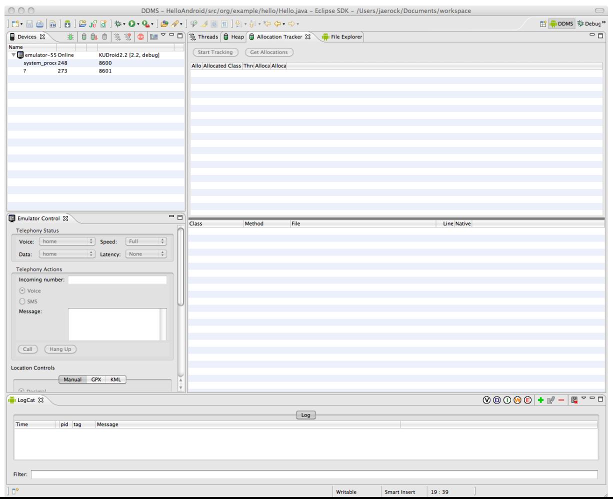
Android Emulator

- An implementation of the Android virtual machine.
- Use this to test and debug your Android applications.



Dalvik Debug Monitoring Service

- Use the DDMS perspective to monitor and control Dalvik virtual machines on which you are debugging your apps.



Android Applications

Android Application

- Written in Java
- The compiled Java code along with resource files and data is bundled by **aapt** tool into an Android package.
  - aapt (Android Asset Packaging Tool)
    - Probably you will not use this directly.
    - IDE plugins utilizes this tool to package the apk file.
  - Android package:
    - A single archive file. Its filename extension is .apk.
    - This apk file is the file that users download to their devices.
- Linux process
  - Every application runs in its own Linux process.
  - Each process has its own virtual machine.

Central Features of Android

- An application can use elements of other applications (should be permitted by the apps).
  - For this to work, an application process can be started when any part of it is needed and instantiate the Java objects for that part.
  - Therefore, Android apps don't have a single entry point (like **main()** function).
  - Rather they have essential components that the system can instantiate and run as needed.
- Four types of components
  - Activities
  - Services
  - Broadcast receivers
  - Content providers

Application Components

Activities

- Application's presentation layer.
- Every screen in you app is an extension of the `Activity` class.
- Each activity is given a default window to draw in.
- Activities use Views to form GUI.
  - Each view controls a particular rectangular space within the window.
  - Views are where the activity's interaction with the user takes place.
  - ContentView is the root view object in the hierarchy of views.
    - `Activity.setContentView()` method.
- Activity is equivalent to Form in desktop environment.

Services

- No visual interface.
- Runs in the background of an indefinite period of time.
- Examples:
  - Play background music, fetch data over the network.
- Each service extends the `Service` base class
- It is possible to connect to an ongoing service (and start the service if it is not already running).
  - You can communicate with service through an interface that the service exposes.
    - Examples: start, pause, stop, restart playback.
- Services run in the main thread of the application process.
  - It is not a sub thread of other components.

Broadcast Receivers

- A Broadcast Receiver receives and reacts to broadcast announcements.
- Broadcast examples from the system
  - The timezone change, the battery is low, a picture has been taken, and etc.
- An application can have any number of receivers to respond to any announcements.
- BRs do not display user interface.
- BRs can start an activity in response to the information they receive.

Intents

An inter-app message passing framework

- Activities, Services, Broadcast Receivers are activated by Intents, asynchronous messages.
- You can broadcast messages system-wide or to a target Activity or Service.
- Then, the system will determine the target(s) that will perform any actions as appropriate.
- Separate methods for activating each type of component.
  - Activity: `Context.startActivty()` or `Activity.startActivityForResult()`
    - The responding activity can look at the initial intent that caused it to be launched by calling `getIntent()`.
  - Service: `Context.startService()`
    - Android calls the service's `onStart()` method and passes it the intent object.
  - Broadcast: `Context.sendBroadcast(), Context.sendOrderedBroadcast(),` …
    - Android delivers the intent to all interested receivers by calling their `onReceive()`

Intents Object and Intent Filters

- Intent Object
  - An abstract description of an operation to be performed.
- Android system finds the appropriate activity, service, or set of broadcast receivers to respond to the intent.
- Two groups of intents
  - Explicit intents:
    - They name the target component.
    - Component names are not known to developers of other apps.
    - So they are used for application internal messages.
  - Implicit intents:
    - They are often used to activate components in other apps.
- For implicit intents
    - Need to test the intent object against **Intents Filters** associated with potential target.

Intent Objects

A bundle of information

- Intent Objects contains information of component that receives the intent and the Android system
- Intent Objects contains
  - Component name - optional
  - Action
    - A string naming the action to be performed.
    - Examples
      - ACTION_CALL: Initiate a phone call
      - ACTION_EDIT: Display data for the user to edit
      - **ACTION_MAIN**: Start up as the initial activity of a task
      - ACTION_BATTERY_LOW: A WARNING THAT THE BATTERIY IS LOW
  - Data
    - The URL of the data to be acted on.
  - Category
    - Examples:
      - CATEGORY_HOME: The activity displays the home screen.
      - **CATEGORY_LAUNCHER**: The activity can be the initial activity and is listed in the top-level application launcher.

Intent Filters

- Intents should be resolved since implicit intents do not name a target component.
- Intent filters are generally in the app's manifest file (AndroidManifest.xml)
- Most apps have a way to start fresh. The following **action** and **category** are default values of an Android project.

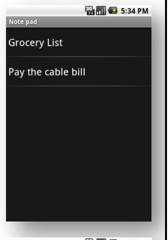
```
<intent-filter>
    <action android:name="android.intent.action.MAIN" />
    <category android:name="android.intent.category.LAUNCHER" />
</intent-filter>
```

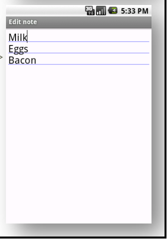
Note Pad Example

- There are three activities: NotesList, NoteEditor, and TitleEditor

  -
    ```
    <activity android:name="NotesList" android:label="@string/title_notes_list">
    …
    </activity>
    <activity android:name="NoteEditor" ...
    …
    </activity>
    <activity android:name="TitleEditor" ...
    …
    </activity>
    ```
- Each activity has intent-filters. Followings are from **NotesList** activity.

  -
    ```
    <intent-filter>
       <action android:name="android.intent.action.MAIN" />
       <category android:name="android.intent.category.LAUNCHER" />
    </intent-filter>
    <intent-filter>
       <action android:name="android.intent.action.VIEW" />
       <action android:name="android.intent.action.EDIT" />
       <action android:name="android.intent.action.PICK" />
       <category android:name="android.intent.category.DEFAULT" />
       <data android:mimeType="vnd.android.cursor.dir/vnd.google.note" />
    </intent-filter>
    ```
- The second intent filter declares that the activity can VIEW, EDIT, PICK in the data URI.





A Simple Dialer

- Your program should have something like…
  ```
  String number = "tel:810-555-1234";
  Intent callIntent = new Intent(Intent.ACTION_CALL, Uri.parse(number));
  startActivity(callIntent);
  ```
- Do not forget to add the permission for phone call in your App Manifest XML file.
  ```
  uses-permission android:name="android.permission.CALL_PHONE"
  ```

Content Providers

- A CP makes a specific set of the app's data available to other apps.
- `ContentProvider` base class should be extended to implement a standard set of methods.
- You can share your data and you can access shared data of other apps.
- Apps do not call `ContentProver` methods directly. Rather they use a `ContentResolver` object and call its methods instead.
- `ContentProviders` are activated when they are targeted by a request from a `ContentResolvers`.

Shutting down Components

- An activity can be shut down by calling its `finish()` method.
- A service can be stopped by calling its `stopSelf()` or `Context.stopService()`.

App Manifest

- Each Android project includes a manifest file, `AndroidManifest.xml` for all apps (same name).
- A structured XML file.
- The principal task of it is to inform Android about the app's components.
  - <activity>, <service>, <receiver> elements

```
<?xml version="1.0" encoding="utf-8"?>
<manifest . . . >
    <application . . . >
        <activity android:name=”edu.kettering.project.FreneticActivity"
                  android:icon="@drawable/small_pic.png"
                  android:label="@string/freneticLabel"
                  . . .  >
        </activity>
        . . .
    </application>
</manifest>
```

App Manifest - Intent Filters

```
<intent-filter . . . >
    <action android:name="android.intent.action.MAIN" />
    <category android:name="android.intent.category.LAUNCHER" />
</intent-filter>
```

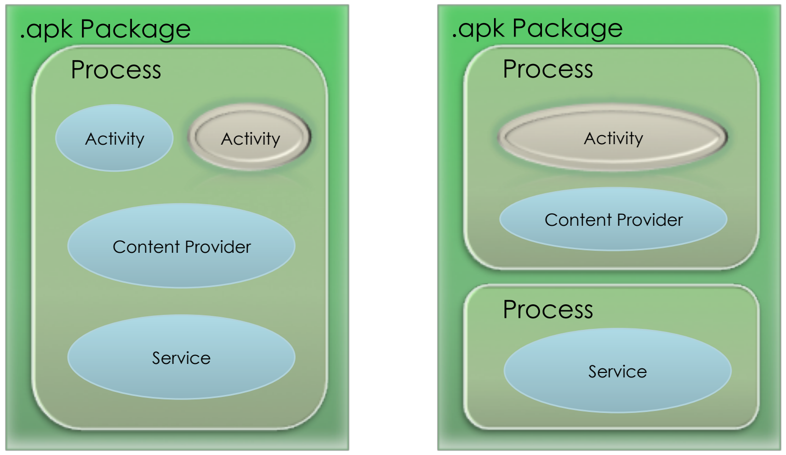
- IFs declare the capabilities of its parent component.
  - What an activity or service can do and what types of broadcasts a receiver can handle.
- Action "`android.intent.action.MAIN`" and category "`android.intent.category.LAUNCHER`" is the most common combination.
  - Note: application launcher: the screen listing apps where users can launch an app.
- The activity is the entry point for the app.

Activity, Tasks, and Processes

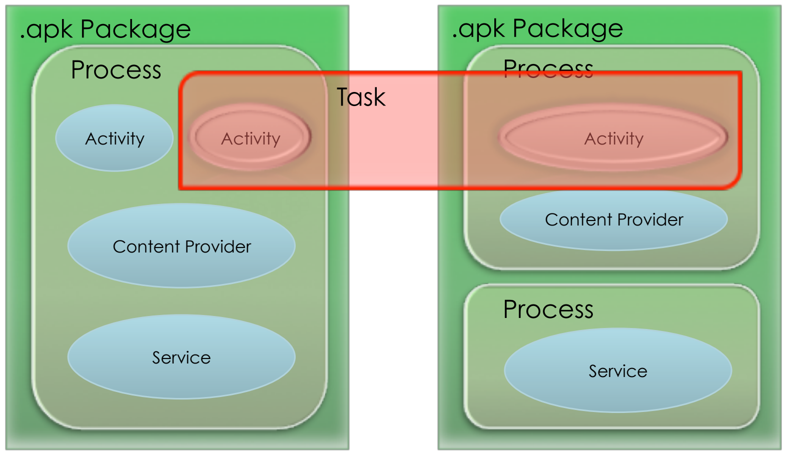
Activities, Tasks, and Processes

- One activity can start another including one in a different app.
  - Example:
    - Your activity needs to display a street map.
    - Assuming there is an activity that can do this.
    - You activity put together an Intent object and pass it to `startActivity()`.
- Definitions
  - Activity
    - The... Android 'Activity'
  - Task
    - A stack of activities
  - Process
    - A standard Linux process

Activities, Tasks, and Processes



Activities, Tasks, and Processes



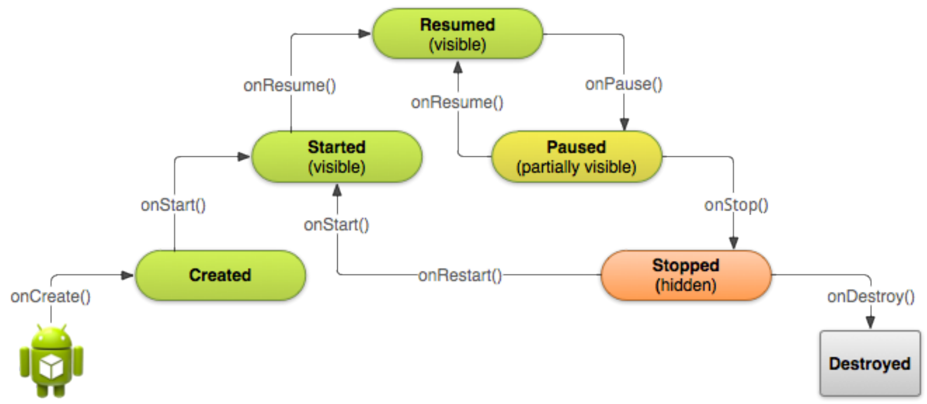
Process and Thread

- Application components are Linux processes.
  - When the first of an app's components needs to be run, Android starts a Linux process for it with a single thread of execution.
- Process
  - It is controlled by the Manifest file.
- Thread
  - User interface should always be quick to respond to user actions.
  - Anything that may not be completed quickly should be assigned to a different thread.
  - Threads are created in code using standard Java Thread objects.
    - Android also provides many convenient classes for managing threads.

Android Activity Lifecycle

- An activity has three states:
  - Active, or running:
    - When it is in the foreground of the screen.
  - Paused:
    - When it lost focus but is still visible to the user.
  - Stopped:
    - When it is completely obscured by another activity.
    - It still remains all state and member information.
    - It may be killed by the system when memory is needed elsewhere.
- Note: When an activity is paused or stopped, the system can drop it from memory or simply kill its process.

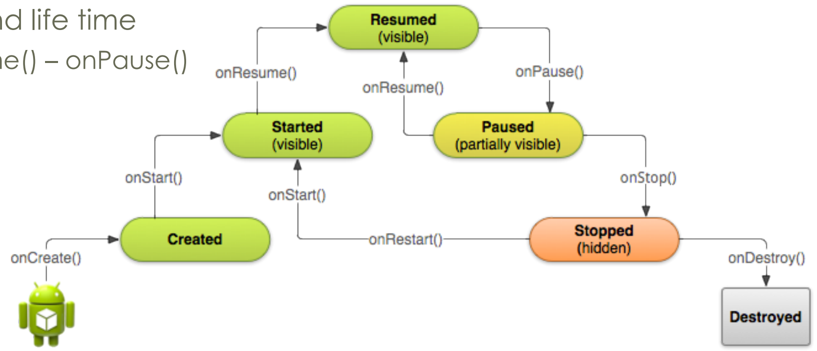
Android Activity Lifecycle



Android Activity Lifecycle

- Entire life time of an activity
  - onCreate() – onDestroy()
- Visible life time
  - onStart() – onStop()
- Foreground life time
  - onResume() – onPause()

```
void onCreate(Bundle savedInstanceState)
void onStart()
void onRestart()
void onResume()
void onPause()
void onStop()
void onDestroy()
```



Android Activity Lifecycle

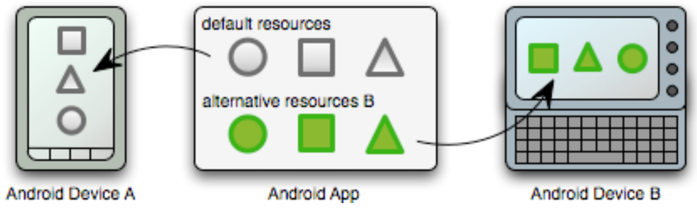
Saving Activity State

- Since the system can shut down an activity, the user may expect to return to the activity and find it in its previous state.
- onSaveInstanceState()
  - Android calls this method before making the activity to being destroyed.
- onRestoreInstanceState()
  - Android calls onRestoreInstanceState() after calling onStart().
- Note that these two methods are not lifecycle methods.
  - They are not called in a proper lifecycle step.
  - You cannot rely on these methods to save some persistent data.

Application Resources

Resource Externalization

- Externalizing resources such as images and strings
  - You can maintain them separately from your actual code.
  - This allows you to provide alternative resources that support different languages, different screen sizes.
    - This is extremely important because Android-powered devices become available with different configurations.



Grouping Resource Types

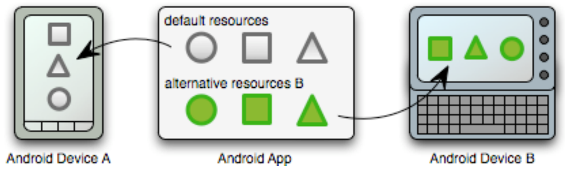
- Place resources in a specific subdirectory of your project's res/ directory.
- Resource directories supported inside project res/ directory.

Directory	Type
anim/	Define tween animation
color/	Define a state list of colors
drawable/	Bitmap files or XML files that
layout/	Define user interface layout
menu/	Options Menu, Context Menu, or Sub Menu
raw/	Arbitrary files to save in their raw form
values/	Strings, integers, colors
xml/	Arbitrary XML files

Providing Alternative Resources

- To specify configuration-specific alternatives for a set of resources:
  - Create a new directory in res/ named in the form <resources_name>-<config_qualifier>.
  - <resources_name> is the directory name of the corresponding default resources.
  - <qualifier> is a name that specifies an individual configuration for which these resources.
  - You can append more than one <qualifier>. Separate each one with a dash.
  - Save the respective alternative resources in this new directory. The resource files must be named exactly the same as the default resource files.
- For example, here are some default and alternative resources:

```
res/
    drawable/
        icon.png
        background.png
    drawable-hdpi/
        icon.png
        background.png
```



Creating Resources

Simple Values

- String
  - <string name="your_name">Kettering</string>
- Color
  - <color name="transparent_blue">#770000FF</color>
  - Format
    - #RGB
    - #RRGGBB
    - #ARGB
    - #AARRGGBB
- Dimensions
  - <dime name="border">5dp</dimen>
  - Scale identifier
    - px (screen pixels)
    - in (physical inches)
    - pt (physical points)
    - mm (physical millimeters)
    - dp (density independent pixels relative to a 160-dpi screen)
    - sp (scale independent pixels)

Supporting Different Screens

- Four general categorized device screens:
  - Sizes: small, normal, large, xlarge
  - Densities: low(ldpi), medium (mdpi), high(hdpi), extra high (xhdpi).
- Create different layouts

  -
    ```
    MyProject/
        res/
            layout/
                main.xml
            layout-large/
                main.xml
    ```
  - Simply reference the layout file in your app as usual.

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);
}
```

Supporting Different Screens

- Create different bitmaps
  - xhdpi: 2.0
  - hdpi: 1.5
  - mdpi: 1.0 (baseline)
  - ldpi: 0.75
- This means that if you generate a 200x200 image for xhdpi devices, you should generate the same resource in 150x150 for hdpi, 100x100 for mdpi, and 75x75 for ldpi devices.
- Then place the files in the appropriate drawable directories

```
MyProject/
    res/
        drawable-xhdpi/
            awesomeimage.png
        drawable-hdpi/
            awesomeimage.png
        drawable-mdpi/
            awesomeimage.png
        drawable-ldpi/
            awesomeimage.png
```

Creating Resources

- Drawable
  - Drawable resources include bitmaps and NinePatch (stretchable PNG) images.
- Layouts
  - Layout resources let you decouple your app's presentation layer.
  - Designing user interfaces in XML rather than constructing them in code.

Accessing Resources

- You can refer a resource via its ID.
- R class
  - All resource IDs are defined in your project's R class.
  - The R class is automatically generated by the aapt tool.
- Resource ID
  - A resource ID has a unique name and is composed of:
    - Resource type:
      - string, drawable, layout, etc.
    - Resouce name:
      - Either the filename (excluding the extension) or the value in the XML android.name attribute, if the resource is a simple value such as a string, a color.
- Accessing a resouce: string is a resource type. hello is a resource name
  - In code: R.string.hello
  - In XML: @string/hello

Layout Definition

- Layout is an architecture about user interfaces in an Activity
- Two ways of definition of layouts
  - XML
    - Use resource editor to make layout.
    - ADT provides the preview of an XML file.
    - The best way is to make a layout is using both the XML editor and the XML graphical editor.
  - In code
    - Create Views and ViewGroups in runtime.

XML for Resource

- Only one root element that should be a View or a ViewGroup.
- Add child elements to the root view.

```
<?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" >
    <TextView android:id="@+id/text"
              android:layout_width="wrap_content"
              android:layout_height="wrap_content"
              android:text="Hello, I am a TextView" />
    <Button android:id="@+id/button"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="Hello, I am a Button" />
</LinearLayout>
```

XML Resource Load

- When you compile your source code, each XML layout file is compiled into a View resource.
- You should load it in your Activity.onCreate().
- XML file: `res/layout/*.xml`
  - If the xml file name is **`main`**`.xml`, then the layout can be accessed by `R.layout.`**`main`** in your source code.

```
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);
}
```

ID of Element

- Any View object has a unique ID.
- In your XML code, the ID is defined with a string.
  - android:id="@+id/**myButton**"
  - @ indicates the rest of the string should be identified as an ID resource.
  - + means adding new resource name.
- In your source code, the ID can be referred by an integer.
  - Button myButton = (Button)findViewById(R.id.**myButton**);
- Example:

```
In XML …
<Button android:id="@+id/myButton"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/my_button_text"/>

In source code …
Button myButton = (Button) findViewById(R.id.myButton);
```

Questions?