**455-555 Week 3 Notes**

**Categories:**

Chap. 3-4 - FRAGMENTS (see PowerPoint)

Chap 4 – Use of the Android Manifest / Gradle builds & dependencies

Chapter 5 - Building Great UI’s (layouts, colors, sizes, hardware!)

Chapter 6 – Intents introduction (see SMS Intents zip demo file)

**Chapter 4 Utilizing the Android Manifest for hardware configs**

**Categories:**

* **Audio**—For applications that require a low-latency or pro-level audio pipeline or require microphone input.
* **Bluetooth**—Where a Bluetooth or BTLE radio is required.
* **Camera**—For applications that require a camera. You can also require (or set as optional) front- or rear-facing, autofocus, manual post-processing, manual sensor, flash, or RAW support.
* **Device Hardware UI**—Where the application is designed for a specific device user interface: such as automotive or watch.
* **Fingerprint**—Requires biometric hardware capable of reading fingerprints.
* **Gamepad**—For games (or apps) that require game controller input, either from the device itself or from a connected gamepad.
* **Infrared**—Indicates a requirement for infrared (IR) capabilities (typically to communicate with other consumer IR devices).
* **Location**—If you require location-based services. You can also specify either network or GPS support explicitly.
* **NFC**—Requires NFC (near-field communications) support.
* **OpenGL ES hardware**—The application requires the OpenGL ES Android Extension Pack installed on the device.
* **Sensors**—Enables you to specify a requirement for any of the potentially available hardware sensors, including a accelerometer, barometer, compass, gyroscope, sensors to detect ambient temperature, heart rate, light, proximity, humidity, and a step counter and step detector.
* **Telephony**—To specify that either telephony in general, or a specific telephony radio (GSM or CDMA) is required.
* **Touchscreen**—To specify the type of touch screen your application requires, including how many distinct input touches can be detected and tracked.
* **USB**—For applications that require either USB host or accessory mode support.
* **Wi-Fi**—Where Wi-Fi networking support is required.
* **Communication software**—The application requires support for Session Initiation Protocol (SIP) services or Voice Over Internet Protocol (VoIP) services.
* **Device management software**—Use these optional software features to specify that your application requires the device support device management features including backup service, device policy enforcement, managed users, user removal, and verified boot.
* **Media software**—If your application requires MIDI support, printing, a “lean back” (television) UI, live TV, or home screen widgets.

Set up uses in the manifest file

<https://developer.android.com/guide/topics/manifest/uses-feature-element#features-reference>

Ex. Set up device to allow ONLY touch screen features, and NOT for usages such as with a d-pad controller you would code as follows in your manifest

<uses-feature android:name="android.hardware.touchscreen"  
    **android:required="true"** />

d-pad controller



## CONFIGURING THE GRADLE BUILD

Each project contains a series of Gradle files used to define your build configuration, consisting of a:

* Project-scoped settings.gradle file that defines which modules should be included when building your application.
* Project-scoped build.gradle file in which the repositories and dependencies for Gradle itself are specified, as well as any repositories and dependencies common to all your modules.
* Module-scoped build.gradle file(s) used to configure build settings for your application, including dependencies, minimum and targeted platform versions, your application’s version information, and multiple build types and product flavors.

**Chap. 5- building Greeeat UI’s!**

Setting Colors

**Colors for text view**

You should have a file in **res/values** called **colors.xml**, it should look like the following:

*<?***xml version="1.0" encoding="utf-8"***?>*<**resources**>  
 <**color name="background"**>#0044AA</**color**>  
 <**color name="font"**>#fff</**color**>  
</**resources**>

Then you can just call the color in layout xml file like this:

**android:textColor="@color/font"  
android:background="@color/background"**

Color definitions – RGB, Hex

Use the color tag to define a new color resource. Specify the color value using a # symbol followed by the (optional) alpha channel, and then the red, green, and blue values using one or two hexadecimal numbers with any of the following notations:

* #RGB
* #RRGGBB
* #ARGB
* #AARRGGBB

The following example shows how to specify Android green and a partially transparent blue:

<color name="android\_green">#A4C639</color>

<color name="transparent\_blue">#770000FF</color>

##### Dimensions - Setting sizes for fonts, images, etc.

Dimensions are most commonly referenced within style and layout resources. They’re useful for defining layout values, such as borders and font heights.

Dimension possibilities…

* dp (density-independent pixels)
* sp (scalable pixels)
* px (screen pixels)
* in (physical inches)
* pt (physical points)
* mm (physical millimeters)

Although you can use any of these measurements to define a dimension, it’s best practice to use either density independent or scalable pixels. These alternatives let you define a dimension using relative scales that account for different screen resolutions and densities to simplify scaling on different hardware.

Scalable pixels are particularly well suited when defining font sizes because they *automatically* scale if the user changes the system font size.

The following XML snippet shows how to specify dimension values for a large font size and a standard border:

Ex.

**android:textSize="32sp"**

**More on Layouts!**

<https://developer.android.com/guide/topics/ui/layout/linear>

Easy and fast layout params

Check into [layout\_gravity, layout\_weight](https://developer.android.com/reference/android/widget/LinearLayout.LayoutParams) features (chapter 5 examples)

<EditText  
        android:layout\_width="match\_parent"  
        android:layout\_height="wrap\_content"  
        android:hint="@string/to" />  
    <EditText  
        android:layout\_width="match\_parent"  
        android:layout\_height="wrap\_content"  
        android:hint="@string/subject" />  
    <EditText  
        android:layout\_width="match\_parent"  
        android:layout\_height="0dp"  
        android:layout\_weight="1"  
        android:gravity="top"  
        android:hint="@string/message" />  
    <Button  
        android:layout\_width="100dp"  
        android:layout\_height="wrap\_content"  
        android:layout\_gravity="right"  
        android:text="@string/send" />

Result follows:

A screenshot of a cell phone

Description automatically generated

Takeaways when setting a layout using layout\_weights

**Equal distribution**

To create a linear layout in which each child uses the same amount of space on the screen, set the **android:layout\_height** of each view to "0dp" (for a vertical layout) or the **android:layout\_width** of each view to "0dp" (for a horizontal layout). Then set the **android:layout\_weight** of each view to "1".

**Unequal distribution**

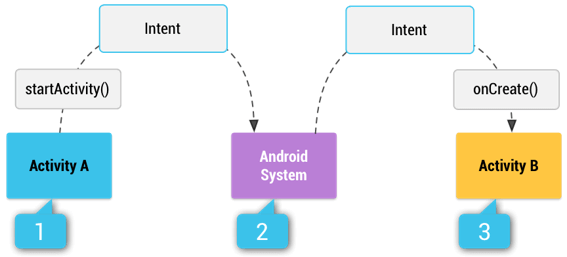
You can also create linear layouts where the child elements use different amounts of space on the screen:

* If there are three text fields and two of them declare a weight of 1, while the other is given no weight, the third text field without weight *doesn't* grow. Instead, this third text field occupies *only* the area required by its content. The other two text fields, on the other hand, expand equally to fill the space remaining after all three fields are measured.
* If there are three text fields and two of them declare a weight of 1, while the third field is then given a weight of 2 (instead of 0), then it's now declared more important than both the others, so it gets half the total remaining space, while the first two share the rest equally.

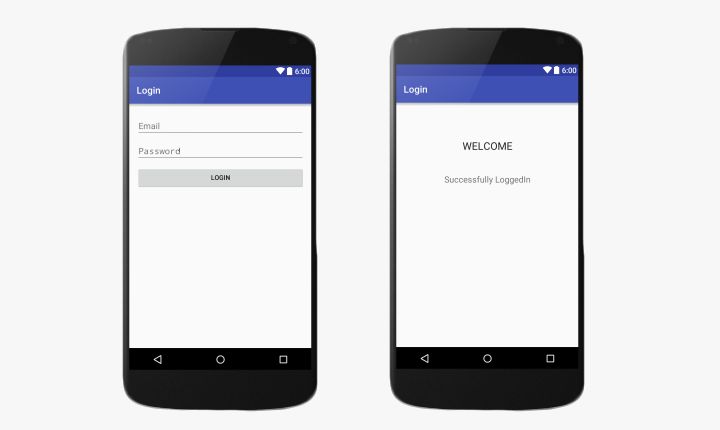
[ Chapter 6 – Intents introduction ]

[**Intents**](https://developer.android.com/reference/android/content/Intent)

An Intent is an object that provides runtime binding between separate components, such as two activities. The Intent represents an app’s "intent to do something." You can use intents for a wide variety of tasks, but in this lesson, your intent starts another activity.



Typical intent to pass some value(s) to another activity, or just to travel to another activity!



Methods

* The Intent constructor takes two parameters to communicate between activities:

Intent intent = **new** Intent(**this**, Main2Activity.**class**);

* + A Context as its first parameter (this is used because the Activity class is a subclass of Context)
  + The Class of the app component to which the system should deliver the Intent (in this case, the activity that should be started).
* The startActivity() method starts an instance of an Activity specified by the Intent.

startActivity(intent);

[ Method options for passing/receiving data between activities ]

* The putExtra() method adds the value to the intent. An Intent can carry data types as key-value pairs called *extras*. It's a good practice to define keys for intent extras using your app's package name as a prefix. This ensures the keys are unique, in case your app interacts with other apps.

intent.putExtra(***SECRET\_MESSAGE***, message);

* The getExtra() method retrieves the value to the intent.

*// Get Intent that started this activity and extract the string*Intent intent = getIntent();  
String message = intent.getStringExtra(MainActivity.***SECRET\_MESSAGE***);

\*\*\*Demo time! Intent use cases: See Demos folder on various Intents –

*filename*: **SMSIntents.zip**

**SMS**

SMS (short message service) is a text messaging service component of most telephone, internet, and mobile-device systems.  It uses standardized communication protocols to enable mobile devices to exchange short text messages. An intermediary service can facilitate a text-to-voice conversion to be sent to landlines. SMS was the most widely used data application, with an estimated 3.5 billion active users, or about 80% of all mobile subscribers, by the end of 2010.

To run SMS, use an **actual** Android device to run app.

Set up device as follows:

-On Android Device, in Settings > Developer, allow for USB Debugging

-Next load apk onto external device from Android Studio.

-Run app on external device.

Trouble Tips

* Connect / Reconnect cable
* Set min sdk in gradle file

Manifest / Code Example follows

*<?***xml version="1.0" encoding="utf-8"***?>*<**manifest xmlns:android="http://schemas.android.com/apk/res/android"  
 package="com.example.mypackage.smsintents"**>

**<uses-feature**

android:**name="android.hardware.telephony"**

android:**required="false**" />  
  
 <**uses-permission android:name="android.permission.SEND\_SMS"**/>  
 <**uses-permission**

**android:name="android.permission.RECEIVE\_SMS"** />  
   
 <**application**

**::**

**import** android.app.Activity;  
**import** android.content.Intent;  
**import** android.net.Uri;  
**import** android.os.Bundle;  
  
**public class** MainActivity **extends** Activity {  
  
 @Override  
 **protected void** onCreate(Bundle savedInstanceState) {  
 **::**

*Uri uriSms = Uri.parse("smsto:7735527766");  
 Intent intentSMS = new Intent(Intent.ACTION\_SENDTO,*

*uriSms);  
 intentSMS.putExtra("sms\_body", "The SMS text");  
 startActivity(intentSMS);* }  
}

Result on Emulator

A screenshot of a cell phone

Description automatically generated

A screenshot of a cell phone

Description automatically generated

[**AirDroid**](https://www.airdroid.com/)

-Install AirDroid from playstore on external device.

-On computer, download [Desktop client](https://www.airdroid.com/get.html) for viewing device over PC/Mac as an option.

A screenshot of a cell phone

Description automatically generated

Options…

A screenshot of a cell phone

Description automatically generated

Enable view mode on external device

A screenshot of a cell phone

Description automatically generated

A screenshot of a cell phone

Description automatically generated

Run app on external virtual device, view on PC /Mac live action!!!

A screenshot of a cell phone

Description automatically generated

A screenshot of a cell phone

Description automatically generated

Refs:

<https://code2care.org/pages/how-to-send-sms-on-android-emulator/>

<https://developer.android.com/studio/profile/android-profiler>

<https://developer.android.com/studio/intro/studio-config>

<https://developer.android.com/studio/command-line/adb>