Grow with Google Challenge Scholarship
Lesson 11 Notes

Pluralization in Android

Part of Android's robust resource framework involves a mechanism for pluralizing strings called "Quantity Strings". In the `strings.xml` file for the Hydration Reminder app, you'll see an example of how pluralization can be used:

```
<plurals name="charge_notification_count">
   <item quantity="zero">Hydrate while charging reminder sent %d times</item>
   <item quantity="one">Hydrate while charging reminder sent %d time</item>
   <item quantity="other">Hydrate while charging reminder sent %d times</item>
</plurals>
```

When you use the plural in code, you specify a quantity number. This number specifies what string should be used. In this case:

- if the number is zero, use `<item quantity="zero">`
- If the number is one, use `<item quantity="one">`
- otherwise use `<item quantity="other">`

Then in the `MainActivity` we have the following Java code to generate the correct String:

```
String formattedChargingReminders = getResources().getQuantityString(R.plurals.charge
_notification_count, chargingReminders, chargingReminders);
```

The first usage of chargingReminder is the quantity number. It determines which version of the pluralized string to use (you must pass in a number). The second usage of chargingReminder is the number that's actually inserted into the formatted string.

For more detail on Quantity Strings, check out the **documentation**

FirebaseJobDispatcher Sample Code

```
Driver driver = new GooglePlayDriver(context);
FirebaseJobDispatcher dispatcher = new FirebaseJobDispatcher(driver);

Job myJob = dispatcher.newJobBuilder()
    // the JobService that will be called
    .setService(MyJobService.class)
    // uniquely identifies the job
    .setTag("complex-job")
    // one-off job
    .setRecurring(false)
    // don't persist past a device reboot
    .setLifetime(Lifetime.UNTIL_NEXT_BOOT)
    // start between 0 and 15 minutes (900 seconds)
    .setTrigger(Trigger.executionWindow(0, 900))
    // overwrite an existing job with the same tag
    .setReplaceCurrent(true)
    // retry with exponential backoff
    .setRetryStrategy(RetryStrategy.DEFAULT_EXPONENTIAL)
    // constraints that need to be satisfied for the job to run
    .setConstraints(
        // only run on an unmetered network
        Constraint.ON_UNMETERED_NETWORK,
        // only run when the device is charging
        Constraint.DEVICE_CHARGING
    )
    .build();
```

For more information, check out the FirebaseJobDispatcher README. This also includes more sample code.

What is Google Play Services

You might be wondering what's up with the `GooglePlayDriver`. `FirebaseJobDispatcher` has a dependency on Google Play Services, which is why you need a `GooglePlayDriver`. So what is Google Play Services?

Google Play Services is app that Google maintains which comes pre-installed on and runs in the background on many, many phones. It is essentially a collection of Services that your app can use to leverage the power of Google products. If the user has the Google Play Services apk installed (and many do) you can use Google Play Services Libraries to easily do things like use the Places API to know where your user is or integrate Google sign in. FirebaseJobDispatcher is one of the many services you can take advantage of via Google Play Services.

Google choose to distribute these services as an installable apk so that updates to the services are not dependent on carrier or OEM system image updates. In general, devices running Android 2.3 (API level 9) or later and have the Google Play services app installed receive updates within a few days.

Google Play Services Udacity Courses

There are several Udacity courses on how to use Google Play Services features, such as Location services and Maps.

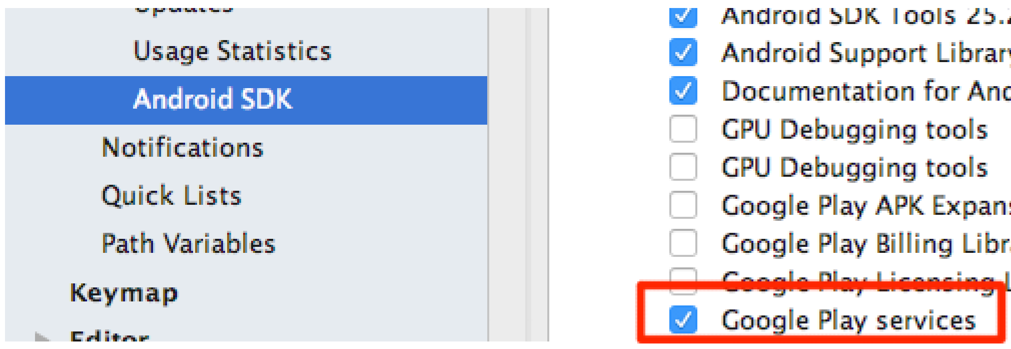
- Google Location Services
- Google Analytics
- Google AdMob
- Google Maps

Installing Google Play Services

To test your app when using the Google Play services SDK, you must use either:

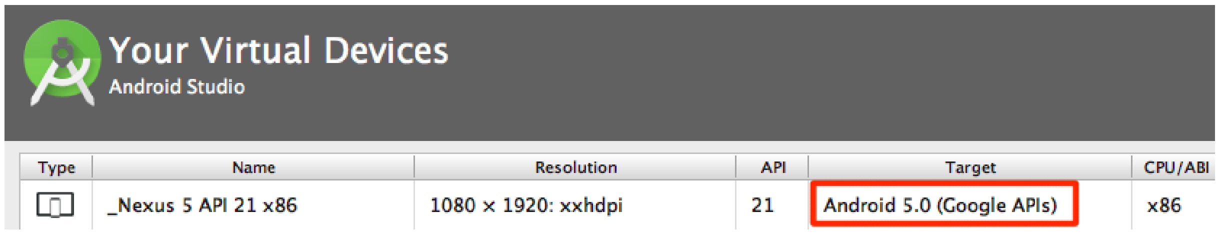
- A compatible Android device that runs Android 2.3 or higher and includes Google Play Store.
- The Android emulator with an AVD that runs the Google APIs platform based on Android 4.2.2 or higher.

You can install Google Play Services on your physical device via the Google Play Store.
To get Google Play Services on an emulator, you first need to make sure you have Google Play Services installed in the SDK manager:



- [✓] Android SDK Tools 25.2.3
- [✓] Android Support Library, rev 23.2.1
- [✓] Documentation for Android SDK
- [ ] GPU Debugging tools
- [ ] GPU Debugging tools
- [ ] Google Play APK Expansion library
- [ ] Google Play Billing Library
- [ ] Google Play Licensing Library
- [✓] Google Play services

Then you need to create an emulator that uses the **Google APIs**:



Type	Name	Resolution	API	Target	CPU/ABI
	_Nexus 5 API 21 x86	1080 × 1920: xxhdpi	21	Android 5.0 (Google APIs)	x86

What if Google Play Services is not Available?

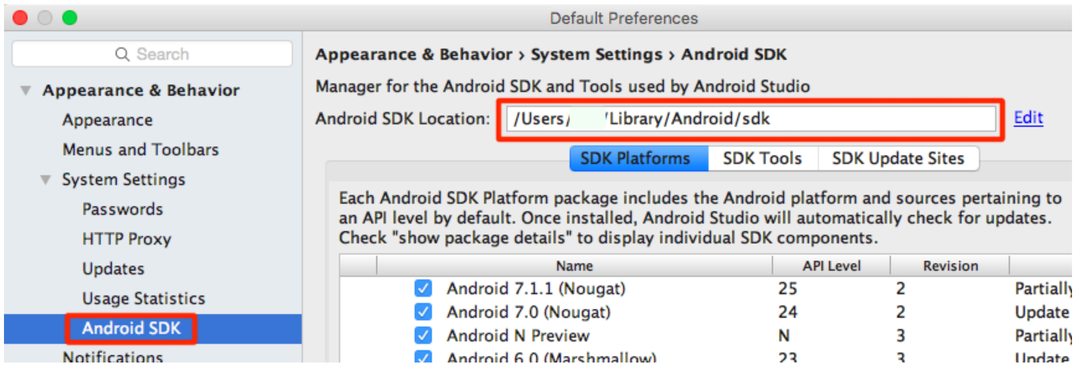
In this class we are only covering `FirebaseJobDispatcher`. Depending on where you're located and who your users are, you might not have access to the Play Store to download Google Play Services. If this is the case, you can use an alternative to FirebaseJobDispatcher known as **android-job**. Android-job is very similar to FirebaseJobDispatcher, but it is not a Google maintained repository and it only offers compatibility back to API 14. The FirebaseJobDispatcher documentation contains a **comparison table** that you can use when making this decision.

Show When Charging

In this exercise you'll make the plug icon change from pink to grey when the app is in the foreground when you plug and unplug the device. Debugging plugging and unplugging the device can be hard on an emulator or live phone. Because of this, you can use the Android Debug Bridge to simulate the phone being plugged and unplugged without actually doing the plugging and unplugging. This is described below.
Note: At the end of this exercise your Hydration Reminder app will contain a small bug - if the phone is plugged in or unplugged when the app is **not** open, the UI will not update. This is because the dynamic broadcast receiver only receives events when the app is in the foreground. We'll be fixing this in the next exercise.

Setup ADB

The Android Debug Bridge, or **adb** as it is affectionately called, is a command line tool. This means that you should be comfortable working in a terminal or shell to use this program. We touched on it briefly in the first lesson. The adb program is stored in your android SDK folder in a subfolder called **platform-tools**. You can find where your SDK is by going to the SDK manager and looking at the SDK location, as shown below:



Manager for the Android SDK and Tools used by Android Studio

Android SDK Location: /Users/ 'Library/Android/sdk

Each Android SDK Platform package includes the Android platform and sources pertaining to an API level by default. Once installed, Android Studio will automatically check for updates. Check "show package details" to display individual SDK components.

	Name	API Level	Revision	
✓	Android 7.1.1 (Nougat)	25	2	Partially
✓	Android 7.0 (Nougat)	24	2	Update
✓	Android N Preview	N	3	Partially
✓	Android 6.0 (Marshmallow)	23	3	Update

Once you have the sdk location, you can use adb by typing:

```
<YOUR SDK LOCATION>/platform-tools/adb
```

If you've added commands to your $PATH before, adb is a great one to add.

Helpful adb Commands

To simulate the phone being unplugged from usb charging you can use:

```
adb shell dumpsys battery set usb 0
```

or if you're on a device Android 6.0 or higher you can use:

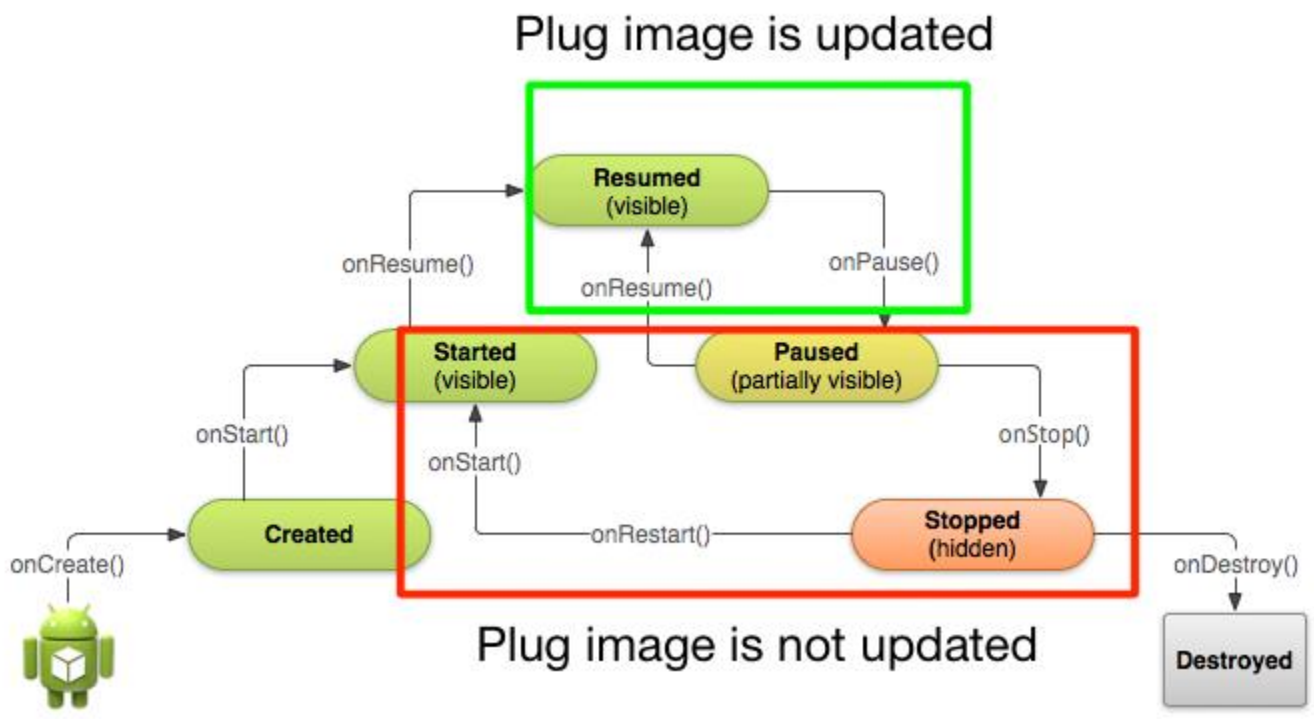
```
adb shell dumpsys battery unplug
```

To "plug" the phone back in, just reset it's charging status using:

```
adb shell dumpsys battery reset
```

Getting the Current Battery State

As mentioned, our code currently contains a bug. Our app adds and removes the dynamic broadcast receiver in `onResume` and `onPause`. When the app is not visible, the plug's image will not update. This can lead to the plug sometimes having the incorrect image when the app starts.



Now we could move the code to dynamically add and remove the broadcast receiver in different lifecycle methods, for example `onCreate` and `onDestroy`, but this would cause us to waste cycles swapping around an image which isn't even on screen. A better approach is to check what the current battery state is when the app resumes and update the image accordingly.
There are two ways to do this, depending on whether you're on API level 23+ or before.

Getting Charging State on API level 23+

To get the current state of the battery on API level 23+, simply use the battery manager system service:

```
BatteryManager batteryManager = (BatteryManager) getSystemService(BATTERY_SERVICE);
boolean isCharging = batteryManager.isCharging();
```

Getting Charging State with a Sticky Intent

Prior to Android 23+ you needed to use a sticky intent to get battery state. As we've seen, a normal, broadcasted intent will be broadcasted, possibly caught by an intent filter, and then disspear after it is processed. A sticky intent is a broadcast intent that sticks around, allowing your app to access it at any point and get information from the broadcasted intent. In Android, a sticky intent is where the current battery state is saved.

You don't need a broadcast receiver for a sticky intent, but you use similar looking code to registering a receiver:

```
IntentFilter ifilter = new IntentFilter(Intent.ACTION_BATTERY_CHANGED);
Intent batteryStatus = context.registerReceiver(null, ifilter);
```

Notice how `registerReceiver` is used, but instead of passing in a broadcast receiver, `null` is passed. The intent filter here is the intent filter for the **sticky intent** **Intent.ACTION_BATTERY_CHANGED**. The `registerReceiver` method will return an intent, and it is that intent which has all of the battery information, which you can use:

```
boolean isCharging = status == BatteryManager.BATTERY_STATUS_CHARGING || status == Ba
tteryManager.BATTERY_STATUS_FULL;
```

For more information on how to getting information about the battery, check out the **Monitoring the Battery Level and Charging State documentation**.
Now that you know how to get battery state, you should be able to complete the following exercise and fix the bug. The code is below.

Note: If you need to check whether the user is on API 23+, you can use the following code:

```
if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.M)
```