Review Material for Lesson 2

Congratulations! You've got two lessons under your belt and Sunshine is now connected to the cloud. Below is a list of the key concepts covered, along with a description of each concept.

New Concepts

- HttpURLConnection
- Logcat
- MainThread vs. Background Thread
- AsyncTask
- Adding Menu Buttons
- values/strings.xml
- Permissions
- JSON Parsing

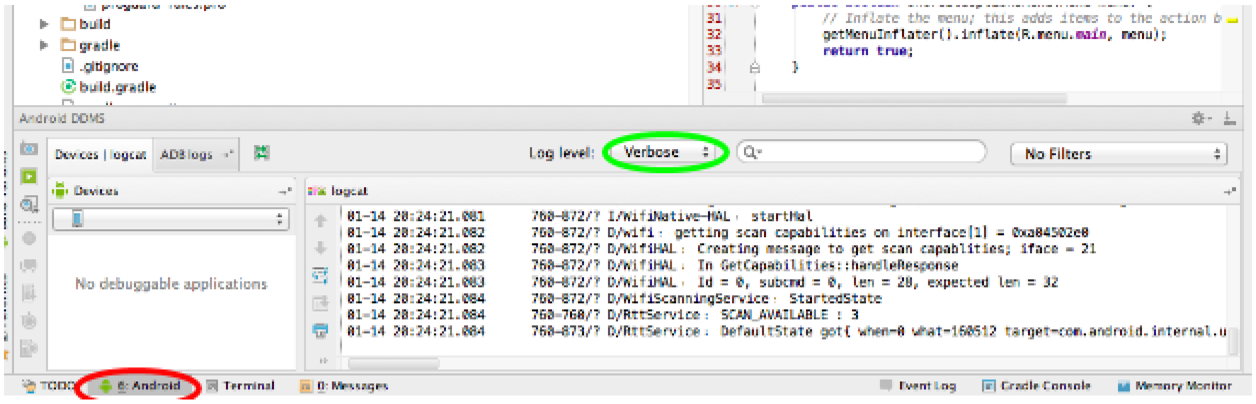
HttpURLConnect

HttpURLConnect is a Java class used to send and receive data over the web. We use it to grab the JSON data from the OpenWeatherMap API. The code to do so is introduced in this video and gist

Logcat

Logcat is the program you can use to view your Android device's logging output. In the course there are three ways to do this:

1. Android Studio has a DDMS window which includes logcat output. Make sure the Android Window is selected.



```
        // Inflate the menu; this adds items to the action b
        getMenuInflater().inflate(R.menu.main, menu);
        return true;
    }
```

Log level: Verbose

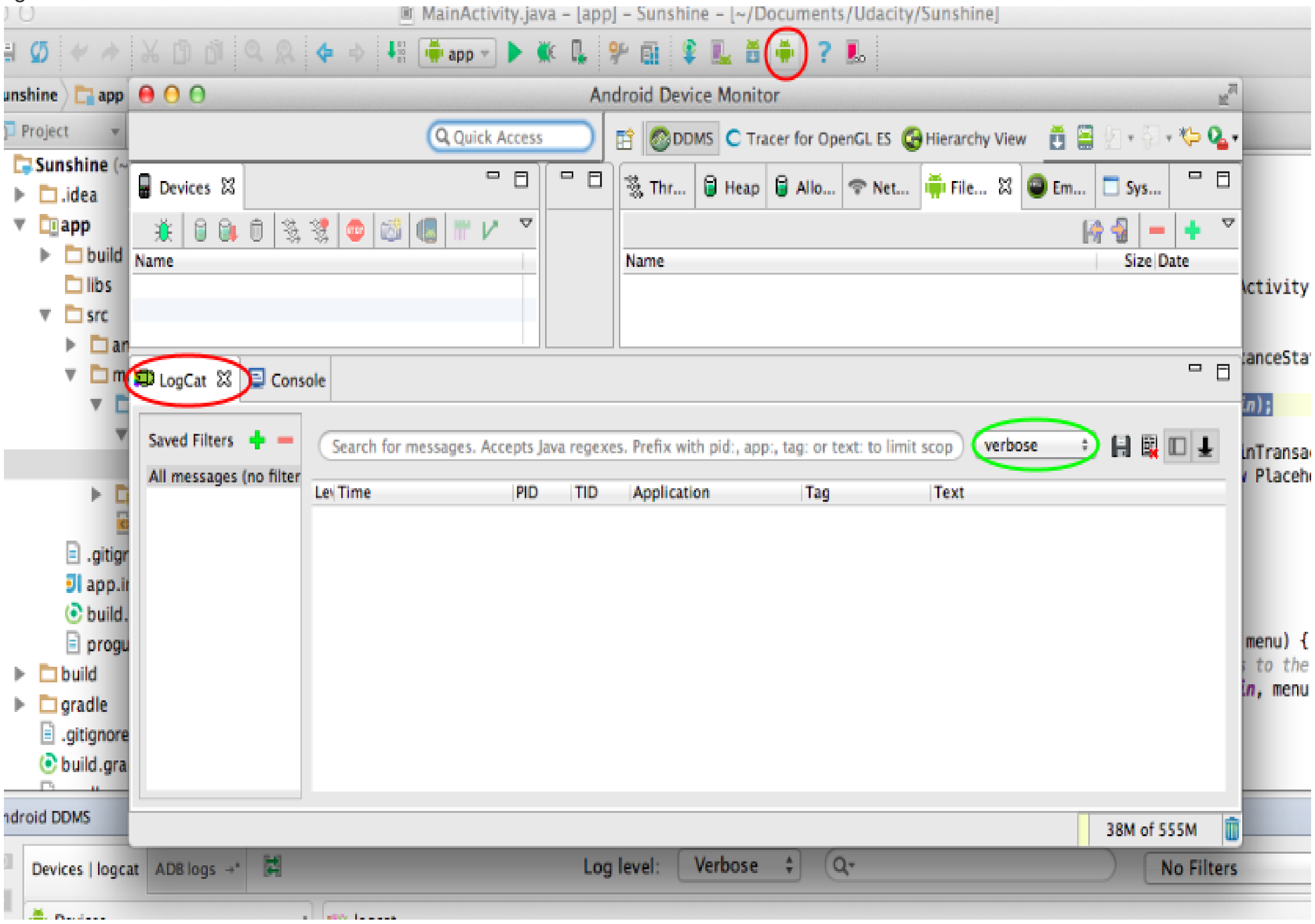
No Filters

No debuggable applications

```
01-14 20:24:21.081    760-872/? I/WifiNative-HAL﹕ startHal
01-14 20:24:21.082    760-872/? D/wifi﹕ getting scan capabilities on interface[1] = 0xa84502e8
01-14 20:24:21.082    760-872/? D/WifiHAL﹕ Creating message to get scan capablities; iface = 21
01-14 20:24:21.083    760-872/? D/WifiHAL﹕ In GetCapabilities::handleResponse
01-14 20:24:21.083    760-872/? D/WifiHAL﹕ Id = 0, subcmd = 0, len = 28, expected len = 32
01-14 20:24:21.084    760-872/? D/WifiScanningService﹕ StartedState
01-14 20:24:21.084    760-760/? D/RttService﹕ SCAN_AVAILABLE : 3
01-14 20:24:21.084    760-873/? D/RttService﹕ DefaultState got{ when=0 what=160512 target=com.android.internal.u
```

2. You can open up a terminal and type `adb logcat`. The developer guide has more information on options you can use to filter the output.

3. You can explicitly open a separate window for Android DDMS and access logcat.



Logs come in five flavours, **Verbose**, **Debug**, **Info**, **Warn** and **Error**. You can filter the logs by selecting the Log Level (shown in green), which will show you only logs of that level and above. For example, only seeing warning and above would show you Warning and Error logs.

In your code you can write statements which will post log messages to logcat. To do so, you use the Log class followed by v, d, i, w or e method, depending on which log level you want the message to be at. Log.w("",""); for example, would output a Warning level log message. Each method takes two strings, the first is the tag, which is used to identify where the log is coming from. The second parameter is the specific log message.

The convention used in the course for the **tag** is to create a String constant `LOG_TAG` which equals the name of the class the constant is in. You can get the class name programmatically. Here's an example for the MainActivity:

```
private final String LOG_TAG = MainActivity.class.getSimpleName();
```

MainThread vs. Background Thread

In Android there is a concept of the **Main Thread** or UI Thread. If you're not sure what a thread is in computer science, check out this wikipedia article. The main thread is responsible for keeping the UI running smoothly and responding to user input. It can only execute one task at a time. If you start a process on the Main Thread which is very long, such as a complex calculation or loading process, this process will try to complete. While it is completing, though, your UI and responsiveness to user input will hang.

Therefore, whenever you need to start a longer process you should consider using another “Background" thread, which doesn’t “block" the Main Thread. An easy (but by no means perfect) way to do this is to create a subclass of AsyncTask.

AsyncTask

AsyncTask is an easy to use Android class that allows you to do a task on a background thread and thus not disrupt the Main Thread. To use AsyncTask you should subclass it as we’ve done with FetchWeatherTask. There are four important methods to override:

- `onPreExecute` - This method is run on the UI before the task starts and is responsible for any setup that needs to be done.
- `doInBackground` - This is the code for the actual task you want done off the main thread. It will be run on a background thread and not disrupt the UI.
- `onProgressUpdate` - This is a method that is run on the UI thread and is meant for showing the progress of a task, such as animating a loading bar.
- `onPostExecute` - This is a method that is run on the UI **after** the task is finished.

Note that when you start an AsyncTask, it is tied to the activity you start it in. When the activity is destroyed (which happens whenever the phone is rotated), the AsyncTask you started will refer to the destroyed activity and not the newly created activity. This is one of the reasons why using AsyncTask for a longer running task is dangerous.

Adding menu buttons

So that we could add a temporary Refresh button, we learned how to add menu buttons. Here are the basic steps.

1. Add an xml file in `res/menu/` that defines the buttons you are adding, their ordering and any other characteristics.
2. If the menu buttons are associated with a fragment, make sure to call `setHasOptionsMenu(true)` in the fragment's `onCreate` method.
3. Inflate the menu in the `onCreateOptionsMenu` with a line `inflater.inflate(R.menu.forecastfragment, menu);`
4. In `onOptionsItemSelected` you can check which item was selected and react appropriately. In the case of Refresh, this means creating and executing a `FetchWeatherTask`.

values/strings.xml

Android has a specific file for all of the strings in your app, stored in `values/strings.xml`. Why? Well besides further helping separate content from layout, the strings file also makes it easy to localize applications. You simply create a `values-language/strings.xml` files for each locale you want to localize to. For example, if you want to create a Japanese version of your app, you would create a `values-ja/strings.xml`. Note, if you put the flag `translatable="false"` in your string it means the string need not be translated. This is useful when a string is a proper noun.

Permissions

By default, applications in Android are **sandboxed**. This means they have their own username, run on their own instance of the virtual machine, and manage their own personal and private files and memory. Therefore, to have applications interact with other applications or the phone, you must request permission to do so.

Permissions are declared in the `AndroidManifest.xml` and are needed for your app to do things like access the internet, send an SMS or look at the phone's contacts. When a user downloads your application, they will see all of the permissions you request. In the interest of not seeming suspicious, it's a good idea to only request the permissions you need.

JSON Parsing

Often when you request data from an API this data is returned in a format like JSON. This is the case for Open Weather Map API. Once you have this JSON string, you need to parse it.

If you're unfamiliar with JSON, take a look at this tutorial.

If you're not sure of the exact structure of the JSON, use a formatter. Here's a good one you can use in the browser.

In Android, you can use the `JSONObject` class, documented here. To use this class you take your JSON string and create a new object:

```
JSONObject myJson = new JSONObject(myString);
```

And then you can use various `get` methods to extract data, such as `getJSONArray` and `getLong`.

Instructor Notes

Important:

The API now **requires** you to have an API key. Learn how to generate an API key here.

Open Weather Map has redesigned its website since the filming of this video (don't you hate when that happens?), so you may notice a few discrepancies between the site and what you see in the video.

Here's a link to some helpful example API calls.
Weather condition codes

The API now **requires** you to have an API key. Learn how to generate an API key here.

Open Weather Map has redesigned its website since the filming of this video (don't you hate when that happens?), so you may notice a few discrepancies between the site and what you see in the video.

Note: You can request weather data by postal code by including it as the city name. You might need to add a comma and the country as well if it gives you a foreign city (ex 94040, USA).

Check this documentation for how to call 7-day forecasts.

Here's a link to some helpful example API calls.
Weather condition codes

Connecting to the Network training guide

Check out this blog post on the Android HTTP clients!

If you're overwhelmed by logs, you can stop ADB logcat while it's running by hitting CTRL+Z.

You can use Monitor or, if you're a command line guru, ADB logcat to view logs.

Reading and Writing Logs

Note: You may have noticed there's a typo in this solution video! The correct answer should read "NetworkOnMainThreadException"

If you can't find the logcat tab in Android Device Monitor, perhaps it is on the bottom third of the screen, at which point, you can drag the logcat window to the main area to make it appear like in the video.

Also, in logcat, you can search for more than just the process ID, for example, you can search for class name like "MainActivity" to narrow down the logs even further.

To display line numbers in the editor, right click on the gray bar beside the code, and select "Show Line Numbers."

API guide on processes and threads

AsyncTask documentation

Actually a more robust way to change the name of the PlaceholderFragment is to use the "Refactor" tool provided by Android Studio. Right click on the class name PlaceholderFragment > Refactor > Rename > type in ForecastFragment and then all relevant references will be updated.

Also, when you move ForecastFragment to its own java file, make sure it's not a static class anymore.

AsyncTask reference
Networking code snippet

We're taking this one step at a time, so just make the menu XML file change in this exercise. If you compile and run the app, there will be no visible change. That's because we haven't inflated the menu item. Inflating it is in the next coding task.

Menu training guide
Declaring string resources

If you have a more recent version of Android Studio, ForecastFragment.java will NOT be automatically created for you. Please not that all Java code in this Node happens in a file called ForecastFragment.java, NOT in MainActivity.java!

Guide to Options Menu

If you're confused about OnCreate and the other new methods introduced here, check out this documentation. Reto will discuss fragment lifecycle methods more in Lesson 5.

Fragment.setHasOptionsMenu(boolean) method

This is what your ForecastFragment.java code should look like at this step. Again, you may need to create this file, but you should not need to edit MainActivity.java. You may need to insert methods like onCreate, onCreateOptionsMenu, et cetera, in ForecastFragment.java if they aren't already there.

For more info on the Fragment lifecycle methods, see the documentation.

Permissions in Android M

In the Android M developer preview, Google introduced a new model of permissions that does not exactly match the model presented in the last video.

The big difference is the introduction of runtime permissions. With Android M, the user doesn't choose to grant permissions at install time. Instead, the user is prompted for the permission when the application needs to use it. The application can choose to prompt the user for multiple permissions at once, and can add a justification for why the permission is needed.

In order to avoid overloading the user with permission choices, Android has added Permission Groups. With permission groups, users say yes or no to a group of permissions, rather than individual permissions. For instance, rather than prompting the user for both the ability to send a text message and the ability to receive a text message, when we ask for the SEND_SMS permission, the user is prompted to give our app access to SMS generally.

What does this mean for me?

When targeting Marshmallow or M preview devices, you'll need to be much more careful about checking for permissions each time you make use of them. A user could say yes to your app generally, but no to it's use of their GPS, for instance. It will be your job to check whether the user granted the permission so you can gracefully handle the case where they did not.

You'll also want to make sure to not store any sort of variable to keep track of whether a permisison has been granted. In the new system, your app is not notified when a permission is revoked, so you'll always need to check with the system, using the Context.checkSelfPermission() method.

For more information on permissions, visit this documentation.

See the security section of the Android Developers site

Android documentation! For permission in Manifest

UriBuilder

If you need a reminder of what the mock looks like, check it out here.

Visual Mocks and Redlines

- PDF of all Visual Mocks and Redlines

  ### Individual PNG files

  - Phone MainActivity
  - Phone Today DetailActivity
  - Phone DetailActivity
  - Tablet MainActivity
  - Phone MainActivity Redlines
  - Phone DetailActivity Redlines
  - Tablet MainActivity Redlines

Sunshine Wireframes (Dan's Sketches)

Images

- Placeholder launcher icon
- Assets to include in app

JSONObject in Android

AsyncTask generic types

Hint: Make mForecastAdapter a global variable (in ForecastFragment). That way you can still access mForecastAdapter within FetchWeatherTask.

Hint: Make sure FetchWeatherTask is not defined as a static class. If you're wondering why, see this link on nested classes in Java.

AsyncTask
ArrayAdapter

The onPostExecute(String[] result) method is being overridden in the FetchWeatherTask class (which is an inner class in ForecastFragment.java)

AsyncTask

ArrayAdapter

Adapter section from Lesson 1
ArrayAdapter source code in Android framework
