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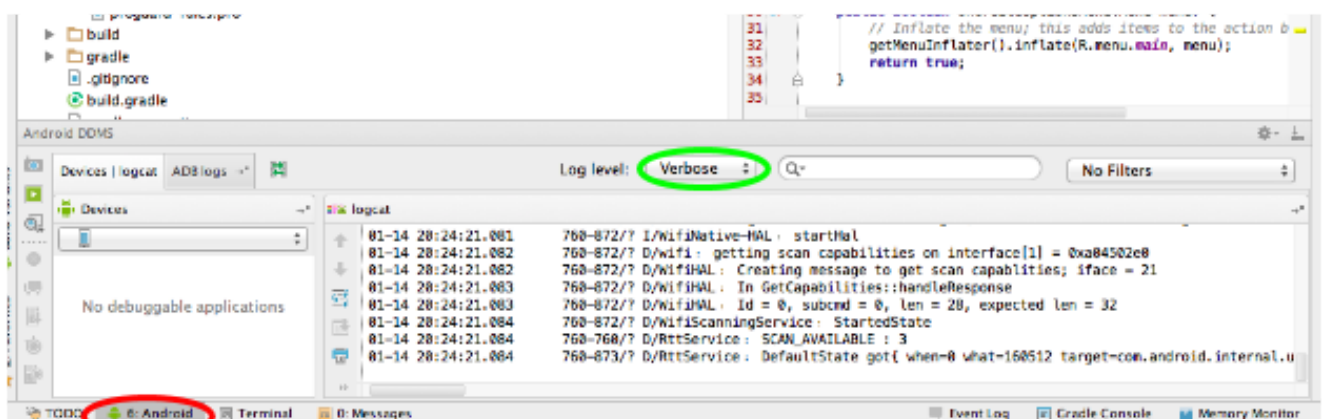
2. You can open up a terminal and type `adb logcat`. The developer guide has more information on options you can use to filter the output.

3. You can explicitly open a separate window for Android DDMS and access logcat.
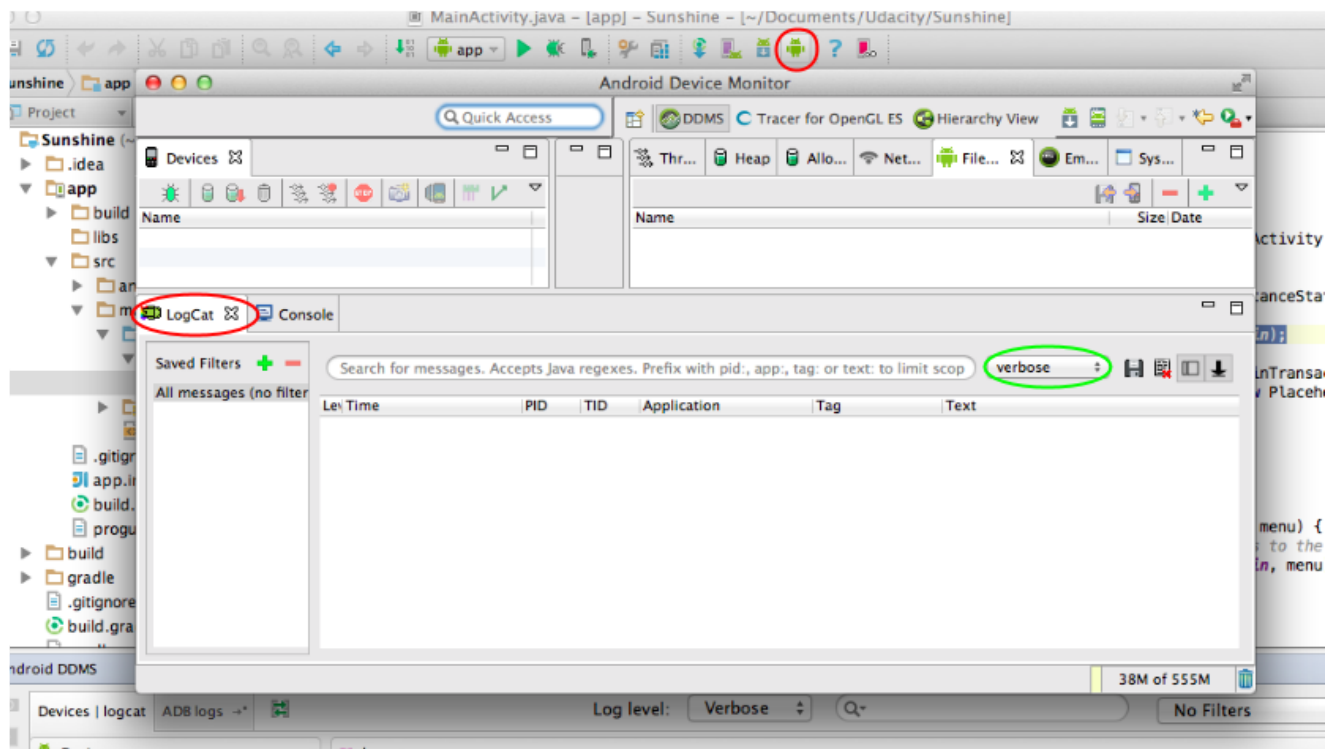
Logs come in five flavours, **Verbose**, **Debug**, **Info**, **Warn** and **Error**. You can filter the logs by selecting the Log Level (shown in green), which will show you only logs of that level and above. For example, only seeing warning and above would show you Warning and Error logs.

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`.