Lecture Notes: AsyncTasks

1. Threads
    a. How a computer does multiple things at the same time
    b. Managed by a thread scheduler built into the operating system
2. The Android main (or UI) thread
    a. You must not block it, or the app will become unresponsive
    b. We don't expect a delay when we scroll, push a button, or type in a text field
    c. Long running operations (including anything that accesses the web, makes a DB call, reads a file, etc.) must be done on a separate thread
3. Various thread abstractions exist to make it easy to do certain kinds of things in a separate thread
4. AsyncTask is a thread abstraction that allows you to start a process and be notified when it finishes (and as it progresses if you want to see progress)
    a. See AsyncWebAccess example
        i. Generic parameters: 1. doInBackground param type, 2. onProgressUpdate param type, 3. doInBackground result and onPostExecute param type.
        ii. doInBackground(…) happens on a different thread.
        iii. onProgressUpdate(…) and onPostExecute(…) happen on the main (UI) thread.
        iv. Need Internet permission (<uses-permission android:name="android.permission.INTERNET" />)
            1. Declare in AndroidManifest.xml
        v. If you are using Android OS 9 or later on your emulator (most of you probably are) you need to enable clear text passwords by placing the following code in the opening <application> tag of your AndroidManifest.xml as follows:

```
<application
    ...
    android:usesCleartextTraffic="true"
```

…

>

vi. The inner class task has easy access to the outer class' UI elements to update them.

vii. **Problem:** task code isn't reusable as an inner class

viii. **Solution:** make the task class a top-level class

1. **New Problem:** the top-level task class doesn't have access to the Activity's UI elements, so it can't update them.
2. **Solution:** make the Activity a listener of the Task, so it can be notified of progress updates and notified when the task is finished. This is an example of the Observer design pattern.

b. See AsyncWebAccessWithListener example

i. Task is now a top-level class that can be instantiated and re-used from anywhere.

ii. The task provides a listener interface (DownloadTaskListener) and allows classes that implement the interface to register themselves with the task.

iii. When events (progress updates and task completion) happen in the task, the task notifies its listeners so they can respond.

iv. The Activity implements the DownloadTaskListener interface and updates it's UI in response to it's listener methods being called.

v. This is better, but you can get full credit with the simplified version of making your task an inner class where it has direct access to the UI elements that need to be updated.

5. Family Map Application

a. Will use one AsyncTask to register, one to login, and one to sync data

b. Sync will be called from register and login

c. AsyncTasks will call ServerProxy methods to access server data

d. Use onPostExecute(…) callback in the task to know when the task is done

i. Invoke listener methods if implementing the listener version
ii. Handle UI updates from onPostExecute(…) if using inner class version.