AsyncTask


Repo: https://github.com/karlmorris/AndroidThreads/tree/AsyncTask
Overview

- A review of threads
- Simplified thread and handler management using AsyncTask
- Features, example, and proper use of AsyncTask
AsyncTask

• Threads and handlers are low level
  – Require more code to manage properly
• Threads are good for certain operations, but not all
AsyncTask cont.

- AsyncTask provides an abstraction of threads and handlers
- Handles many coordination and communication concerns
- Allows worker thread to provide updates on progress
AsyncTask operations

- **onPreExecute()**
  - Invoked on the UI thread before the task is executed

- **doInBackground(Params...)**
  - Invoked on the background thread immediately after onPreExecute() finishes executing.

- **onProgressUpdate(Progress...)**
  - Invoked on the UI thread after a call to publishProgress(Progress...).

- **onPostExecute(Result...)**
  - Invoked on the UI thread after the background computation finishes.
What is AsyncTask good at

• Short lived tasks
• When you need to communicate with UI thread
• When you don't need or want to manipulate handlers
What is AsyncTask bad at

- Long running tasks
  - Not tied to Activity life cycle
  - Can cause memory leaks
private class DownloadFilesTask extends AsyncTask<URL, Integer, Long> {
    protected Long doInBackground(URL... urls) {
        int count = urls.length;
        long totalSize = 0;
        for (int i = 0; i < count; i++) {
            totalSize += Downloader.downloadFile(urls[i]);
            publishProgress((int) ((i / (float) count) * 100));
        }
        return totalSize;
    }

    protected void onProgressUpdate(Integer... progress) {
        setProgressPercent(progress[0]);
    }

    protected void onPostExecute(Long result) {
        showDialog("Downloaded "+ result + " bytes");
    }
}
AsyncTask Example

Execute using:

```java
new DownloadFilesTask().execute(url1, url2, url3);
```
Proper Usage

- Must be loaded on the UI thread
- `execute()` must be called on the UI thread
- Do not call any of the four operational methods directly
- Only execute a task once
Considerations

- Recent versions of Android execute tasks sequentially (on a single worker thread)
- If desired, tasks can be run in parallel by calling `executeOnExecutor()`
- A task can be canceled with a call to `cancel()`
  - `doInBackground()` continues to run, but `onCancelled()` will be invoked instead of `onPostExecute()` afterwards
  - You can exit background task sooner by checking `isCancelled()`