Université Mohammed V
FACULTE DES SCIENCES
RABAT / FSR
Département informatique

Master IAO
Master II–Semestre 3
Cours

Mobile & Cloud Computing

Pr. REDA Oussama Mohammed

2015/2016

AsyncTask

Multi-threading in Android

Android AsyncTasks

The **AsyncTask** class allows you to run **non-UI Threads that interact with the UI thread** in a well defined way.

This class allows to **perform background operations** and **publish results on the UI thread without** having to **manipulate other threads** and/or handlers.

AsyncTasks are given high priority, so they should ideally be used for short operations (**a few seconds at the most**). If you need to keep threads running for long periods of time, it is highly recommended you use the various APIs provided by the java.util.concurrent package such as Executor, ThreadPoolExecutor and FutureTask.

Android AsyncTasks

The AsyncTask is designed to be a helper class, and there are four common methods you will often need to implement:

- onPreExecute()
- doInBackground(Params… values)
- onProgressUpdate(Progress... values)
- onPostExecute(Result result)

AsyncTasks API

```
Something doInBackground(String... params) {
        return null;
}

protected void onPreExecute() {
}

protected void onPostExecute(Something b) {
}

protected void onProgressUpdate(Type... values) {
}
```

Android AsyncTasks

1. **onPreExecute()** – This *calls on the UI thread before the thread starts* running. This method is usually *used to setup the task*, for example by *displaying a progress bar*.

2. **doInBackground(Params… values)** – this is the method that *runs on the background thread*. In this method you should *put all the code you want the application to perform in background*. The doInBackground() is *called* immediately *after onPreExecute()*. When it finishes, it *sends the result to the onPostExecute()*.

Android AsyncTasks

3. **onProgressUpdate(Progress... values)** - called when you invoke ***publishProgress()*** in the doInBackground(). *Runs on the UI thread*.

4. **onPostExecute(Result result)** – called *on the UI thread* after the background thread finishes. It takes *as parameter the result* received *from doInBackground()*.

Android: AsyncTask

AsyncTask is a Thread helper class (Android only).

- ✧ Computation running on a **background** thread.
- ✧ Results are published on the **UI** thread.

RULES

- ➢ AsyncTask must be created on the UI thread.
- ➢ AsyncTask can be executed only once.
- ➢ AsyncTask must be canceled to stop the execution.

How Android threading works with *AsyncTasks*

- Create a class that extends **AsyncTask**
- To start the new thread, call the AsyncTask's ***execute*** method

- When ***execute*** is called, Android does the following:
    1. runs *onPreExecute* in the main (UI) thread
    2. runs *doInBackground* in a background thread
    3. runs *onPostExecute* in the main (UI) thread

Android: AsyncTask

```
private class MyTask extends AsyncTask<Par, Prog, Res>
```

Par → type of parameters sent to the AsyncTask
Prog → type of progress units published during the execution
Res → type of result of the computation

EXAMPLES

```
private class MyTask extends AsyncTask<Void,Void,Void>
```

```
private class MyTask extends AsyncTask<Integer,Void,Integer>
```

Creating an AsyncTasks

You create an AsyncTask object by first creating a class derived from the AsyncTask class. The derived class must use generics to identify the parameter data types to the doInBackground, onProgressUpdate, and onPostExecute methods respectively like so:

private class *MyTask* extends AsyncTask<String, Void, String>

In the class you create, you add code for whatever AsyncTask methods you want to use. Then after you have added this class as a subclass in your Activity or View code, you do the following to run the AsyncTask at some other place in your code, and you pass it whatever values (or array of values) you wish to pass to it:

new MyTask().*execute*(outgoingLine);

Android: AsyncTask

EXECUTION of the ASYNCTASK

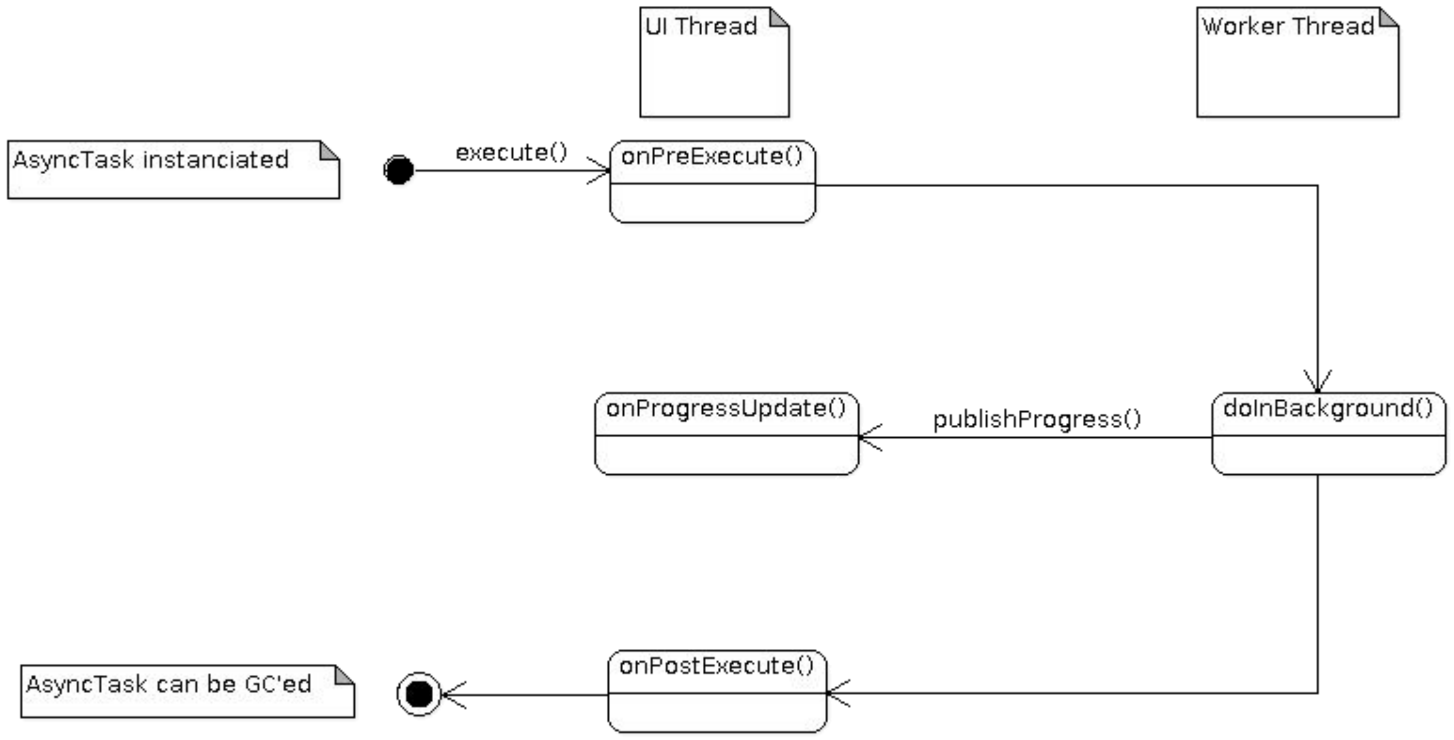
The UI Thread invokes the **execute** method of the AsyncTask:

(new Task()).**execute**(param1, param2 … paramN)

After **execute** is invoked, the task goes through four steps:

1. **onPreExecute**() → invoked on the UI thread
2. **doInBackground**(Params…) →computation of the AsyncTask
   - ✧ can invoke the publishProgress(Progress…) method
3. **onProgressUpdate**(Progress …) → invoked on the UI thread
4. **onPostExecute**(Result) → invoked on the UI thread

Asynchronous Tasks : the lifecycle



See GoogleSearch

AsyncTask as Abstraction

```
private class DownloadFilesTask extends AsyncTask<URL, Integer, Long>
{
     protected Long doInBackground(URL... urls) { // on some background
thread
        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) { // on UI thread!
         setProgressPercent(progress[0]);
    }
    protected void onPostExecute(Long result) { // on UI thread!
         showDialog("Downloaded " + result + " bytes");
    }
}
```

new DownloadFilesTask().execute(url1, url2, url3); // call from UI thread!

Université Mohammed V
FACULTE DES SCIENCES
RABAT / FSR
Département informatique

Master IAO
Master II–Semestre 3
Cours

Mobile & Cloud Computing

Pr. REDA Oussama Mohammed

2015/2016