Android Worker Threads within a GUI

# Introduction:

By default, the User Interface in the android framework is controlled by one thread, often referred to as the UI thread or the “main” thread of execution. It is responsible for everything the user sees on the screen while interacting with the application. This is not ideal as operations which take longer to compute will block the UI making our app unresponsive for the user and force them to wait until the operation finishes or even worst, trigger the Application Not Responding notification, which allows the user to close our application. We can avoid this with the user of worker threads.

# What are worker threads?

Worker threads have been introduced in order to take the load off the UI thread which is responsible for controlling the UI. Having all of our operations on the UI thread can have detrimental effects to the performance of our application. All the time intensive tasks such as querying the database, carrying out complex calculation or performing network functionality would mean that the GUI that the user is interacting with, would appear to be hanging as it becomes unresponsive to input until the operation finishes.

With game development it is essential to carry out all of the work and calculations in a worker thread, whilst the UI thread should focus on the graphical aspect of our application such as drawing animations and responding to user input.

Worker threads are basically threads which work in the background of our application, they are referred to as “worker” as they carry out work which the user can’t see and they are not allowed to interact with the UI directly, only objects which are part of the UI have access to the thread, an example of this would be the View objects.

In order to allow the background threads to communicate with the UI thread, the Android framework provides us with a few useful tools in order to allow communication between the worker threads as well as the UI thread. The most common one is to use a Handler. Handlers are responsible for thread management and allow messages to be exchanged between the worker thread as well as the UI thread.

# AsyncTask:

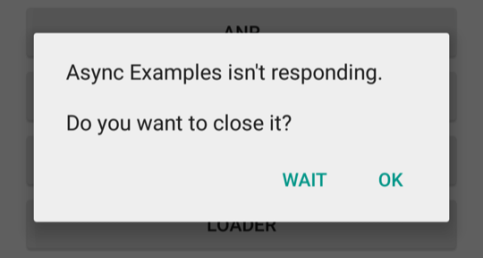
A common alternative to using Handlers is the use of AsyncTask subclass which provides us with a set of useful methods which make it even easier for background threads to communicate with the UI thread and exchange messages.   
The process of passing on a message from the worker thread onto the UI thread is as simple as using a few methods which I will briefly cover below:

**doInBackground():**   
This is where all of the work in our worker thread is carried out, all of the complex calculations, database queries and other time intensive operations would go here. Whenever we would want to communicate with the UI thread and update it with the progress from our operation, we call the **publishProgress()** method from within this method which invokes the **onProgressUpdate()**, this is where we specify the reaction to our method call, this could be as simple as updating the progress bar.

**onPostExectute():**This method would be called after all of our time intensive background work has been finished, inside of this method we would specify the action carried out after that has taken place. It could be some meaningful change in UI or as simple as displaying a little message.

For example, if we used the above described structure to represent a user downloading an application of the web, all the intensive processing would go iniside of the doInBackground() method, which would allow the user to still interact with the UI. Once in a while the publishProgress() method would be called to update the UI on the progress of that activity, could be as simple as updating the progress bar. Finally the onPostExecute() the UI could display a little window notifying the user that the file has finished downloading.

# Application Not Responding (ANR):

Android has a system guard implemented against unresponsive applications which hang for extensive amounts of time, making the user wait.

Generally the ANR dialog box will appear after the application has been unresponsive to user input for about 5 seconds; it will then throw up the dialog box asking the user whether they want to close the application as it is not responding.

In order to avoid the ANR dialog boxes from appearing on our application it is vital to make sure that the operations do not block the UI thread which would result in the GUI being unresponsive.

The use of worker threads allows us to minimize the risk of our application being sluggish or unresponsive and having to deal with the dreaded ANR dialog box. On top of using worker threads alongside our UI thread it is important to make sure that the workload that is carried out directly in our UI thread takes as little processing time as possible.

# Sources:

<https://developer.android.com/guide/components/processes-and-threads.html#WorkerThreads>

<https://developer.android.com/training/articles/perf-anr.html>

<https://developer.android.com/reference/android/os/AsyncTask.html>

<https://developer.android.com/images/anr.png>