# Android and AsyncTask

AsyncTask enables proper and easy use of the UI thread. This class allows you to perform background operations and publish results on the UI thread without having to manipulate threads and/or handlers.

AsyncTask is designed to be a helper class around Thread and Handler and does not constitute a generic threading framework. AsyncTasks 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.

An asynchronous task is defined by a computation that runs on a background thread and whose result is published on the UI thread. An asynchronous task is a generic class defined by 3 generic types[[1]](#footnote-1), called Params, Progress and Result, and 4 steps, called onPreExecute, doInBackground, onProgressUpdate and onPostExecute.

## Usage

AsyncTask must be subclassed to be used. The subclass will override at least one method (doInBackground(Params...)), and most often will override a second one (onPostExecute(Result).), but sometimes the method onPreExecute() is used and the method onProgressUpdate(Progress) too.

Here is an example of subclassing:

**private class** AsyncTaskRunner **extends** AsyncTask<Integer, String, String> {  
 String **resp**;  
 @Override  
 **protected** String doInBackground(Integer... params) {  
 . . .

**return resp**;  
 }  
  
 @Override  
 **protected void** onPostExecute(String result) {  
 *. . .*

}  
  
 @Override  
 **protected void** onPreExecute() {

**. . .**

}  
  
 @Override  
 **protected void** onProgressUpdate(String... text) {  
 **. . .**  
 }  
}

Once created, a task is executed very simply:

 new AsyncTaskRunner.execute(…an integer parameter here in this case…);

## AsyncTask's generic types

The three types used by an asynchronous task are the following:

1. Params, the type of the parameters sent to the task upon execution.
2. Progress, the type of the progress units published during the background computation.
3. Result, the type of the result of the background computation.

Not all types are always used by an asynchronous task. To mark a type as unused, simply use the type Void:

 private class AsyncTaskExample extends AsyncTask<Void, Void, Void> { ... }

## The 4 steps

When an asynchronous task is executed, the task goes through 4 steps:

1. onPreExecute(), invoked on the UI thread before the task is executed. This step is normally used to setup the task, for instance by showing a progress bar in the user interface.
2. doInBackground(Params...), invoked on the background thread immediately after onPreExecute() finishes executing. This step is used to perform background computation that can take a long time. The parameters of the asynchronous task are passed to this step. The result of the computation must be returned by this step and will be passed back to the last step. This step can also use publishProgress(Progress...) to publish one or more units of progress. These values are published on the UI thread, in the onProgressUpdate(Progress...) step.
3. onProgressUpdate(Progress...), invoked on the UI thread after a call to publishProgress(Progress...). The timing of the execution is undefined. This method is used to display any form of progress in the user interface while the background computation is still executing. For instance, it can be used to animate a progress bar or show logs in a text field.
4. onPostExecute(Result), invoked on the UI thread after the background computation finishes. The result of the background computation is passed to this step as a parameter.

## Cancelling a task

A task can be cancelled at any time by invoking cancel(boolean). Invoking this method will cause subsequent calls to isCancelled() to return true. After invoking this method, onCancelled(Object), instead of onPostExecute(Object) will be invoked after doInBackground(Object[]) returns. To ensure that a task is cancelled as quickly as possible, you should always check the return value of isCancelled() periodically from doInBackground(Object[]), if possible (inside a loop for instance.)

## Threading rules

There are a few threading rules that must be followed for this class to work properly:

* The AsyncTask class must be loaded on the UI thread.
* The task instance must be created on the UI thread.
* execute(Params...) must be invoked on the UI thread.
* Do not call onPreExecute(), onPostExecute(Result), doInBackground(Params...), onProgressUpdate(Progress...) manually.
* The task can be executed only once (an exception will be thrown if a second execution is attempted.)

### Android AsyncTask Example

To start an AsyncTask the following snippet must be present in the MainActivity class:

**runner** = **new** AsyncTaskRunner();  
**runner**.execute(time);

In the above snippet we’ve used a simple classname that extends AsyncTask and execute method is used to start the background thread.

In this example we’ll implement an AsyncTask that makes a process to go to sleep for a given period of time as set by the user.

We define this period of time in a edit field of UI and send the value of it as parameter

**runner** = **new** AsyncTaskRunner();  
String sleepTime = **time**.getText().toString();  
**int** time = Integer.*parseInt*(sleepTime)\*1000;  
**runner**.execute(time);

The layout activity\_main is definied as follows:

<**RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"  
 xmlns:tools="http://schemas.android.com/tools"  
 android:layout\_width="match\_parent"  
 android:layout\_height="match\_parent"  
 tools:context=".MainActivity"** >  
  
 <**TextView  
 android:id="@+id/tv\_time"  
 android:layout\_width="wrap\_content"  
 android:layout\_height="wrap\_content"  
 android:textSize="10pt"  
 android:textColor="#444444"  
 android:layout\_alignParentLeft="true"  
 android:layout\_marginRight="9dip"  
 android:layout\_marginTop="20dip"  
 android:layout\_marginLeft="10dip"  
 android:text="Backgroung in Seconds:"**/>  
 <**EditText  
 android:id="@+id/in\_time"  
 android:layout\_width="150dip"  
 android:layout\_height="wrap\_content"  
 android:background="@android:drawable/editbox\_background"  
 android:layout\_toRightOf="@id/tv\_time"  
 android:layout\_alignTop="@id/tv\_time"  
 android:inputType="number"** />  
 <**Button  
 android:id="@+id/btn\_run"  
 android:layout\_width="wrap\_content"  
 android:layout\_height="wrap\_content"  
 android:text="Run Async task"  
 android:layout\_below="@+id/in\_time"  
 android:layout\_centerHorizontal="true"  
 android:layout\_marginTop="64dp"** />  
 <**Button  
 android:id="@+id/btn\_cancel"  
 android:layout\_width="wrap\_content"  
 android:layout\_height="wrap\_content"  
 android:text="Cancel Async task"  
 android:layout\_below="@+id/btn\_run"  
 android:layout\_centerHorizontal="true"  
 android:layout\_marginTop="64dp"** />  
  
  
 <**TextView  
 android:id="@+id/tv\_result"  
 android:layout\_width="wrap\_content"  
 android:layout\_height="wrap\_content"  
 android:textSize="7pt"  
 android:layout\_alignParentBottom="true"  
 android:layout\_centerHorizontal="true"** />  
 <**ProgressBar  
 android:id="@+id/determinateBar"  
 style="?android:attr/progressBarStyleHorizontal"  
 android:layout\_width="wrap\_content"  
 android:layout\_height="wrap\_content"  
 android:layout\_below="@id/btn\_cancel"  
 android:layout\_marginLeft="23dp"  
 android:layout\_marginTop="20dp"  
 android:indeterminate="false"  
 android:max="100"  
 android:minHeight="50dp"  
 android:minWidth="300dp"  
 android:progress="1"** />  
  
</**RelativeLayout**>

The MainActivity.java is defined below:

**package** . . .;  
  
**import** android.os.AsyncTask;  
**import** android.os.Bundle;  
**import** android.support.v7.app.AppCompatActivity;  
**import** android.view.View;  
**import** android.widget.Button;  
**import** android.widget.EditText;  
**import** android.widget.ProgressBar;  
**import** android.widget.TextView;  
  
**public class** MainActivity **extends** AppCompatActivity {  
 **private** Button **buttonrun**,**buttoncancel**;  
 **private** EditText **time**;  
 **private** TextView **finalResult**;  
 **private** ProgressBar **progressbar**;  
 AsyncTaskRunner **runner**;  
  
 @Override  
 **protected void** onCreate(Bundle savedInstanceState) {  
 **super**.onCreate(savedInstanceState);  
 setContentView(R.layout.***activity\_main***);  
 **progressbar**=(ProgressBar)findViewById(R.id.***determinateBar***);  
 **time** = (EditText) findViewById(R.id.***in\_time***);  
 **buttonrun** = (Button) findViewById(R.id.***btn\_run***);  
 **buttoncancel** = (Button) findViewById(R.id.***btn\_cancel***);  
 **finalResult** = (TextView) findViewById(R.id.***tv\_result***);  
 **buttonrun**.setOnClickListener(**new** View.OnClickListener() {  
 @Override  
 **public void** onClick(View v) {  
 **runner** = **new** AsyncTaskRunner();  
 String sleepTime = **time**.getText().toString();  
 **int** time = Integer.*parseInt*(sleepTime)\*1000;  
 **runner**.execute(time);  
 }  
 });  
 **buttoncancel**.setOnClickListener(**new** View.OnClickListener() {  
 @Override  
 **public void** onClick(View v) {  
 **runner**.cancel(**false**);  
 }  
 });  
 }  
  
 **private class** AsyncTaskRunner **extends** AsyncTask<Integer, Integer, String> {  
  
 **private** String **resp**;  
 **private boolean processfinish**=**true**;  
  
  
 @Override  
 **protected** String doInBackground(Integer... params) {  
 **try** {  
 **long** totalSize = 0;  
 **while**(totalSize<=100) {  
 Thread.*sleep*(params[0]/10);  
 publishProgress((**int**) (totalSize));  
 totalSize += 10;  
 **if** (isCancelled()){  
 **processfinish**=**false**;  
 **break**;  
 }  
 }

**if**(**processfinish**==**false**)  
 **resp**=**"Canceled"**;  
 **else  
 resp**=**"Slept for "** + (**double**)params[0]/1000/10**"** +

**" miliseconds 10 times"**;

} **catch** (InterruptedException e) {  
 e.printStackTrace();  
 **resp** = e.getMessage();  
 } **catch** (Exception e) {  
 e.printStackTrace();  
  
 **resp** = e.getMessage();  
 }  
 **return resp**;  
 }  
 @Override  
 **protected void** onPreExecute() {  
 **finalResult**.setText(**"Wait for....."**+ **time**.getText()+**"seconds"** );  
 }  
  
 @Override  
 **protected void** onPostExecute(String result) {  
 *// execution of result of Long time consuming operation* **finalResult**.setText(result);  
  
 }  
  
 @Override  
 **protected void** onProgressUpdate(Integer... progress) {  
 **progressbar**.setProgress(progress[0]);  
 }  
  
 @Override  
 **protected void** onCancelled(String result) {  
 **finalResult**.setText(result);  
  
 }  
 }  
}

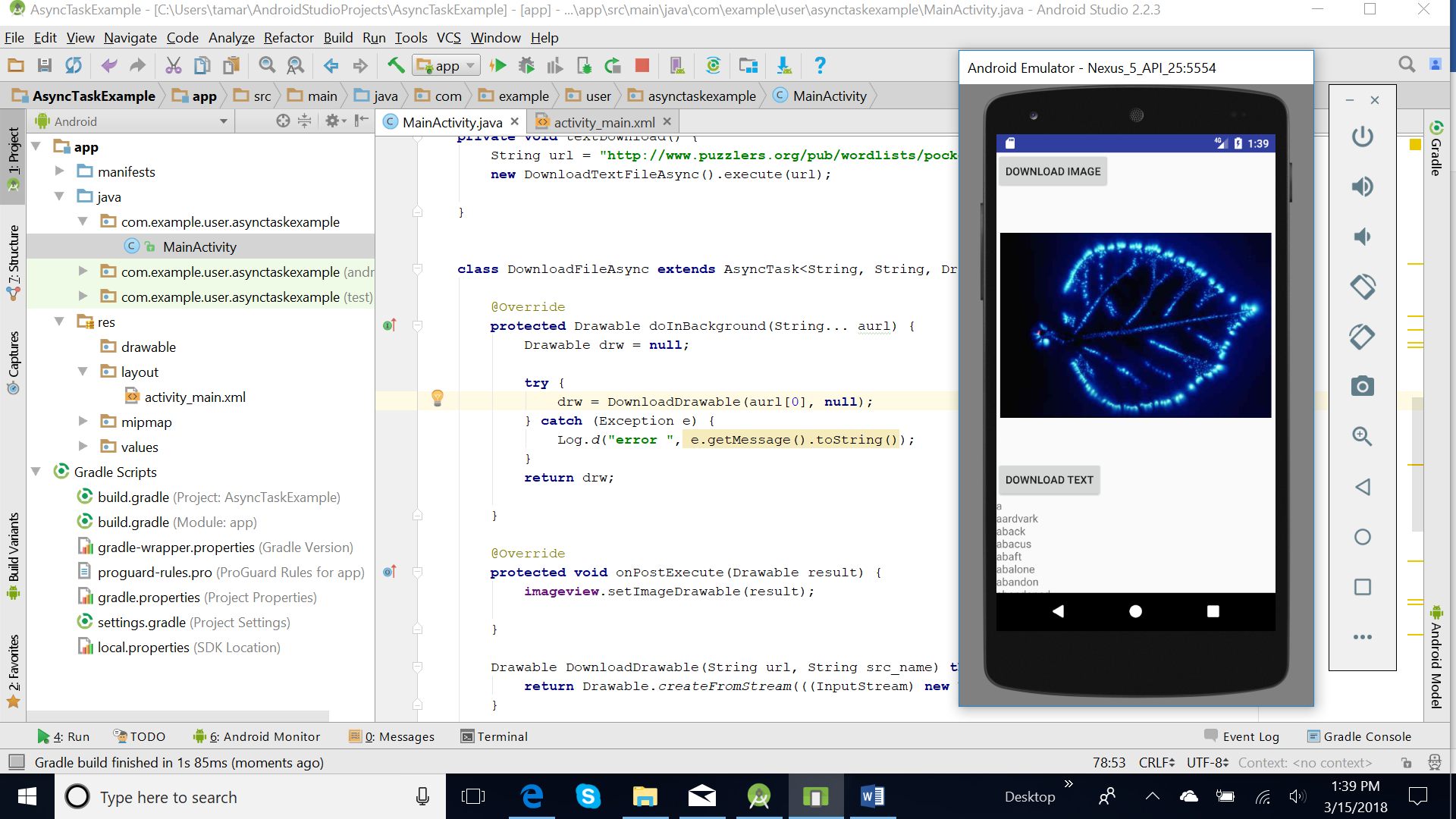
Notes:

* The java.lang.Thread.sleep(long millis)method causes the currently executing thread to sleep for the specified number of milliseconds, subject to the precision and accuracy of system timers and schedulers.
* In the above code we’ve used AsyncTaskRunner class to perform the AsyncTask operations. The time in seconds is passed as a parameter to the class and a ProgressBar is displayed for the given amount of time.
* The images given below are the outputs produced by the project where the time set by the user is 5 seconds.

|  |  |
| --- | --- |
|  |  |

## Another Example, Class Work

Let work on a project in Android, where we have to download an image in remote. The first part in the picture.



The Layout of the Main Activity:

<**RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"  
 xmlns:tools="http://schemas.android.com/tools"  
 android:layout\_width="match\_parent"  
 android:layout\_height="match\_parent"** >  
  
 <**Button  
 android:id="@+id/MyButton"  
 android:layout\_width="wrap\_content"  
 android:layout\_height="wrap\_content"  
 android:layout\_alignParentTop="true"  
 android:text="Download Image"** />  
  
  <**ImageView  
 android:id="@+id/imageView1"  
 android:layout\_width="350dp"  
 android:layout\_height="350dp"  
 android:layout\_centerHorizontal="true"  
 android:layout\_below="@id/MyButton"**/>  
  
 <**TextView  
 android:id="@+id/text\_file"  
 android:layout\_width="fill\_parent"  
 android:layout\_height="fill\_parent"  
 android:layout\_below="@id/MyButton1"  
 android:text="The caption"**/>  
</**RelativeLayout**>

And the code of the MainActivity

1. Link the with layout
2. Create a listener for the button, when we click on the button the method startDownload()is called.
3. In this method an object of AsyncTask class extension, DownloadFileAsync is created and the method execute() is called. The parameter is an Internet address of a picture, for example **http://www.ottophoto.com/kirlian/kirlian\_1/kirlian12.jpg**
4. To download a picture from internet, we are using our method DownloadDrawable

Drawable DownloadDrawable(String url, String src\_name)

**throws** IOException {  
 **return** Drawable.*createFromStream*(((InputStream)

**new** URL(url).getContent()), src\_name);  
 }

and to call it we need the lines

Drawable drw = **null**;  
 **try** {  
 drw = DownloadDrawable(urlAddrress, **fileName**);  
 } **catch** (Exception e) {  
 Log.*d*(**"error "**, e.getMessage().toString());  
 }

Some notes here:

* Class URL represents a Uniform Resource Locator, a pointer to a "resource" on the World Wide Web
* [getContent](https://developer.android.com/reference/java/net/URL.html#getContent())() Gets the contents of this URL.
* A Drawable is a general abstraction for "something that can be drawn." Most often you will deal with Drawable as the type of resource retrieved for drawing things to the screen; the Drawable class provides a generic API for dealing with an underlying visual resource that may take a variety of forms.
* [createFromStream](https://developer.android.com/reference/android/graphics/drawable/Drawable.html#createFromStream(java.io.InputStream, java.lang.String))([InputStream](https://developer.android.com/reference/java/io/InputStream.html) is, [String](https://developer.android.com/reference/java/lang/String.html) srcName) Create a drawable from an inputStream.
* InputStream is an abstract class, is the superclass of all classes representing an input stream of bytes.

# BufferedInputStream

public class BufferedInputStream   
extends [FilterInputStream](https://developer.android.com/reference/java/io/FilterInputStream.html)

|  |  |  |  |
| --- | --- | --- | --- |
| [java.lang.Object](https://developer.android.com/reference/java/lang/Object.html) | | | |
| ↳ | [java.io.InputStream](https://developer.android.com/reference/java/io/InputStream.html) | | |
|  | ↳ | [java.io.FilterInputStream](https://developer.android.com/reference/java/io/FilterInputStream.html) | |
|  |  | ↳ | java.io.BufferedInputStream |

Wraps an existing [InputStream](http://spot.pcc.edu/~mgoodman/developer.android.com/reference/java/io/InputStream.html) and buffers the input.

## Assignment, Continuing Class Work

In similarity with the previous example add a new AsyncTask class and use it to read a text file in remote. The necessary lines that you need to read remotely are

BufferedInputStream in = **null**;  
String result=**""**;  
  
**try** {  
 URL url = **new** URL(theURLPath);  
  
 in = **new** BufferedInputStream(url.openStream());  
  
 **byte** data[] = **new byte**[1024];  
 **int** count;

//we read until we meet the byte -1  
 **while** ((count = in.read(data, 0, 1024)) != -1) {

//this converts byte-s into string  
 String s = **new** String(data);  
 . . .  
 }  
  
  
} **catch** (Exception e) {  
 Log.*d*(**"error "**, e.getMessage().toString());  
}

The layout has all the elements of the previous example, but you add a new button after the ImageView and before the TextView controls. It is better to make this TextView Control Scrollable, and for this you need these lines in this control

**android:lines="10"  
android:scrollbars="vertical"  
android:scrollbarStyle="insideOverlay"  
android:fadeScrollbars="true"  
android:fadingEdge="vertical"**

and add this line in code

**textview**.setMovementMethod(**new** ScrollingMovementMethod());

**References**

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

<https://www.journaldev.com/9708/android-asynctask-example-tutorial>

<http://abhiandroid.com/programming/asynctask>

https://developer.android.com/reference/android/graphics/drawable/Drawable.html

1. A generic type is a generic class or interface that is parameterized over types. [↑](#footnote-ref-1)