Lab: Asynchronous tasks and Sensors

J.-F. Lalande

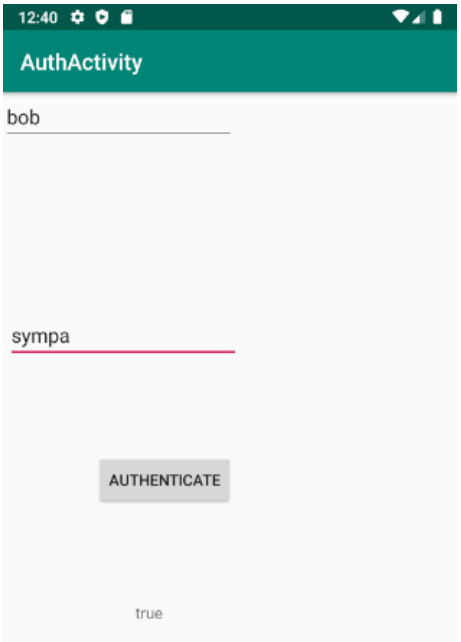
jean-francois.lalande@centralesupelec.fr

This labs intends to work with HTTP request : a lot of Android applications need to interact with a remote server. A lot of libraries can help to do this, but the goal of this lab is to build your request by yourself. Before working with the service Flickr, we first setup a simple HTTP request that we will use to authenticate the user. Then, we will move to Flickr in order to display images in a ListView object.

1 Authentication with a simple asynchronous task

In this part, we intend to authenticate the user against a server. As we want to avoid to develop a server, we will use the service `https://httpbin.org/` [1] that helps to perform tests with a remote service.

At the end, you should obtained the following result :



1.1 Authentication activity design

Exercise 1 Create an activity Authentication, that displays two EditText : one for the login of the user, one for the password.

Exercise 2 Create a button "Authenticate" for starting the authentication.

1. Thanks to Ken Reitz !

1.2 Setup of an HTTP request

We will handle the authentication with an HttpURLConnection object. You can look at this code :

```
URL url = null;
try {
    url = new URL("http://www.android.com/");

    HttpURLConnection urlConnection = (HttpURLConnection) url.openConnection();
    try {
        InputStream in = new BufferedInputStream(urlConnection.getInputStream());
        String s = readStream(in);
        Log.i("JFL", s);
    } finally {
        urlConnection.disconnect();
    }
} catch (MalformedURLException e) {
    e.printStackTrace();
} catch (IOException e) {
    e.printStackTrace();
}
```

Exercise 3 Put this code in your button. Where is the readStream() method ? [2] Solve this problem.

Exercise 4 Try to execute your code. You should have problems. Where to see these problems ? When executed, an Android application sends all debug information to the logcat output of AndroidStudio. To understand what is the issue with your code, you should have a look at this output. Open the Logcat tab on the bottom part of AndroidStudio and choose the "Info" level of information (it will filter the verbose and debug output but will keep the Errors). Re-run your application and check the nature of the error. Try to solve the error(s) with the following hints :

- **Cleartext HTTP traffic not permitted** [3] : try using httpS
- **java.net.SocketException : socket failed : EPERM** [4] : add the INTERNET permission, clean and rebuild your application.
- **android.os.NetworkOnMainThreadException** : you are doing a long task in the main thread : this is not allowed ! For now, you can simply use a Thread for which you override the run() method [5].

At this point, you should see the result of the HTTP request in the Logcat output.

1.3 Authenticating

Exercise 5 Open you web browser and test the url `https://httpbin.org/basic-auth/bob/sympa` : you should be authorized only if you enter the login bob and the password sympa. This service is for test purpose : it implements a Basic Authentication mechanism that allows only "bob" with the password "sympa" to succeed.

2. You can have a look at the discussion here : https://stackoverflow.com/questions/8376072/whats-the-readstream-method-i-just-can-not-find-it-anyw
3. https://stackoverflow.com/questions/45940861/android-8-cleartext-http-traffic-not-permitted
4. https://stackoverflow.com/questions/56266801/java-net-socketexception-socket-failed-eperm-opera
5. You can look at this page for starting a Thread : http://tutorials.jenkov.com/java-concurrency/creating-and-starting-threads.html

Exercise 6 Replace the url by the url "". Test in your Android app by just replacing the url `www.android.com` by `https://httpbin.org/basic-auth/bob/sympa`. For now, your code do not send any login and password. Thus, you should obtain the following error :

```
java.io.FileNotFoundException: https://httpbin.org/basic-auth/bob/sympa
```

It is expected. Indeed, this request currently does not send any authentication parameters.

Exercise 7 To solve this problem, add the following to to you HttpUrlConnection object :

```
String basicAuth = "Basic " + Base64.encodeToString("bob:sympa".getBytes(),
 Base64.NO_WRAP);
connection.setRequestProperty ("Authorization", basicAuth);
```

Exercise 8 Replace the credentials by the ones coming from the two EditText.

Exercise 9 Refresh a TextView called "result" to display the result of the authentication. Before displaying the result from the JSON, try to display "My result here" in the TextView. You should[6] obtain this error :

```
CalledFromWrongThreadException:
Only the original thread that created a view hierarchy can touch its views.
```

This is due to the fact that you try to change a TextView (a graphical element) from outside of the main thread. It is not allowed, because it may crash the application because of concurrency accesses. A possible solution consists in using the runOnUiThread methods that sends a Runnable object to be executed in the main thread of the application. This runnable will contain the code that should refresh the result TextView :

```
runOnUiThread(new Runnable() {
        @Override
        public void run() {
        // We refresh the result TextView
        ...
        }
}
```

Try to refresh the TextView with the string "My result here", just for testing. In the exercice we will replace this string by the real result.

Exercise 10 The result obtained from the HTTP connection should contains something like :

```
{
      "authenticated": true,
      "user": "bob"
}
```

You can reinstantiate this string as a JSONObject. Use the constructor of a JSONObject by passing the String as a parameter. Get the value associated to "authenticated" and put it in a variable named "res". Now, you want to send the object "res" to the Runnable object that modifies the TextView. You have two possibilities :

6. It is not guaranteed : https://stackoverflow.com/questions/15928551/no-error-calling-textview-settext-from-non-ui-thread-why

— You can add a constructor to the object created by new Runnable(). In this case you should put the code in an inner class[7]. By creating explicitly a class, you can create a constructor that takes a parameter : you will send "res" as a parameter, and store it as an attribute and you will be able to use in the run() method.
— You can store "res" as an attribute "result" of the activity. Then, in the run() method of the Runnable, you can access such an attribute by writting MainActivity.this.result. It is possible because Runnable is an anonymous class embedded in MainActivity and thus, attributes of MainActitivy can be accessed using the notation MainActivity.this.

Of course, httpbin.org is just for demo purpose and you will never see the login/password in the url of the service :p

2 Asynchronous tasks for a real remote service : Flickr

In this section, we focus on doing better implementation of asynchronous task than using the class Thread. In the previous section you have performed a simple Thread. You may have experienced several difficulties : you have to start the Thread, override the constructor to send parameters to the Thread, run the modification of the UI outside of this Thread. This is particularly painful, but instructive.

In this part we will work on better implementations of asynchronous tasks with another use case that is more fun than authenticating a user : displaying photos from the Flickr database.

Exercise 11 Create a new Android application called "Flickr app".

2.1 Discovering the API

We propose to use the Flickr service that provides an API for requesting images.

FIGURE 1 – Flickr by Paul Downey - CC-BY

Check the following JSON API from flickr in your web browser :

`http://www.flickr.com/services/feeds/photos_public.gne?tags=trees&format=json`

If you look at this url at different time, you will see that the data is often updated. Maybe not for trees, but for cats, it's pretty sure :

`http://www.flickr.com/services/feeds/photos_public.gne?tags=cats&format=json`

Do not poll this url too much because we (our IP) may be banned from Flickr. Images can be accessed using the media tag. Different sizes are available :

7. Right button > Refactor > Extract to inner class

`https://live.staticflickr.com/65535/50652853707_e193d62b53_m.jpg`
`https://live.staticflickr.com/65535/50652853707_e193d62b53_s.jpg`

Exercise 12 Why the answer of the server is not really using the JSON format ? What should be removed ?

2.2 Request the JSON data using an AyncTask

In this part, we intend to develop an AsyncTask responsible of getting the JSON data from the service.

Exercise 13 We propose to develop a class that extends `AsyncTask<String, Void, JSONObject>`. Why choosing such a type ?

Exercise 14 Create a button "Get an image" in your activity. Override the `OnClickListener` with a new class `GetImageOnClickListener` that extends `View.OnClickListener`. In the `OnClick`, start a new asynctask of type `AsyncFlickrJSONData`. Give, as a parameter the url :

`https://www.flickr.com/services/feeds/photos_public.gne?tags=trees&format=json`

Exercise 15 Create the class `AsyncFlickrJSONData` that extends `AsyncTask<String, Void, JSONObje` Develop the following functionalities :

— In `doInBackground`, perform the HTTP connection, and reinstantiate the JSON object.
— In `onPostExecute`, Log the obained JSON object in Logcat [8]

2.3 Downloading an image

Exercise 16 Add image to your layout with id "image".

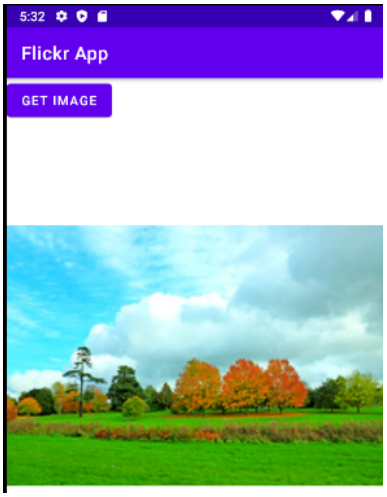
Exercise 17 Decode the url of the first image by manipulating the JSONObject in the `onPostExecute` method of `AsyncFlickrJSONData`. You can use :

— getJSONArray for getting an array associated to a key ;
— getJSONObject for getting an object associated to a key ;
— getJSONString for getting a String associated to a key.

8. Be careful : the string returned by Flickr contains an extra "jsonFlickrFeed({...})". You should only keep the {...} and remove "jsonFlickrFeed(" and the last ")". If not, the instantiation of the JSONObject will fail. You can use the method subSequence of the class STring.

Exercise 18 Create a class `AsyncBitmapDownloader` that extends `AsyncTask<String, Void, Bitmap>`. This class downloads an image given its input URL. To create the Bitmap object, you can use the `BitmapFactory` class that provides a method `decodeStream`, as shown below. When the bitmap is built, put it in your `ImageView`.

```
InputStream in = new BufferedInputStream(urlConnection.getInputStream());
Bitmap bm = BitmapFactory.decodeStream(in);
```



You should now be confortable with AsyncTask. You should conclude that it is more convenient for separating the task and the update of the graphical interface between the different threads.

2.4 Displaying a list of images using Volley - the legacy ListView method

ListView is in the "legacy" way of implementing lists. As it is a little bit easier to use that RecyclerView, we propose to use them in this section. Two steps are proposed towards our goal :

— Step 1 : We display the list of URLs in a ListView.
— Step 2 : we replace these TextViews by images.

2.4.1 Displaying a list of the images URLs

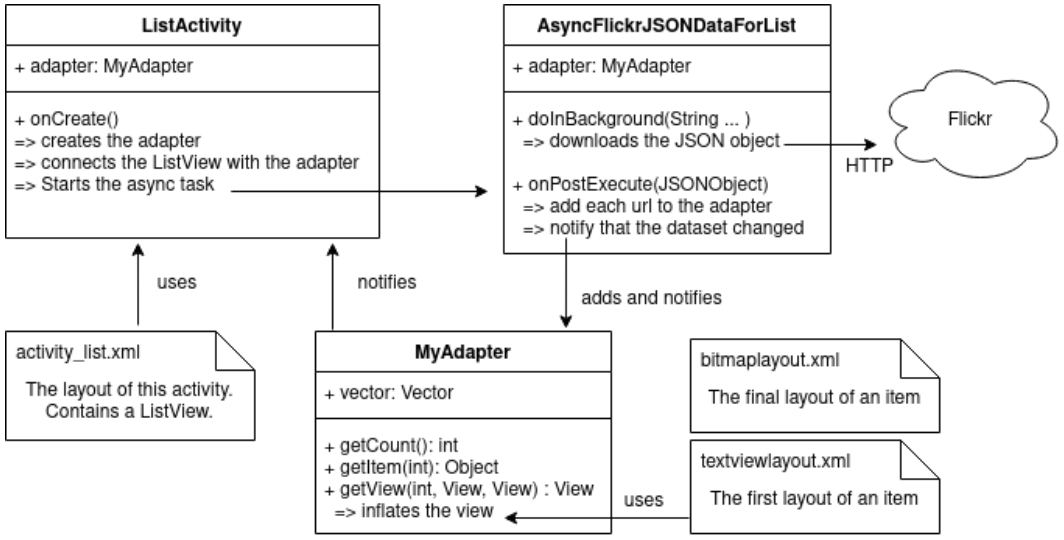
We recall that the list of images will be received from Flickr, as done in the previous section. We will decode again the JSON object but, instead of using the first image by calling `getJSONObject(0)`, we will iterate over all images and add them in an adapter. The adapter is connected to the list and we will notify the list that some data just arrived, in order to refresh the list.

The global overview of the needed classes are represented below :

Exercise 19 Create a new Activity called `ListActivity`. Create the associated layout `activity_list.xml` with only one element `ListView` attached to the root. Put the id `list` for this `ListView`. On your main activity, create a new button that sends the user to `ListActivity`.

Exercise 20 In `ListActivity` create an adapter of type `MyAdapter`. This class is a new class that extends `BaseAdapter`. In `ListActivity`, link your `ListView` with this aapter.

Exercise 21 Generate the class `MyAdapter` that extends `BaseAdapter`. Create an attribute `vector` of type `Vector<String>`. This vector will store the url that we will get from the JSONObject. Add a method `dd(String url)` that adds a received url in your vector. Implement the method `getCount`. The other method last important method `getView` will be implemented later. We can for now put a log call such as `Log.i("JFL", "TODO");`



Exercise 22 We need now an asynctask that downloads the list of URL from Flickr and that populates our adapter with the obtained data. Duplicate the class `AsyncFlickrJSONData` and call it `AsyncFlickrJSONDataForList`. Review your code and perform the following modifications :

- `doInBackground` is obviously the same ;
- `onPostExecute` soud be adapted : instead of launching the download of the first image, this method should iterate over all images of the `JSONArray` contained in the received `JSONObject`. For each, item we need to put the URL into the adapter : we need to have this adapter available in `AsyncFlickrJSONDataForList`. Pass `MyAdapter` as a parameter of the constructor `AsyncFlickrJSONDataForList` when called in the activity `ListActivity`. Store `MyAdapter` as an attribute. When adding an URL into the adapter add a log line with a call to `Log.i("JFL", "Adding to adapter url : " + url);`.

Exercise 23 Add this stage, we can do a quick test of your code. Nothing will be displayed on the smartphone as the `getView` method is still not implemented, but you should observe in the logcat some logs. Check that you see :

```
Adding to adapter url : https://live.staticflickr.com/65535/50664142118_...jpg
Adding to adapter url : https://live.staticflickr.com/65535/50664125043_...jpg
Adding to adapter url : https://live.staticflickr.com/65535/50664936462_...jpg
```

Exercise 24 Nevertheless, you could have expected to see "TODO" in the logcat, because we put in the `getView` method the line `Log.i("JFL", "TODO");` (this method still returns null). Returning a null object is probably a bad idea but if we do not see "TODO" in the logcat, it means that the `getView` method is never called. How is it possible? It is due to the adapter that should inform the `ListView` that some new data has arrived. In `AsyncFlickrJSONDataForList`, after populating the adapter, make a call to the method `notifyDataSetChanged()` on the adapter. Make a new test. You shoud see a "TODO" appearing in the logcat. Great! But your code crash, because `getView` is not supposed to return a null object.

Exercise 25 Implement `getView`. To make it simple, you can just inflate the layout `R.layout.textviewlay` This layout will contains a `LinearLayout` and a simple `TextView`. After inflating, get the `TextView` with `findViewById`. Then, replace the text with the text from the vector at position `i`. Test your new implementation. That's it!

2.4.2 Downloading images using Volley

Because downloading is painful, we propose in this part to use Volley, an helper called Volley is now available in Android. It handles multiple HTTP requests in a queue and simplifies the code, avoiding to develop an `AsyncTask`. It also provides a cache to speed up the download if the image has been already downloaded.

Exercise 26 Add the following lines to the dependencies in build.gradle :

```
implementation 'com.android.volley:volley:1.1.1'
```

Exercise 27 Have a look to this page of the documentation : `https://developer.android.com/training/volley/requestqueue?hl=zh-tw#singleton`. Import the class `MySingleton` in your project. Solve the missing imports. Now we can instantiate a request queue by doing :

```
// Get a RequestQueue
RequestQueue queue = MySingleton.getInstance(viewGroup.getContext()).
                getRequestQueue();
```

Exercise 28 In the `getView` method of `MyAdapter` comment the code that inflates and modifies the TextView. Now, inflate from a new layout file `bitmaplayout.xml` a layout that contains an ImageView. This is this image that will be replaced when Volley has finished to download the image.

Exercise 29 You need to create an `ImageRequest` object. Instantiate an object ImageRequest with the right parameters [9]. For the second parameter, see the next exercise.

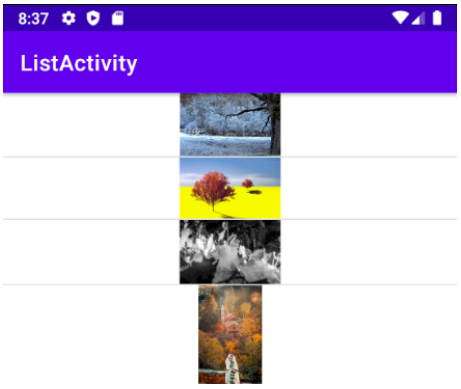
Exercise 30 The second parameter is the *response listener* i.e. the listener that is called when the request is finished. Its type will be `Response.Listener<Bitmap>`. Create this parameter using a lambda expression :

```
Response.Listener<Bitmap> rep_listener = response -> {
// TODO
}
```

The code that goes here is very simple : response is of type Bitmap : it is the bitmap that Volley has downloaded. Thus this code is executed **when** the download has completed. The code to add in this lambda expression is the modification of the image just inflated previously with the method `setImageBitmap`.

Exercise 31 Test your implementation : it works!

9. In case the construtor is deprecated, you can have a look here : `https://stackoverflow.com/questions/33271864/android-volley-imagerequest-deprecated`



2.5 Customize the type of image (bonus)

Exercise 32 Tired of trees ? Classical application have settings the user wish to modify. Android provide facilities for handling preferences. Customize the request to Flickr (currently asking for "cats") by adding a *PreferenceActivity* where the string can be edited. Add a preference to handle the caching of images or not.

3 Geolocalized images

Exercise 33 Get the coordinate of your smartphone. Print these coordinates in the logs, for debug purpose.

Exercise 34 Use the Flickr photos search API [10] to get geolocalized data. Several modifications in your code will be necessary. In particular, the JSON url will look like the following :

```
String url = new String("https://api.flickr.com/services/rest/?" +
                                "method=flickr.photos.search" +
                "&license=4" +
                "&api_key=xxxxxxxxxx" +
                "&has_geo=1&lat=" + lat +
                "&lon=" + lon + "&per_page=1&format=json");
```

10. https://www.flickr.com/services/api/flickr.photos.search.html