CAVARD GABRIEL

M1 SE2

Mobile Development on Android TP2

GitHub link: https://github.com/bigateau/TP2-Android.git

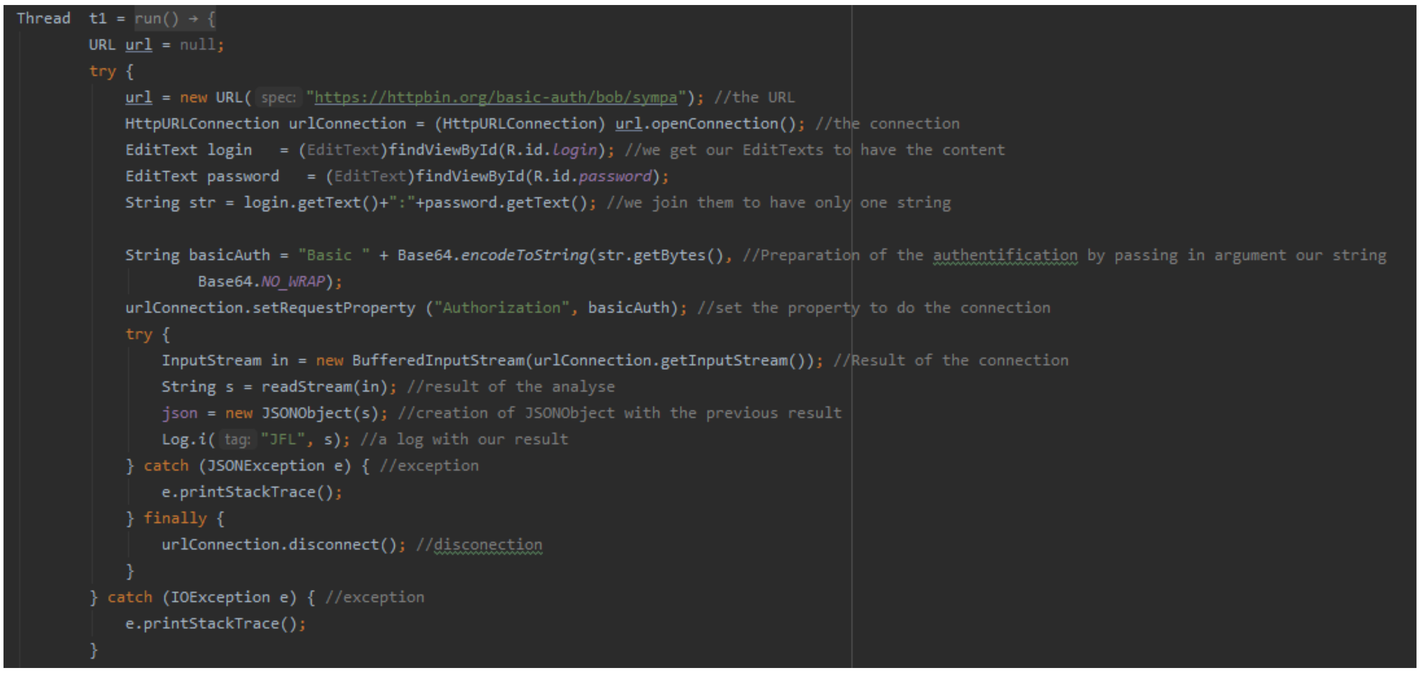
Part1:

The first part of the TP was to make an authentication with a simple asynchronous task. The exercise was to put some strings in EditText and if the logins are correct you have a message with the log print on the main activity.

For an asynchronous task we need to create a thread that will be execute when the button authentication is clicked.

This thread is composed of three parts: the connection to the URL, a function to get the result of the connection and a way to display the result.

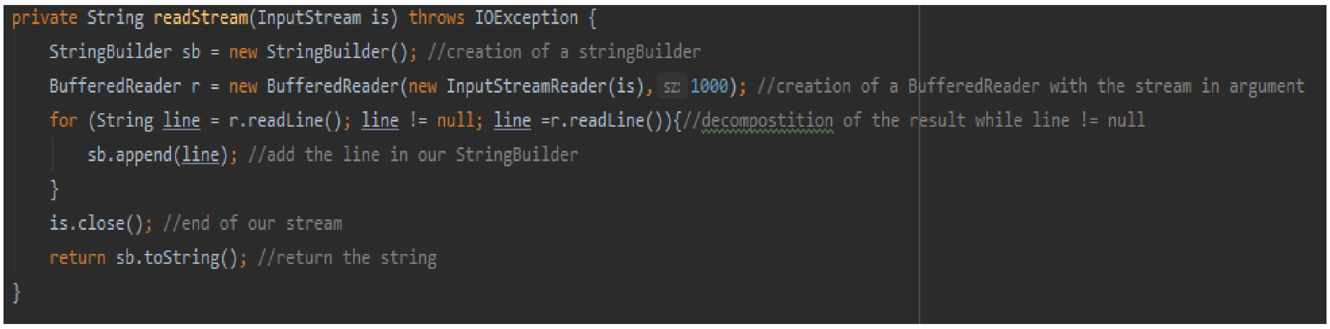
- Here the connection:



```
Thread  t1 = run() → {
        URL url = null;
        try {
            url = new URL( spec: "https://httpbin.org/basic-auth/bob/sympa"); //the URL
            HttpURLConnection urlConnection = (HttpURLConnection) url.openConnection(); //the connection
            EditText login   = (EditText)findViewById(R.id.login); //we get our EditTexts to have the content
            EditText password   = (EditText)findViewById(R.id.password);
            String str = login.getText()+":"+password.getText(); //we join them to have only one string

            String basicAuth = "Basic " + Base64.encodeToString(str.getBytes(), //Preparation of the authentification by passing in argument our string
                    Base64.NO_WRAP);
            urlConnection.setRequestProperty ("Authorization", basicAuth); //set the property to do the connection
            try {
                InputStream in = new BufferedInputStream(urlConnection.getInputStream()); //Result of the connection
                String s = readStream(in); //result of the analyse
                json = new JSONObject(s); //creation of JSONObject with the previous result
                Log.i( tag: "JFL", s); //a log with our result
            } catch (JSONException e) { //exception
                e.printStackTrace();
            } finally {
                urlConnection.disconnect(); //disconection
            }
        } catch (IOException e) { //exception
            e.printStackTrace();
        }
```

One thing very important here is to not forget to give the android permissions for INTERNET and ACCESS_NETWORK_STATE. The program can do the URL connection without them.

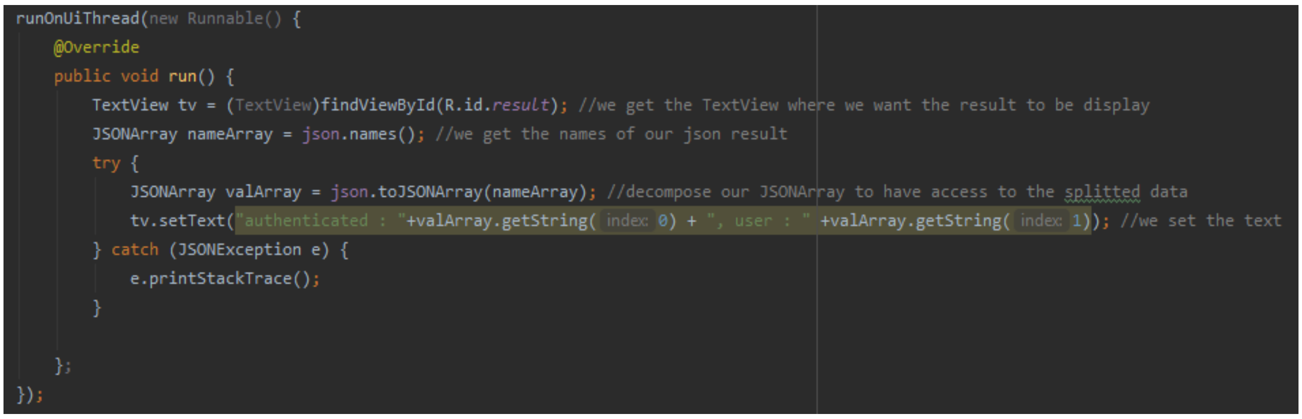
- The readStream function that is mentioned to be add in the question 3:



```
private String readStream(InputStream is) throws IOException {
    StringBuilder sb = new StringBuilder(); //creation of a stringBuilder
    BufferedReader r = new BufferedReader(new InputStreamReader(is), sz: 1000); //creation of a BufferedReader with the stream in argument
    for (String line = r.readLine(); line != null; line =r.readLine()){//decompostition of the result while line != null
        sb.append(line); //add the line in our StringBuilder
    }
    is.close(); //end of our stream
    return sb.toString(); //return the string
}
```

Here nothing very important to specify. We just have to be sure and to have well understand how it works. We are working with a InputStream and we have to return a string. That is why we have the InputStream.

- The RunOnUIThread:



```
runOnUiThread(new Runnable() {
    @Override
    public void run() {
        TextView tv = (TextView)findViewById(R.id.result); //we get the TextView where we want the result to be display
        JSONArray nameArray = json.names(); //we get the names of our json result
        try {
            JSONArray valArray = json.toJSONArray(nameArray); //decompose our JSONArray to have access to the splitted data
            tv.setText("authenticated : "+valArray.getString( index: 0) + ", user : " +valArray.getString( index: 1)); //we set the text
        } catch (JSONException e) {
            e.printStackTrace();
        }

    };
});
```

All the code is in a Thread, and this thread does not have the permission to access and modify the TextView. That is why I use a RunOnUiThread to do this. This Thread is connected to the Main Activity so the Main thread will execute this part, the Thread with the authorization.

Part2:

The second part of the TP was to have a program with two pages. The first (The main) displays one image. Then you have a button to get to the second page. The button send user. On the second page we have a list of few image displayed on a ListView on the screen.

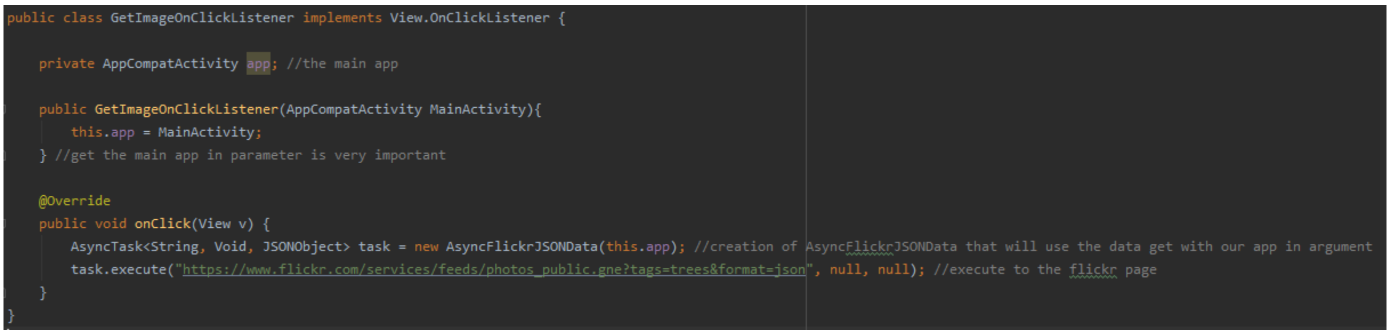
I use globally the same way as the first part but with more details. That is why I will just show you the big specifications.

Do not forget to give the android permissions for INTERNET and ACCESS_NETWORK_STATE.

First, it is important to talk about the first function, display one picture.

When we click on the button “Get an image”, a specific OnClickListener is created. It is the same as the first part of the TP, but we have and keep the Main activity as an attribute. It is the main purpose of the TP: keep the MainActivity in each page. Because if we do not, at the end we will not be able to modify the ImageView.

All the pictures are in the Flicks service, an API that provides images. The GetImageOnClickListener will create an AsyncFlickrJSONData with the MainActivity in argument and execute with the URL of the Flicks.

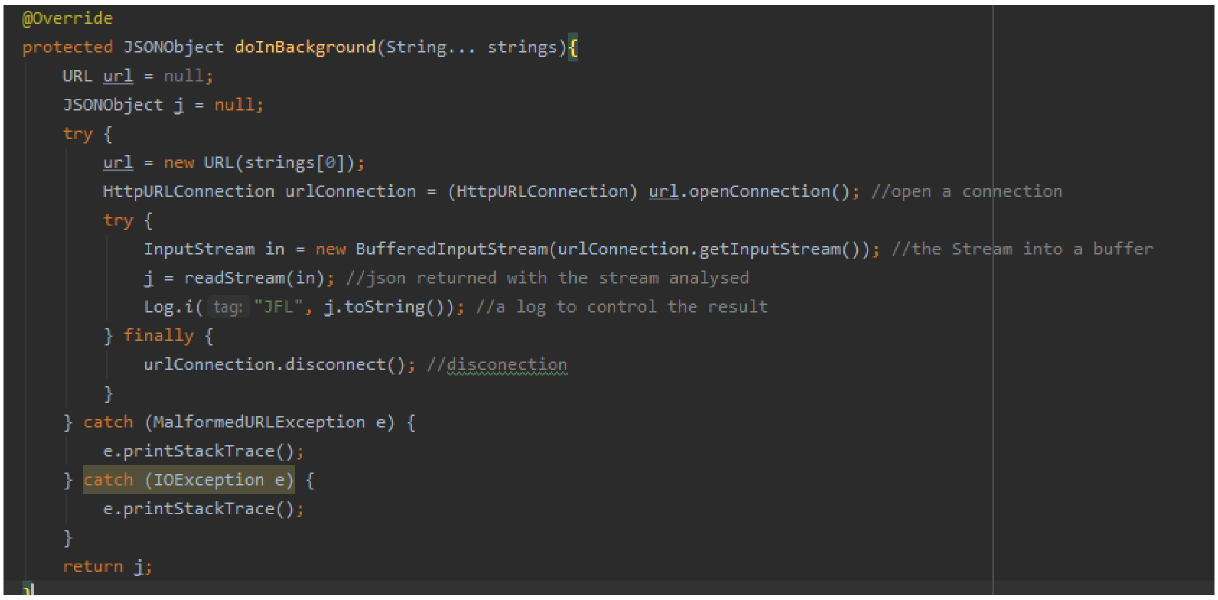


```java
public class GetImageOnClickListener implements View.OnClickListener {

    private AppCompatActivity app; //the main app

    public GetImageOnClickListener(AppCompatActivity MainActivity){
        this.app = MainActivity;
    } //get the main app in parameter is very important

    @Override
    public void onClick(View v) {
        AsyncTask<String, Void, JSONObject> task = new AsyncFlickrJSONData(this.app); //creation of AsyncFlickrJSONData that will use the data get with our app in argument
        task.execute("https://www.flickr.com/services/feeds/photos_public.gne?tags=trees&format=json", null, null); //execute to the flickr page
    }
}
```

The AsyncFlickrJSONData extends AsyncTask<String, Void, JSONObject>. It is decomposed in two parts: the *doInBackground* and the *onPostExecute.* The doInBackground will do tasks in background, the onPostExecute is doing tasks at the end of the run.

At the beginning we store the MainActivity as an attribute. Very important.

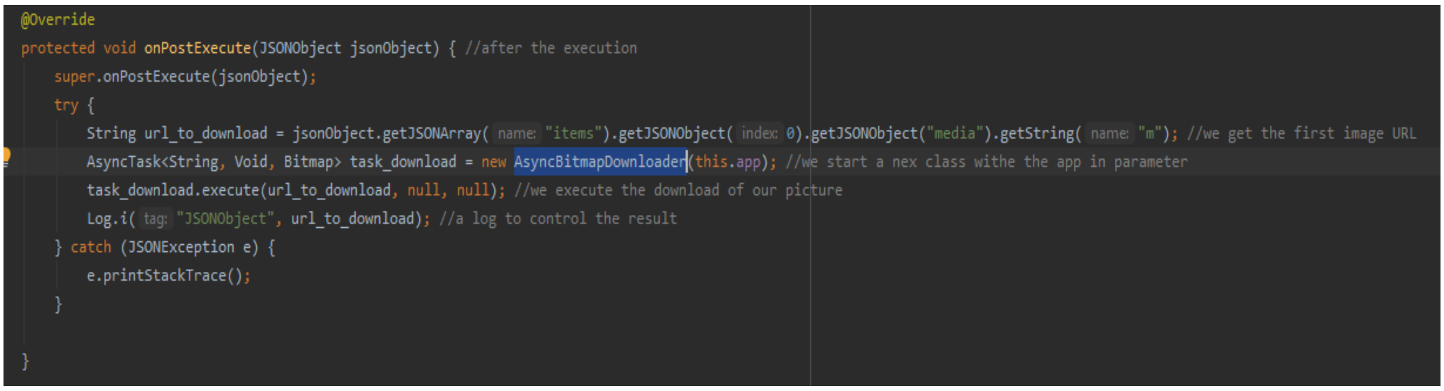
Then in the doInBackground the connection as seen before will be execute and at the end we have a JSON object with the answer of the Flicks API



```java
@Override
protected JSONObject doInBackground(String... strings){
    URL url = null;
    JSONObject j = null;
    try {
        url = new URL(strings[0]);
        HttpURLConnection urlConnection = (HttpURLConnection) url.openConnection(); //open a connection
        try {
            InputStream in = new BufferedInputStream(urlConnection.getInputStream()); //the Stream into a buffer
            j = readStream(in); //json returned with the stream analysed
            Log.i( tag: "JFL", j.toString()); //a log to control the result
        } finally {
            urlConnection.disconnect(); //disconection
        }
    } catch (MalformedURLException e) {
        e.printStackTrace();
    } catch (IOException e) {
        e.printStackTrace();
    }
    return j;
}
```

The readStream is not the same as the part 1 but do the same things. The only difference is instead of returning a string it returns a JSON object.

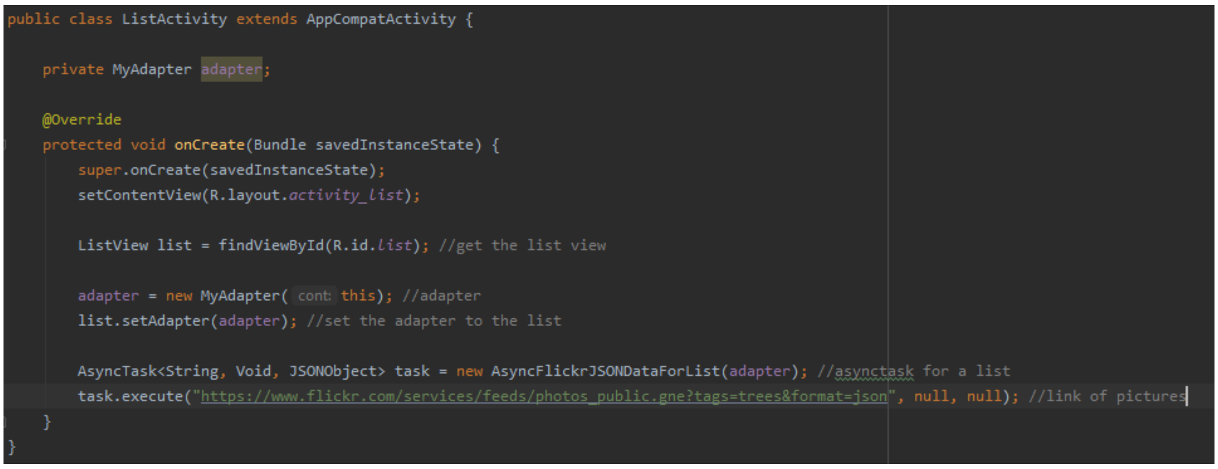
And at the end, in the onPostExecute, we do another connection by create an AsyncBitmapDownloader with the main activity in argument and an execution with the link of only one picture.



```
@Override
protected void onPostExecute(JSONObject jsonObject) { //after the execution
    super.onPostExecute(jsonObject);
    try {
        String url_to_download = jsonObject.getJSONArray( name: "items").getJSONObject( index: 0).getJSONObject("media").getString( name: "m"); //we get the first image URL
        AsyncTask<String, Void, Bitmap> task_download = new AsyncBitmapDownloader(this.app); //we start a nex class withe the app in parameter
        task_download.execute(url_to_download, null, null); //we execute the download of our picture
        Log.i( tag: "JSONObject", url_to_download); //a log to control the result
    } catch (JSONException e) {
        e.printStackTrace();
    }

}
```

Finally, in the AsyncBitmapDownloader, we have the same structure. The only difference it is that the doInBackground works with Bitmap and not JSON Object. The doInBackground will download the stream result of the picture URL and the onPostExecute will execute and set the image in the Main activity, because since the beginning we passed the Main activity from object to object just for this moment, the moment when we will set the picture on the main activity. I decided to not put pictures because it is closed to what I already showed, and I prefer to pass to the last part of the TP2.

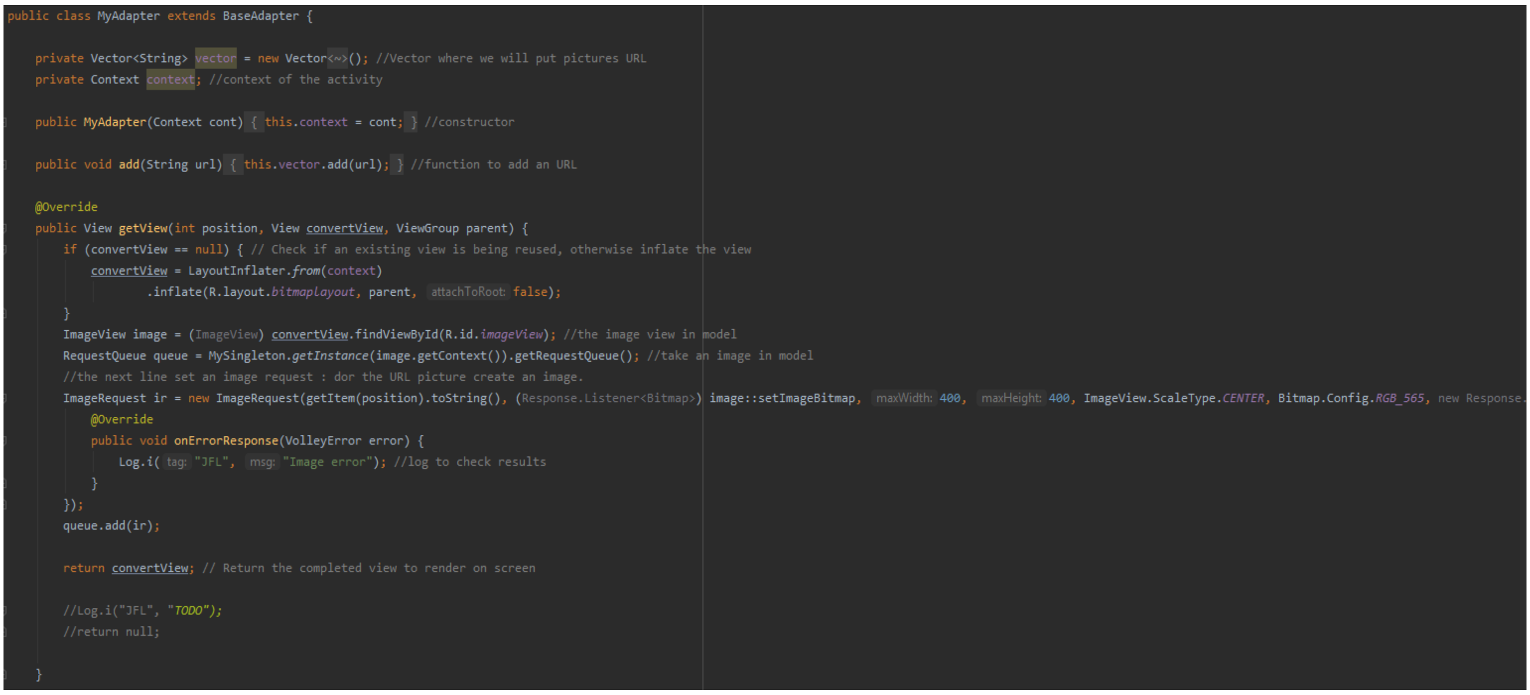
The last part of the TP is the display of a lot of pictures in a list view. It appends when we click on the button send_user and a redirection to the ListActivity page is set.



```
public class ListActivity extends AppCompatActivity {

    private MyAdapter adapter;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_list);

        ListView list = findViewById(R.id.list); //get the list view

        adapter = new MyAdapter( cont: this); //adapter
        list.setAdapter(adapter); //set the adapter to the list

        AsyncTask<String, Void, JSONObject> task = new AsyncFlickrJSONDataForList(adapter); //asynctask for a list
        task.execute("https://www.flickr.com/services/feeds/photos_public.gne?tags=trees&format=json", null, null); //link of pictures
    }
}
```

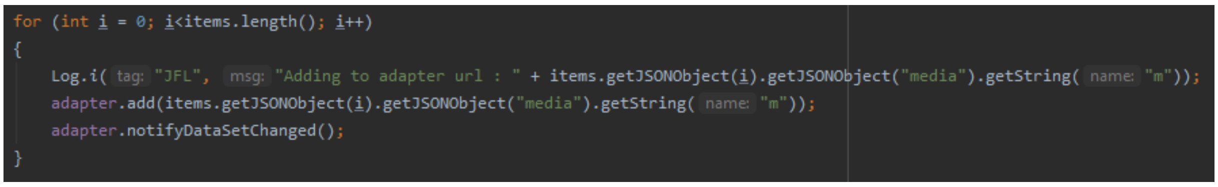
The AsyncFlickrJSONDataForList has the same structure as seen before, the only difference it is that it does for all pictures in the URL API not only one.

This time, it is different. We will use an adapter. The role of our adapter is to store our URLs in a vector, and for each URL, ask to prepare an image to add to the list. And thanks to singleton, the adapter will explain to the singleton to prepare an image. And then, an image request in set and the picture is download into a bitmap and add to the singleton.



```java
public class MyAdapter extends BaseAdapter {

    private Vector<String> vector = new Vector<~>(); //Vector where we will put pictures URL
    private Context context; //context of the activity

    public MyAdapter(Context cont) { this.context = cont; } //constructor

    public void add(String url) { this.vector.add(url); } //function to add an URL

    @Override
    public View getView(int position, View convertView, ViewGroup parent) {
        if (convertView == null) { // Check if an existing view is being reused, otherwise inflate the view
            convertView = LayoutInflater.from(context)
                    .inflate(R.layout.bitmaplayout, parent,  attachToRoot: false);
        }
        ImageView image = (ImageView) convertView.findViewById(R.id.imageView); //the image view in model
        RequestQueue queue = MySingleton.getInstance(image.getContext()).getRequestQueue(); //take an image in model
        //the next line set an image request : dor the URL picture create an image.
        ImageRequest ir = new ImageRequest(getItem(position).toString(), (Response.Listener<Bitmap>) image::setImageBitmap,  maxWidth: 400,  maxHeight: 400, ImageView.ScaleType.CENTER, Bitmap.Config.RGB_565, new Response.
            @Override
            public void onErrorResponse(VolleyError error) {
                Log.i( tag: "JFL",  msg: "Image error"); //log to check results
            }
        });
        queue.add(ir);

        return convertView; // Return the completed view to render on screen

        //Log.i("JFL", "TODO");
        //return null;

    }
```

The getView is done each time that an URL is add to the vector thanks to those lines in the AsyncFlickrJSONDataForList:



```java
for (int i = 0; i<items.length(); i++)
{
    Log.i( tag: "JFL",  msg: "Adding to adapter url : " + items.getJSONObject(i).getJSONObject("media").getString( name: "m"));
    adapter.add(items.getJSONObject(i).getJSONObject("media").getString( name: "m"));
    adapter.notifyDataSetChanged();
}
```

<u>Conclusion</u>:

I did not put all my code screens because there is a lot of comments in my code and I have decided to put only the main importance points of the structure and functionalities of this TP.

This TP was very interesting and well formed. We started with somethings very simple and then we developed and upgraded our way of thinking to have a stronger project and have more functionalities very useful. Now I know how to use an API in an android project and how to do asynchronous tasks.