LAB 2 : Asynchronous tasks and Sensors

https://github.com/eldondanonino/Andoid-Lab-2

Note : This lab report only covers the questions 1 – 20 included. Facing technical difficulties combined to personal and understanding issues, I regret the fact that I must present to you an incomplete work, however I believe that the aspects that were explored work just fine :)

Q3 : Where is readStream()?

At the beginning, nowhere : readStream is a user generated function that we must implement ourselves (thank you Stack Overflow). There are different ways to do it, and I have an alternate version in the second part for debugging purposes.

Q4 : Where can we see compiling problems?

They are located in the build output. For further information, the logcat output offers everything we need for debugging purposes.

Q13 : Why extend AsyncTask<String, Void, JSONObject> ?

Here, String represents the parameter type sent to the task, void means that there is no progress publishing, while JSONObject is the type of the result!

Code Explanation :

In the first part of the lab, the interesting interaction is between the Thread we create and its body. It allows to circumvent limitations of Android and offers a clean onCreate.

```
go.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        MyThread t = new MyThread();
        t.start();
    }
});
```

To access elements of the activity, we can look it up by ID

```
EditText loginText = (EditText) findViewById(R.id.Login);
EditText pwdText = (EditText) findViewById(R.id.pwd);
TextView textView = (TextView) findViewById(R.id.textView2);
```

We also saw how to interpret a pseudo JSON sent by a website and how to transform it (the buffered input is made into a true JSON from a string form)

```
InputStream in = new BufferedInputStream(urlConnection.getInputStream());
String s = readStream(in);
Log.i( tag: "JFL", s);
JSONObject res = new JSONObject(s);
```

The use of permissions in the manifest is an interesting side note enabling communication with the web

```
<uses-permission android:name="android.permission.INTERNET" />
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
```

The second part is where we really get in the meat of the subject, with actual asynchronous structuration.

Both AsyncBitmapDownloader and AsyncFlickrJSONData are extending AsyncTask with their respective parameters and share the internalm structure bound to it.

```
class AsyncBitmapDownloader extends AsyncTask<String, Void, Bitmap> //we send a string, and get a bitmap
{...}


class AsyncFlickrJSONData extends AsyncTask<String, Void, JSONObject> { //we send a string and get a JSONObject
```

They both use doInBackground an onPostExecute to manage the tasks (background for DOIB or at the end for OPE)

```
@Override
protected JSONObject doInBackground(String... strings) {...}


@Override
protected void onPostExecute(JSONObject obj2) {...}
```

The logical path in the program starts on the click, where we our custom getImageOnClickListener waits for the interaction. Within it, we create a new AsyncFlikrJSONData element that will do the actual communication with the API.

```
class getImageOnClickListener implements View.OnClickListener
    @Override
    public void onClick(View v) {
        AsyncFlickrJSONData bob = new AsyncFlickrJSONData();
        bob.execute("https://www.flickr.com/services/feeds/pho
```

As seen in part one, we extract the JSON (but we have to trim it a bit because Flikr gives a pseudo JSON raw).

```
InputStream in = new BufferedInputStream(urlConnection.getInputStream());
String s = readStream(in);
// Log.i("S" , s);
s = s.subSequence(15, s.length() - 1).toString(); //We trim out the non-JSON
```

Once we curated our data, AFJD can return it for the onPostExecute to use! It will extract the wanted part of the JSON (here the URL of the image) and update the activity. (Here, since I did not get to the end of image downloads, it only displays the URL in the app)

```
tmp = obj2.getJSONArray( name: "items").getJSONObject( index: 0).getJSONObject("media");
link = tmp.getString( name: "m"); //we extract the url
```

Finally, the last thing I did before not being able to go fonward was the new list activity, reachable for the user via a button

```
list.setOnClickListener(new View.OnClickListener() { //Clicking the button sends the user to a new list
    @Override
    public void onClick(View v) {
        Intent k = new Intent( packageContext: MainActivity.this, ListActivity.class);
        startActivity(k);
    }
});
```

In this lab I have learned to use asynchronous tasks to my advantage, notably to listen for specific events and use those to reach out to an external API. The JSON manipulations can be a bit finnicky and combined to the limitations of older android versions it was very challenging. However, it allowed me to get a better grasp on android development and the restrictions linked to its compatibility.