Übung 4: Kommunikation & Nebenläufigkeit – HTTP, JSON, XML, AsyncTasks und Threads

In dieser Übung geht es um Backend-Kommunikation und Nebenläufigkeit unter Android. Konkret implementieren Sie HTTP-Anfragen, um Bilder und Texte bei einem HTTP-Server abzuholen. Zusätzlich werden Sie JSON Ressourcen mit Hilfe eines Retrofit-Services abrufen und darstellen. Im Weiteren sollen länger dauernde Aufgaben nicht den *main*-Thread einer Android-Applikation blockieren. Sie verwenden dazu die beiden Programmierkonzepte für Nebenläufigkeit auf der Android-Plattform: die Klasse `AsyncTask` und die Klasse `Thread`.

0. **Neue App: Com & Con (Communication & Concurrency)**
   Erstellen Sie ein neues Android-Applikationsprojekt. Diese Übung besteht aus verschiedenen Teilaufgaben, welche in der Main-Activity von diesem Projekt soweit als möglich resp. sinnvoll dargestellt werden. Diese `MainActivity` (Layout-Datei: `activity_main.xml`) wird am Schluss ungefähr so aussehen wie der Screenshot rechts oben, verwenden Sie dazu eine `ScrollView` mit einem `LinearLayout`, siehe Übung 2 für Details dazu.

1. **HTTP Demos: Texte und Bilder mittels HTTP übertragen**
   Erstellen Sie eine neue Activity `HttpDemosActivity`, welche zwei Knöpfe „Lade Binäre Datei (Bild)“ und „Lade Dokument (Text)“ hat. Beim Drücken dieser Knöpfe sollen entsprechend ein Bild, resp. ein Text aus dem Internet geladen und dargestellt werden. Daten-URLs:
   - http://wherever.ch/hslu/loremIpsum.txt
   - http://wherever.ch/hslu/homer.jpg

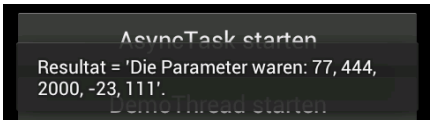
   Bauen Sie wie in der Vorlesung beschrieben mittels eines OkHttp-Clients eine HTTP-Verbindung auf und holen Sie die entsprechenden Dateien. Stellen Sie die erhaltenen Daten entsprechend in einer `ImageView`, resp. `TextView` dar, siehe Screenshot rechts. Verwenden Sie hierzu einen `AsyncTask`, damit das UI nicht blockiert wird.

2. **HTTP Demos: JSON mit Retrofit abfragen**
   Erweitern Sie die `HttpDemosActivity` um einen weiteren Knopf, mit dem eine JSON-Webservice konsumiert mittels Retrofit konsumiert wird. Verwenden Sie dazu die folgende beiden URL:
   - http://www.nactem.ac.uk/software/acromine/dictionary.py?sf=HTTP

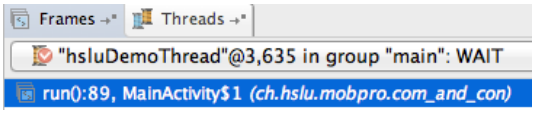
   Schreiben Sie zum Laden dieser Ressource einen Retrofit-Service, komplett mit den Antwort-Transferobjekten und konfigurieren Sie den Service mit einem entsprechenden JSON-Mapper ihrer Wahl (Gson, Jackson, …) Stellen Sie den relevanten Inhalt der Antwort auf dem Bildschirm dar, siehe Screenshot rechts.

3. **[OPTIONAL] „Freeze“-Button**
   Fügen Sie der `MainActivity` einen neuen Knopf „GUI 7 Sekunden blockieren“ hinzu. Wird dieser gedrückt, soll der aktuelle Thread (= main-Thread = UI-Thread) 7 Sekunden warten mittels `Thread.sleep(...)`. Entsprechend ist die App 7 Sekunden lang blockiert und Sie reagiert nicht auf Benutzerinteraktion...

4. **[OPTIONAL] Warten in AsyncTask**
   Fügen Sie nun der `MainActivity` einen neuen Knopf „AsyncTask starten“ hinzu. Wird dieser gedrückt, soll in einem `AsycTask` 7 Sekunden gewartet werden mittels `Thread.sleep(...)`. Implementieren Sie dazu eine eigene Klasse `AsyncDemoTask`. Dem `AsyncDemoTask` sollen als Parameter Integer-Objekte übergeben werden können und als Resultat wird ein String zurückgeliefert, welcher alle Integer-Werte enthält und entsprechend in einem Toast ausgegeben wird, siehe Screenshot rechts. Schauen Sie, dass maximal ein AsncTask laufen kann. Setzen Sie Breakpoints um die Ausführung in verschiedenen Threads nachvollziehen zu können.



5. **Warten in eigenem Thread**
   Fügen Sie nun der `MainActivity` einen neuen Knopf „DemoThread starten“ hinzu. Wird dieser gedrückt, soll in einem eigenen Thread 7 Sekunden gewartet werden mittels `Thread.sleep(...)`. Während der Ausführung des Threads soll der Text des Buttons „[DemoThread läuft…]“ lauten. Implementieren Sie dazu einen eigenen `Thread` (keinen `AsyncTask`!) und schauen Sie, dass maximal einer ihrer Thread am laufen ist. Setzen Sie Breakpoints um die Ausführung in verschiedenen Threads nachvollziehen zu können, siehe Screenshot rechts.



6. **AsyncTask mit Progress**
   Implementieren Sie zum Abschluss eine eigene Klasse `MultiAsyncTask` welche von `AsyncTask` erbt. Diese Klasse soll eine Liste von Text-Dateien mit je einem Filmtitel aus dem Internet herunterladen und in der App darstellen. Jedes Mal, wenn eine Text-Datei (`http://wherever.ch/hslu/title0.txt` für Titelnummern von 0 bis 4) angekommen ist, soll dies mit einem `Toast` angezeigt werden (Methode `AsyncTask.onProgressUpdate`) und am Schluss (Methode `AsyncTask.onPostExecute`) sollen alle Filmtitel in einem Dialog angezeigt werden, siehe Screenshots unten. Die Klasse `MultiAsyncTask` soll dabei folgende Signatur haben:
   `public class MultiAsyncTask extends AsyncTask<URL, String, Void>`
   Hinweis: Verzögern Sie das Herunterladen künstlich, indem Sie zwei Sekunden warten nach dem Herunterladen einer Datei, damit stauen sich die Toasts nicht zu sehr und das Ganze fühlt sich etwas „realistischer“ (d.h. verzögerter) an.