ANDROID - Lab 4

Lesson goals:

- What is a JSON and how to convert a java class to one
- Making Server Requests
- Creating BroadcastReceiver

JSON (Javascript Object Notation)

JavaScript Object Notation (JSON) is a standard text-based format for representing structured data of any programming language

Since the JSON format is text only, it can easily be sent to and from a server. It needs to be converted to a native object when you want to access the data.

Converting a string to a native object is called *deserialization*, while converting a native object to a string so it can be transmitted across the network is called *serialization*.

A JSON string can be stored in its own file, which is basically just a text file with an extension of .json, and a MIME type of application/json.
(https://developer.mozilla.org/en-US/docs/Glossary/MIME_type)

```
{
   "employee": {
      "name":      "sachin",
      "salary":    56000,
      "married":   true
   }
}
```

This is how a json file looks like. It contains a key and a value. Here employee is the key and the object in the curly brackets is the value. Also “name” is the key and “sachin” is the value.

```
{ "Employee" :
   [
    {"id":"101","name":"Sonoo Jaiswal","salary":"50000"},
    {"id":"102","name":"Vimal Jaiswal","salary":"60000"}
   ]
}
```

The square bracket represents a list, so here “Employee” is the key to an array of employee objects.

Serialization (Convert java class to JSON data):

```
public class Person {

    private String name;
    private String surname;
       .....
}
```

```
{"surname":"Swa",
 "name":"Android",
  ....
}
```

```
public JSONObject toJson() throws JSONException {
    JSONObject userJSON = new JSONObject();

    userJSON.put( name: "id", id);
    userJSON.put( name: "first_name", firstName);
    userJSON.put( name: "last_name", lastName);

    JSONArray moviesJsonArray = new JSONArray();
    for (Movie movie : movies) {
        moviesJsonArray.put(movie.toJson());
    }

    userJSON.put( name: "movies", moviesJsonArray);

    return userJSON;
}
```

```
public JSONObject toJson() throws JSONException {
    JSONObject movieJSON = new JSONObject();

    movieJSON.put( name: "id", id);
    movieJSON.put( name: "name", name);
    movieJSON.put( name: "description", description);

    return movieJSON;
}
```

```java
public class User {
    public int id;
    public String firstName;
    public String lastName;
    public List<Movie> movies;
```

```java
public class Movie {
    public int id;
    private String name;
    private String description;
```



```
{
  "id":123,
  "first_name":"Gabriel",
  "last_name":"Alexandru",
  "movies":[
    {
      "id":234,
      "name":"The Avengers",
      "description":"Earth's mightiest heroes must come together."
    },
    {
      "id":235,
      "name":"Star Wars: Episode I",
      "description":"Two Jedi escape a hostile blockade to find allies and come across a young boy who may bring balance to the Force."
    },
```

Deserialization (Parse JSON Data)

First, instantiate a parser that helps you to get the values inside the JSON

```java
JSONObject userJson = new JSONObject(data);
```

Then extract data from it to create the User class

```java
int id = userJson.getInt( name: "id");
String firstName = userJson.getString( name: "first_name");
String lastName = userJson.getString( name: "last_name");
```

Extract each movie as JSONObject from JSONArray and parse it

```java
JSONArray moviesList = userJson.getJSONArray( name: "movies");
for(int index = 0; index < moviesList.length(); index++) {
    JSONObject movieJSON = moviesList.getJSONObject(index);

    movies.add(new Movie().fromJSON(movieJSON));
}
```

GSON

Gson is a Java library that can be used to convert Java Objects into their JSON representation. It can also be used to convert a JSON string to an equivalent Java object. Gson can work with arbitrary Java objects including pre-existing objects that you do not have source-code of.

It provides simple toJson() and fromJson() methods to convert Java objects to JSON and vice-versa.

For more information and some examples:
https://sites.google.com/site/gson/gson-user-guide

Moshi

Moshi is a modern JSON library for Android and Java. It makes it easy to parse JSON into Java objects.

Moshi has a smaller API than other libraries like Jackson or Gson without compromising on functionality.

First, add moshi to your app dependency

```
implementation "com.squareup.moshi:moshi:1.11.0"
```

Moshi allows us to convert any Java values into JSON and back again anywhere we need to for whatever reasons – e.g. for file storage, writing REST APIs, whatever needs we might have.

Moshi works with the concept of a *JsonAdapter* class. This is a typesafe mechanism to serialize a specific class into a JSON string and to deserialize a JSON string back into the correct type:

```
public class MovieModel {
  private String title;
  private String duration;
}
Moshi moshi = new Moshi.Builder().build();
JsonAdapter<MovieModel> jsonAdapter = moshi.adapter(MovieModel.class);
```

Once you have built your *JsonAdapter*, we can use it whenever we need to in order to convert our values to JSON using the *toJson()* method:

```
MovieModel movie = new MovieModel("My MovieModel", "100min");
String json = jsonAdapter.toJson(movie);
// {"duration":"100min","imageId":0,"title":"My MovieModel"}
```

And convert JSON back into the expected Java types with the corresponding *fromJson()* method:

```
MovieModel movie2 = jsonAdapter.fromJson(json);
```

Retrofit

Retrofit is a type-safe REST client for Android, Java and Kotlin developed by Square. The library provides a powerful framework for authenticating and interacting with APIs and sending network requests with OkHttp.

It makes downloading JSON data from a web API fairly straightforward. Once the data is downloaded then it is parsed into a Plain Old Java Object (POJO) which must be defined for each "resource" in the response.

Setup

Make sure to require Internet permission in your AndoroidManifest.xml file:

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
  package="ro.example.myapplication">

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

Add the dependencies to your app/build.gradle file

```
implementation "com.squareup.retrofit2:retrofit:2.9.0"
implementation "com.squareup.okhttp3:logging-interceptor:5.0.0-alpha.2"
implementation 'com.squareup.okhttp3:okhttp:5.0.0-alpha.2'

//Moshi
implementation "com.squareup.moshi:moshi:1.11.0"
implementation "com.squareup.retrofit2:converter-moshi:2.9.0"
```

For this lab you will use an open API for movies: http://www.omdbapi.com/
You will see that in order to make a request you need an apiKey. They will provide it when you register on their platform, through an email.

Retrofit is a REST client, which means it is used on the client side (Android) to make HTTP requests to the REST API (in our case, The Movie OMDb API) and also process the response.

A REST API defines a set of functions which developers can perform requests and receive responses via HTTP protocol such as GET and POST.

To use Retrofit in your application you will need:

1. **An interface which defines the HTTP operations**

Annotations on the interface methods and its parameters indicate how a request will be handled.

Every method inside an interface represents one possible API call. It must have a HTTP annotation (GET, POST, etc.) to specify the request type and the relative URL. The return value wraps the response in a Call object with the type of the expected result.

```
@GET("users/list")
```

You can also specify query parameters in the URL.

```
@GET("users/list?sort=desc")
```

```
@GET("group/{id}/users")
Call<List<User>> groupList(@Path("id") int groupId, @Query("sort") String sort);
```

An object can be specified for use as an HTTP request body with the @Body annotation.

```
@POST("users/new")
Call<User> createUser(@Body User user);
```

For more antations: https://square.github.io/retrofit/

In this lab, you will turn the following API call of the OMDb Api into a Java method.

http://img.omdbapi.com/?apikey=[yourkey]&s=[search_string]

You can see it contains no relative URL and two query parameters: *apiKey* and *s*. It is a GET request.

```
public interface ApiService {
  @GET("/")
  Call<SearchMovieModel> getMovieList(
    @Query("apiKey") String apiKey,
    @Query("s") String search
  );
}
```

In order to create the data model you get as a response(SearchMovieModel class), you can simply change the values in the above link and open it in a new tab. *yourkey* will be the key you get via email and the *search_string* some text you want to search in the movie list.

For example: http://www.omdbapi.com/?apiKey=ca513130&s=avenger

What you see now is the JSON the api returns. To make it more readable, copy it and paste it in an online JSON parser. (http://json.parser.online.fr/)

You can see the *Search* key hold a list of movies. And a movie contains a *Title*, an *Year*, etc. Your models should look like this:

```
public class MovieModel {
  //    @Json(name = "titlu")
  private String Title;
  private String Year;
  private String imdbID;
  private String Poster;
}

public class SearchMovieModel {
  private List<MovieModel> Search;
  private Long totalResults;
}
```

The names of your class field must be the same with the one in the JSON, if you want them to be different, you can add the @Json annotation with the name of the field in the JSON and the field may hold another name.

2. **A Retrofit class which holds an implementation of the ApiService interface.**

To send out network requests to an API, we need to use the Retrofit builder class and specify the base URL for the service.

Note also that you need to specify a factory for deserializing the response using a library, in this case the Moshi library.

```
public class ApiBuilder {
  private static ApiService apiBuilder;
  private final static String BASE_URL = "http://www.omdbapi.com";
  public final static String API_KEY = "ca513130";

  static ApiService getInstance(){
    if(apiBuilder == null){
      HttpLoggingInterceptor interceptor = new HttpLoggingInterceptor();
      interceptor.setLevel(HttpLoggingInterceptor.Level.BODY);
      OkHttpClient client = new
OkHttpClient.Builder().addInterceptor(interceptor).build();

      Retrofit retrofit = new Retrofit.Builder()
          .baseUrl(BASE_URL)
          .client(client)
          .addConverterFactory(MoshiConverterFactory.create())
          .build();
```

```
            apiBuilder = retrofit.create(ApiService.class);
        }
        return apiBuilder;
    }
}
```

HttpLoggingInterceptor is used if you want to see logs about the requests in logcat. Interceptors are a powerful mechanism that can monitor, rewrite, and retry calls.

The interceptor must be added to the OkHttpClient and the client to the retrofit builder.

OkHttpClient is a factory for calls, which can be used to send HTTP requests and read their responses. OkHttp performs best when you create a single OkHttpClient instance and reuse it for all of your HTTP calls. This is because each client holds its own connection pool and thread pools. Reusing connections and threads reduces latency and saves memory.

Now that you got everything set up you can make the request. Add this to your activity.

```
Call<SearchMovieModel> call =
  ApiBuilder.getInstance().getMovieList(ApiBuilder.API_KEY, "avenger");
call.enqueue(new Callback<SearchMovieModel>() {
    @Override
    public void onResponse(Call<SearchMovieModel> call,
  Response<SearchMovieModel> response) {
        List<MovieModel> list = response.body().getSearch();
    }

    @Override
    public void onFailure(Call<SearchMovieModel> call, Throwable t) {

    }
});
```

If you get an error like:

HTTP FAILED: java.net.UnknownServiceException: CLEARTEXT communication to www.omdbapi.com not permitted by network security policy

Go to your manifest and add to your application tag the usesCleartextTraffic = “true” attribute. This error is showing because you try to make a request via http, instead of https, and it is not secure.

BroadcastReceiver

Android apps can send or receive broadcast messages from the Android system and other Android apps. These broadcasts are sent when an event of interest occurs. For example, the Android system sends broadcasts when various system events occur, such as when the system boots up or the device starts charging.

Apps can register to receive specific broadcasts. When a broadcast is sent, the system automatically routes broadcasts to apps that have subscribed to receive that particular type of broadcast.

The broadcast message itself is wrapped in an Intent object whose action string identifies the event that occurred (for example android.intent.action.AIRPLANE_MODE). The intent may also include additional information bundled into its extra field. For example, the airplane mode intent includes a boolean extra that indicates whether or not Airplane Mode is on.

Apps can receive broadcasts in two ways: through manifest-declared receivers and context-registered receivers.

Manifest-declared receivers

Important Note**: If your app targets API level 26 or higher, you cannot use the manifest to declare a receiver for *implicit* broadcasts (broadcasts that do not target your app specifically), except for a few implicit broadcasts that are exempted from that restriction.**

Create a subclass of BroadcastReceiver and implement onReceive(Context, Intent).

```
public class MyBroadcast extends BroadcastReceiver {

    @Override
    public void onReceive(Context context, Intent intent) {
        if (intent.getAction().equals(Intent.ACTION_BOOT_COMPLETED))
            ...
    }
}
```

Specify the `<receiver>` element in your app's manifest. The intent filters specify the broadcast actions your receiver subscribes to.

```
<receiver android:name=".MyBroadcast">
  <intent-filter>
    <action android:name="android.intent.action.ACTION_BOOT_COMPLETED" />
  </intent-filter>
</receiver>
```

The system package manager registers the receiver when the app is installed. The receiver then becomes a separate entry point into your app which means that the system can start the app and deliver the broadcast if the app is not currently running.

Context-registered receivers

You must create an instance of BroadcastReceiver, create an IntentFilter and register the receiver.

To stop receiving broadcasts, call `unregisterReceiver(BroadcastReceiver)`. Be sure to unregister the receiver when you no longer need it or the context is no longer valid.

Be mindful of where you register and unregister the receiver, for example, if you register a receiver in *onCreate(Bundle)* using the activity's context, you should unregister it in *onDestroy()* to prevent leaking the receiver out of the activity context. If you register a receiver in *onResume()*, you should unregister it in *onPause()* to prevent registering it multiple times

```
public class ClockBroadcast extends BroadcastReceiver {

  TickListener listener;
  void setupListener(TickListener listener){
     this.listener = listener;
  }

  @Override
  public void onReceive(Context context, Intent intent) {
     if (intent.getAction().equals(Intent.ACTION_TIME_TICK) && listener!=null)
        listener.tick();
  }
}
```

This broadcast listens to current time changing. The action is sent every minute.

```
ClockBroadcast receiver;
IntentFilter filter;

@Override
protected void onCreate(Bundle savedInstanceState) {
        ...
        initReceiver();
}
```

```
private void initReceiver(){
  receiver = new ClockBroadcast();
  receiver.setupListener(this);

  filter = new IntentFilter();
  filter.addAction(Intent.ACTION_TIME_TICK);
}

@Override
protected void onResume() {
  super.onResume();
  registerReceiver(receiver, filter);
}

@Override
protected void onPause() {
  super.onPause();
  unregisterReceiver(receiver);
}
```

A list of broadcast actions: https://developer.android.com/reference/android/content/Intent#standard-broadcast-actions

Sending broadcasts

- The sendOrderedBroadcast(Intent, String) method sends broadcasts to one receiver at a time. As each receiver executes in turn, it can propagate a result to the next receiver, or it can completely abort the broadcast so that it won't be passed to other receivers. The order receivers run in can be controlled with the android:priority attribute of the matching intent-filter; receivers with the same priority will be run in an arbitrary order.

- The sendBroadcast(Intent) method sends broadcasts to all receivers in an undefined order. This is called a Normal Broadcast. This is more efficient, but means that receivers cannot read results from other receivers, propagate data received from the broadcast, or abort the broadcast.

- The LocalBroadcastManager.sendBroadcast method sends broadcasts to receivers that are in the same app as the sender. If you don't need to send broadcasts across apps, use local broadcasts. The implementation is much more efficient (no interprocess communication needed) and you don't need to worry about any security issues related to other apps being able to receive or send your broadcasts.

```
private void sendBroadcast(){
  Intent intent = new Intent();
```

```
    intent.setAction("ro.example.broadcast.MY_NOTIFICATION");
    intent.putExtra("data", "Nothing to see here, move along.");
    sendBroadcast(intent);
}
```

To make the broadcast receiver unavailable to external applications, add the attribute android:exported=false in the manifest. When you send a broadcast, it is possible for the external applications too to receive them. This can be prevented by specifying this limitation.