

Mobile Application Development

Produced
by

David Drohan (ddrohan@wit.ie)

Department of Computing & Mathematics
Waterford Institute of Technology

<http://www.wit.ie>



Waterford Institute of Technology
INSTITIÚID TEICNEOLAÍOCHTA PHORT LÁIRGE





Android & Retrofit

A type-safe HTTP client for Android & Java





Retrofit

A type-safe HTTP client for Android and Java

Introduction

Retrofit turns your HTTP API into a Java interface.

```
public interface GitHubService {  
    @GET("users/{user}/repos")  
    Call<List<Repo>> listRepos(@Path("user") String user);  
}
```

The Retrofit class generates an implementation of the GitHubService interface.

```
Retrofit retrofit = new Retrofit.Builder()  
    .baseUrl("https://api.github.com/")  
    .build();  
  
GitHubService service = retrofit.create(GitHubService.class);
```

Each Call from the created GitHubService can make a synchronous or asynchronous HTTP request to the remote webserver.

```
Call<List<Repo>> repos = service.listRepos("octocat");
```

[Introduction](#)

[API Declaration](#)

[Retrofit Configuration](#)

[Download](#)

[Contributing](#)

[License](#)

[Javadoc](#)

[StackOverflow](#)



Agenda & Goals

- ❑ Investigate the use of Retrofit in App Development
- ❑ Be aware of the different Retrofit Annotations and Classes and how, when and where to use them
- ❑ Recap on Java interfaces
- ❑ Understand how to integrate Retrofit into an Android App
- ❑ Revisit our **CoffeeMate** Case Study



What is it?

- ❑ Retrofit is a Java Library that turns your REST API into a Java interface
- ❑ Simplifies HTTP communication by turning remote APIs into declarative, type-safe interfaces
- ❑ Developed by Square (Jake Wharton)
- ❑ Retrofit is one of the most popular HTTP Client Library for Android as a result of its simplicity and its great performance compared to the others (next slide)
- ❑ Retrofit makes use of **OkHttp** (from the same developer) to handle network requests.



Retrofit Performance Analysis

	One Discussion	Dashboard (7 requests)	25 Discussions
AsyncTask	941 ms	4,539 ms	13,957 ms
Volley	560 ms	2,202 ms	4,275 ms
Retrofit	312 ms	889 ms	1,059 ms



Why Use it?

- ❑ Developing your own type-safe HTTP library to interface with a REST API can be a real pain: you have to handle many functionalities such as making connections, caching, retrying failed requests, threading, response parsing, error handling, and more.
- ❑ Retrofit, on the other hand, is very well planned, documented, and tested—a battle-tested library that will save you a lot of precious time and headaches.



The Basics

- ❑ Again, Retrofit2 is a flexible library that uses annotated interfaces to create REST calls. To get started, let's look at our CoffeeMate example that makes a **GET** request for Coffees.
- ❑ Here's the **Coffee** class we're using:
- ❑ Much simpler if field names matches server model (but doesn't have to, see later)

```
public class Coffee
{
    public String name;
    public String shop;
    public double rating;
    public double price;
    public boolean favourite;

    public String _id;
    public String usertoken;
    public String address;
    public String googlephoto;
    public Marker marker = new Marker();

    public Coffee() {...}

    public Coffee(String name, String shop, double rating,
                 double price, boolean fav, String token,
                 double lat, double lng, String path)
    {...}

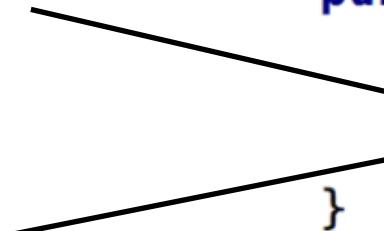
    @Override
    public String toString() {...}
}
```



The Service interface *

- Once we've defined the class, we can make a service interface to handle our API. A **GET** request to load all Coffees could look something like this:

```
public interface CoffeeService {  
    @GET("/coffees/{token}")  
    Call<List<Coffee>> getAllCoffees(@Path("token") String token);  
}
```



The code snippet shows a Java interface named `CoffeeService`. It contains a single method `getAllCoffees` which takes a `String token` as a parameter. The method is annotated with `@GET("/coffees/{token}")`, where `/coffees` is the endpoint and `{token}` is a placeholder for the parameter. Two arrows point from the annotations in the code to the corresponding text in the slide: one arrow points from the `@GET` annotation to the word `GET` in the explanatory text below, and another arrow points from the `{token}` placeholder to the word `token` in the explanatory text.

- Note that the `@GET` annotation takes the endpoint we wish to hit. As you can see, an implementation of this interface will return a `Call` object containing a list of Coffees.

Retrofit 2.0 introduced a new **Call** class (it also exists in OkHttp package).



Call

- Models a single request/response pair
- Separates request creation from response handling
- Each instance can only be used once...
- ...instances can be cloned
- Supports both synchronous and asynchronous execution.
- Can be (actually) canceled



Beyond GET – other types of Calls

- ❑ Retrofit2 is not limited to GET requests. You may specify other REST methods using the appropriate annotations (such as @POST, @PUT and @DELETE).

- ❑ Here's another version of our **CoffeeService** interface (with full CRUD support)

```
public interface CoffeeService {  
  
    @GET("/coffees/{token}")  
    Call<List<Coffee>> getAllCoffees(@Path("token") String token);  
  
    @POST("/coffees/{token}")  
    Call<Coffee> createCoffee(@Path("token") String token,  
                               @Body Coffee coffee);  
  
    @GET("/coffees/{token}/{id}")  
    Call<Coffee> retrieveCoffee(@Path("token") String token,  
                               @Path("id") String id);  
  
    @PUT("/coffees/{token}/{id}")  
    Call<Coffee> updateCoffee(@Path("token") String token,  
                               @Path("id") String id,  
                               @Body Coffee coffee);  
  
    @DELETE("/coffees/{token}/{id}")  
    Call<List<Coffee>> deleteCoffee(@Path("token") String token,  
                                       @Path("id") String id);  
}
```



Calling the API

- ❑ So how do we use this interface to make requests to the API?
- ❑ Use **Retrofit2** to create an implementation of the above interface, and then call the desired method.
- ❑ Retrofit2 supports a number of converters used to map Java objects to the data format your server expects (JSON, XML, etc). we'll be using the **Gson** converter.

- **Gson**: com.squareup.retrofit2:converter-gson
- **Jackson**: com.squareup.retrofit2:converter-jackson
- **Moshi**: com.squareup.retrofit2:converter-moshi
- **Protobuf**: com.squareup.retrofit2:converter-protobuf
- **Wire**: com.squareup.retrofit2:converter-wire
- **Simple XML**: com.squareup.retrofit2:converter-simplexml
- **Scalars (primitives, boxed, and String)**: com.squareup.retrofit2:converter-scalars



Aside - CoffeeMate Android Client

❑ coffeeamateweb api endpoints

```
{ method: 'GET', path: '/coffees/{token}', config: CoffeeApi.findAll },
{ method: 'GET', path: '/coffees/{token}/{id}', config: CoffeeApi.findById },
{ method: 'POST', path: '/coffees/{token} ', config: CoffeeApi.addCoffee },
{ method: 'PUT', path: '/coffees/{token}/{id}', config: CoffeeApi.updateCoffee },
{ method: 'DELETE', path: /coffees/{token}/{id}', config: CoffeeApi.deleteCoffee }
```

❑ Use CoffeeService for

- ❑ Adding / Updating / Deleting a Coffee
- ❑ Listing All Coffees
- ❑ Finding a single Coffee



Android Networking (Using Retrofit) in CoffeeMate.5.1



Steps to integrate Retrofit into your App

1. Set up your Project Dependencies & Permissions
2. Create Interface for API and declare methods for each REST Call, specifying method type using Annotations -
 @GET, @POST, @PUT, etc. For parameters use - @Path,
 @Query, @Body
3. Use **Retrofit** to build the service client
4. Make the REST Calls as necessary using the relevant Callback mechanism



1. Project Dependencies & Permissions *

- Add the required dependencies to your build.gradle

```
dependencies {  
    compile fileTree(dir: 'libs', include: ['*.jar'])  
    compile 'com.android.support:appcompat-v7:26.1.0'  
    compile 'com.android.support:support-v4:26.1.0'  
    compile 'com.android.support:design:26.1.0'  
    compile 'com.makeramen:roundedimageview:2.2.1'  
    compile 'com.android.support.constraint:constraint-layout:1.0.2'  
    testCompile 'junit:junit:4.12'  
    compile project(':volley')  
    compile 'com.google.code.gson:gson:2.8.2'  
    compile 'com.google.android.gms:play-services-auth:11.8.0'  
  
    compile 'com.squareup.retrofit2:retrofit:2.1.0'  
    compile 'com.squareup.retrofit2:converter-gson:2.1.0'  
    compile 'com.squareup.okhttp3:okhttp:3.7.0'  
}
```

- Add the necessary permissions to your manifest – BEFORE/OUTSIDE the application tag

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



2. Create interface for API

The screenshot shows the Android Studio interface. On the left, the Project and Structure toolbars are visible. The Project tree shows a project named "CoffeeMate.5.1" with an "app" module. Inside "app", there are "manifests", "java", "res", "volley", and "Gradle Scripts" directories. The "java" directory contains "ie.cm" and "api" packages. "api" contains "CoffeeApi" and "CoffeeService". "CoffeeService" is highlighted with a blue selection bar. Below "api" are "fragments" and "main" packages, which contain "CoffeeMateApp". Under "models", there are "Coffee", "Coords", and "Marker" classes. At the bottom, "ie.cm" has sub-directories for "androidTest" and "test". The "res" and "volley" directories are also listed.

```
public interface CoffeeService {  
  
    @GET("/coffees/{token}")  
    Call<List<Coffee>> getAllCoffees(@Path("token") String token);  
  
    @POST("/coffees/{token}")  
    Call<Coffee> createCoffee(@Path("token") String token,  
                               @Body Coffee coffee);  
  
    @GET("/coffees/{token}/{id}")  
    Call<Coffee> retrieveCoffee(@Path("token") String token,  
                               @Path("id") String id);  
  
    @PUT("/coffees/{token}/{id}")  
    Call<Coffee> updateCoffee(@Path("token") String token,  
                               @Path("id") String id,  
                               @Body Coffee coffee);  
  
    @DELETE("/coffees/{token}/{id}")  
    Call<List<Coffee>> deleteCoffee(@Path("token") String token,  
                                      @Path("id") String id);  
}
```



3. Build Service Client - CoffeeMateApp *

- ☐ Implement the necessary interface & variables

```
public class CoffeeMateApp extends Application
{
    //...
    public CoffeeService coffeeService;
    //...
    public String serviceURL = "http://coffeemateweb.herokuapp.com";
}
```

Our
CoffeeService
instance

Note the **serviceURL**



3. Build Service Client - CoffeeMateApp *

- Create the proxy service '**coffeeService**', with the appropriate Gson parsers

```
@Override  
public void onCreate() {  
    super.onCreate();  
  
    //...  
    Gson gson = new GsonBuilder().create();  
  
    OkHttpClient okHttpClient = new OkHttpClient.Builder()  
        .connectTimeout( timeout: 30, TimeUnit.SECONDS )  
        .writeTimeout( timeout: 30, TimeUnit.SECONDS )  
        .readTimeout( timeout: 30, TimeUnit.SECONDS )  
        .build();  
  
    Retrofit retrofit = new Retrofit.Builder()  
        .baseUrl(serviceURL)  
        .addConverterFactory(GsonConverterFactory.create(gson))  
        .client(okHttpClient)  
        .build();  
  
    coffeeService = retrofit.create(CoffeeService.class);  
  
    Log.v( tag: "coffeemate", msg: "Coffee Service Created" );  
}
```



3. Build Service Client - CoffeeMateApp *

Gson for converting our JSON

OkHttpClient for communication timeouts (optional)

Retrofit.Builder to create an instance of our interface

```
@Override  
public void onCreate() {  
    super.onCreate();  
  
    //...  
    Gson gson = new GsonBuilder().create();  
  
    OkHttpClient okHttpClient = new OkHttpClient.Builder()  
        .connectTimeout( timeout: 30, TimeUnit.SECONDS )  
        .writeTimeout( timeout: 30, TimeUnit.SECONDS )  
        .readTimeout( timeout: 30, TimeUnit.SECONDS )  
        .build();  
  
    Retrofit retrofit = new Retrofit.Builder()  
        .baseUrl(serviceURL)  
        .addConverterFactory(GsonConverterFactory.create(gson))  
        .client(okHttpClient)  
        .build();  
  
    coffeeService = retrofit.create(CoffeeService.class);  
  
    Log.v( tag: "coffeemate", msg: "Coffee Service Created");  
}
```



4. Calling the API - CoffeeFragment *

- ☐ Implement the necessary interface & variables

```
public class CoffeeFragment extends Fragment implements AdapterView.OnItemClickListener,  
    View.OnClickListener,  
    AbsListView.MultiChoiceModeListener,  
    CallBack<List<Coffee>>  
{  
    //...  
  
    public Call<List<Coffee>> call;  
  
    public CoffeeMateApp app = CoffeeMateApp.getInstance();
```

the **call** object, for
making the requests

Note the
Callback
interface



4. CoffeeFragment – onResume() *

```
@Override  
public void onResume() {  
    super.onResume();  
  
    //...  
  
    call = (Call<List<Coffee>>) app.coffeeService.getAllCoffees(app.googleToken);  
    call.enqueue(callback: this);  
}
```

enqueue() allows
for asynchronous
callback to our service



4. CoffeeFragment – onResponse() *

- ❑ Triggered on a successful call to the API
- ❑ Takes 2 parameters
 - ❑ The **Call** object
 - ❑ The expected **Response** object
- ❑ Converted JSON result stored in **response.body()**

```
@Override  
public void onResponse(Call<List<Coffee>> call, Response<List<Coffee>> response) {  
    app.coffeeList = response.body();  
  
    listAdapter = new CoffeeListAdapter(getActivity(), deleteListener: this, app.coffeeList);  
    coffeeFilter = new CoffeeFilter(app.coffeeList, filterText: "all", listAdapter);  
    setListView(listView);  
  
    //...  
}
```



4. CoffeeFragment – onFailure() *

- ❑ Triggered on an unsuccessful call to the API
- ❑ Takes 2 parameters
 - ❑ The **Call** object
 - ❑ A **Throwable** object containing error info
- ❑ Probably should inform user of what's happened

```
@Override  
public void onFailure(Call<List<Coffee>> call, Throwable t) {  
    Toast.makeText(getActivity(),  
        text: "Coffee Service Unavailable. Try again later",  
        Toast.LENGTH_LONG).show();  
}
```



4. Anonymous Callbacks *

Anonymous Callback
allows for multiple calls
in same class

```
private void deleteACoffee(String id) {  
    Call<List<Coffee>> callDelete = app.coffeeService.deleteCoffee(app.googleToken, id);  
    callDelete.enqueue(new Callback<List<Coffee>>() {  
        @Override  
        public void onResponse(Call<List<Coffee>> call, Response<List<Coffee>> response) {  
            Toast.makeText(getApplicationContext(), text: "Successfully Deleted Coffee" ,  
                Toast.LENGTH_LONG).show();  
        }  
  
        @Override  
        public void onFailure(Call<List<Coffee>> call, Throwable t) {  
  
            Toast.makeText(getApplicationContext(), text: "Unable to Delete Coffee : " + "ERROR: " +  
                t.getMessage(), Toast.LENGTH_LONG).show();  
        }  
    });  
}
```



CoffeeMateWeb + Mobile

The image displays the CoffeeMate application across three platforms: a mobile phone emulator, another mobile phone emulator, and a desktop web browser.

Mobile Phone Emulator 1 (Left): Shows the "Coffee Check In" screen. It includes fields for "Name : Test Name", "Shop : Test Shop", "Price : 1.99", and a "Star Rating" section with four blue stars. A "SAVE COFFEE" button is at the bottom.

Mobile Phone Emulator 2 (Middle): Shows the "Recently Added Coffee's" screen. It lists six coffee entries with details like name, shop, price, star rating, and a delete icon. A "Successfully Added Coffee" message is visible at the bottom.

Desktop Browser (Right): Shows the "List All Coffees" page from the CoffeeMateWeb application. The page has a header with navigation links like Home, Add, My Coffees, View on Map, About, Contact, and user info (daveydrohan@gmail.com). Below the header is a table listing coffee entries:

	Name	Shop	Price	Actions
4 ⭐	Cuppa Ta	Mam's Place	€0.99	Remove Edit
4 ⭐	Black Coffee	Ardkeen Stores	€2.59	Remove Edit
3.5 ⭐	new nn	my shop	€2.5	Remove Edit
3 ⭐	Test Name	Test Shop	€1.99	Remove Edit
3 ⭐	Mocca Latte	Bishops Place	€1.49	Remove Edit
1 ⭐	Coffee Latte	Tesco	€2.99	Remove Edit

At the bottom of the page, there is a link to "View the tutorial on [GitBook](#)".



Headers

- ❑ If you wish to add headers to your calls, you can annotate your method or arguments to indicate this.
- ❑ For instance, if we wanted to add a content type and a specific user's authentication token, we could do something like this:

```
@POST("/coffees")
@Headers({ "Content-Type: application/json" })
Call<Coffee> createCoffee(@Body Coffee coffee,
                           @Header("Authorization") String token);
```



Bridging the Gap Between Your Code & Your API

❑ Variable Names

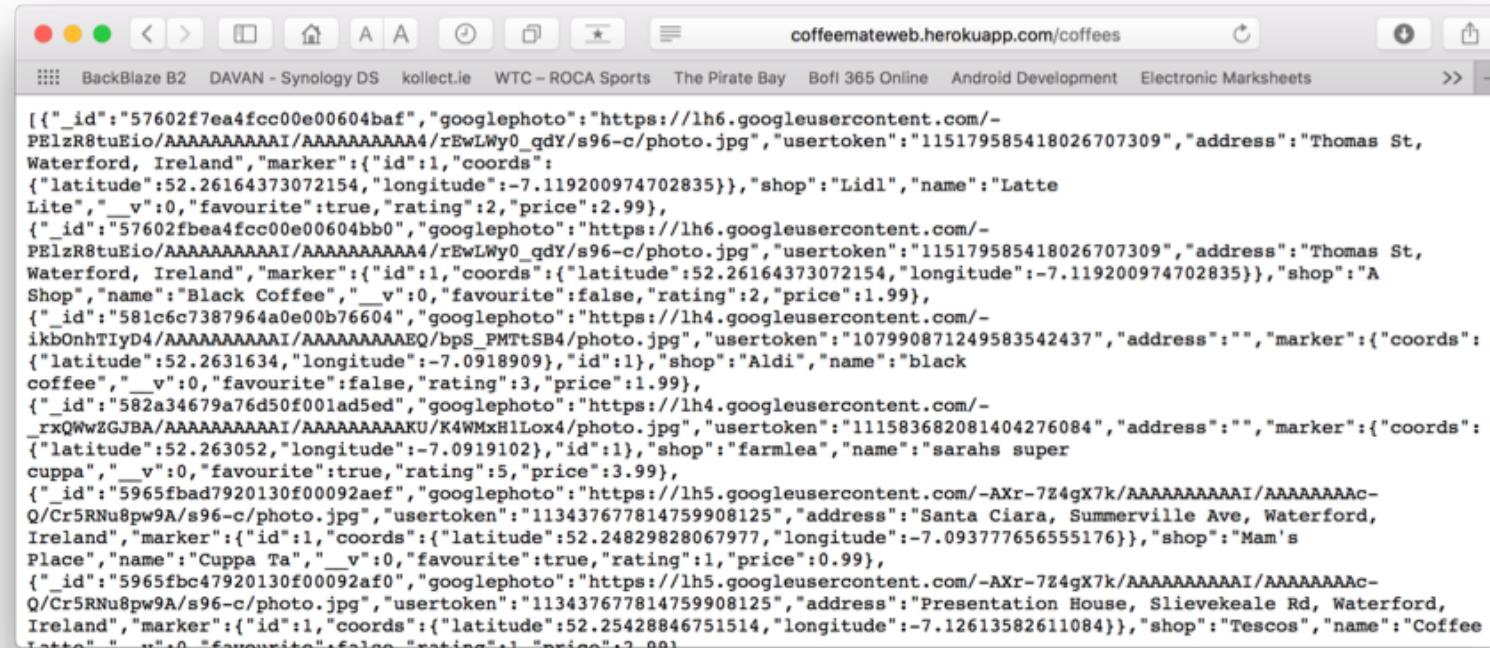
- ❑ In the previous examples, we assumed that there was an exact mapping of instance variable names between the Coffee class and the server. This will often not be the case, especially if your server uses a different spelling convention than your Android app.
- ❑ For example, if you use a Rails server, you will probably be returning data using `snake_case`, while your Java probably uses `camelCase`. If we add a `dateCreated` to the Coffee class, we may be receiving it from the server as `date_created`.
- ❑ To create this mapping, use the `@SerializedName` annotation on the instance variable. For example:

```
@SerializedName("date_created")
private Date dateCreated;
```



Bridging the Gap Between Your Code & Your API

- You can also create your models automatically from your JSON response data by leveraging a very useful tool:
jsonschema2pojo - <http://www.jsonschema2pojo.org>
- Grab your JSON string, visit the above link and paste it in, like so



```
[{"_id": "57602f7ea4fcc00e00604baf", "googlephoto": "https://lh6.googleusercontent.com/-PElzR8tuEio/AAAAAAAAAAI/AAAAAAAABAA/rEwLWyo0_qdY/s96-c/photo.jpg", "usertoken": "115179585418026707309", "address": "Thomas St, Waterford, Ireland", "marker": {"id": 1, "coords": {"latitude": 52.26164373072154, "longitude": -7.119200974702835}}, "shop": "Lidl", "name": "Latte Lite", "v": 0, "favourite": true, "rating": 2, "price": 2.99}, {"_id": "57602fbea4fcc00e00604bb0", "googlephoto": "https://lh6.googleusercontent.com/-PElzR8tuEio/AAAAAAAAAAI/AAAAAAAABAA/rEwLWyo0_qdY/s96-c/photo.jpg", "usertoken": "115179585418026707309", "address": "Thomas St, Waterford, Ireland", "marker": {"id": 1, "coords": {"latitude": 52.26164373072154, "longitude": -7.119200974702835}}, "shop": "A Shop", "name": "Black Coffee", "v": 0, "favourite": false, "rating": 2, "price": 1.99}, {"_id": "581c6c7387964a0e00b76604", "googlephoto": "https://lh4.googleusercontent.com/-ikbOnhTIyD4/AAAAAAAAAAI/AAAAAAAEEQ/bpS_PMTTSB4/photo.jpg", "usertoken": "107990871249583542437", "address": "", "marker": {"coords": {"latitude": 52.2631634, "longitude": -7.0918909}, "id": 1}, "shop": "Aldi", "name": "black coffee", "v": 0, "favourite": false, "rating": 3, "price": 1.99}, {"_id": "582a34679a76d50f001ad5ed", "googlephoto": "https://lh4.googleusercontent.com/-rxQWwZGJBA/AAAAAAAAAAI/AAAAAAAAKU/K4WMxHlLox4/photo.jpg", "usertoken": "111583682081404276084", "address": "", "marker": {"coords": {"latitude": 52.263052, "longitude": -7.0919102}, "id": 1}, "shop": "farmlea", "name": "sarahs super cuppa", "v": 0, "favourite": true, "rating": 5, "price": 3.99}, {"_id": "5965fbad7920130f00092aef", "googlephoto": "https://lh5.googleusercontent.com/-AXr-7Z4gX7k/AAAAAAAAAAI/AAAAAAAAC-Q/Cr5RNu8pw9A/s96-c/photo.jpg", "usertoken": "113437677814759908125", "address": "Santa Ciara, Summerville Ave, Waterford, Ireland", "marker": {"id": 1, "coords": {"latitude": 52.24829828067977, "longitude": -7.093777656555176}}, "shop": "Mam's Place", "name": "Cuppa Ta", "v": 0, "favourite": true, "rating": 1, "price": 0.99}, {"_id": "5965fbc47920130f00092af0", "googlephoto": "https://lh5.googleusercontent.com/-AXr-7Z4gX7k/AAAAAAAAAAI/AAAAAAAAC-Q/Cr5RNu8pw9A/s96-c/photo.jpg", "usertoken": "113437677814759908125", "address": "Presentation House, Slievekeale Rd, Waterford, Ireland", "marker": {"id": 1, "coords": {"latitude": 52.25428846751514, "longitude": -7.12613582611084}}, "shop": "Tescos", "name": "Coffee Totto", "v": 0, "favourite": false, "rating": 1, "price": 0.99}
```

Bridging the Gap Between Your Code & Your API

Your JSON goes here

The screenshot shows a web browser window for the [jsonschema2pojo](http://www.jsonschema2pojo.org) website. The page title is "jsonschema2pojo" and the subtitle is "Generate Plain Old Java Objects from JSON or JSON-Schema.". A sample JSON code is pasted into the main input area:

```
1 { "_id": "57602f7ea4fcc00e00604baf",
2   "googlephoto": "https://lh6.googleusercontent.com/-PElzR8tu
3   "usertoken": "115179585418026707309",
4   "address": "Thomas St, Waterford, Ireland",
5   "marker": { "id": 1, "coords": { "latitude": 52.26164373072154, "longitude": -8.351452555555556 },
6   "shop": "Lidl",
7   "name": "Latte Lite",
8   "v": 0,
9   "favourite": true,
10  "rating": 2,
11  "price": 2.99 },
12 { "_id": "57602fbea4fcc00e00604bb0", "googlephoto": "https://lh6.googleusercontent.com/-PElzR8tu
13 }
```

The configuration sidebar on the right includes the following fields:

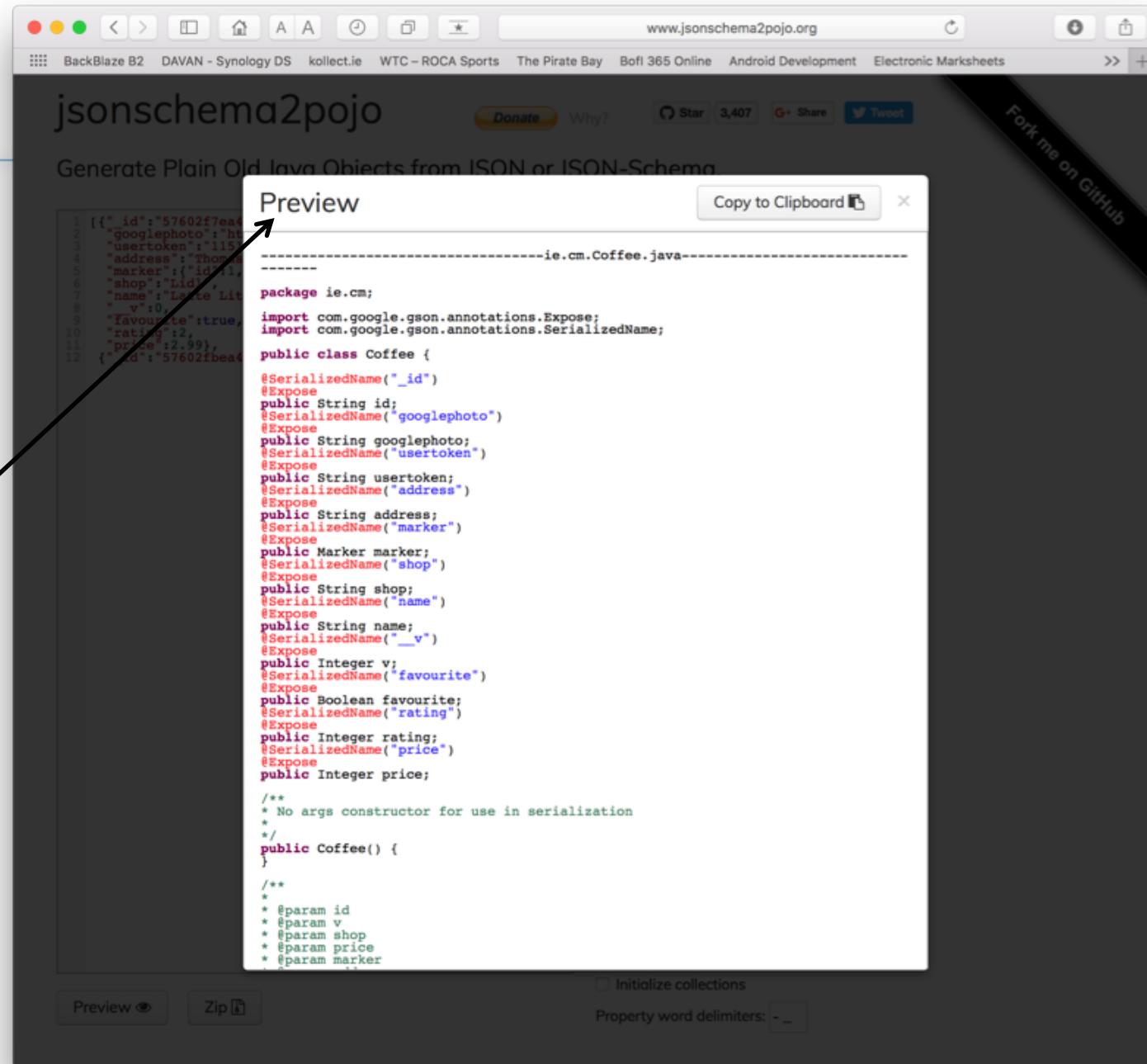
- Package: `ie.cm`
- Class name: `Coffee`
- Target language:
 Java Scala
- Source type:
 JSON Schema JSON
 YAML Schema YAML
- Annotation style:
 Jackson 2.x Jackson 1.x
 Gson Moshi None
- Checkboxes (unchecked):
Generate builder methods
Use primitive types
Use long integers
Use double numbers
Use Joda dates
Use Commons-Lang3
Include getters and setters
Include constructors
Include `hashCode` and `equals`
Include `toString`
Include JSR-303 annotations
Allow additional properties
Make classes serializable
Make classes parcelable
Initialize collections
- Property word delimiters: `- _`

At the bottom are "Preview" and "Zip" buttons.



Bridging the Gap Between Your Code & Your API

Your annotated class





Bridging the Gap Between Your Code & Your API

□ Date Formats

- ❑ Another potential disconnect between the app and the server is the way they represent date objects.
- ❑ For instance, Rails may send a date to your app in the format "yyyy-MM-dd'T'HH:mm:ss" which Gson will not be able to convert to a Java **Date**. We have to explicitly tell the converter how to perform this conversion. In this case, we can alter the Gson builder call to look like this:

```
Gson gson = new GsonBuilder()  
        .setDateFormat("yyyy-MM-dd'T'HH:mm:ss")  
        .create();
```



Full List of Retrofit Annotations

- @POST
- @PUT
- @GET
- @DELETE
- @Header
- @PATCH

- @Path
- @Query
- @Body



References

- ❑ <http://square.github.io/retrofit/>
- ❑ <https://spin.atomicobject.com/2017/01/03/android-rest-calls-retrofit2/>
- ❑ <https://code.tutsplus.com/tutorials/getting-started-with-retrofit-2--cms-27792>

