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");
```

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 Donation 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 Donation example that makes a GET request for Donations.
- Here's the Donation class we'll be using:
- Much simpler if field names matches server model (but doesn't have to, see later)

```
public class Donation
{
    public String   _id;
    public int      amount;
    public String   paymenttype;
    public int      upvotes;
    public double   latitude;
    public double   longitude;


    public Donation (int amount, String paymenttype,
                     int upvotes, double lat, double lng)
    {
        this.amount = amount;
        this.paymenttype = paymenttype;
        this.upvotes = upvotes;
        this.latitude = lat;
        this.longitude = lng;
    }

    public Donation ()
    {...}

    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 Donations could look something like this:

```
public interface DonationService
{
    @GET("/donations")
    Call<List<Donation>> getAllDonations();
}
```

- 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 Donations.

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

(screen is taken from Jack Whartons presentation during Droidcon NYC 2015, available here)

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`

Calling the API *

- First, implement the necessary interface & variables

```
public class DonationApp extends Application
                         implements Callback<List<Donation>>
{
    public final int        target        = 10000;
    public int              totalDonated  = 0;
    public List <Donation>  donations     = new ArrayList<Donation>();
    //public DBManager dbManager;

    public DonationService donationService;
    public boolean         donationServiceAvailable = false;
    public String          service_url   = "http://donationweb-4-0.herokuapp.com/";
    //public String          service_url   = "http://10.0.2.2:4000";
                                           //Standard Emulator IP Address
```

Note the **Callback** interface

Note the **service_url**

DonationApp - `onCreate()`

- Create the proxy service ‘donationService’, with the appropriate Gson parsers

```
@Override
public void onCreate()
{
  super.onCreate();
  Gson gson = new GsonBuilder().create();

  Retrofit retrofit = new Retrofit.Builder()
      .baseUrl(service_url)
      .addConverterFactory(GsonConverterFactory.create(gson))
      .build();
  donationService = retrofit.create(DonationService.class);

  Log.v("Donation", "Donation App Started");
}
```

Calling the API – `onCreate()` *

`Gson` for converting our JSON

```
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();
```

`OkHttpClient` for communication timeouts (optional)

`Retrofit.Builder` to create an instance of our interface

```
Retrofit retrofit = new Retrofit.Builder()
        .baseUrl(service_url)
        .addConverterFactory(GsonConverterFactory.create(gson))
        .client(okHttpClient)
        .build();
```

```
donationService = retrofit.create(DonationService.class);
```

```
Log.v( tag: "Donate",  msg: "Donation Service Created");
```

`enqueue()` allows for asynchronous callback to our service

```
Call<List<Donation>> call1 = (Call<List<Donation>>) donationService.getAllDonations();
call1.enqueue( callback: this);
```

Calling the API – `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<Donation>> call, Response<List<Donation>> response) {
    serviceAvailableMessage();
    Log.v( tag: "retrofit", msg: "JSON = " + response.raw());
    donations = response.body();
    donationServiceAvailable = true;
}
```

Calling the API – `onFailure()` *

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

```
@Override
public void onFailure(Call<List<Donation>> call, Throwable t) {
    donationServiceAvailable = false;
    Log.v( tag: "retrofit",t.getMessage());
    serviceUnavailableMessage();
}
```

Calling the API – Helper methods

- Our 2 helper methods for user feedback

```
void serviceUnavailableMessage()
{
    Toast toast = Toast.makeText( context: this,
            text: "Donation Service Unavailable. Try again later",
            Toast.LENGTH_LONG);
    toast.show();
}

void serviceAvailableMessage()
{
    Toast toast = Toast.makeText( context: this,
            text: "Donation Contacted Successfully",
            Toast.LENGTH_LONG);
    toast.show();
}
```

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 **DonationService** interface

```
public interface DonationService
{
    @GET("/donations")
    Call<List<Donation>> getAllDonations();

    @GET("/donations/{id}")
    Call<List<Donation>> getDonation(@Path("id") String id);

    @DELETE("/donations/{id}")
    Call<Donation> deleteDonation(@Path("id") String id);

    @POST("/donations")
    Call<Donation> createDonation(@Body Donation donation);

    @PUT("/donations/{id}/votes")
    Call<Donation> upVoteDonation(@Path("id") String id,
                                  @Body Donation donation);
}
```

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 user's authentication token, we could do something like this:

```
@POST("/donations")
@Headers({"Content-Type: application/json"})
Call<Donation> createDonation(@Body Donation donation,
                              @Header("Authorization") String token);
```

Bridging the Gap Between Your Code & Your API

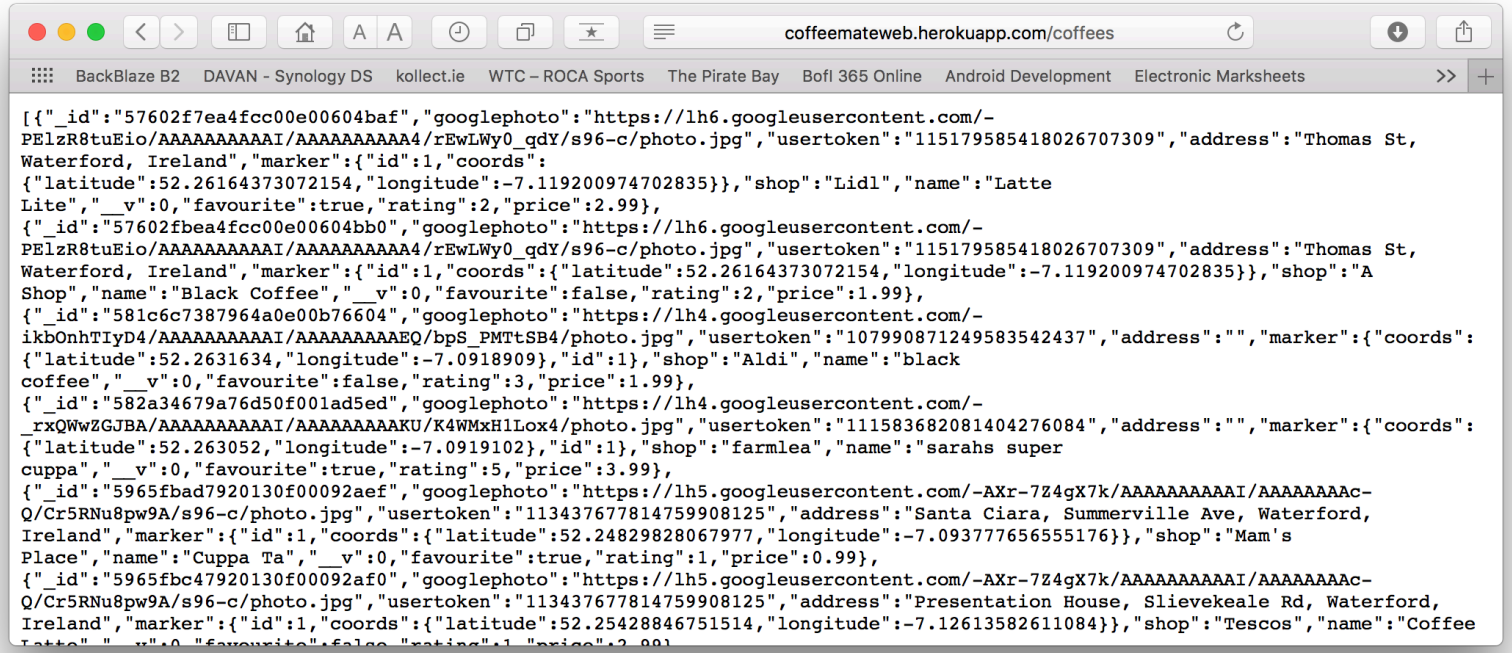
Variable Names

- In the previous examples, we assumed that there was an exact mapping of instance variable names between the Donation 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 Donation 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;
```

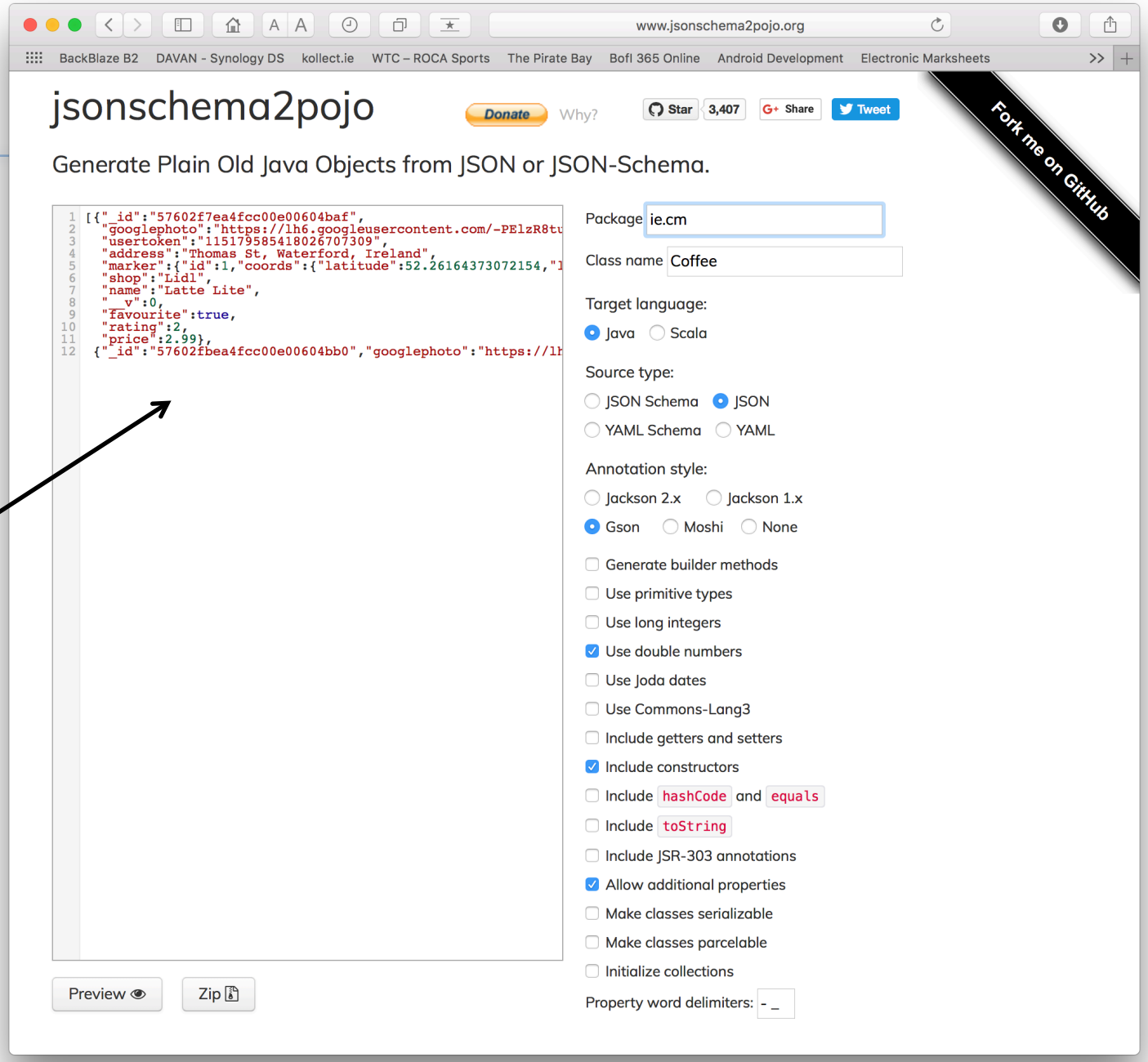
Bridging the Gap Between Your Code & Your API

- 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



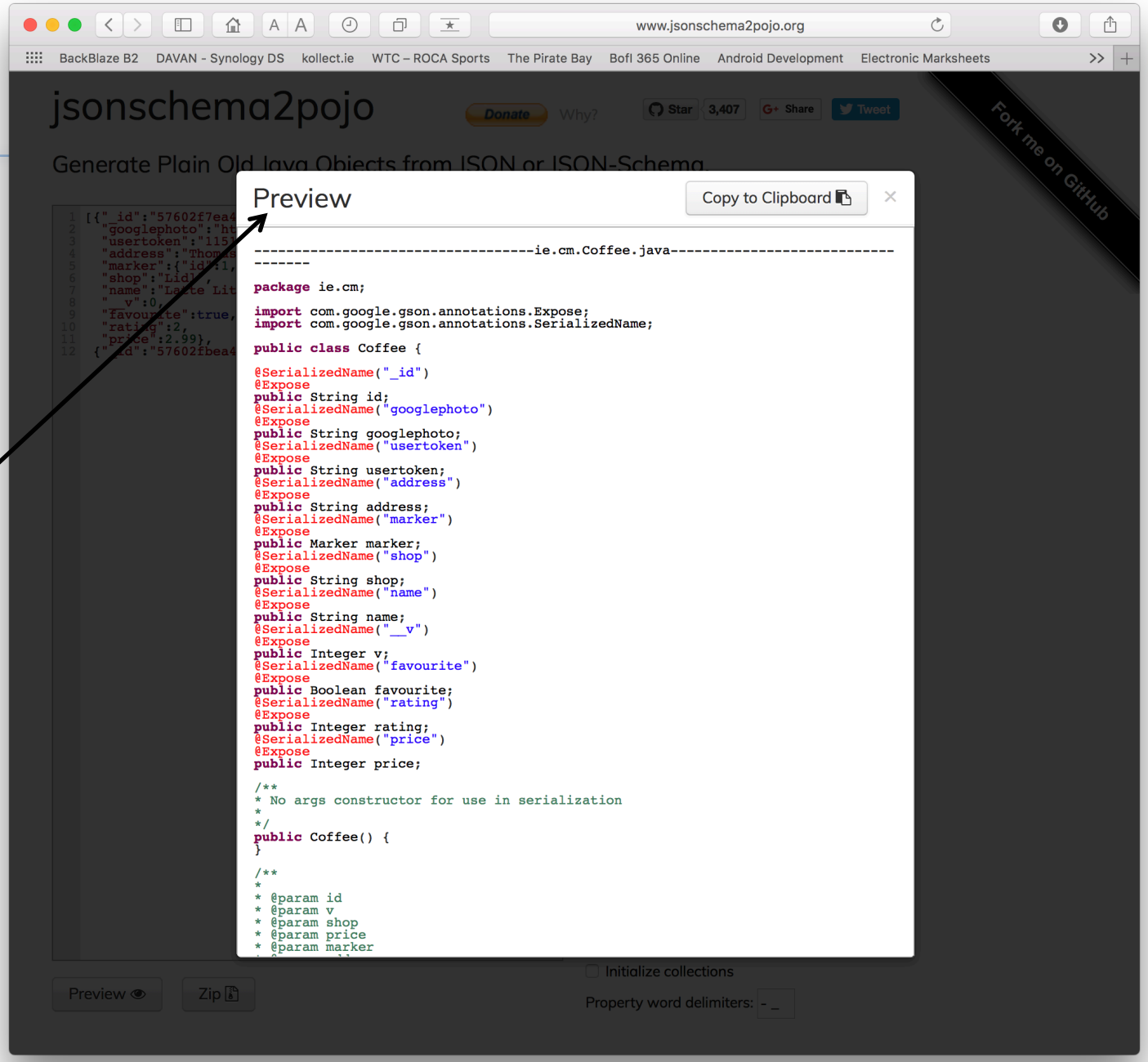
Bridging the Gap Between Your Code & Your API

Your JSON goes here



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

A bit about JSON

Question?

- Given a particular set of data, how do you store it permanently?
  - What do you store on disk?
  - What format?
  - Can you easily transmit over the web?
  - Will it be readable by other languages?
  - Can humans read the data?
- Examples:
  - A Square
  - A Dictionary
  - A Donation…

Storage Using Plain Text?

- Advantages
  - Human readable (good for debugging / manual editing)
  - Portable to different platforms
  - Easy to transmit using web

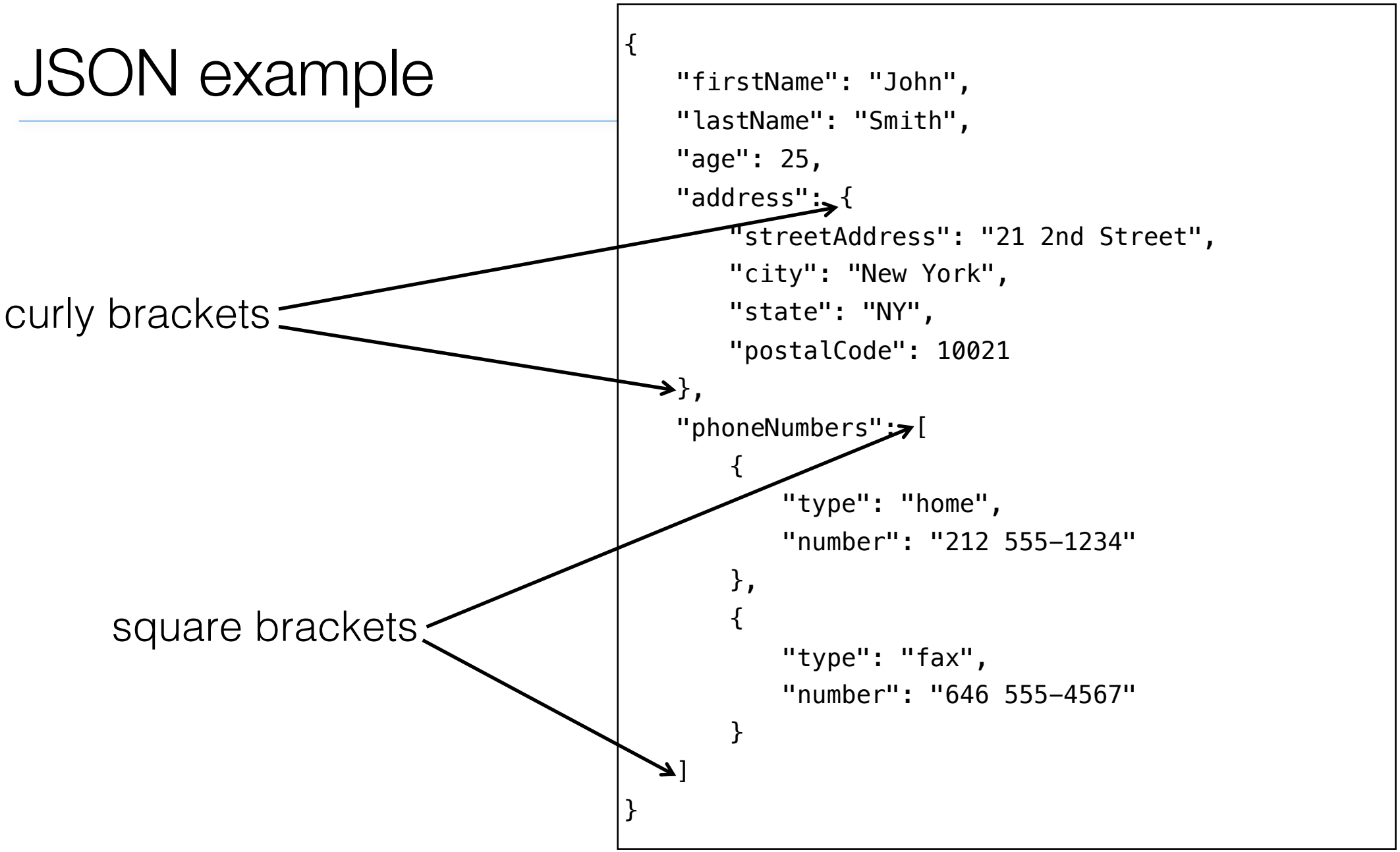
- Disadvantages
  - Takes more memory than necessary

- Alternative? - use a standardized system -- **JSON**
  - Makes the information more portable

JavaScript Object Notation – What is it?

- JSON is lightweight text-data interchange format
- JSON is language independent*
  - *JSON uses JavaScript syntax for describing data objects
    - JSON parsers and JSON libraries exists for many different programming languages.
- JSON is "self-describing" and easy to understand
- **JSON - Evaluates to JavaScript Objects**
  - The JSON text format is syntactically identical to the code for creating JavaScript objects.
  - Because of this similarity, instead of using a parser, a JavaScript program can use the built-in **eval()** function and execute JSON data to produce native JavaScript objects.

JSON example



A Donation in JSON


```
{
"_id":"5a23238c3032e512006c6de1",
"longitude”:-7.08867073059082,
"latitude":52.25166129217113,
"amount":1001,
"paymenttype":"PayPal",
"__v":0,
"upvotes":1
}
```

JSON Values

JSON values can be:

- A number (integer or floating point)
- A string (in double quotes)
- A boolean (true or false)
- An *object* (in curly brackets)
- An *array* (in square brackets)
- null

JSON Values

- Object
  - Unordered set of name-value pairs
  - names must be strings
  - { name1 : value1, name2 : value2, …, nameN : valueN }

- Array
  - Ordered list of values
  - [value1, value2, … valueN]

Google’s Gson

https://sites.google.com/site/gson/gson-user-guide

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 is an open-source project hosted at http://code.google.com/p/google-gson.

Gson can work with arbitrary Java objects including pre-existing objects that you do not have source-code of.

Android Networking (Using Retrofit) in Donation.4.1

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.

1. Project Dependencies & Permissions *

- Add the required dependencies to your build.gradle

```
dependencies {
    compile fileTree(dir: 'libs', include: ['*.jar'])
    testCompile 'junit:junit:4.12'
    compile 'com.android.support:appcompat-v7:25.4.0'
    compile 'com.android.support:design:25.4.0'
    compile 'com.squareup.retrofit2:retrofit:2.1.0'
    compile 'com.google.code.gson:gson:2.8.2'
    compile 'com.squareup.retrofit2:converter-gson:2.1.0'
    compile 'com.squareup.okhttp3:okhttp:3.7.0'
}
```

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

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

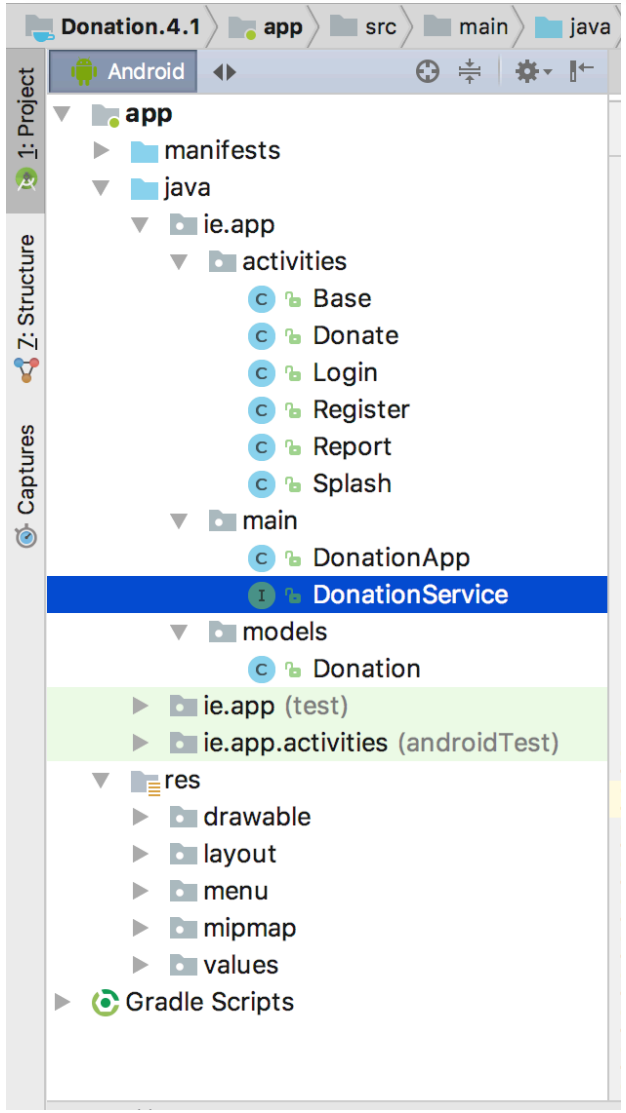
Aside - Donation Android Client

- **donation-web** api endpoints

```
{ method: 'GET', path: '/donations', config: DonationsApi.findAllDonations },
{ method: 'GET', path: '/donations/{id}', config: DonationsApi.findOneDonation },
{ method: 'POST', path: '/donations', config: DonationsApi.makeDonation },
{ method: 'PUT', path: '/donations/{id}/votes', config: DonationsApi.incrementUpvotes },
{ method: 'DELETE', path: '/donations/{id}', config: DonationsApi.deleteOneDonation }
```

- Use **donation-service** for
  - Donate
  - List All Donations
  - Find one Donation
  - Delete one Donation
  - Upvote a Donation

2. Create interface for API




```
public interface DonationService
{

    @GET("/donations")
    Call<List<Donation>> getAllDonations();

    @GET("/donations/{id}")
    Call<List<Donation>> getDonation(@Path("id") String id);

    @DELETE("/donations/{id}")
    Call<Donation> deleteDonation(@Path("id") String id);

    @POST("/donations")
    Call<Donation> createDonation(@Body Donation donation);

    @PUT("/donations/{id}/votes")
    Call<Donation> upVoteDonation(@Path("id") String id,
                                  @Body Donation donation);
}
```

3. Build Client using Retrofit

```
public class DonationApp extends Application
                         implements Callback<List<Donation>>
{
    public final int        target        = 10000;
    public int              totalDonated = 0;
    public List <Donation> donations     = new ArrayList<Donation>();
    //public DBManager dbManager;

    public DonationService donationService;
    public boolean          donationServiceAvailable = false;
    public String           service_url  = "http://donationweb-4-0.herokuapp.com/";
    //public String           service_url  = "http://10.0.2.2:4000";
                                           //Standard Emulator IP Address
```

3. Build Client using Retrofit

```
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(service_url)
        .addConverterFactory(GsonConverterFactory.create(gson))
        .client(okHttpClient)
        .build();

donationService = retrofit.create(DonationService.class);

Log.v( tag: "Donate",  msg: "Donation Service Created");

Call<List<Donation>> call1 = (Call<List<Donation>>) donationService.getAllDonations();
call1.enqueue( callback: this);
```

3. Build Client using Retrofit

```
@Override
public void onResponse(Call<List<Donation>> call, Response<List<Donation>> response) {
    serviceAvailableMessage();
    Log.v( tag: "retrofit", msg: "JSON = " + response.raw());
    donations = response.body();
    donationServiceAvailable = true;
}

@Override
public void onFailure(Call<List<Donation>> call, Throwable t) {
    donationServiceAvailable = false;
    Log.v( tag: "retrofit",t.getMessage());
    serviceUnavailableMessage();
}
```

3. Build Client using Retrofit *

```
public class Report extends Base implements OnItemClickListener,
                                             OnClickListener,
                                             Callback<List<Donation>> {

    ListView listView;
    DonationAdapter adapter;
    SwipeRefreshLayout mSwipeRefreshLayout;
    Call<List<Donation>> call1;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_report);

        mSwipeRefreshLayout = (SwipeRefreshLayout)
                              findViewById(R.id.report_swipe_refresh_layout);
        listView = (ListView) findViewById(R.id.reportList);
        listView.setOnItemClickListener(this);

        call1 = app.donationService.getAllDonations();
        call1.enqueue( callback: this);

        mSwipeRefreshLayout.setOnRefreshListener(() -> {
                call1 = app.donationService.getAllDonations();
                call1.enqueue( callback: Report.this);
        });

    }
```

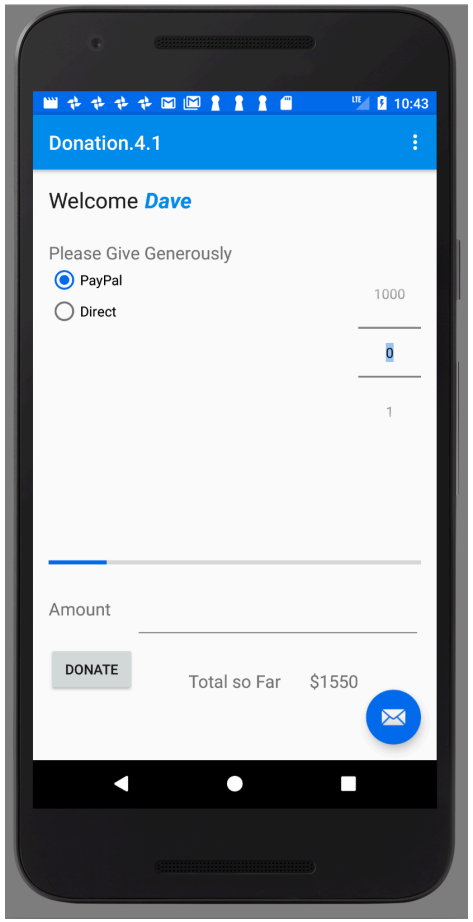
Call declared here for reuse

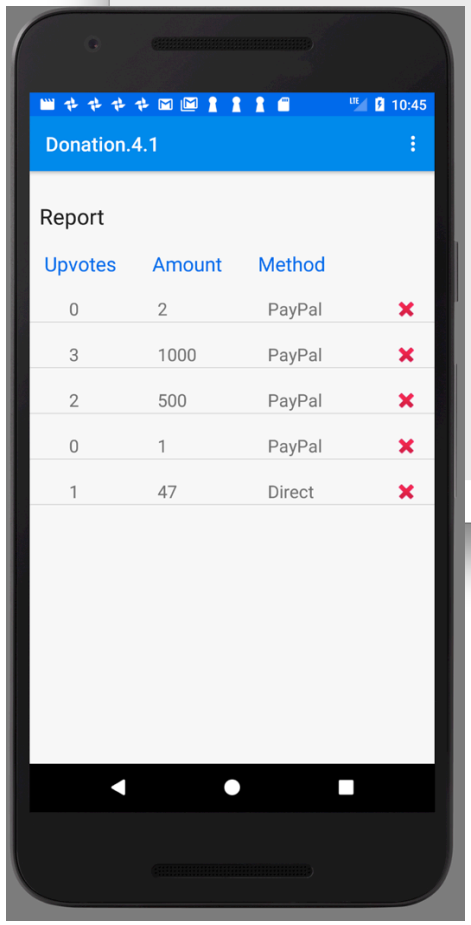
3. Build Client using Retrofit *

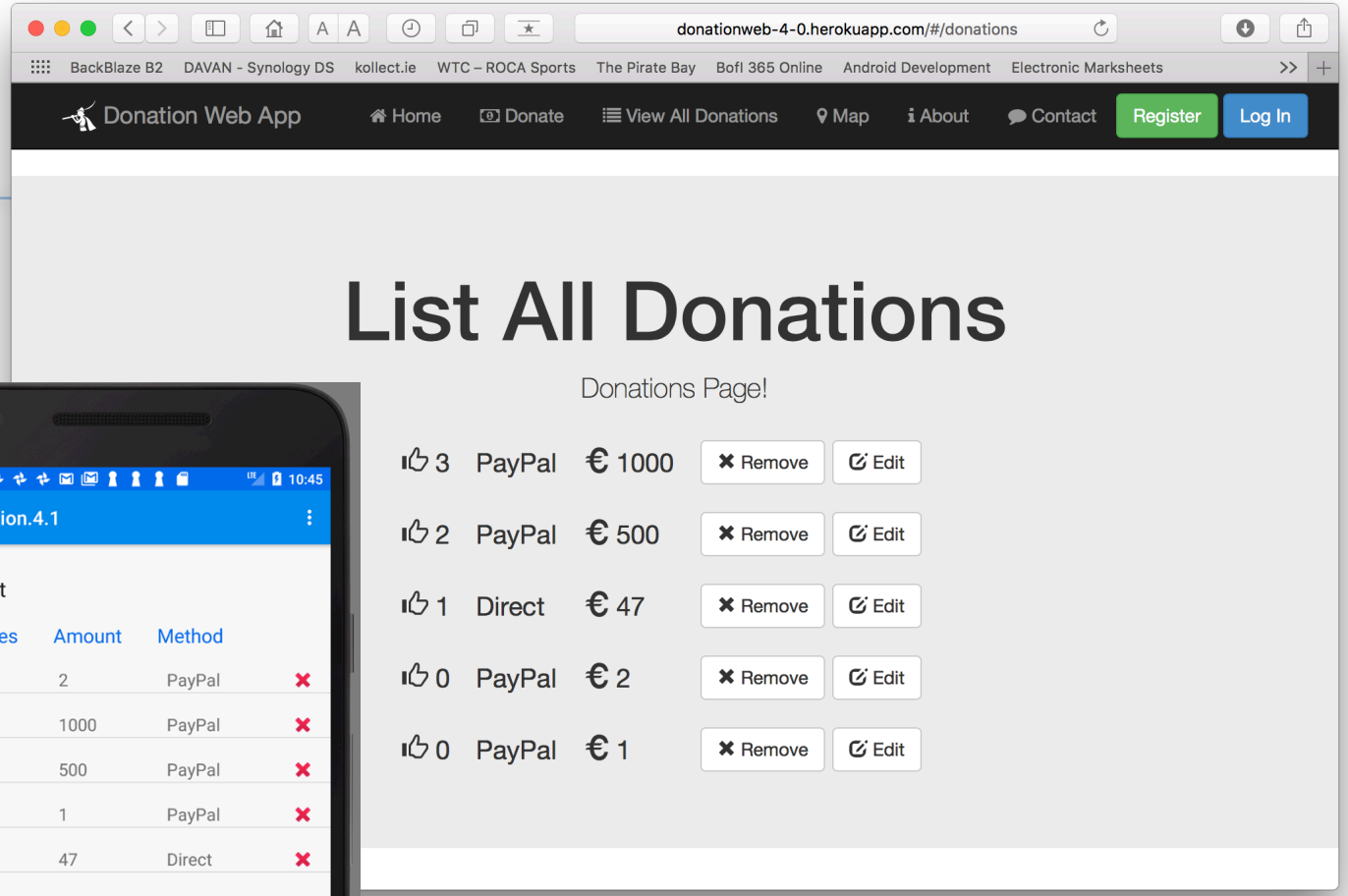
Anonymous Callback
allows for multiple calls
in same class

```
@Override
public void onItemClick(AdapterView<?> arg0, View row, int pos, long id) {

    String _id = row.getTag().toString();
    Call<List<Donation>> call2 = app.donationService.getDonation(_id);
    call2.enqueue(new Callback<List<Donation>>() {
                    @Override
                    public void onResponse(Call<List<Donation>> call,
                                           Response<List<Donation>> response) {
                        Toast toast = Toast.makeText( context: Report.this,  text: "Donation : " +
                                response.body() + " Selected.", Toast.LENGTH_LONG);
                        toast.show();
                    }

                    @Override
                    public void onFailure(Call<List<Donation>> call, Throwable t) {
                        Toast toast = Toast.makeText( context: Report.this,
                                text: "Error retrieving Donation", Toast.LENGTH_LONG);
                        toast.show();
                    }
                }
    );
}
```

Donation Web + Mobile







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


Thanks!