

CPOG Week 3 Video Walkthrough Notes

CodePath Observer Group

Consumer Key (API Key): [REDACTED]

Consumer Secret (API Secret): [REDACTED]

Base URL: <https://api.twitter.com/1.1/>

Endpoints:

- Get the home timeline for the user.
GET https://api.twitter.com/1.1/statuses/home_timeline.json
count=25
since_id=1 //all tweets
- Create a new tweet (compose)

Result from Get home timeline for user:

```
[
  {
    "coordinates": null,
    "truncated": false,
    "created_at": "Tue Aug 28 21:16:23 +0000 2012",
    "favorited": false,
    "id_str": "240558470661799936",
    "in_reply_to_user_id_str": null,
    "entities": {
      "urls": [

    ],
      "hashtags": [

    ],
      "user_mentions": [

    ]
    },
    "text": "just another test",
    "contributors": null,
    "id": 240558470661799936,
```

```
"retweet_count": 0,
"in_reply_to_status_id_str": null,
"geo": null,
"retweeted": false,
"in_reply_to_user_id": null,
"place": null,
"source": "<a href='\"//realitytechnicians.com/\"' rel='\"nofollow\"'>OAuth Dancer Reborn</a>",
"user": {
  "name": "OAuth Dancer",
  "profile_sidebar_fill_color": "DDEEF6",
  "profile_background_tile": true,
  "profile_sidebar_border_color": "C0DEED",
  "profile_image_url": "http://a0.twimg.com/profile_images/730275945/oauth-dancer_normal.jpg",
  "created_at": "Wed Mar 03 19:37:35 +0000 2010",
  "location": "San Francisco, CA",
  "follow_request_sent": false,
  "id_str": "119476949",
  "is_translator": false,
  "profile_link_color": "0084B4",
  "entities": {
    "url": {
      "urls": [
        {
          "expanded_url": null,
          "url": "http://bit.ly/oauth-dancer",
          "indices": [
            0,
            26
          ],
          "display_url": null
        }
      ]
    }
  },
}
```

```
    "description": null
  },
  "default_profile": false,
  "url": "http://bit.ly/oauth-dancer",
  "contributors_enabled": false,
  "favourites_count": 7,
  "utc_offset": null,
  "profile_image_url_https": "https://si0.twimg.com/profile_images/730275945/oauth-dancer_normal.jpg",
  "id": 119476949,
  "listed_count": 1,
  "profile_use_background_image": true,
  "profile_text_color": "333333",
  "followers_count": 28,
  "lang": "en",
  "protected": false,
  "geo_enabled": true,
  "notifications": false,
  "description": "",
  "profile_background_color": "C0DEED",
  "verified": false,
  "time_zone": null,

  "profile_background_image_url_https": "https://si0.twimg.com/profile_background_images/80151733/oauth-dance-dance.png",
  "statuses_count": 166,

  "profile_background_image_url": "http://a0.twimg.com/profile_background_images/80151733/oauth-dance.png",
  "default_profile_image": false,
  "friends_count": 14,
  "following": false,
  "show_all_inline_media": false,
  "screen_name": "oauth_dancer"
```

```
    },
    "in_reply_to_screen_name": null,
    "in_reply_to_status_id": null
  },
  {
    "coordinates": {
      "coordinates": [
        -122.25831,
        37.871609
      ],
      "type": "Point"
    },
    "truncated": false,
    "created_at": "Tue Aug 28 21:08:15 +0000 2012",
    "favorited": false,
    "id_str": "240556426106372096",
    "in_reply_to_user_id_str": null,
    "entities": {
      "urls": [
        {
          "expanded_url": "http://blogs.ischool.berkeley.edu/i290-abdt-s12/",
          "url": "http://t.co/bfj7zkDJ",
          "indices": [
            79,
            99
          ],
          "display_url": "blogs.ischool.berkeley.edu/i290-abdt-s12/"
        }
      ],
      "hashtags": [
      ],
      "user_mentions": [
```

```
{
  "name": "Cal",
  "id_str": "17445752",
  "id": 17445752,
  "indices": [
    60,
    64
  ],
  "screen_name": "Cal"
},
{
  "name": "Othman Laraki",
  "id_str": "20495814",
  "id": 20495814,
  "indices": [
    70,
    77
  ],
  "screen_name": "othman"
}
]
},
"text": "lecturing at the \"analyzing big data with twitter\" class at @cal with @othman http://t.co/bfj7zkDJ",
"contributors": null,
"id": 240556426106372096,
"retweet_count": 3,
"in_reply_to_status_id_str": null,
"geo": {
  "coordinates": [
    37.871609,
    -122.25831
  ],
  "type": "Point"
}
```

```
    },
    "retweeted": false,
    "possibly_sensitive": false,
    "in_reply_to_user_id": null,
    "place": {
      "name": "Berkeley",
      "country_code": "US",
      "country": "United States",
      "attributes": {
      },
      "url": "http://api.twitter.com/1/geo/id/5ef5b7f391e30aff.json",
      "id": "5ef5b7f391e30aff",
      "bounding_box": {
        "coordinates": [
          [
            [
              -122.367781,
              37.835727
            ],
            [
              -122.234185,
              37.835727
            ],
            [
              -122.234185,
              37.905824
            ],
            [
              -122.367781,
              37.905824
            ]
          ]
        ],
      }
    },
  ],
}
```

```
    "type": "Polygon"
  },
  "full_name": "Berkeley, CA",
  "place_type": "city"
},
"source": "<a href='\"//www.apple.com/\"' rel='\"nofollow\"'>Safari on iOS</a>",
"user": {
  "name": "Raffi Krikorian",
  "profile_sidebar_fill_color": "DDEEF6",
  "profile_background_tile": false,
  "profile_sidebar_border_color": "C0DEED",
  "profile_image_url": "http://a0.twimg.com/profile_images/1270234259/raffi-headshot-casual_normal.png",
  "created_at": "Sun Aug 19 14:24:06 +0000 2007",
  "location": "San Francisco, California",
  "follow_request_sent": false,
  "id_str": "8285392",
  "is_translator": false,
  "profile_link_color": "0084B4",
  "entities": {
    "url": {
      "urls": [
        {
          "expanded_url": "http://about.me/raffi.krikorian",
          "url": "http://t.co/eNmnM6q",
          "indices": [
            0,
            19
          ],
          "display_url": "about.me/raffi.krikorian"
        }
      ]
    }
  },
  "description": {
```

```
"urls": [  
  
]  
}  
},  
"default_profile": true,  
"url": "http://t.co/eNmnM6q",  
"contributors_enabled": false,  
"favourites_count": 724,  
"utc_offset": -28800,  
  
"profile_image_url_https": "https://si0.twimg.com/profile_images/1270234259/raffi-headshot-casual_normal.png",  
"id": 8285392,  
"listed_count": 619,  
"profile_use_background_image": true,  
"profile_text_color": "333333",  
"followers_count": 18752,  
"lang": "en",  
"protected": false,  
"geo_enabled": true,  
"notifications": false,  
"description": "Director of @twittereng's Platform Services. I break things.",  
"profile_background_color": "C0DEED",  
"verified": false,  
"time_zone": "Pacific Time (US & Canada)",  
"profile_background_image_url_https": "https://si0.twimg.com/images/themes/theme1/bg.png",  
"statuses_count": 5007,  
"profile_background_image_url": "http://a0.twimg.com/images/themes/theme1/bg.png",  
"default_profile_image": false,  
"friends_count": 701,  
"following": true,  
"show_all_inline_media": true,
```



```
    "screen_name": "raffi"
  },
  "in_reply_to_screen_name": null,
  "in_reply_to_status_id": null
},
{
  "coordinates": null,
  "truncated": false,
  "created_at": "Tue Aug 28 19:59:34 +0000 2012",
  "favorited": false,
  "id_str": "240539141056638977",
  "in_reply_to_user_id_str": null,
  "entities": {
    "urls": [

    ],
    "hashtags": [

    ],
    "user_mentions": [

    ]
  },
  "text": "You'd be right more often if you thought you were wrong.",
  "contributors": null,
  "id": 240539141056638977,
  "retweet_count": 1,
  "in_reply_to_status_id_str": null,
  "geo": null,
  "retweeted": false,
  "in_reply_to_user_id": null,
  "place": null,
  "source": "web",
```

```
"user": {
  "name": "Taylor Singletary",
  "profile_sidebar_fill_color": "FBFBFB",
  "profile_background_tile": true,
  "profile_sidebar_border_color": "000000",

  "profile_image_url": "http://a0.twimg.com/profile_images/2546730059/f6a8zq58mg1hn0ha8vie_normal.jpeg",
  "created_at": "Wed Mar 07 22:23:19 +0000 2007",
  "location": "San Francisco, CA",
  "follow_request_sent": false,
  "id_str": "819797",
  "is_translator": false,
  "profile_link_color": "c71818",
  "entities": {
    "url": {
      "urls": [
        {
          "expanded_url": "http://www.rebelmouse.com/episod/",
          "url": "http://t.co/Lxw7upbN",
          "indices": [
            0,
            20
          ],
          "display_url": "rebelmouse.com/episod/"
        }
      ]
    },
    "description": {
      "urls": [
        ]
      }
    }
  },
}
```

```
"default_profile": false,
"url": "http://t.co/Lxw7upbN",
"contributors_enabled": false,
"favourites_count": 15990,
"utc_offset": -28800,

"profile_image_url_https": "https://si0.twimg.com/profile_images/2546730059/f6a8zq58mg1hn0ha8vie_norm
al.jpeg",
"id": 819797,
"listed_count": 340,
"profile_use_background_image": true,
"profile_text_color": "D20909",
"followers_count": 7126,
"lang": "en",
"protected": false,
"geo_enabled": true,
"notifications": false,
"description": "Reality Technician, Twitter API team, synthesizer enthusiast; a most excellent adventure
in timelines. I know it's hard to believe in something you can't see.",
"profile_background_color": "000000",
"verified": false,
"time_zone": "Pacific Time (US & Canada)",

"profile_background_image_url_https": "https://si0.twimg.com/profile_background_images/643655842/hzfv12
wini4q60zzrthg.png",
"statuses_count": 18076,

"profile_background_image_url": "http://a0.twimg.com/profile_background_images/643655842/hzfv12wini4q6
0zzrthg.png",
"default_profile_image": false,
"friends_count": 5444,
"following": true,
"show_all_inline_media": true,
```

```
    "screen_name": "episod"
  },
  "in_reply_to_screen_name": null,
  "in_reply_to_status_id": null
}
]
```

00:10:14 android-rest-client-template - oauth tool from github.

github.com/codepath/android-rest-client-template

download this project's zip file, expand, rename it to MySimpleTweet

import the new MySimpleTweet into Android Studio.

launch Android Studio, click on 'Open an existing Android Studio project'

choose the new folder.

Important!!!

remove the '/Gradle' from the Gradle Project field.

select the 'Use default gradle wrapper'

click OK.

project loads, then click 'sync with Gradle'

RestClient.java - object responsible for communicating with the rest api.

Each method here is an endpoint.

00:14:40

RestApplication.java - object that is 'booted up' when the application is run.

When the application is run, it will look for the name of the class:

<application

android:name=".RestApplication"

LoginActivity.java - where the user will sign in.

00:14:55 RestClientApp (RestApplication in our latest download).

Extends AppCompatActivity - give persistence functionality so we'll be able to store things into SQLite.

getRestClient - returns RestClient that our activities will use to access data from the api.

00:15:15 LoginActivity - where the user will sign in.

In the xml - user will use login button.

Login button - taken to twitter, authenticate, if things work well, use will then redirected back to application which will then use the login credentials to make requests. This is the OAuth authentication flow.

00:16:16 Models - SampleModel

Model for each of the things, i.e. tweet model, user model, ...

Loaded with ActiveAndroid - for persistent models.

Persistent based Object Relational Mapping

00:17:10 Libraries Used:

scribe.java

Android Async HTTP

codepath-oauth

Picasso

ActiveAndroid

Reference class notes.

Now to change the names to make this our application.

AndroidManifest.xml

```
minSdkVersion = 16
```

```
targetSdkVersion = 21
```

Rename com.codepath.apps.restclienttemplate by right clicking on package and selecting refactor-rename to 'mysimpletweets'.

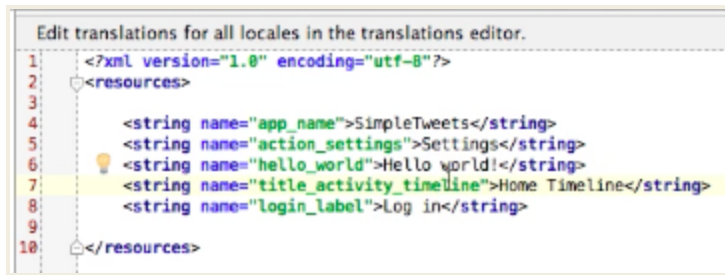
Notice the change in the package name as well as the 'package' field in the manifest.

Change the title of the app:

```
values/strings.xml => change app_name tag to 'SimpleTweets'
```

```
Also change title_activity_photos tag to 'Home TimeLine'
```

```
Change the 'title_activity_photos' to 'title_activity_timeline'
```



00:19:29 Boot up Emulator and RUN

00:20:08 Now need to configure for our api.

The RestClient is our interface to the backend.

Refactor/rename RestClient to TwitterClient.

Refactor/rename RestClientApp to TwitterApplication

00:21:34 - Replace the constant values in **TwitterClient.java** with ones that we need for our api.

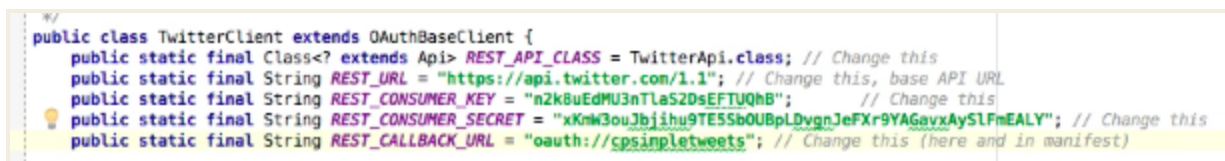
REST_API_CLASS = TwitterApi

REST_CONSUMER_KEY= "OUR KEY"

REST_CONSUMER_SECRET="OUR SECRET"

REST_URL="OUR BASE URL"="<https://api.twitter.com/1.1>"

REST_CALLBACK_URL="oauth://cpsimpletweets" - need to change in Manifest



00:22:58 Relaunch Emulator

The app runs but fails when trying to authenticate because Twitter tries to go to our callback url with the access token which does not exist and thus not properly handled.

We go to the Manifest file and modify the intent-filter to say we can handle the above web request.

change to: android:host="cpsimpletweets"

00:25:45 Run again.

If we get back to our Login screen after we authenticate, then the app is running and a token was received.

If problems, make sure timestamp on the emulator matches the actual time.

Now to complete the authentication process:

res/layout/**activitylogin.xml** to change <Button android:text="@..."

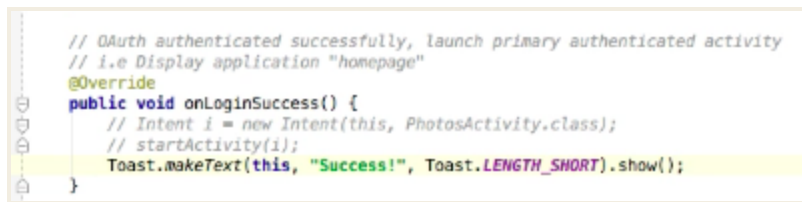
Use cmd-click on the text to get to the string resource => change "Login_label" string from 'Login' to 'Connect to Twitter'.

Student may want to add styling...

So after the authentication, the user is brought back to our **LoginActivity**.

We now need to code up our onLoginSuccess method.

Let's add a toast here: `Toast.makeText(this, "Success!", Toast.LENGTH_SHORT).show();`



```
// OAuth authenticated successfully, launch primary authenticated activity
// i.e Display application "homepage"
@Override
public void onLoginSuccess() {
    // Intent i = new Intent(this, PhotosActivity.class);
    // startActivity(i);
    Toast.makeText(this, "Success!", Toast.LENGTH_SHORT).show();
}
```

00:28:57 - Rerun

But we don't need to re-authenticate because of persistent data. So to test, we'll uninstall the app and rerun.

If successful, we want to add an authenticated activity to display the latest tweets.

Need to create the new timeline activity:

With LoginActivity selected, go to File/New/Activity/BlankActivity.

Change Activity Name to: **TimelineActivity**

Layout Name: activity_timeline

Title: Home Timeline

Menu Resource Name: menu_timeline

Click finish.

In the **activity_timeline.xml**

Delete the "HelloWorld"

Add ListView, align left top, right bottom, align to parent on all 4 corners.

Name the ListView => id: lvTweets

Got to Text tab and remove padding after the layout_height.

00:32:50 Go to back to the **LoginActivity.java**

Uncomment the two lines regarding new Intent and startActivity.

Change the new Intent to launch the TimelineActivity.class.

```
// OAuth authenticated successfully, launch primary authenticated activity
// i.e Display application "homepage"
@Override
public void onLoginSuccess() {
    Intent i = new Intent(this, TimelineActivity.class);
    startActivity(i);
}
```

33:24 Rerun to see the blank timeline.

To show timeline, we need to make a call to our api.

00:34:30 To do this we need to go to **TwitterClient.java**.

To reach a particular end point we need to create a method to represent it.

In this case we create getHomeTimeline()

These endpoint generally just need a callback which defines what happens when the data comes in.

00:35:13 Copy template parms for our method.

Define the url that we're going to hit: String apiUrl = getApiUrl("statuses/home_timeline.json");

Next, we specify the parameters:

RequestParams params = new RequestParams();

params.put("count", 25);

params.put("since_id", 1);

```
// METHOD == ENOPOINT

// HomeTimeline - Gets us the home timeline
// GET statuses/home_timeline.json
// count=25
// since_id=1
public void getHomeTimeline(AsyncHttpResponseHandler handler) {
    String apiUrl = getApiUrl("statuses/home_timeline.json");
    // Specify the params
    RequestParams params = new RequestParams();
    params.put("count", 25);
    params.put("since_id", 1);
    // Execute the request
    getClient().get(apiUrl, params, handler);
}

// COMPOSE TWEET
```

9

Then we need to execute the request:


```
getClient().get(apiUrl, params, handler); // our handler is our callback.
```

!!! doesn't seem to matter whether we use the above statement or the statement below !!!

```
client.get(apiUrl, params, handler);
```

We do the above procedure for each endpoints, including composing a tweet.

Now to test this, we will execute this request from our TimelineActivity.

00:38:00 We need to create access to our TwitterClient from our TimelineActivity.

In **TimelineActivity.java**

Create: private TwitterClient client;

In onCreate, we instantiate our client: client = TwitterApplication.getRestClient(); // this give us a singleton client, which means that we'll be using the same client across all our activities.

```
private TwitterClient client;

@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_timeline);
    client = TwitterApplication.getRestClient(); // singleton client
    populateTimeline();
}

private void populateTimeline() {
    |
}
```

Create: populateTimeline method;

To send api req to get timeline json and fill list view by creating the tweets object from json, including downloading and serializing the json data.

// [] = JSON Array

// {} = JSON Object

00:41:21 Need to define our populateTimeline method to get the client to call the getHomeTimeline().

Since we know we're getting back json data, we'll use the json callback function.

```
client.getHomeTimeline(new JsonHttpResponseHandler() {})
```

In the callback method we need to define Success and Failure.

00:42:08 onSuccess wants to use a JSONObject, but your response root is an array, so we use JSONArray instead for our response.

00:42:58 Onsuccess, place a Log.d("DEBUG", json.toString()); // where json is our array. print to logcat

Hit alt-enter while cursor is on top of the Log.d import debug log library.

OnFailure, place another Log.d("DEBUG", errorResponse.toString()); // print out the error response.

Add breakpoint at the two log statements and at the client.getHomeTimeline () statement inside the populateTimeline method.s

```
public class TimelineActivity extends ActionBarActivity {

    private TwitterClient client;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_timeline);
        client = TwitterApplication.getRestClient(); // singleton client
        populateTimeline();
    }

    // Send an API request to get the timeline json
    // Fill the listView by creating the tweet objects from the json
    private void populateTimeline() {
        client.getHomeTimeline(new JsonHttpResponseHandler() {
            // SUCCESS
            @Override
            public void onSuccess(int statusCode, Header[] headers, JSONArray json) {
                Log.d("DEBUG", json.toString());
            }

            // FAILURE
            @Override
            public void onFailure(int statusCode, Header[] headers, Throwable throwable, JSONObject errorResponse) {
                Log.d("DEBUG", errorResponse.toString());
            }
        });
    }
}
```

00:43:47 Hit the Debug button to start emulation

At breakpoint at onSuccess, hover over json to see the values and 25 objects.

We now need to deserialize each of these 25 json objects each of which is a tweet.

Then turn them into models which can then be used to display in our view.

00:45:29 Create the models to hold our json data

Right click on models folder, select new Java Class.

Create the tweet class => Name: Tweet

We want this model to parse the json and store the data, encapsulate state logic or display logic for this data.

Tweet class will represent all our attributes data.

List of attributes that we care about:

private String body;

private long uid;

private String CreatedAt;

```
private User user;
```

Recall the response that we get back:

An Array:

```
[
```

Inside we have a bunch of Java (JSON) objects

```
{
```

```
},
```

```
{
```

```
...
```

```
}
```

```
]
```

Deserialize the JSON and build tweet objects.

Turn it into a Java object w/ all the field filled in. // Tweet.fromJSON("{...}") => <Tweet>

```
public static Tweet fromJSON(JSONObject jsonObject) {
```

```
    // need to construct a blank tweet
```

```
    Tweet tweet = new Tweet();
```

```
    try {
```

```
        // Extract the values from the JSON, store them, return the tweet object.
```

```
        // store the body
```

```
        tweet.body = jsonObject.getString("text");
```

```
        // store the unique ID, etc.
```

```
        tweet.uid = jsonObject.getLong("id");
```

```
        tweet.createdAt = jsonObject.getString("created_at");
```

```
        // tweet.user = ... J(tbd)
```

```
    } catch (JSONException) {
```

```
        e.printStackTrace();
```

```
    }
```

```
    // we want to return that tweet but in a more usable form.
```

```
    return tweet;
```

```
}
```

Will have errors, but click on red underline and hit alt-enter or click on lightbulb, select surround with try catch. Move tweet attributes inside the try/catch to get above.

```
// Parse the JSON + Store the data, encapsulate state logic or display logic
public class Tweet {
    // List out the attributes
    private String body;
    private long uid; // unique id for the tweet
    // private User user;
    private String createdAt;

    // Deserialize the JSON and build Tweet objects
    // Tweet.fromJSON("{ ... }") => <Tweet>
    public static Tweet fromJSON(JSONObject jsonObject) {
        Tweet tweet = new Tweet();
        // Extract the values from the json, store them
        try {
            tweet.body = jsonObject.getString("text");
            tweet.uid = jsonObject.getLong("id");
            tweet.createdAt = jsonObject.getString("created_at");
            // tweet.user = ...
        } catch (JSONException e) {
            e.printStackTrace();
        }
        // Return the tweet object
        return tweet;
    }
}
```

00:51:08 - Generate Getters for the attributes via Code/Generate/Getter and selecting the three attribute variables in our Tweet model.

00:51:43 - We're almost done with the Tweet model. We want to store user as a separate object that we can link to.

Go to model folder, right click, select New/Java Class, Name: [User](#)

In this User.java we want to define it to match the JSON response.

Again, we'll list the attribute:

private String name;

private long uid;

private String screenName;

private String profileImageUrl;

Deserialize the user json => to a User object

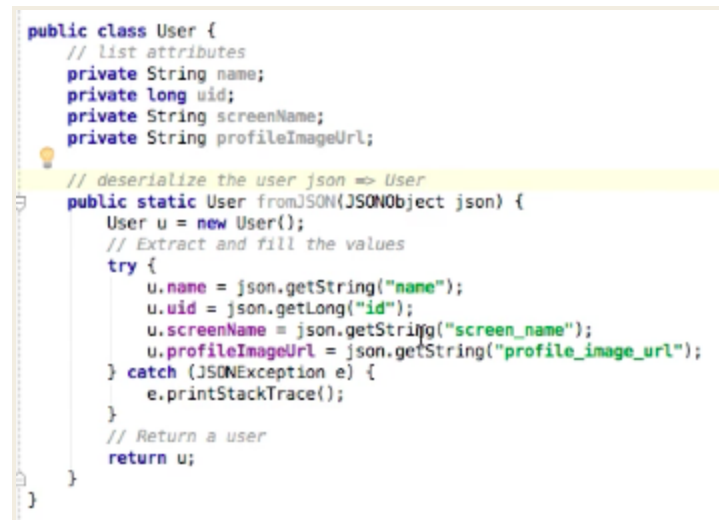
```
public static User fromJSON(JSONObject json) {
    User u = new User();
    // Extract and fill the values from the json
    try {
        u.name = json.getString("name");
```

```

        u.uid = json.getLong("id");
        u.screenName = json.getString("screen_name");
        u.profileImageUrl = json.getString("profile_image_url");
    } catch (JSONException e) {
        e.printStackTrace();
    }

    // Return a user object
    return u;
}

```



```

public class User {
    // List attributes
    private String name;
    private long uid;
    private String screenName;
    private String profileImageUrl;

    // deserialize the user json => User
    public static User fromJSON(JSONObject json) {
        User u = new User();
        // Extract and fill the values
        try {
            u.name = json.getString("name");
            u.uid = json.getLong("id");
            u.screenName = json.getString("screen_name");
            u.profileImageUrl = json.getString("profile_image_url");
        } catch (JSONException e) {
            e.printStackTrace();
        }

        // Return a user
        return u;
    }
}

```

Select the first red underlined text, hit alt-enter, select Surround with try/catch, and move all the attributes inside the try/catch.

Generate getters here as well. From the menu, select Code/Generate/Getter, select all 4 attributes from our User model and click OK.

```

// list attributes
private String name;
private long uid;
private String screenName;
private String profileImageUrl;

public String getName() {
    return name;
}

public long getUid() {
    return uid;
}

public String getScreenName() {
    return screenName;
}

public String getProfileImageUrl() {
    return profileImageUrl;
}

// deserialize the user json - Here

```

00:55:08 - Now that we have our User, we'll go back to our [Tweet](#) model and now make correct references to our User model.

Make sure we have a private variable user to store embedded user object.

in our try/catch:

```
tweet.user = User.fromJSON(jsonObject.getJSONObject("user")); // note that we use the key "user"
```

to get our User object. ??? Do we need to call User.fromJSON()??? Can't we just use
 jsonObject.getJSONObject("user")??? ⇒ We need to specify the User class so it knows to return the User
 object back to us here.

Add a getter for user:User via Code/Generate/Getter ...

We now have models, data coming in from the json, and the list view. We now need to 'connect the dots'.

00:56:46 - go to [TimelineActivity.java](#)

In onSuccess()

We have the json here

Need to deserialize the json data

Create Models (pull the data into the model object)

Load the model data into the listview.

to load the model into the listview, we need to build an adapter.

00:57:30 - Right click on the com....mysimpletweets under the java folder and above the models folder,
 select New/Java Class/, Name: [TweetsArrayAdapter](#)

This object will be responsible for:

```
// Taking the Tweet objects and turning them into Views that will be displayed in the list
// To do this we need to extend from some adapter datasource. In this case the datasource is going to be an
// ArrayList{}
public class TweetsArrayAdapter extends ArrayAdapter<Tweet> {

}
```

Need to create a constructor, select the light bulb or the class name and hit alt-enter and select Create constructor matching super. Change the params in this constructor to reflect the proper resource (data source) being sent.

```
    public TweetsArrayAdapter(Context context, List<Tweet> tweets) {
        super (context, android.R.layout.simple_list_item_1, tweets); // need to add the middle
// later we'll want to override and setup our own custom template
    }
```

```
package com.codepath.apps.mysimpletweets;

import android.content.Context;
import android.widget.ArrayAdapter;

import com.codepath.apps.mysimpletweets.models.Tweet;
import java.util.List;

// Taking the Tweet objects and turning them into Views displayed in the list
public class TweetsArrayAdapter extends ArrayAdapter<Tweet> {
    public TweetsArrayAdapter(Context context, List<Tweet> tweets) {
        super(context, android.R.layout.simple_list_item_1, tweets);
    }

    // Override and setup custom template
}
```

00:59:53 - Going back to our [TimelineActivity.java](#)

We'll need to instantiate our adapter that we defined above.

```
private TwitterClient client;

private ArrayList<Tweet> tweets; // need the array list that 'powers' the adapter of type Tweet
private TweetsArrayAdapter aTweets;

private ListView lvTweets; // also need to define our list view that we will use.
```

```

public class TimelineActivity extends ActionBarActivity {

    private TwitterClient client;
    private ArrayList<Tweet> tweets;
    private TweetsArrayAdapter aTweets;
    private ListView lvTweets;

    @Override
    protected void onCreate(Bundle savedInstanceState) {

```

01:00:39 - Modify onCreate

// Find the list view

```
lvTweets = (ListView) findViewById(R.id.lvTweets);
```

// Create the arraylist (our data source)

```
tweets = new ArrayList<>();
```

// Construct the adapter from the data source

```
aTweets = new TweetsArrayAdapter(this, tweets);
```

// Connect the adapter to the listview

```
lvTweets.setAdapter(aTweets);
```

// Get the client and populate the timeline

```
client = TwitterApplication.getRestClient(); // singleton client
```

```
populateTimeline();
```

```

public class TimelineActivity extends ActionBarActivity {

    private TwitterClient client;
    private ArrayList<Tweet> tweets;
    private TweetsArrayAdapter aTweets;
    private ListView lvTweets;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_timeline);
        // Find the listview
        lvTweets = (ListView) findViewById(R.id.lvTweets);
        // Create the arraylist (data source)
        tweets = new ArrayList<>();
        // Construct the adapter from data source
        aTweets = new TweetsArrayAdapter(this, tweets);
        // Connect adapter to list view
        lvTweets.setAdapter(aTweets);
        // Get the client
        client = TwitterApplication.getRestClient(); // singleton client
        populateTimeline();
    }

    // Send an API request to get the timeline json
    // Fill the listview by creating the tweet objects from the json
    private void populateTimeline() {
        client.getHomeTimeline(new RequestParameters() {

```

01:02:16 - Getting the tweets into the adapter - Modify [populateTimeline\(\)](#)

In getHomeTimeline/onSuccess

// DESERIALIZE JSON

// CREATE MODELS AND ADD THEM TO THE ADAPTER

// LOAD THE MODEL DATA INTO LISTVIEW

```
ArrayList<Tweet> tweets = Tweet.fromJSONArray(json);
```

// we want to ask the Tweet model to iterate through the json array and return a list of tweets.

aTweets.addAll(Tweet.fromJSONArray(json)); // pass the json array to our new method which will take our array of tweets - this method will return all the items that we will submit to the listview, returning a bunch of tweets. This method will exist in the Tweet model

```
// Send an API request to get the timeline json
// Fill the listview by creating the tweet objects from the json
private void populateTimeline() {
    client.getHomeTimeline(new JsonHttpResponseHandler() {
        // SUCCESS
        @Override
        public void onSuccess(int statusCode, Header[] headers, JSONArray json) {
            Log.d("DEBUG", json.toString());
            // DESERIALIZE JSON
            // CREATE MODELS AND ADD THEM TO THE ADAPTER
            // LOAD THE MODEL DATA INTO LISTVIEW
            aTweets.addAll(Tweet.fromJSONArray(json));
        }

        // FAILURE
        @Override
        public void onFailure(int statusCode, Header[] headers, Throwable throwable, JS
```

01:04:16 - Create that new method, Tweet.fromJSONArray(json), to our Tweet model. - [Tweet.java](#)

// The goal is to pass a list of items and output a list of tweets

// Tweet.fromJSONArray([{...}, {...}...]) => List<Tweet>

```
public static ArrayList<Tweet> fromJSONArray(JSONArray jsonArray){
```

```
    // Start with a new array list
```

```
    ArrayList<Tweet> tweets = new ArrayList<>();
```

```
    // Iterate the json array and create tweets
```

```
    for (int i=0; i<jsonArray.length(); i++) {
```

```
        // extract out the tweet json for the particular item in the array
```

```
        try {
```

```
            JSONObject tweetJson = jsonArray.getJSONObject(i); // Notice that we have an error here
```

because we need try/catch around this.

```
            Tweet tweet = Tweet.fromJSON(tweetJson); // use this function that was created before.
```

```
            if (tweet != null) {
```

```
                // if successful, add tweets to list of tweets.
```

```
                tweets.add(tweet);
```

```
            }
```

```

    } catch (JSONException e) {
        e.printStackTrace();
        continue; // even if fail, go ahead and process the other tweets.
    }

}

// Return the finished list
return tweets;
}

```

In Tweet.java:



```

// Tweet.fromJSONArray([ { ... }, { ... } ] => List<Tweet>
public static ArrayList<Tweet> fromJSONArray(JSONArray jsonArray) {
    ArrayList<Tweet> tweets = new ArrayList<>();
    // Iterate the json array and create tweets
    for (int i = 0; i < jsonArray.length(); i++) {
        try {
            JSONObject tweetJson = jsonArray.getJSONObject(i);
            Tweet tweet = Tweet.fromJSON(tweetJson);
            if (tweet != null) {
                tweets.add(tweet);
            }
        } catch (JSONException e) {
            e.printStackTrace();
            continue;
        }
    }
    // Return the finished list
    return tweets;
}

```

01:07:27 - Now we can go back to our TimelineActivity and populate the tweets into the adapter and then to the views!

====> 01:07:45 - Run and see the list of tweets!

To verify data is what we expect:

Breakpoint at TimelineActivity.populateTimeline.onSuccess - aTweets.add All(Tweet.fromJSONArray(json));

Stop at breakpoint.

In our Variables pane, expand aTweets.

Copy Tweet.fromJSONArray(json), right click on it and select Add to watches

Let's add a debug statement after the above statement:

```
Log.d("DEBUG", aTweets.toString());
```

Breakpoint on this line instead.

Debug (click on the run/debug button)

Expand aTweets/mObjects to see the 25 tweets in the Variable pane.

Expand each tweet to see its body, createdAt, and user.

Now that we know that it's working properly, let's customize the list item to get a better display of the tweets.

Then we'll go into the adapter and inflate our own tweet templates.

Start by creating that template.

01:10:43 - Go to layout folder, right click and select New/Layout Resource File, name: item_tweet.xml, Root element: RelativeLayout, click OK.

01:11:38 - Drag ImageView to the top left, layout:width: 50dp, layout:height: 50dp,

For background in the properties pane: FF5F8BE6

Move a TextView to the right of the ImageView. Align parent top.

Move another TextView below the above TextView and the right of the ImageView.

Set the ImageView id: ivProfileImage

Set the top first TextView text: Billy - could be any name.

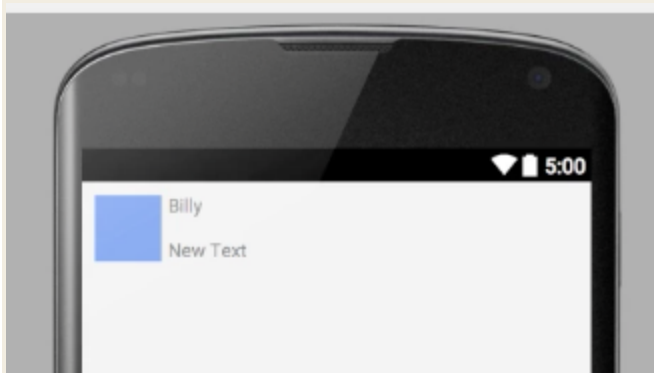
Set the top first TextView id: tvUserName

Set the bottom TextView id: tvBody

Move the bottom TextView so it's to the right of the ImageView and align bottom to bottom of the ImageView.

Add padding All (for ImageView): 10dp

Set ImageView layout:margin:right to 5dp.



This the layout for each of our tweet item.

01:14:03 - Go to the [TweetsArrayAdapter.java](#) to override the getView method.

```
// Override and setup custom template
// ViewHolder pattern
@Override
public View getView(int position, View convertView, ViewGroup parent) {
    // 1. Get the tweet
    Tweet tweet = getItem(position);
    // 2. Find or inflate the template
    if (convertView == null) {
        convertView = LayoutInflater.from(getContext()).inflate(R.layout.item_tweet, parent, false);
    }
    // 3. Find the subviews to fill with data in the template
    ImageView ivProfileImage = (ImageView) convertView.findViewById(R.id.ivProfileImage);
    TextView tvUserName = (TextView) convertView.findViewById(R.id.tvUserName);
    TextView tvBody = (TextView) convertView.findViewById(R.id.tvBody);
    // 4. Populate data into the subviews
    tvUserName.setText(tweet.getUser().getScreenName());
    tvBody.setText(tweet.getBody());
    ivProfileImage.setImageResource(android.R.color.transparent); // clear out the old image for a recycled view
    Picasso.with(getContext()).load(tweet.getUser().getProfileImageUrl()).into(ivProfileImage);
    // 5. Return the view to be inserted into the list
    return convertView;
}
```

Step 2:

If convertView is null, that means the template is not being recycled and we'll need to inflate it.

We'll pass it the parent container so it'll know how to resize itself and false to not to insert it into the parent yet.

Step 3:

Need to find the subview within the convertView. i.e. the ImageView.

Need to find the two textview as well.

Step 4:

Populate the data into the subviews.

For the image we need to do something different: need to clear the content in case it has left over data (???? why does this matter??? Wouldn't it write over what was there???)

We'll use Picasso to fetch the image from the url (network) and to retrieve the image and insert it into the view.

Step 5:

We then return the convertView which will then be inserted into the list.

01:18:40 - For homework and for the 'real' version of this app, we should review the ViewHolder Pattern.

Should apply this ViewHolder pattern to every array adapter to optimize performance.

Using an ArrayAdapter with ListView | CodePath Android Cliffnotes

[C guides.codepath.com/android/Using-an-ArrayAdapter-with-ListView#improving-performance-with-the-viewholder-pattern](https://guides.codepath.com/android/Using-an-ArrayAdapter-with-ListView#improving-performance-with-the-viewholder-pattern)
codepath/android-rest-client-template
Home | CodePath Android Cliffnotes
Using an ArrayAdapter with ListView | Code...

Improving Performance with the ViewHolder Pattern

To improve performance, we should modify the custom adapter by applying the **ViewHolder** pattern which speeds up the population of the ListView considerably by caching view lookups for smoother, faster item loading:

01:19:17 - Now that we have our own custom getView, we can set the 2nd parameter of our adapter to 0.

```
public TweetsArrayAdapter(Context context, List<Tweet> tweets) {  
    super(context, 0, tweets);  
}
```

01:19:33 - Now we run our app!

01:20:24 - Review/Summary of how app works.

Download the rest client template.

Define the rest client, in this case for Twitter.

Define the end points.

Look at the responses and construct resources/models based on the responses.

Deserialize the json object after the request.

Load the json into the models.

Then use the models for the UI. Taking the data into the views.

Need to look into pagination in twitter.

Need to look at a few more endpoints.

