Networking, Step by Step

Follow these basic steps in order to download images, files, XML, or JSON using **HttpURLConnection**.

1. Declare a **URL Connection**
2. Open **InputStream** to connection
3. **Download** and **decode** data (Bitmap, JSON)
4. Wrap in **AsyncTask**

Open InputStream

Use **HttpURLConnection** to connect to the server and establish and input stream.

```
URL url = new URL("http://www.google.com");
URLConnection conn = url.openConnection();
conn.connect();
InputStream in = conn.getInputStream();
```

Now you use **InputStream** as a way to get access to the response from the HTTP request.

Downloading Images

If the URL is an image, then process the stream using a Bitmap decoder and set the Bitmap of an ImageView.

```
// Setup InputStream from previous slide
InputStream in = conn.getInputStream();
// Convert stream to Bitmap
Bitmap bitmap = BitmapFactory.decodeStream(in);
in.close();
// Load Bitmap into ImageView
ImageView img = (ImageView) findViewById(R.id.img);
img.setImageBitmap(bitmap);
```

Downloading JSON

If the response is JSON, use a **BufferedReader** and **JSONTokenizer** to decode the JSON.

```
InputStream in = ...; // From previous slide

StringBuilder stringBuilder = new StringBuilder();
BufferedReader reader =
    new BufferedReader(new InputStreamReader(in));
String line;
while ((line = reader.readLine()) != null) {
    stringBuilder.append(line);
}

JSONObject jsonDict =
    new JSONTokenizer(stringBuilder.toString()).nextValue();
```

Internet Permissions

In order to execute network requests, you must **request the appropriate permission** in the AndroidManifest.xml

```
<?xml version=”1.0” encoding=”utf-8”?>
<manifest xmlns:android=”http://schemas.android.com/apk/res/android”
package=”net.learn2develop.Networking” android:versionCode=”1” android:
versionName=”1.0”>

<uses-sdk android:minSdkVersion=”14” />

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

UIThread

UIThread is the **primary thread** of the application which manages the user interface.

- All UI updates (modifying a text label, laying out a view) **MUST** be done on the UIThread.
- Long running operations such as networking **MUST NOT** happen on the UIThread.
- If long running operations happen on the UIThread, this **blocks** the application from responding and the app appears to be "frozen".

StrictMode

Android **doesn't permit** network access in the main thread by **default**, so that the UI remains responsive. For testing, we can **temporarily allow** networking in the main thread.

```
StrictMode.setThreadPolicy(
    new StrictMode.ThreadPolicy.Builder().
        permitNetwork().build()
);
```

AsyncTask

Use AsyncTasks to execute long running operations in a background thread.

```
private class MyAsyncTask extends AsyncTask<String, Void, Bitmap> {
     public Bitmap doInBackground(String... strings) {
         // Some long-running task like downloading an image.
     }
     protected void onPostExecute(Bitmap result) {
         // This method is executed in the UIThread
         // with access to the result of the task
     }
}

private void someMethod() {
    new MyAsyncTask().execute("http://images.com/image.jpg");
}
```

AsyncTask Generic Types

AsyncTask accept three generic types to inform the background work being done.

- AsyncTask<**Params**, **Progress**, **Result**>
  - **Params** - the type that is passed into the *execute()* method.
  - **Progress** - the type that is used within the task to track progress.
  - **Result** - the type that is returned by *doInBackground()*.

AsyncTask Methods

AsyncTask has multiple events that can be overridden.

- **`onPreExecute`** - executed in the main thread to do things like create the initial progress bar view.
- **`doInBackground`** - executed in the background thread to do things like network downloads.
- **`onProgressUpdate`** - executed in the main thread after the publishProgress method is called from doInBackground.
- **`onPostExecute`** - executed in the main thread to do things like set image views.

Async HTTP Client

AsyncHTTPClient is a library that streamlines working with network request. Features include:

- Default headers
- Network reachability monitoring
- URL and Query string serialization
- Easy Multipart form requests
- Automatic Async handling
- Content-type Parsing (JSON, Binary)

An easier way...

Use the **android-async-http** library to make simple async network requests.

```
import com.loopj.android.http.*;
...

AsyncHttpClient client = new AsyncHttpClient();
client.get("http://www.google.com", new
    AsyncHttpResponseHandler() {
        @Override
        public void onSuccess(String response) {
            System.out.println(response);
        }
    }
);
```

Universal Image Loader

The **Universal-Image-Loader** is a widely used library with **unified** memory and disk **caches** for images and which can **load remote images** into any view with ease.

```
ImageView myImage =
    (ImageView) this.findViewById(R.id.my_image);

ImageLoader imgLoader = ImageLoader.getInstance();

imgLoader.displayImage(
    "http://www.awesome.com/myimage.jpg", myImage);
```

Working with APIs

The most common tasks on Android:

- Establishing a user **session** (authentication)
- Fetching data from an **API**
- Translating API data to **JSONObject**
- Transforming JSON to **Java Models**

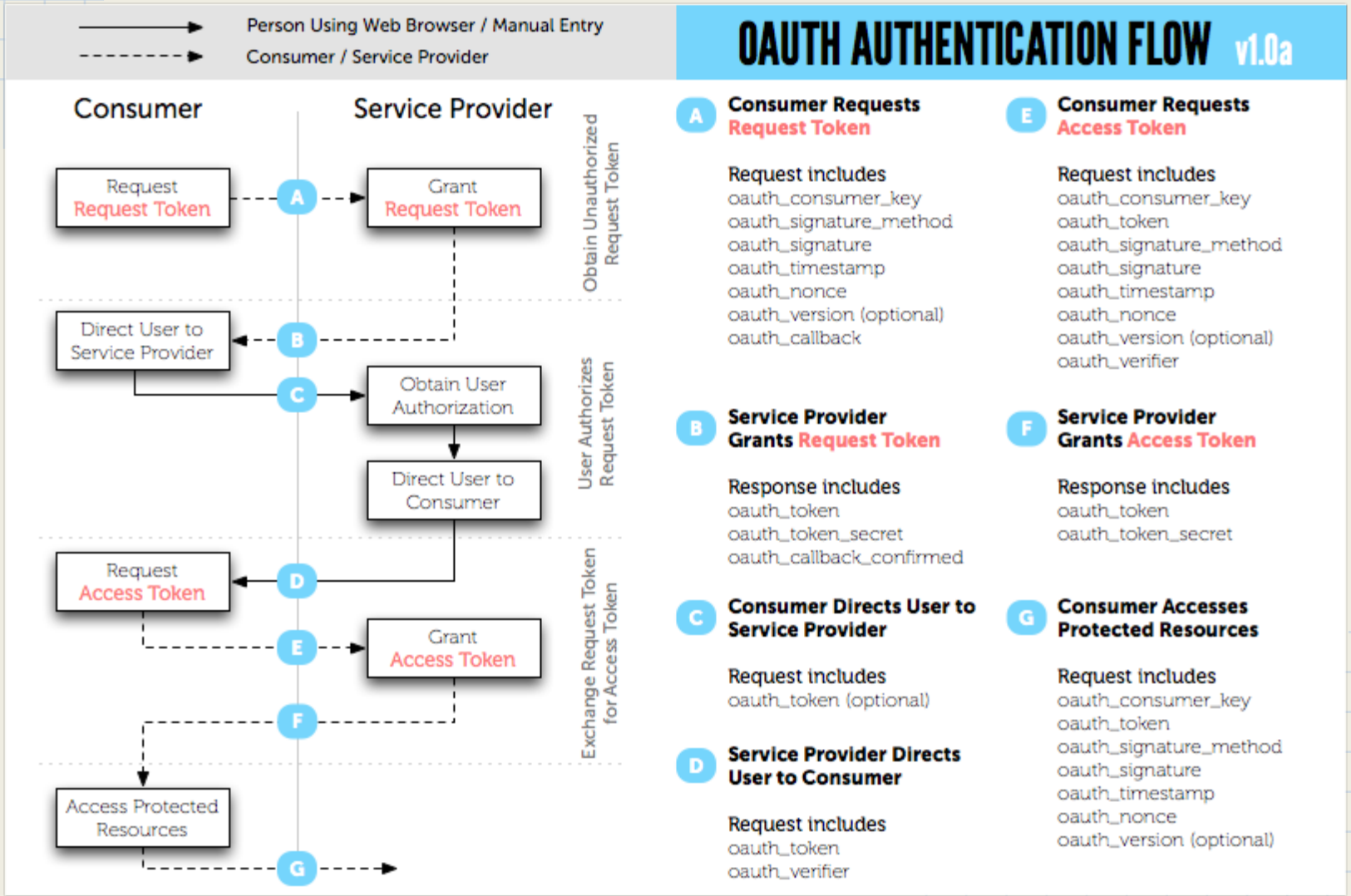
There are many **third-party libraries** to help with each of these steps.

Authentication

Interacting with APIs typically involves some type of authentication. Common techniques include:

- OAuth 1.0a (Yelp, Twitter, Tumblr)
- OAuth 2.0 (Facebook, Instagram, Google)
- Built-in authentication for Facebook and Twitter

OAuth 1.0a


Person Using Web Browser / Manual Entry
Consumer / Service Provider
Consumer
Service Provider
Request Request Token
A
Grant Request Token
Obtain Unauthorized Request Token
Direct User to Service Provider
B
C
Obtain User Authorization
Direct User to Consumer
User Authorizes Request Token
Request Access Token
D
E
Grant Access Token
F
Exchange Request Token for Access Token
Access Protected Resources
G
OAUTH AUTHENTICATION FLOW v1.0a
A Consumer Requests Request Token
Request includes
oauth_consumer_key
oauth_signature_method
oauth_signature
oauth_timestamp
oauth_nonce
oauth_version (optional)
oauth_callback
B Service Provider Grants Request Token
Response includes
oauth_token
oauth_token_secret
oauth_callback_confirmed
C Consumer Directs User to Service Provider
Request includes
oauth_token (optional)
D Service Provider Directs User to Consumer
Request includes
oauth_token
oauth_verifier
E Consumer Requests Access Token
Request includes
oauth_consumer_key
oauth_token
oauth_signature_method
oauth_signature
oauth_timestamp
oauth_nonce
oauth_version (optional)
oauth_verifier
F Service Provider Grants Access Token
Response includes
oauth_token
oauth_token_secret
G Consumer Accesses Protected Resources
Request includes
oauth_consumer_key
oauth_token
oauth_signature_method
oauth_signature
oauth_timestamp
oauth_nonce
oauth_version (optional)

OAuth 1.0a

OAuth 1.0a is kind of a pain. Common issues include:

- Signing is really complex and hard to debug
- Out of sync clock will cause signature mismatch
- Inconsistent implementation across providers
- Multipart uploads (like photo uploading)

OAuth 1.0a

OAuth 1.0a is so complex because it provides secure authentication over an unsecure transport.

OAuth 2.0

Most APIs are moving to OAuth 2.0 which sidesteps OAuth 1.0a complexity by requiring secure requests:

- Requires **secure https**
- Initial authentication grants access token
- Subsequent requests include access token

Networking Wrap-up

- Network requests **require** the INTERNET application **permission** and must be run a **background thread**.
- **AsyncTask** is a simple **threading** strategy for doing work in the **background**.
- Third-party **libraries** like **AsyncHttpClient** and **Universal-Image-Loader** are commonly used to simplify common networking tasks.
- **OAuth** is the industry standard for Authentication for APIs