Build X: Android

Lecture Seven:
Networking and APIs

Notes: http://android.kcl.tech/

Before We Begin...

- Go to **http://android.kcl.tech** and download the **api-example.zip** file under Lecture Seven.

- Check that you can open it in Android Studio without errors.

Networking and APIs

- What is an API?
- HTTP Requests
- Threads
- Using a Web-Based API in Android

What is an API?

What is an API?

“An API is a set of programming instructions and standards for accessing a software application or tool.”

- Basically, they are a way for your software to **use someone else’s software**.
- APIs can provide **data**, **functionality**, **connections**, etc.

What is an API?

- We’re going to use the **World Bank API** as an example.
  - It provides access to over 8,000 economical and environmental data points.

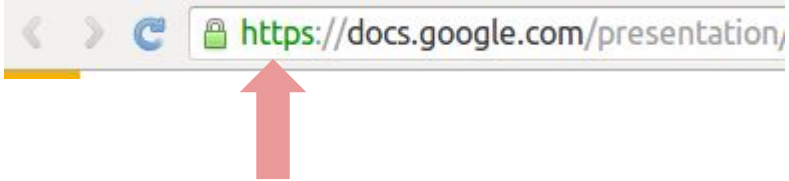
- This is also the API featured in the 2nd-year SEG project.
  - What a coincidence…

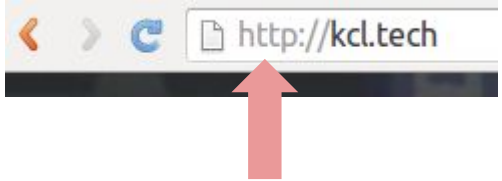
HTTP Requests

HTTP Requests

- HTTP = *Hyper-Text Transfer Protocol*
  - A set of rules for sending text from one place to another

- You've used it millions of times already:





HTTP Requests

An application sends a **request**…

...and the resource sends a **response**.

HTTP Requests: A Request

- **Five** main components:
  - A **verb**
  - A **path**
  - A **protocol**
  - A set of **headers** *(optional)*
  - A **body** *(optional)*

HTTP Requests: A Request

```
GET  http://example.com/api/coolstuff  HTTP/1.1
```

HTTP Requests: A Request

```
GET http://example.com/api/coolstuff  HTTP/1.1
```

HTTP Requests: A Request

```
GET http://example.com/api/coolstuff HTTP/1.1
```

HTTP Requests: A Request

```
GET  http://example.com/api/coolstuff  HTTP/1.1
```

HTTP Requests: A Request

```
GET  http://example.com/api/coolstuff  HTTP/1.1
Accept: text/json
Cache-Control: no-cache
```

HTTP Requests: A Request

```
GET http://example.com/api/coolstuff  HTTP/1.1
Accept: text/json
Cache-Control: no-cache
```

HTTP Requests: A Request

```
GET http://example.com/api/coolstuff HTTP/1.1
Accept: text/json
Cache-Control: no-cache
```

HTTP Requests: A Request

```
GET  http://example.com/api/coolstuff HTTP/1.1
Accept: text/json
Cache-Control: no-cache
```

HTTP Requests: A Request

```
GET  http://example.com/api/coolstuff  HTTP/1.1
Accept: text/json
Cache-Control: no-cache
```

HTTP Requests: A Request

```
POST  http://example.com/api/coolstuff  HTTP/1.1
Accept: text/json
Cache-Control: no-cache

{
    "message": "I send this!",
    "version": 2.45
}
```

HTTP Requests: A Request

```
POST http://example.com/api/coolstuff  HTTP/1.1
Accept: text/json
Cache-Control: no-cache


{
    "message": "I send this!",
    "version": 2.45
}
```

HTTP Requests: A Request

```
POST http://example.com/api/coolstuff HTTP/1.1
Accept: text/json
Cache-Control: no-cache


{
    "message": "I send this!",
    "version": 2.45
}
```

HTTP Requests: A Request

```
POST  http://example.com/api/coolstuff HTTP/1.1
Accept: text/json
Cache-Control: no-cache


{
    "message": "I send this!",
    "version": 2.45
}
```

HTTP Requests: A Request

```
POST  http://example.com/api/coolstuff  HTTP/1.1
Accept: text/json
Cache-Control: no-cache

{
    "message": "I send this!",
    "version": 2.45
}
```

HTTP Requests: A Request

```
POST  http://example.com/api/coolstuff  HTTP/1.1
Accept: text/json
Cache-Control: no-cache

{
    "message": "I send this!",
    "version": 2.45
}
```

HTTP Requests: A Request

```
POST  http://example.com/api/coolstuff  HTTP/1.1
Accept: text/json
Cache-Control: no-cache
```

```
{
    "message": "I send this!",
    "version": 2.45
}
```

HTTP Requests: A Response

- **Three** main components:
  - A **status code**
  - A set of **headers** *(technically optional)*
  - A **body** *(optional)*

HTTP Requests: A Response

```
HTTP/1.1  200  OK
```

HTTP Requests: A Response

```
HTTP/1.1  200  OK
```

HTTP Requests: A Response

```
HTTP/1.1 200  OK
```

Side Note: Status Codes

- **200 OK**
- 301 Moved Permanently
- 403 Forbidden
- 404 Not Found
- 418 I’m a Teapot
- 500 Internal Server Error

HTTP Requests: A Response

```
HTTP/1.1  200  OK
Content-Type: text/json


{
    "message": "You're awesome!",
    "validation": "No, really."
}
```

HTTP Requests: A Response

```
HTTP/1.1 200 OK
Content-Type: text/json


{
    "message": "You're awesome!",
    "validation": "No, really."
}
```

HTTP Requests: A Response

```
HTTP/1.1 200  OK
Content-Type: text/json


{
    "message": "You're awesome!",
    "validation": "No, really."
}
```

HTTP Requests: A Response

```
HTTP/1.1  200  OK
Content-Type: text/json

{
    "message": "You're awesome!",
    "validation": "No, really."
}
```

HTTP Requests: A Response

```
HTTP/1.1  200  OK
Content-Type: text/json

{
    "message": "You're awesome!",
    "validation": "No, really."
}
```

HTTP Requests: A Response

```
HTTP/1.1  200  OK
Content-Type: text/json
```

```
{
    "message": "You're awesome!",
    "validation": "No, really."
}
```

HTTP Requests: JSON

- A format for encoding data, widely used on the web.

- Easy to write, human-readable, easy for computers to understand, etc.

- A very lightweight format.
  - Like XML on a diet.

HTTP Requests: JSON

```
{
    "name": "Mark",
    "age": 22,
    "is_alive": true,
    "job": {
        "company": "Unitu Ltd.",
        "role": "Software Engineer"
     },
     "friends": ["Alan", "Steve"]
}
```

HTTP Requests: JSON vs. XML

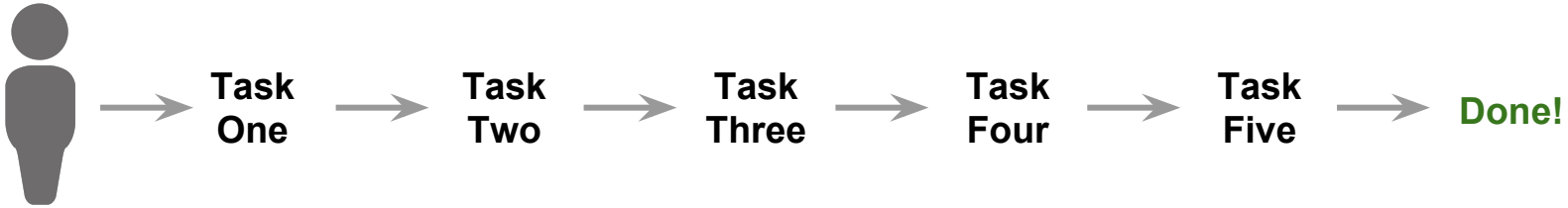
```
<person>
    <name>Mark</name>
    <age>22</age>
    <is_alive>true</is_alive>
    <job>
        <company>Unitu Ltd.</company>
        <role>Software Engineer</role>
    </job>
    <friends>
        <friend>Alan</friend>
        <friend>Steve</friend>
    </friends>
</person>
```

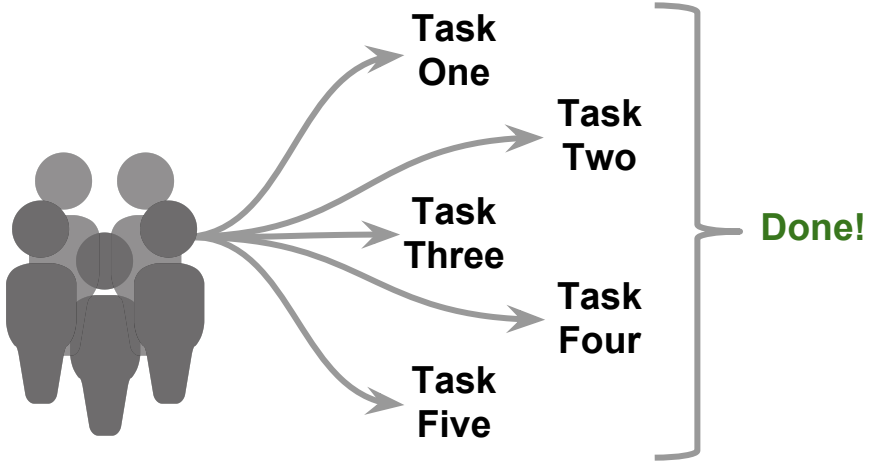
53% larger!

Threads

Threads

- Threads allow software to work on **multiple tasks at once**.





Threads: The UI Thread

- The **UI Thread** runs your app's user interface, and by default all of your computation as well.

- When you app stutters or the UI freezes, that's because you're **doing too much work** on the UI Thread.

- The UI Thread **must be kept fairly unbusy**!

Threads: Networking on the UI Thread

- **This is a huge no-no!**

- A network request **takes time to complete**, so you must never do it on the UI Thread!

- You need to **create a new thread** to do any network work!


MAKE GIFS AT GIFSOUP.COM

Break! (10 mins)

Using a Web-Based API in Android

Using an API

- There are **two distinct steps** in a single API transaction.

- **Making the request**.

- **Parsing the result**.

Using an API: Making the Request

- As we know, we need to use a new thread. We could handle the threads manually, but that's complex.

- We can use an `AsyncTask` to do the hard work for us!

- These are used in the same was as an adapter, or a database helper: we create our own class that **extends** `AsyncTask`.

Using an API: Making the Request

```
public class ApiRequest extends AsyncTask {
    // ...
}
```

Using an API: Making the Request

- The `AsyncTask` class takes three generic type arguments.

- A generic type tells the class what type of objects it will be dealing with.
  - You’ve used them before: `ArrayList<String>` or `HashMap<String, Object>`.

- `AsyncTask` requires three type parameters:
  - Parameter(s)
  - Progress updates
  - Result

Using an API: Making the Request

```
public class ApiRequest extends AsyncTask<String, Double, String> {
    // ...
}
```

Using an API: Making the Request

- We can override `doInBackground(...)` to control what the class does on the background thread it will create.
  - In our case, this is make a network request.

- The **arguments** will be whatever type you define for parameters, and the **return type** will be whatever type you specified for result.

Using an API: Making the Request

```
public class ApiRequest extends AsyncTask<String, Double, String> {
    protected String doInBackground(String... params) {
        // do something in the background, like a network action
        return "Must return a String, or null”;
    }
}
```

Using an API: Parsing the Result

- We can override `onPostExecute(...)` to manage what the class does when the result arrives.

- The **argument** will be whatever type you specified for result, and it does not have a return type (i.e. it is a void method).

- **<u>This runs on the UI Thread.</u>**

Using an API: Making the Request

```
public class ApiRequest extends AsyncTask<String, Double, String> {
    protected String doInBackground(String... params) {
        // do something in the background, like a network action
        return "Must return a String, or null”;
    }

    protected void onPostExecute(String result) {
        // use the result
    }
}
```


TOTALLY AWESOME!

Example:
Basic String API

http://dev.markormesher.co.uk/kcltech/api.txt

Example:
Basic String API

Done!

Bonus GIF.