Lecture 7

Networking I

CIS 195-201 - Dominic Holmes

Attendance

tiny.cc/cis195-att

Today

- DispatchQueue
- JSON
- Codable
- Live Demo

DispatchQueue

Main

```
DispatchQueue.main.async {
    // what is DispatchQueue?
    // what is .main?
    // what is .async?
}
```

Main

```
DispatchQueue.main.async {
    // what is DispatchQueue?
    // what is .main?
    // what is .async?
}
```

The docs:

- DispatchQueue
- Main
- Async

True or False

“I should begin HTTP requests from within a DispatchQueue.main.async block.”

True or False

“I have an array of 100 contacts. I should sort this on a background thread.”

True or False

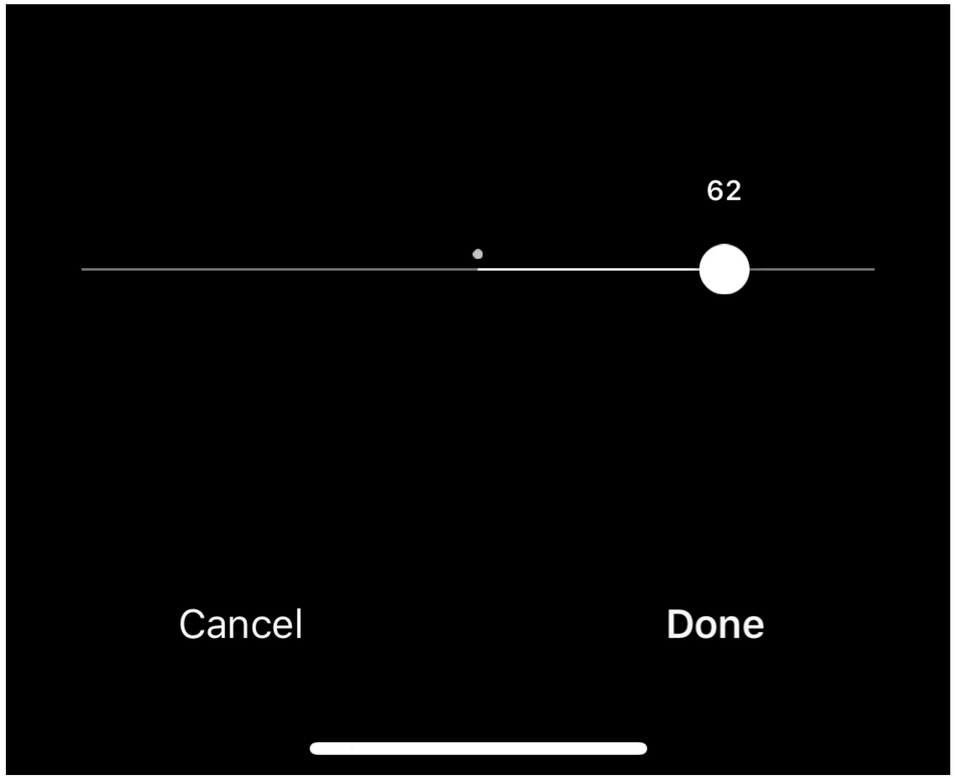
“I have an array of 100,000 contacts. I should sort this on a background thread.”

True or False

“Expensive animations should be called on a background thread to keep the main thread free.”

True or False

“I’m an Instagram developer. I want the user to drag a slider to edit a filter on a photo. I should perform the image editing on the main thread.”



JSON

JSON

- JavaScript Object Notation
  - The most common way to pass information around the Web and communicate with servers
- Not the fastest, but *definitely the easiest to use*
  - *Does that make it the fastest? Development speed above all!*
- I recommend using a Chrome/Safari extension to make JSON prettier

JSON Examples

```
{
  "calendar": [
    {
      "end": "2020-03-15",
      "name": "Spring Term Break",
      "start": "2020-03-07"
    },
    {
      "end": "2020-03-16",
      "name": "Classes Resume",
      "start": "2020-03-16"
    }
  ]
}
```

```
{
  "document": {
    "venue": [
      {
        “id": 593,
        "dateHours": [
          {
            "date": "2019-10-20",
            "meal": [
              {
                "close": "15:00:00",
                "open": "11:00:00",
                "type": "Brunch"
              },
              {
                "close": "20:00:00",
                "open": "17:00:00",
                "type": "Dinner"
              }
            ]
          },
```

Open Object with “{”

Field of type Int

Open Array with “[”

Field of type String OR Date, depending how you look at it

Field of type Array<Object>

Field of type String. Could be defined as an Enum?

JSON Examples

- **api.pennlabs.org/dining/venues**
  - Returns a list of dining halls on campus
- **developer.nps.gov/api/v1/parks?api_key=<YOUR-API-KEY>**
  - Returns a list of national parks, with an api key passed into the header
- **https://api.pennlabs.org/laundry/halls**
  - Returns a list of laundry rooms

What is an API?

- "Application Programming Interface"
- From the national parks service API docs: *"An application programming interface (API) is a set of requirements that allows one application to talk to another."*
- Put simply: **it is a URL that you visit, which returns a JSON file *(or other data)***
- An essential component of any API is a defined *structure* for the data.
  - For example, the `/parks` route will always return a list of parks, and each park will always have a name and a id. In this case, '/parks' is an *endpoint*.
- APIs typically have different **endpoints** — each one returns a different set of information

Codable

Codable in a nutshell

- This is just a protocol, that you can choose to adopt or not
- This makes it **very easy** to convert data from JSON-format (or many other formats!) into Swift Structs
  - This is usually a difficult and error-prone process (manually decoding dictionaries). Swift makes it easy.

Codable in a nutshell

- This is just a protocol, that you can choose to adopt or not
- This makes it **very easy** to convert data from JSON-format (or many other formats!) into Swift Structs
  - This is usually a difficult and error-prone process (manually decoding dictionaries). Swift makes it easy.
- `typealias Codable = Encodable & Decodable`

How would you describe Codable?

- Magic?

- The perfect protocol?

- A cool breeze on a hot summer day?

Codable Example

```
{
  "calendar": [
    {
      "end": "2020-03-15",
      "name": "Spring Term Break",
      "start": "2020-03-07"
    },
    {
      "end": "2020-03-16",
      "name": "Classes Resume",
      "start": "2020-03-16"
    }
  ]
}
```

Codable Example

```
{
  "calendar": [
    {
      "end": "2020-03-15",
      "name": "Spring Term Break",
      "start": "2020-03-07"
    },
    {
      "end": "2020-03-16",
      "name": "Classes Resume",
      "start": "2020-03-16"
    }
  ]
}
```

Codable Example

```
{
 "end": "2020-03-15",
 "name": "Spring Term Break",
 "start": "2020-03-07"
}
```

```
struct UniversityEvent {
    let end: String
    let name: String
    let start: String
}
```

Codable Example

```
{
 "end": "2020-03-15",
 "name": "Spring Term Break",
 "start": "2020-03-07"
}
```

```
struct UniversityEvent {
    let end: String
    let name: String
    let start: String
}
```

Before Codable, we would create a dictionary from the json….
And then manually check each key.

Before Codable

```
{
 "end": "2020-03-15",
 "name": "Spring Term Break",
 "start": "2020-03-07"
}
```

```
struct UniversityEvent {
    let end: String
    let name: String
    let start: String
}
```

```
let myJSONDict: Dictionary<String, Any> = // convert JSON to dict...

if let end = myJSONDict["end"] as? String,
   let name = myJSONDict["name"] as? String,
    let start = myJSONDict["start"] as? String {

    let myEvent = UniversityEvent(end: end, name: name, start: start)
}
```

Before Codable, we would create a dictionary from the json….
And then manually check each key…

Before Codable: This is Hard!

```
let myJSONDict: Dictionary<String, Any> =
// convert JSON to dict...

if let end = myJSONDict["end"] as?
String,
   let name = myJSONDict["name"] as?
String,
    let start = myJSONDict["start"] as?
String {

    let myEvent = UniversityEvent(end:
end, name: name, start: start)
}
```

Decoding this into a dictionary in a TYPE SAFE way — not easy!


```
{
  "document": {
    "venue": [
      {
        “id": 593,
        "dateHours": [
          {
            "date": "2019-10-20",
            "meal": [
              {
                "close": "15:00:00",
                "open": "11:00:00",
                "type": "Brunch"
              },
              {
                "close": "20:00:00",
                "open": "17:00:00",
                "type": "Dinner"
              }
            ]
          },
```

So what does Codable do?

```
{
 "end": "2020-03-15",
 "name": "Spring Term Break",
 "start": "2020-03-07"
}
```

```swift
struct UniversityEvent: Codable
{
    let end: String
    let name: String
    let start: String
}
```

Just conforming to Codable lets us transform our struct into JSON!

Codable uses the variable name as the key, and the actual value as the value.

The key thing with Codable: **every nested type inside a Codable struct must ALSO be Codable.** Basic types like String and Int get this functionality for free.

So what does Codable do?

```
{
 "end": "2020-03-15",
 "name": "Spring Term Break",
 "start": "2020-03-07"
}
```

```swift
let myEvent = UniversityEvent(end: "10-21-20",
                              name: "Event Name",
                              start: "10-19-20")

// Convert to JSON
let json = JSONEncoder().encode(value: Encodable)
```

So what does Codable do?

```
{
 "end": "2020-03-15",
 "name": "Spring Term Break",
 "start": "2020-03-07"
}
```

```swift
let myEvent = UniversityEvent(end: "10-21-20",
                              name: "Event Name",
                              start: "10-19-20")

// Convert to JSON
let json = try! JSONEncoder().encode(myEvent)
```

Codable Example

```
struct Calendar: Codable {                    2  Type 'Calendar' does not conform to protocol 'Decodable'
    let calendar: [UniversityEvent]

    struct UniversityEvent {
        let end: String
        let name: String
        let start: String
    }
}
```

Codable Example

```
{
  "calendar": [
    {
      "end": "2020-03-15",
      "name": "Spring Term Break",
      "start": "2020-03-07"
    },
    {
      "end": "2020-03-16",
      "name": "Classes Resume",
      "start": "2020-03-16"
    }
  ]
}
```

```
struct Calendar: Codable {
    let calendar: [UniversityEvent]

    struct UniversityEvent: Codable {
        let end: String
        let name: String
        let start: String
    }
}
```

More Codable Features: Dates!

```
struct Calendar: Codable {
    let calendar: [UniversityEvent]

    struct UniversityEvent: Codable {
        let end: Date
        let name: String
        let start: Date
    }
}

let decoder = JSONDecoder()
let customFormatter = DateFormatter()
// nsdateformatter.com
customFormatter.dateFormat = "M-dd-yy"
decoder.dateDecodingStrategy = .formatted(customFormatter)

let calendar = try! decoder.decode(Calendar.self, from: myData)
```

More Codable Features: Custom Variable Names

```
struct Calendar: Codable {
    let calendar: [UniversityEvent]

    struct UniversityEvent: Codable {
        let end: Date
        let name: String
        let start: Date
    }

    // If you want your variable names different from the JSON
    enum CodingKeys: String, CodingKey {
        case end = "end-date"
        case name = "event-name"
        case start = "start-date"
    }
}
```

www.swiftjson.guide

The best Codable guide I've found. Especially needed for date behavior, CodingKeys, etc.

Due Before Next Class

- **Nothing! Enjoy your break :)**

Links

- Survey: **tiny.cc/cis195-att**
- Piazza: **tiny.cc/cis195-piazza**

Live Demo: Penn Mobile Dining

https://api.pennlabs.org/laundry/halls/ids