Networking with URLSession

Hands-On Challenges

Networking with URLSession

Audrey Tam

Notice of Rights

Notice of Liability

Trademarks

Challenge #10: Customizing the Delegate Queue

By Audrey Tam

In HalfTunes-APIManager, replace the creation of `downloadsSession` with the following:

```
let delegateQueue = OperationQueue()
delegateQueue.name = "sessionDelegateQueue"
delegateQueue.qualityOfService = .userInitiated
apiManager.downloadsSession = URLSession(configuration: configuration,
  delegate: self, delegateQueue: delegateQueue)
```

Disable all breakpoints except the one in the `URLSessionDownloadDelegate didFinishDownloadingTo` method.

Build and run, enter a search term, then quickly download as many songs as you can before the breakpoint stops execution.

In the debug navigator, examine the thread where the breakpoint stopped: it has a name — `sessionDelegateQueue` — and its QOS is USER_INITIATED.

There might be other threads with the same name.

Continue execution, and more `sessionDelegateQueue` threads will appear, even though the delegate queue is serial! But only one is actually executing. Open the others, and you'll see they're all waiting on a mutual exclusion (mutex) lock.

Suppose we really want a **concurrent** delegate queue: we can specify a concurrent dispatch queue as the delegate queue's `underlyingQueue`. Add the following line:

```
delegateQueue.underlyingQueue = DispatchQueue.global(qos: .userInitiated)
```

The previous `qualityOfService` statement is now redundant, so you can comment it out.

Build and run, enter a search term, then quickly download as many songs as you can before the breakpoint stops execution.

In the debug navigator, the breakpoint thread doesn't have the name `sessionDelegateQueue`, but it's `user-initiated-qos` and concurrent.

Continue execution, and more `user-initiated-qos` threads will appear, but still, only one is actually executing. Open the others, and you'll see they're all waiting on a mutual exclusion (mutex) lock.

So it doesn't seem to do much good to specify a concurrent `underlyingQueue` ... but try one more thing: set the `maxConcurrentOperationCount` property:

```
delegateQueue.maxConcurrentOperationCount = 3
```

Build and run etc again. Now, 2 of the threads are executing! An iPhone has only 2 CPUs, so you won't get more than 2 running concurrently. Curiously, this trick also works with the default serial delegate queue.

Note: You can create a custom (private) dispatch queue with a name:

```
// Serial private dispatch queue
let mySerialDelegateQueue = DispatchQueue(label: "serialDelegateQueue")
// Concurrent private dispatch queue
let myConcurrentDelegateQueue = DispatchQueue(label:
"concurrentDelegateQueue", attributes: .concurrent)
```

Quality of service defaults to `.default`. You can set the private queue's `qos` property:

```
mySerialDelegateQueue.qos = .userInitiated
```