iOS Application Development

Lecture 7: Saving Data • Closures • Animations • Working with the Web

Prof. Dr. Jan Borchers
Media Computing Group
RWTH Aachen University

WS '22/'23 • hci.rwth-aachen.de/ios
Saving Data
Encoding and Decoding With Codable

- Use Encoder & Decoder objects to encode/decode Codable objects
- Codable ↔ Data

```swift
struct Note: Codable {
    let title: String
    let text: String
    let timestamp: Date
}

let myNote = Note(title: "Dry cleaning", text: "Pick up suit from dry cleaners", timestamp: Date())

let propertyListEncoder = PropertyListEncoder()
if let encodedNote = try? propertyListEncoder.encode(myNote) {
    print(encodedNote)
}

let propertyListDecoder = PropertyListDecoder()
if let decodedNote = try? propertyListDecoder.decode(Note.self, from: encodedNote) {
    print(decodedNote)
}
```
Storing Data: Sandboxing & the FileManager

```swift
let archiveURL = documentsDirectory.appendingPathComponent("notesData")
  .appendingPathExtension("plist")
```
let documentsDirectory = 
    FileManager.default.urls(for: .documentDirectory, 
        in: .userDomainMask).first!

let archiveURL = 
    documentsDirectory.appendingPathComponent("notesData") 
        .appendingPathExtension("plist")

let propertyListDecoder = PropertyListDecoder()
if let retrievedNoteData = try? Data(contentsOf: archiveURL),
    let decodedNote = try? propertyListDecoder.decode(Note.self, 
        from: retrievedNoteData) { print(decodedNote) }

let propertyListEncoder = PropertyListEncoder()
let encodedNote = try? propertyListEncoder.encode(myNote)
try? encodedNote?.write(to: archiveURL, options: .noFileProtection)
Closures
Syntax

```swift
func sum(numbers: [Int]) -> Int {
    // Code that adds together the numbers array
    return total
}
```

```swift
let sumClosure = { (numbers: [Int]) -> Int in
    // Code that adds together the numbers array
    return total
}
```
// A closure with no parameters and no return value
let printClosure = { () -> Void in
    print("This closure does not take any parameters and does not return a value.")
}

// A closure with parameters and no return value
let printClosure = { (string: String) -> Void in
    print(string)
}

// A closure with parameters and a return value
let randomNumberClosure = { (minValue: Int, maxValue: Int) -> Int in
    // Code that returns a random number between `minValue` and `maxValue`
}
Passing Closures as Arguments

```swift
let sortedTracks = tracks.sorted(by: { (firstTrack: Track, secondTrack: Track) -> Bool in
    return firstTrack.trackNumber < secondTrack.trackNumber
})

let sortedTracks = tracks.sorted { (firstTrack: Track, secondTrack: Track) -> Bool in
    return firstTrack.trackNumber < secondTrack.trackNumber
} // trailing closure syntax!

let sortedTracks = tracks.sorted { (firstTrack, secondTrack) in
    return firstTrack.trackNumber < secondTrack.trackNumber
}

let sortedTracks = tracks.sorted { return $0.trackNumber < $1.trackNumber }

let sortedTracks = tracks.sorted { $0.trackNumber < $1.trackNumber }

let sortedTracks = tracks.sorted(by: <)
```
Collection Functions Using Closures

// Initial array
let firstNames = ["Johnny", "Nellie", "Aaron", "Rachel"]

// Creates an empty array that will be used to store the full names
var fullNames: [String] = []

for name in firstNames {
    let fullName = name + " Smith"
    fullNames.append(fullName)
}

let fullNames = firstNames.map { (name) -> String in
    return name + " Smith"
}

let fullNames = firstNames.map { $0 + " Smith" }
Collection Functions Using Closures

```swift
let numbers = [4, 8, 15, 16, 23, 42]

var numbersLessThan20: [Int] = []

for number in numbers {
    if number < 20 {
        numbersLessThan20.append(number)
    }
}

let numbersLessThan20 = numbers.filter { number -> Bool in return number < 20 }

let numbersLessThan20 = numbers.filter { $0 < 20 }
```
let numbers = [4, 8, 15, 16, 23, 42]

var total = 0

for number in numbers {
    total = total + number
}

let total = numbers.reduce(0) { (currentTotal, newValue) -> Int in
    return currentTotal + newValue
}

let total = numbers.reduce(0, { $0 + $1 })

let total = numbers.reduce(0, +)
Closures Capture Their Environment

• Anything in scope when the closure is created is in scope inside the closure

```swift
animate {
    self.view.backgroundColor = .red
}
```

• If view is removed during animation, view is held in memory until the closure finishes
Animations
Animations

- Transition from one state to another
- Can define the character of an app
Animations

• Transition from one state to another

• Can define the character of an app

• Direct the user’s attention
Animations

• Transition from one state to another
• Can define the character of an app
• Direct the user’s attention
• Keep the user oriented
Animations

• Transition from one state to another
• Can define the character of an app
• Direct the user’s attention
• Keep the user oriented
• Connect to user behaviors
What Can Be Animated?

- UIView:
  - alpha
  - backgroundColor
  - bounds
  - center
  - frame
  - transform
UIView Animation Methods

animate(withDuration:animations:)  
animate(withDuration:animations:completion:)  
animate(withDuration:delay:options:animations:completion:)  
animate(withDuration:delay:usingSpringWithDamping:initialSpringVelocity:options:animations:completion:)

UIView.animate(withDuration: 2.0) {
    //animation closure
    view.alpha = 0.3
}
UIView Animation Methods

animate(withDuration:animations:)
animate(withDuration:animations:completion:)
animate(withDuration:delay:options:animations:completion:)
animate(withDuration:delay:usingSpringWithDamping:initialSpringVelocity:options:animations:completion:)

```swift
UIView.animate(withDuration: 1.0, animations: {
    //animation closure
    view.alpha = 1.0
}) { (_: Bool) in
    //completion closure
    UIView.animate(withDuration: 1.0) {
        //second animation closure
        view.alpha = 0.0
    }
}
```
The Transform Property

• Type: CGAffineTransform

<table>
<thead>
<tr>
<th>Type</th>
<th>Initializer</th>
<th>Parameter Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Scale</td>
<td>init(scaleX: CGFloat, y: CGFloat)</td>
<td>The factors by which to scale your view</td>
</tr>
<tr>
<td>Rotate</td>
<td>init(rotationAngle: CGFloat)</td>
<td>The angle (in radians) by which to rotate your view. Positive value = counterclockwise</td>
</tr>
<tr>
<td>Translate</td>
<td>init(translationX: CGFloat, y: CGFloat)</td>
<td>The value by which to move (shift) your view</td>
</tr>
</tbody>
</table>

• You can combine transform objects:

```swift
let scaleTransform = CGAffineTransform(scaleX: 2.0, y: 2.0)
let rotateTransform = CGAffineTransform(rotationAngle: .pi)
let combinedTransform = scaleTransform.concatenating(rotateTransform)
```
Animation in Practice

• Use animation and motion effects judiciously

• Strive for realism and credibility

• Use consistent animation

• Make animations optional
Working With the Web
Working With the Web: What's in a Request?

• URL: https://itunes.apple.com/us/app/keynote/id409183694?mt=12
  
  Protocol | Subdomain | Domain | Path | Query
  
  • Request Type (most common are GET and POST)
  
  • Headers (e.g., User-Agent, authentication, cookies)
  
  • Body (HTML, CSS, images, JSON, …)
Create a URL

• URL:

```swift
let url = URL(string: "https://www.apple.com")!
```

• Create a data task and the request:

```swift
Task {
    let (data, response) = try await URLSession.shared.data(from: url)
    if let httpResponse = response as? HTTPURLResponse,
       httpResponse.statusCode == 200,
       let string = String(data: data, encoding: .utf8) {
        print(string)
    }
}
```
Working With an API

NASA Astronomy Picture of the Day (APOD)

API: https://api.nasa.gov/
Working With the APOD API

- Use the demo key `DEMO_KEY` to get the image of the day

```swift
Task {
    let (data, response) = try await URLSession.shared.data(from: url)
    if let httpResponse = response as? HTTPURLResponse,
      httpResponse.statusCode == 200,
    let string = String(data: data, encoding: .utf8) {
        print(string)
    }
}
```

```json
{
    "date": "2005-02-22",
    "explanation": "Are Saturn’s auroras like Earth’s? To help answer this question, the Hubble Space Telescope and the Cassini spacecraft monitored Saturn’s South Pole simultaneously as Cassini closed in on the gas giant in January 2004. Hubble snapped images in ultraviolet light, while Cassini recorded radio emissions and monitored the solar wind. Like on Earth, …",
    "hdurl": "http://apod.nasa.gov/apod/image/0502/saturnauroras_hst_big.jpg",
    "media_type": "image",
    "service_version": "v1",
    "title": "Persistent Saturnian Auroras",
    "url": "http://apod.nasa.gov/apod/image/0502/saturnauroras_hst.jpg"
}
```
Modify a URL With URL Components

- Dynamically add query items to a URL using URLComponents

```swift
// Build the URL
var components = URLComponents(string: "https://api.nasa.gov/planetary/apod")!
components.queryItems = [
    "api_key": "DEMO_KEY",
    "date": "2013-07-16"
].map {
    URLQueryItem(name: $0.key, value: $0.value)
}

// Perform the network request
Task {
    let (data, response) = try await URLSession.shared.data(from: components.url!)
    if let httpResponse = response as? HTTPURLResponse,
        httpResponse.statusCode == 200,
        let string = String(data: data, encoding: .utf8) {
        print(string)
    }
}
```
NASAAstronomyPictureoftheDayApp

1. Create Url

2. Request Data with API keys

3. Create a Swift model

4. Decode JSON

5. Update UI
The Swift Model

• The PhotoInfo model:

```swift
struct PhotoInfo: Codable {
    var title: String
    var description: String
    var url: URL
    var copyright: String?

    enum CodingKeys: String, CodingKey {
        case title
        case description = "explanation"
        case url
        case copyright
    }
}
```

Prof. Dr. Jan Borchers: iOS Application Development
NASA Astronomy Picture of the Day App

1. Create Url ✔
2. Request Data with API keys ✔
3. Create a Swift model ✔
4. Decode JSON
5. Update UI

Last May 16 the Moon slid through Earth’s shadow, completely immersed in the planet’s dark umbra for about 1 hour and 25 minutes during a total lunar eclipse. In this composit ed timelapse view, the partial and total phases of the eclipse were captured as the Moon tracked above the horizon from Amundsen-Scott South Pole Station. There it shared a cold and stary south polar night with a surging display of the aurora australis and central Milky Way. In the foreground are the BICEP (right) and South Pole telescopes at the southernmost station’s Dark Sector Laboratory. But while polar skies can be spectacular, you won’t want to go to the South Pole to view the total lunar eclipse coming up on November 8. Instead, that eclipse can be seen from Inuvik in Alaska, Alaska.
```swift
var components = URLComponents(string: "https://api.nasa.gov/planetary/apod")!
components.queryItems = [
    "api_key": "DEMO_KEY",
    "date": "2013-07-16"
].map { URLQueryItem(name: $0.key, value: $0.value) }

// Perform the network request
Task {
    let (data, response) = try await URLSession.shared.data(from: components.url)!
    let jsonDecoder = JSONDecoder()
    if let httpResponse = response as? HTTPURLResponse, httpResponse.statusCode == 200,
        let photoInfo = try? jsonDecoder.decode(PhotoInfo.self, from: data) {
            print(photoInfo)
        }
}
```
func fetchPhotoInfo() -> PhotoInfo{
    // Build the URL
    var components = URLComponents(string: "https://api.nasa.gov/planetary/apod")!
    components.queryItems = [
        "api_key": "DEMO_KEY",
        "date": "2013-07-16"
    ].map { URLQueryItem(name: $0.key, value: $0.value) }

    // Perform the network request
    let (data, response) = try await URLSession.shared.data(from: components.url!)

    let jsonDecoder = JSONDecoder()

    if let httpResponse = response as? HTTPURLResponse,
    httpResponse.statusCode == 200,
    let photoInfo = try? jsonDecoder.decode(PhotoInfo.self, from: data) {
        return photoInfo
    }
}
func fetchPhotoInfo() async throws -> PhotoInfo{
    // Build the URL
    var components = URLComponents(string: "https://api.nasa.gov/planetary/apod")!
    components.queryItems = [
        "api_key": "DEMO_KEY",
        "date": "2013-07-16"
    ].map { URLQueryItem(name: $0.key, value: $0.value) }

    // Perform the network request
    let (data, response) = try await URLSession.shared.data(from: components.url!)
    let jsonDecoder = JSONDecoder()

    if let httpResponse = response as? HTTPURLResponse,
       httpResponse.statusCode == 200,
       let photoInfo = try? jsonDecoder.decode(PhotoInfo.self, from: data) {
        return photoInfo
    }
}

Async Calls

Prof. Dr. Jan Borchers: iOS Application Development
func fetchPhotoInfo() async throws -> PhotoInfo{
    // Build the URL
    var components = URLComponents(string: "https://api.nasa.gov/planetary/apod")!
    components.queryItems = [
        "api_key": "DEMO_KEY",
        "date": "2013-07-16"
    ].map { URLQueryItem(name: $0.key, value: $0.value) }

    // Perform the network request
    let (data, response) = try await URLSession.shared.data(from: components.url!)

    guard let httpResponse = response as? HTTPURLResponse,
        httpResponse.statusCode == 200 else {
        throw PhotoInfoError.itemNotFound
    }

    let jsonDecoder = JSONDecoder()
    let photoInfo = try jsonDecoder.decode(PhotoInfo.self, from: data)
    return(photoInfo)
}

class PhotoInfoError: Error, LocalizedError {
    case itemNotFound
}
NASA Astronomy Picture of the Day App

1. Create URL
2. Request Data with API keys ✓
3. Create a Swift model ✓
4. Decode JSON ✓
5. Update UI
Update the UI

```swift
override func viewDidLoad() {
    super.viewDidLoad()

    Task {
        do {
            let photoInfo = try await fetchPhotoInfo()
            updateUI(with: photoInfo)
        } catch {
            updateUI(with: error)
        }
    }
}

func updateUI(with photoInfo: PhotoInfo) {
    Task {
        do {
            let image = try await fetchImage(from: photoInfo.url)
            title = photoInfo.title
            imageView.image = image
            descriptionLabel.text = photoInfo.description
            copyrightLabel.text = photoInfo.copyright
        } catch {
            updateUI(with: error)
        }
    }
}
```
Concurrency
Multi Threading in iOS

• Run multiple tasks at the same time
• Run slow or expensive tasks in the background
• Free the main thread so it responds to the UI
Synchronous and Asynchronous

• Synchronous
  • One task completes before another begins
  • Ties up the main thread (main queue)

• Asynchronous
  • Multiple tasks run simultaneously on multiple threads (concurrency)
  • Tasks run in the background thread (background queue)
  • Frees up the main thread
Swift Concurrency

- Swift uses Actors to protect against concurrent updates
- A special Actor called the MainActor is used for UIKit
  - Standard UIKit controllers use the MainActor
  - Safe to update UI in a Task’s closure that was created in the context of the MainActor
  - Code after a method that can suspend (marked with `await`) will run synchronously in the context of the MainActor
Grand Central Dispatch

- Allows your app to execute multiple tasks concurrently on multiple threads
- Assigns tasks to "dispatch queues" and assigns priority
- Controls when your code is executed
Grand Central Dispatch

• Main queue
  • Created when an app launches
  • Highest priority
  • Used to update the UI and respond quickly to user input

• Background queues
  • Lower priority
  • Used to run long-running operations
Dispatch Queue

DispatchQueue.global(qos: .background).async {
    // Do some background work
    DispatchQueue.main.async {
        // Update the UI to indicate the work has been completed
    }
}