iOS Application Development

Lecture 8: Clousures, Improving Complex Input Screens, and Animations

Simon Völker & Philipp Wacker
Media Computing Group
RWTH Aachen University

hci.rwth-aachen.de/ios
Seminar Review Meetings

• First Seminar meetings are next week:

<table>
<thead>
<tr>
<th>Topic</th>
<th>Group</th>
<th>Presentation</th>
<th>Discussions</th>
</tr>
</thead>
<tbody>
<tr>
<td>MapKit, CoreLocation</td>
<td>Group 8</td>
<td>18.11.19, first group</td>
<td>11.11.19</td>
</tr>
<tr>
<td>Debugging in Xcode and Instruments</td>
<td>Group 4</td>
<td>18.11.19, second group</td>
<td>11.11.19</td>
</tr>
<tr>
<td>SiriKit</td>
<td>Group 2</td>
<td>19.11.19, second group</td>
<td>12.11.19</td>
</tr>
<tr>
<td>Extensions &amp; Inter-App communication</td>
<td>Group 5</td>
<td>19.11.19, first group</td>
<td>12.11.19</td>
</tr>
<tr>
<td>WatchOS</td>
<td>Group 14</td>
<td>25.11.19, second group</td>
<td>18.11.19</td>
</tr>
<tr>
<td>Swift UI</td>
<td>Group 11</td>
<td>25.11.19, first group</td>
<td>18.11.19</td>
</tr>
<tr>
<td>Scene Kit</td>
<td>Group 7</td>
<td>26.11.19, first group</td>
<td>19.11.19</td>
</tr>
<tr>
<td>GameplayKit</td>
<td>Group 9</td>
<td>26.11.19, second group</td>
<td>19.11.19</td>
</tr>
<tr>
<td>RealityKit &amp; Reality Composer</td>
<td>Group 1</td>
<td>09.12.19, first group</td>
<td>02.12.19</td>
</tr>
<tr>
<td>Networking on iOS</td>
<td>Group 6</td>
<td>2.12.19, first group</td>
<td>25.11.19</td>
</tr>
<tr>
<td>ClassKit</td>
<td>Group 3</td>
<td>2.12.19, second group</td>
<td>25.11.19</td>
</tr>
<tr>
<td>Bringing People into AR</td>
<td>Group 13</td>
<td>09.12.19, second group</td>
<td>02.12.19</td>
</tr>
<tr>
<td>Core Data</td>
<td>Group 10</td>
<td>3.12.19, first group</td>
<td>26.11.19</td>
</tr>
<tr>
<td>Core ML + Create ML</td>
<td>Group 12</td>
<td>3.12.19, second group</td>
<td>26.11.19</td>
</tr>
</tbody>
</table>
Seminar

• 20 minute presentations + 10 min Q&A

• Deliverables: Slides + well documented demo code
Closures
Closures

• Similar to Objective-C Blocks and Java Lambdas

• Self-contained blocks of functionality
  • „Anonymous functions“

```swift
{ (parameters) -> return type in
  statements
}
```
Closure Syntax

• Function:

```swift
func sumFunction(numbers: [Int]) -> Int {
    var total = 0
    //Code
    return total
}

let sum = sumFunction(numbers: [5, 34, 12, 42])
```

• Closure:

```swift
let sumClosure = { (numbers: [Int]) -> Int in
    var total = 0
    //Code
    return total
}

let sum = sumClosure([8, 34, 16, 42])
```
Closure Types

- **No parameters, no return value:**
  ```swift
  let printClosure1 = { () -> Void in
      print("This closure does not take any parameters and does not return a value."
  }
  ```

- **With parameters, no return value:**
  ```swift
  let printClosure2 = { (string: String) -> Void in
      print(string)
  }
  ```

- **No parameters, with return value:**
  ```swift
  let randomNumberClosure1 = { () -> Int in
      return 0
  }
  ```

- **With parameters, with return value:**
  ```swift
  let randomNumberClosure2 = { (minValue: Int, maxValue: Int) -> Int in
      return 0
  }
  ```
Trailing Closure

Closure as the only argument:

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

Closure as the last argument:

```swift
func performRequest(url: String, response: (Int) -> Void)
{
}
performRequest(url: "https://www.apple.com") { (data) in
    print(data)
}
```
Simplifying Closures

```swift
let sortedTracks = tracks.sorted {
    return firstTrack.starRating < secondTrack.starRating
}
```

**Infer the return type:**

```swift
let sortedTracks = tracks.sorted {
    return firstTrack.starRating < secondTrack.starRating
}
```

**Use placeholder arguments:**

```swift
let sortedTracks = tracks.sorted {
    return $0.starRating < $1.starRating
}
```

**Automatic return**

```swift
let sortedTracks = tracks.sorted {
    $0.starRating < $1.starRating
}
```
let names = "Johnny", "Nellie", "Aaron", "Rachel"
var fullNames: [String] = []

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

Use the map() function:

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

Short version:

let fullNames = names.map { $0 + " Smith" }
let numbers = [4, 8, 15, 16, 23, 42]  
var numbersLessThan20: [Int] = []  
for number in numbers {  
    if number < 20 {  
        numbersLessThan20.append(number)  
    }  
}  

Use the filter() function:  
let numbersLessThan20 = numbers.filter { (number) -> Bool in  
    return number < 20  
}  

Short version:  
let numbersLessThan20 = numbers.filter { $0 < 20 }
let numbers = [8, 6, 7, 5, 3, 0, 9]

var total = 0

for number in numbers {
    total = total + number
}

Use the reduce() function:

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

Short version:

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

```swift
func makeIncrementer(forIncrement amount: Int) -> () -> Int {
    var runningTotal = 0
    func incrementer() -> Int {
        runningTotal += amount
        return runningTotal
    }
    return incrementer
}

let incrementByTen = makeIncrementer(forIncrement: 10)

incrementByTen() // 10
incrementByTen() // 20
incrementByTen() // 30

let incrementBySeven = makeIncrementer(forIncrement: 7)
incrementBySeven() // 7
incrementByTen() // 40
let alsoIncrementBy10 = incrementByTen
alsoIncrementBy10() // 50
```
Improving Complex Input Screens
Improving Complex Input Screens
Animation
Why Animations?

• Direct the User's Attention

• Keep the User Oriented

• Connect User Behaviors
What Can Be Animated?

- **UIView:**
  - frame
  - bounds
  - center
  - transform
  - alpha
UIView Animations

Animation Closures:

```swift
animate(withDuration:animations:)
animate(withDuration:animations:completion:)
animate(withDuration:delay:options:animations:completion:)
```

Animate with Duration:

```swift
UIView.animate(withDuration: 2.0) {
    aView.alpha = 0.3
}
```
Animation Playground