17. November 2015

Übung 1 mit Swift

Architektur verteilter Anwendungen

Thorsten Kober Head Solutions iOS/OS X, SemVox GmbH

Überblick

1 Einführung

2 Typen, Optionals und Pattern Matching

3 Speichermanagement

4 Closures

5 Vererbung, Protocols und Extensions

1

Einführung

Einführung

iOS

Mac OS

Watch OS

tv OS

IDE

Sprachen

C

Objective-C

C++

Sprachen

C

Objective-C

C++

Swift

Übersetzung

LLVM Compiler Infrastructure

Übersetzung

Programmiersprache

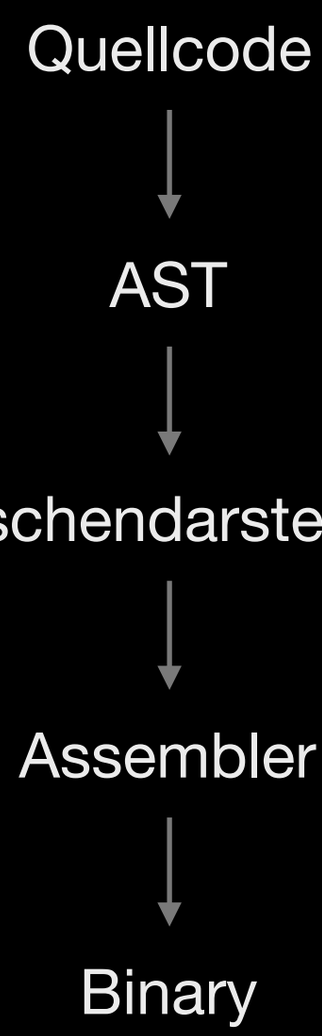
C | Objective-C | C++ | Swift

Compiler Frontend

CLANG | Swift Compiler

Compiler Backend

LLVM



Swift

Einflüsse

Rust

Smalltalk

C

Objective-C

C#

C++

Python

Haskel

D

CLU

Ruby

Werte

Sicher

Modern

Intuitiv

Schnell

2

Typen, Optionals und Pattern Matching

Typisierung

```
let text: String = "Hallo Welt!"
```

Typinferenz

```
let text = "Hallo Welt!"
```

Typinferenz

```
let languageName = "Swift"
```

let languageName: String

```
let latestVersion = 2.1
```

let latestVersion: Double

```
let introduced = 2014
```

let introduced: Int

```
let isAwesome = true
```

let isAwesome: Bool

Typsicherheit

```
let net = 100

let tax = 0.19

let gross = net * tax
```

Typsicherheit

```
let net = 100

let tax = 0.19

let gross = net * tax
```

Binary operator '*' cannot be applied to operands of type 'Int' and 'Double'

Typsicherheit

```
let net = 100

let tax = 0.19

let gross = Double(net) * tax
```

Typsicherheit

```
let net: Double = 100

let tax = 0.19

let gross = net * tax
```

Optionals

```
var error: NSError = nil
error = doSomething()
```

Optionals

```
var error: NSError = nil
```

Nil cannot initialize specified type 'NSError'

```
error = doSomething()
```

Optionals

```
var error: NSError? = nil
error = doSomething()
```

Optionals

```
var int: Int?

var double: Double?

var string: String?

var array: [String]?

var dictionary: [String: Int]?
```

Unwrapping

```
var error: NSError? = nil
error = doSomething()

print(error!)
```

Unwrapping

```
var error: NSError? = nil
error = doSomething()

print(error!)
```

CRASH, fatal error: unexpectedly found nil while unwrapping an Optional value

Unwrapping

```
var error: NSError? = nil

error = doSomething()

if error != nil {
    print(error!)
}
```

Optional Binding

```
var error: NSError? = nil

error = doSomething()

if let e = error {
    print(e)
}
```

Unwrapping

```
if let error = doSomething() {
    print(error)
}
```

Optional Changing

```
var john: Person?
let street = john?.residence?.address?.street
```

Tuples

```
let httpResult = (404, "NotFound")
```

Tuples

```
var httpResult: (Int, String) = (200, "")
```

Tuples

```
var httpResult: (code: Int, message: String)

httpResult = (200, "")

print("code -> \(httpResult.code)“)

print("message -> \(httpResult.message)")
```

Tuples

```
var httpResult: (Int, String)
print("code -> \(httpResult.0)")

print("message -> \(httpResult.1)")
```

Tuples

```
var httpResult: (Int, String) = (200, "")
let (codeOnly, _) = httpResult

print("code -> \(codeOnly)")
```

Pattern Matching

Switch

```
let value: Int

switch value {

    case 1:
        print("One")

    case 2:
        print("Two")

    default:
        print("Just some number")

}
```

Switch

```
let value: String

switch value {

    case "one":
        print("Loneliest number you'll ever do")

    case "two":
        print("Can be as bad as one")

    default:
        print("Just some number")

}
```

Switch

```
let value: Int

switch value {

    case 1...4:
        print("a few")

    case 5...12:
        print("a lot")

    default:
        print("a ton")

}
```

Enums

```
enum TrainStatus {
    case OnTime
    case Delayed
}
```

Enums

```
enum TrainStatus {
    case OnTime
    case Delayed(Int)
}
```

Pattern Matching

```
switch trainStatus {

    case .OnTime:
        print("On time")

    case .Delayed:
        print("Delayed")

}
```

Pattern Matching

```
switch trainStatus {

    case .OnTime:
        print("On time")

    case .Delayed(let minutes):
        print("Delayed by \(minutes) minutes")

}
```

Pattern Matching

```
switch trainStatus {

    case .OnTime:
        print("On time")

    case .Delayed(1):
        print("Nearly on time")

    case .Delayed(2...10):
        print("Almost on time, I swear")

    case .Delayed(_):
        print("Deutsche Bahn")

}
```

Pattern Matching

```
enum VacationStatus {
    case Traveling(TrainStatus)
    case Relaxing(daysLeft: Int)
}
```

Pattern Matching

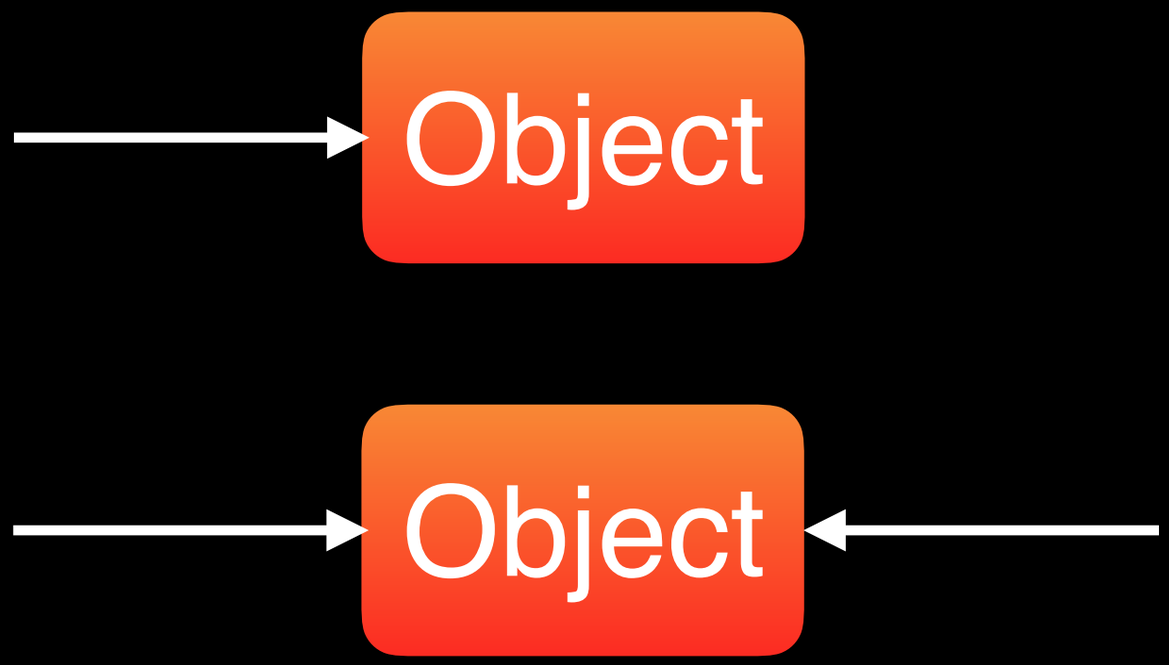
```
switch vacationStatus {

    case .Relaxing(let days):
        print("\(days) days left in paradise.“)

    case .Traveling(.OnTime):
        print("Train's on time.")

    case .Traveling(.Delayed(1...15)):
        print("Train is delayed.")

    case .Traveling(.Delayed(_)):
        print("#DB #railfail")

}
```

3

Speichermanagement

Garbage Collector

Reference Counting



`alloc()`

`retain()`

`release()`

`release()`

`dealloc()`

Reference Counting

Automatic Reference Counting

4

Closures

Closures vs Function Pointer

```
void calle(void (*callback)()) {

int calle_scope_var;

    // ...

    callback();

}
```

```
void callback() {

    // ...

}
```

Closures vs Function Pointer

```
func caller(callback: () -> ()) {

var calleScopeVar: Int?

    // ...

    callback()

}
```

```
func callback() {

    print( calleScopeVar )

}
```

Closures in Swift

```
() -> ()
```

Closures in Swift

```
() -> Void
```

Closures in Swift

```
(url: NSURL) -> (statusCode: Int, message: String)
```

Closures in Swift

```
let myClosure = { (url: NSURL) -> (statusCode: Int, message: String) in
    return (200, "Success")
}
```

Closures in Swift

```
let myClosure = { (url: NSURL) -> (statusCode: Int, message: String) in
    return (200, "Success")
}
```

let myClosure: (url: NSURL) -> (statusCode: Int, message: String)

Closures in Swift

```
let myClosure = { (_: NSURL) -> (Int, String) in
    return (200, "Success")
}
```

let myClosure: (NSURL) -> (Int, String)

Closures in Swift

```
func sort(byKey: String, inAscendingOrder ascending: Bool) { }
```

Closures in Swift

```
func sort(byKey: String, inAscendingOrder ascending: Bool) { }
```

external label internal label

Closures in Swift

```
func sort(byKey: String, inAscendingOrder ascending: Bool) {
    print("key -> \(byKey)")
    print("ascending -> \(ascending)")
}

sort("", inAscendingOrder: true)
```

Closures in Swift

```
func sort(byKey: String, ascending: Bool) {
    print("key -> \(byKey)")
    print("ascending -> \(ascending)")
}

sort("", ascending: true)
```

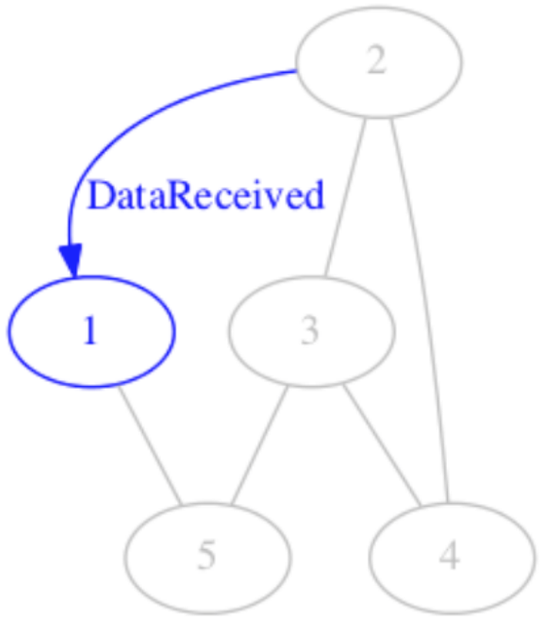
Closures in Swift

Untitled

Node	Event	Level	Description	Timestamp
2	Connect	Info	Peer '3' changed status to Connected	1447683815.28933
2	Processing	Warning	Start spreading rumor 'Hallo_Welt'	1447683815.28944
2	DataSent	Debug	Sent message (121 bytes) to 1	1447683815.28988
2	DataSent	Debug	Sent message (121 bytes) to 3	1447683815.2899
2	DataSent	Debug	Sent message (121 bytes) to 4	1447683815.28992
1	DataReceived	Info	Received message (121 bytes) from 2	1447683815.32437
1	DataSent	Debug	Sent message (121 bytes) to 5	1447683815.32499
1	Processing	Success	Peer '1' accepted rumor 'Hallo_Welt'	1447683815.32505
5	DataReceived	Info	Received message (121 bytes) from 1	1447683815.33905
5	DataSent	Debug	Sent message (121 bytes) to 3	1447683815.33954
5	Processing	Success	Peer '5' accepted rumor 'Hallo_Welt'	1447683815.33957
3	DataReceived	Info	Received message (121 bytes) from 5	1447683815.35695

▼ Message


```
{
  "timestamp" : 1447683815.289574,
  "payload" : {
    "rumorText" : "Hallo_Welt"
  },
  "sender" : "2",
  "type" : 1
}
```

Closures statt Delegates

```
func dotFromTopology(topology: AVATopology, vertexDecorator: AVAGraphvizVertexDecorator,
                     adjacencyDecorator: AVAGraphvizAdjacencyDecorator) -> String {
        var result = "digraph G {"
        for vertex in topology.vertices {
            result += "\n\(vertex) \(self.stringFromVertexDecoration(vertexDecorator(vertex: vertex)))"
        }
        for adjacency in topology.adjacencies {
            let decoration = adjacencyDecorator(adjacency: adjacency)
            let from: AVAVertex
            let to: AVAVertex
            if (decoration.direction == .InOrder) {
                from = adjacency.v1
                to = adjacency.v2
            } else {
                from = adjacency.v2
                to = adjacency.v1
            }
            result += "\n\(from) -> \(to) \(self.stringFromAdjacencyDecoration(decoration))"
        }
        result += "\n}"
        return result
    }
```

Closures statt Delegates

```
typealias AVAGraphvizVertexDecorator = (vertex: AVAVertex) -> AVAGraphvizVertexDecoration
```

```
typealias AVAGraphvizAdjacencyDecorator = (adjacency: AVAAdjacency) -> AVAGraphvizAdjacencyDecoration
```

```
func dotFromTopology(topology: AVATopology, vertexDecorator: AVAGraphvizVertexDecorator,
                     adjacencyDecorator: AVAGraphvizAdjacencyDecorator) -> String {
        var result = "digraph G {"
        for vertex in topology.vertices {
            result += "\n\(vertex) \(self.stringFromVertexDecoration(vertexDecorator(vertex: vertex)))"
        }
        for adjacency in topology.adjacencies {
            let decoration = adjacencyDecorator(adjacency: adjacency)
            let from: AVAVertex
            let to: AVAVertex
            if (decoration.direction == .InOrder) {
                from = adjacency.v1
                to = adjacency.v2
            } else {
                from = adjacency.v2
                to = adjacency.v1
            }
            result += "\n\(from) -> \(to) \(self.stringFromAdjacencyDecoration(decoration))"
        }
        result += "\n}"
        return result
    }
```

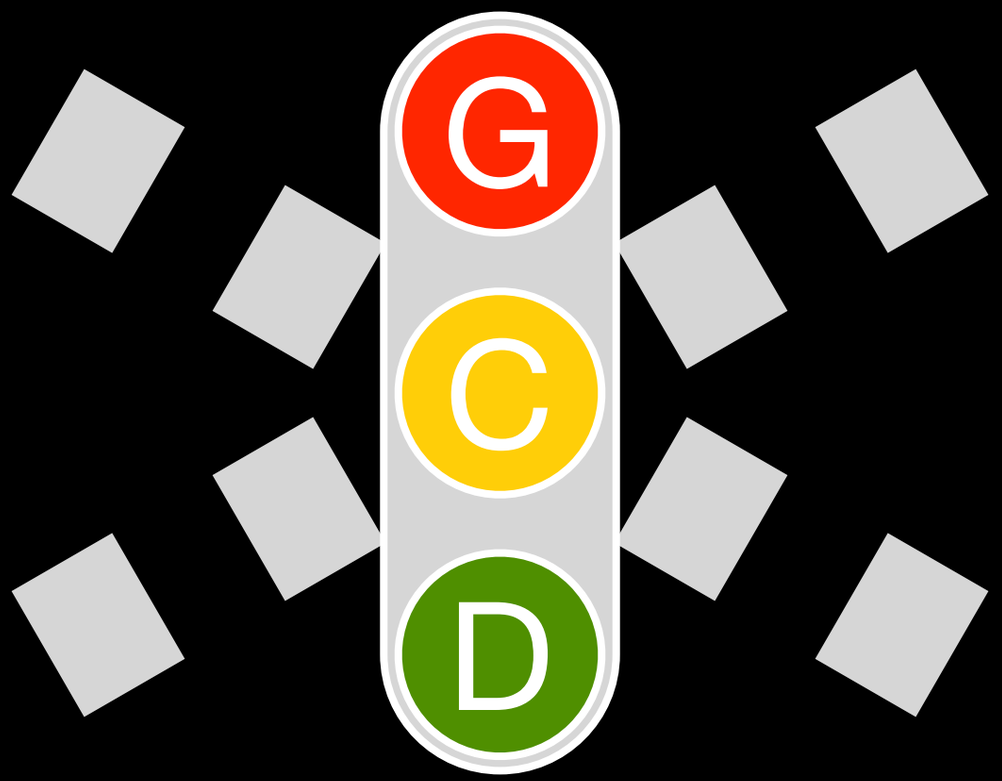
Closures statt Delegates

```
typealias AVAGraphvizVertexDecoration = (color: AVAGraphvizColor, style: AVAGraphvizLineStyle)


typealias AVAGraphvizVertexDecorator = (vertex: AVAVertex) -> AVAGraphvizVertexDecoration


typealias AVAGraphvizAdjacencyDecoration = (direction: AVAGraphvizAdjacencyDirection,
            color: AVAGraphvizColor, style: AVAGraphvizLineStyle, label: String?)


typealias AVAGraphvizAdjacencyDecorator = (adjacency: AVAAdjacency) -> AVAGraphvizAdjacencyDecoration


func dotFromTopology(topology: AVATopology, vertexDecorator: AVAGraphvizVertexDecorator,
                     adjacencyDecorator: AVAGraphvizAdjacencyDecorator) -> String {
        var result = "digraph G {"
        for vertex in topology.vertices {
            result += "\n\(vertex) \(self.stringFromVertexDecoration(vertexDecorator(vertex: vertex)))"
        }
        for adjacency in topology.adjacencies {
            let decoration = adjacencyDecorator(adjacency: adjacency)
            let from: AVAVertex
            let to: AVAVertex
            if (decoration.direction == .InOrder) {
                from = adjacency.v1
                to = adjacency.v2
            } else {
                from = adjacency.v2
                to = adjacency.v1
            }
            result += "\n\(from) -> \(to) \(self.stringFromAdjacencyDecoration(decoration))"
        }
        result += "\n}"
        return result
    }
```

Closures statt Delegates

```
let dot = GRAPHVIZ.dotFromTopology(topology, vertexDecorator: { (vertex: AVAVertex) ->
AVAGraphvizVertexDecoration in

    // vertexDecorator Closure

    return (vertex == ownPeerName ? AVAGraphvizBlue : AVAGraphvizGrey, AVAGraphvizSolid)

}, adjacencyDecorator: { (adjacency: AVAAdjacency) -> AVAGraphvizAdjacencyDecoration in

    // adjacencyDecorator Closure

    if adjacency.v1 == ownPeerName || adjacency.v2 == ownPeerName {
        let vertex = adjacency.v1 == ownPeerName ? adjacency.v2 : adjacency.v1
        let style = state.connectedPeers.contains(vertex) ? AVAGraphvizSolid : AVAGraphvizDotted
        return (AVAGraphvizAdjacencyDirection.Undirected, AVAGraphvizBlue, style, nil)
    } else {
        return (AVAGraphvizAdjacencyDirection.Undirected, AVAGraphvizGrey, AVAGraphvizSolid, nil)
    }

})
```

Grand Central Dispatch

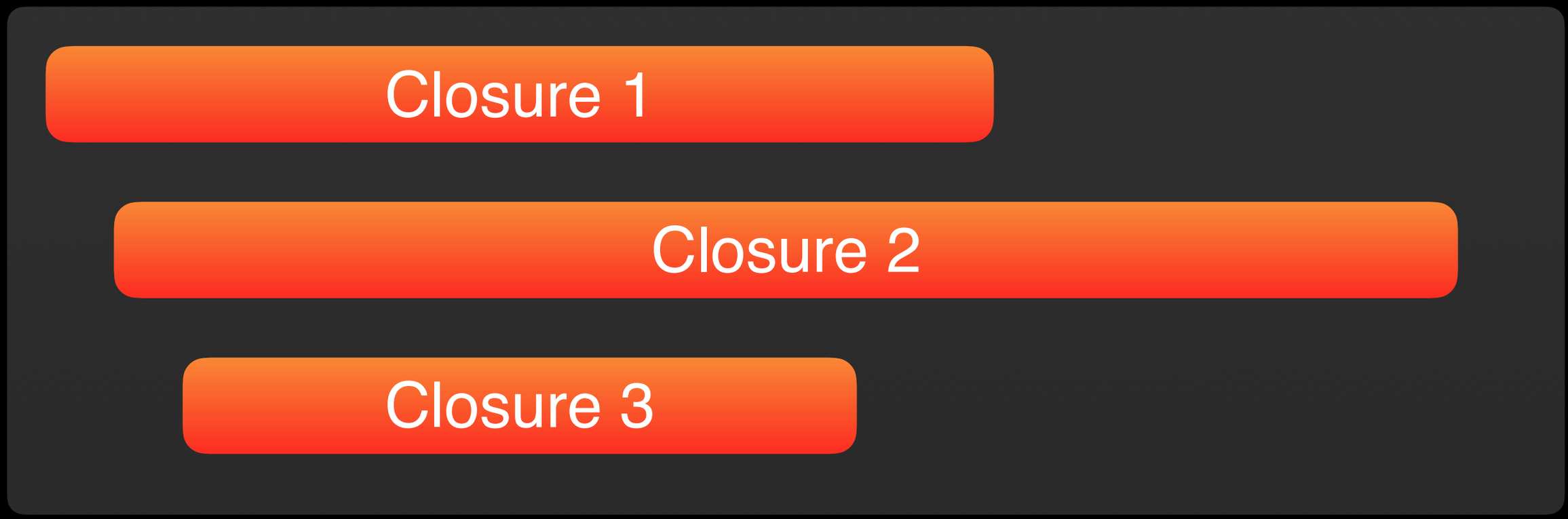


Grand Central Dispatch

DISPATCH_QUEUE_SERIAL

Closure 1 Closure 2 Closure3

DISPATCH_QUEUE_CONCURRENT



Nebenläufigkeit mit Closures

```
var renderingQueue = dispatch_queue_create(„ava.graphviz_adapter.rendering“, DISPATCH_QUEUE_CONCURRENT)

func renderPNGFromDOTFile(filePath: String, concurrent: Bool = true, result: (image: NSImage?) -> ()) {

  let rendering: dispatch_block_t = { () -> Void in

        // ... Rendering Stuff ...

        if concurrent {
            dispatch_async(dispatch_get_main_queue(), { () -> Void in
                result(image: NSImage(data: data))
            })
        } else {
            result(image: NSImage(data: data))
        }
    }

    if concurrent {
        dispatch_async(self.renderingQueue, rendering)
    } else {
        rendering()
    }

}
```

5

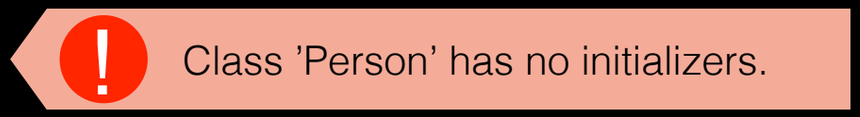
Vererbung, Protocols und Extensions

Klassen

```
class Person {
    var firstname: String
    var lastname: String
}
```

Klassen

```
class Person {
    var firstname: String
    var lastname: String
}
```



Klassen

```
class Person {
    var firstname: String
    var lastname: String
    init() {
    }
}
```

Klassen

```
class Person {

    var firstname: String

    var lastname: String

    init() {

    }

}
```

Return from initializer without initializing all stored properties

Klassen

```
class Person {
    var firstname: String
    var lastname: String
    init(firstname: String, lastname: String) {
        self.firstname = firstname
        self.lastname = lastname
    }
}
```

Klassen

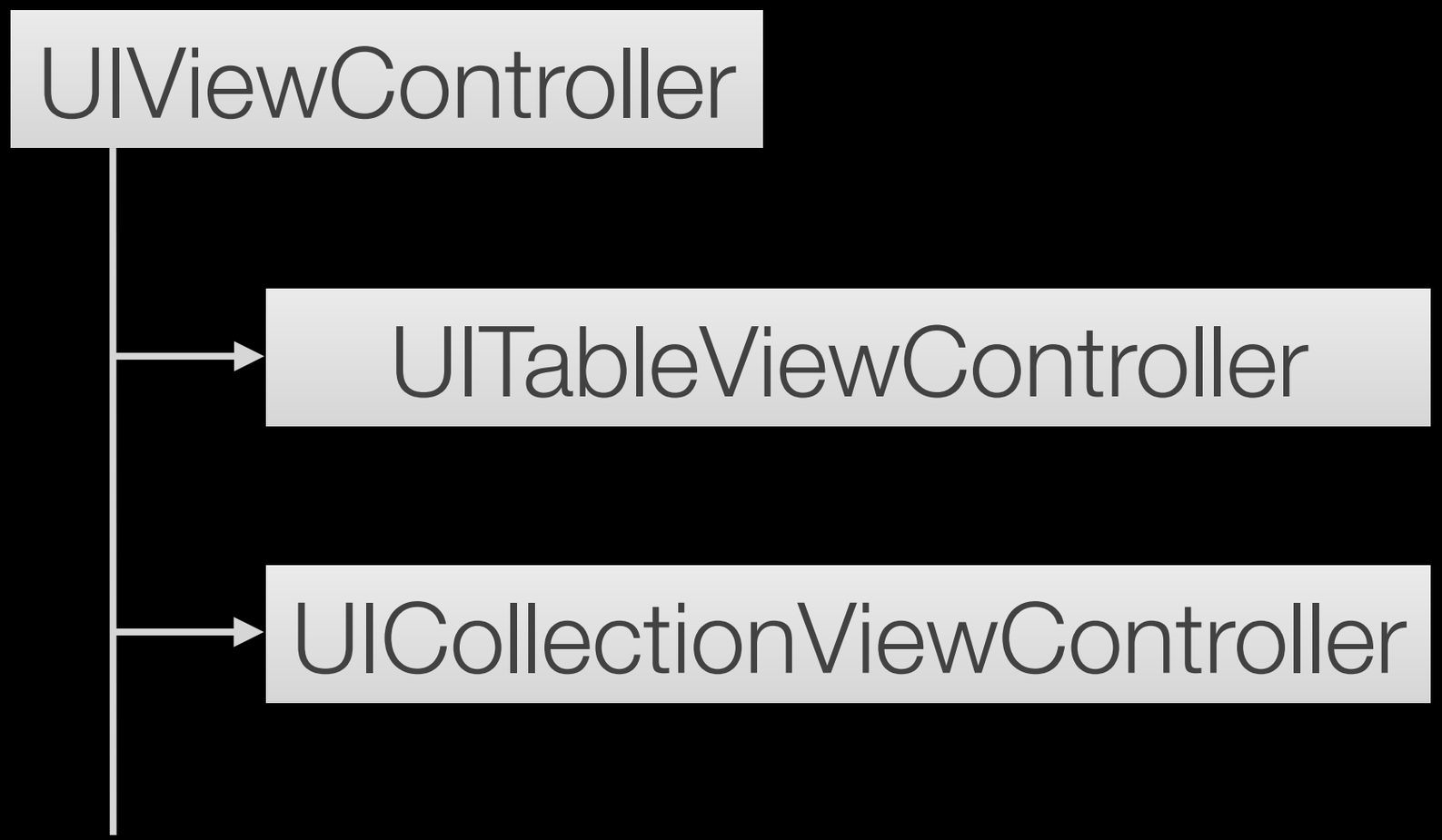
```
class Person {
    var firstname: String?
    var lastname: String?
}
```

Protocols

```
protocol AVAJSONConvertable {

    init(json: AVAJSON) throws

    func toJSON() -> AVAJSON

}
```

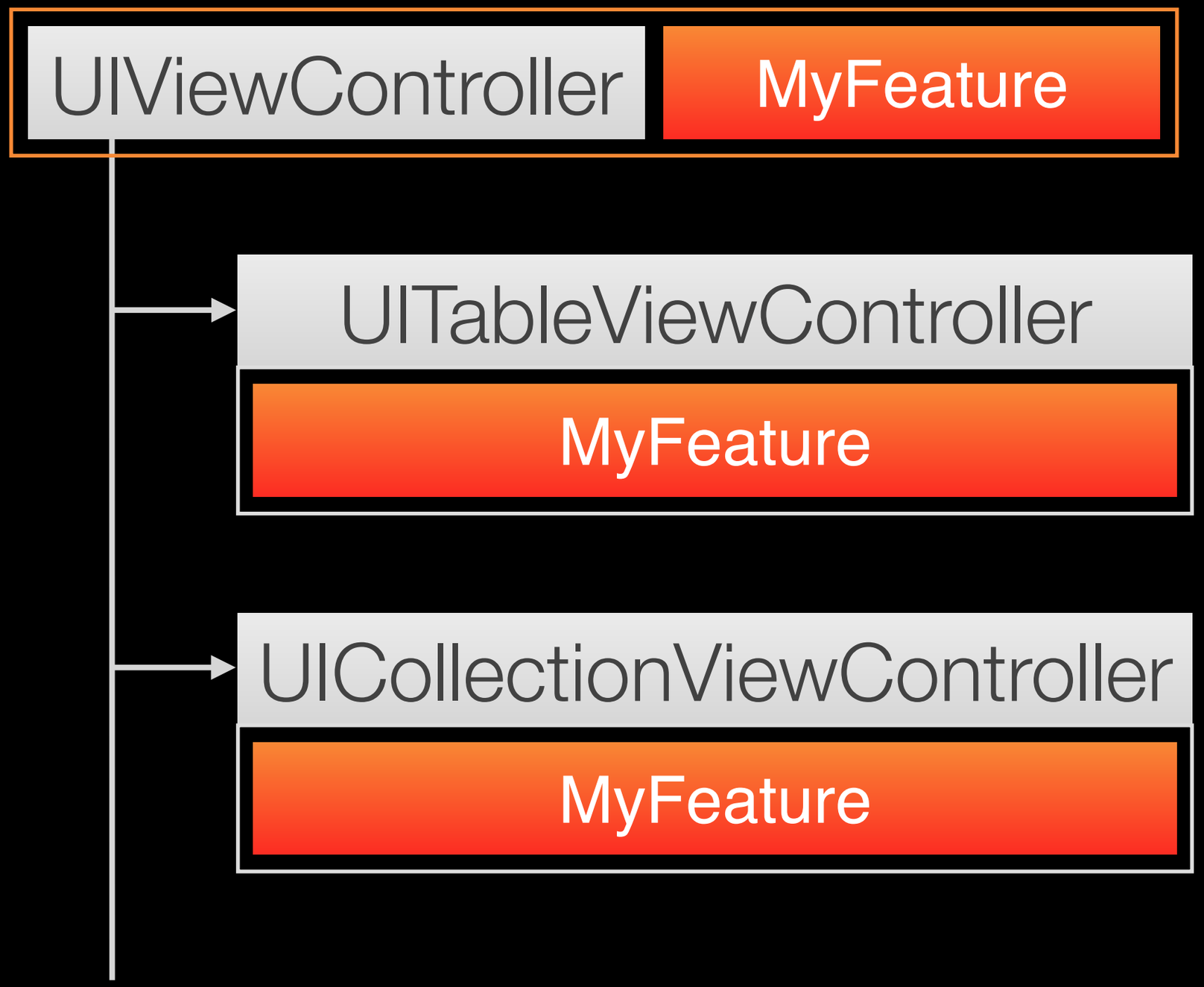
```
protocol AVAService: class {

    func startWithBufferedMessage(messages: [AVAMessage])

    func nodeManager(nodeManager: AVANodeManager,
didReceiveApplicationDataMessage message: AVAMessage)

    init(setup: AVASetup)

}
```

Extensions


UIViewController
UITableViewController
UICollectionViewController

. . .

Extensions


UIViewController
MyFeature
UITableViewController
MyFeature
UICollectionViewController
MyFeature
. . .

Extensions

```
extension Double {
    var km: Double { return self * 1_000.0 }
    var m: Double { return self }
    var cm: Double { return self / 100.0 }
    var mm: Double { return self / 1_000.0 }
    var ft: Double { return self / 3.28084 }
}

let oneInch = 25.4.mm
```

Extensions

```
extension NSApplication: AVALogging {

    func log(entry: AVALogEntry) {
        // ...
    }

    func setupLogger() {
        // ...
    }
}
```

Mehr über Swift

Introduction to Swift	Session 402	developer.apple.com/videos/wwdc2014/
Intermediate Swift	Session 403	developer.apple.com/videos/wwdc2014/
Advanced Swift	Session 404	developer.apple.com/videos/wwdc2014/
What’s New in Swift	Session 106	developer.apple.com/videos/wwdc2015/
The Swift Programming Language	iBook	iTunes / developer.apple.com