Swift: Ein Überblick

Nikolaj Schumacher

Ablauf

- Geschichte
- Sprache
- Ausblick

Geschichte

Objective-C

* 1983

Smalltalk

* 1972

C

* 1972

Objective-C

- C + Message Passing
- im Schatten von C++
- von niemandem verwendet

Objective-C

- C + Message Passing
- im Schatten von C++
- von **fast** niemandem verwendet

NeXT

* 1985

NeXT

- gegründet von Steve Jobs
- NeXTSTEP
- NeXTcube

Objective-C bei NeXT

- objektorientierte APIs
- GCC

NeXT

* 1985 † 1997

NeXT

* 1985 † 1997

Übernahme durch Apple

Apple

* 1976 † 1997

Übernahme durch NeXT

Objective-C bei Apple

- OS X (2001)
- Objective-C 2.0 (2006)
- iPhone (2007)
- LLVM/Clang ersetzen GCC (2011)

```objc
@interface Example: NSObject
- (void)doSomethingWithArgument: (int)x and: (int)y;
@end

@implementation Example
- (void)doSomethingWithArgument: (int)x and: (int)y {
    …
}

@end


Example *example = [[Example alloc] init];
[example doSomethingWithArgument:42 and:43];
```

Swift

* 2014

Swift

- Chris Lattner
- erster Commit 2010
- Swift 1.0 (2014)
- Swift 2.0 (2015)
- Open Source (2015)

Ziele

- sicher
- schnell
- ausdrucksstark

Ziele

- sicher (statt C)
- schnell
- ausdrucksstark

Ziele

- sicher (statt C)
- schnell (statt dynamisch)
- ausdrucksstark

Ziele

- sicher (statt C)
- schnell (statt dynamisch)
- ausdrucksstark (statt geschwätzig)

Einflüsse

C-Syntax

Haskell

C#

Objective-C

Ruby

CLU

D

Rust

Python

Kulturschock

- von dynamisch zu statisch
- von kompakt zu komplex
- von geschwätzig zu knapp

Sprache

Tools

Mac

- OS X
- Xcode
- Playgrounds

Open Source

- Linux
- swiftc
- IBM Swift Sandbox

Hello World

```
print("Hello World")
```

```
#!/usr/bin/xcrun swift -i
print("Hello World”)
```

Variablen

```
let pi = 3.141592653589783
var radius = 2.0
radius = 3.0
```

Variablen

- Konstanten: let
- Variablen: var
- Typen?

Typen

- statisch typisiert
- Typherleitung
- (nicht immer eindeutig)
- (nicht schnell)

```swift
let pi: Double = 3.141592653589783
var radius: Double = 2.0
radius = 3.0
```

```
var variable: Any = 3.14159265358979783
variable = 42
variable = "String"
variable = NSObject()
```

Schleifen

```
let primes = [2, 3, 5, 7, 11, 13, 17]
for prime in primes {
    print(prime)
}
```

```swift
let primes = [2, 3, 5, 7, 11, 13, 17]
for i in 0..<primes.count {
    print("\(i): \(primes[i])")
}
```

```swift
let primes = [2, 3, 5, 7, 11, 13, 17]
for i in primes.indices {
    print("\(i): \(primes[i])")
}
```

Closures

Closures

```
var closure: Int -> Int = {
    x in 2 * x
}

closure(42)
```

Funktionen

- Closures mit Namen
- Overloading
- Overloading anhand Rückgabetyp

Funktionen

```
func function(x: Int) -> Int {
    return 2 * x
}

function(42)
```

Funktionen

```
func increment(value: Int, by: Int) -> Int {
    return value + by
}

increment(10, by: 2)
```

Optionals

`null`

“I call it my billion-dollar mistake.”

–Tony Hoare

Optionals

- kein `null`
- `nil` ;)
- alle Typen (auch z.B. `Int`)

Vorteile Optionals

- nicht der Standardfall
- erzwungener Test vor Verwendung

Optionals

```
var optionalString: String?
```

```
if let string = optionalString {
    print(string)
}
```

Optionals

```
var optionalString: String?

guard let string = optionalString else {
    return
}

print(string)
```

Optionals

```
var optionalString: String?
```

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

Runtime-Check

Optionals

```
var optionalString: String?
```

```
print(optionalString ?? "unbekannt")
```

Optionals

```
var optionalString: String?
```

```
optionalString?.uppercaseString
```

Klassen & Structs

Klassen & Structs

- Referenz-Semantik
- Heap
- Polymorphie

- Value-Semantik
- Stack
- kopiert
- immutable

Structs in Stdlib

- `Int`, `Long`, `Float`, `Double`, `Bool`
- `Array`, `Set`, `Dictionary`
- fast alles

ARC

ARC

- Automatic Reference Counting
- statt Garbage Collection
- (nur Klassen)
- vgl. `smart_ptr` in C++

ARC

- Anzahl der Referenzen auf jedes Objekt wird gezählt
- Compiler generiert Code bei jeder Zuweisung
- Optimierer entfernt den Code oft wieder

Book

3

Page

1

Book

2

Page

1


Book
1
Page
1


Book
0
Page
1

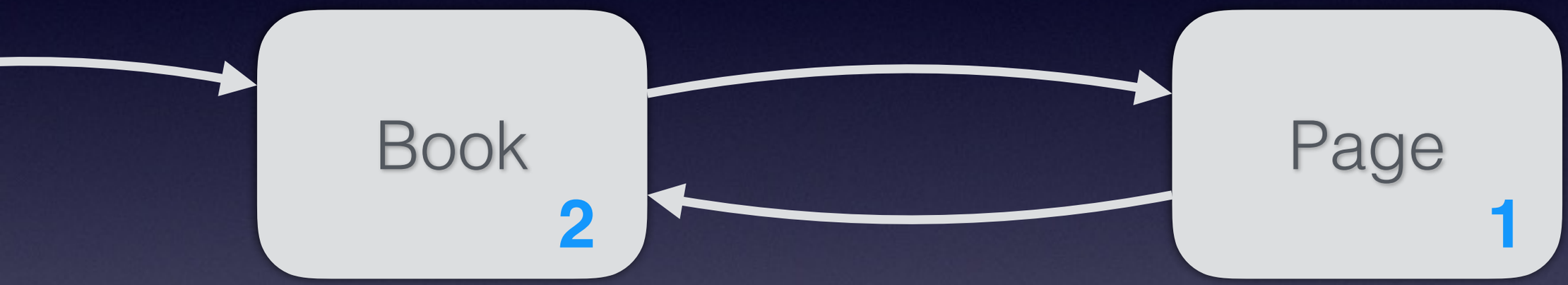
Page

0

ARC versus GC

- unterbrechungsfrei
- zeitnahe Speicherfreigabe
- Zyklen

Zyklen



Zyklen

Lösung

Lösung

- `strong`
- `weak`
- `unowned`

Zyklen

```
class ViewController {
    var button: Button

    func loadView() {
        button.onClick = {
            self.handleClick()
        }
    }

    func handleClick() {
        …
    }
}
```

Zyklen

```
class ViewController {

    var button: Button

    func loadView() {
        button.onClick = {
            [weak self] in
            self?.handleClick()
        }
    }

    func handleClick() {
        …
    }
}
```

Properties

Properties

```
class Klasse {

    let konstante = 42

    var variable = 42

    private(set) var halbPrivat = 42

    var berechneteVariable: Int {
        return 42
    }
}
```

Properties

```
class Klasse {

    var variable: Int {
        get {
            return 42
        }
        set {
            assert(newValue == 42)
        }
    }
}
```

Properties

```
class Klasse {

    var variable = 42 {
        willSet {
            print("will set \(newValue)")
        }
        didSet {
            print("did set \(oldValue)")
        }
    }
}
```

Properties

```
struct Klasse {

    lazy var variable = calculate()
}

func calculate() -> Int {
    // let me think
    // about it for a while
    return 42
}
```

Properties

```
window.position.x = 0
```

set position

set x

Enums

Enums

```
enum Direction {
    case North
    case West
    case East
    case South
}
```

Raw Values

```
enum Direction: Character {
    case North = "N"
    case West = "W"
    case East = "E"
    case South = "S"
}
```

Associated Values

```
enum Shape {
    case Circle(radius: Double)
    case Line(length: Double)
    case Rectangle(width: Double,
                   height: Double)
}

let shape: Shape =
    .Rectangle(width: 10, height: 20)
```

Extensions

Extensions

Erweiterung Funktionalität bestehender Typen

Extensions

```
extension String {
    func reverse() -> String {
        return String(characters.reverse())
    }
}

"foo".reverse()
```

Extensions

```
extension Array where Element: Comparable {
    func maxElement() -> Element? {
        guard let first = first else {
            return nil
        }
        return reduce(first, combine: {
            a, b in max(a, b)
        })
    }
}

[5, 6].maxElement()
```

Protocol Extensions

```
protocol Reversible {
    func reverse() -> Self
}

extension String: Reversible {
    func reverse() -> String {
        return String(characters.reverse())
    }
}

let reversible: Reversible = "foo"
reversible.reverse()
```

Fehlerbehandlung

Objective-C

- Exceptions seit Objective-C 2.0
- Frameworks nicht Exception-safe
- Exceptions für unbehandelbare Fehler
- `NSError`-Parameter für behandelbare Fehler

Swift

- keine Exceptions
- kein `NSError` (seit Swift 2)
- kompatibel mit Objective-C

Errors

```
enum HttpError: Int, ErrorType {
    case FileNotFound = 404
}

throw HttpError.FileNotFound
```

Throwing

```
func load(url: String) throws {
    throw HttpError.FileNotFound
}
```

Catching

```
do {
    try load("https://entwicklertag.de")
} catch HttpError.FileNotFound {
    print("Ups")
}
```

Rethrows

```
extension Array {
    func forEach(
        closure: Element throws -> Double)
        rethrows {

        for element in self {
            try closure(element)
        }
    }
}
```

Sonstiges

- Generics
- Tupel `("answer", 42)`
- Pattern Matching
- funktionale Programmierung
- Operator Overloading

Kuriositäten

defer

```
private var lock = OSSpinLock()
private var unsafeValue: Int

public var value: Int {
    OSSpinLockLock(&lock)
    defer {OSSpinLockUnlock(&lock)}

    return unsafeValue
}
```

Unicode

```
let 👽 = "alien"
```

Custom Operators

```
infix operator <=> {}

func <=>(left: Int, right: Int) -> Int {
    if left < right { return -1 }
    if left > right { return  1 }
    return 0
}

5 <=> 10
```

Access Control

- public: global sichtbar
- internal: innerhalb des Moduls sichtbar
- private: innerhalb der Datei sichtbar

Ausblick

Open Source

- GitHub
- swift-evolution (Mailing-Liste und Proposals)

IBM

- Technologie-Partnerschaft mit Apple
- IBM Swift Sandbox
- IBM Cloud (Bluemix)

Ressourcen

- swift.org
- developer.apple.com/swift
- „The Swift Programming Language“ (eBook)
- swiftlang.ng.bluemix.net