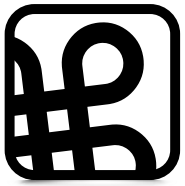


Macoun

Altes und Neues in Swift

Nikolaj Schumacher

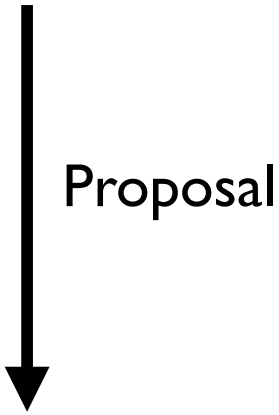
Ablauf

- swift-evolution
- Syntax
- Standardbibliothek
- Zukunft

swift-evolution

Mailing-Listen

- swift-users
- swift-evolution
- swift-evolution-announce
- swift-dev



Versionen

Xcode 8.2	Xcode 8.3	Xcode 9
Swift 2.3	-	-
Swift 3.1	Swift 3.1	Swift 3.2
-	-	Swift 4.0

Versionen

Xcode 8.2	Xcode 8.3	Xcode 9
swift-3.1-branch	swift-3.1-branch	swift-3.2-branch
-	-	swift-4.0-branch
-	-	master

Swift 4 und 3.2

Swift 3.2 unterstützt fast alle Swift 4 Features

Swift 4 und 3.2

- „Swift 3“-Code in Xcode 9 ist **nicht** abwärtskompatibel
- (relevant für Framework-Autoren)

Syntax

Key Paths

- Swift 3
- Nur NSObject/dynamic
- Typ: String

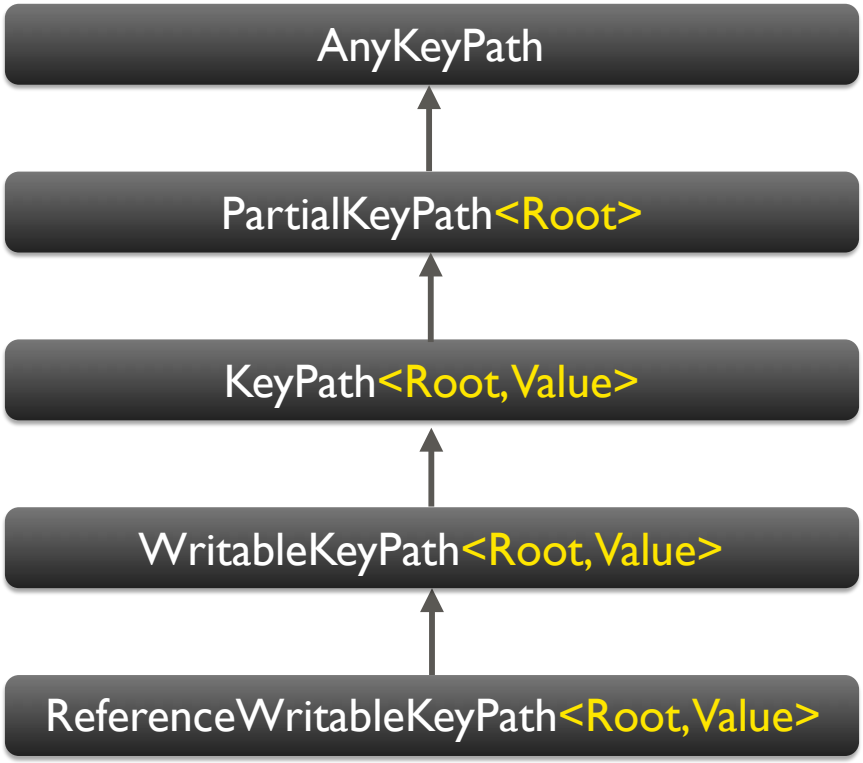
```
class C {
    var property: Int
}

let keyPath: String =
    #keyPath(C.property)
```

Key Paths

- Swift 4
- spezieller Typ

```
class C {
    var property: Int
}

let keyPath: KeyPath<C, Int> =
    \C.property
```

Key Paths


AnyKeyPath
PartialKeyPath<Root>
KeyPath<Root, Value>
WritableKeyPath<Root, Value>
ReferenceWritableKeyPath<Root, Value>

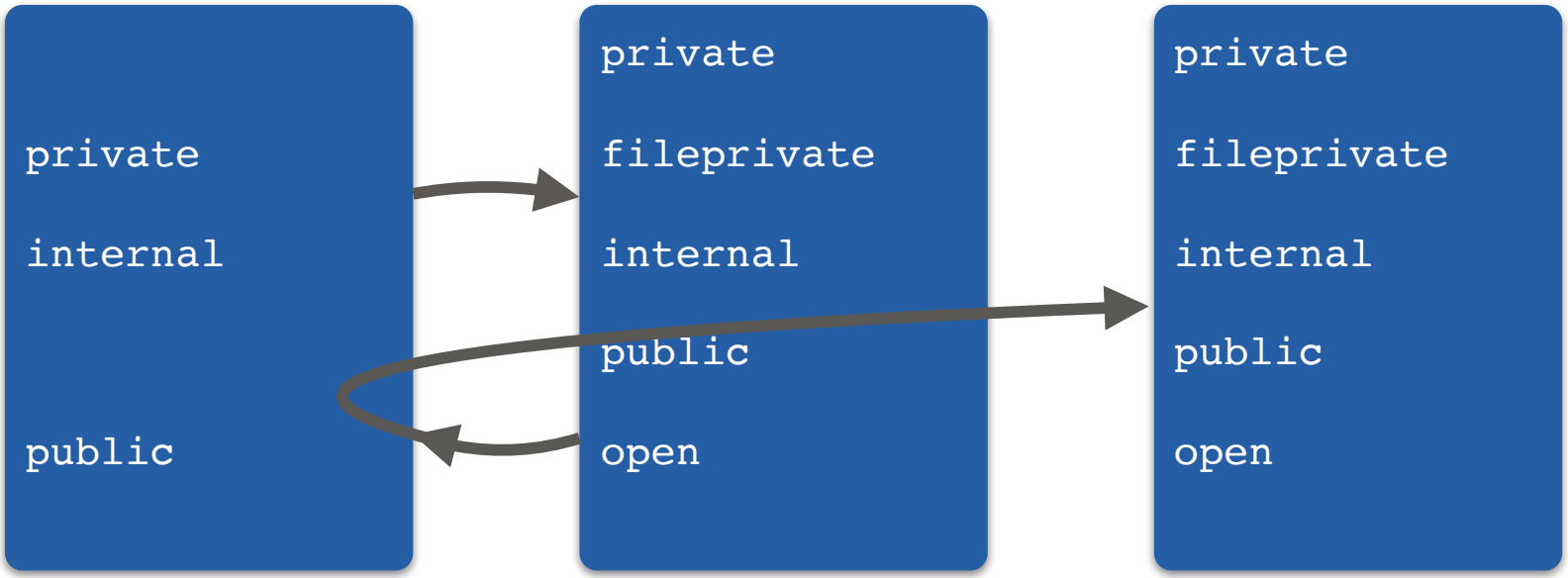
KVO

- Swift 3 und Swift 4-Syntax ✅
- weiterhin nur bei @objc Typen

Key Paths

Demo

private and back again



⚠️ ~~Swift 3.2~~

private and back again

- private-Zugriff aus Extension in derselben Datei erlaubt
- Default-Argumente müssen public sein

⚠️ ~~Swift 3.2~~

Multiline Strings

```
let string =

    "Zeile 1\n"

    + "Zeile 2\n"

    + "Zeile 3\n"
```

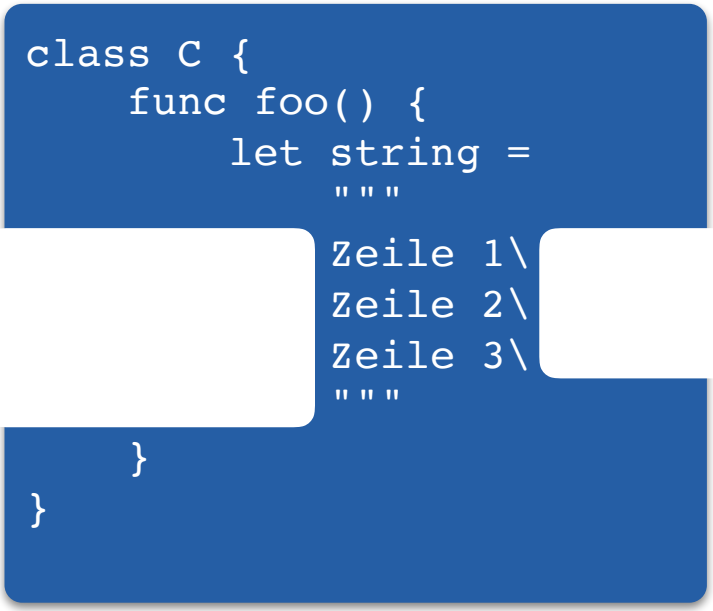
```
let string =

    """
    Zeile 1
    Zeile 2
    Zeile 3
    """
```

Multiline Strings

```
class C {
    func foo() {
        let string =
"""
Zeile 1
Zeile 2
Zeile 3
"""
    }
}
```

```
class C {
    func foo() {
        let string =
            """
            Zeile 1
            Zeile 2
            Zeile 3
            """
    }
}
```

Multiline Strings



Multiline Strings

```
let string =
    """
    "zitiert"
    \(evaluiert)
    """
```

@objc oder nicht

- Attribut für Typen/Methoden
- aus Objective-C sichtbar

⚠️ ~~Swift 3.2~~

@objc oder nicht

- Thunks
- „Adapter“-Code
- vergrößert Binaries

⚠️ ~~Swift 3.2~~

@objc oder nicht

- nicht mehr automatisch für
  - @objc, NSObject-basierte Typen
  - dynamic, optional

- alte Regeln @objcMembers (z.B. XCTestCase)

⚠️ ~~Swift 3.2~~

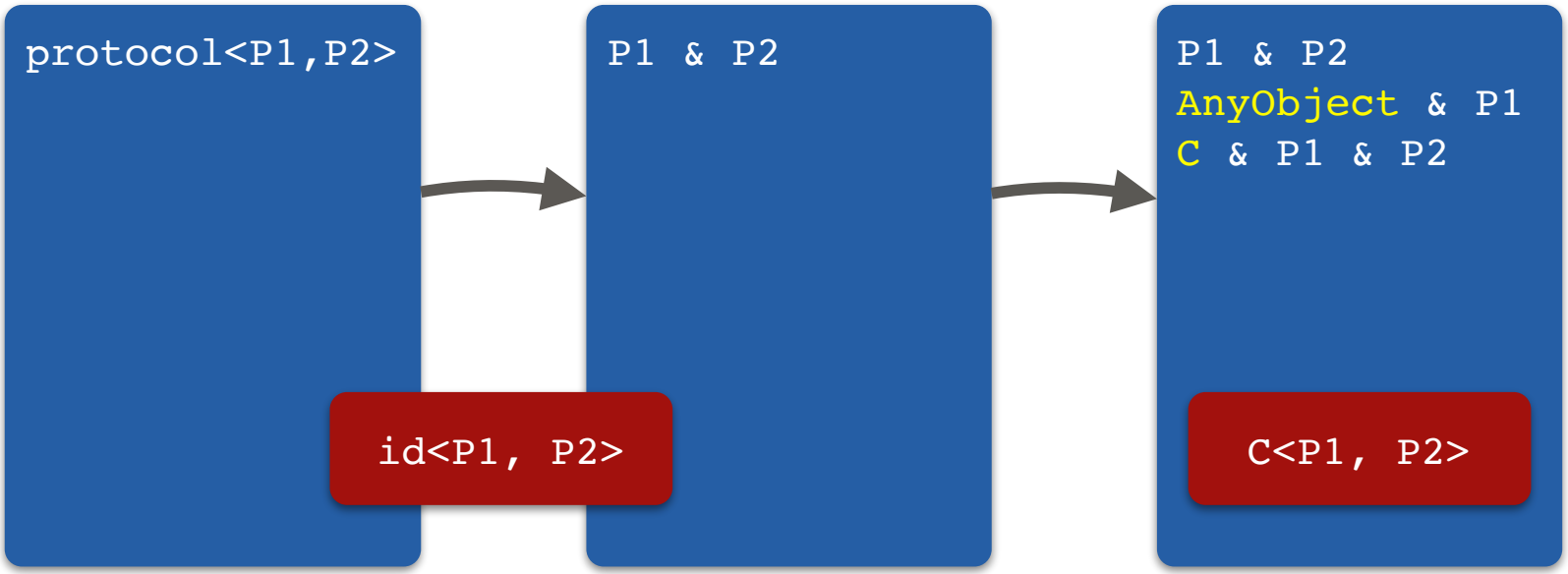
generische Subscripts

```
class C {
    subscript<T: FloatingPoint>(x: T) {
        …
    )
}
```

Associated Types mit Constraints

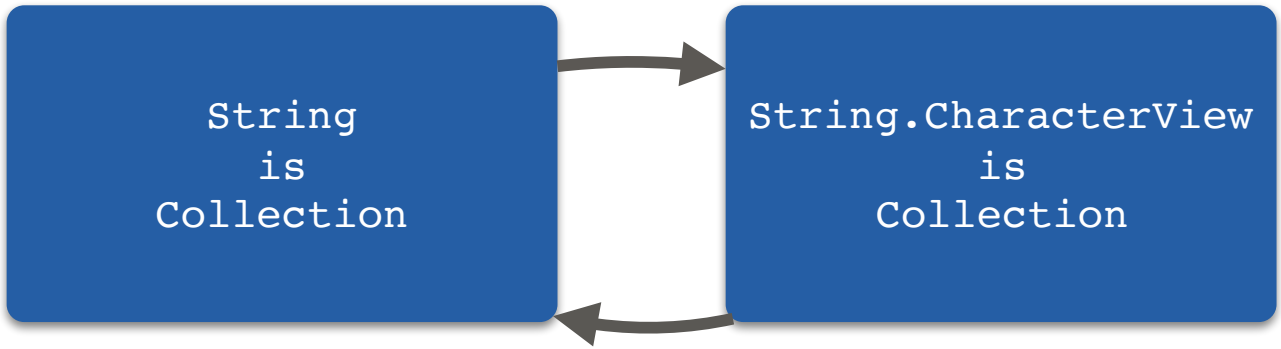
```
class C {
    associatedtype A: Sequence
        where A.Element == Int
}
```

Type Existentials



Standard-Bibliothek

Strings



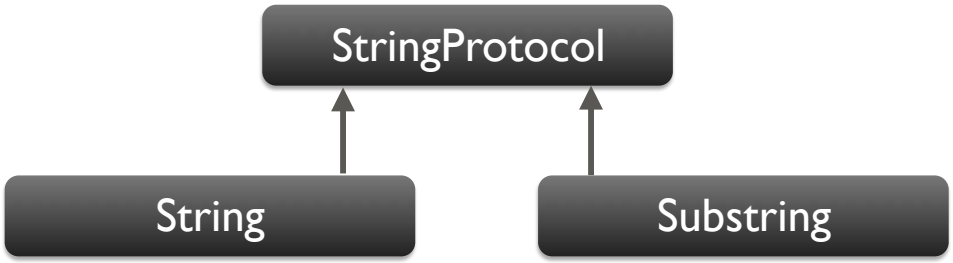
String Views

String	**"abc"**	**"🇩🇪"**
CharacterView	a, b, c	🇩🇪
UTF8View	61, 62, 63	F0, 9F, 87, A9, F0, 9F, 87, AA
UTF16View	61, 62, 63	D83C, DDE9, D83C, DDEA
UnicodeScalarView	a, b, c	🇩, 🇪

Unicode 9

Substring & StringProtocol

- Substring ≈ ArraySlice
- referenziert Basis-String



⚠️ ~~Swift 3.2~~

Collections

- Dictionary Tuple-Initializer
- Dictionary Grouping-Initializer
- Dictionary merge
- Dictionary/Set filter
- Subscript mit Default
- MutableCollection swapAt

Codable

- Umwandlung Objekte in bytes (JSON, Plists, etc.)
- NSCoder
- automatisch

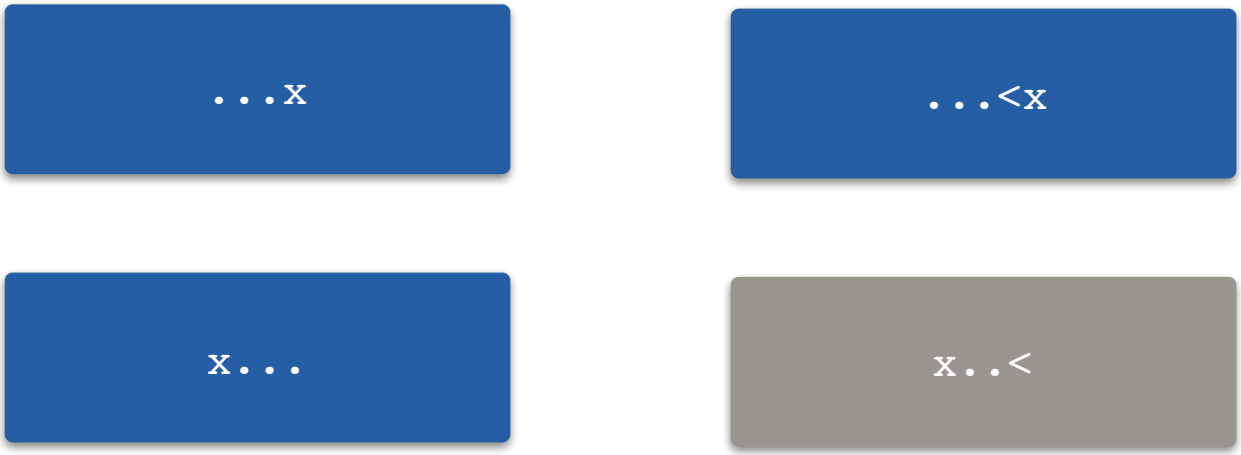
Codable

Demo

Codable

- kompatibel mit NSKeyedArchiver/NSKeyedArchiver
- Cast für NSCoder notwendig
- manuelles Codieren möglich

einseitige Intervalle



einseitige Intervalle

```
let array = ["a", "b", "c", "d", "e"]

array[...2]
 → ["a", "b", "c"]

array[2...]
 → ["c", "d", "e"]
```

einseitige Intervalle

```
switch zahl {
    case ..<0:
        print("negativ")
    case 0:
        print("0")
    default:
        print("positiv")
}
```

Reduce mit inout

- Reduce = Startwert + wiederholte Kombination mit Array-Elementen
- Bsp.: Summe von 1 bis n
  (1...n).reduce(0, +)

Reduce mit inout

- Bei jedem Schritt neuer Wert
- Problematisch z.B. bei Collections
- quadratische Laufzeit

Reduce mit inout

```
["a", "b", "b", "c", "c"].reduce([:]) {

    (result: [String: Int], element) in

    var copy = result
    copy[element] = result[element, default: 0] + 1
    return copy
}
```

Reduce mit inout

```
["a", "b", "b", "c", "c"].reduce(into: [:]) {
    (result: inout [String: Int], element) in
    result[element] = result[element, default: 0] + 1
}
```

Bringt eure Toten raus

ABI-Stabilität

- Swift 3.1 Sourcecode kann mit Swift 4 gemischt werden
- Swift 3.1 binäre Frameworks können **nicht** gemischt werden
- alle Frameworks müssen mitgeliefert werden

➡️ Swift 5

SE-0042

Flattening function types of unapplied method references

```
class C {
    func foo(arg: Int) -> String {}
}

let x: (C) -> (Int) -> String = C.foo
let x: (C, Int) -> String = C.foo

C.foo(c)(25)
C.foo(c, 25)
```

SE-0110

Distinguish between single-tuple and multiple-argument function types

```
let f: (Int, Int) -> Void = { x in
    print(x.0)
    print(x.1)
}

let f: (Int, Int) -> Void = { (x, y) in
    print(x)
    print(y)
}
```

SE-0143

Conditional Conformances

```
extension Array: Equatable
        where Element: Equatable {

    static func ==(lhs: Array<Element>, rhs: Array<Element>) -> Bool {
        …
    }

}
```

Zukunft

Swift 4.1

- automatisches `Equatable` und `Hashable`
- Erweiterungen Unsafe-Typen

Swift 5

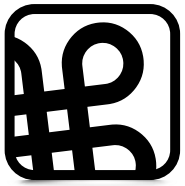
- ABI-Stabilität
- Diskussion: Memory Ownership
- Diskussion: Concurrency

Fragen?

https://github.com/nschum/talks

Vielen Dank

https://github.com/nschum/talks



Macoun