BEUTH HOCHSCHULE FÜR TECHNIK BERLIN
University of Applied Sciences

MAD4

Enumerations,Klassen, Strukturen

Prof. Dr. Dragan Macos

Enumerations

- Typ für eine Menge aufgezählter Werte

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

Werte des Typen CompasPoint

Oder so

```
enum Planet {
    case Mercury, Venus, Earth, Mars, Jupiter, Saturn,
        Uranus, Neptune
}
```

Verwendung

```
var directionToHead = CompassPoint.West
```

Wenn der Typ bekannt ist

```
directionToHead = .East
```

Beispiel

```
directionToHead = .South
switch directionToHead {
case .North:
    println("Lots of planets have a north")
case .South:
    println("Watch out for penguins")
case .East:
    println("Where the sun rises")
case .West:
    println("Where the skies are blue")
}
// prints "Watch out for penguins"
```

Assoziierte Werte

```
enum Barcode {
    case UPCA(Int, Int, Int)
    case QRCode(String)
}
```

Bis 2.953 Zeichen langer String

```
var productBarcode = Barcode.UPCA(8, 85909_51226, 3)
```

```
productBarcode = .QRCode("ABCDEFGHIJKLMNOP")
```

Beispiel

Konstantendefinition. Wert der Konstante: erste Komponente des Wertes UPCA.

Anstatt „let“ kann man auch mit „var“ Variable definieren.

```
switch productBarcode {
case .UPCA(let numberSystem, let identifier, let check):
    println("UPC-A with value of \(numberSystem), \
        (identifier), \(check).")
case .QRCode(let productCode):
    println("QR code with value of \(productCode).")
}
// prints "QR code with value of ABCDEFGHIJKLMNOP."
```

```
case let .UPCA(numberSystem, identifier, check):
```

Wenn alle Assoziierten Werte als Konstanten extrahiert werden → nur ein „let“ ist schicker.

Rohwerte

- Wenn alle assoziierte Werte den selben Typen haben

```
enum ASCIIControlCharacter: Character {
    case Tab = "\t"
    case LineFeed = "\n"
    case CarriageReturn = "\r"
}

enum Planet: Int {
    case Mercury = 1, Venus, Earth, Mars, Jupiter, Saturn,
        Uranus, Neptune
}

let possiblePlanet = Planet.fromRaw(7)
```

Kürzere Form zum Zuweisen der Werte von 1-8

Versucht, das Element mit dem Wert 7 zu finden. Optional.

Beispiel

```
let positionToFind = 9
if let somePlanet = Planet.fromRaw(positionToFind) {
    switch somePlanet {
    case .Earth:
        println("Mostly harmless")
    default:
        println("Not a safe place for humans")
    }
} else {
    println("There isn't a planet at position \
        (positionToFind)")
}
// prints "There isn't a planet at position 9"
```

Klassen, Strukturen

- Definition

```
struct Resolution {
    var width = 0
    var height = 0
}
class VideoMode {
    var resolution = Resolution()
    var interlaced = false
    var frameRate = 0.0
    var name: String?
}
```

Gleich initialisieren.

Neue Typ. Großgeschrieben.

Gleich initialisieren. Aufruf des struct-Initialisierers.

Gleich initialisieren.

- Verwendung/Erzeugung von Instanzen

```
let someResolution = Resolution()
let someVideoMode = VideoMode()
```

- Gezielte Member-Initilaisierung von Strukturen

```
let vga = Resolution(width: 640, height: 480)
```

Strukturen/Enumerations: „Value Types“

- Sie werden immer kopiert.

```
let hd = Resolution(width: 1920, height: 1080)
var cinema = hd
cinema.width = 2048
println("cinema is now \(cinema.width) pixels wide")
// prints "cinema is now 2048 pixels wide"

println("hd is still \(hd.width) pixels wide")
// prints "hd is still 1920 pixels wide"
```

- Genauso die Enumerations

Klassen sind Referenztypen (Reference Types)

- Werden nicht kopiert. Es wird immer der Zeiger auf das Objekt einer Klasse referenziert.

```
let tenEighty = VideoMode()
tenEighty.resolution = hd
tenEighty.interlaced = true
tenEighty.name = "1080i"
tenEighty.frameRate = 25.0
let alsoTenEighty = tenEighty
alsoTenEighty.frameRate = 30.0
println("The frameRate property of tenEighty is now \
        (tenEighty.frameRate)")
// prints "The frameRate property of tenEighty is now
        30.0"
```

- Mit „===“ und „!===“ fragen wir ob es sich um identische Objekte handelt.

Properties

- Variablen und Konstanten, die zu Klassen und Strukturen gehören (Attribute)
- Stored Properties
  - Properties, die bestimmte Werte haben
  - Einfache Konstanten und Variablen innerhalb der Klasse
- Computed Properties
  - Properties, die auf andere Properties zugreifen und bestimmte Berechnungen durchführen.

```
struct FixedLengthRange {
    var firstValue: Int
    let length: Int
}
var rangeOfThreeItems = FixedLengthRange(firstValue: 0,
        length: 3)
// the range represents integer values 0, 1, and 2
rangeOfThreeItems.firstValue = 6
// the range now represents integer values 6, 7, and 8
```

Stored Properties

Lazy stored Properties

- Sie sollen nur dann initialisiert werden, wenn man diese wirklich braucht.
- Sonst ist die Initialisierung zu teuer.

```
class DataImporter {
    /*
    DataImporter is a class to import data from an
        external file.
    The class is assumed to take a non-trivial amount of
        time to initialize.
    */
    var fileName = "data.txt"
    // the DataImporter class would provide data importing
        functionality here
}

class DataManager {
    @lazy var importer = DataImporter()
    var data = String[]()
    // the DataManager class would provide data management
        functionality here
}
```

```
let manager = DataManager()
manager.data += "Some data"
manager.data += "Some more data"
// the DataImporter instance for the importer property has
        not yet been created

println(manager.importer.fileName)
// the DataImporter instance for the importer property has
        now been created
// prints "data.txt"
```

Importer wurde nicht gesetzt.

Erst hier.

Computed Properties

- Klassen, Strukturen und Enumerations können Properties generieren, die keine einfachen Werte besitzen. Ihre Werte werden berechnet.
- Sie stellen getter- und optional setter-Funktionalitäten zur Verfügung

Beispiel

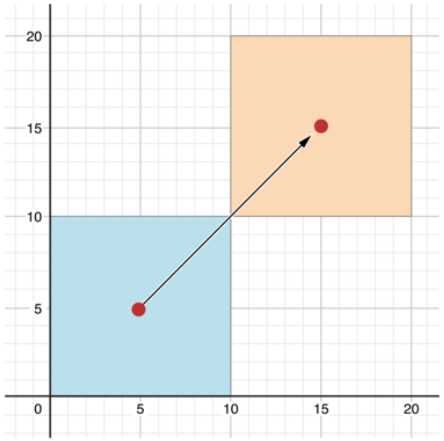
Punkt mit zwei Koordinaten

Größe
(Höhe, Breite)

Rechteck.
Anfang (unten links),
Größe,
Zentrum – Computed Property.

getter, setter

```
struct Point {
    var x = 0.0, y = 0.0
}
struct Size {
    var width = 0.0, height = 0.0
}
struct Rect {
    var origin = Point()
    var size = Size()
    var center: Point {
    get {
        let centerX = origin.x + (size.width / 2)
        let centerY = origin.y + (size.height / 2)
        return Point(x: centerX, y: centerY)
    }
    set(newCenter) {
        origin.x = newCenter.x - (size.width / 2)
        origin.y = newCenter.y - (size.height / 2)
    }
    }
}
var square = Rect(origin: Point(x: 0.0, y: 0.0),
    size: Size(width: 10.0, height: 10.0))
let initialSquareCenter = square.center
square.center = Point(x: 15.0, y: 15.0)
println("square.origin is now at (\(square.origin.x), \
        (square.origin.y))")
// prints "square.origin is now at (10.0, 10.0)"
```



Quellen

Die meisten Sourcecode-Beispiele und die Sprachdefinition der Sprache Swift wurden aus:

Aple Inc. „The Swift Programming Language.“ iBooks. https://itun.es/de/jEUH0.l

genommen.

Eventuelle andere Quellen bzw. eigene Beispiele werden an den entsprechenden Stellen direkt angegeben bzw. gekennzeichnet.