SWIFT 4.2

ENUMERATIONS

WAS SIND ENUMERATIONS?

- “Eigener Datentyp”
- Gruppierung zusammenhängender Werte
- Beginnen mit Großbuchstaben
- Syntax:

```
enum SomeEnumeration {
    // enumeration definition goes here
}
```

DEKLARATION

- Multiline:

```
enum CompassPoint {
    case north
    case east
    case south
    case west
}
```

- Singleline:

```
enum CompassPoint {
    case north, east, south, west
}
```

VERWENDUNG

- Variablen initialisieren:

```
var directionToHead = CompassPoint.west
```

- Wert ändern:

```
directionToHead = .north
```

VERWENDUNG

- Werte abfragen:

```
switch directionToHead {
    case .north :
        print ( “Lots of planets have a north” )
    case .east :
        print ( “Where the sun rises” )
    case .south :
        print ( “Watch out for penguins” )
    case .west :
        print ( “Where the skies are blue” )
}
```

- Nur bestimmte Werte abfragen:

```
switch directionToHead {
    case .north :
        print ( “You’re on the right way” )
    default :
        print ( “Please head north to reach your destination” )
}
```

ÜBER ENUMS ITERIEREN

- Als iterierbar kennzeichnen:

```
enum  Beverage :CaseIterable  {
        case coffee, tea, juice
}
```

- iterieren:

```
for beverage in Beverage.allCases  {
        print ( beverage )
}
```

- allCases.count:

```
let numberOfChoices = Beverage.allCases.count
print ( “\(numberOfChoices) beverages available“)
```

ASSOCIATED VALUES

- Deklaration:

```
enum Barcode {
    case upc(Int, Int, Int, Int)
    case qr(String)
}
```

- Verwendung

```
var productBarcode = Barcode.upc(8, 85909, 51226, 3)

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

ASSOCIATED VALUES

- switch-case:

```
switch  productBarcode  {
        case .upc (let numbersystem, let manufacturer, let product, let check) :
                print ( "UPC: \(numbersystem), \(manufacturer), \(product), \(check)" )
        case .qr (let productCode) :
                print ( "QR: \(productcode)" )
}
```

- Kurzschreibweise:

```
switch  productBarcode  {
        case let .upc (numbersystem, manufacturer, product, check) :
                print ( "UPC: \(numbersystem), \(manufacturer), \(product), \(check)" )
        case let .qr (productCode) :
                print ( "QR: \(productcode)" )
}
```

RAW VALUES

- vordefinierte Werte der Enum-Cases
- nicht veränderbar
- Bsp.:

```
enum ASCIIControlCharacter {
        case tab = "\t"
        case lineFeed = "\n"
        case carriageReturn = "\r"
}
```

IMPLICITLY ASSIGNED RAW VALUES

- Werte der Enum-Cases werden automatisch erzeugt
- Bsp. Nummerieren:

```
enum Planet : Int {
        case mercury = 1, venus, earth, mars, jupiter, saturn, uranus, neptune
}
print ( Planet.earth.rawValue )
// prints “3”
```

- Bsp. String als Wert:

```
enum Planet : String {
        case mercury, venus, earth, mars, jupiter, saturn, uranus, neptune
}
print ( Planet.earth.rawValue )
// prints “earth”
```

IMPLICITLY ASSIGNED RAW VALUES

- Variablen initialisieren:

```
enum  Planet :Int  {
        case mercury = 1, venus, earth, mars, jupiter, saturn, uranus, neptune
}
```

```
if let planetEarth = Planet (rawValue: 3)  {    // returns an Optional of Planet
        print ( planetEarth )
        // prints “Planet.earth”
}
```

RECURSIVE ENUMS

- Enum, die eine weitere Instanz derselben Enum beinhaltet
- Syntax:

```
enum  ArithmeticExpression  {
        case number(Int)
        indirect case  addition(ArithmeticExpression, ArithmeticExpression)
        indirect case  multiplication(ArithmeticExpression, ArithmeticExpression)
}
```

- Kurzschreibweise:

```
indirect enum   ArithmeticExpression  {
        case number(Int)
        case  addition(ArithmeticExpression, ArithmeticExpression)
        case  multiplication(ArithmeticExpression, ArithmeticExpression)
}
```

RECURSIVE ENUMS

- Bsp.:

```
let five = ArithmeticExpression.number(5)
let four = ArithmeticExpression.number(4)

var sum = ArithmeticExpression.addition(five, four)
var product = ArithmeticExpression.multiplication(sum, five)
```

RECURSIVE ENUMS

- In rekursiven Funktionen:

```
func evaluate(_ expression :ArithmeticExpression) -> Int {
        switch  expression {
                case let .number(value):
                        return value
                case let .addition(left, right):
                        return evaluate(left) + evaluate(right)
                case let .multiplication(left, right):
                        return evaluate(left) * evaluate(right)
        }
}

var  sum = ArithmeticExpression.addition(five, four)
var  product = ArithmeticExpression.multiplication(sum, five)

print(evaluate(product))  // prints “45”
```

BEISPIEL

Erstelle eine Enumeration Drink, welche entweder Tee, Kaffee, oder Wasser sein kann.
Jeder Drink hat einen bestimmten fixen Preis (z.b Wasser 0.5€, Tee 1.5€ und Kaffee 2.0€)

Danach erstelle eine zweite Enumeration CashdeskObject, welches entweder eine Double-Zahl oder eine Addition aus 2 CashdesObjects darstellt.

Erstelle nun 2 gekaufte Artikel (Drinks) und schreibe eine Funktion, die diese Drinks anschließend zusammenrechnet.

Gib den Endpreis aus.

SWIFT 4.2

ENUMERATIONS