BEUTH HOCHSCHULE FÜR TECHNIK BERLIN
University of Applied Sciences

MAD5

Protokolle

Prof. Dr. Dragan Macos

Definition

- Eine Menge von Methoden, Properties, Konstanten und anderen Eigenschaften, die eine Teil des Softwarecodes beinhalten muss.
- Syntax

```
protocol SomeProtocol {

    // Protokolldefinition

}
```

Protokoll realisieren

- .. Oder „Protokoll einhalten“

```
struct SomeStructure: FirstProtocol,
        AnotherProtocol {
    // Strukturdefinition
}

class SomeClass: SomeSuperclass,
        FirstProtocol, AnotherProtocol {
    // Klassendefinition
}
```

Diese Struct würde gerne das Protokoll *FirstProtokoll* einhalten

Zuerst die Oberklasse, wenn es welche gibt, danach Protokolle.

Anforderungen eines Protokolls

- Properties (Typ- und Instanz-) mit bestimmten Namen und Typen
- Ein Protokoll sagt nicht, ob ein Property „computed“ oder „stored“ ist.
- Ein Protokoll spezifiziert, ob ein Property einen setter haben muss oder nicht.
- Beispiel

```
protocol SomeProtocol {
    var mustBeSettable: Int { get set }
    var doesNotNeedToBeSettable: Int { get }
}
protocol AnotherProtocol {
    class var someTypeProperty: Int { get set }

}
```

Es muss ein getter und ein setter geben.

Anforderung nach einem Type-Property. Immer mit „class“ trotz den anderen Bezeichnungen für Type-Props bei enum und struct.

Beispiel: Protokoll und Protokoll realisieren

```
protocol FullyNamed {
    var fullName: String { get }
}


struct Person: FullyNamed {
    var fullName: String
}
let john = Person(fullName: "John Appleseed")
// john.fullName is "John Appleseed"
```

Noch ein Beispiel

```
protocol FullyNamed {
    var fullName: String { get }
}
```

???

```
class Starship: FullyNamed {
    var prefix: String?
    var name: String
    init(name: String, prefix: String? = nil)
        {
        self.name = name
        self.prefix = prefix
    }
    var fullName: String {
    return (prefix ? prefix! + " " : "") +
        name
    }
}
var ncc1701 = Starship(name: "Enterprise",
        prefix: "USS")
// ncc1701.fullName is "USS Enterprise"
```

Beispiel

```
protocol RandomNumberGenerator {
    func random() -> Double
}
```

Protokoll

```
class LinearCongruentialGenerator: RandomNumberGenerator {
    var lastRandom = 42.0
    let m = 139968.0
    let a = 3877.0
    let c = 29573.0
    func random() -> Double {
        lastRandom = ((lastRandom * a + c) % m)
        return lastRandom / m
    }
}
let generator = LinearCongruentialGenerator()
println("Here's a random number: \(generator.random())")
// prints "Here's a random number: 0.37464991998171"
println("And another one: \(generator.random())")
// prints "And another one: 0.729023776863283"
```

Diese Klasse möchte gerne das Protokoll einhalten.

.. Und hält das Protokoll hier ein ....

Protokolle als Typen

- Protokolle können als Typen verwendet werden.
- Dies bedeutet „Irgendein Typ, der das Protokoll einhält“.

- Protokollelemente, die nicht zwingend umgesetzt werden sollen, werden mit „@optional“ gekennzeichnet.

Protokolle

- Nachträgliche Protokollkonformität deklarieren durch „extension“

```
protocol TextRepresentable {
    func asText() -> String
}

extension Dice: TextRepresentable {
    func asText() -> String {
        return "A \(sides)-sided dice"
    }
}
```

- Protokolle können (ver)erben

Prüfung der Protokolleinhaltung

- „is“ – liefert true, wenn eine Instanz zu einem Protokoll konform ist.
- „as“ – Forciertes Casting zur Protokollkonformität

Delegation

- Ein Entwurfsmuster (Design Pattern)
- Eine Möglichkeit, Funktionalitäten an einen anderen Typen zu delegieren
- Wenn wir etwas an jemanden delegieren, müssen wir sicher sein, das er das auch kann ☺

Beispiel

Das Protokoll „ErdbebenDelegate“, das das *delegate* *braucht*

<protocol>
ErdbebenDelegate

func ortung() -> String

ErdbebenManager

var delegate:
ErdbebenDelegate

func schickeHelfer()-> ()

Protokoll als Typ

```
print("Hilfe senden nach:" +
self.delegate.ortung() + " ")
```

Das delegate. An dem wird die Funktion „ortung“ aufgerufen, um die Hilfe dorthin zu schicken.

Beispiel

Das Protokoll „ErdbebenDelegate“, das das *delegate* *braucht*

ErdbebenManager

var delegate:
ErdbebenDelegate

func schickeHelfer()-> ()

Das delegate. An dem wird die Funktion „ortung“ aufgerufen, um die Hilfe dorthin zu schicken.

```
print("Hilfe senden nach:" +
self.delegate.ortung() + " ")
```

<protocol>
ErdbebenDelegate

func ortung() -> String

Sie halten das Protokoll „ErdbebenDelegate“ ein.

<class>
SpiessPerson:
ErdbebenDelegate

var name = " ”
var ort = " ”
var steuernummer = 0

```
func ortung() -> String {
    return ort
}
```

<struct>
UebungsgelaendeBW:
ErdbebenDelegate

var gpsKoordinaten =
"0.0.0"

```
func ortung() -> String {
        return gpsKoordinaten
    }
```

<struct>
Hippie:
ErdbebenDelegate

var name = " "
var mq:
MarihuanaQualitaet

```
func ortung() -> String {
        return ”++Nirvana++"
    }
```

Beispiel

ErdbebenManager

var delegate:
ErdbebenDelegate

func schickeHelfer()-> ()

<protocol>
ErdbebenDelegate

func ortung() -> String

```
let erdbebenManager = ErdbebenManagement()
let buerger =  SpiessPerson(name: "WilliSpieß", ort: "Kleingarten", steuernummer: 1)
let hippie = Hippie(name: "Krishna", mq: nil)
let bwGelaendeBrandenburg = UebungsgelaendeBW(gpsKoordinaten: "GPS: <123.234.23>")
erdbebenManager.delegate = buerger
erdbebenManager.schickeHelfer()
erdbebenManager.delegate = hippie
erdbebenManager.schickeHelfer()
erdbebenManager.delegate = bwGelaendeBrandenburg
erdbebenManager.schickeHelfer()
```

<class>
SpiessPerson:
ErdbebenDelegate

var name = " ”
var ort = " ”
var steuernummer = 0

```
func ortung() -> String {
    return ort
}
```

<struct>
UebungsgelaendeBW:
ErdbebenDelegate

var gpsKoordinaten =
"0.0.0"

```
func ortung() -> String {
        return gpsKoordinaten
    }
```

<struct>
Hippie:
ErdbebenDelegate

var name = " "
var mq:
MarihuanaQualitaet

```
func ortung() -> String {
        return ”++Nirvana++"
    }
```

Beispiel

ErdbebenManager

var delegate:
ErdbebenDelegate

func schickeHelfer()-> ()

Console Output

```
Hilfe senden nach:Kleingarten
Hilfe senden nach:+++++Nirvana+++++
Hilfe senden nach:GPS: <123.234.23>
```

<protocol>
ErdbebenDelegate

func ortung() -> String

```
let erdbebenManager = ErdbebenManagement()
let buerger =  SpiessPerson(name: "WilliSpieß", ort: "Kleingarten", steuernummer: 1)
let hippie = Hippie(name: "Krishna", mq: nil)
let bwGelaendeBrandenburg = UebungsgelaendeBW(gpsKoordinaten: "GPS: <123.234.23>")
erdbebenManager.delegate = buerger
erdbebenManager.schickeHelfer()
erdbebenManager.delegate = hippie
erdbebenManager.schickeHelfer()
erdbebenManager.delegate = bwGelaendeBrandenburg
erdbebenManager.schickeHelfer()
```

<class>
SpiessPerson:
ErdbebenDelegate

var name = " ”
var ort = " ”
var steuernummer = 0

```
func ortung() -> String {
    return ort
}
```

<struct>
UebungsgelaendeBW:
ErdbebenDelegate

var gpsKoordinaten =
"0.0.0"

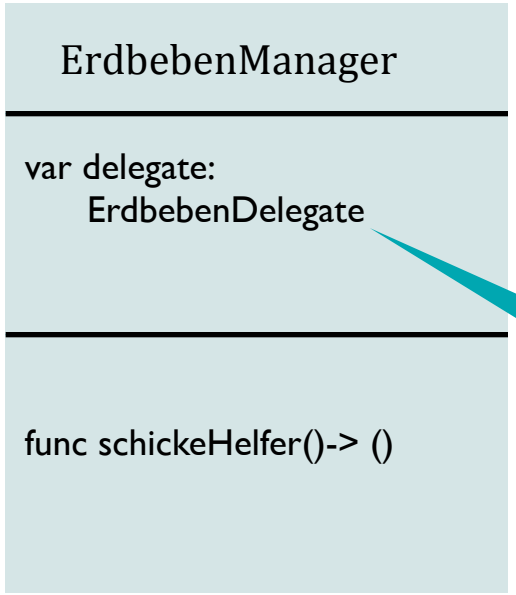
```
func ortung() -> String {
      return gpsKoordinaten
  }
```

<struct>
Hippie:
ErdbebenDelegate

var name = " "
var mq:
MarihuanaQualitaet

```
func ortung() -> String {
      return ”++Nirvana++"
  }
```

Delegate, das Hauptprinzip



Das delegate-Objekt wird unterschiedlich gesetzt. Egal welches Objekt das ist, kann er die Funktion „ortung“ ausführen.
Die „ortung“ oder „Sag wo Du bist“, wird ausgelagert. Delegiert.