UNIVERSITAS SCIENTIARUM SZEGEDIENSIS
UNIVERSITY OF SZEGED
Department of Software Engineering

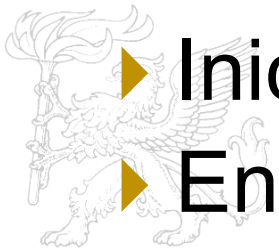
Okosóra, Okostelefon és OkosTV - Apple Swift alapú alkalmazás fejlesztés

Dr. Bilicki Vilmos

Szoftverfejlesztés Tanszék

Swift programozás alapjai II.

- Kollekciók: tömb, szótár, halmaz
- Vezérlési szerkezetek: if, switch, for, for-in, while, repeat-while, guard
- Operátorok
- Függvény, closure, tuple
- Osztályok, objektumok
- Inicializálás
- Enum
- Struct



Swift programozás alapjai III.

- Beágyazott típusok
- Extension
- Protokoll
- Delegate
- Hibakezelés
- Generikus adatszerkezetek
- Access Control (public, private, internal)

Beágyazott típusok

- Gyakran célszerű helyi kiszolgálló osztályokat használni (class/enum/struct)
- Tetszőleges mélységű beágyazás
- Kívülről is el lehet érni a teljes elérési útvonallal

```
struct BlackjackCard {

    // nested Suit enumeration
    enum Suit: Character {
        case spades = "♠", hearts = "♡", diamonds = "◇",
          clubs = "♣"
    }

    // nested Rank enumeration
    enum Rank: Int {
        case two = 2, three, four, five, six, seven, eight,
          nine, ten
        case jack, queen, king, ace
        struct Values {
            let first: Int, second: Int?
        }
        var values: Values {
            switch self {
            case .ace:
                return Values(first: 1, second: 11)
            case .jack, .queen, .king:
                return Values(first: 10, second: nil)
            default:
                return Values(first: self.rawValue, second:
              nil)
            }
        }
    }

    // BlackjackCard properties and methods
    let rank: Rank, suit: Suit
    var description: String {
        var output = "suit is \(suit.rawValue),"
        output += " value is \(rank.values.first)"
        if let second = rank.values.second {
            output += " or \(second)"
        }
        return output
    }
}
```

```
let heartsSymbol = BlackjackCard.Suit.hearts.rawValue

// heartsSymbol is "♡"
```

Extension

- Meglévő típusok bővítése új képességekkel (class/struct/enum/protocol)
- Forráskód nélkül is működik
- Képességek
  - Számított adattagok
  - Példány és típus metódusok
  - Inicializálók
  - Indexek
  - Beágyazott típusok
  - Protokollok

Számított érték példa

```
extension Double {

    var km: Double { return self * 1_000.0 }

    var m: Double { return self }

    var cm: Double { return self / 100.0 }

    var mm: Double { return self / 1_000.0 }

    var ft: Double { return self / 3.28084 }

}

let oneInch = 25.4.mm

print("One inch is \(oneInch) meters")

// Prints "One inch is 0.0254 meters"

let threeFeet = 3.ft

print("Three feet is \(threeFeet) meters")

// Prints "Three feet is 0.914399970739201 meters"
```

Metódusok,…

```
extension Int {
    subscript(digitIndex: Int) -> Int {
        var decimalBase = 1
        for _ in 0..<digitIndex {
            decimalBase *= 10
        }
        return (self / decimalBase) % 10
    }
}
746381295[0]
// returns 5
746381295[1]
// returns 9
746381295[2]
// returns 2
746381295[8]
// returns 7
```

```
extension Int {
    enum Kind {
        case negative, zero, positive
    }
    var kind: Kind {
        switch self {
        case 0:
            return .zero
        case let x where x > 0:
            return .positive
        default:
            return .negative
        }
    }
}
```

```
func printIntegerKinds(_ numbers: [Int]) {
    for number in numbers {
        switch number.kind {
        case .negative:
            print("- ", terminator: "")
        case .zero:
            print("0 ", terminator: "")
        case .positive:
            print("+ ", terminator: "")
        }
    }
    print("")
}
printIntegerKinds([3, 19, -27, 0, -6, 0, 7])
// Prints "+ + - 0 - 0 + "
```

Protokol

- Java – Interface
  - Megadhat adatagot is
- Metódusok, adattagok és egyébb tulajdonságok követelményét rögzíti
- Megvalósíthatja
  - Osztály (Class)
  - Struktúra (Struct)
  - Felsorolás (Enum)
- A típus amely adott protokollnak megfelel az protokoll konform

Tulajdonság követelmények

```
protocol SomeProtocol {

    var mustBeSettable: Int { get set }

    var doesNotNeedToBeSettable: Int { get }

}
```

```
protocol AnotherProtocol {

    static var someTypeProperty: Int { get set }

}
```

Metódus követelmények

- Metódus szintaktika metódus törzs nélkül

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

```
protocol Togglable {
    mutating func toggle()
}
```

```
enum OnOffSwitch: Togglable {
    case off, on
    mutating func toggle() {
        switch self {
        case off:
            self = on
        case on:
            self = off
        }
    }
}
var lightSwitch = OnOffSwitch.off
lightSwitch.toggle()
// lightSwitch is now equal to .on
```

Protokollok mint típusok

- Nem valósít meg konkrét funkcionalitást
- Teljes értékű típus
  - Visszatérési érték
  - Tulajdonság
  - Belső elem (tömb, …)

```
var d6 = Dice(sides: 6, generator: LinearCongruentialGenerator())
for _ in 1...5 {
    print("Random dice roll is \(d6.roll())")
}
// Random dice roll is 3
// Random dice roll is 5
// Random dice roll is 4
// Random dice roll is 5
// Random dice roll is 4
```

```
class Dice {
    let sides: Int
    let generator: RandomNumberGenerator
    init(sides: Int, generator: RandomNumberGenerator) {
        self.sides = sides
        self.generator = generator
    }
    func roll() -> Int {
        return Int(generator.random() * Double(sides)) + 1
    }
}
```

Protokoll öröklődés

- Egy vagy több protokolltól örökölhet

```
protocol PrettyTextRepresentable: TextRepresentable {

    var prettyTextualDescription: String { get }

}
```

```
extension SnakesAndLadders: PrettyTextRepresentable {

    var prettyTextualDescription: String {

        var output = textualDescription + ":\n"

        for index in 1...finalSquare {

            switch board[index] {

            case let ladder where ladder > 0:

                output += "▲ "

            case let snake where snake < 0:

                output += "▼ "

            default:

                output += "○ "

            }

        }

        return output

    }

}
```

Protokoll szűkítés

- class only
- feltételes

```
protocol SomeClassOnlyProtocol: class, SomeInheritedProtocol {

    // class-only protocol definition goes here

}
```

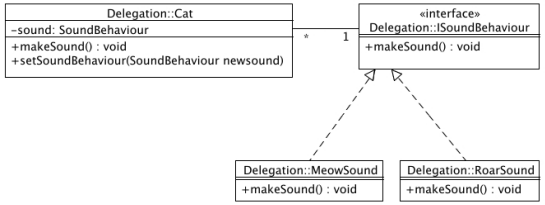
```
@objc protocol CounterDataSource {

    @objc optional func increment(forCount count: Int) -> Int

    @objc optional var fixedIncrement: Int { get }

}
```

Alap implementáció

```
extension PrettyTextRepresentable  {

    var prettyTextualDescription: String {

        return textualDescription

    }

}
```

Delegate

```
protocol DiceGame {
    var dice: Dice { get }
    func play()
}
protocol DiceGameDelegate {
    func gameDidStart(_ game: DiceGame)
    func game(_ game: DiceGame, didStartNewTurnWithDiceRoll diceRoll: Int)
    func gameDidEnd(_ game: DiceGame)
}
```



- Tervezési minta a funkcionalitás kiszervezésére
- Protokoll segítségével valósítható meg
  - Esemény kezelése
  - Adat lekérése
  - …

```
class SnakesAndLadders: DiceGame {
    let finalSquare = 25
    let dice = Dice(sides: 6, generator: LinearCongruentialGenerator())
    var square = 0
    var board: [Int]
    init() {
        board = Array(repeating: 0, count: finalSquare + 1)
        board[03] = +08; board[06] = +11; board[09] = +09; board[10] = +02
        board[14] = -10; board[19] = -11; board[22] = -02; board[24] = -08
    }
    var delegate: DiceGameDelegate?
    func play() {
        square = 0
        delegate?.gameDidStart(self)
        gameLoop: while square != finalSquare {
            let diceRoll = dice.roll()
            delegate?.game(self, didStartNewTurnWithDiceRoll: diceRoll)
            switch square + diceRoll {
            case finalSquare:
                break gameLoop
            case let newSquare where newSquare > finalSquare:
                continue gameLoop
            default:
                square += diceRoll
                square += board[square]
            }
        }
        delegate?.gameDidEnd(self)
    }
}
```

Generikus adatszerkezet

- Flexibilis, újrahasználható függvény, típus

```
func swapTwoInts(_ a: inout Int, _ b: inout Int) {
    let temporaryA = a
    a = b
    b = temporaryA
}
```

```
func swapTwoValues<T>(_ a: inout T, _ b: inout T) {
    let temporaryA = a
    a = b
    b = temporaryA
}
```

Generikus típusok

- Egyedi osztályok, struktúrák, felsorolás típusok

```
struct Stack<Element> {
    var items = [Element]()
    mutating func push(_ item: Element) {
        items.append(item)
    }
    mutating func pop() -> Element {
        return items.removeLast()
    }
}
```

```
extension Stack {
    var topItem: Element? {
        return items.isEmpty ? nil : items[items.count - 1]
    }
}
```

Típus kényszerek

- Protokoll alapon szűkíthető

```
func findIndex<T: Equatable>(of valueToFind: T, in array:[T]) -
          > Int? {
    for (index, value) in array.enumerated() {
        if value == valueToFind {
            return index
        }
    }
    return nil
}
```

```
func allItemsMatch<
    C1: Container, C2: Container
    where C1.ItemType == C2.ItemType, C1.ItemType: Equatable>
    (_ someContainer: C1, _ anotherContainer: C2) -> Bool {

    // check that both containers contain the same number of
          items
    if someContainer.count != anotherContainer.count {
        return false
    }

    // check each pair of items to see if they are equivalent
    for i in 0..<someContainer.count {
        if someContainer[i] != anotherContainer[i] {
            return false
        }
    }

    // all items match, so return true
    return true

}
```

Hozzáférés vezérlés

- Implementációs részletek elrejtése
- Perferált interfész megadása
- Modul és forrás fájl alapú
  - Public – modulon belül és kívül
  - Internal - modulon belül
  - Private – fájlon belül

```
public class SomePublicClass {          // explicitly public
          class

    public var somePublicProperty = 0    // explicitly public
          class member

    var someInternalProperty = 0         // implicitly internal
          class member

    private func somePrivateMethod() {}  // explicitly private
          class member

}


class SomeInternalClass {               // implicitly internal
          class

    var someInternalProperty = 0         // implicitly internal
          class member

    private func somePrivateMethod() {}  // explicitly private
          class member

}


private class SomePrivateClass {        // explicitly private
          class

    var somePrivateProperty = 0          // implicitly private
          class member

    func somePrivateMethod() {}          // implicitly private
          class member

}
```

Hibakezelés

- Könnyűsúlyú hiba kezelés, return-nal egy súlyú
  - Nincs verem visszapörgetés
- Kivétel dobó metódusok
- Ami nem dob tovább azt belül kell lekezelni
- Kivétel gyakran enum

```
func canThrowErrors() throws -> String


func cannotThrowErrors() -> String
```

```
func vend(itemNamed name: String) throws {
    guard let item = inventory[name] else {
        throw VendingMachineError.invalidSelection
    }

    guard item.count > 0 else {
        throw VendingMachineError.outOfStock
    }

    guard item.price <= coinsDeposited else {
        throw VendingMachineError.insufficientFunds(coinsNeeded:
      item.price - coinsDeposited)
    }
```

Hibakezelés

- Hiba elkapás: do-catch

```
var vendingMachine = VendingMachine()
vendingMachine.coinsDeposited = 8
do {
    try buyFavoriteSnack(person: "Alice", vendingMachine: vendingMachine)
} catch VendingMachineError.invalidSelection {
    print("Invalid Selection.")
} catch VendingMachineError.outOfStock {
    print("Out of Stock.")
} catch VendingMachineError.insufficientFunds(let coinsNeeded) {
    print("Insufficient funds. Please insert an additional \(coinsNeeded)
          coins.")
}
// Prints "Insufficient funds. Please insert an additional 2 coins."
```

Hiba - Optional

```
func fetchData() -> Data? {
    if let data = try? fetchDataFromDisk() { return data }
    if let data = try? fetchDataFromServer() { return data }
    return nil
}
let photo = try! loadImage(atPath: "./Resources/John Appleseed.jpg")
```



```
func processFile(filename: String) throws {
    if exists(filename) {
        let file = open(filename)
        defer {
            close(file)
        }
        while let line = try file.readline() {
            // Work with the file.
        }
        // close(file) is called here, at the end of the scope.
    }
}
```

Swift programozás alapjai III.

- Protokoll, extension
- Delegate
- Hibakezelés
- Generikus adatszerkezetek
- Access Control (public, private, internal)

Összefoglalva

- Objektumorientált programozás
- Swift: változók, konstansok, adattípusok kezelése, optional
- Memóriakezelés
- ARC- strong/weak/unowned
- Strong Reference Cycle
- Kollekciók: tömb, szótár, halmaz
- Vezérlési szerkezetek: if, switch, for, for-in, while, repeat-while, guard
- Operátorok
- Függvény, closure, tuple
- Osztályok, objektumok
- Inicializálás
- Enum
- Struct
- Protokoll, extension
- Delegate
- Hibakezelés
- Generikus adatszerkezetek
- Access Control (public, private, internal)