UNIVERSITAS SCIENTIARUM SZEGEDIENSIS
UNIVERSITY OF SZEGED
Department of Software Engineering

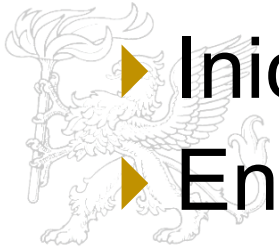
Okosóra, Okostelefon és OkosTV - Apple Swift alapú alkalmazás fejlesztés

Dr. Bilicki Vilmos

Szoftverfejlesztés Tanszék

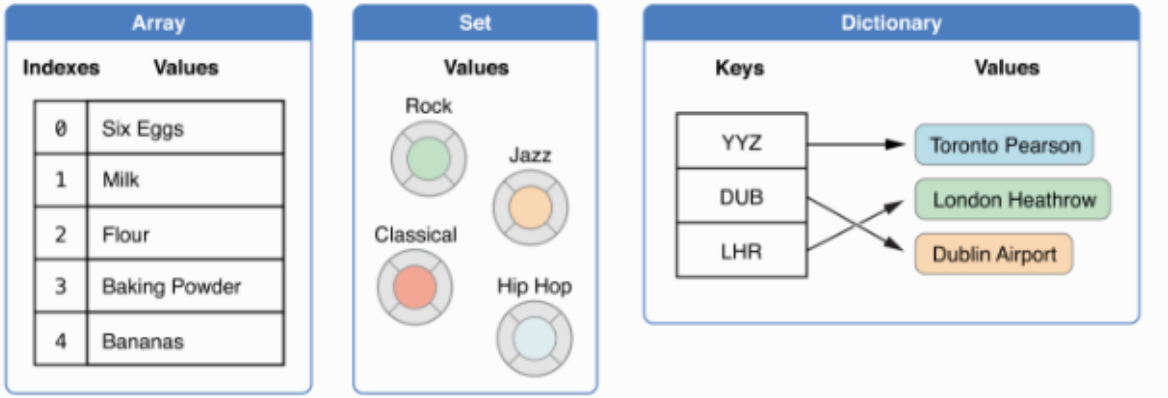
Swift programozás alapjai II.

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



Kollekciók

- Három típus: Tömbök, Halmazok, Szótárak
- Deklarálástól függően változtatható vagy nem (var vs let)
- Érték szerint adódnak át (szálbiztosabb)



Tömbök

- Adott típusú objektumok rendezett listája
- Adott objektum többször is szerpelhet a listában
- NSArray burkolva

```
var someInts = [Int]()
print("someInts is of type [Int] with \(someInts.count)
// Prints "someInts is of type [Int] with 0 items."
```

```
someInts.append(3)
// someInts now contains 1 value of type Int
someInts = []
// someInts is now an empty array, but is still of type [Int]
```

```
var threeDoubles = Array(repeating: 0.0, count: 3)
// threeDoubles is of type [Double], and equals [0.0, 0.0, 0.0]
```

```
var sixDoubles = threeDoubles + anotherThreeDoubles
// sixDoubles is inferred as [Double], and equals [0.0, 0.0, 0.0, 2.5,
        2.5, 2.5]
```

```
var shoppingList: [String] = ["Eggs", "Milk"]
// shoppingList has been initialized with two initial items
```

```
shoppingList[4...6] = ["Bananas", "Apples"]
// shoppingList now contains 6 items
```

```
shoppingList += ["Baking Powder"]
// shoppingList now contains 4 items
shoppingList += ["Chocolate Spread", "Cheese", "Butter"]
// shoppingList now contains 7 items
```

```
for item in shoppingList {
    print(item)
}
```

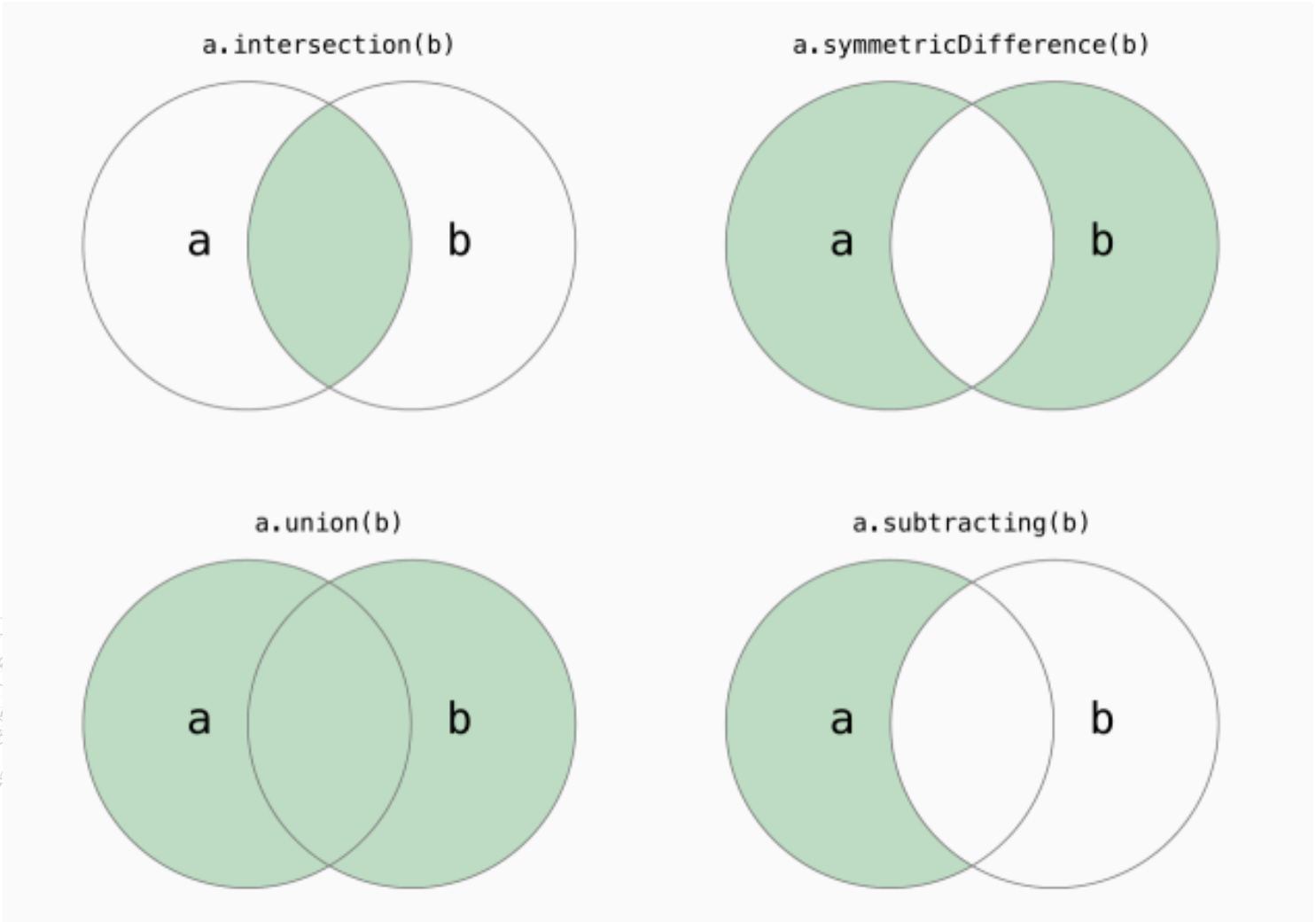
Halmazok

- Adott típus különböző egyedeit tárolja
- NSSet osztály burkolva
- A tárolandó típusnak kivonaltolhatónak kell lennie (hashValue – Hashable protokoll)

```
var letters = Set<Character>()
print("letters is of type Set<Character> with \(letters.count) items.")
letters.insert("a")
// letters now contains 1 value of type Character
letters = []
// letters is now an empty set, but is still of type Set<Character>
var favoriteGenres: Set = ["Rock", "Classical", "Hip hop"]
```

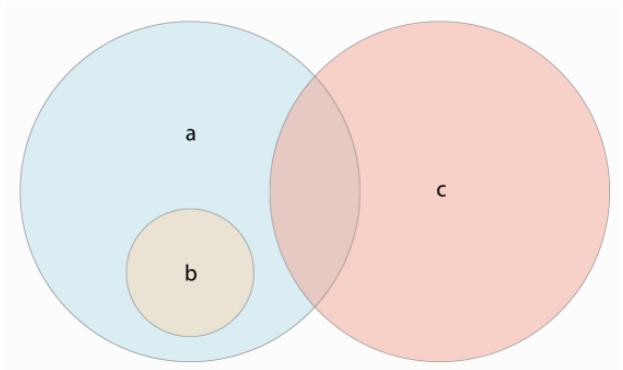
```
for genre in favoriteGenres {
    print("\(genre)")
}
```

Halmaz műveletek



Halmaz műveletek

- == - minden elem azonos
- isSubset(of:) – teljesen tartalmazza
- isSuperset(of:) - teljesen tartalmazza
- isStricSubstet(of:), isStrictSuperset(of:)
- isDisjoint(with:) - van-e közös elem



Szótárak

- Nem rendezett kulcs érték párok tárolója (homogén kulcs, homogén érték)
- NSDictionary burkolva

```
var namesOfIntegers = [Int: String]()
// namesOfIntegers is an empty [Int: String] dictionary
namesOfIntegers[16] = "sixteen"
// namesOfIntegers now contains 1 key-value pair
namesOfIntegers = [:]
// namesOfIntegers is once again an empty dictionary of type [Int:
        String]
var airports: [String: String] = ["YYZ": "Toronto Pearson", "DUB":
        "Dublin"]
var airports = ["YYZ": "Toronto Pearson", "DUB": "Dublin"]
```

```
airports["LHR"] = "London Heathrow"
// the value for "LHR" has been changed to "London Heathrow"
for (airportCode, airportName) in airports {
    print("\(airportCode): \(airportName)")
}
```

Vezérlési szerkezetek

- Java/C vezérlő szerkezetek
  - for-in
  - while
  - do while
  - if
  - switch
    - minden értéket le kell fedni
    - több azonos találat
    - nem kell break, első találat
    - intervallum illesztés
    - rendezett n-es véges lista (tuple)
    - érték kötés
    - where

```
let somePoint = (1, 1)
switch somePoint {
case (0, 0):
    print("(0, 0) is at the origin")
case (_, 0):
    print("(\(somePoint.0), 0) is on the x-axis")
case (0, _):
    print("(0, \(somePoint.1)) is on the y-axis")
case (-2...2, -2...2):
    print("(\(somePoint.0), \(somePoint.1)) is inside the box")
default:
    print("(\(somePoint.0), \(somePoint.1)) is outside of the box")
}
// Prints "(1, 1) is inside the box"
```

```
let approximateCount = 62
let countedThings = "moons orbiting Saturn"
var naturalCount: String
switch approximateCount {
case 0:
    naturalCount = "no"
case 1..<5:
    naturalCount = "a few"
case 5..<12:
    naturalCount = "several"
case 12..<100:
    naturalCount = "dozens of"
case 100..<1000:
    naturalCount = "hundreds of"
default:
    naturalCount = "many"
}
print("There are \(naturalCount) \(countedThings).")
```

```
let anotherPoint = (2, 0)
switch anotherPoint {
case (let x, 0):
    print("on the x-axis with an x value of \(x)")
case (0, let y):
    print("on the y-axis with a y value of \(y)")
case let (x, y):
    print("somewhere else at (\(x), \(y))")
}
```

```
let someCharacter: Character = "e"
switch someCharacter {
case "a", "e", "i", "o", "u":
    print("\(someCharacter) is a vowel")
case "b", "c", "d", "f", "g", "h", "j", "k", "l", "m",
     "n", "p", "q", "r", "s", "t", "v", "w", "x", "y", "z":
    print("\(someCharacter) is a consonant")
default:
    print("\(someCharacter) is not a vowel or a consonant")
}
```

```
let yetAnotherPoint = (1, -1)
switch yetAnotherPoint {
case let (x, y) where x == y:
    print("(\(x), \(y)) is on the line x == y")
case let (x, y) where x == -y:
    print("(\(x), \(y)) is on the line x == -y")
case let (x, y):
    print("(\(x), \(y)) is just some arbitrary point")
}
```

Vezérlés átadás

- Elemei
  - continue – ciklus következő lépésre lépés
  - break – az egész ciklust befejezi (for, … switch)
  - falltrough – switch-ben nem csak az elsőt veszi
  - return
  - thorw
- Címkézett kifejezések
  - Megadhatjuk amire vonatkozik az átadás
- guard - Korai kilépés, adott kód blokk további végrehajtása ettől függ

```
gameLoop: while square != finalSquare {
    diceRoll += 1
    if diceRoll == 7 { diceRoll = 1 }
    switch square + diceRoll {
    case finalSquare:
        // diceRoll will move us to the final square, so the game is
          over
        break gameLoop
    case let newSquare where newSquare > finalSquare:
        // diceRoll will move us beyond the final square, so roll again
        continue gameLoop
    default:
        // this is a valid move, so find out its effect
        square += diceRoll
        square += board[square]
    }
}
print("Game over!")
```

```
func greet(person: [String: String]) {
    guard let name = person["name"] else {
        return
    }

    print("Hello \(name)!")

    guard let location = person["location"] else {
        print("I hope the weather is nice near you.")
        return
    }

    print("I hope the weather is nice in \(location).")
}
```

```
var mathFunction: (Int, Int) -> Int = addTwoInts
```

Függvények

- Első osztályú elemek – típusként kezelhetőek
- Címkézett paraméterek, alapértelmezett értékkel
- Változó hosszúság paraméterlista
- In-out kezelése
- Több visszatérési érték (tuple)

```
func someFunction(parameterWithoutDefault: Int,
        parameterWithDefault: Int = 12) {
    // In the function body, if no arguments are passed
        to the function
    // call, the value of parameterWithDefault is 12.
}
someFunction(parameterWithoutDefault: 3,
        parameterWithDefault: 6) //
        parameterWithDefault is 6
someFunction(parameterWithoutDefault: 4) //
        parameterWithDefault is 12
```

```
func arithmeticMean(_ numbers: Double...) -> Double {
    var total: Double = 0
    for number in numbers {
        total += number
    }
    return total / Double(numbers.count)
}
arithmeticMean(1, 2, 3, 4, 5)
// returns 3.0, which is the arithmetic mean of these
        five numbers
arithmeticMean(3, 8.25, 18.75)
// returns 10.0, which is the arithmetic mean of these
        three numbers
```

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

Closure – Zárvány (Lambda kif.)

- Első osztályú objektum
- Eléri a környezetében definiált változókat, konstansokat, ezekre erős hivatkozást készít
- Általánosítva
  - Függvény
  - Beágyazott függvény
  - Closure

Closure példa

```
func backward(_ s1: String, _ s2: String) ->
          Bool {
    return s1 > s2
}
var reversedNames =
          names.sorted(isOrderedBefore:
          backward)
```

```
reversedNames = names.sorted(isOrderedBefore: {
          (s1: String, s2: String) -> Bool in
          return s1 > s2 } )
```

```
reversedNames = names.sorted(isOrderedBefore: {
          s1, s2 in return s1 > s2 } )
```

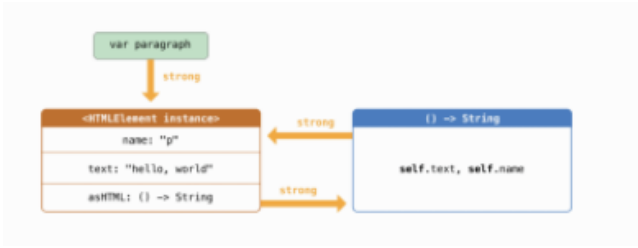
```
reversedNames = names.sorted(isOrderedBefore: {
          s1, s2 in s1 > s2 } )
```

```
reversedNames = names.sorted(isOrderedBefore: {
          $0 > $1 } )
```

```
reversedNames = names.sorted(isOrderedBefore: >)
```

Closure referencia

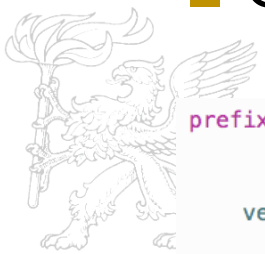
```
let heading = HTMLElement(name: "h1")
let defaultText = "some default text"
heading.asHTML = {
    return "<\(heading.name)>\(heading.text ??
           defaultText)</\(heading.name)>"
}
print(heading.asHTML())
// Prints "<h1>some default text</h1>"
```




```
class HTMLElement {

    let name: String
    let text: String?

    lazy var asHTML: () -> String = {
        [unowned self] in
        if let text = self.text {
            return "<\(self.name)>\(text)</\
           (self.name)>"
        } else {
            return "<\(self.name) />"
        }
    }
```

Operátorok

- Alap, fejlett operátor
- Saját operátor
  - infix (pl.: A+B)
  - prefix (pl.:-A)
  - postfix (pl.: +=)
  - értékadás (pl:=)
- Operátor függvény
  - Operátor kiterjesztése



```
prefix func +++ (vector: inout Vector2D) ->
          Vector2D {
    vector += vector
    return vector
}
```

```
prefix func - (vector: Vector2D) -> Vector2D {
    return Vector2D(x: -vector.x, y: -vector.y)
}
```

```
func += (left: inout Vector2D, right: Vector2D)
          {
    left = left + right
}
```

```
struct Vector2D {
    var x = 0.0, y = 0.0
}
func + (left: Vector2D, right: Vector2D) ->
          Vector2D {
    return Vector2D(x: left.x + right.x, y:
          left.y + right.y)
}
```

```
func == (left: Vector2D, right: Vector2D) ->
          Bool {
    return (left.x == right.x) && (left.y ==
          right.y)
}
func != (left: Vector2D, right: Vector2D) ->
          Bool {
    return !(left == right)
}
```

Tuple (Rendezett n-es)

- Egy értékbe tud több értéket zárni

```
let http404Error = (404, "Not Found")
// http404Error is of type (Int, String), and equals
          (404, "Not Found")
```

```
let (statusCode, statusMessage) = http404Error
print("The status code is \(statusCode)")
// Prints "The status code is 404"
print("The status message is \(statusMessage)")
// Prints "The status message is Not Found"
```

```
let http200Status = (statusCode: 200, description: "OK")
```



```
print("The status code is \(http200Status.statusCode)")
// Prints "The status code is 200"
print("The status message is \
          (http200Status.description)")
// Prints "The status message is OK"
```

Osztályok

- Nem kell osztálynként külön fájl
- Indexelhető
- Inicializáció/Deinicailazáció
- Eseményekezlés
  - willSet
  - didSet
- Referencia szerinti kezelés
- Identitás operátor (referencia)
  - ===
  - !==
- Egyezőség (tartalom szerint)

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

```
class StepCounter {
    var totalSteps: Int = 0 {
        willSet(newTotalSteps) {
            print("About to set totalSteps to \
          (newTotalSteps)")
        }
        didSet {
            if totalSteps > oldValue  {
                print("Added \(totalSteps - oldValue)
          steps")
            }
        }
    }
}
```

Osztályok

- Tuljadonságok
  - Tárolt
    - laza
    - normál
  - Számított
  - Típus adattag (statikus)

```
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
}
```

```
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))
: initialSquareCenter = square.center
iare.center = Point(x: 15.0, y: 15.0)
int("square.origin is now at (\(square.origin.x), \
        (square.origin.y))")
Prints "square.origin is now at (10.0, 10.0)"
```

```
struct SomeStructure {
    static var storedTypeProperty = "Some value."
    static var computedTypeProperty: Int {
        return 1
    }
}
enum SomeEnumeration {
    static var storedTypeProperty = "Some value."
    static var computedTypeProperty: Int {
        return 6
    }
}
class SomeClass {
    static var storedTypeProperty = "Some value."
    static var computedTypeProperty: Int {
        return 27
    }
    class var overrideableComputedTypeProperty: Int {
        return 107
    }
}
```

Metódusok

- Hasonló mint a függvény
- A self-et használhatja
- Struct/Enum esetén a módosítshoz:
  - mutating


```
struct Point {
    var x = 0.0, y = 0.0
    mutating func moveBy(x deltaX: Double, y deltaY:
            Double) {
        x += deltaX
        y += deltaY
    }
}
var somePoint = Point(x: 1.0, y: 1.0)
somePoint.moveBy(x: 2.0, y: 3.0)
print("The point is now at (\(somePoint.x), \
          (somePoint.y))")
// Prints "The point is now at (3.0, 4.0)"
```

Öröklődés

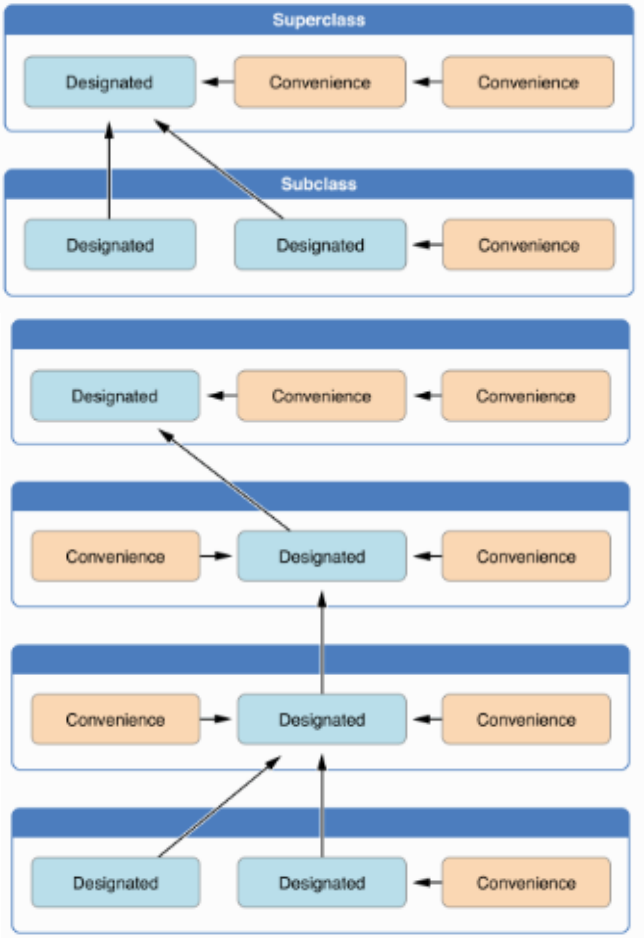
- Osztályokra vonatkozik
- Felülírás
  - override
  - final

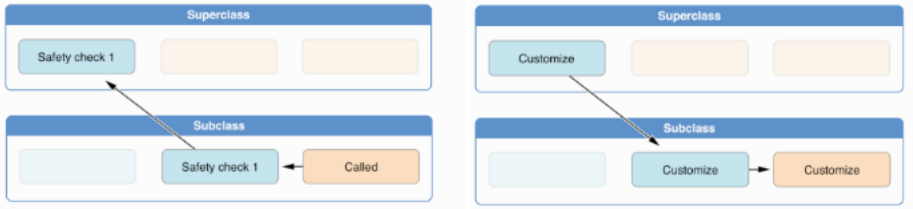
```
class Bicycle: Vehicle {

    var hasBasket = false

}
```

```
class Car: Vehicle {

    var gear = 1

    override var description: String {

        return super.description + " in gear \(gear)"

    }

}
```

Inicializálás

- Az osztály/struktúra/enum felkészítése a használatra
- Nem lehet inicializálatlan adattag
- Testreszabás
  - Paraméterezett
  - Alapértelmezett inicializálók
- Típusai
  - Elsődleges/Kijelölt
  - Kényelmi
- Kétfázisú





Struct

- Ugyanaz mint az osztály csak
  - Érték szerint adódik át
  - Nincs referencia számolás (csak egy referencia lehet)
  - Nincs öröklődés
  - Nincs deinicializálás


```
struct Resolution {
    var width = 0
    var height = 0
}
```

Enum

- Első osztályú elem
- Érték szerint adódik át
- Nyers érték nem csak int lehet

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

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

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

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

```
switch productBarcode {
case .upc(let numberSystem, let
          manufacturer, let product, let
          check):
    print("UPC: \(numberSystem), \
          (manufacturer), \(product), \
          (check).")
case .qrCode(let productCode):
    print("QR code: \(productCode).")
}
```

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