Class

בשיעור זה נכיר

- הגדרת מחלקות ויצירת עצמים מהמחלקות
- מהות Reference Type
- בשיעור הבא נעמוד על מהות Value Type

Class

מחלקה זוהי תבנית ליצירת עצמים.
נשתמש במחלקות על מנת לממש את מודל התוכנית שלנו על פי עקרנות תכנות מונחה עצמים והמודל MVC

הגדרות:
1. עצם הנוצר ממחלקה, מוקצה לו זיכרון.

2. כל המשתנים שאינם אופציונאליים מחוייבים באתחול.

3. אתחול יכול להעשות בהצהרה או במתודת אתחול.

Class

מסקנה מהנ״ל:

יצירת עצם גוררת יצירת ערכים ויתכן אף עצמים ביניהם

המנעו מיצירת משתנים ואובייקטים אשר אינכם צריכים.
בעת עבודה עם מבנים רקורסיביים כדוגמת עץ בינארי, הימנעו מיצירה רקורסיבית.

הנה דוגמא לאיך כן ליצור Node בעץ בינארי.

נתחיל מהבסיס...

```
class Node{
    var left : Node?
    var right : Node?
    var value : Int
    init(){
        value = 0
    }
}
```

Class

עלינו לייצג סטודנט במכללה.
מבנה הנתונים הוא כדלהלן:

תלמיד מורכב מנתוניו האישיים (ת.ז. , שם, כתובת וכו׳) ומערך של השיגיו.

הישג מורכב מ: קורס, סמסטר, מבחן גמר.

קורס מורכב מ: מספר מזהה, שם, נקודות זכות
סמסטר מורכב מ: שנה, סוג (סוג הוא ערך enum)
מבחן גמר מורכ ב: ציון והערה (מחרוזת)

Class

מחלקת קורס

```
class Course{
    var recordID : Int = 0
    var name : String = ""
    var points : Int = 0
}
```

כל המשתנים של אובייקט מהמחלקה קורס מאותחלים בצורה ברירת מחדל שהגדרנו.

נראה יצירה של אובייקטים מהמחלקה קורס

```
var course1 = Course()

var course2 : Course = Course()

var course3 : Course?
course3 = Course()
```

```
var array : Array <Course> = [course1,course2,course3!]
```

Class

מחלקת קורס
שימוש במשתנים לקריאה ולכתיבה נעשה באמצעות dot-notation
או בעברית, נקודה

```
course1.recordID = 1
course1.name = "מיון"
course1.points = 1

course3!.recordID = 3
course3!.name = "קורס אייפון"
course3!.points = 4

course2.recordID = course3!.recordID - course1.recordID
```

Class

מחלקת מבחן גמר

```
class FinalExam{
    var grade : Int8 = 0
    var comment : String?
}
```

```
let finalExam = FinalExam()
finalExam.grade = 99
finalExam.comment = "Only G-d is perfect"

print("finalExam grade:\(finalExam.grade)")

if let comment = finalExam.comment{
    println(" comment:\(comment)")
} else {
    println("")
}
```

Class

מחלקת סמסטר

```
enum SemesterType{
    case Fall
    case Winter
    case Summer

    func getSemeterTypeLetter() -> Character{
        switch self{
        case .Fall:
            return "א"
        case .Winter:
            return "ב"
        case .Summer:
            return "ג"
        }
    }
}
```

```
class Semester{
    var year : Int = 2014
    var type : SemesterType = SemesterType.Fall
}
```

```
var semester = Semester()
semester.year = 2013
semester.type = .Summer

println("semester \(semester.year)\(semester.type.getSemeterTypeLetter())")
```

Class

מחלקת הישג

```
class StudentProgress{
    var course : Course = Course()
    var semester : Semester = Semester()
    var finalExam : FinalExam?
}

var prog = StudentProgress()

prog.course.points = 5

var finalExam = FinalExam()
prog.finalExam = finalExam
finalExam.grade = 80

let grade = prog.finalExam!.grade
```

Class

ולבסוף, מחלקת סטודנט

```
class Student{
    var firstName : String = ""
    var lastName : String = ""
    var socialSecurityNumber : String = ""
    var phoneNumber : String = ""
    var recordID : String = ""

    var progressRecords : Array <StudentProgress> = []
}
```

Reference Value

כל האובייקטים שיצרנו הינם reference type

כלומר כאשר נכתוב `var student = Student()`

student הוא רק התייחסות לעצם שנוצר מהמחלקה Student

איך זה משפיע?

ראשית נשים לב לשורה הבאה `var student2 = student`

האם student2 הוא העתק של student או שהוא רק התייחסות נוספת לאותו עצם שאליו מתייחס student?

מכיוון שעצמים הנוצרים ממחלקות הינם reference type, התשובה היא שמדובר בהתייחסות נוספת לאותו עצם

Reference Value

אשר על כן שינוי ערך ב student2 משנה את הערך גם ב student

```
student2.firstName = "David"
let fName = student.firstName
```

כדי לבדוק אם שני משתנים או קבועים מתייחסים לאותו עצם נשתמש ב
=== שנקרא is-identical

```
if student === student2{
    println("identical")
}
```

אולם כדי לבדוק אם שני משתנים שאינם בהכרח אותו עצם, זהים בתוכנם נשתמש ב
== שנקרא is-equal

```
var student3 = Student()

if student3 == student2{
    println("equal")
}
```

Reference Value

תכונות נוספות האופייניות למחלקה ולעצמים

הגדרת משתנים מאחסני ערכים
הגדרת מתודות על מחלקה
הגדרת מתודות על עצמים
אפשרות להגדיר גישת sub-script כמו למערך ו\או מילון
הגדרת מתודות אתחול
הרחבת מחלקה
הצהרת ומימוש פרוטוקול
ירושה
Casting
אי-אתחול, כלומר שחרור עצם

Properties

כפי שראינו ניתן להגדיר משתנים לעצם. (אגב, גם ל enum)
נעמוד על עניין זה לעומקו.

משתנה של עצם, נקרא לו מכאן והלאה property
יכול להיות משתנה או קבוע (var או let)

```
class MyClsas{
    var aVar : Int = 3
    let aConstant : String = "string"
}
```

Properties

אתחול לא מיידי

ניתן להגדיר property שיאותחל רק בעת שתתקבל קריאה אליו

הדבר שימושי עבור משתנים שזמן הייצור שלהם ארוך ו\או כבד
וכן אובייקטים שאין צורך בזמינות מיידית שלהם אם בכלל

אובייקט כזה יקרא lazy ויסומן באמצעות @lazy

נראה דוגמא בשקופית הבאה

Properties

```
class Node{
    @lazy var leftNode : Node = Node()
    @lazy var rightNode : Node = Node()
    var value : Int = 0
    var doesStoreValue : Bool = false
}

var root : Node = Node()
root.leftNode.value = 1
```

lazy

```
class BackedUpInfo{
    var info : Dictionary <String,String>
    init(){
        //Gather backedup data into info
        info = Dictionary <String,String> ()
    }
}

class User{
    var fullName : String = ""
    @lazy var backedUpInfo : BackedUpInfo = BackedUpInfo()
}
```

Properties

משתנים מחושבים.
ניתן להגדיר properties כך שערכם יחושב בעת ביצוע פעולת ה get
וכן ניתן להגדיר שבעת פעולת ה set יתבצעו חישובים נוספים.

נראה דוגמא:

```
class Square{
    var size : Float = 10
    var scose : Float {
    get {
        return powf(size, 2)
    }
    set (newScope){
        size = sqrtf(newScope)
    }
    }
}
```

הערה: במקרה זה קראנו לערך החדש בשם newScope
במידה ולא היינו מגדירים דבר, שמו היה newValue

Properties

```
class Person{
    //read-only property
    var socialSecurityNumber : String{
    get{
        return self.socialSecurityNumber
    }
    }

    var firstName : String = "first"
    var lastName : String = "last"

    //A computed read-only property
    var fullName : String{
    get {
        return firstName + " " + lastName
    }
    }

}
```

משתנים לקריאה בלבד.
לעיתים, נרצה להגדיר תכונה
שלא ניתנת לשינוי חיצוני - readonly

להלן דוגמא

```
24
25 var person = Person()
26 let fullname = person.fullName
27 person.fullName = "test"
28
29 person.socialSecurityNumber = "123"
30
```

Properties

ניטור חיי ה property
ניתן לדעת את שני האירועים הבאים:

1. לפני ביצוע הצבה

2. אחרי ביצוע הצבה

```
class Car{
    var licenseNumber : String = "00-000-00"{
    willSet (newLicenseNumber){
        println("about to set from \(licenseNumber) to \(newLicenseNumber)")
    }

    didSet{
        println("license number was \(oldValue) and now its \(licenseNumber)")
    }
    }
}
```

Properties

עד כה ראינו משתנים שהם ספציפיים לעצם.
מה לגבי גלובליים?
אמנם, אפשר תמיד לשים משתנה מחוץ להצהרה על מחלקה.
אבל אז הוא גלובלי לאו דווקא בהיקף שאנו רוצים.

בפיתוח מונחה עצמים עולה לעיתים הצורך במשתנה גלובלי ברמת מחלקה ספציפית.

כאשר נגדיר משתנה גלובלי למחלקה נשתמש במילה השמורה class
וכאשר נגדיר משתנה גלובלי למבנה או ל enum נשתמש במילה השמורה static

Properties

```
class Square{
    class var maxSize : Double{
        get{
        if (self.maxSize <= 0){
        self.maxSize = 10
        }
        return self.maxSize
        }
        set{
            self.maxSize = newValue
    }
    }

    var size : Double = 10
}
```

להלן דוגמת מימוש במחלקה וב enum

```
enum AreaCode : Int{
    static var localPrefixDigit : Int{
    get{
    return 0
    }
    }

    case Jerusalem = 2, Center, North
}
```

Properties

דוגמא נוספת

```
class AirPlane{
    class var maxHeight : Double{
        get{
            return 1000
    }
    }

    var height : Double = 0{
    didSet {
        if height > AirPlane.maxHeight{
            //illegal height
            self.height = AirPlane.maxHeight
        }
    }
    }
}

var f16 = AirPlane()
f16.height = 1001
let currentHeight = f16.height
```

סיכום

לסיכום

- למדנו יצירת מחלקות ועצמים
- למדנו הגדרת משתנים של עצמים
- למדנו הגדרת משתנים של מחלקות
- למדנו את מהות reference types

סיכום

שאלות?