







**UNIVERSITY**

ЛЕКЦІЯ 8

# “Перерахування. Структури”

8

# Перерахування. Структури

Перерахування

Структури

Властивості

Ініціалізатори

Методи

Індексатори

Розширення

# Перерахування. Структури

## Перерахування

Структури

Властивості

Ініціалізатори

Методи

Індексатори

Розширення

# Перерахування

Перерахування - це тип, що представляє *перерахований* набір споріднених *значень*.  
*Enumerations* у Swift представляють значення *будь-якого* типу.

Це тип за значення.

Swift надає синтаксис опису перерахування наступного вигляду:

```
accessLevel enum EnumTypeName [: TypeOfElements] {  
    case Element1, Element2 [= rawValue ], ..., ElementA [= rawValue ]  
    case ElementB [= rawValue ]  
    case ElementC [= rawValue ]  
}
```

# Перерахування

Swift надає синтаксис опису сутності типу перерахування наступного вигляду:

```
let someConstantEntity [: EnumTypeName] [ = [EnumTypeName].Element ]  
var someConstantEntity [: EnumTypeName] [ = [EnumTypeName].Element ]
```



# Перерахування



```
3 import UIKit
4
5 enum Values: Int {
6     case one
7     case two
8     case three
9 }
10
11 let number = Values.one
12
```

one

# Перерахування

```
11 typedef NSInteger, Values) {  
12     One,  
13     Two,  
14     Three,  
15 };  
16  
17  
18 int main(int argc, const char * argv[]) {  
19     @autoreleasepool {  
20  
21         Values value = One;  
22  
23         // do stuff here  
24     }  
25     return 0;  
26 }  
27
```

# Перерахування: асоціативні значення

Swift надає можливість опису перерахування, елементами яких є певні асоціативні сутності.

# Перерахування: асоціативні значення



```
13 enum Color {
14     case RGB(Int, Int, Int)
15     case RGBA(Int, Int, Int, Int)
16 }
17
18 let rgbColor = Color.RGB(10, 10, 10)
19 let rgbaColor = Color.RGBA(10, 10, 10, 10)
20
21 switch rgbColor {
22     case .RGBA(let red, let green, let blue, let alpha):
23         print("\(red) - \(green) - \(blue)- \(alpha)")
24     case .RGB(let red, let green, let blue):
25         print("\(red) - \(green) - \(blue)")
26 }
```

# Перерахування: вихідні значення

Кожний елемент перерахування має вихідне значення, яке представляє деяке значення того чи іншого типу, на основі якого можна створити елемент перерахування.

Доступ до вихідного значення здійснюється за допомогою конструкції наступного вигляду:

`enumEntity.rawValue`

*Raw value* доступне лише для читання.

Перерахування з асоціативними значеннями не мають вихідних значень.

# Перерахування: вихідні значення



```
5 enum Values: Int {  
6     case one  
7     case two  
8     case three  
9 }  
10  
11 let number = Values.one  
12  
13 number.rawValue
```

one	<input type="checkbox"/>
0	<input type="checkbox"/>

# Перерахування: вихідні значення

```
11 typedef NS_ENUM(NSInteger, Values) {
12     One,
13     Two,
14     Three,
15 };
16
17
18 int main(int argc, const char * argv[]) {
19     @autoreleasepool {
20
21         Values value = One;
22
23         int rawValue = value;
24         NSLog(@"%i", rawValue);
25
26         // do stuff here
27     }
28     return 0;
29 }
30 |
```

2018-03-04 17:59:32.641796+0200 test[12094:3194801] 0  
Program ended with exit code: 0

# Перерахування: робота з перерахуваннями

Робота з сутностями типу перерахувань здебільшого відбувається у поєднанні з умовними виразами та перемиканнями.



# Перерахування: рекурсивні



Перерахування можуть містити елементи, що є перерахуваннями.

Swift надає синтаксис опису рекурсивного перерахування наступного вигляду:

```
accessLevel/ indirect enum EnumTypeName {  
    indirect case Element1, Element2, ..., ElementA  
    indirect case ElementB  
    indirect case ElementC  
}
```

Рекурсивними перерахуваннями можуть бути перерахування лише з асоціативними елементами.

# Перерахування: рекурсивні



```
21 indirect enum ColorType {  
22     case RGB(Color)  
23     case CMYK(Int, Int, Int, Int)  
24 }  
25  
    let someColor = ColorType.RGB(Color.RGB(10, 10, 10))
```

# Перерахування: Optional

Optional є перерахуванням наступного вигляду:

```
public enum Optional<Wrapped> : ExpressibleByNilLiteral {  
    case None  
    case Some(Wrapped)  
}
```

Рівноцінними описом сутності Optional типу є:

Wrapped?

# Перерахування: Optional



```
40 let someValue = 5
41 let someOptional: Int? = nil
42
43 switch someOptional {
44 case .some(someValue):
45     print("the value is \(someValue)")
46 case .some(let val):
47     print("the value is \(val)")
48 default:
49     print("nil")
50 }
```

5  
nil  
  
"nil\n"

# Перерахування. Структури

## Перерахування

Структури

Властивості

Ініціалізатори

Методи

Індексатори

Розширення

# Перерахування. Структури

Перерахування

**Структури**

Властивості

Ініціалізатори

Методи

Індексатори

Розширення

# Структури

Структура - це *іменований* тип загального призначення, що представляє гнучку конструкцію, яка містить дані деяких типів та визначає поведінку роботи над ними, а також служить у якості блока коду при побудові програм.

Структура - це *тип за значення*.

Елементи структури, які представляють дані, називається *властивостями* структури.

Елементи структури, які представляють інтерфейс для роботи з даними, що містяться в структурі, називаються *методами* структури.

# Структури

Усі базові типи за значенням у мові програмування Swift є *структурами*.

До них належать:

Int, UInt (Int8, Int16, Int32, Int64, UInt8, UInt16, UInt32, UInt64)

Float, Double (Float32, Float64, Float80)

Bool

Character

String

Array (CollectionOfOne, EmptyCollection)

Set

Dictionary



# Структури



Swift визначає наступний синтаксис опису структури:

```
accessLevel struct StructureName: Protocol1, ... {  
    // let or var properties declaration  
    // init methods definition  
    // func methods definition  
    // subscript methods definition  
}
```

*accessLevel* - рівень доступу (public, internal, private)

# Структури



Objective-C визначає наступний синтаксис опису структури:

```
struct StructureName {  
    //property  
}
```

# Структури: екземпляр



Структура - це тип даних, а сутність типу структури називається її *екземпляром* або об'єктом.

Swift надає наступний синтаксис оголошення та створення екземпляра структури.

```
let constantInstanceName: StructureName = StructureName(...)
```

```
var constantInstanceName: StructureName = StructureName(...)
```

# Структури: екземпляр

Структура - це тип даних, а сутність типу структури називається її *екземпляром* або об'єктом.

Swift надає наступний синтаксис оголошення та створення екземпляра структури.

```
struct StructureName name;
```

# Структури



```
52 //STRUCT
53
54 struct Date {
55     var day: Int
56     var month: Int
57     var year: Int
58 }
59
60 let date = Date(day: 12, month: 12, year: 2018)
61 print(date)
```

Date  
"Date(day: 12, month:..."

# Структури

OBJ-C

```
17 struct DateValue
18 {
19     int day;
20     int month;
21     int year;
22 };
23
24 int main(int argc, const char * argv[]) {
25     @autoreleasepool {
26
27         struct DateValue newData;
28         newData.day = 22;
29         newData.month = 12;
30         newData.year = 2018;
31
32         NSLog(@"%i, %i, %i", newData.day, newData.month, newData.year);
33
34         Values value = One;
35
36         int rawValue = value;
37         NSLog(@"%i", rawValue);
38
39     }
40     return 0;
41 }
```

```
2018-03-04 18:21:52.821718+0200 test[12713:3230728] 0
2018-03-04 18:21:52.822092+0200 test[12713:3230728] 22, 12, 2018
Program ended with exit code: 0
```

# Перерахування. Структури

Перерахування

**Структури**

Властивості

Ініціалізатори

Методи

Індексатори

Розширення

# Перерахування. Структури

Перерахування

Структури

**Властивості**

Ініціалізатори

Методи

Індексатори

Розширення



# Властивості



Властивості - це сутності, які асоціюють значення із структурою або перерахуванням. Swift поділяє властивості на зберігаючі (*stored*) та обчислювані (*computed*).

Властивості, які зв'язують значення із екземпляром структури або перерахування, називається властивостями екземпляра (*instance properties*).

Властивості, які зв'язують значення із типом структури або перерахування, називається властивостями типу (*type properties*).

# Властивості: зберігаючі (stored)



Зберігаючі властивості зберігають сталі або змінні значення екземляру структури або її типу. Перерахування *не зберігати* значень.

Синтаксично зберігаючі властивості представлені константами (let) та змінними (var).

```
let constantPropertyName [: TypeOfValue] [ = defaultValue]
```

```
var variablePropertyName [: TypeOfValue] [ = defaultValue]
```

# Властивості: обчислювані (computed)



Обчислювані властивості обчислюють, а не зберігають, значення у структурах або перерахуваннях.

Синтаксично обчислювані властивості представлені лише змінними (var).

# Властивості: обчислювані (computed)



Обчислювана властивість складається з метода *селектора* (*getter*) та необов'язкового метода мутатора або *модифікатора* (*setter*).

```
var propertyName: TypeOfValue {  
    get {  
        return propertyName  
    }  
  
    set(newValue) {  
        propertyName = newValue  
    }  
}
```

# Властивості: обчислювані (computed)



```
54 struct Date {  
55     var day: Int  
56     var month: Int  
57     var year: Int  
58  
59     var currentDayDescription :String {  
60         get {  
61             return "Current day of month - \(day)"  
62         }  
63     }  
64 }  
65  
66 let date = Date(day: 12, month: 12, year: 2018)  
67 print(date)  
68  
69 date.currentDayDescription
```

"Current day of month..."

Date

"Date(day: 12, month:..."

"Current day of mont..."

# Властивості: типу



Властивості типу - це властивості, описані на рівні типу таким чином, що зміна їх значень впливає “на всі” екземпляри типу, у якому вони фігурують:

```
static let/var propertyName...
```

Звертання до властивостей типу відбувається через ім'я типу, а не екземпляра (сутність) цього типу:

```
EntityType.propertyName
```

# Властивості: спостерігачі (observers)



Властивості у Swift надають механізм спостерігання за зміною своїх значень.

Синтаксис використання механізму спостерігання у Swift має наступний вигляд:

```
let/var propertyName: TypeOfValue {  
    didSet(newValue) {  
        // викликається перед тим, як propertyName буде присвоєно newValue  
    }  
  
    didSet {  
        // викликається після того, як propertyName було встановлено значення  
    }  
}
```

# Перерахування. Структури

Перерахування

Структури

**Властивості**

Ініціалізатори

Методи

Індексатори

Розширення



# Перерахування. Структури

Перерахування

Структури

Властивості

**Ініціалізатори**

Методи

Індексатори

Розширення

# Ініціалізатори

Ініціалізатор - це спеціальна функція із назвою *init*, метою якої є створення екземпляра типу, у якому вона описана.

Процес створення екземпляра типу з допомогою ініціалізатора називається ініціалізацією.

Звернення до ініціалізатора у межах тіла типу відбувається за назвою *init*.

Звернення до ініціалізатора поза межами тіла типу (створення екземпляра) відбувається за назвою *типу*.

# Ініціалізатори

Ініціалізатор зобов'язаний *ініціалізувати* або *надати* значення за замовчуванням усім зберігаючим властивостям, які не мають такого значення на етапі їх оголошення.

Задачею ініціалізації є *підготовка* екземпляра до використання.

# Ініціалізатори: failable

Failable ініціалізатор - це ініціалізатор, який створює екземпляр Optional типу.

Swift надає два failable ініціалізатори:

```
init?() {  
    // initialization or  
    // return nil  
}
```

```
init!() {  
    // initialization or  
    // return nil  
}
```

# Перерахування. Структури

Перерахування

Структури

Властивості

**Ініціалізатори**

Методи

Індексатори

Розширення

# Перерахування. Структури

Перерахування

Структури

Властивості

Ініціалізатори

**Методи**

Індексатори

Розширення

# Методи

Методи - це функції, які дозволяють виконувати операції над властивостями.

Методи поділяються на методи екземпляра та методи типа.

Методи, які виконують операції над властивостями *екземпляра та(або) типа*, називається *методами екземпляра*.

Методи, які виконують операції над властивостями *типа*, називається *методами типа*.

`self` - це ключове слово для доступу до всіх властивостей та методів екземпляра.

`self` - це власне сам екземпляр.

# Методи: мутація

Методи у структурах та перерахуваннях не можуть змінювати значення властивостей за замовчування. Структури і перерахування є типами за значенням. `mutating` - ключове слово, яке вказується в оголошенні функції і надає можливість функціям змінювати значення властивостей або самих екземплярів.



# Перерахування. Структури

Перерахування

Структури

Властивості

Ініціалізатори

**Методи**

Індексатори

Розширення

# Перерахування. Структури

Перерахування

Структури

Властивості

Ініціалізатори

Методи

**Індексатори**

Розширення

# Індексатори

Індексатор - це метод, який дозволяє робити вибірку значення (-нь) за індексом або кількома індексами).

Swift надає наступний синтаксис опису індексатора:

```
accessLevel subscript(indexName: IndexType, ...) -> ReturnType {  
    // return some value(s)  
}
```



# Індексатори: subscript

75	<code>struct Matrix {</code>		
76	<code>    let rows: Int, columns: Int</code>		
77	<code>    var grid: [Double]</code>		
78	<code>    init(rows: Int, columns: Int) {</code>		
79	<code>        self.rows = rows</code>		
80	<code>        self.columns = columns</code>		
81	<code>        grid = Array(repeating: 0.0, count: rows * columns)</code>		
82	<code>    }</code>		
83	<code>    func indexIsValid(row: Int, column: Int) -&gt; Bool {</code>		
	<code>        return row &gt;= 0 &amp;&amp; row &lt; rows &amp;&amp; column &gt;= 0 &amp;&amp; column &lt; columns</code>	(3 times)	
85	<code>    }</code>		
86	<code>    subscript(row: Int, column: Int) -&gt; Double {</code>		
87	<code>        get {</code>		
88	<code>            assert(indexIsValid(row: row, column: column), "Index out of range")</code>	3.2	
	<code>            return grid[(row * columns) + column]</code>		
90	<code>        }</code>		
91	<code>        set {</code>		
92	<code>            assert(indexIsValid(row: row, column: column), "Index out of range")</code>	(2 times)	
	<code>            grid[(row * columns) + column] = newValue</code>		
94	<code>        }</code>		
95	<code>    }</code>		
96	<code>}</code>		
97			
	<code>var matrix = Matrix(rows: 2, columns: 2)</code>	Matrix	
	<code>matrix[0, 1] = 1.5</code>	1.5	
	<code>matrix[1, 1] = 3.2</code>	3.2	
101			
	<code>let somematrixInValue = matrix[1, 1]</code>	3.2	
103			

# Перерахування. Структури

Перерахування

Структури

Властивості

Ініціалізатори

Методи

**Індексатори**

Розширення

# Перерахування. Структури

Перерахування

Структури

Властивості

Ініціалізатори

Методи

Індексатори

**Розширення**

# Розширення

Розширення - це спеціальна конструкція, яка дозволяє додавати функціональність до вже існуючого типу даних.




Swift надає наступний синтаксис опису розширення:

```
extension ExistingTypeName: Protocol1, ... {  
    // additional functionality  
}
```

Розширення *не можуть* описувати зберігаючі властивості для структур, а також перераховуючі елементи для перерахувань.

# Розширення



<pre>75 struct Matrix { 76     let rows: Int, columns: Int 77     var grid: [Double] 78     init(rows: Int, columns: Int) { 79         self.rows = rows 80         self.columns = columns 81         grid = Array(repeating: 0.0, count: rows * columns) 82     } 83 84     subscript(row: Int, column: Int) -&gt; Double { 85         get { 86             assert(indexIsValid(row: row, column: column), "Index out of range") 87             return grid[(row * columns) + column] 88         } 89         set { 90             assert(indexIsValid(row: row, column: column), "Index out of range") 91             grid[(row * columns) + column] = newValue 92         } 93     } 94 } 95 96 extension Matrix { 97     func indexIsValid(row: Int, column: Int) -&gt; Bool { 98         return row &gt;= 0 &amp;&amp; row &lt; rows &amp;&amp; column &gt;= 0 &amp;&amp; column &lt; columns 99     } 100 }</pre>	3.2	
	(2 times)	
	(3 times)	







UNIVERSITY