
SIGMA
Software


SIGNL
Software
UNIVERSITY


SIGMA
Software
UNIVERSITY

ЛЕКЦІЯ 10

“Категорії. Розширення. Протокол. Делегат.”

10

SIGMA SOFTWARE UNIVERSITY

Категорії. Розширення. Протокол. Делегат.

Категорії

Розширення

Протокол

Делегат

Категорії. Розширення. Протокол. Делегат.

Категорії

Розширення

Протокол

Делегат

Категорії - оголошення

Додає новий функціонал до існуючого класу

В Objective-C катергорії мають імена, в Swift - ні

Категорії - обмеження

Не можна додавати змінні

Можливі проблеми при назві методів, тому необхідно додавати свої префікси

Категорії - доступний функціонал

Додавати обчислювані властивості

Додавати методи класу і методи обєкту

Додавати нові ініціалізатори

Додавати сабскріпти

Додавати вкладені типи

Наслідувати протокол

SIGMA SOFTWARE UNIVERSITY

Категорії - оголошення

OBJ-C

```
#import "ClassName.h"
@interface ClassName (CategoryName)
    // Оголошення методів
@end
```

SIGMA SOFTWARE UNIVERSITY

Категорії - оголошення

OBJ-C

```
#import "ClassName.h"
@interface ClassName (CategoryName)
    // Оголошення методів
@end
```

SIGMA SOFTWARE UNIVERSITY

Категорії - оголошення

```
extension SomeType {
        // new functionality to add to SomeType goes here
}

extension SomeType: SomeProtocol, AnotherProtocol {
    // implementation of protocol requirements goes here
}
```

SIGMA SOFTWARE UNIVERSITY

Категорії - реалізація, властивості

```
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"
```

Категорії - реалізація, ініціалізатори

```
extension Rect {
    init(center: Point, size: Size) {
        let originX = center.x - (size.width / 2)
        let originY = center.y - (size.height / 2)
        self.init(origin: Point(x: originX, y: originY), size: size)
    }
}
```

SIGMA SOFTWARE UNIVERSITY

Категорії - реалізація, методи

```
extension Int {
    func repetitions(task: () -> Void) {
        for _ in 0..<self {
            task()
        }
    }
}
```

SIGMA SOFTWARE UNIVERSITY

Категорії - реалізація, вкладені типи

```
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
        }
    }
}
```

Категорії. Розширення. Протокол. Делегат.

Категорії

Розширення

Протокол

Делегат

SIGMA SOFTWARE UNIVERSITY

Категорії. Розширення. Протокол. Делегат.

Категорії

Розширення

Протокол

Делегат

SIGMA SOFTWARE UNIVERSITY

Розширення - оголошення

OBJ-C

```
@interface ClassName ()
    // Оголошення методів
@end
```

SIGMA SOFTWARE UNIVERSITY

Розширення - обмеження

Розширення не можуть бути для будь-якого класу, тільки для класу з оригінальною імплементацією

В розширення додаються приватні методи і приватні змінні для певного класу

Методи оголошення в розширення недоступні в похідних класах

Категорії. Розширення. Протокол. Делегат.

Категорії

Розширення

Протокол

Делегат

SIGMA SOFTWARE UNIVERSITY

Категорії. Розширення. Протокол. Делегат.

Категорії

Розширення

Протокол

Делегат

SIGMA SOFTWARE UNIVERSITY

Протокол - завдання

Очікується, що клас який підтримує протокол виконає описані в протоколі функції

Підтримка протокола на рівні об’єкта, не розкриваються методи та реалізація самого класу

Оскільки відсутнє множинне наслідування, то відбувається об’єднання по загальним властивостям декількох класів

Протокол - оголошення

OBJ-C

```
@protocol ProtocolName
@required
    // Оголошення методів
@optional
    // Оголошення методів
@end
```

SIGMA SOFTWARE UNIVERSITY

Протокол - оголошення

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

SIGMA SOFTWARE UNIVERSITY

Протокол - наслідування

Протоколи можуть наслідувати інші протоколи.

Наслідування додає до вимог підпротокола (subprotocol) вимоги суперпротокола (superprotocol).

Композиція протоколів

Сутність може приймати один або декілька протоколів.

Звідси випливає, що екземпляр протокольного типу може приймати декілька протоколів, компонуючи їх у тимчасовому протоколі

Протокол - наслідування

OBJ-C

```
@protocol ProtocolName <Protocol1, Protocol2>
```

SIGMA SOFTWARE UNIVERSITY

Протокол - наслідування

```
protocol SomeProtocol {
    // protocol definition goes here
}

protocol SomeProtocol2: SomeProtocol {
    // protocol definition goes here
}
```

SIGMA SOFTWARE UNIVERSITY

Відповідність протоколам

Сутність протокольного типу, може бути перевірена на відповідність деякому протоколу за допомогою механізму приведення типів.

Відповідність протоколам

Приведення типів - це механізм, завдяки якому екземпляр деякого типу може бути перевірений на відповідність цьому типу, або приведений до деякого іншого типу.

Приведення типів відбувається завдяки конструкціям *is* та *as* (*as?* та *as!*)

Результатом використання конструкції *is* над деяким екземпляром деякого типу є логічне значення, яке стверджує чи відповідає цей екземпляр деякому іншому вказаному типу.

Bool = instance is TypeName

Відповідність протоколам

Приведення типів - це механізм, завдяки якому екземпляр деякого типу може бути перевірений на відповідність цьому типу, або приведений до деякого іншого типу.

Приведення типів відбувається завдяки конструкціям *is* та *as* (*as?* та *as!*)

Результатом використання конструкцій *as* над деяким екземпляром деякого типу є екземпляр деякого іншого вказаного типу, якщо оригінальний екземпляр може бути приведений до нього.

TypeName? = instance as? TypeName
TypeName! = instance as! TypeName

Протоколи: відповідність протоколам

Класи, структури та перерахування можуть відповідати вимогам протоколів, тобто приймають протоколи.

Клас, структура, перерахування може приймати більше ніж один протокол.

Протокол - підтримка

OBJ-C

```
@interface ClassName: SuperClassName <Protocol1, Protocol2>
@end

if ([object conformsToProtocol:@protocol(Protocol1)]) {
//реалізація
}
```

SIGMA SOFTWARE UNIVERSITY

Протокольний тип

Протоколи не виконують жодної функціональності. Однак будь-який протокол є повноцінним типом.

Розширення протоколів

Протоколи можуть бути розширені.

Розширення задають реалізацію за замовчування.

Розширення протоколів

```swift
//: Protocols — Extensions

import Foundation

protocol Printable {
    var character: Character { get set }
}

extension Printable {
    func toString(base: Bool = false) {
        print(character)
    }
}

struct Letter: Printable {
    var character: Character = "b"

    func toString(base: Bool) {
        if base {
            (self as Printable).toString()
        } else {
            print(String(character).uppercaseString)
        }
    }
}

let character: Character = "a"
let letter = Letter(character: character)

letter.toString(false); letter.toString(true)
```

```
A
a
```

SIGMA SOFTWARE UNIVERSITY

Optional Requirement

Протоколи можуть надавати вимоги, яким сутності можуть не обов’язково відповідати.

Такий підхід забезпечується лише з допомогою механізм сумісності (interoperability) з Objective-C, де сутності не зобов’язані відповідати усім вимогам протоколів.

Конструкція @objc використовується для режиму сумісності з Objective-C на Swift.

Optional Requirement

```
@objc protocol Some {
    @objc optional func someFunc()
}
```

Категорії. Розширення. Протокол. Делегат.

Категорії

Розширення

Протокол

Делегат

SIGMA SOFTWARE UNIVERSITY

Категорії. Розширення. Протокол. Делегат.

Категорії

Розширення

Протокол

Делегат

SIGMA SOFTWARE UNIVERSITY

Делегування

Протоколи допомагають реалізувати шаблон програмування, який називається Делегуванням (delegation).

Делегування - це шаблон проектування, екземпляру класа або структури делегувати деякі власні повноваження екземпляру деякого іншого типу.

Делегат

OBJ-C

```
@protocol Protocol <NSObject>
@required
- (void)protocolMethod1;
- (void)protocolMethod2;
@end

@interface Delegate : NSObject

@property (nonatomic, weak) id <Protocol> delegate;

-(void)method1;
-(void)method2;

@end
```

SIGMA SOFTWARE UNIVERSITY

Делегування

```
1  //: Protocol - Delegation
2
3  import Foundation
4
5  struct Owner {
6
7      var text: String? {
8          didSet {
9              printText()                          Owner
10         }
11     }
12
13     func printText() {
14         print(text ?? "")                        "Hello World\n"
15     }
16 }
17
18 var owner = Owner()                             Owner
19 owner.text = "Hello World"                      Owner
```

SIGMA SOFTWARE UNIVERSITY

Делегування

```
1  //: Protocol - Delegation
2
3  import Foundation
4
5  struct Owner {
6
7      var text: String? {
8          didSet {
9              printText()                                  Owner
10         }
11     }
12
13     func printText() {
14         print(text ?? "")                                "Hello World\n"
15     }
16 }
17
18 struct Printer {
19
20 }
21
22 var owner = Owner()                                     Owner
23 owner.text = "Hello World"                              Owner
24
25 let printer = Printer()                                 Printer
26
27 // printer should print the owner's text when it changes
```

SIGMA SOFTWARE UNIVERSITY

Делегування

```
1  //: Protocol - Delegation                                          
2                                                                     
3  import Foundation                                                  
4                                                                     
5  protocol OwnerDelegate {                                           
6                                                                     
7      func printText(text: String)                                   
8  }                                                                  
9                                                                     
10 struct Owner {                                                     
11                                                                    
12     var delegate: OwnerDelegate?                                   
13                                                                    
14     var text: String? {                                            
15         didSet {                                                   
16             delegate?.printText(text ?? "")                        ()
17         }                                                          
18     }                                                              
19 }                                                                  
20                                                                    
21 struct Printer: OwnerDelegate {                                   
22                                                                    
23     func printText(text: String) {                                 
24         print(text)                                                "Hello World\n"
25     }                                                              
26 }                                                                  
27                                                                    
28 let printer = Printer()                                           Printer
29                                                                    
30 var owner = Owner()                                               Owner
31 owner.delegate = printer                                          Owner
32 owner.text = "Hello World"                                       Owner
```

SIGMA SOFTWARE UNIVERSITY

Делегування

```
1  //: Protocol - Delegation
2
3  import Foundation
4
5  protocol OwnerDelegate {
6
7      func printText(text: String)
8  }
9
10 struct Owner {
11
12     var delegate: OwnerDelegate?
13
14     init(delegate: OwnerDelegate? = nil) {
15         self.delegate = delegate
16     }
17
18     var text: String? {
19         didSet {
20             delegate?.printText(text ?? "")                    (2 times)
21         }
22     }
23 }
24
25 struct Printer: OwnerDelegate {
26
27     func printText(text: String) {
28         print(text)                                            (2 times)
29     }
30 }
31
32 var owner = Owner(delegate: Printer())                         Owner
33 owner.text = "Hello World"                                     Owner
34 owner.text = "Hello Delegate"                                  Owner
```

SIGMA SOFTWARE UNIVERSITY

Делегування

```swift
//: Protocol - Delegation

import Foundation

protocol OwnerDelegate: class {

    func printText(text: String)
}

class Owner {

    weak var delegate: OwnerDelegate?

    init(delegate: OwnerDelegate? = nil) {
        self.delegate = delegate
    }

    var text: String? {
        didSet {
            delegate?.printText(text ?? "")
        }
    }
}

class Printer: OwnerDelegate {

    func printText(text: String) {
        print(text)
    }
}

var owner = Owner(delegate: Printer())
owner.text = "Hello World"
owner.text = "Hello Delegate"
```

(2 times)

```
Owner
Owner
Owner
```

SIGMA SOFTWARE UNIVERSITY


SIGMA
Software
UNIVERSITY