
SIGMA
Software


SIGNL
Software
UNIVERSITY


SIGMA
Software
UNIVERSITY

ЛЕКЦІЯ 6

“Властивості об’єктів”

6

SIGMA SOFTWARE UNIVERSITY

Властивості об’єктів

Властивості в Objective-C

Властивості в Swift

SIGMA SOFTWARE UNIVERSITY

Властивості в Objective-C

@property

Аксесори: (getters and setters)

Атрибути властивостей

@syntesize

@dynamic

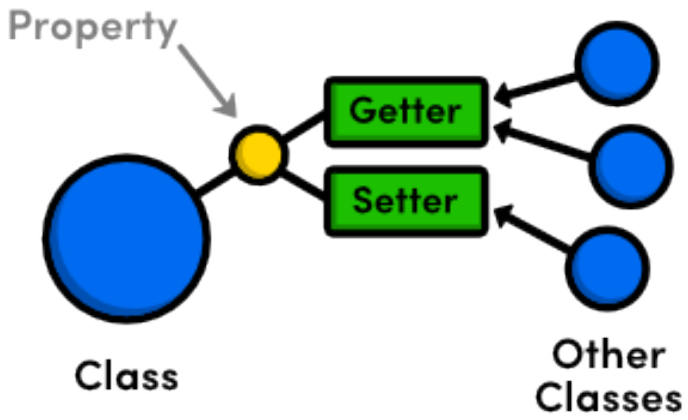
SIGMA SOFTWARE UNIVERSITY

@property

властивості - дозволяють іншим класам дізнаватися інформацію про об’єкт

OBJ-C

```
@property (copy, nonatomic) NSString *title;

@property (<#attributes#>) <#type#> <#name#>;
```



SIGMA SOFTWARE UNIVERSITY

@property

Ви пишете

// SomeObj.h

```
#import <Foundation/Foundation.h>

@interface SomeObj : NSObject

@property BOOL propertyName;

@end
```

// SomeObj.m

```
#import "SomeObj.h"

@implementation SomeObj

@synthesize propertyName = _propertyName;    // Optional for Xcode 4.4+
@end
```

Компілятор генерує:

```
//getter
- (BOOL)propertyName
{
    return _propertyName;
}

//setter
- (void)setPropertyName:
   (BOOL)newValue
{
    _propertyName = newValue;
}
```

OBJ-C

SIGMA SOFTWARE UNIVERSITY

@property

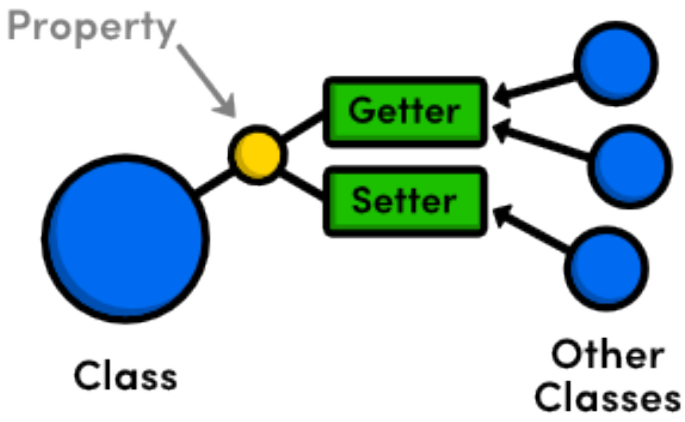
OBJ-C

```
// main.m

#import <Foundation/Foundation.h>
#import "SomeObj.h"

int main(int argc, const char * argv[]) {
    @autoreleasepool {
        SomeObj *obj = [[SomeObj alloc] init];
        obj.someProperty = YES;                 // [obj setSomeProperty:YES]
        NSLog(@"%d", obj.someProperty);         // [obj someProperty]
    }
    return 0;
}
```

SIGMA SOFTWARE UNIVERSITY

Аксесори: (getters and setters)

«…accessor are used as an abstraction for interacting with the object’s underlying data.»



Accessors: (getter= and setter=)

можна використати будь-які власні імена

```
@property BOOL (getter = myAwesomePropertyGetter, setter = evenBetterSetterName) propertyName;
```

OBJ-C

SIGMA SOFTWARE UNIVERSITY

Атрибути властивостей

Список атрибутів

atomic (default)
nonatomic
strong (default for object)
weak
readwrite (default)
readonly
getter=
setter=
copy
assign (default for primitives, pre-ARC)

retain (pre-ARC) —> strong
unsafe_unretained (pre-ARC) —> weak

@syntesize

- “It tells the compiler to synthesize the setter and/or getter if you do not include them in the @implementation” - not required from xCode 4.4 (mid of 2013)
- the modern runtime (iPhone apps, and 64bit apps) synthesizes the ivars
- Naming conventions:
    - The getter method has the same name as the property. The getter method for a property called **firstName** will also be called **firstName**.
    - The setter method starts with the word “set” and then uses the capitalized property name.The setter method for a property called **firstName** will be called **setFirstName**:.

@dynamic

@dynamic just tells the compiler that the getter and setter methods are implemented not by the class itself but somewhere else (like the superclass or will be provided at runtime) - implementing the accessors is delegated.

Super class:
@property (strong, nonatomic) UIButton *someButton;
...
@synthesize someButton;

OBJ-C

Subclass:
@property (strong, nonatomic) IBOutlet UIButton *someButton;
...
@dynamic someButton;

Using the @dynamic directive essentially tells the compiler "don't worry about it, a method is on the way." (Apple)

Властивості в Swift

Stored Properties

Computed Properties

Property Observers

Global and Local Variables

Type Properties

SIGMA SOFTWARE UNIVERSITY

Stored Properties

store constant and variable values as part of an instance

are provided only by classes and structures

associated with instances of a particular type or with typeItself (type properties)

support observers

Stored Properties

```
// variable - use var
var myProperty:String?
myProperty = "Something"

// constants - use let
let myConstants:String = "something"
//myConstants = "new Value" //error Cannot assign to value: 'myConstants' is a 'let'
constant
```

SIGMA SOFTWARE UNIVERSITY

Stored Properties

```
struct FixedLengthRange {
    var firstValue: Int
    let length: Int
}

var rangeOfThreeItems = FixedLengthRange(firstValue: 0, length: 3)
// the range represents integer values 0, 1, and 2

rangeOfThreeItems.firstValue = 6
// the range now represents integer values 6, 7, and 8
```

SIGMA SOFTWARE UNIVERSITY

Lazy Stored Properties

initial value is not calculated until the first time it is used

Lazy Stored Properties

```
class DataImporter {
    /*
     DataImporter is a class to import data from an external file.
     The class is assumed to take a nontrivial 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
}

let manager = DataManager()
manager.data.append("Some data")
manager.data.append("Some more data")
// the DataImporter instance for the importer property has not yet been created

print(manager.importer.filename) //first usage here
```

Computed Properties

computed properties calculate (rather than store) a value

provided by classes, structures, and enumerations

associated with instances of a particular type or with typeItself (type properties)

Computed Properties

```
51  // Computed
52
53  struct Something {
54      var value1 = 0
55      var value2 = 0
56  }
57
58  struct ResultOfSomething {
59      var obj1 = Something()
60      var obj2 = Something()
61
62      var someCalcResult:Int {
63          get {
                return obj1.value1 * obj2.value2                    8
65          }
66      }
67  }
68
    var obj = Something()                                           Something
    obj.value1 = 2                                                  Something
71
    var obj2 = Something()                                          Something
    obj2.value2 = 4                                                 Something
74
    var result = ResultOfSomething()                                ResultOfSomething
    result.obj1 = obj                                               ResultOfSomething
    result.obj2 = obj2                                              ResultOfSomething
78
    print(result.someCalcResult)                                    "8\n"
```

SIGMA SOFTWARE UNIVERSITY

Observers для властивостей

observe and respond to changes in a property’s value

can add property observers to any stored properties you define, except for lazy stored properties

add property observers to any inherited property

Observers для властивостей

willSet

didSet

Observers для властивостей

```
81  //Observers for properties
82
83  class StepCounter {
84      var totalSteps: Int = 0 {
85          willSet(newTotalSteps) {
                print("About to set totalSteps to \(newTotalSteps)")                    (3 times)

            About to set totalSteps
                   to 896...

87          }
88          didSet {
89              if totalSteps > oldValue  {
                    print("Added \(totalSteps - oldValue) steps")                       (3 times)
91              }
92          }
93      }
94  }
    let stepCounter = StepCounter()                                                     StepCounter
    stepCounter.totalSteps = 200                                                        StepCounter
97  // About to set totalSteps to 200
98  // Added 200 steps
    stepCounter.totalSteps = 360                                                        StepCounter
100 // About to set totalSteps to 360
101 // Added 160 steps
    stepCounter.totalSteps = 896                                                        StepCounter
103 // About to set totalSteps to 896
104 // Added 536 steps
```

SIGMA SOFTWARE UNIVERSITY

Global and Local Variables

Global variables are variables that are defined outside of any function, method, closure, or type context

Local variables are variables that are defined within a function, method, or closure context

обидва типи можуть бути computed або stored

Type Properties

```
// Type properties
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
    }
}

print(SomeStructure.storedTypeProperty)
// Prints "Some value."
SomeStructure.storedTypeProperty = "Another value."
print(SomeStructure.storedTypeProperty)
// Prints "Another value."
print(SomeEnumeration.computedTypeProperty)
// Prints "6"
print(SomeClass.computedTypeProperty)
// Prints "27"
```

```
6
27
"Some value.\n"
"Another value.\n"
"6\n"
"27\n"
```

SIGMA SOFTWARE UNIVERSITY

Список використаних ресурсів

Declaring properties (URL).

Properties attributes (URL)

Data encapsulation (URL)

Dynamic properties (URL)

Properties in Swift (URL)


SIGMA
Software
UNIVERSITY