SCIENCES SORBONNE UNIVERSITÉ

Objective-C, classes attributes and properties

Fabrice.Kordon@lip6.fr

LIP6

Production of getters & setters

Automated process

Described in the interface

- Property : @property
  - **Associated to a class attribute**
  - **Specifying access, memory management and concurrent behavior**

Automated implementation (in «.m»)

- Naming conventions (for myAttribute)
  - **getter : myAttribute (of myAttribute type)**
  - **setter : setMyAttribute (void + a parameter)**

- Keyword to synthesize: @synthesize
  - **Unnecessary since Xcode 5**

Access modes for attributes

Read only

- Keyword : readonly
- Only the getter is generated

Read/write

- Keyword : readwrite
- Getter and setter are generated

Otherwise

- You must implement it manually

Concurrent access to attributes

Protected access

- Keyword : atomic
- Manages locks
  - **Preserve access integrity**
  - **degraded performances**

No protection

- Keyword : nonatomic

Memory management for attributes

Basic assignment

- Keyword : assign

Duplication of the attribute

- Keyword : copy
  - **Creates a new identical object**

Incrementing the reference counter

- Keyword : retain
  - **Let's discuss this later**

Relation to a parent object (since ARC)

- Keyword : strong or weak
  - **Let's discuss this later**

Changing naming conventions

Not recommended

- Defined for backward compatibility in Apple's APIs
  - **have a look at Foundation's interfaces**
- Does not change the expected signature

For getters

- Keyword : getter=myName
  - **Uses myName**

For setters

- Keyword : seter=myName
  - **Uses myName**

Student (interface revisited)


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

@interface Student : NSObject {
    // instance variables
    int number;
    NSString *name;
}

// methods
- (int) myNumber;
- (NSString*) myName;
- (NSString*) identity;

- (void) setNumber:(int)aNumber;
- (void) setName:(NSString*)aName;
@end
```

Student (interface revisited)


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

@interface Student : NSObject

@property (readwrite, nonatomic, assign) int number;
@property (readwrite, nonatomic, copy) NSString *name;

// methods
- (NSString*) identity;

@end
```

Avoid name changes?

for getters in our case

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

@interface Student : NSObject

@property (readwrite, nonatomic, assign, getter=myNumber) int number;
@property (readwrite, nonatomic, copy, getter=myName) NSString *name;

// methods
- (NSString*) identity;

@end
```

Student implementation (revisited)


```objc
@implementation Student

- (int) myNumber {
    return number;
}

- (NSString*) myName {
    return [NSString stringWithFormat:@"%@",name];
}

- (NSString*) identity {
    return [NSString stringWithFormat:@"%@ (%d)", name, number];
}

- (void) setNumber:(int)aNumber {
    number = aNumber;
}

- (void) setName:(NSString*)aName {
    name = [aName copy];
}
@end
```

Student implementation (revisited)


```objc
#import "Student.h"

@implementation Student

- (NSString*) identity {
    return [NSString stringWithFormat:@"%@ (%d)", _name, _number];
}
@end
```

Variant

Two possible ways to access attributes

```objc
#import "Student.h"

@implementation Student

- (NSString*) identity {
    return [NSString stringWithFormat:@"%@ (%d)", [self myName], [self myNumber]];
}
@end
```

As a conclusion…

- If you redefine setters and/or getters?
  - Your writing is considered
- You are all set to write your own classes
- memory management discussed further