SCIENCES SORBONNE UNIVERSITÉ

# Objective-C, the object life cycle

Fabrice.Kordon@lip6.fr

LIP6

# As an introduction



## Since iOS 5

- Automatic Reference Counting (ARC)
  - **Manages references counters of objects**

## Principle

- iOS does it for you
  - **incrementing and decrementing «reference counters»**
  - **Memory management system (explained later)**

## But…

- Memory is important

  - **In some areas, it must be handled explicitly**
  - **Typically embedded systems or OS programming**
- You are computer scientists
- So we will have a look at it

# As an introduction



## Since iOS 5

- Automatic Reference Counting (ARC)
  - **Manages references counters of objects**

## Principle

- iOS does it for you
  - **incrementing an**
  - **Memory manage**

## But…

- Memory is important
  - **In some areas, it must be handled explicitly**
  - **Typically embedded systems or OS programming**
- You are computer scientists
- So we will have a look at it

**This video**

Discussion on the object life cycle

# Life cycle of an object



## Creation

- Allocation (find some space in the memory)
  - **+(id) alloc**
- Init
  - **-(id) init**

## Destruction

- Deallocation
  - **-(id) dealloc**
  - **-dealloc relies on a +dealloc**

## Methods implemented in NSObject

- Remind, all classes inherit from NSObject

# Example, creating a Student



## Typical creation of a Student (reminder)

```
Student *s= [[Student alloc] init];
```

## Overloading init

- Always invoke «super» first

```
- (id) init {
    if (self = [super init]) { // always test
        [self setNumber:0];
        [self setName:@"Tartempion"];
    }
    return self; // always return self
}
```

# Overloading init, some rules



## some overload of init

- (id) init;
  - **The default one, no need to declare it in the interface**
- (id) initWithNumber:(int)aNumber;
- (id) initWithNumber:(int)aNumber andName:(NSString*) aName;

## Init chain to be respected

- initWithNumber: andName:
  - **Invokes initWithNumber:**
- initWithNumber:
  - **Invokes init**

# Application to Student



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

@interface Student : NSObject

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

// Initialization
- (id) initWithNumber:(int)aNumber;
- (id) initWithNumber:(int)aNumber andName:(NSString*)aName;

// methods
- (NSString*) identity;

@end
```

# Application to Student



```m
#import "Student.h"

@implementation Student

- (id) init {
    if (self = [super init]) { // always test
        [self setNumber:0];
        [self setName:@"Tartempion"];
    }
    return self; // always return self
}

- (id) initWithNumber:(int)aNumber {
    [[self init] setNumber:aNumber];
    return self;
}

- (id) initWithNumber:(int)aNumber andName:(NSString*)aName {
    [[self initWithNumber:aNumber] setName:aName];
    return self;
}

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

# Application to Student



```
Student *s= [[Student alloc] initWithNumber:1 andName:@"Steve J."];
NSLog(@"Student name is ''%@''", [s identity]);
```

```
Student name is ''Steve J. (1)''
```

# About deallocation



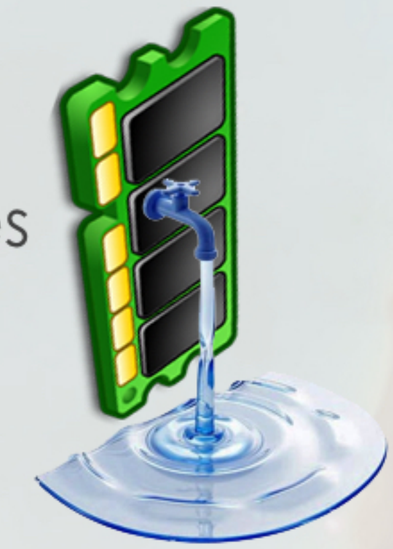
## Need to deallocate embedded objects

- Here, name
  - **It is a NSString**
- Not number
  - **not a class**

```
- (void) dealloc {
    [_name release];
    [super dealloc];
}
```

## Rule: balance allocations & deallocations

- Otherwise, memory leaks
  - **And then crash**
- Later, we will explose simple rules
  - **Instrument useful to detect leaks**



# ARC activated by default



- **Memory automatically handled by iOS**
  - This is quite nice

- **But…**
  - In this course, no ARC (with Objective-C)

# As a conclusion…



## Be ready for memory management

- Counting references = typical approach
- To be exposed soon