iPhone development

Andreu Urruela

Table of contents

Day I

- Introduction
  - Environment (Xcode, IB, Instruments, etc...)
  - Licenses and profiles.... all you need at TID!
- Objective-C: basis
  - Class model
  - Memory management (iphone without GC)
  - Delegate mechanism
- UIKit the graphical library
  - Views and our first application

Table of contents

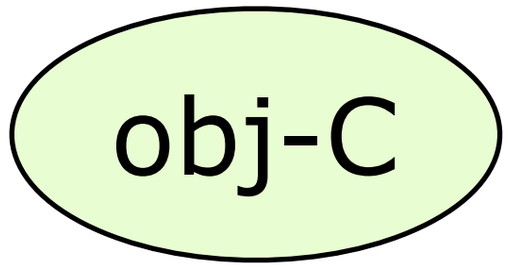
Day II

- Objective-C: a deeper vision
  - Categories
  - Properties
- Advance graphic library
  - UITableView (the best)
  - UIAlertView, UIScrollView, etc...
  - Advanced native app

Introduction

Introduction

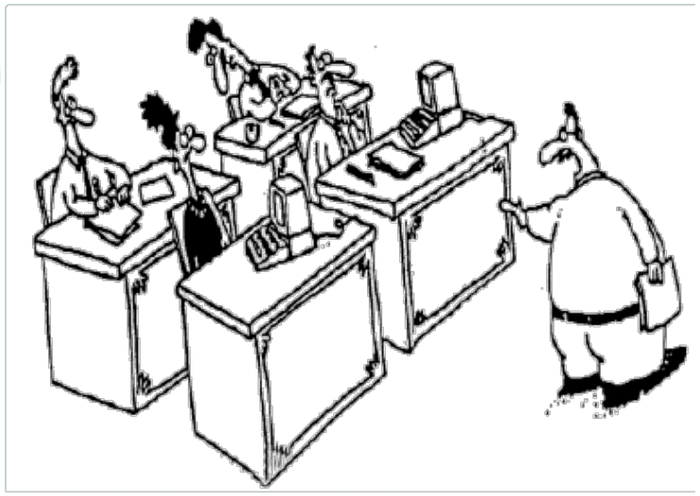
- It is a reduced BSD distribution with no shell :(
- Application developed with obj-C
- UI interface based on UIKit (obj-C framework)
- All the libraries available: cocoa-touch (cocoa for mac)



- Thin layer on top of C
- Used primarily on mac OS X, iPhone OS, GNUStep
- Compiled with GCC 4.0 (with objective-C support)
- Popularized by NextStep

Environment

Thinking & designing

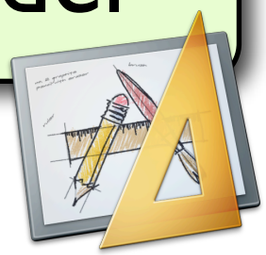


Coding

Debugging

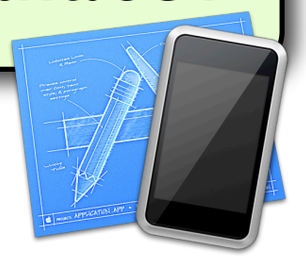
Optimizing

Interface Builder

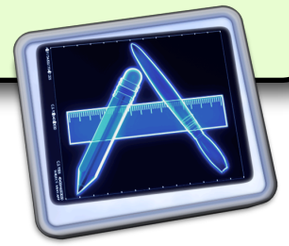


Xcode

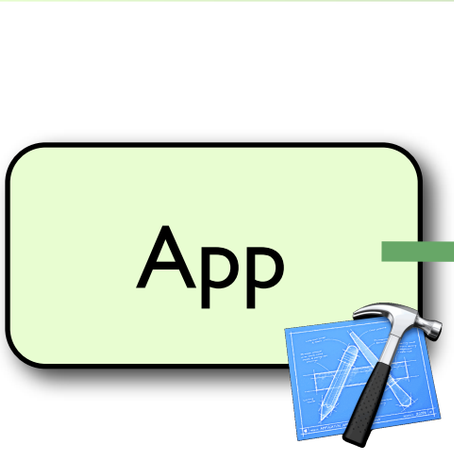
iPhone Simulator



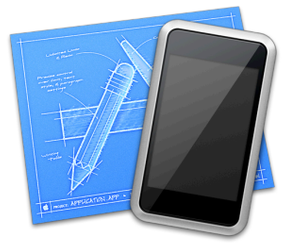
Instruments



Licenses



Debugging



Iphone Simulator

Development phones

Public distribution

AppStore

universal availability

limited distribution

AddHoc

limited availability (identifiers needed)

Licenses

What do I need?

Developer Private key
+
Developer certificate

Provisining profile

- Definitions
  - Private key: Private key generated by the the developer
  - Developer certificate: Generated by Apple using your private key. So only you can “sign” applications using this certificate
  - Application ID: Identifier of an application. It can be “*”
  - Provisioning: Small file connecting developer certificate - application id - set of terminals

Licenses:

Normal process for individual development

Private key is generated using key-chain

Private key is submitted to apple
you got your "developer certificate"
you got access to the "program protal"

In the "program portal" ...

Create an App identifier

Add your iphone device identifiers

Create the provisioning profile
certificate, app-Id and devices

Licenses:

Normal process for individual development

+

Provisioning

Debug on real devices

+

ADHOC - Deployment Provisioning

Distribute to the set of terminals included in the provisioning

+

Deployment Provisioning

Submit applications to Apple store

Licenses:

Normal process in Telefónica I+D

Private key and developer certificate: Andreu Urruela

Unique identifier for everybody
(Not necessary to download)

Provisioning

Unique provisioning for everybody with ALL internal phones

ADHOC - Deployment Provisioning

Unique ADHOC deployment provisioning for everybody with ALL internal phones

---

Deployment Provisioning

Deployment Provisioning

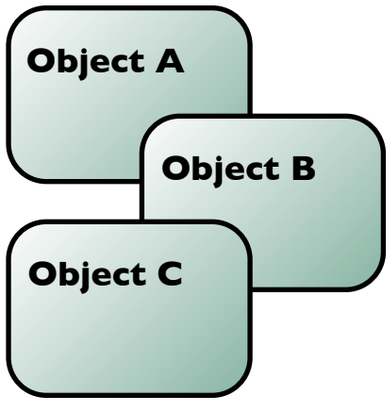
Deployment Provisioning

Deployment Provisioning

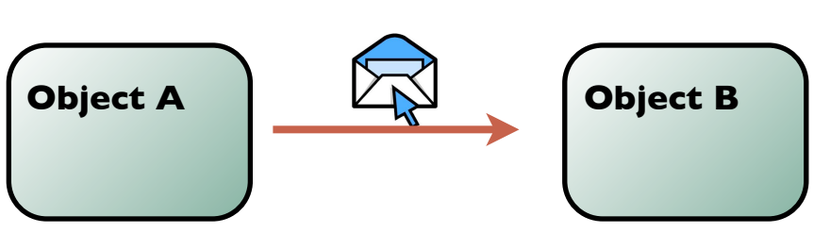
Dedicated deployment for applications ready for App Store

Objective-C:

Objective-C:



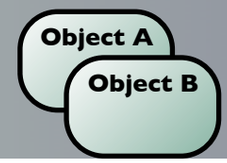
Objects



Messages

Memory management

- Object-oriented thin layer over C
- Everything is decided at run-time
- Messages are passed between objects
- Loosely typed (id for all objects)



Interface definition

Object A

int a;

id otherObject

pointer to

Object B

-(void) functionToDoSomething: (int) a;

Implementation definition

Interface definition

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

@interface nameOfTheClass : NSObject {

   int a;
   id otherObject;
   UIView *myView;
}

+(void) classFunctionToDoSomething;
-(void) functionToDoSomething;
-(int) functionThatReturnSomething;
-(void) functionWithParameter:(int) _a;
-(void) funcWithParameter:(int) _a andParameter:(int)_b;

@end
```

Import = include (ony including once)

@interface: key-work to define classes

id is a generic pointer to an object

static methos or class methods

multiple ways to define object methods

Very common: name of the function between parameters

Implementation definition

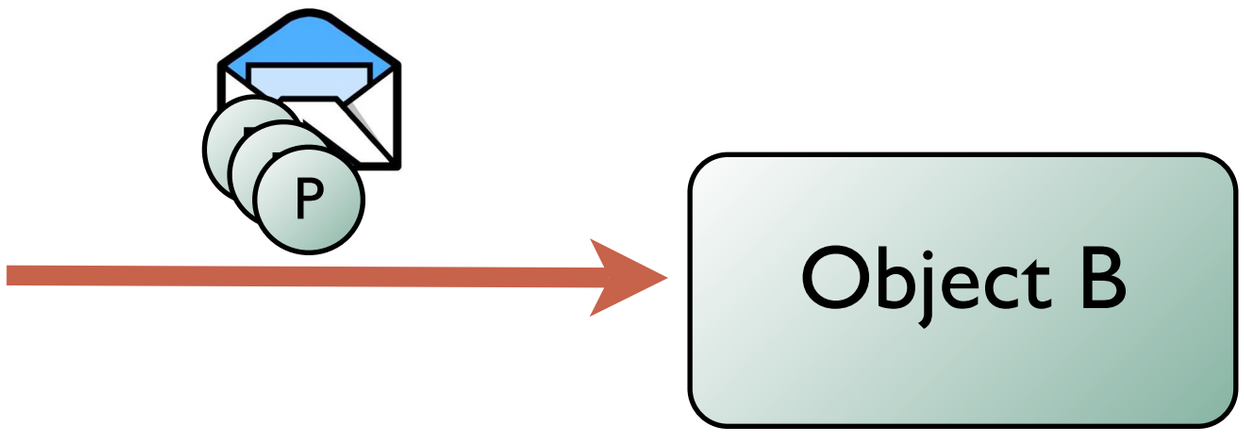
```
#import "nameOfTheClass.h"

@implementation nameOfTheClass

-(void) functionToDoSomething {
   NSLog(@"Hi there..");
}

-(int) functionThatReturnSomething {
   return 5;
}

-(void) functionWithParameter:(int) _a {
   NSLog(@"Parameter %d",_a);
}

-(void) funcWithParameter:(int) _a andParameter:(int)_b {
   NSLog(@"Parameter %d and %d",_a,_b);
}

@end
```

@implementation: key-word to define implementation

@"string" defining literals as NSString*

Very common: name of the function between parameters

"Sending a message" to an object
is equivalent to
"call a method" of the object

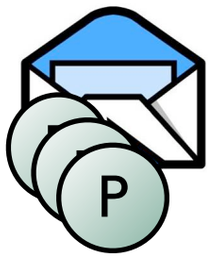



```
-(void) functionToDoSomething;
-(int) functionThatReturnSomething;
-(void) functionWithParameter:(int) _a;
-(void) funcWithParameter:(int) _a
                andParameter:(int)_b;
```

Syntax: [object method:parameter]

```
[otherObject functionToDoSomething];

int value = [otherObject functionThatReturnSomething]

[otherObject functionWithParameter: 25];

[otherObject funcWithParameter: 25 andParameter: 26];
```

Classes itself are also “objects”
so they can receive messages (static)



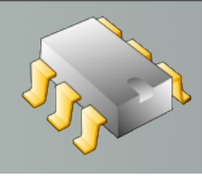
Class B

```
+(void) classFunctionToDoSomething;
```

Syntax: [className method:parameter]

```
[ClassName classFunctionToDoSomething];
```

Objective-C:



alloc - release

```
//Alloc memory
myView = [[UIView alloc] init];

//Use myView for whatsoever..

//Send a "release" message
[myView release];

// If I use "myView" here...
// I can get an error (or not if I am lucky)
```

Alocation and initialization
init is not a reserved word: convention
alloc set the memory-counter to 1

release : reduce the memory-counter by 1
deallocation is performed when counter=0

All the objects have a “momery-counter”.
This shows the number of owners

```
@implementation nameOfTheClass

-(id) init
{
   if ([super init])
   {
         //Initialization code
   }
   return self;   //return itself (rare exceptions)
}

-(void) dealloc
{
   //Dealloc memory

   [super dealloc];
}

// Implementation of the other functions...

@end
```

Initialization of the class
(***init*** is not a reserved word)

Deallocation of “allocated objects”
(***dealloc*** is not strictly a reserved word)

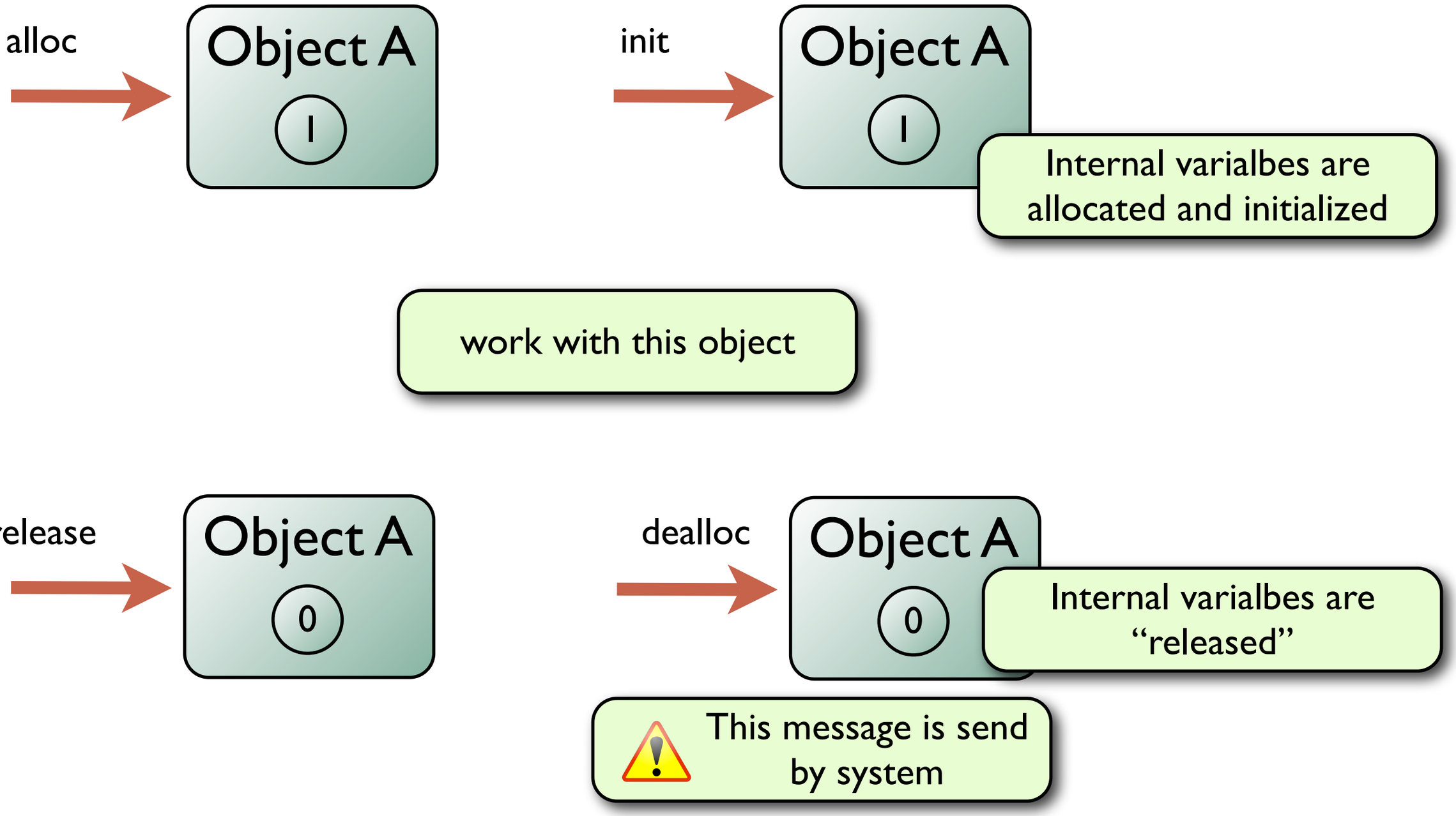
dealloc is automatically called by system
when the object itself is deallocated
(a release has been sent)

```
@interface nameOfTheClass
UIView *myView;
@end

@implementation nameOfTheClass
-(id) init{
  if ([super init]){

      //Initialization code
      myView = [[UIView alloc] init];
  }
  return self;  //return itself (rare exceptions)
}

-(void) dealloc{
  [myView release];    //Dealloc memory
  [super dealloc];
}
@end
```

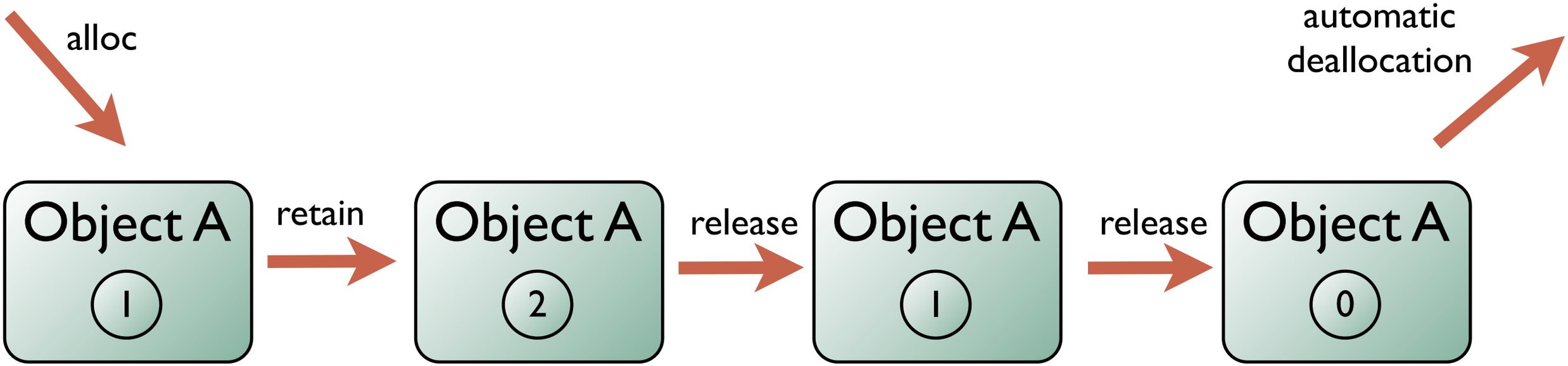
In the init, we usually alloc space for internal variables

In “dealloc” we usually “release” all allocated variables

Normal live-cicle of an object



Complete live-cycle of an object



Why ?

Multiple owners of an object
Every owner increase the memory-counter

Object A

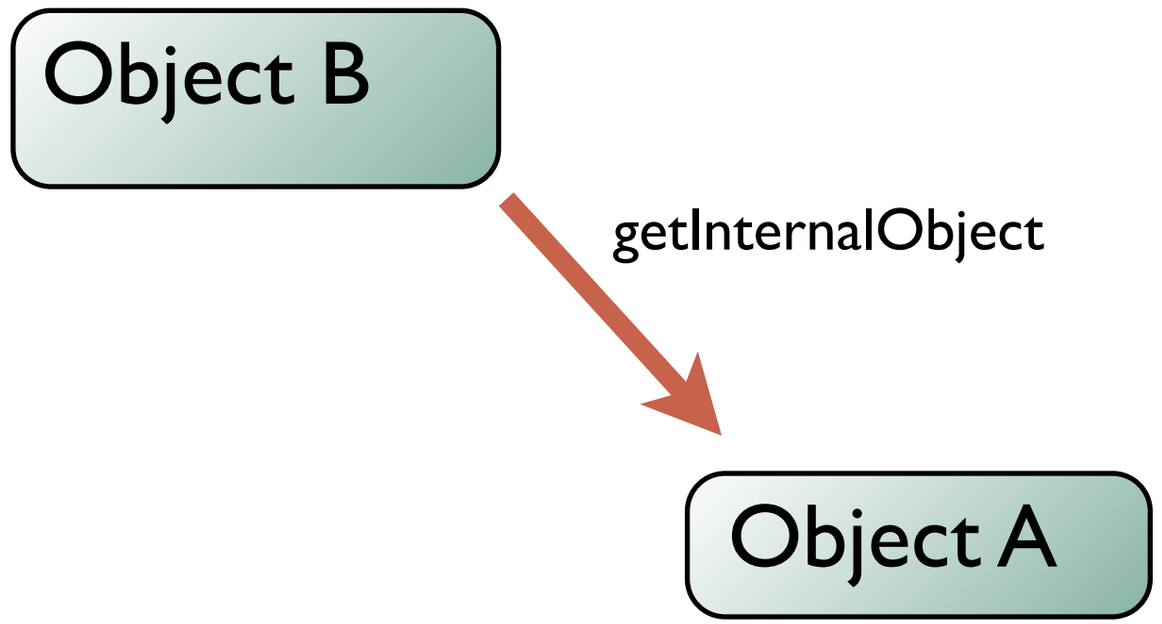
```
id internalObject;
```

```
-(id) getInternalObject
{
   return internalObject;
}
```

Object B

```
id internalObject;
```

```
internalObject = [objA getInternalobject];
```



Object B has a copy of an object "owned" by A

What happens of objA is deallocated by others....???

Object A

```
id internalObject;
```

```
-(id) getInternalObject
{
   return internalObject;
}
```

Object B

```
id internalObject;
```

```
internalObject = [[objA getInternalobject] retain];
```

now Object B is responsible
to send a “release”

otherwise: leak!

Objective-C: alloc - release

Your options

If you “alloc” or “retain” you own the object.
Otherwise: weak pointer

Your rights

If you own an object...
nobody will deallocate the object until you want!

Your obligations

for every “alloc” or “retain” you should send a “release”

Example: arrays

NSArray and NSMutableArray contain a fixed/variable number of objects (ids)

The array itself is allways owner of the contained elements to avoid deallocation of internal objects

```
//Init the mutable array with an initial capacity of 5 elements
NSMutableArray* array = [[NSMutableArray alloc] initWithCapacity:5];

//Init a UIView element
UIView* myView = [[UIView alloc] init];

//Add the new view to the array
[array addObject:myView]; // myView has received here a "retain" message

//Release the object (no automatic deallocation because array keeps the ownership)
[myView release];
```

Problem: setMethod

```
@interface nameOfTheClass
id element;
@end

@implementation nameOfTheClass
-(void) setElement:(id) _element {
   ???
}

-(void) dealloc {
   ???
}
@end
```

```
@implementation nameOfTheClass
-(void) setElement:(id) _element {
   element = _element;
}
-(void) dealloc {
[super dealloc];
}
@end
```

```
@implementation nameOfTheClass
-(void) setElement:(id) _element {
   element = [_element retain];
}
-(void) dealloc {
   [_element release];
   [super dealloc];
}
@end
```

old objects are no released: leaked objects!!

Problem: setMethod

```
@interface nameOfTheClass
id element;
@end

@implementation nameOfTheClass
-(void) setElement:(id) _element {
   ???
}

-(void) dealloc {
   ???
}
@end
```

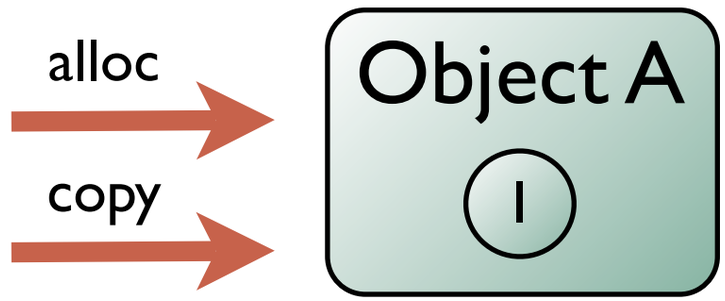
```
@implementation nameOfTheClass
-(void) setElement:(id) _element {
   [element release];
   element = [_element retain];
}
-(void) dealloc {
   [_element release];
   [super dealloc];
}
@end
```

if _element and element are the same, object is deallocated by mistake

```
@implementation nameOfTheClass
-(void) setElement:(id) _element {
   if (element != _element)
        element = [_element retain];
}
-(void) dealloc {
   [_element release];
   [super dealloc];
}
@end
```

Bonus: copy & autorelease

Copy duplicate the content of an object returning a new object.
It increments the reference-counter by 1



autorelease is a normal release except that deallocation is not performed until the context finish. Useful for factory functions

```
-(id) createElement{
  id a = [[Object alloc] init];
  return a;
}
```

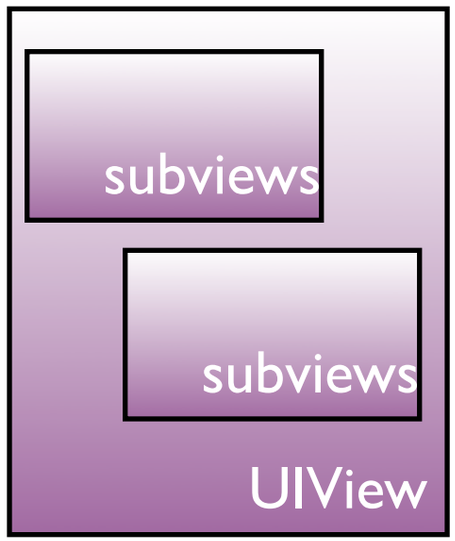
```
-(id) createElement{
  id a = [[Object alloc] init];
  return [a release];
}
```

```
-(id) createElement{
  id a = [[Object alloc] init];
  return [a autorelease];
}
```

UIKit

UIKit

UIKit is the framework with all graphical elements for native iphone apps

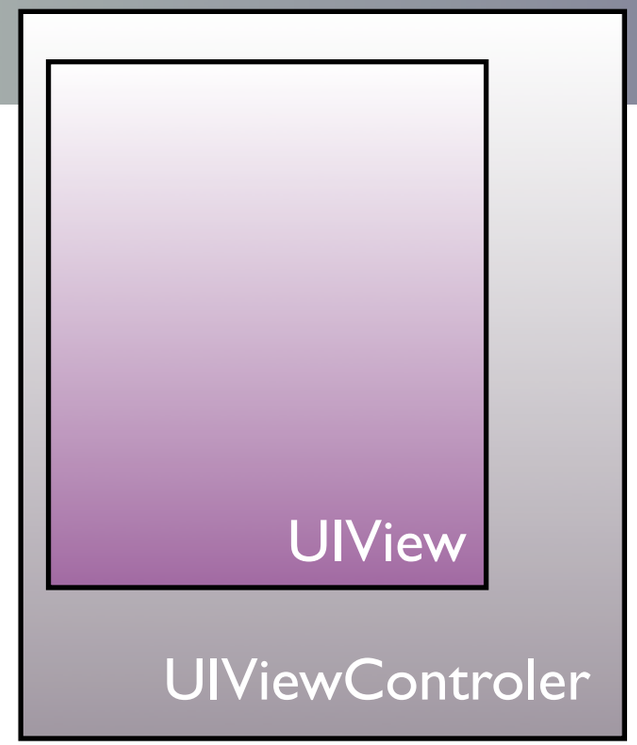


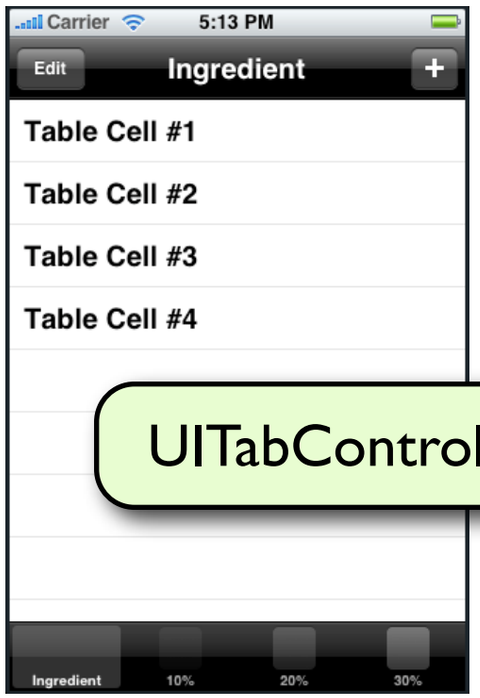
UIView → A rectangle on the screen

- UIImageView
- UILabels
- UIControls → Special events are fired
  - UISlider
  - UIButton

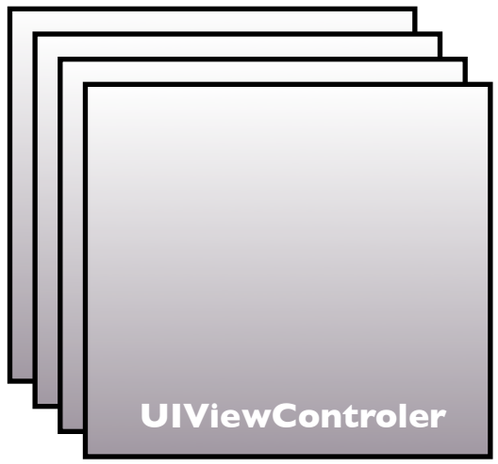
UIViewController:
Object controlling and owning a view

UIViewController managers labels and titles in the UINavigationController and UITabBarController





UITabController

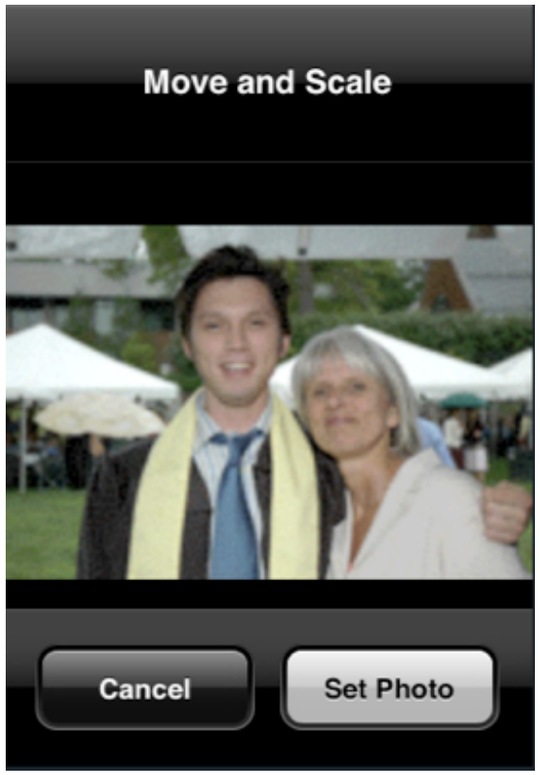


UINavigationController

UIKit

UITableView

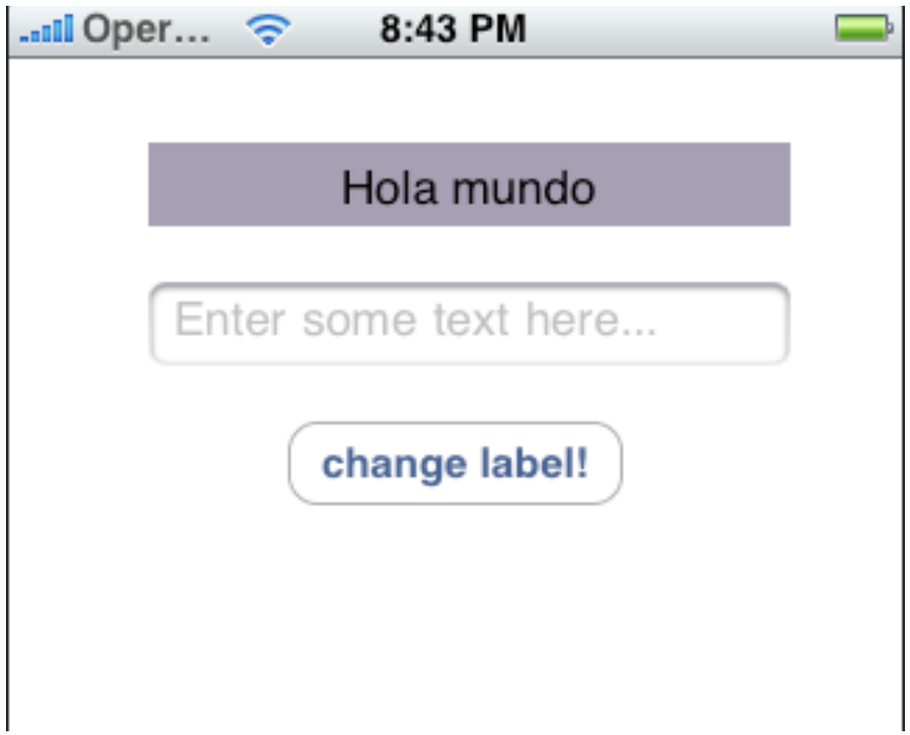
UIImagePicker



both from camera and from albums

UIKit

First example



UIlabel

UITextField

UIButton