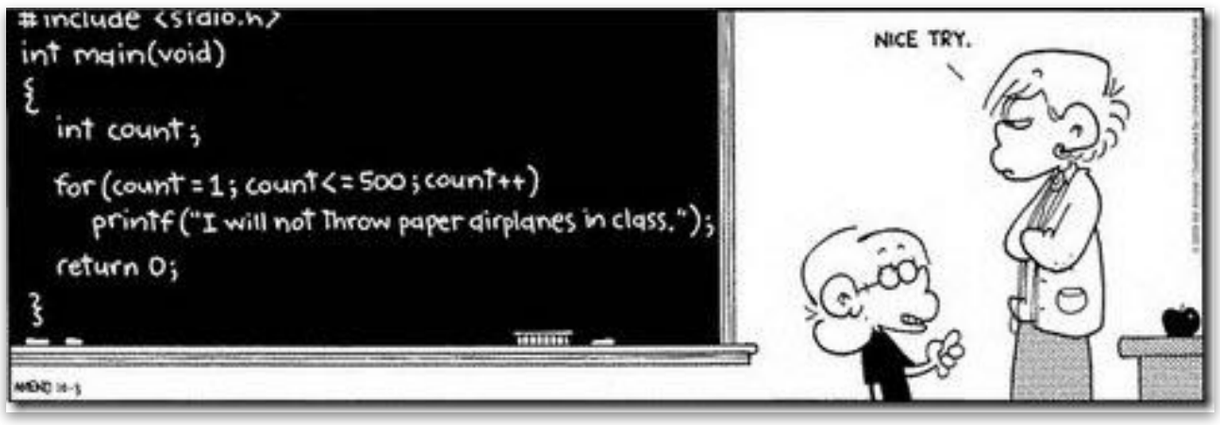
IUT Lyon 1
l'excellence technologique

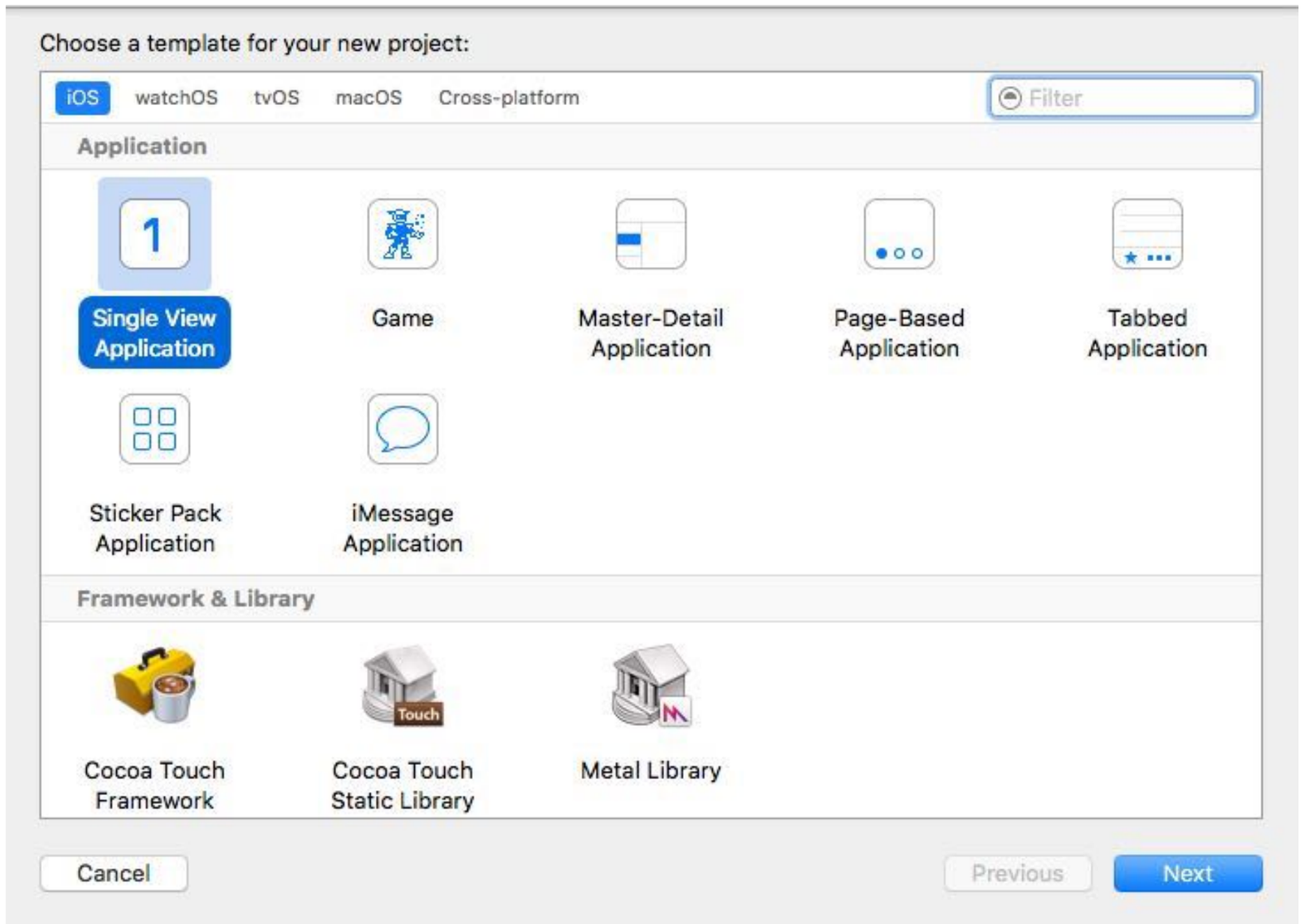
Développement iOS



Première application

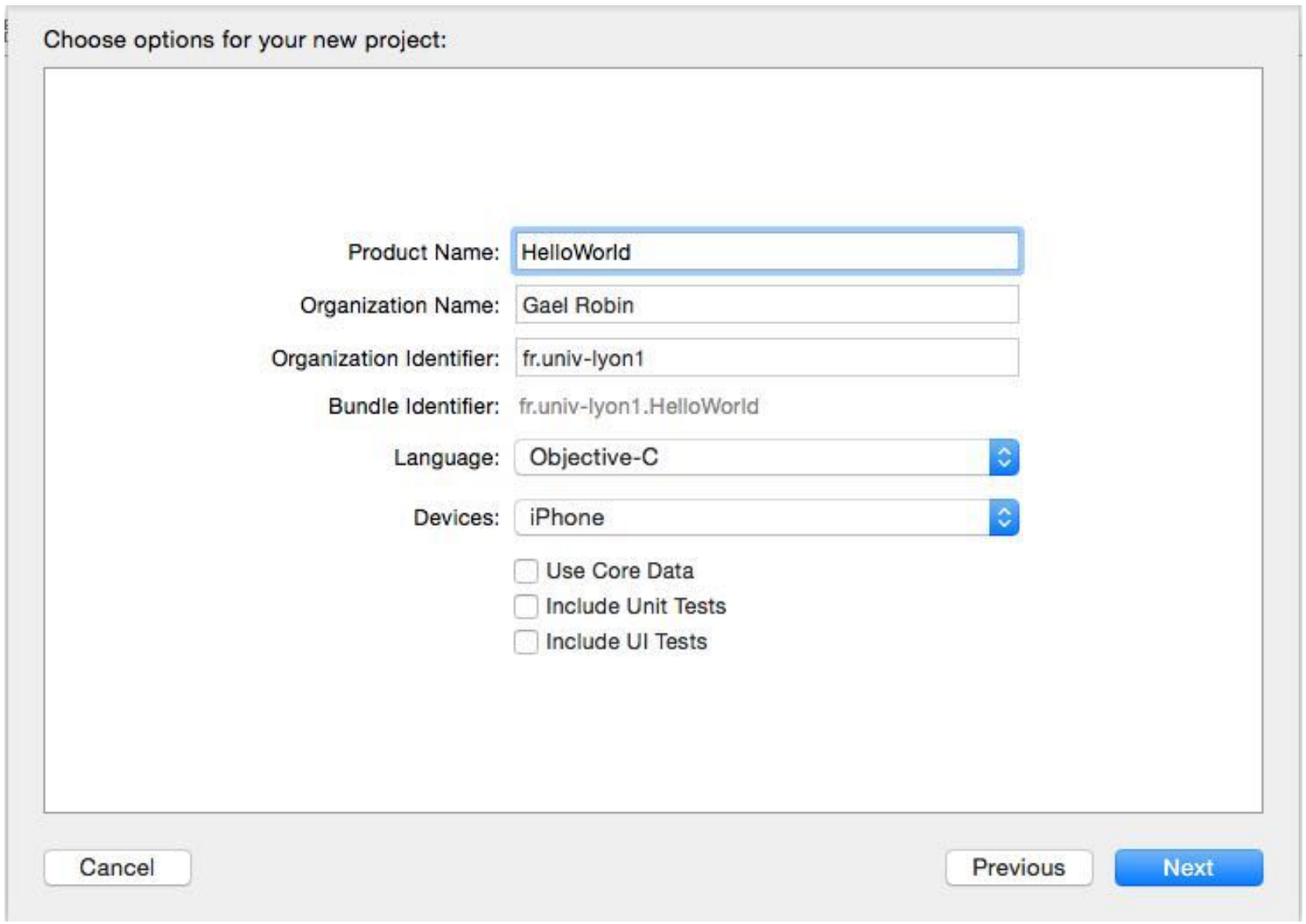
Création d'un projet iPhone

- Lancer Xcode (IDE) et créer un nouveau projet HelloWorld en utilisant le template (Single View Application)



Création d'un projet iPhone

Options de création : iPhone uniquement



- Premier projet iPhone
  - Repérer le main et le modifier pour ajouter un affichage de la chaîne de texte “Hello World !” sur la sortie console

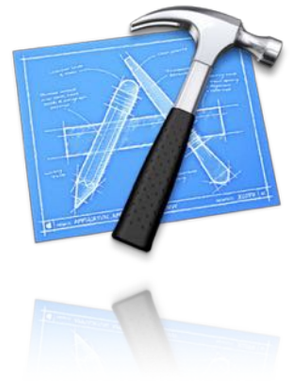
```
NSLog(@"Hello World !");
```

  - Repérer le fichier “AppDelegate.m”
    - Ajouter un message dans la méthode

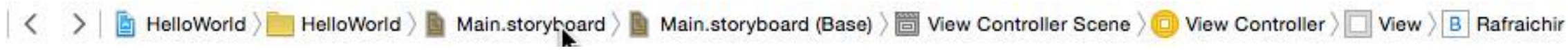
```
- (BOOL)application:(UIApplication *)application
didFinishLaunchingWithOptions:(NSDictionary
*)launchOptions
```

    - Tester

- Raccourcis Xcode
  - Notations pour les touches spéciales MAC
    - ⌘ (Command ou)
    - ^ (Control)
    - ⌥ (Option ou alt)
    - ⇧ (Shift)
  - Basculer entre le fichier d'en-tête .h et le fichier d'implémentation .m : ^⌘↑
  - Build ⌘B
  - Run ⌘R (compile le projet si besoin)



- Xcode
  - Espaces de travail
    - ⌘ T : Créer un onglet
  - Navigation
    - Panneau Project Navigator
    - Fil d'Ariane (*breadcrumb*)



    - ⌘ ⇧ O : Quick open
    - ⌘ clic sur un élément (variable, méthode, fichier .h, …) pour aller à sa définition
    - ⌘ ⇧ O : Library (composants d'interface utilisateur)

- Xcode
  - Multi-curseurs
    - ^ ⇧ Clic : ajouter un nouveau curseur
    - ⌥ Clic Glissé : sélection en colonne
    - ^ ⇧ Up/Down : sélection en colonne (au clavier)

- Raccourcis Xcode
  - Complétion automatique
    - Accepter la complétion : ↩
    - Accepter le morceau de nom jusqu'à la majuscule suivante : ⇥ (complétion partielle)
    - Utilisation de la notation Camel Case (majuscules pour séparer les mots)
  - Afficher la liste des choix disponibles : esc ou ^ + espace

```
-(IBAction)refresh{
    NSDate *now = [[NSDate alloc] initWithTimeIntervalSinceNow:0.0];
    NSString *newLabel = [[NSString alloc] initWithF];
    label.text = newLabel;
}

/*
// The designated initializer. Overri
- (id)initWithNibName:(NSString *)nib
    if ((self = [super initWithNibNam
        // Custom initialization
```

```
M id initWithFormat:(NSString *)format arguments:(va_list)argList
M id initWithFormat:(NSString *)format
M id initWithFormat:(NSString *)format locale:(id)locale
M id initWithFormat:(NSString *)format locale:(id)locale arguments:(va_list)argList
NSString *                                                    π
```

  - Passer à l’argument suivant : ⇥

- Xcode
  - Obtenir de la documentation sur les méthodes
    - ⌥ clic sur le nom d'une méthode pour une aide contextuelle

```
NSString *newLabel = [[NSString alloc] initWithFormat:@"%@" arguments:now];
label.text = newLabel;
```

initWithFormat:arguments: (NSString)

Abstract: Returns an NSString object initialized by using a given format string as a template into which the remaining argument values are substituted according to the user's default locale.

Declaration: `- (id)initWithFormat:(NSString *)format arguments:(va_list)argList`

Parameters:

`format`: A format string.

`argList`: A list of arguments to substitute into format.

Return Value: An NSString object initialized by using format as a template into which the values in argList are substituted according to the user's default locale.

Related Documents: Property List Programming Guide, String Programming Guide

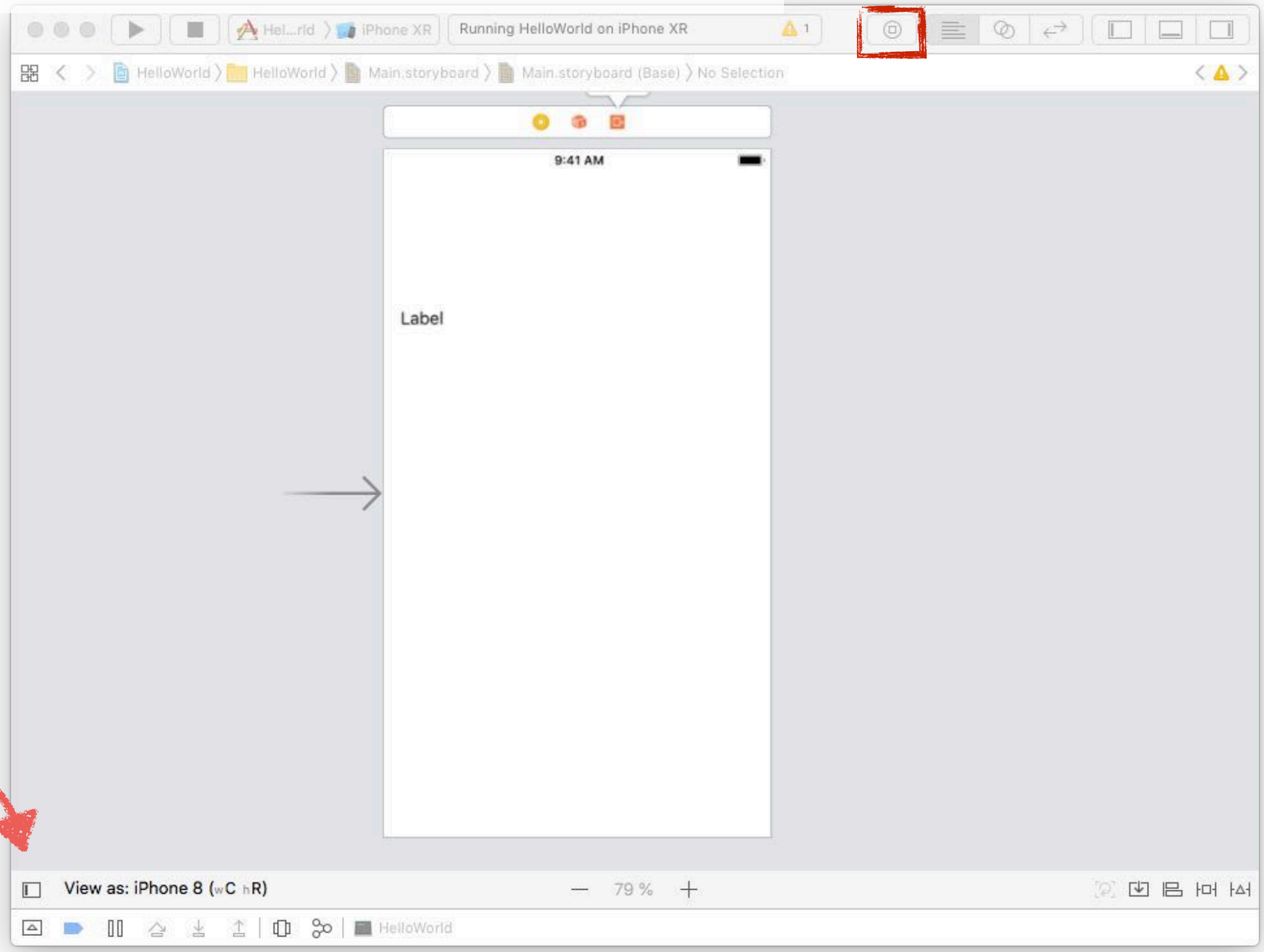
Related API: init, initWithBytes:length:encoding:, initWithBytesNoCopy:length:encoding:freeWhenDone:, initWithCString:, initWithCString:encoding:, initWithCString:length:, initWithCStringNoCopy:length:freeWhenDone:, initWithCharacters:length:, initWithCharactersNoCopy:length:freeWhenDone:, initWithData:encoding:, initWithFormat:, initWithFormat:locale:, initWithFormat:locale:arguments:, initWithString:, initWithUTF8String:,

Aide plus complète

- Premier projet iPhone
  - Création d'une interface graphique avec un bouton (`UIButton`) et un label (`UILabel`)
  - Ouvrir le storyboard

Premier projet iPhone

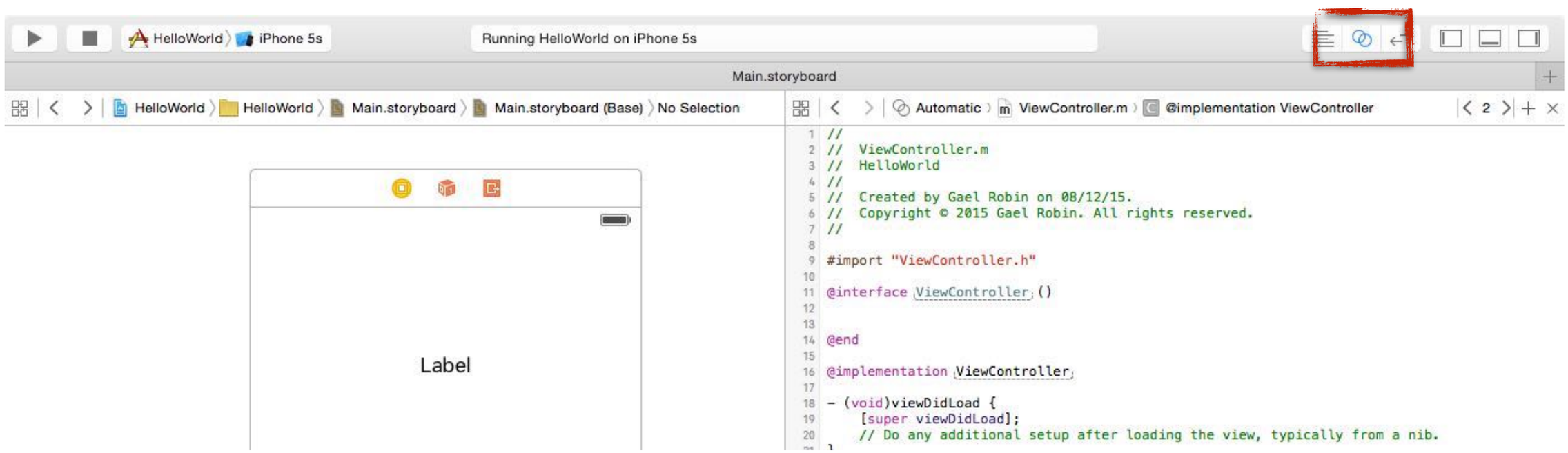
Library



Document outline

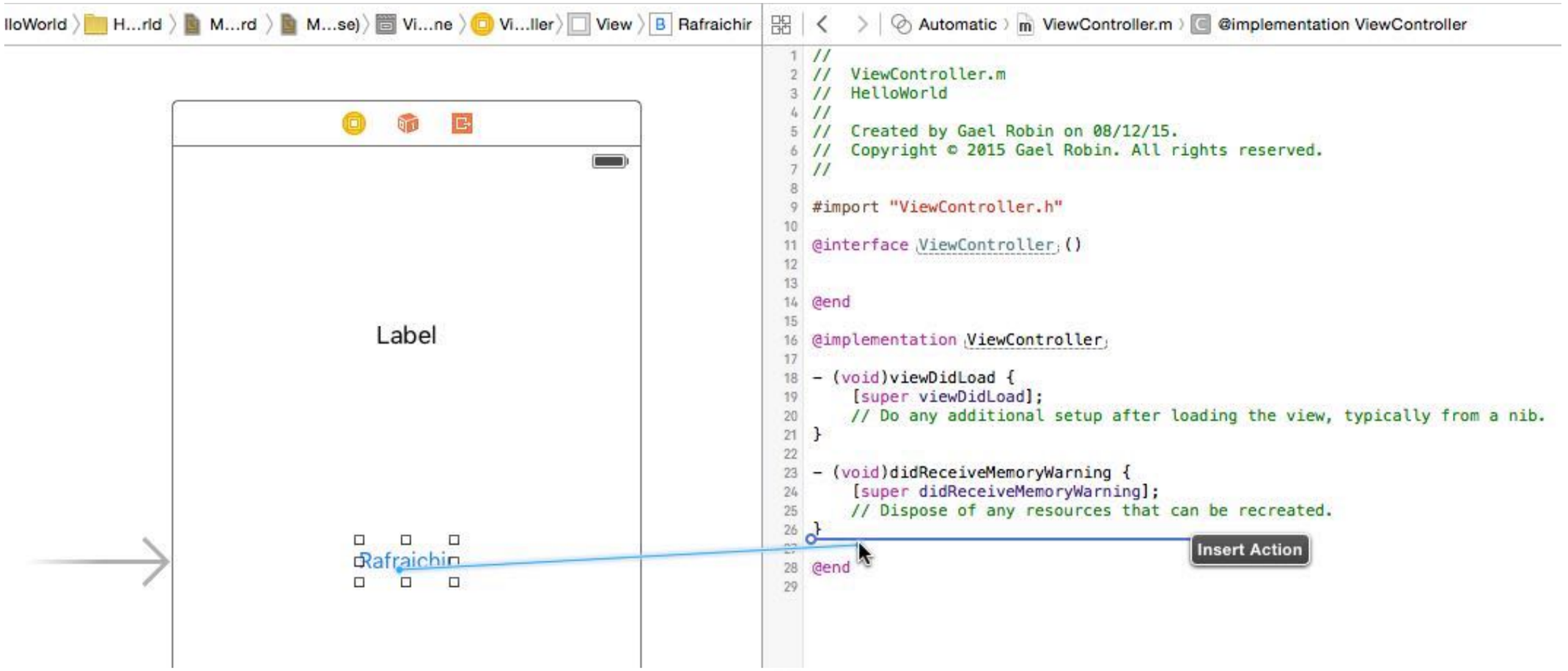
- Premier projet iPhone
  - Faire glisser un label et un bouton
    - Élargir le label jusqu'aux guides latéraux
    - Régler les attributs pour centrer le texte
    - Modifier le texte du bouton pour qu'il affiche "Rafraichir", ajuster sa largeur et le centrer
  - Lancer l'application dans le simulateur

- Premier projet iPhone
  - Lier l'interface graphique et le code
    - Pour modifier le texte du label il faut posséder une référence vers l'objet de l'interface graphique
    - Il faut également lier l'action associée à l'appui sur le bouton à une méthode d'un objet
    - Sélectionner le storyboard et cliquer sur l'icône permettant d'invoquer l'assistant



- Premier projet iPhone
  - Lier l’interface graphique et le code
    - Chaque écran de l'application apparaissant sur le storyboard est géré par un contrôleur de vue qui hérite de la classe `UIViewController`
    - Le template "Single view application" ne définit qu'une seule vue qui est gérée par une instance de la classe `ViewController`

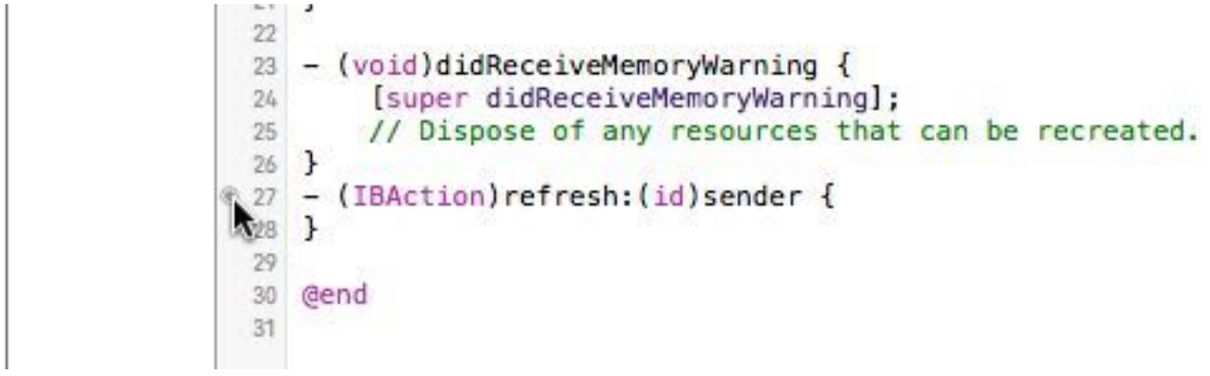
- Premier projet iPhone
  - Lier l’interface graphique et le code
    - Les liens s'effectuent en faisant un clic droit (ou ^ clic) sur l'élément de départ (bouton par exemple) et en glissant vers le code
    - Lier l'action du bouton en ciblant la zone @implementation du fichier ViewController.m



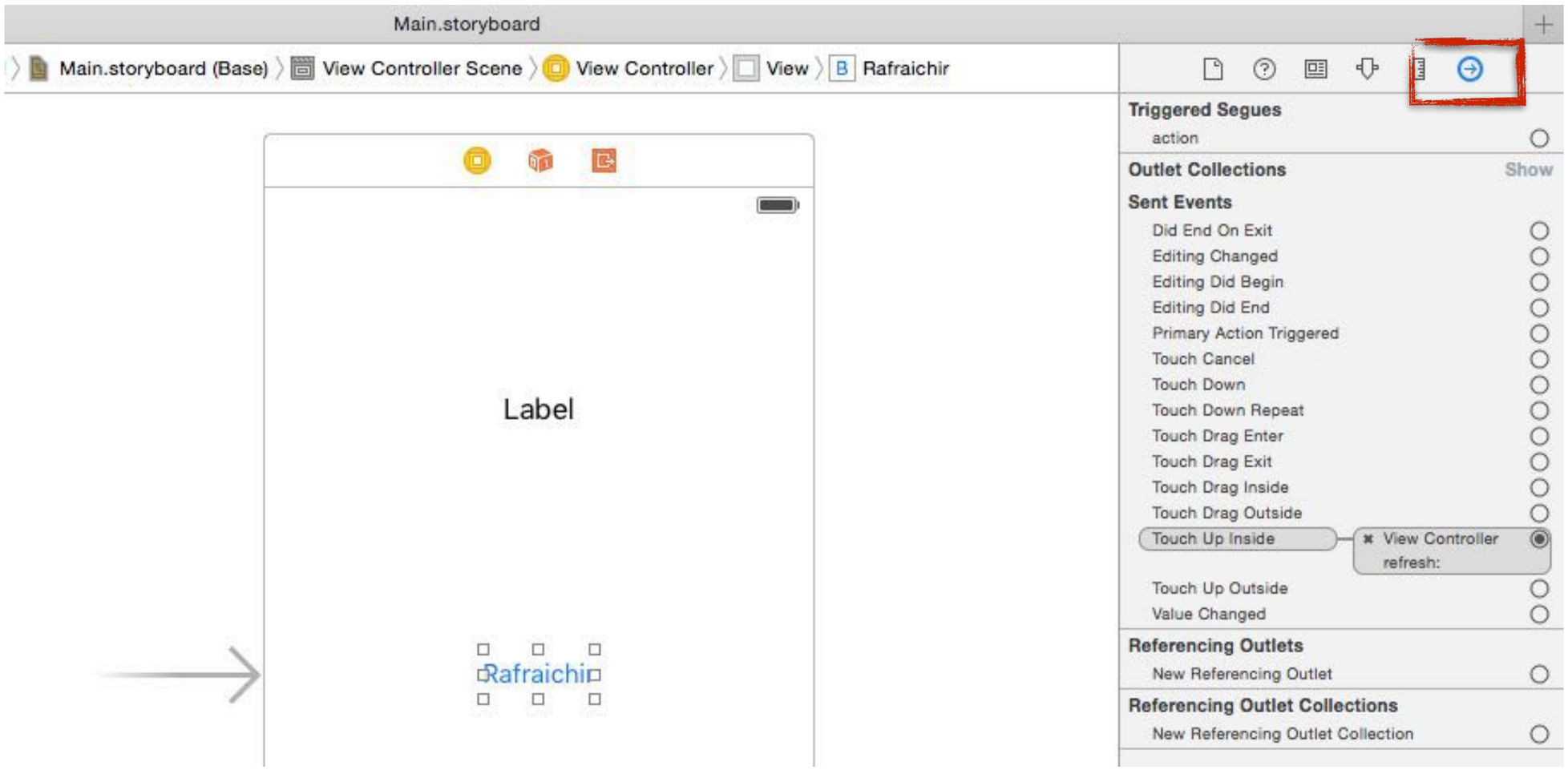

```
//
//  ViewController.m
//  HelloWorld
//
//  Created by Gael Robin on 08/12/15.
//  Copyright © 2015 Gael Robin. All rights reserved.
//

#import "ViewController.h"

@interface ViewController ()


@end

@implementation ViewController

- (void)viewDidLoad {
    [super viewDidLoad];
    // Do any additional setup after loading the view, typically from a nib.
}

- (void)didReceiveMemoryWarning {
    [super didReceiveMemoryWarning];
    // Dispose of any resources that can be recreated.
}
@end
```

- Premier projet iPhone
  - Lier l'interface graphique et le code
    - Vérifier que le type de connexion correspond bien à une Action
    - Saisir le nom de la méthode qui sera invoquée lors du clic sur le bouton (`refresh`)

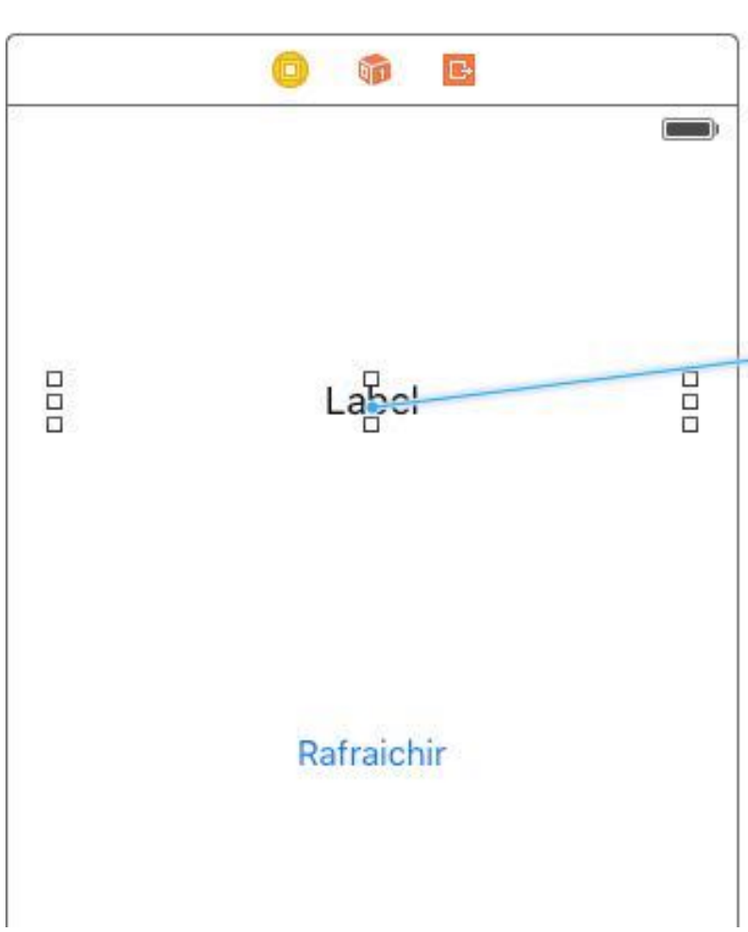
- Premier projet iPhone
  - Lier l'interface graphique et le code
    - Vérifier les connexions
      - En survolant le point en regard de la méthode



- Premier projet iPhone
  - Lier l'interface graphique et le code
    - Vérifier les connexions
      - Dans l'inspecteur des connexions



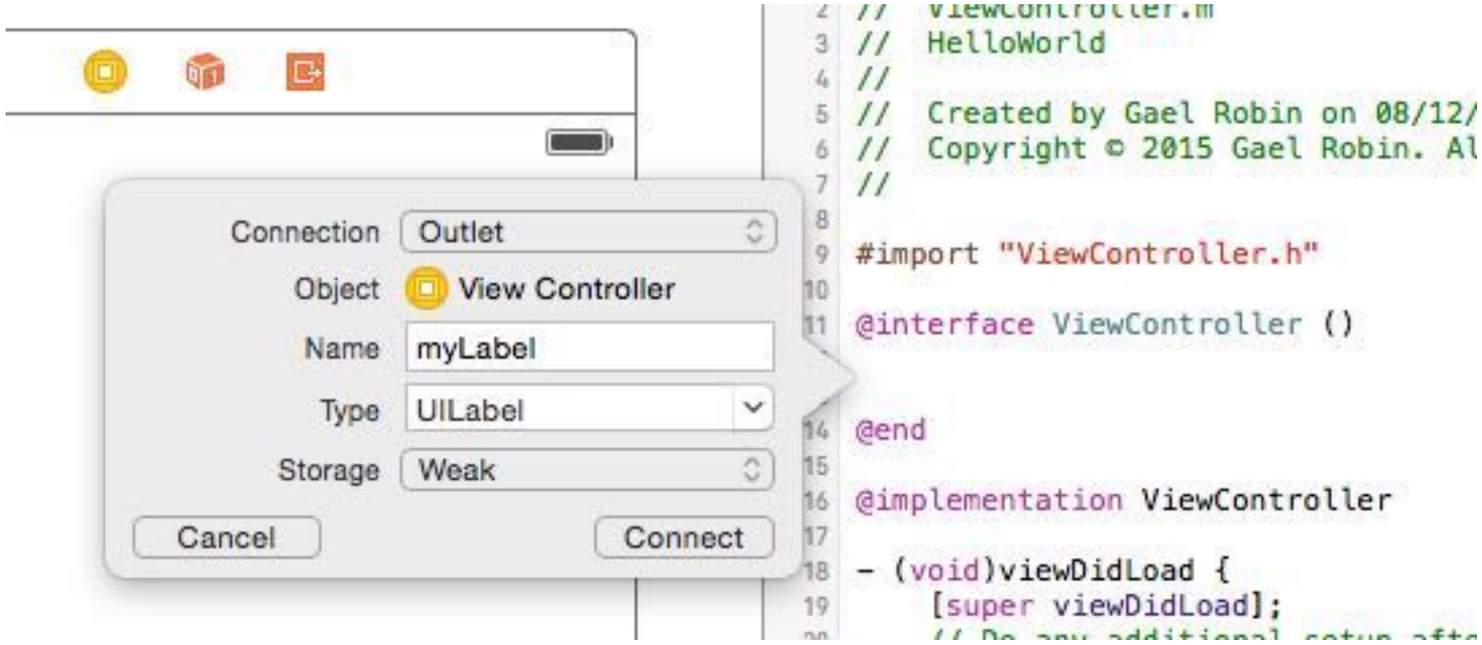
- Premier projet iPhone
  - Lier l'interface graphique et le code
    - Lier le label à une variable d'instance privée




```
1  //
2  //  ViewController.m
3  //  HelloWorld
4  //
5  //  Created by Gael Robin on 08/12/15.
6  //  Copyright © 2015 Gael Robin. All rights reserved.
7  //
8
9  #import "ViewController.h"
10
11 @interface ViewController ()
12
13
14 @end
15
16 @implementation ViewController
17
18 - (void)viewDidLoad {
19     [super viewDidLoad];
20     // Do any additional setup after loading the view, typically from a nib.
21 }
22
23 - (void)didReceiveMemoryWarning {
24     [super didReceiveMemoryWarning];
25     // Dispose of any resources that can be recreated.
26 }
27
28 - (IBAction)refresh:(id)sender {
29 }
30
31 @end
32
```

Insert Outlet or Outlet Collection

- Premier projet iPhone
  - Lier l'interface graphique et le code
    - Lier le label à une variable d'instance privée
    - Vérifier que le type de connexion correspond bien à un Outlet
    - Saisir le nom de la propriété qui stockera une référence vers l'objet label (`myLabel`)



Propriétés "privées"

- Premier projet iPhone
  - Implémentation de l'action déclenchée par l'appui sur le bouton

```
#import "ViewController.h"

@interface ViewController ()

@property (weak, nonatomic) IBOutlet UILabel *myLabel;

@end

@implementation ViewController

- (void)viewDidLoad {
    [super viewDidLoad];
}

- (void)didReceiveMemoryWarning {
    [super didReceiveMemoryWarning];
}

- (IBAction)refresh:(id)sender {
    [self.myLabel setText:@"Hello world!"];
}

@end
```

ViewController.m

- Premier projet iPhone
  - Modifier la méthode précédente pour afficher la date et l'heure au format : 04/10/2010 23:30:45
  - Utiliser les classes suivantes
    - `NSDate`

```
NSDate *now = [[NSDate alloc] initWithTimeIntervalSinceNow:0.0];
```

    - `NSDateFormatter` (se référer à la documentation)
  - Tester l'application

- Syntaxe Objective C
  - Création d’un objet
    - Utilisation de méthodes statiques “convenience constructors” qui regroupent l’allocation et l’initialisation
    - Modifier le code pour utiliser
      - NSDate

```
NSDate *now = [NSDate date];
```

Syntaxe Objective C - Anatomie d'une classe - Synthèse

```objc
//  SampleClass.h

#import <Foundation/Foundation.h>

/* @class remplace #import pour déclarer l'existence d'une classe sans avoir à inclure la totalité du fichier .h */
@class SomeOtherClass;

/*
 Pas de véritable définition de variables dans cette zone (c'est techniquement faisable si vous savez ce que vous faites mais il faut l'éviter).
 On peut définir ici :
    - des types : typedef
    - des enums
    - des externs : les externs ressemblent à des définitions de variables mais sont en réalité une promesse que cette variable sera effectivement déclarée ailleurs dans le programme (dans un autre fichier). En objective-C, cela doit être utilisé seulement pour déclarer des constantes (en générales des chaînes de caractères comme les clés utilisées dans des tableaux associatifs NSDictionary, ou des identifiants de notifications).
*/

extern NSString * const MYSomethingHappenedNotification;

@interface SampleClass : NSObject
{
    // Historiquement : déclarations de variables d'instance.
    // Obsolète : Ne rien mettre ici.
}

// Propriétés publiques (chaque ligne @property définit un getter et éventuellement un setter)
@property (readonly, nonatomic, strong) SomeOtherClass *someProperty;

// Méthodes de classe (équivalent de static en Java)
+ (id)someClassMethod;

// Méthodes d'instance
- (SomeOtherClass *)doWork;

@end
```

```objc
//  SampleClass.m
#import "SampleClass.h"
#import "SomeOtherClass.h"

/* Constantes externes (variables globales). */
NSString * const MYSomethingHappenedNotification =
@"MYSomethingHappenedNotification";
NSString * const kRPErrorDomain = @"com.myIncredibleApp.errors";

/* Variables de classe - il s'agit en fait d'une variable C globale mais visible seulement depuis le fichier dans lequel elle est définie */
static int count = 0;

// Class extension
@interface SampleClass (){
    //Variables d'instance privées (rare)
    int somePrivateInt;
}
/* Méthodes et propriétés privées
Re-declaration d'une version en lecture/écriture d'une propriété publique en lecture seule -> génère un getter et un setter dont seul le getter sera visible après import du .h */
@property (readwrite, nonatomic, strong) SomeOtherClass *someProperty;

@end

@implementation SampleClass
{
    // Variables d'instance privées
    /* A éviter. Préférer la synthèse automatique des accesseurs et de la variable d'instance associée à une propriété. Les exceptions sont généralement pour des variables non-ObjC (objets Core Foundation ou C++). */
}

// Implémentations des méthodes définies dans les interfaces publiques et privées
// #pragma mark - permet d'organiser le code
#pragma mark - Class Methods

+ (id)someClassMethod
{
    return [[SampleClass alloc] init];
}

#pragma mark - Instance Methods

- (SomeOtherClass *)doWork
{
    // Implement this
    return [[SomeOtherClass alloc] init];
}

@end
```

- Syntaxe Objective C - Liens utiles
  - Cheat sheet
    - https://github.com/iwasrobbed/Objective-C-CheatSheet#properties-and-variables
  - Déclarations des variables (instance / globales / …)
    - https://stackoverflow.com/questions/12632285/declaration-definition-of-variables-locations-in-objectivec
    - https://stackoverflow.com/questions/15762259/declaring-variable-inside-and-outside-of-implementation
  - Cocoa Coding guidelines
    - https://developer.apple.com/library/content/documentation/Cocoa/Conceptual/CodingGuidelines/CodingGuidelines.html

- Syntaxe Objective C - Variables d'instances
  - Accès aux variables d'instance depuis d'autres classes
    - Approche recommandée : accesseurs et mutateurs (getters and setters)
      - Convention de nommage :
        - L'accesseur est nommé en utilisant le nom de la valeur retournée
        - Le mutateur commence par "set" et est suivi du nom de la valeur retournée (avec la première lettre mise en majuscule)

```
- (UInt8)age;
- (void)setAge:(UInt8)age;
- (NSString *)name;
- (void)setName:(NSString *)value;
```

        - L'accesseur est préfixé par "get" lorsqu'il retourne un pointeur sur une variable qui n'est pas un objet (utile lorsque la méthode retourne plusieurs valeurs)

```
- (void)getRed:(CGFloat *)red green:(CGFloat *)green blue:(CGFloat *)blue alpha:(CGFloat *)alpha;
```

Syntaxe Objective C - Variables d'instances

Convention de nommage des accesseurs

- Plus qu'une simple convention de nommage :
  - Base des mécanismes de Key-Value Coding et Key-Value Observing qui permettent de mettre simplement en oeuvre le design pattern "Observateur", l'intégration des objets dans le framework de persistance Core Data, ...
  - Utilisé par Interface Builder pour créer graphiquement des liens entre objets

- Syntaxe Objective C - Variables d'instances
  - Avantages des accesseurs
    - Flexibilité de l'implémentation
      - les propriétés peuvent être stockées dans des variables d'instances, ou dans une base de données, calculées en utilisant d'autres champs, récupérées sur un serveur, ...
      - L'utilisateur de la classe n'a pas besoin de connaître les détails (meilleure encapsulation)
      - L'implémentation peut-être modifiée sans impacter les utilisateurs
    - Maintenance allégée
      - Tous les accès à la variable se font à travers les accesseurs ➪ limite le nombre d'endroits où le code doit être modifié en cas de changement des propriétés

- Syntaxe Objective C - Variables d'instances
  - Avantages des accesseurs
    - Gestion "centralisée" de la mémoire (mécanisme de comptage de références)
      - Restriction de la gestion de la mémoire aux accesseurs ➪ facilite l'application des conventions de Cocoa et limite la gestion de la mémoire à quelques méthodes isolées
    - Les mutateurs (setters) simplifient le débuggage
      - Si une propriété prend une valeur incorrecte ou suspecte, il est facile de mettre un point d'arrêt dans la seule méthode susceptible de modifier cette propriété (le mutateur `setMaVariable`) pour identifier le code en cause
    - Les mutateurs (setters) permettent de contraindre la plage des valeurs autorisées ou de déclencher une mise à jour lors de la modification d'une propriété (ex : signaler au framework graphique qu'il faut rafraichir l'affichage lors de la modification d'une propriété de couleur, ...)

- Syntaxe Objective C - Variables d’instances
  - Avantages des accesseurs
    - Exemple
    - Situation de départ
      - La propriété age de la classe Person est stockée dans une variable d’instance de type UInt8

```
@interface Person : NSObject

@property(nonatomic) UInt8 age;
@end
```

```
@implementation Person
//générée automatiquement
@end
```

- Syntaxe Objective C - Variables d'instances
  - Avantages des accesseurs
    - Modification de la classe
      - Stockage de la date de naissance plutôt que de l'age

```
@interface Person : NSObject
@property(nonatomic) UInt8 age;
@property(nonatomic) NSDate *birthDate;
@end

@implementation Person
//...
-(UInt8)age
{
   NSDate *now = [NSDate date];
   double timeInterval = [now timeIntervalSinceDate:birthDate]/(365.25*24*3600);
   return (UInt8)(timeInterval);
}
//...
@end
```

      - Les utilisateurs utilisent toujours : `[test age]`

Syntaxe Objective C - Propriétés

- Déclaration de propriétés avec la directive `@property`
  - Doit figurer dans la liste des méthodes déclarées dans l'interface (publique ou privée) : équivaut à déclarer les accesseurs
- Possibilité de synthétiser automatiquement le code des accesseurs
  - Par utilisation de `@synthesize`
  - Implicitement : Xcode synthétise les accesseurs et peut également synthétiser les variables d'instance si elle n'ont pas été déclarées
- On peut combiner la synthèse avec l'implémentation de l'un des accesseurs ➪ seuls les accesseurs non trouvés dans l'implémentation seront générés automatiquement

Syntaxe Objective C - Propriétés

```
@interface HelloWorldViewController : UIViewController

@property(nonatomic, strong) IBOutlet UILabel *label;

-(IBAction)refresh:(id)sender;

@end
```

Déclaration d'une propriété / synthèse des accesseurs et de la variable d'instance

```
@implementation HelloWorldViewController

@synthesize label;

-(IBAction)refresh:(id)sender{

   label.text = @"Hello World !";
}
```

Syntaxe Objective C - Propriétés

```
@interface HelloWorldViewController : UIViewController

@property(nonatomic, strong) IBOutlet UILabel *label;

-(IBAction)refresh:(id)sender;

@end
```

Déclaration d'une propriété / synthèse des accesseurs et de la variable d'instance avec un nom différent

```
@implementation HelloWorldViewController

@synthesize label = _label;

-(IBAction)refresh:(id)sender{

   _label.text = @"Hello World !";
}
```

Syntaxe Objective C - Propriétés

```
@interface HelloWorldViewController : UIViewController {
}

@property(nonatomic, strong) IBOutlet UILabel *label;

-(IBAction)refresh:(id)sender;

@end
```

Déclaration d'une propriété / synthèse automatique des accesseurs et de la variable d'instance


```
@implementation HelloWorldViewController


-(IBAction)refresh:(id)sender{

   _label.text = @"Hello World !";
}
```

Le compilateur ajoute automatiquement : `@synthesize label = _label;`

- Syntaxe Objective C - Variables d'instances
  - "Dot syntax"
    - Accès aux propriétés d'un objet à l'aide d'un "."
      - Provoque un appel aux accesseurs de la propriété
      - Les deux codes ci-dessous sont équivalents

```
NSColor *color = graphic.color;
CGFloat xLoc = graphic.xLoc;
BOOL hidden = graphic.hidden;
int textCharacterLength = graphic.text.length;
if (graphic.textHidden != YES) {
   graphic.text = @"Hello";
}
graphic.bounds = NSMakeRect(10.0, 10.0, 20.0, 120.0);
```

```
NSColor *color = [graphic color];
CGFloat xLoc = [graphic xLoc];
BOOL hidden = [graphic hidden];
int textCharacterLength = [[graphic text] length];
if ([graphic isTextHidden] != YES) {
   [graphic setText:@"Hello"];
}
[graphic setBounds:NSMakeRect(10.0, 10.0, 20.0, 120.0)];
```

- Gestion de la mémoire - Historique
  - Allocation par la méthode `alloc`
  - Libération par la méthode `dealloc`
    - Cette méthode ne doit JAMAIS être appelée directement
  - Difficulté commune à tous les langages et environnement
    - Comment savoir à quel moment un objet doit être libéré ?
    - Un même objet peut être utilisé par plusieurs autres sans que ceux-ci n'en ait connaissance
  - Solution : utilisation d'un compteur de références ("retain count")
    - Initialisé à 1 lors de l'allocation (`[monObjet alloc]`)
    - Lorsqu'un objet n'est plus utilisé, il faut lui envoyer le message `release`

- Gestion de la mémoire
  - Retain count
    - Lors de la réception d'un message `release`, le compteur de référence est décrémenté et s'il tombe à zéro, le système appellera la méthode `dealloc` de l'objet
    - La méthode `dealloc` d'un objet doit relâcher tous les objets qui étaient référencés par l'objet désalloué

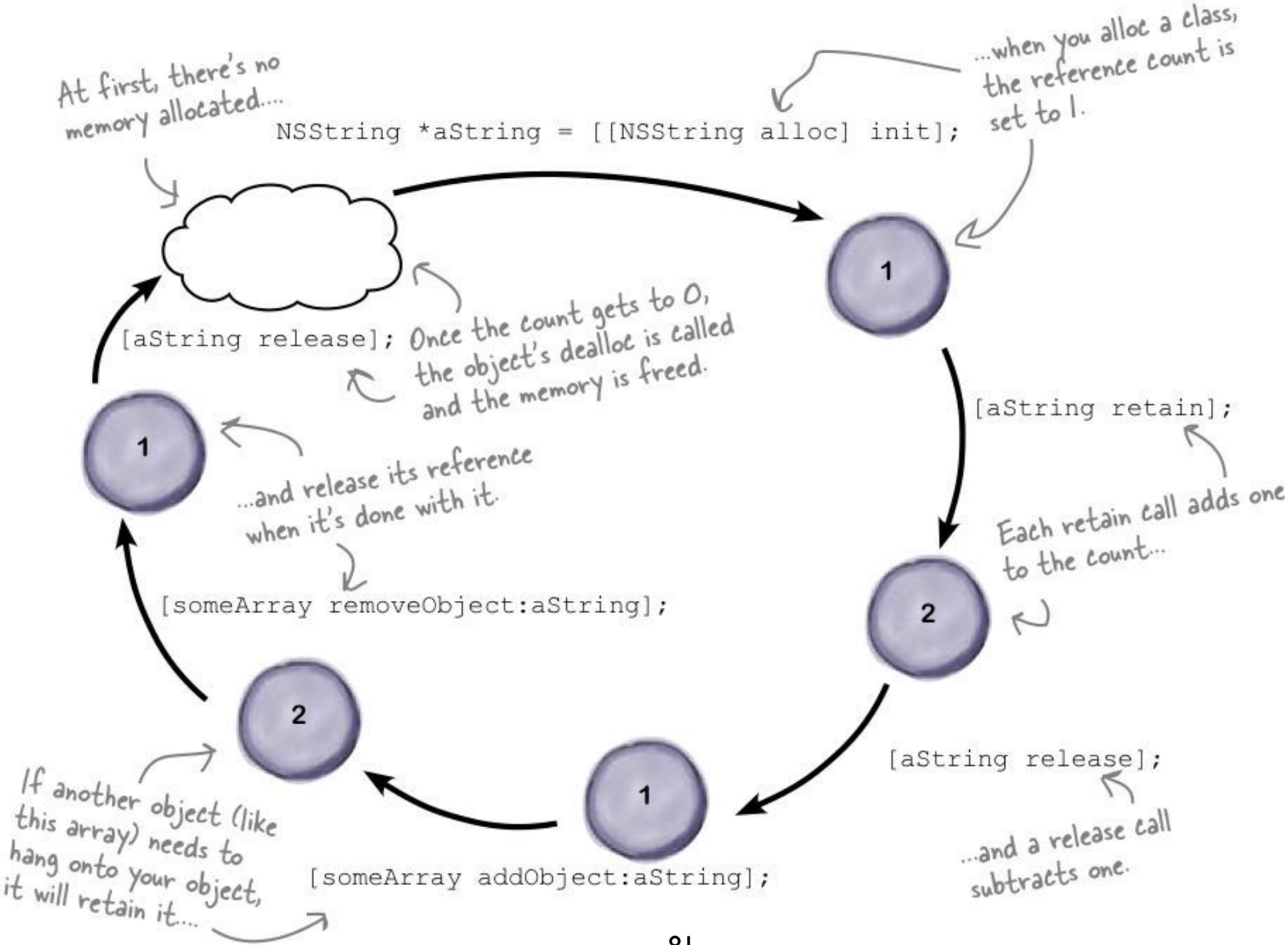
```
- (void)dealloc {
    [name release];
    [super dealloc];
}
```

- Pour s'assurer qu'un objet alloué par "quelqu'un d'autre" alors que l'on en a encore besoin il faut lui envoyer le message `retain` qui incrémente le compteur de références

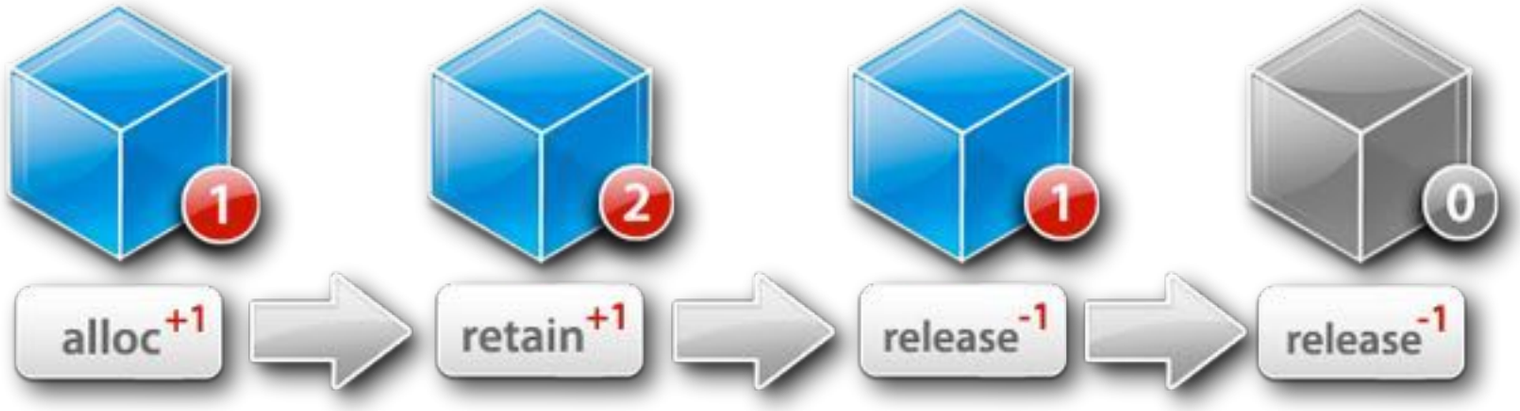
```
-(void)setName:(NSString *)newName{
   [newName retain];
   [_name release];
   _name = newName;
}
```

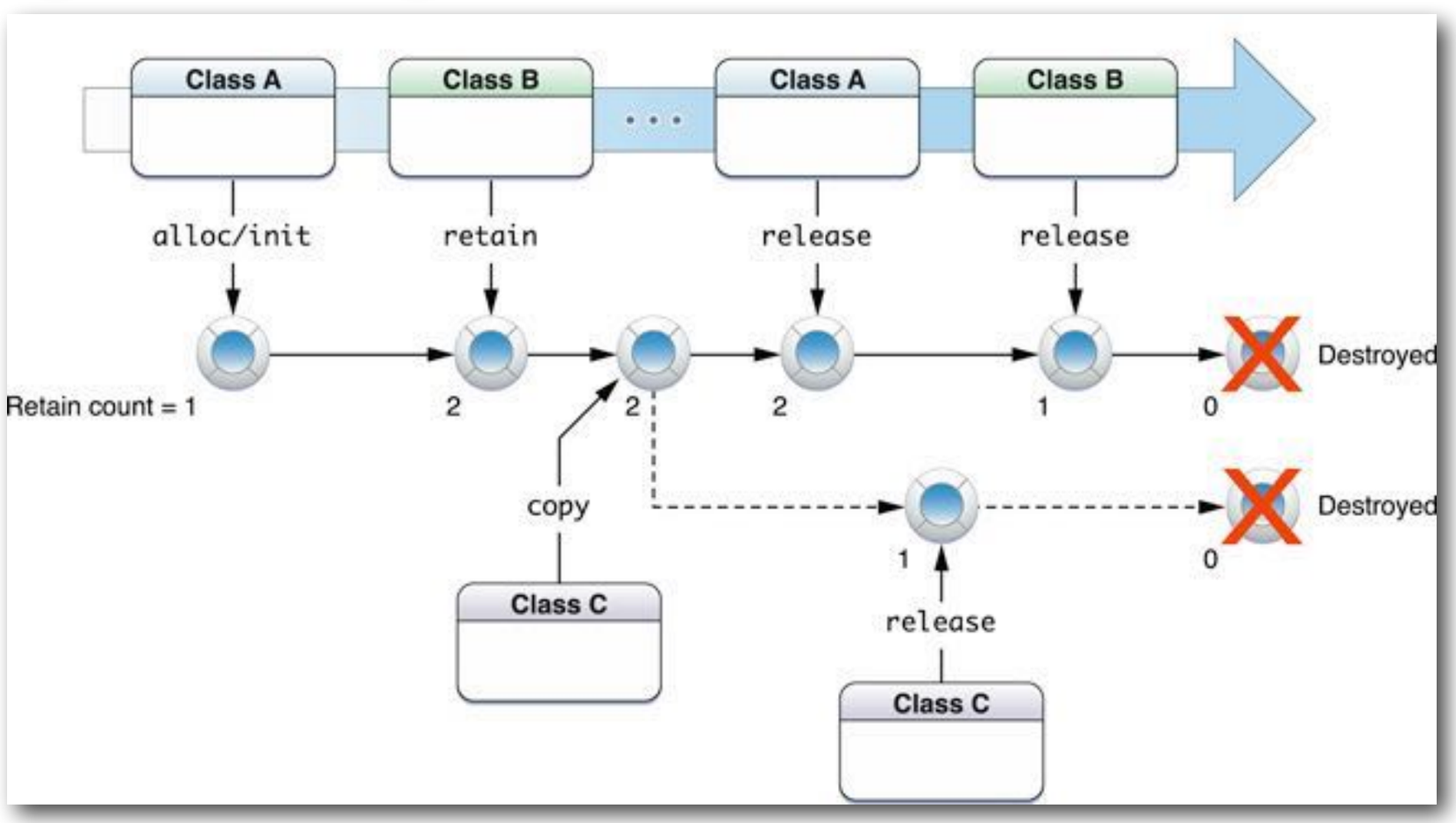
Pourquoi l'ordre de ces opérations ne doit pas être inversé ?

Gestion de la mémoire



Gestion de la mémoire - Résumé





- Gestion de la mémoire - Règle fondamentale
  - Envoyer `release` or `autorelease` aux seuls objets que vous possédez
    - Vous possédez un objet
      - si vous le créez en utilisant une méthode qui commence par “`alloc`”, “`new`” ou qui contient “`copy`” (par exemple, alloc, newObject, or mutableCopy),
      - ou si vous lui avez envoyé un message `retain`
  - Vous utilisez `release` ou `autorelease` pour abandonner la propriété d’un objet. `autorelease` signifie juste “envoie un message `release` dans le futur”

Gestion de la mémoire

- Si vous avez besoin de stocker un objet reçu (argument d'un message) dans une variable d'instance vous devez lui envoyer `retain` ou le copier.

- Un objet reçu en argument est normalement garanti de rester valide à l'intérieur de la méthode dans laquelle il a été reçu (les exceptions concernent le multithreading ou en cas d'appel explicite à `release` sur l'objet)

- Les objets sont retenus lors de l'ajout dans une classe de collection (ex : `NSArray`) et relâchés lorsqu'ils en sont extraits

Gestion de la mémoire

A quoi sert `autorelase` ?

- “Convenience constructor” de la classe `Person`

```
+(instancetype)createPersonWithAge:(UInt8)newAge {
    Person *person = [[Person alloc] initWithAge:newAge];
    return person;
}
```

- L’utilisateur de cette méthode ne serait pas responsable du relâchement de la référence (`release`) car le nom de la méthode ne contient pas l’un des mots listés dans la règle (préfixe : init, copy ou new)
- La méthode `createPersonWithAge` appelle `alloc` et est donc responsable de l’appel à `release`
- Impossible d’appeler `release` après `return` et si `release` est appelé avant `return` l’objet sera désalloué et la méthode retournera un pointeur sur un objet désalloué.
- Quand appeler `release` ? Plus tard...

- Gestion de la mémoire
  - Autorelease
    - Ici "Plus tard" signifie "après `return`" et suffisamment tard pour que l'appelant est le temps de stocker la valeur retournée dans une variable et d'appeler `retain` dessus si besoin.
    - Solution : `autorelease`

```
+(instancetype)createPersonWithAge:(UInt8)newAge {
    Person *person = [[Person alloc] initWithAge:newAge];
    return [person autorelease];
}
```

    - Mise en oeuvre

```
Person *toto = [Person createPersonWithAge:24];
[toto retain];
```

    - Les objets qui reçoivent le message `autorelease` sont placés dans un pool d'autorelease (collection d'objets qui reçoivent le message `release` à la fin de la boucle de traitement des événements de l'application).
      - Voir la fonction main pour la création du pool d'autorelease

- Gestion de la mémoire
  - Depuis le passage à ARC (Automatic Reference Counting), toute cette mécanique est réalisée automatiquement par le compilateur
  - Il existe quelques situations qui nécessitent une intervention "manuelle"
    - Casser des cycles de références
    - Introduire un pool d'autorelease supplémentaire dans une boucle génératrice de nombreux objets temporaires
      - `@autorelease`

Syntaxe Objective C - Propriétés

- Attributs des propriétés
  `@property(nonatomic, strong)`
  - Le code généré par le mot clé `@synthesize` dépend des attributs spécifiés lors de la déclaration de la propriété
    - Atomicité
      - Par défaut les accesseurs générés sont prévus pour être utilisés de façon concurrente depuis plusieurs thread
      - le code ainsi généré garantit l'atomicité des accès (mise en attente des threads si un thread est déjà "entré" dans la méthode) -> cette protection n'est généralement pas suffisante
      - Surcoût lors des appels aux méthodes (temps de prise et de libération des verrous associés)
      - Utilisation du mot clé `nonatomic` lorsque les accesseurs seront appelés depuis un thread unique -> LE PLUS UTILISE

Syntaxe Objective C - Propriétés

Attributs des propriétés

Noms des accesseurs

- Par défaut les accesseurs pour une propriété

```
@property float value;
```

sont construits comme suit :

```
- (float)value;
- (void)setValue:(float)newValue;
```

- Modification du nom des méthodes (pas recommandé)

```
@property(getter=retournerValue,setter=modifierValue) float value;
```

Syntaxe Objective C - Propriétés

- Attributs des propriétés
  - Caractère modifiable
    - Propriétés accessible en lecture et écriture

```
@property(readwrite) float value;
```

      - C’est la valeur par défaut
    - Propriétés accessible en lecture seule

```
@property(readonly) float value;
```

      - Le bloc d’implémentation ne nécessite que la présence d’un “getter”. Si `@synthesize` est utilisé aucun setter ne sera généré

Syntaxe Objective C - Propriétés

Attributs des propriétés

Modification des setters

`assign`

- La variable passée en paramètre est simplement assignée à la variable d'instance
- Utile pour tous les types non objet (l'assignation est équivalent à une copie)

```
@property(assign) NSInteger value;
```

- `weak` (équivalent d'assign pour les objets) - permet de casser des cycles de références (mis à `nil` automatiquement quand l'objet référencé disparait)

```
@property(weak) id target;
```

Syntaxe Objective C - Propriétés

- Attributs des propriétés
  - Modification des setters
    - `strong`
      - La variable (pointeur sur un objet) est retenue par le setter afin que l'objet ne soit pas désalloué

```
@property(strong) NSString *title;
```

    - `copy`
      - L'argument passé en paramètre est copié
      - La variable d'instance pointera vers un objet différent de celui qui a été passé en paramètre

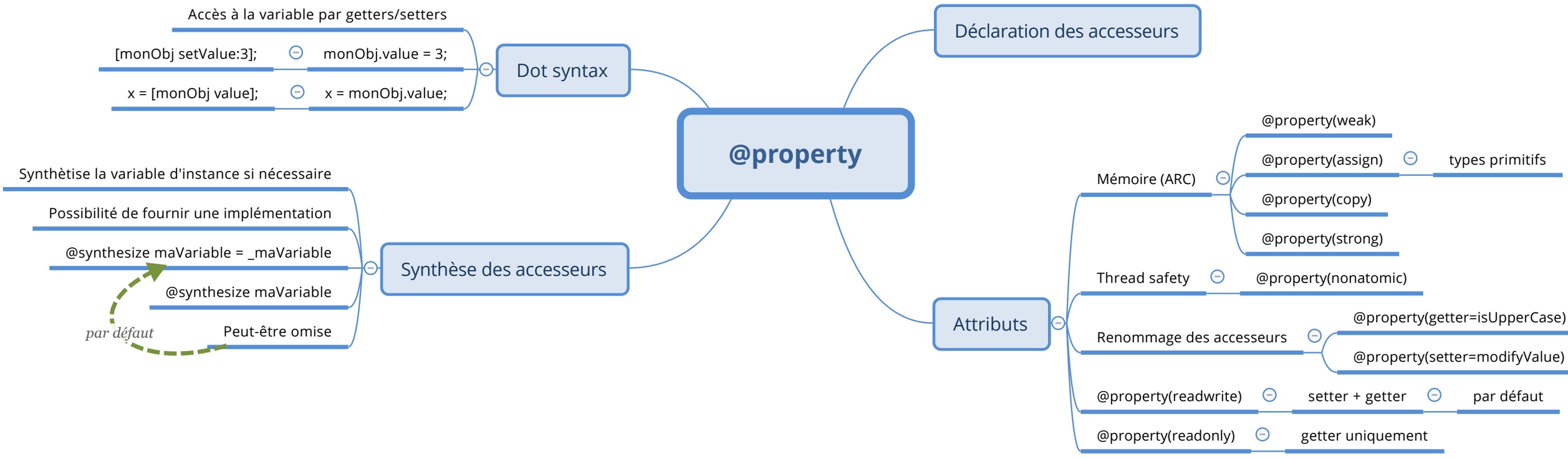
```
@property(copy) NSString *title;
```

- Syntaxe Objective C - Propriétés
  - Attributs des propriétés
    - Modification des setters
      - `copy`
        - La copie est utile quand l'argument n'est pas immuable :

```
NSMutableString *mutableString = [NSMutableString stringWithString:@"initial value"];
[someObject setStringValue:mutableString];
[mutableString setString:@"different value"];
```

        - Si la méthode `setStringValue` se contentait de retenir la valeur passée en argument les modifications opérées sur l'objet `mutableString` se répercuteraient sur la propriété de l'objet `someObject` sans que celui-ci n'en soit informé

Syntaxe Objective C - Propriétés



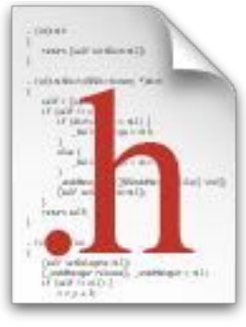
- Syntaxe Objective C - Méthodes privées
  - Il n'existe pas de support direct des méthodes privées au niveau du langage
  - La solution passe par l'utilisation d'une "Category" spéciale appelée "Class Extensions"
  - Une catégorie permet de rajouter des méthodes à une classe existante (y compris à une classe dont on ne possède pas le code source)
    - permet d'étendre les fonctionnalités d'une classe sans avoir recours à l'héritage
    - permet également de répartir le code source d'une classe importante sur plusieurs fichiers ou d'ajouter des fonctionnalités à une classe d'un framework depuis un autre framework

- Syntaxe Objective C - Méthodes privées
  - Category - exemple
    - UITableView (définie dans le framework UIKit) ajoute des propriétés row et section à la classe NSIndexPath (définie dans le framework Foundation) pour manipuler plus facilement cet index dans le contexte des vues de type liste

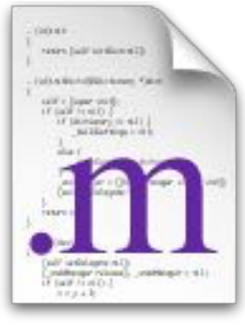
```
// This category provides convenience methods to make it easier to use an
NSIndexPath to represent a section and row
@interface NSIndexPath (UITableView)

+ (NSIndexPath *)indexPathForRow:(NSInteger)row inSection:(NSInteger)section;

@property(nonatomic,readonly) NSInteger section;
@property(nonatomic,readonly) NSInteger row;

@end
```

- Syntaxe Objective C - Méthodes privées
  - Class extension (catégorie anonyme)
    - A la différence d'une catégorie classique le compilateur vérifie que les méthodes déclarées dans la catégorie sont implémentées

```
@interface MyClass : NSObject

@property (readonly) float value;

@end
```



```
@interface MyClass ()

@property (readwrite) float value;

@end

@implementation MyClass

...

@end
```



Exercice

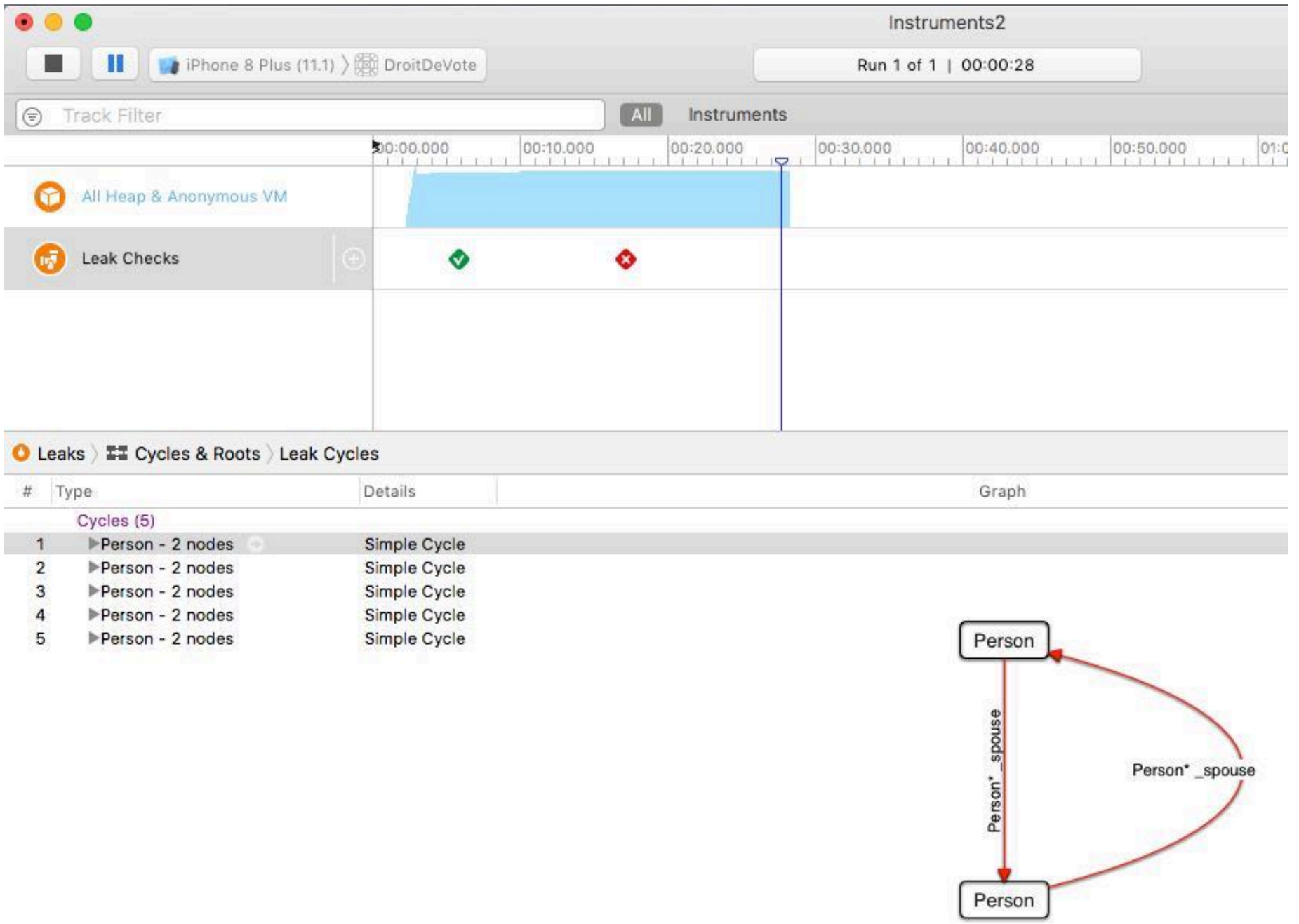
- Créer une application dont l'interface graphique présente un "slider" et 4 labels
  - Le slider permet de modifier l'âge d'un objet de type Person
  - Le label de l'âge et du droit de vote doivent se mettre à jour en temps réel
  - L'âge ne peut prendre que des valeurs entières
  - Dans un deuxième temps l'âge ne pourra prendre que des valeurs multiples de 5 et le curseur devra se trouver à la position correspondante

Person
NSString *name; UInt8 age;
- (instancetype) initWithAge:(UInt8)newAge; + (instancetype) createPersonWithAge:(UInt8)newAge; - (BOOL)canLegallyVote; - (void) vote;

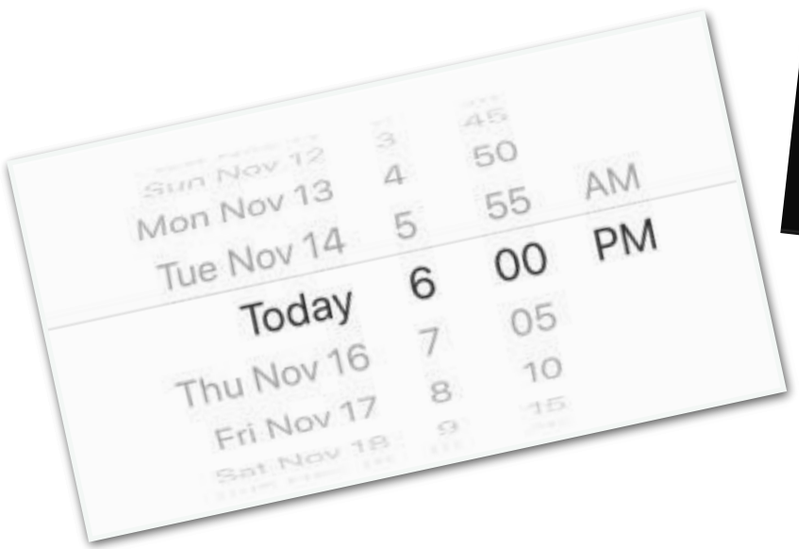
Penser à vérifier les fuites mémoire

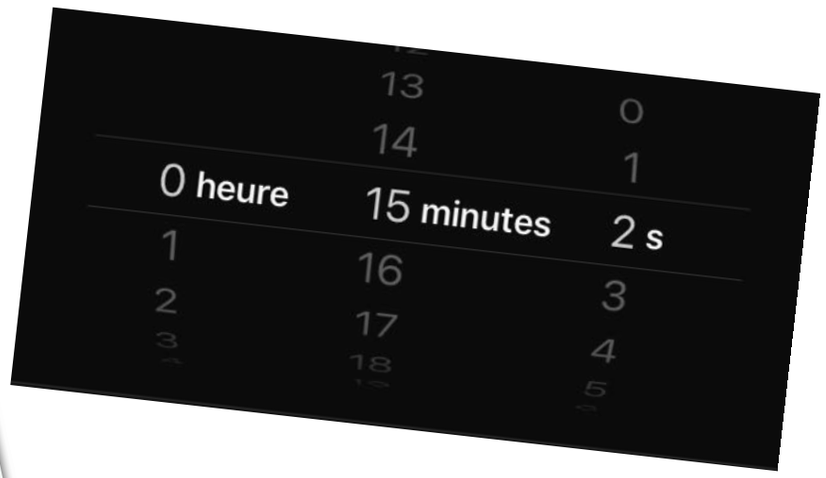
- Exercice
  - Créer un nouveau projet et réutiliser la classe `Person`
  - Ajouter une propriété `spouse` de type `Person` à la classe `Person`
    - Donner l'attribut `strong` à cette propriété
  - Créer une vue contenant un bouton qui crée 2 personnes mariées l'une à l'autre à chaque appui
  - Analyser les fuites mémoires (afficher les "retain cycles ») en lançant Profile depuis le menu Product et en choisissant leaks

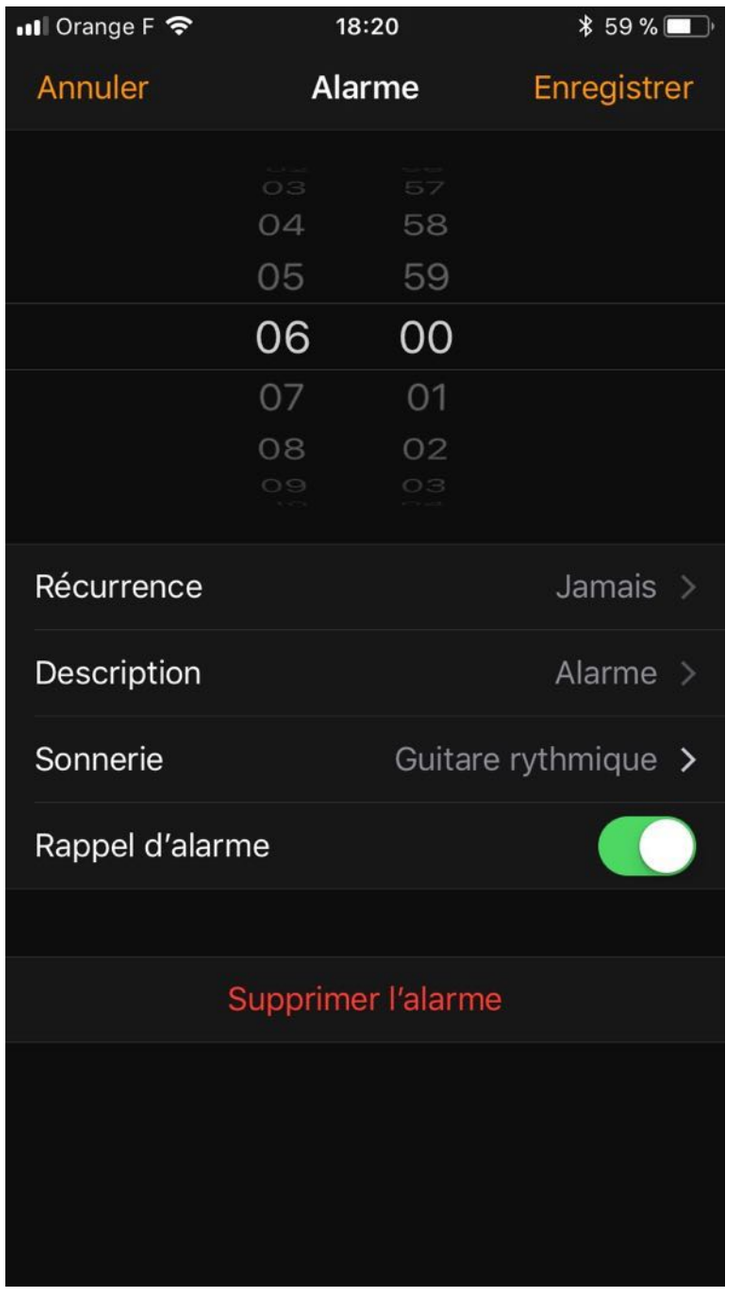
- Exercice
  - Leaks profile



- Utilisation d’un composant UIPickerView
  - Equivalent des menus déroulants présentant un choix multiple
  - Composant dont la largeur occupe tout l’écran de l’iPhone
  - Version spécifique pour le choix de dates et heures







- Utilisation d'un composant UIPickerView
  - Le contrôleur est responsable du rendu à l'écran (apparence, animations, etc.), mais il ne sait rien des données à afficher, ni quoi faire lorsque l'utilisateur sélectionne une entrée
  - Les composants comme le Picker n'attendent pas qu'on leur dise ce qu'il faut afficher
  - UIPickerView possède deux propriétés qui pointent sur des objets
    - dataSource

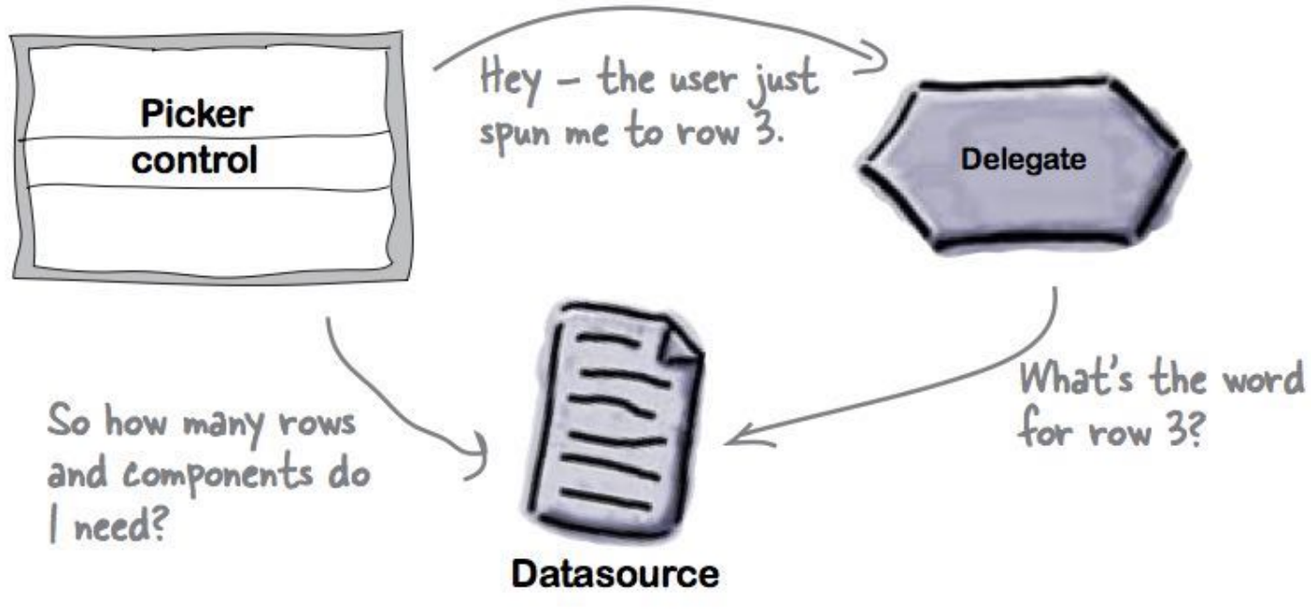
      ```
      @property(nonatomic, assign) id<UIPickerViewDataSource> dataSource
      ```

      - Le contrôle demande à l'objet datasource ce dont il a besoin
      - La "datasource" doit fournir les informations dans un format que le contrôle attend. Elle fournit le nombre de composants (colonnes) et le nombre total de lignes

- Utilisation d'un composant UIPickerView
  - delegate

    ```
    @property(nonatomic, assign) id<UIPickerViewDelegate> delegate
    ```

    - Le délégué doit implémenter les méthodes qui retournent le rectangle dans lequel sont dessinées les lignes
    - Il fournit également le contenu de chaque ligne d'un composant sous la forme de chaîne de caractère ou de vue
  - Il répond aux sélections et désélections de l'utilisateur



Syntaxe Objective C - Protocoles

- Déclaration d'une variable `id<UIPickerViewDelegate> delegate`
  - L'objet vers lequel pointe la variable d'instance delegate doit implémenter le protocole `UIPickerViewDelegate`
  - Un protocole est l'équivalent d'une interface en Java
    - déclare des méthodes qui peuvent être implémentées par n'importe quelle classe
    - certaines méthodes peuvent être optionnelles
    - Une classe qui implémente un protocole le précise dans la déclaration de l'interface
    - Une même classe peut implémenter plusieurs protocoles

```
@interface InstantwitViewController : UIViewController<UIPickerViewDataSource, UIPickerViewDelegate>
```

Syntaxe Objective C - Protocoles

Définition d'un protocole

```
@protocol MyProtocol

- (void)requiredMethod;

@optional
- (void)anOptionalMethod;
- (void)anotherOptionalMethod;

@required
- (void)anotherRequiredMethod;
@end
```

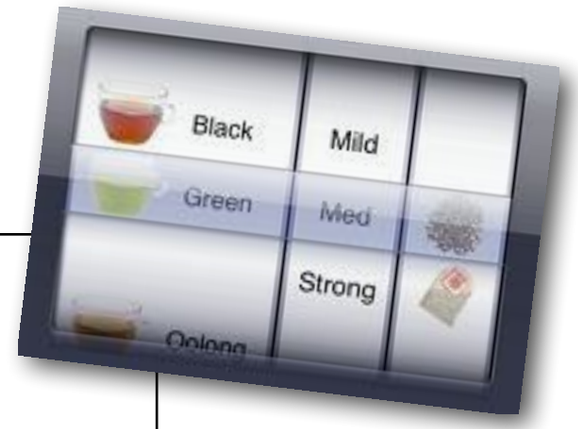
Possibilité de tester si un objet implémente un protocole

```
if ( ! [receiver conformsToProtocol:@protocol(MyXMLSupport)]  )
```

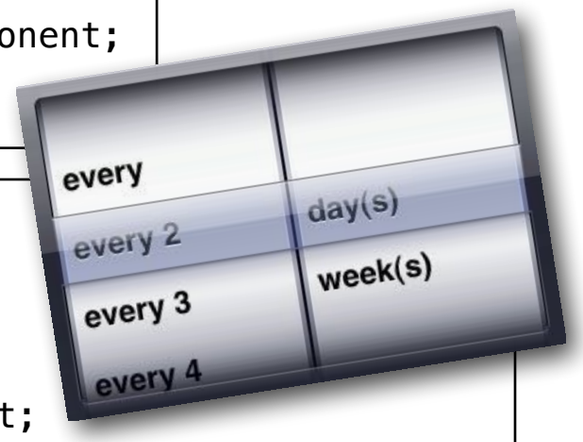
Utilisation d'un composant UIPickerView

Définitions des protocoles utilisés par UIPickerView

UIPickerView.h




```
@protocol UIPickerViewDataSource<NSObject>
@required

// returns the number of 'columns' to display.
- (NSInteger)numberOfComponentsInPickerView:(UIPickerView *)pickerView;

// returns the # of rows in each component..
- (NSInteger)pickerView:(UIPickerView *)pickerView numberOfRowsInComponent:(NSInteger)component;
@end
```




```
@protocol UIPickerViewDelegate<NSObject>
@optional

// returns width of column and height of row for each component.
- (CGFloat)pickerView:(UIPickerView *)pickerView widthForComponent:(NSInteger)component;
- (CGFloat)pickerView:(UIPickerView *)pickerView rowHeightForComponent:(NSInteger)component;

// these methods return either a plain UIString, or a view (e.g UILabel) to display the row for the component.
// for the view versions, we cache any hidden and thus unused views and pass them back for reuse.
// If you return back a different object, the old one will be released. the view will be centered in the row rect
- (NSString *)pickerView:(UIPickerView *)pickerView titleForRow:(NSInteger)row forComponent:(NSInteger)component;
- (UIView *)pickerView:(UIPickerView *)pickerView viewForRow:(NSInteger)row forComponent:(NSInteger)component
reusingView:(UIView *)view;

- (void)pickerView:(UIPickerView *)pickerView didSelectRow:(NSInteger)row inComponent:(NSInteger)component;

@end
```

- Utilisation d'un composant UIPickerView
  - Créer une application qui comporte un composant UIPickerView avec 2 colonnes utilisé pour produire rapidement un message à twitter
    - La première colonne permet de décrire l'activité
      - dors, mange, suis en cours, galère, cours, poireaute
    - La deuxième colonne permet d'ajouter l'état d'esprit
      - ;), :), :(, :O, 8), :o, :D, mdr, lol
  - L'appui sur le bouton tweet it doit provoquer la mise en forme d'une phrase qui sera affichée sur la console dans un premier temps
  - Dans un deuxième temps ajouter un champ de texte dans lequel saisir le message

- Gestion du clavier
  - Apparition automatique dès qu'un champ de texte a le "focus"
  - La fenêtre de l'application redirige les événements vers le "firstResponder"
    - lorsqu'un champ texte devient "first responder" il demande au système de faire apparaitre le clavier
    - pour que le clavier disparaisse il faut que le composant texte renonce à son statut de "first responder" par un appel à la méthode resignFirstResponder

- Gestion du clavier
  - Difficultés
    - Quand on a plusieurs champs de texte il faut savoir qui est le “first responder”

- Gestion du clavier
  - Solutions
    - Méthode `endEditing` de la classe UIView qui recherche dans la vue et dans ses sous-vues un champ de texte qui est le “first responder” et qui lui demande de relacher son statut de “first responder”
      - Le contrôleur de la vue possède une référence vers sa vue (self.view)
    - Les événements utilisés pour faire disparaître le clavier sont :
      - un appui sur le fond (en dehors des composants) : installer un "tap gesture recognizer" sur le fond)
      - un appui sur le bouton done du clavier (évènement DidEndOnExit)

- Ajout de l'envoi d'un tweet
  - Ajouter la fonctionnalité de partage du message prérempli
    - A partir d'iOS 11 le partage via twitter ou Facebook n'est plus géré nativement (il faut utiliser les frameworks de twitter et de Facebook)
    - Autre solution : passer la main au contrôleur permettant le partage via d'autres applications (**UIActivityViewController**)
      - Pour effectuer une transition de la vue principale vers celle de partage il faut que le controleur de vue actif "présente" le controleur de vue qui va prendre la suite
        - `presentViewController`
      - Lorsque l'opération de partage est finie (annulation ou envoi) afficher un message (UIAlertController) pour indiquer si l'utilisateur a annulé l'envoi ou non

Blocks

- Le contrôleur de vue **UIActivityViewController** possède une méthode qui permet de fournir un bloc de code qui sera exécuté lorsque la composition sera terminée
  - ce contrôleur ne possède pas d'objet delegate et s'appuie sur un ajout du langage - les blocks - pour prévenir le contrôleur
- la propriété completionWithItemsHandler est de type block
  - Ce block est un morceau de code qui sera exécuté plus tard
  - Lors de son exécution, ce block recevra un argument afin de savoir si l'opération a été annulée ou si le partage a été effectué

Blocks

- Le concept de block revient à stocker une référence vers un morceau de code dans une variable
- Ce code a accès à tous les éléments du contexte dans lequel il est déclaré (variables locales à la méthode à l'intérieur de laquelle il se trouve, variables d'instances, ...)
- Il peut être stocké dans une variable, passé en argument à une méthode, ...
- Les blocks sont l'équivalent des clotures (*closures*) ou des expressions lambda en Java (JDK 8), en C# (=>), et dans de nombreux autres langages (dont Swift)

- Blocks - syntaxe
  - Déclaration d'un block
    - `float (^operation)(float,float);`
      - variable dont le nom est operation
      - contient un block qui attend 2 arguments de type float
      - retourne un float
  - Création d'un block

```
operation = ^(float a, float b){
        return a-b;
    };
```

  - Utilisation

```
float res = operation(_sliderA.value,_sliderB.value);
```

Blocks - utilisation

- Les blocks fournissent un mécanisme de callback puissant car ils ont accès à tout le contexte disponible au moment de leur création
  - cela évite d'avoir à rendre accessible certaines données depuis plusieurs méthodes d'un même objet en passant par des variables d'instances ou en transmettant un objet de contexte
- Les frameworks Cocoa migrent vers une utilisation importante des blocks (solution exclusive sur de nombreux nouveaux frameworks)
- Usage
  - Completion handlers, Notification Handlers, Error Handlers
  - Enumeration (opération sur tous les éléments d'une collection, filtrage, recherche,tri)
  - Animation des vues et transitions
  - Programmation concurrente

Blocks - exemples

```
[UIView animateWithDuration:0.2 animations:^{
                        view.alpha = 0.0;
                        }
                    completion:^(BOOL finished){
                        [view removeFromSuperview];
                        }
                    ];
```

```
NSArray *array = @[@"A", @"B", @"C", @"A", @"B", @"Z", @"G", @"are", @"Q"];
NSSet *filterSet = [NSSet setWithObjects: @"A", @"Z", @"Q", nil];

BOOL (^test)(id obj, NSUInteger idx, BOOL *stop);

test = ^(id obj, NSUInteger idx, BOOL *stop) {
    if (idx < 5) {
        if ([filterSet containsObject: obj]) {
            return YES;
        }
    }
    return NO;
};

NSIndexSet *indexes = [array indexesOfObjectsPassingTest:test];
```