

Développer pour iOS

Applications pour iPhone, iPad et iPod touch

Développer pour iOS

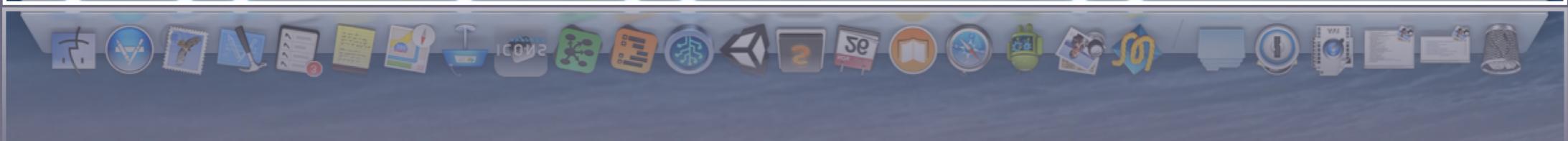
- Des outils (Xcode, Instruments, simulateur)
- Un langage (Objective-C)
- Des frameworks (Foundation, UIKit...)
- Design Patterns (MVC, delegation, KVO...)

XCode

- Environnement de développement
- Disponible uniquement pour MacOSX
- Gratuit (téléchargement depuis l'AppStore)
- Intègre le SDK iOS
 - Pour avoir la dernière version du SDK, il faut la dernière version d'XCode

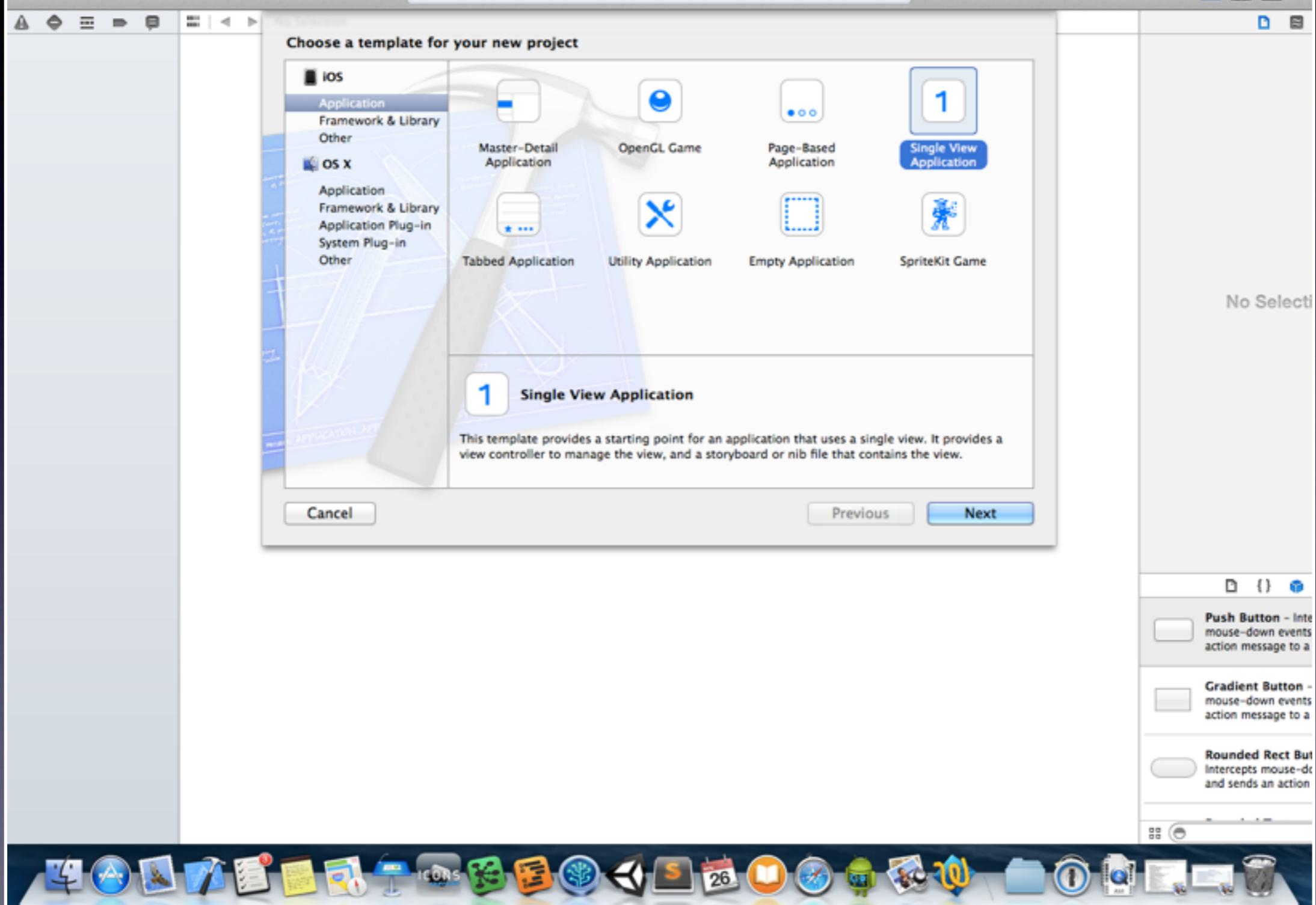
XCode

- Outil de création d'interface graphique WYSIWYG (Storyboards)
- Assistant «binding» Storyboard - Code
- Outils de versionning (git)
- Outils d'analyse / monitoring
- Documentation et quick help



Loading

No Issues



Loading

No Issues



Choose options for your new project:

Product Name:

Organization Name: Florian Burel

Company Identifier: fr.florianburel

Bundle Identifier: fr.florianburel.ProductName

Class Prefix: iPad iPhone Universal

Device: iPhone

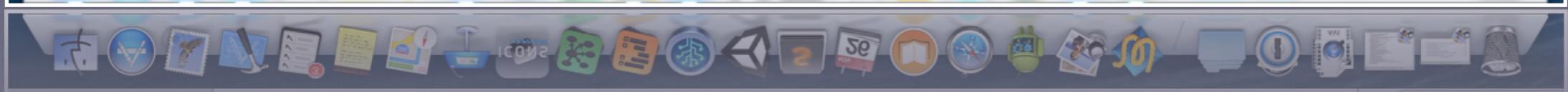
Cancel Previous Next

No Selection

Push Button - Intercepts mouse-down events and sends an action message to a

Gradient Button - Intercepts mouse-down events and sends an action message to a

Rounded Rect But



Loading No Issues

Bureau

Choose options for your new project

Search

DIRECTORIES

- flo
- Dropbox
- Tous mes fichiers
- Images
- Applications
- Documents
- Bureau**
- Téléchargements
- Vidéos
- Musique

CES

- MacBook Air de florian
- Disque distant
- Silverlight

;

encrypted

je sais pas

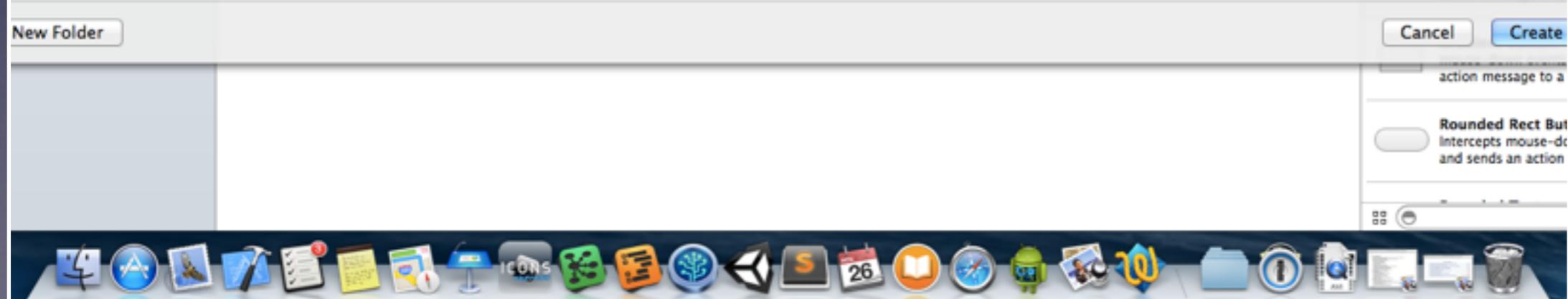
Name

	Name	Date Modified	Size	Kind	Date Added
	Capture d'écran 2013-11-26 à 09.43.37	aujourd'hui 09:43	1,6 MB	PNG	09:43
	Capture d'écran 2013-11-26 à 09.43.48	aujourd'hui 09:43	522 KB	PNG	09:43
	Capture d'écran 2013-11-26 à 09.44.13	aujourd'hui 09:44	521 KB	PNG	09:44
>	Polygone	hier 19:03	--	Folder	hier

Source Control: Create git repository on My Mac

Xcode will place your project under version control

New Folder Cancel Create



Démo

Objective-C

Le langage de programmation pour iOS

Objective-C

- Language orienté objet
 - Classes, Instances
 - Encapsulation, Heritage, Polymorphisme
 - messages / méthodes
- framework Foundation
 - **NSString, NSObject ...**

Une première classe!

- **Le fichier .h**
 - appelé interface
 - les dépendances
 - déclaration publiques
 - «mode d'emploi»
- **Le fichier .m**
 - appelé implementation
 - code
 - méthodes privés
 - variable de classe

Exemple

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

@interface Employee : NSObject

// Méthodes de class
+ (id) newEmployee;

// Méthodes d'instance
- (id) initWithName:(NSString *)name;
- (void) setWork:(int)travail
            withDeadLine:(NSDate *)date;
- (int) raise;
- (void) setRaise:(int)raise;

@end
```

```
#import "Employee.h"

@implementation Employee
{
    // Variable d'instance
    int _raise;
}

- (int) raise
{
    return _raise;
}

- (void) setRaise:(int)raise
{
    _raise = raise;
}
/* ... */
@end
```

signature d'une méthode

- (`int`) `raise;`
 - méthode d'instance (-)
 - retourne un `int`
 - ne prends pas de paramètre
 - pas de () !!!
 - correspond a la signature `raise`

signature d'une méthode

- + (**id**) newEmployee;
- méthode de classe (+)
- retourne un **id** (un objet, typage faible)
- ne prends pas de paramètre
- correspond a la signature *newEmployee*

signature d'une méthode

- `(void) setRaise:(int)raise;`
 - méthode d'instance (-)
 - retourne void
 - prends UN paramètre de type int
 - correspond a la signature `setRaise:`

signature d'une méthode

- `(void)setWork:(int)travail withDeadLine:(NSDate *)date;`
 - méthode d'instance (-)
 - retourne void
 - prends deux paramètres
 - correspond a la signature
setWork:withDeadLine:

Instanciation

- On utilise les pointeurs
 - Les objet ont une adresse fixe
 - La variable contient l'adresse

```
Employee * florian = ...;
```

Instanciation

- Pas de «new»
- NSObject fournit l'API de création
 - Reserver une zone mémoire
 - +(id) alloc;
 - Initialise l'objet (iVar à zéro)
 - -(id) init;

```
Employee * florian = [[Employee alloc] init];
```

Passes le message!

- Java : maVariable.methode();
- C++ : maVariable -> methode();
- en Obj-C : [maVariable **methode**];

Exemples de messages

- Envoyer un message à une classe

```
Employee * florian = [Employee newEmployee];
```

- Envoyer un message à une instance avec 1 paramètre

```
[florian setRaise:200];
```

- Si l'on a plus d'un argument?

```
[florian setWork:work withDeadLine:today];
```

Apple : Convention d'écriture

- un nom de méthode & variable commence par une minuscule
- Un nom de classe commence par une majuscule
- Utiliser le style camelCase
- Les getters ne sont pas préfixés de ‘get’
- Les constantes commencent par un ‘k’

File Edit View Find Navigate Editor Product Debug Source Control Window Help

Polygone.xcodeproj — ViewController.m

Build Polygone: Succeeded | Yesterday at 19:08 No Issues

ViewController.m

```
@interface ViewController : UIViewController
```

```
@end
```

```
@implementation ViewController
```

```
- (IBAction)updatePolygoneNumberOfSide:(id)sender {
```

```
    NSLog(@"Hey");
```

```
}
```

```
- (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
```

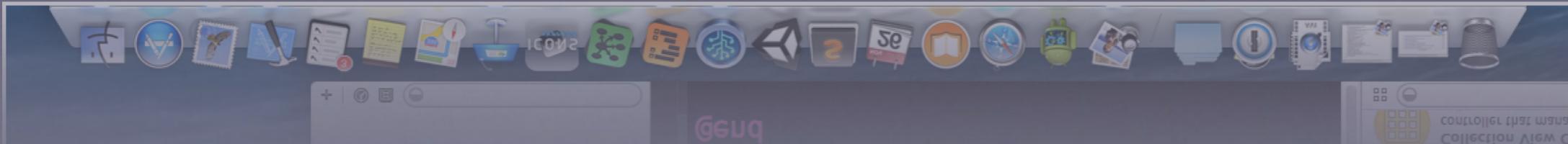
Identity and Type
Name ViewController
Type Default - Obj-C Class
Location Relative to Group
View Controller
Full Path /Users/flo/Devel/Polygone/Polygone/ViewController.h

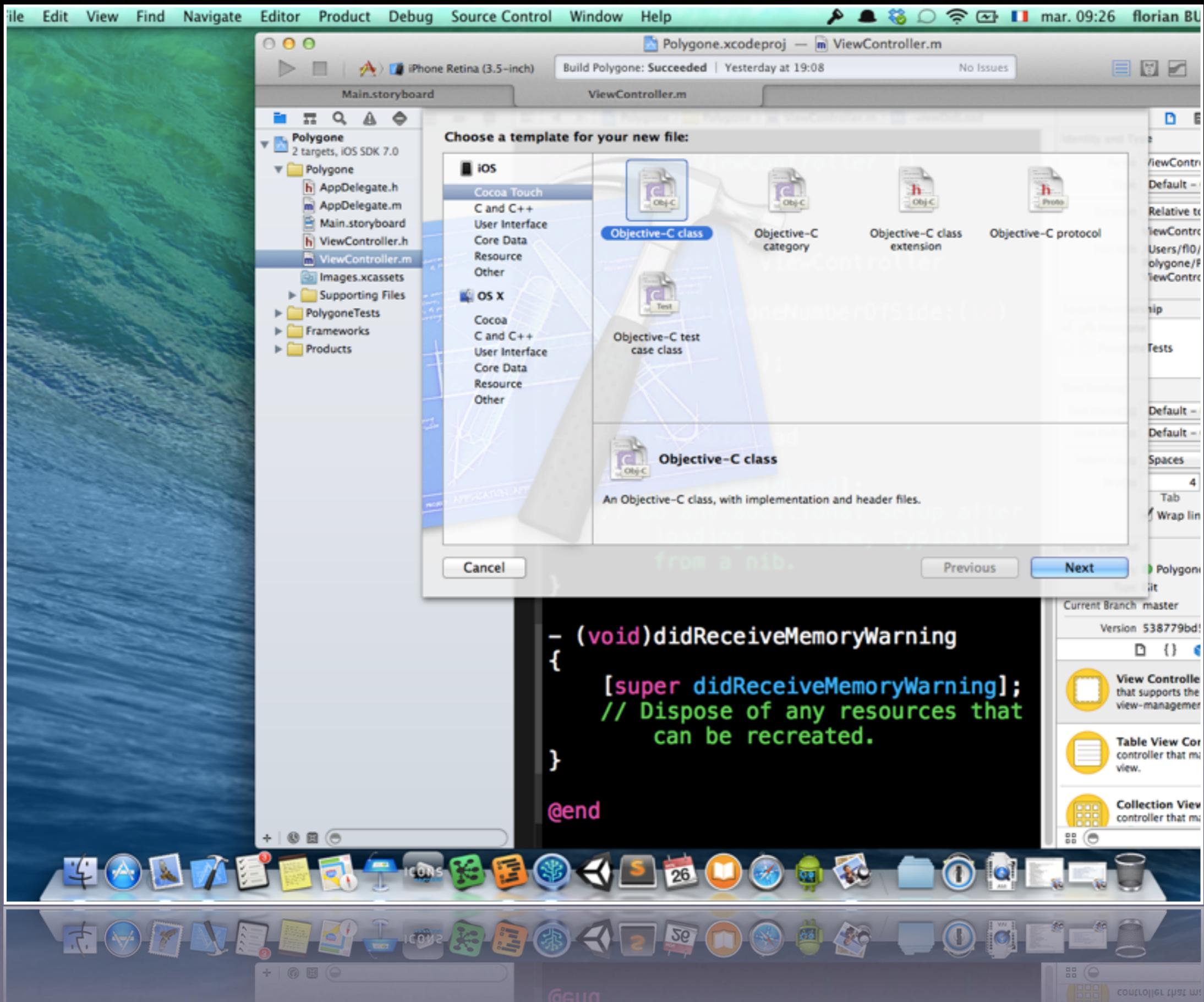
Target Membership
 Polygone
 PolygoneTests

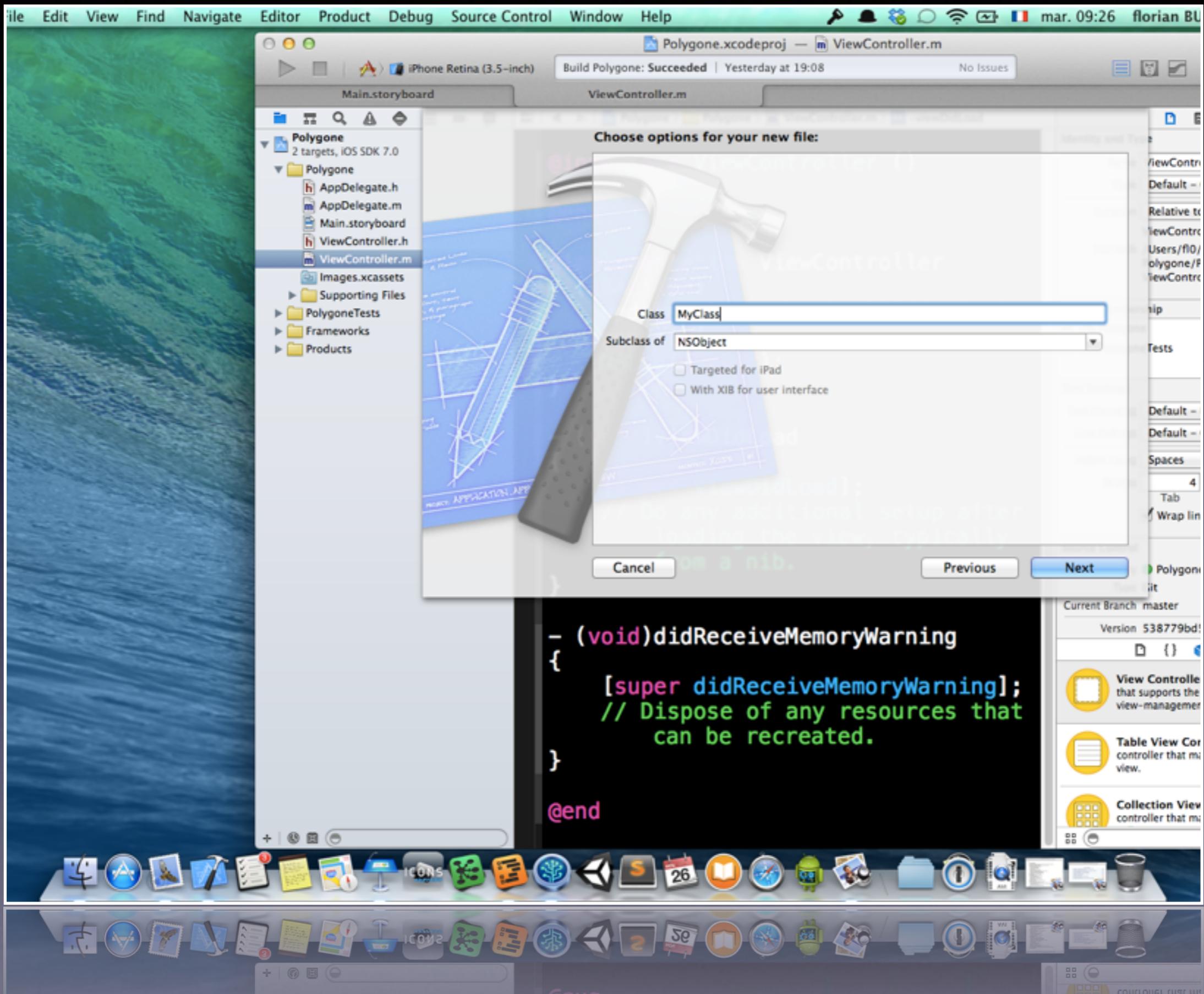
Text Settings
Text Encoding Default – Unicode
Line Endings Default – OS X
Indent Using Spaces
Widths 4
Tab
 Wrap lines

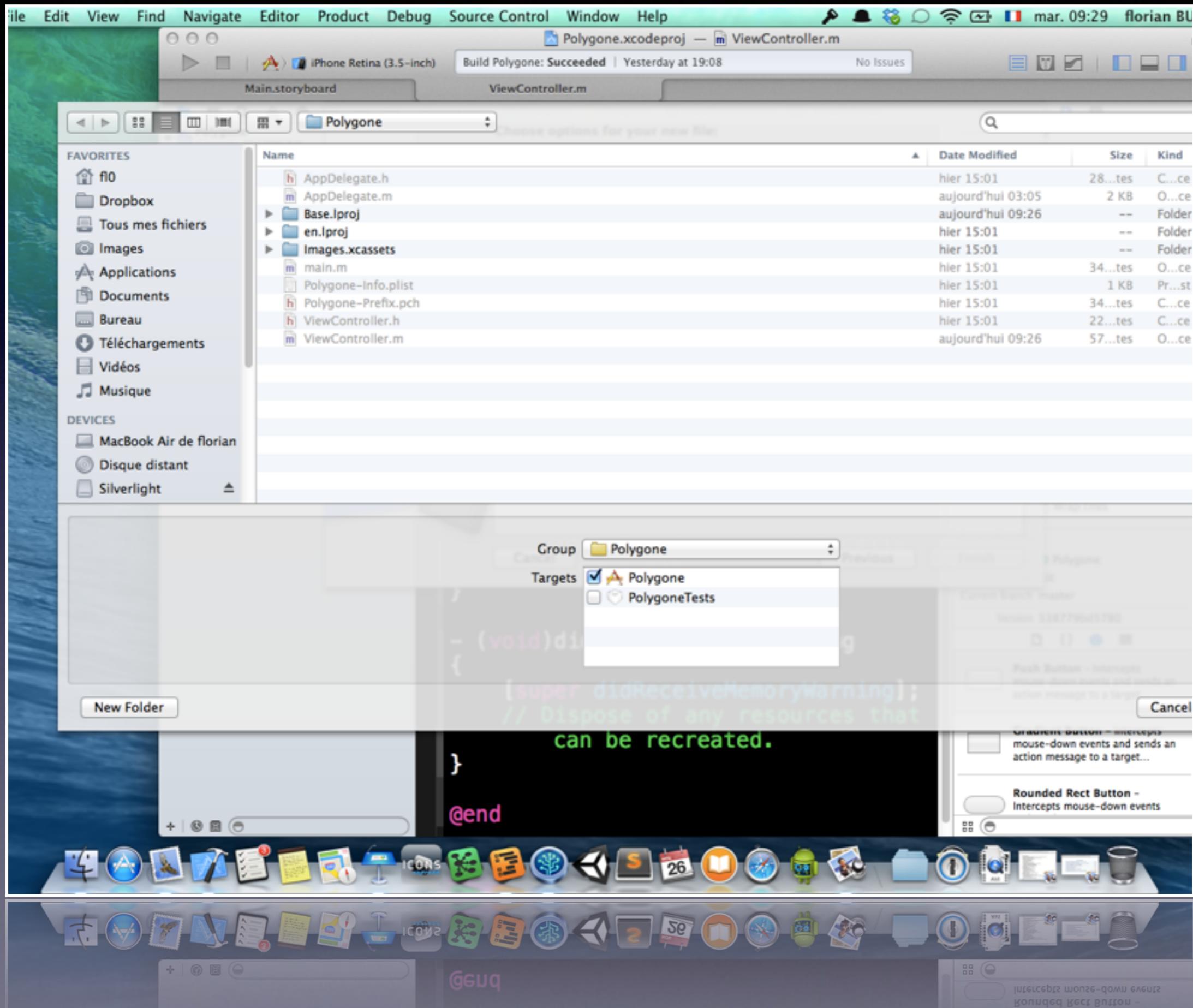
Source Control
Repository Polygone
Type Git
Current Branch master
Version 538779bd578

View Controller – that supports the full view-management interface.
Table View Controller – controller that manages a table view.
Collection View Controller – controller that manages a collection view.









Démo

Les variables d'instance

- Se déclarent dans l'implementation

```
int _raise;
```

- Rendu accessible via des accessors déclarer dans l'@interface
 - (**void**) raise;
 - (**int**) setRaise:(**int**)raise;
- Accessor ensuite implémenté dans le .m
 - Code, en général, sans grande valeur ajoutée

@property

- Declaration / creation *dynamique** des accessors
- Donne accès a la dotSyntaxe
 - Meilleur lisibilité du code

@property

- Se place dans l'**@interface**

```
@property () int raise;
```

- Remplace les déclaration d'accessors

```
- (void) raise;  
- (int) setRaise:(int) raise;
```

- Déclare la iVar **int _raise**

```
int _raise;
```

@property

- Génère au runtime les accessors manquants
 - Crée des accessors simple
 - Seulement les accessors non implémentés manuellement
 - ! Respecte les conventions de nommage

@property

- `@property () int raise;`
 - `getter : - (int) raise;`
 - `setter : - (void) setRaise:(int)raise`
 - `ivar : int _raise;`
- `@property () NSString * name;`
 - `getter : - (NSString *) name;`
 - `setter : - (void) setName:(NSString *)name`
 - `ivar : NSString * _name;`

@property

- Se place dans l'**@interface**

```
@property () Type (*) variable;
```

- Configurable via des options
 - optionnel mais souvent nécessaire
 - ordre des paramètres non important

```
@property (READABILITY, MEMORY_MGMT, ATOMICITY) Type  
(*) variable;
```

Visibilité

- readwrite : déclare / génère un setter et un getter (*defaut*)
- readonly : déclare / génère uniquement un getter

Gestion de la mémoire

- assign : Reservé aux types primitifs (int...)
- strong : Augmente le retainCount* de l
 - L'objet reçu ne sera pas détruit tant que la property «`pointera`» vers lui
- weak : Ne modifie pas le retainCount*
- Spécifique à certains design pattern

Atomicité

- atomic : Empêche l'accès concourant aux accessors
- nonatomic : ne protège l'accès aux accessors

@property

- Avec les @property

```
@interface Employee : NSObject
@property (readwrite, strong) NSString * name;
@property (readonly) NSString * address;
@end
```

- Sans les @property

```
@interface Employee : NSObject
- (NSString *) name;
- (void) setName:(NSString *)name;
- (NSString *) address;
@end
```

@property

- Rendu possible grâce aux @properties

```
florian.name = @"Burel"  
// équivaut à écrire  
[florian setName:@"Burel"]
```

- Avantages:
 - Code plus lisible (moins de [])
 - pas de compromis sur l'encapsulation
 - Proche de la syntaxe des *structs C*

Démo

Mon init a moi!

- Méthode hérité par NSObject
 - `(id)init;`
 - Pas besoin de la déclarer dans le l'`@interface`
 - `(id)init {
 self = [super init];
 if (self) {
 // some code here
 }
 return self;
}`

init 2.0

- Respecter les conventions de nommage
 - `(id)initWithSite:(FASite *)newSite;`
 - `(id)initWithArticle:(FAArticle *)newArticle forSite:(FASite *)site;`
- Méthode à déclarer dans l'`@interface`
- !!! Impossible d'interdire l'accès à l'init classique
 - Assez peu utilisé:
 - Méthode de classe dites ‘constructeur’
 - ‘lazy getters’

lazy getter

- Instantiation de la iVar dans le getter
- Idéal lorsque couplé au property

```
- (UIWebView *)webView
{
    if(!_webView)
    {
        _webView = [[UIWebView alloc] init];
        _webView.hidden = YES;
    }
    return _webView;
}
```

Pas si null que ça

- En objective-C, null n'existe pas.
- La valeur affectée par défaut est `nil` (`NULL`)
- On peut envoyer des messages à `nil` sans risque
- Pas de `NullPointerExceptions`

Le typage dynamique

- Comme le java ou le C#, l'objective-C est fortement typé

```
NSString * str = @“toto”;
```

- Possibilité, pour les objet, d'utiliser le typage dynamique

```
id str = @“toto”; // Notez l'absence de l'étoile!!!!
```

- **id** represente alors une instance, dont la classe est non-définie au moment du build

Le Framework Foundation

Les classes de bases

Foundation

- Framework (boîte à outils) de base d'Apple pour l'Objective-C
- Compatible Mac & iOS
- Classes de bases

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

NSString

- Déclaration littérale

```
NSString * str = @“Hello World”;
```

- Concaténation

```
NSString * str = [NSString stringWithFormat:@“Hello %@”,  
@“Florian”];
```

- Autres méthodes usuelles de la classe NSString
 - (**BOOL**) isEqualToString:(**NSString** *)stringToCompare;
 - (**int**) length;
 - (**double**) doubleValue;
 - (**NSRange**) rangeOfString:(**NSString** *)stringToSearch;

NSArray

- Collection ordonnées
- Les éléments sont positionnés à un index
- On récupère les éléments en fonction de cet index

```
NSArray * myArray = @{@"florian", @"mickaël"};
NSString * firstObject = myArray[0];
NSString * lastObject = [myArray lastObject];
```

NSDictionary

- Collection clef / valeur
- Les éléments sont associés à une clef
- On récupère les éléments en fonction de cet clefs

```
NSDictionary * colors = @{
    @"noir": [UIColor blackColor],
    @"blanc": [UIColor whiteColor]
};
Employee * white = colors[@"blanc"];
```

NSSet

- Sorte de NSArray mais non ordonné
- Pas fait pour récupérer un élément en particulier
- Algorithme efficace d'énumération et de tri

Autres wrappers

- NSDate

```
+ (id)date;      // Retourne la date du jour  
+ (id)dateWithTimeIntervalSince1970:  
(NSTimeInterval)secs;
```

- NSNumber

```
+ (NSNumber *)numberWithInt:(int)value;  
+ (NSNumber *)numberWithFloat:(float)value;
```

- NSURL

```
+ (id)URLWithString:(NSString *)URLString;
```

NSData

- Représente un paquet de donnée
 - Peut se récupérer depuis une URL
(locale ou distante)
- + `(id)dataWithContentsOfURL:(NSURL *)url;`
- Peut aussi s'écrire à une URL donnée
 - Beaucoup d'autre classe peuvent se construire à partir d'un NSData

Les Vues

Manipulation, création, dessin

Qu'est-ce qu'une vue?

- `UIView` est une classe représentant une zone rectangulaire sur l'écran dans laquelle on peut dessiner et/ou gérer la réception d'évènements tactiles
- `UILabel`
- `UIButton`

Qu'est-ce qu'une vue?

- Sous-classe de UIView
- Hiérarchisée
- Plan
- Système de coordonnées

La hiérarchie des vues

- Une seule supervue
 - `(UIView *) superView;`
- 0, 1 ou plusieurs sous-vue
 - `(NSArray *) subviews;`
- L'ordre des sous-vue est important
 - Les dernières dans l'Array sont au dessus
 - `(void) bringSubviewToFront:(UIView *)subview;`

La hiérarchie des vues

- Le plus souvent construite via StoryBoard
- Possible aussi en code:

```
// Ajoute «subview» en tant que sous-vue du destinataire  
- (void) addSubview:(UIView *)subview;
```

```
// Enlève le destinataire de sa «superview»  
- (void) removeFromSuperview;
```

- Lorsque l'on veut ajouter une vue à une autre, il faut définir l'emplacement où l'on veut qu'elle s'affiche

Coordonnées

- `CGFloat` = float mais special CoreGraphic
- `CGPoint` = (`CGFloat` x, `CGFloat` y)
- `CGSize` = (`CGFloat` width, `CGFloat` height)
- `CGRect` = (`CGPoint` origin, `CGSize` size)

Coordonnées

- l'origine d'une vue est le coin supérieur gauche
- l'unité est le point (même rendu sur un retina que sur un écran classique)

les @properties utiles

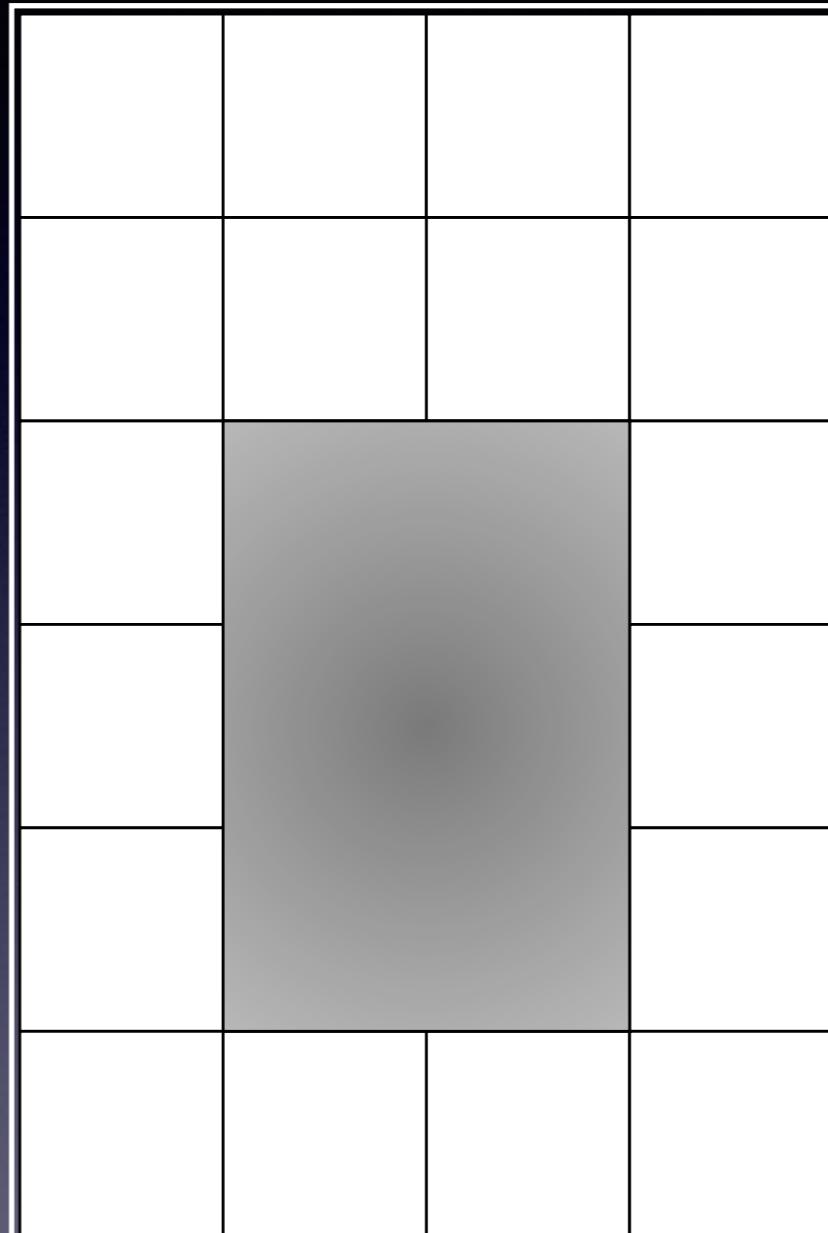
- `@property CGRect bounds;`
 - origine et taille «vue de l'intérieur»
- `@property CGPoint center;`
 - Centre de la vue dans sa superview
- `@property CGRect frame;`
 - origine et taille dans sa super vue

Pour la vue «grise»:

bounds = (0, 0, 2, 3)

frame = (1, 2, 2, 3)

center = (2, 3.5)

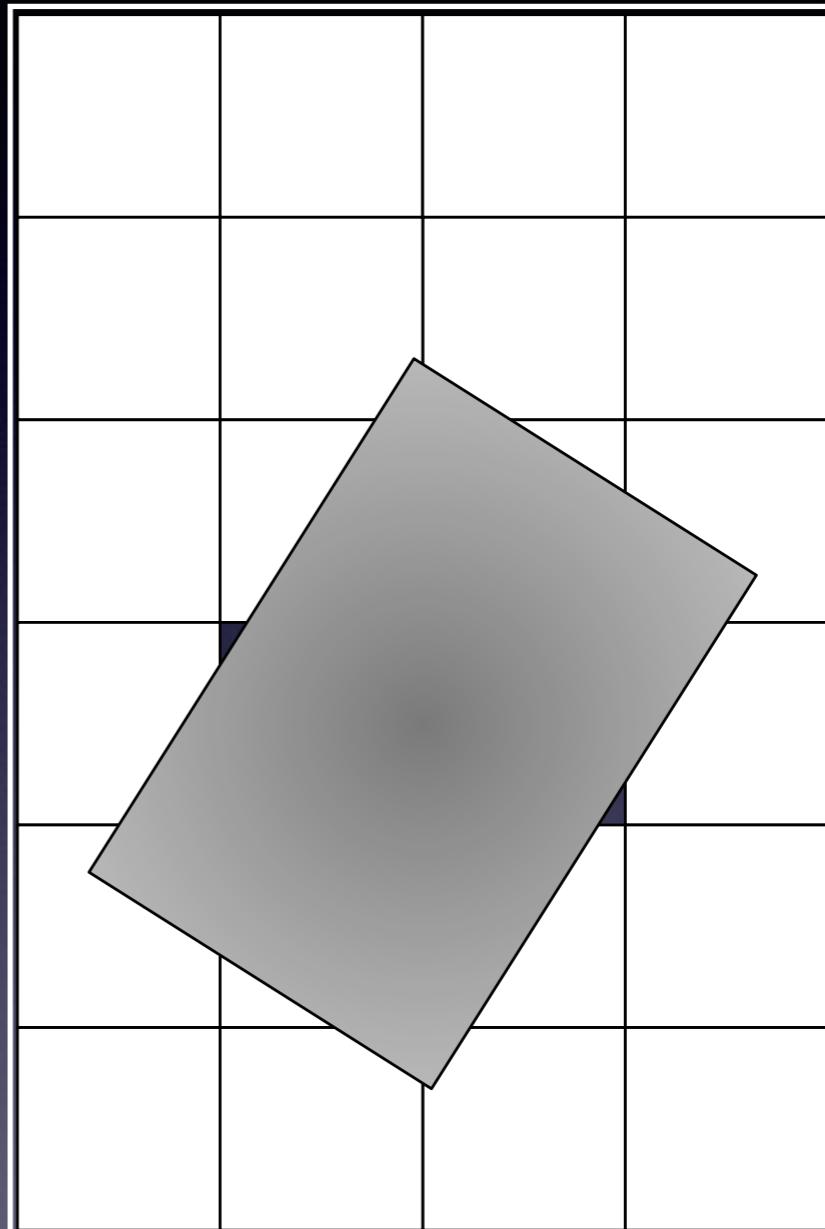


Pour la vue «grise»:

bounds = (0, 0, 2, 3)

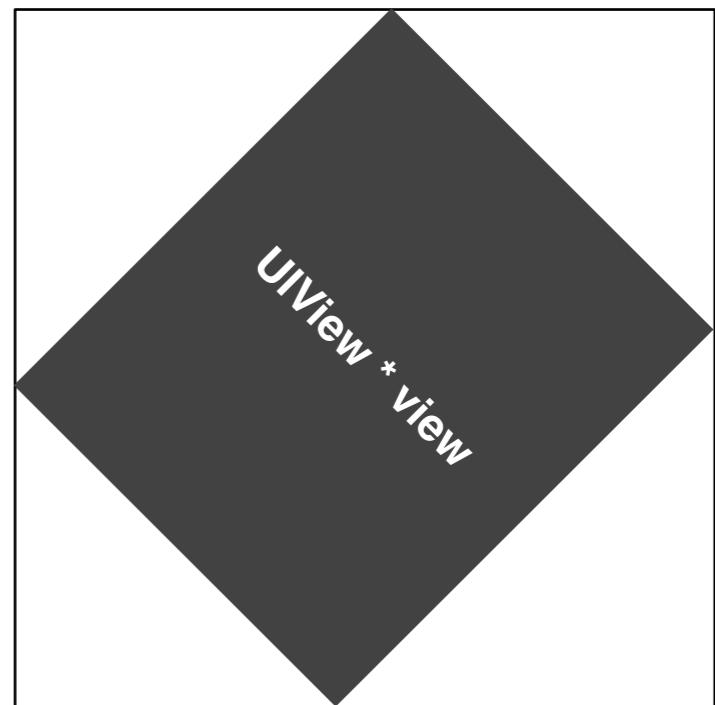
frame = (0.5, 1.7, 3, 3.4)

center = (2 , 3.5)



- Dans l'implementation de la vue, utiliser bounds plutôt que frame
- Garder center et frame pour le contrôle d'une vue depuis sa supervue

`view.frame.size != view.bounds.size`



Exemple

- Creation d'un échiquier

```
- (UIView *) getChessBoard
{
    UIView * board = [[UIView alloc] init];
    board.frame = CGRectMake(ORIGINE_X,
                            ORIGINE_Y,
                            SIZE_WIDTH,
                            SIZE_HEIGHT);
    [self addSquares:board];
    return board;
}
```

Exemple

```
- (void) addSquares:(UIView *)view {
    CGFloat height = view.bounds.size.height / NB_COL;
    CGFloat width = view.bounds.size.width / NB_ROW;

    for(int row = 0; row < NB_ROW; row++){
        for (int col = 0; col < NB_COL; col++){
            UIView * square = [[UIView alloc] init];
            square.frame = CGRectMake(row * width,
                                      (NB_COL - col) * height,
                                      width, height);
            [view addSubview:square];
        }
    }
}
```

Sous-classer UIView

- Beaucoup de sous-classes déjà disponible
 - UILabel, UIButton, UIImageView...
- On peut créer ses propres composants:
 - Dessiner quelque chose de spécifique
 - Implémenter son propre comportement
 - Il faut alors sous-classer UIView

Notice pour sous-classer UIView

- méthode à surcharger

```
// Initialisateur par default
- (id)initWithFrame:(CGRect)frame
- (id)initWithCoder:(NSCoder *)coder;

// Methode à surcharger si on veut dessiner (CoreGraphics
// ou OpenGL)
- (void) drawRect:(CGRect)rect;

// Appelée à chaque fois que la vue a besoin de
// reorganiser ses subviews (1er affichage, reorientation...)
- (void) layoutSubviews;
```

Comment dessiner

- Utilisation de l'API CoreGraphic
 - API écrit en C
 - Etape 1 : Récuperer le context
 - Etape 2 : Définir une forme
 - Etape 3 : Définir les couleurs
 - Etape 4 :Appliquer

le GraphicContext

- Représente une zone de mémoire tampon
- Peut désigner l'écran, un pdf, png...
- On ajoute des traits / formes
- On l'envoie à la carte graphique pour rendu

Fonctions

```
// Recupération du context
CGContextRef ctx = UIGraphicsGetCurrentContext();

// Commence un path (une forme dans le context)
CGContextBeginPath(ctx);

// Déplace le curseur
CGContextMoveToPoint(ctx, x, y);
CGContextAddLineToPoint(ctx, x, y);

// Ferme la forme active automatiquement
CGContextClosePath(ctx);

// Definie les couleurs de traits et de remplissage
UIColor * color = [UIColor redColor];
CGContextSetFillColorWithColor(ctx, color.CGColor);
CGContextSetStrokeColorWithColor(ctx, color.CGColor);

// Envoie le context a la CG pour rendu
CGContextDrawPath(ctx, kCGPathFillStroke);
```

Démo

Répondre aux gestes

- Tap (click : simple ou multiple)
- Pan (déplacement du doigt sur l'écran)
- Swipe (tourner une page)
- Rotation (tourner avec 2 doigts)
- Pinch (zoomer avec 2 doigts)
- Long Press (maintien appuyé)

UIGestureRecognizer

- UITapGestureRecognizer
- PanGestureRecognizer
- SwipeGestureRecognizer
- LongPressGestureRecognizer
- RotationGestureRecognizer
- PinchGestureRecognizer

Concept

- Lors du touch, le GR appelle une méthode (action) sur un objet (target)
- Init d'un GR avec une paire target/action.

```
UIGestureRecognizer * tap = [[UITapGestureRecognizer  
alloc] initWithTarget:self action:@selector(handleTap)];
```

- Il faut ensuite attacher le GR à une UIView
[imageView addGestureRecognizer:tap];

@selector ???

- En objective-C, un selector (SEL) est une référence à une méthode
- 2 signatures possibles pour les actions:
 - `(void) handleGesture;`
 - `(void) handleGesture:(UIGestureRecognizer *)sender;`
- selector correspondants:

```
SEL noParamSel = @selector(handleGesture);  
SEL oneParamSel = @selector(handleGesture:);
```

Interroger un GR

- Connaitre la vue propriétaire d'un GR

```
@property(nonatomic,readonly) UIView *view;
```

- Plus d'infos...

- (CGPoint)locationInView:(UIView*)view;

```
@property (nonatomic) CGFloat rotation;
```

```
@property (nonatomic) CGFloat scale;
```

- (CGPoint)translationInView:(UIView *)view;

Démo

Design strategies

Partie I

Objectifs

- Ecrire moins de code
 - Rendre mon code réutilisable
 - composant graphique : label, bouton
 - Rendre mon code interopérable
 - Même algo pour macOS et iOS

Strategie MVC

Classe «metier»:

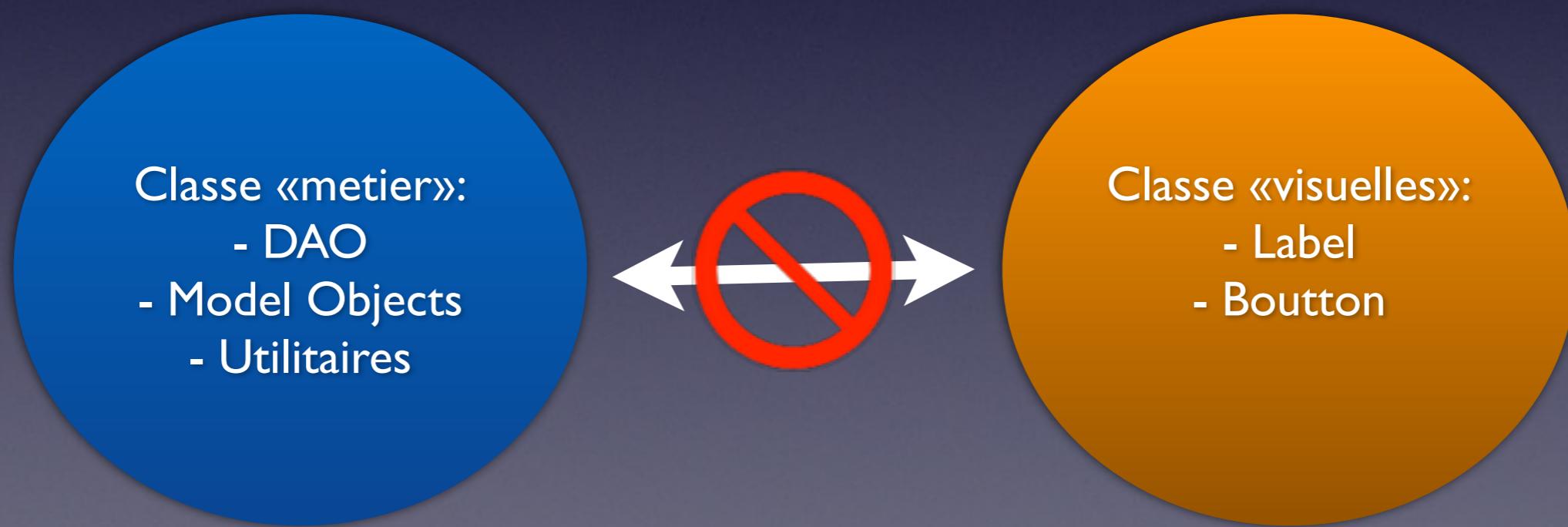
- DAO
- Model Objects
- Utilitaires

Classe «visuelles»:

- Label
- Boutton

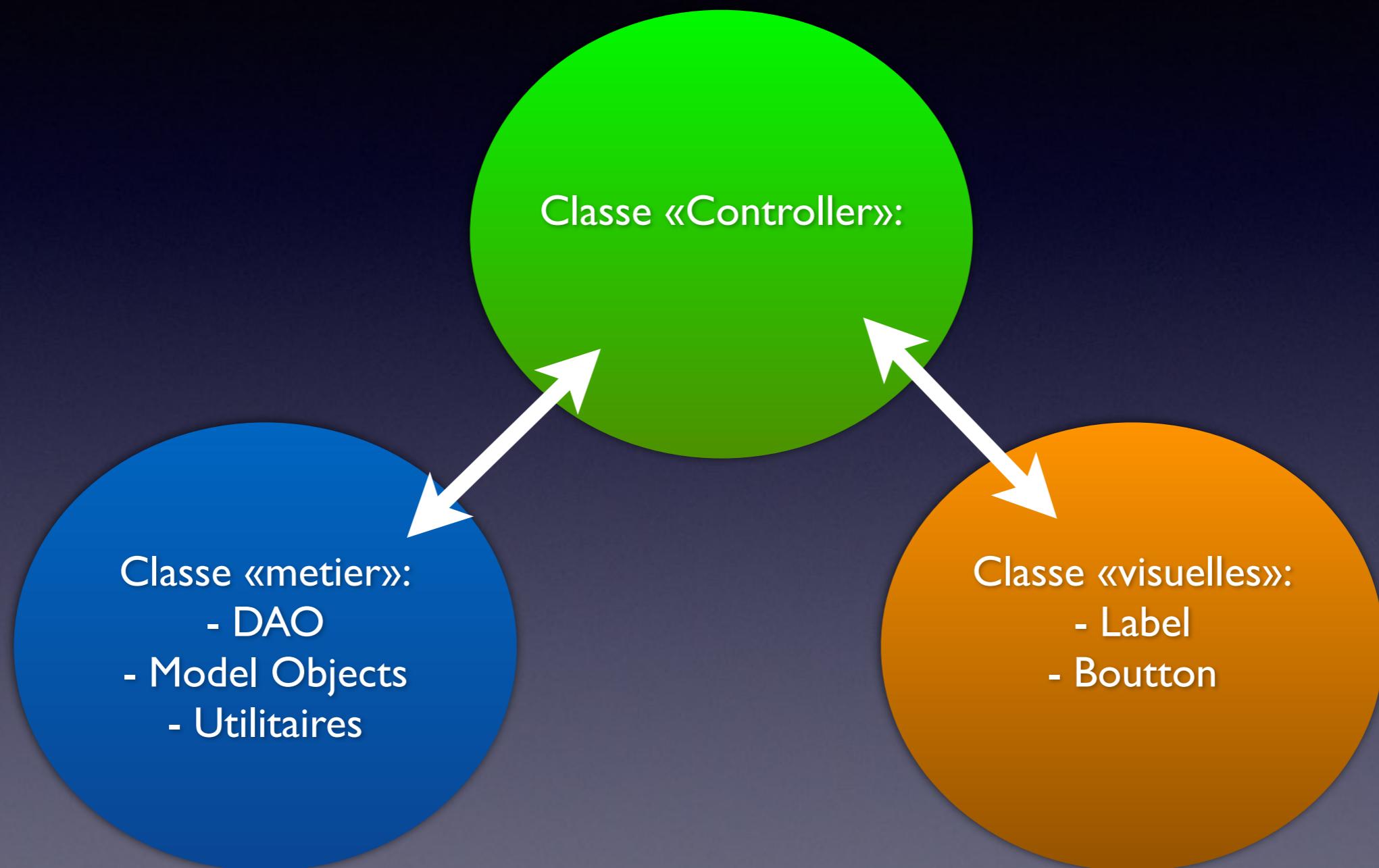
Le design MVC préconise la séparation du code métier (interopérable) du code graphique (réutilisable d'une appli sur l'autre mais spécifique au support)

Strategie MVC



Pour garantir l'interopératibilité et la réutilisabilité des composants, le métier et le visuel ne doivent pas communiquer directement

Strategie MVC



On utilise donc un autre type de classe: les contrôleur.
Ceux ci ont pour mission de synchroniser le métier et le visuel

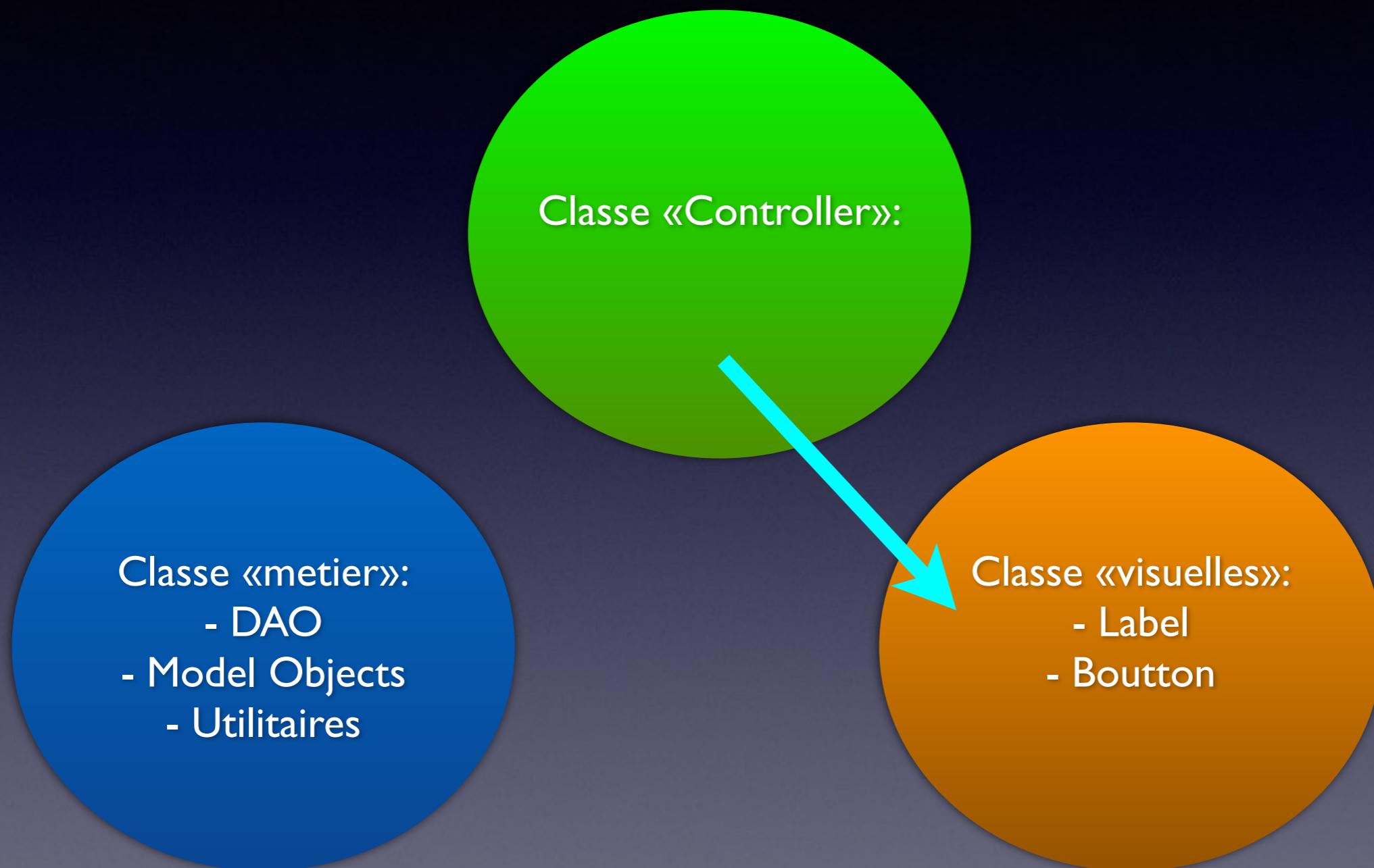
Communication

Classe «Controller»:

Classe «metier»:
- DAO
- Model Objects
- Utilitaires

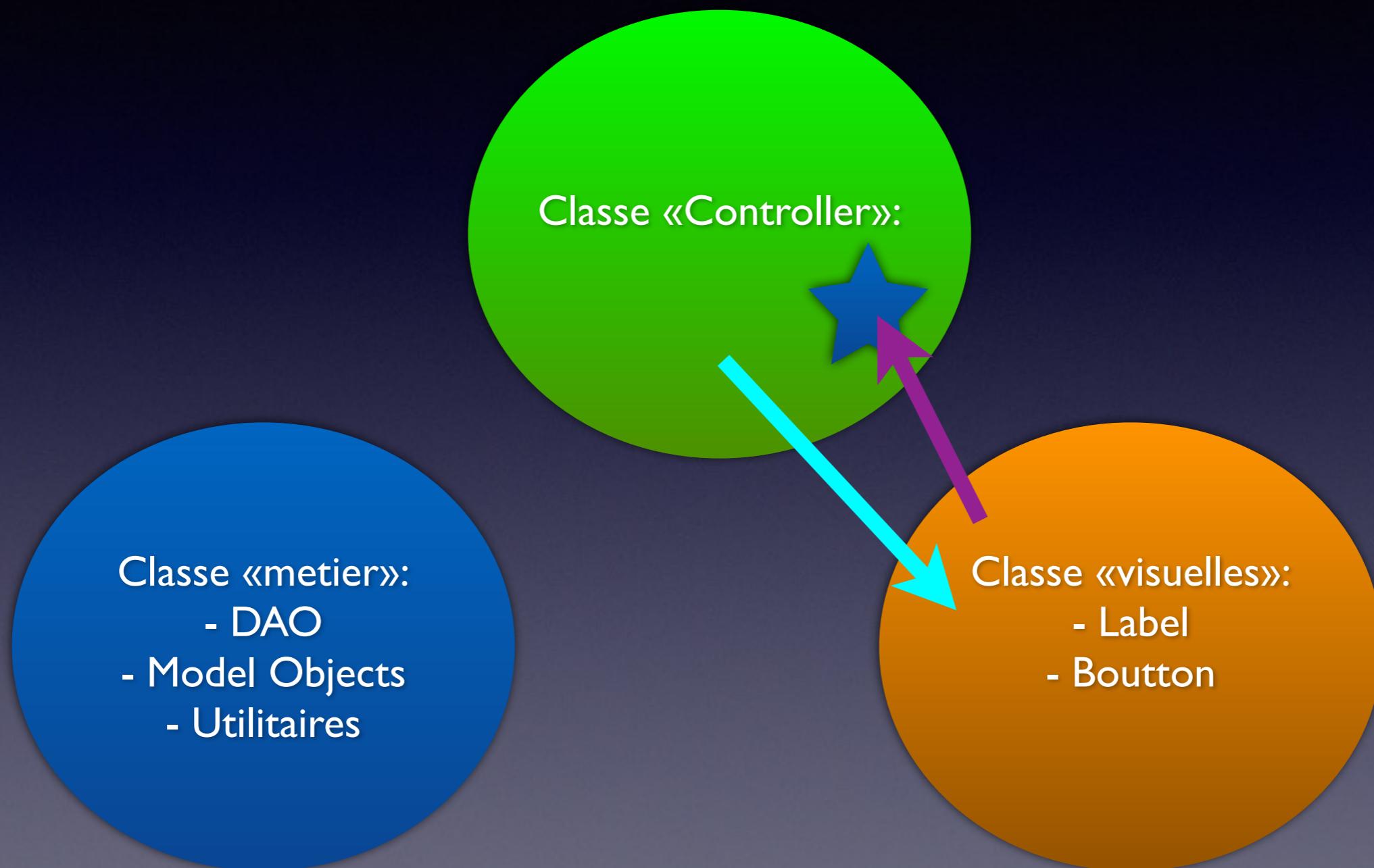
Classe «visuelles»:
- Label
- Boutton

Outlets



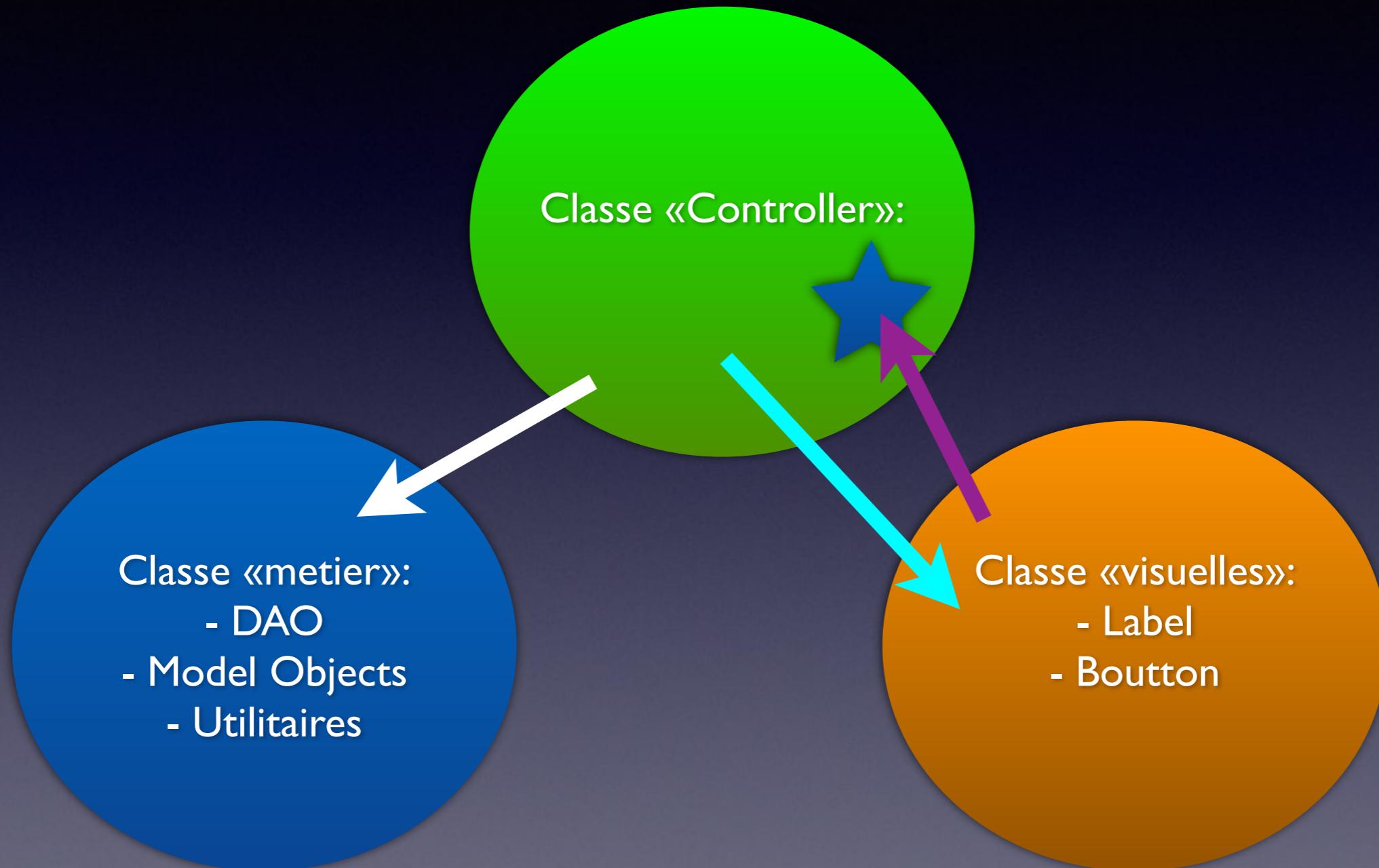
Le contrôleur connaît les différents éléments de l'interface graphique et peut leurs envoyer des messages

target - action



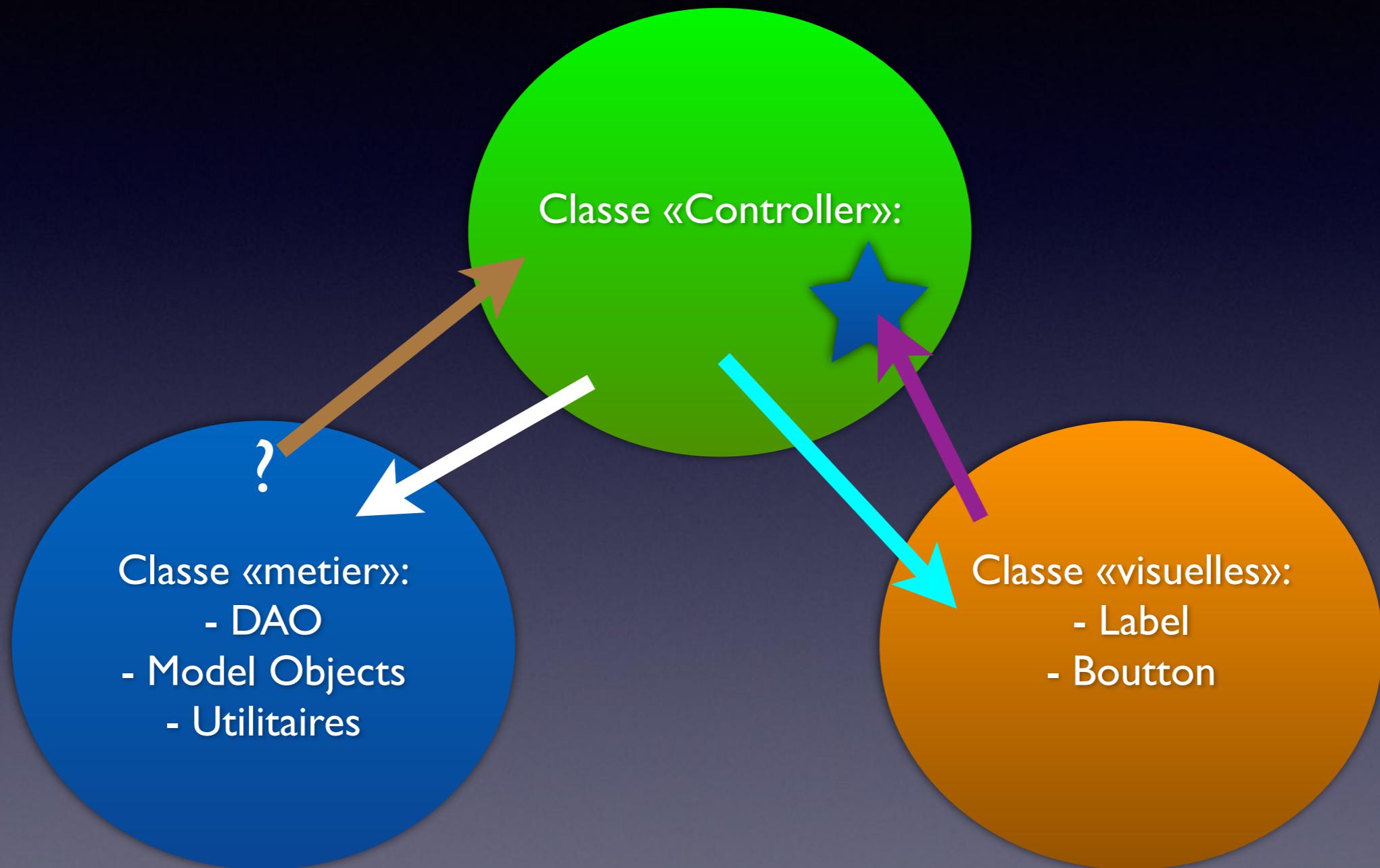
Les classes visuelles ne connaissent pas le controller, toutefois le controller peut lui demander de lui envoyer un message spécifique sur déclenchement d'évènement

MVC en action



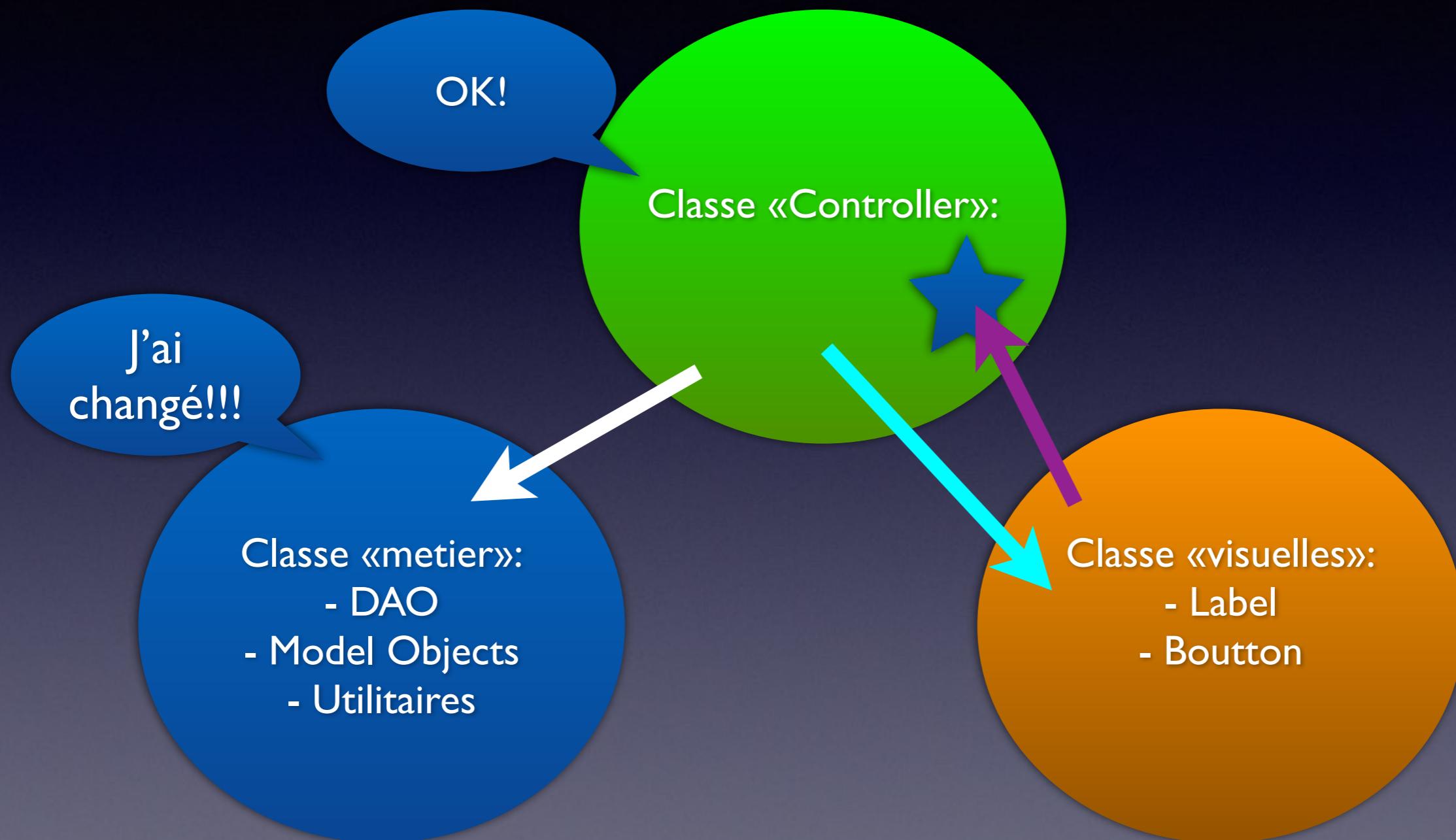
Evidemment le contrôleur connaît son modèle et peut l'interroger et le mettre à jour.

Limite!



Le model peut-il prévenir le controller de changement?

Key Values Observing



NON!!!! Par contre il peut envoyer un signal et le controller peut «écouter» ce signal...

Protocole / Délégation

- Design pattern
- Délégation de tâches à un autre objet
- Protocole de communication

Pourquoi?

- Permet de spécialiser ses classes
- Améliore la réutilisabilité des composants
- Rends le code plus clair

Comment?

- Délègue une partie du comportement d'une classe à un autre objet
- Le *composant réutilisable* implémente un comportement générique et demande des informations spécifiques (données à afficher ou comportement à adopter) à son *delegate*

Exemple I

- Dans le cadre de la création d'un jeu d'échecs, on crée une classe réutilisable «échiquier» capable d'afficher l'échiquier et les pièces
- On veut que cette classe puisse être réutilisée pour différentes variantes des règles des échecs (regular, crazy-house...)
- Comment faire?

Exemple I

- On ne code pas les règles dans la classe «Echiquier»
 - on se limite à l'affichage et au déplacement «stupide» des pieces
- A chaque «tentative de déplacement» l'échiquier demandera à son delegate (qui connaît les règles spécifiques de la partie en cours) si le mouvement est possible.

Exemple 2

- Dans le cadre de la creation d'un reader, on veut créer une classe capable de «swiper» entre plusieurs pages
- On souhaite que cette classe soit réutilisable dans plusieurs projet (reader pour pdf, slider de photos...)
- Comment faire?

Exemple 2

- On se limite au comportement générique
 - affichage d'une vue
 - gestion du numéro de page affichée
 - Evements swipe-avant, swipe-arriere
 - A chaque «changement de page» le reader demande a son delegate quelle page afficher à l'index en cours.

En pratique

- Lors de l’instanciation du composant réutilisable, on lui affecte un delegate
- Le composant réutilisable déclare la liste des «questions» qu’il peut poser à son delegate dans un protocole
- le delegate implemente le protocole
- un peu similaire aux interfaces java

Protocole

- Similaire à `@interface`, mais sans code
- Déclare une liste de méthodes
- Se place dans un fichier *header*

```
@protocol readerDelegate
- (UIView *) pageAtIndex:(int)idx;

@end
```

Protocole

- Contrairement aux interfaces java, certaines méthodes peuvent être optionnelles

```
@protocol ReaderDelegate
```

```
@required
```

```
- (UIView *) pageAtIndex:(int)idx;
```

```
@optional
```

```
- (NSString *) legendForPageAtIndex:(int)idx;
```

```
- (void) didSwitchToPage:(int)idx;
```

```
@end
```

- Dans son interface, le composant réutilisable spécifie qu'il a besoin d'une instance qui «réponds» aux questions du protocole
- L'utilisation du typage faible id permet de s'affranchir du type de l'objet qui sera delegate

```
@interface readerView : UIView  
@property (weak) id<ReaderDelegate> delegate;  
@end
```

- L'objet qui deviendra delegate spécifie qu'il se conforme au protocole
- Cela signifie qu'il implémente, au minimum, toutes les méthodes **@required**

```
@interface myViewController : UIViewController  
    <ReaderDelegate>  
@end  
  
@implementation myViewController  
- (UIView *) pageAtIndex:(int)idx  
{  
    UIImage * img = _listeImage[idx]  
    return [[[UIImageView alloc] initWithImage:img];  
}
```

- Le composant peut communiquer avec son delegate:
 - envoie direct d'un message **@required** du protocole

```
UIView * viewToDisplay = [self.delegate pageAtIndex:++_pageIdx];
```

- Tester si le delegate repond à un message **@optionnal**

```
if( [self.delegate respondsToSelector:@selector(didSwitchToPage:)] )  
{...}
```

Démo

ViewControllers

le C, de A à Z

UIViewController

- Une classe du framework UIKit (iOS)
- Le C de votre MVC
- Chaque «écran» d'une application à un viewController associé
- Programmation évènementielle
- a une property très importante:

```
@property(nonatomic, strong) UIView *view;
```

Le cycle de vie des ViewController

```
// Apres chargement du xib, la méthode viewDidLoad est  
appelé  
- (void)viewDidLoad;  
  
// A chaque apparition / disparition de la vue, ces  
méthode sont appelées:  
- (void)viewWillAppear:(BOOL)animated;  
- (void)viewDidAppear:(BOOL)animated;  
- (void)viewWillDisappear:(BOOL)animated;  
- (void)viewDidDisappear:(BOOL)animated;
```

La tête en bas!

- Connaitre son orientation actuelle

```
@property(nonatomic,readonly) UIInterfaceOrientation  
interfaceOrientation;
```

- Message reçu lors de la réorientation

```
// Code executé de manière animé  
- (void)willAnimateRotationToInterfaceOrientation:  
(UIInterfaceOrientation)toInterfaceOrientation duration:  
(NSTimeInterval)duration;
```

Comment les utiliser?

- Créer une nouvelle sous-classe de UIViewController
- Faire le lien entre le storyBoard et le ViewController
- Lier les eventuelles IBOutlet / IBAction
- Répondre aux évènements...

Démo

UITableView

Présenter ses données en liste

Qu'est-ce qu'une UITableView?

- Une sous-classe de UIScrollView
- Présentation des données sous forme de liste
- Affiche 1 colonne à la fois
- Très commun sur iOS
- Hautement personnalisable

- 2 styles au choix, définit lors de l'init

- `(id)initWithFrame:(CGRect)frame style:(UITableViewStyle)style;`

```
typedef enum {
    UITableViewStylePlain,
    UITableViewStyleGrouped
} UITableViewStyle;
```

- Liée à 2 protocoles:

```
@property(nonatomic,assign) id <UITableViewDataSource>
dataSource;
@property(nonatomic,assign) id <UITableViewDelegate>
delegate;
```

Les styles de tableView



UITableViewStylePlain



UITableViewStyleGrouped

dataSource

- Contrôle ce que doit afficher la tableView:

@required

- `(NSInteger)tableView:(UITableView *)tableView numberOfRowsInSection:(NSInteger)section;`
- `(UITableViewCell *)tableView:(UITableView *)tableView cellForRowAtIndexPath:(NSIndexPath *)indexPath;`

@optional

- `(NSInteger)numberOfSectionsInTableView:(UITableView *)tableView;`
- ...

delegate

- Contrôle comment la tableView est affichée et ce qu'elle fait:

```
@optional
- (UIView *)tableView:(UITableView *)tableView
viewForHeaderInSection:(NSInteger)section;
- (UIView *)tableView:(UITableView *)tableView
viewForFooterInSection:(NSInteger)section;

- (void)tableView:(UITableView *)tableView didSelectRowAtIndexPath:
(NSIndexPath *)indexPath;
...
```

NSIndexPath

- Largement utilisé dans les méthodes du protocole UITableViewDelegate

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

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

- Utilisé par les UITableView pour représenter les coordonnées (section, row) d'une UITableViewCell

UITableViewCell

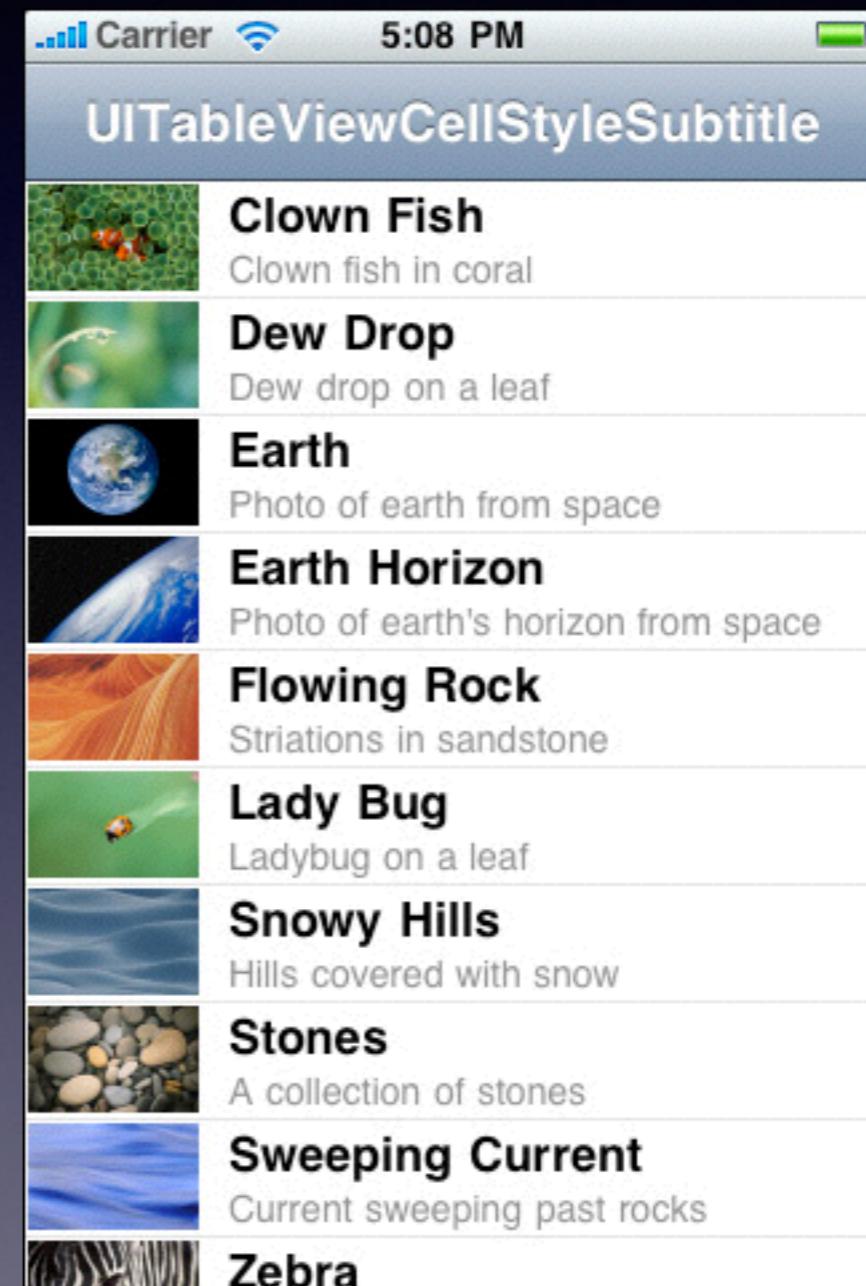
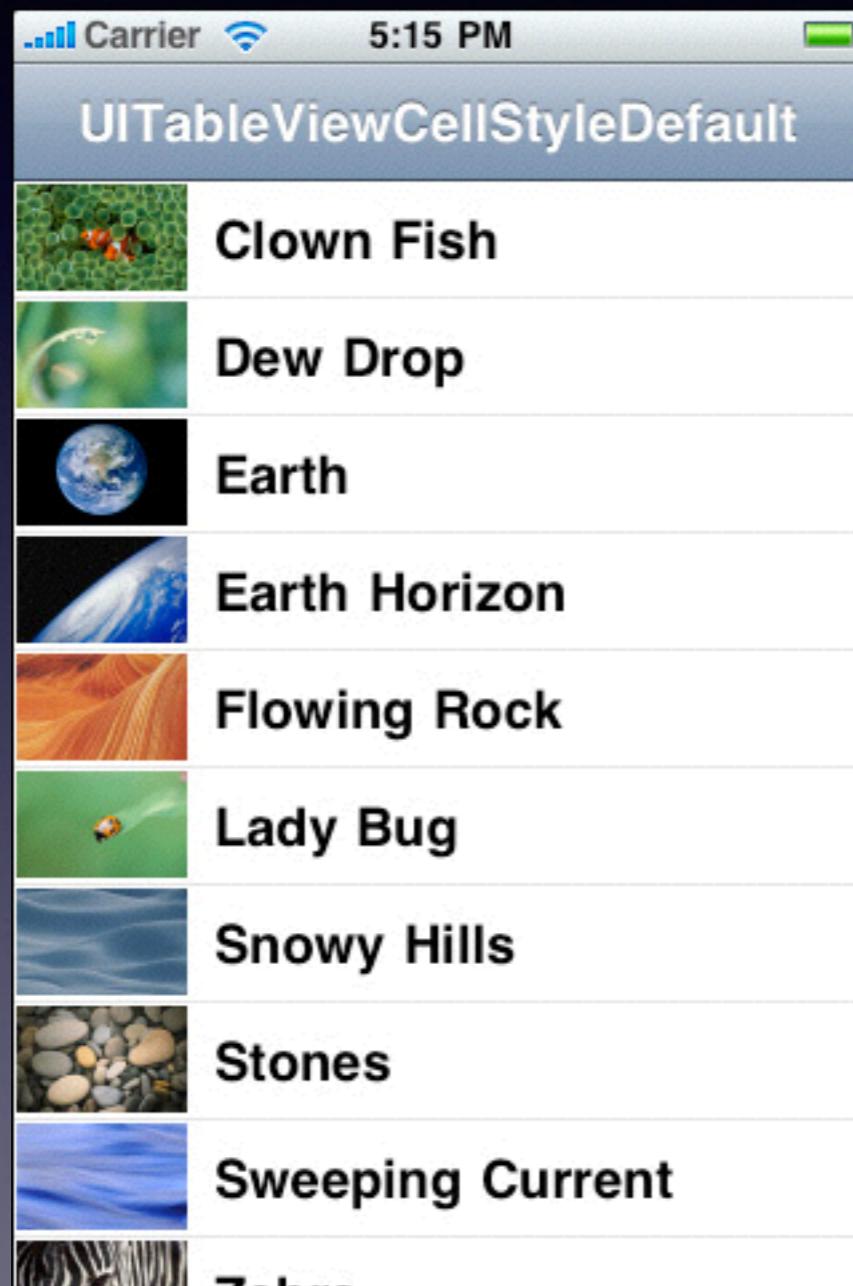
- Une vue représentant une cellule

```
@property(nonatomic,readonly,retain) UIImageView *imageView;
@property(nonatomic,readonly,retain) UILabel      *textLabel;
@property(nonatomic,readonly,retain) UILabel
*detailedTextLabel;
@property(nonatomic,readonly,retain) UIView       *contentView;
@property(nonatomic,retain)    UIView           *backgroundView;
```

- 4 style prédéfinis

```
typedef enum {
    UITableViewCellStyleDefault,
    UITableViewCellStyleValue1,
    UITableViewCellStyleValue2,
    UITableViewCellStyleSubtitle
} UITableViewCellStyle;
```

Les différents styles de cellules

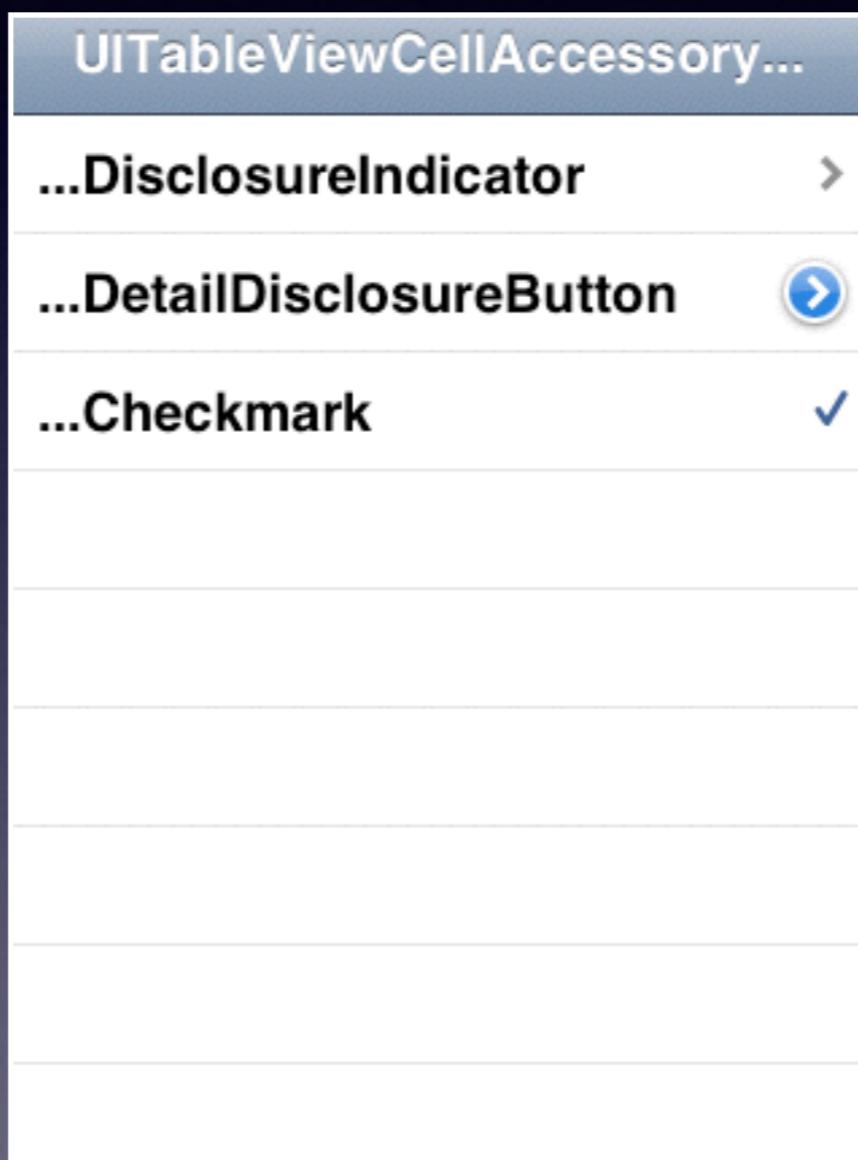


Les différents styles de cellules

UITableViewCellStyleValue1	
Clown Fish	Clown fish in coral
Dew Drop	Dew drop on a leaf
Earth	Photo of earth from space
Earth Hori...	Photo of earth's horizon from...
Flowing Rock	Striations in sandstone
Lady Bug	Ladybug on a leaf
Snowy Hills	Hills covered with snow
Stones	A collection of stones
Sweeping Cur...	Current sweeping past...
Zebra	Stripes of a zebra

UITableViewCellStyleValue2	
Clown Fi...	Clown fish in coral
Dew Drop	Dew drop on a leaf
Earth	Photo of earth from space
Earth...	Photo of earth's horizon from...
Flowing...	Striations in sandstone
Lady Bug	Ladybug on a leaf
Snowy...	Hills covered with snow
Stones	A collection of stones
Sweepin...	Current sweeping past rocks
Zebra	Stripes of a zebra

AccessoryView



Création d'une cellule

- Via le delegate, dans la méthode dédiée
 - Appelé automatiquement
 - Appelé «au besoin» (call-by-need)
 - L'indexPath reçu correspond à la position de la cellule que la tableView cherche à afficher.
 - `(UITableViewCell *)tableView:(UITableView *)tableView
cellForRowAtIndexPath:(NSIndexPath *)indexPath{...}`

Création d'une cellule

- Etape 1 : Récupérer la cellule depuis le design StoryBoard

```
#define CELL_ID @"id de la cellule tel que dans le Storyboard"  
- (UITableViewCell *)tableView:(UITableView *)tableView  
cellForRowAtIndexPath:(NSIndexPath *)indexPath  
{  
    UITableViewCell * cell = [tableView  
dequeueReusableCellWithIdentifier:CELL_ID  
forIndexPath:indexPath];  
  
    // Configure the cell  
  
    return cell;  
}
```

Création d'une cellule

- Grace à l'indexPath, on peut récupérer l'objet à afficher et configurer la cell.

```
- (UITableViewCell *)tableView:(UITableView *)tableView
cellForRowAtIndexPath:(NSIndexPath *)indexPath
{
    Objet * obj = [_myListOfObject objectAtIndex:indexPath.row];

    UITableViewCell *cell = [tableView
dequeueReusableCellWithIdentifier:CELL_ID forIndexPath:
indexPath];

    cell.textLabel.text = obj.title;
    cell.detailTextLabel.text = obj.subtitle;
    cell.imageView.image = [UIImage imageNamed:obj.imageData];

    return cell;
}
```

reuseldentifier?

- Permet de ne créer qu'un petit nombre de cellules - en leur allouant un reuseldentifier
- Les cellules déjà créées et passant hors-écran sont placées dans une file d'attente
- On les récupère ensuite en demandant à la tableView de «dépiler» une cellule avec un reuseldentifier spécifique

UITableViewController

- Il est commun qu'un même viewController soit à la fois dataSource et delegate
- UITableViewController
`@property(nonatomic, strong) UITableView *tableView`
- Disponible depuis le storyBoard

Rafraîchir les id (iOS6)

- Fonctionnalité de UITableViewController

```
@property (nonatomic, retain) UIRefreshControl  
*refreshControl
```

- Créer / paramétrer un refreshControl

```
self.refreshControl = [[UIRefreshControl  
alloc] init];  
[self.refreshControl addTarget:self  
action:@selector(refresh)  
forControlEvents:UIControlEventValueChanged];
```

Démo

Les blocks

Passer vos instructions en paramètres

Qu'est-ce qu'un block

- Un block d'instruction
- Délimité par des accolades
- Passable en argument à une méthode

Exemples

```
- (void) displayAds
{
    [UIView animateWithDuration:.3
        animations:^{
            self.adView.center = CGPointMake(self.view.center.x,
                self.view.bounds.size.height + 40)
        }];
}

- (void) refresh
{
    [self.book fetchChapters: ^{
        self.chapterList = self.book.chapters;
        [self.tableView reloadData];
        [self endRefreshing];
    }];
}
```

Récupérer les variables

- Possibilité d'utiliser les variables declarées avant le block dans le block,
 - uniquement en read-only
- Possibilité de modifier les variables d'instances

Definir un type de block

- Utiliser la fonction C `typedef` pour créer vos type de block perso

```
// Block ne prenant pas de paramètre et ne retournant  
// pas d'information  
typedef void(^SimpleBlock)();
```

```
// Block recevant un index et un objet en entrée  
typedef void(^EnumerateBlock)(int idx, id object);
```

```
// Block retournant un BOOL et prenant un objet en  
// entrée  
typedef BOOL(^PredicateBlock)(id object);
```

Creer des méthodes utilisant des blocks

- Les types de blocks et les blocks s'utilise comme des types et des variables classique
 - `(void) executeSimpleBlock:(SimpleBlock)simpleBlock;`
 - `(void) enumerateObjectUsingBlock:(EnumerateBlock)enumerateBlock;`
 - `(NSArray *) objectPassingTest:(PredicateBlock)predicateBlock;`

Exécuter un block

```
- (void) executeSimpleBlock:(SimpleBlock)simpleBlock
{
    simpleBlock();
}

- (void) enumerateObjectUsingBlock:(EnumerateBlock)enumerateBlock
{
    for (int i = 0; i < self.chapters.count; i++) {
        enumerateBlock(i, [self.chapters objectAtIndex:i]);
    }
}
- (NSArray *) objectPassingTest:(PredicateBlock)predicateBlock
{
    for (int i = 0; i < self.chapters.count; i++)
        testResult = predicateBlock([self.chapters
objectAtIndex:i]);
}
```

Quand utiliser les blocks?

- Enumération / Tri sur des collections
- Completion Handler
- Animation
- Grand Central Dispatch

Grand Central Dispatch

- GCD est un API écrit en C
- L'idée est d'avoir plusieurs «queue d'opération»
- Le système lance les opérations des queue dans différents thread
- Si une queue est bloquée les autres, elles, continuent à fonctionner normalement

- Obtenir une queue pour le travail asynchrone

```
dispatch_get_global_queue(dispatch_queue_priority_t priority, unsigned long flags);
```

```
#define DISPATCH_QUEUE_PRIORITY_HIGH 2  
#define DISPATCH_QUEUE_PRIORITY_DEFAULT 0  
#define DISPATCH_QUEUE_PRIORITY_LOW (-2)
```

- Ajouter un block à une queue

```
dispatch_async(myQueue, ^{self.data = [NSData dataWithContentsOfURL:url]});
```

- Obtenir la queue principale

```
dispatch_get_main_queue();
```

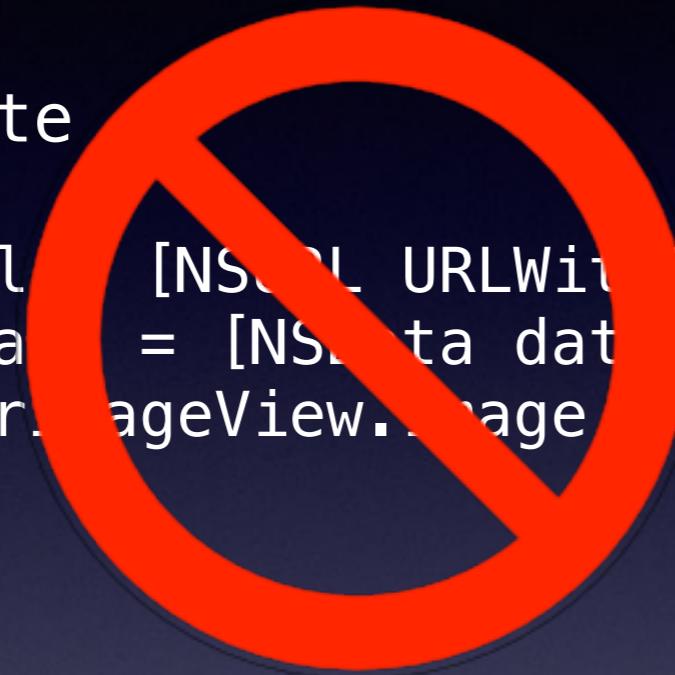
Chargement des images

```
- (void) update
{
    NSURL * url = [NSURL URLWithString:self.movie.imageURL];
    NSData * data = [NSData dataWithContentsOfURL:url];
    self.posterImageView.image = [UIImage imageWithData:data];
}
```

Probablement très long car on télécharge l'image à chaque fois et on bloque l'interface graphique...

Chargement des images

```
- (void) update
{
    NSURL * url = [NSURL URLWithString:self.movie.imageURL];
    NSData * data = [NSData dataWithContentsOfURL:url];
    self.posterImage.image = [UIImage imageWithData:data];
}
```



Comment utiliser le multithreading à notre avantage ici?

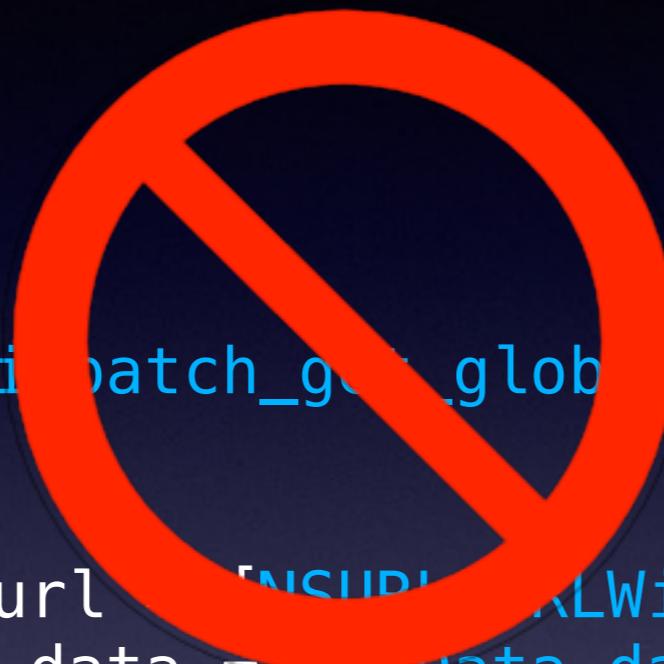
Chargement des images

```
- (void) update
{
    dispatch_async(dispatch_get_global_queue(DISPATCH_QUEUE_PRIORITY_HIGH, 0), ^{
        NSURL * url = [NSURL URLWithString: self.movie.imageURL];
        NSData * data = [NSData dataWithContentsOfURL:url];
        self.posterImageView.image = [UIImage imageWithData:data];
    });
}
```

Etape 1 : Traiter l'opération en asynchrone

Chargement des images

```
- (void) update
{
    dispatch_async(dispatch_get_global_queue(DISPATCH_QUEUE_PRIORITY_HIGH, 0), ^{
        NSURL * url = [NSURL URLWithString: self.movie.imageURL];
        NSData * data = [NSData dataWithContentsOfURL:url];
        self.posterImageView.image = [UIImage imageWithData:data];
    });
}
```



Problème : Les appels UIKit peuvent seulement être effectué dans le main thread
Comment contourner ce problème?

Chargement des images

```
- (void) update
{
    dispatch_async(dispatch_get_global_queue(DISPATCH_QUEUE_PRIORITY_H
IGH, 0), ^{
        NSURL * url = [NSURL URLWithString:self.movie.imageUrl];
        NSData * data = [NSData dataWithContentsOfURL:url];
        dispatch_async(dispatch_get_main_queue(), ^{
            self.posterImageView.image = [UIImage
imageWithData:data];
        });
    });
}
```

Solution : ressortir les appels UIKit dans le main thread

Chargement des images

```
- (void) update
{
    dispatch_async(dispatch_get_global_queue(DISPATCH_QUEUE_PRIORITY_H
IGH, 0), ^{
        NSURL * url = [NSURL URLWithString:self.movie.imageURL];
        NSData * data = [NSData dataWithContentsOfURL:url];
        dispatch_async(dispatch_get_main_queue(), ^{
            self.posterImageView.image = [UIImage
imageWithData:data];
        });
    });
}
```



Démo

Persistiance

La mémoire de votre app

Plusieurs possibilités:

- Preferences
- Le système de fichier
- Core Data
- Cache
- Document

Preferences

- Utilisation de la classe NSUserDefaults

```
NSUserDefaults * preferences = [NSUserDefaults standardUserDefaults];
```

- Manipuler les paires clefs - valeurs

- `(id)objectForKey:(NSString *)defaultName;`
- `(void)setObject:(id)value forKey:(NSString *)defaultName;`
- `(void)removeObjectForKey:(NSString *)defaultName;`

- Peut être réinitialisé

- + `(void)resetStandardUserDefaults;`

L’arborescence iOS

- iOS à un système de fichier Unix
- Votre application peut «écrire» uniquement dans votre «sandbox»
 - sécurité
 - vie privée
 - propreté (en cas de suppression de l’app)

le gestionnaire de fichier

- `NSFileManager`
 - Vérifier l'existence d'un fichier
 - Copier, déplacer, créer, énumérer...
 - Retrouver l'url d'un dossier système
 - Thread safe

```
NSFileManager * fileManager = [[NSFileManager alloc]
init];
```

Dossier système?

- Votre sandbox contient déjà :
 - Dossier Bundle
 - lecture seule
- NSDocumentDirectory
 - Sauvegardé automatiquement
- cf NSSearchPathDirectory

A quelle adresse?

- Demande au fileManager
 - `(NSArray *)URLsForDirectory:(NSSearchPathDirectory)directory
inDomains:(NSSearchPathDomainMask)domainMask;`
- Sur iOS, on utilise le `NSUserDomainMask`
- Retourne une `NSArray` de `NSURL` ...
 - Système mono-utilisateur = 1 seul objet dans l'array
 - On utilise le `lastObject`

Construire une URL

- Méthodes utiles de `NSString`

- + `(NSString *)pathWithComponents:(NSArray *)components;`
- `(NSArray *)pathComponents;`
- `(NSString *)lastPathComponent;`
- `(NSString *)stringByDeletingLastPathComponent;`
- `(NSString *)stringByAppendingPathComponent:(NSString *)str;`
- `(NSString *)pathExtension;`
- `(NSString *)stringByDeletingPathExtension;`
- `(NSString *)stringByAppendingPathExtension:(NSString *)str;`

Ecrire / Lire

- En passant par **NSString** (texte brute)
- Ecriture
 - **(BOOL)writeToURL:(NSURL *)url atomically:(BOOL)useAuxiliaryFile encoding:(NSStringEncoding)enc error:(NSError **)error;**
- Lecture
 - + **(id)NSStringWithContentsOfFile:(NSString *)path encoding:(NSStringEncoding)enc error:(NSError **)error;**

Ecrire / Lire

- En passant par NSData
- Ecriture
 - `(BOOL)writeToURL:(NSURL *)url atomically:(BOOL)atomically;`
- Lecture
 - + `(id)dataWithContentsOfURL:(NSURL *)url;`

SQLite

- Base de données simple (1 seul fichier)
- supporte le langage SQL
- API en C

Core Data

- API + ORM de SQLite
- Meilleure gestion du modèle relationnel
- Pas de language SQL
- !!! Les classes CoreData ne sont pas ‘thread-safe’
- Tous les appels doivent être faits dans la queue principale

Definir le schemas de base de données

- Utilisation d'un outil WYSIWYG
 - Chaque table est une **entity**
 - Une **entity** contient plusieurs **attributes**
 - Deux entities peuvent être liés par une **relationship**
- Génération automatisée des classes

Récupérer la database

- Crée à la demande par la classe AppDelegate

```
@property (readonly, strong, nonatomic)  
NSManagedObjectContext *managedObjectContext;
```

- Récuperation de la base de données

```
AppDelegate * appDelegate = [[UIApplication  
sharedApplication] delegate];  
NSManagedObjectContext * context =  
appDelegate.managedObjectContext;
```

Ajouter un objet

!!!! Ne jamais utiliser alloc - init

- On ajoute une ligne vide dans la DB

```
Employee * newEmployee = [NSEntityDescription  
insertNewObjectForEntityForName:@"Employee"  
inManagedObjectContext:context];
```

- Affecter les valeurs a l'enregistrement

```
newEmployee.name = @"Florian";  
newEmployee.startDate = [NSDate date];
```

Interroger la base de données

- On utilise la classe `NSFetchRequest` pour créer une requête (SELECT)

```
NSFetchRequest * request = [[NSFetchRequest  
alloc] initWithEntityName:@"Employee"];
```

- Ajout d'un filtre conditionnel (WHERE)

```
request.predicate = [NSPredicate  
predicateWithFormat:@"department == %@", AND region = %@  
", @"sales", @"europe"];
```

NSFetchRequest

- Tri des résultat (ORDER BY)

```
NSSortDescriptor * sd = [NSSortDescriptor  
sortDescriptorWithKey:@"salesThisYear" ascending:YES];  
request.sortDescriptors = @[sd];
```

- Limite du nombre de résultat retourné

```
request.fetchLimit = 1;
```

- Execution de la requête

```
NSArray * result = [context  
executeFetchRequest:request error:nil];
```

Suppression

- Pas de possibilité de faire une requête de suppression globale
- Supprimer l'élément à la fois
 - `(void) fireEmployee:(Employee *)employe { [context deleteObject:employe]; }`

Sauvegarder les changements

- Modification en base non persistante
- Necessaire de sauvegarder les changement
- Méthodes de NSManagedObjectContext
 - `(BOOL)hasChanges;`
 - `(BOOL)save:(NSError **)error;`

Quand sauvegarder

- Quand on veut :-)
- Attention à ne pas trop sauvegarder!
- Quand l'application devient inactive
 - callback dans le AppDelegate
 - Le AppDelegate fournit une méthode centralisé ‘saveContext’

Démo

Core Location

API de Géolocalisation

Core Location

- API de géolocalisation
 - Connaître la position de l'utilisateur
 - *Geocoding*
 - *Region Monitoring*

Obtenir une localisation

- Objet de base : **CLLocation**

```
//Coordonnées : longitude, latitude
@property(nonatomic, readonly) CLLocationCoordinate2D coordinate;

// Direction obtenue via la boussole (double: de 0 à 359.99)
@property(nonatomic, readonly) CLLocationDirection course;

// Vitesse en mètres par seconde
@property(nonatomic, readonly) CLLocationSpeed speed;

// Altitude en mètres
@property(nonatomic, readonly) CLLocationDistance altitude;

@property(nonatomic, readonly) NSDate *timestamp;
```

Comment obtenir un CLLocation

- On passe par un CLLocationManager
 - Vérifier la possibilité de se localiser
 - Créer un CLLocationManager
 - Configurer le manager (type, précision)
 - Se déclarer en delegate
 - Démarrer le manager

CLLocationManager

- Vérifier les possibilités matérielles
 - + (**BOOL**)locationServicesEnabled
 - + (**BOOL**)headingAvailable
 - + (**BOOL**)significantLocationChangeMonitoringAvailable
 - + (**BOOL**)regionMonitoringEnabled

Definir la precision

- Constantes prédefinies

```
kCLLocationAccuracyBestForNavigation;  
kCLLocationAccuracyBest;  
kCLLocationAccuracyNearestTenMeters;  
kCLLocationAccuracyHundredMeters;  
kCLLocationAccuracyKilometer;  
kCLLocationAccuracyThreeKilometers;
```

- `@property desiredAccuracy` du location manager

```
self.locationManager.desiredAccuracy =  
kCLLocationAccuracyHundredMeters;
```

Type de localisation

- Localisation en continue

```
@property(assign) CLLocationDistance distanceFilter;  
- (void)startUpdatingLocation;  
- (void)stopUpdatingLocation;
```

- Boussole en continu

```
@property(assign, nonatomic) CLLocationDegrees  
headingFilter  
- (void)startUpdatingHeading;  
- (void)stopUpdatingHeading;
```

- Notifié même si l'application est arrêté

Type de localisation

- Observer une région en particulier
 - notifie quand on entre une CLRegion
 - (`void`)`startMonitoringForRegion:(CLRegion *)region`;
 - (`void`)`stopMonitoringForRegion:(CLRegion *)region`;
 - Observer des déplacements «significatifs»
 - Utilise probablement les tours cellulaires
 - (`void`)`startMonitoringSignificantLocationChanges`;
 - (`void`)`stopMonitoringSignificantLocationChanges`;

CLLocationManagerDelegate

- Méthodes optionnelles
- Réponse à écouter:
 - `(void)locationManager:(CLLocationManager *)manager didUpdateLocations:(NSArray *)locations`
 - `(void)locationManager:(CLLocationManager *)manager didUpdateHeading:(CLHeading *)newHeading`
 - `(void)locationManager:(CLLocationManager *)manager didEnterRegion:(CLRegion *)region`
 - `(void)locationManager:(CLLocationManager *)manager didExitRegion:(CLRegion *)region`
 - `(void)locationManager:(CLLocationManager *)manager didFailWithError:(NSError *)error;`

Mapkit

Cartographie

MapKit

- API graphique
- Affiche une localisation sur la planète
- Affiche une carte et des annotations

MapKit

- Framework permettant d'inclure des cartes directement dans votre application

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

- Carte (zoom, déplacement)
- Annotations avec callout
- Titre, sous-titre, accessoryViews

MapView

- S'ajoute en code ou depuis IB
- Contrôler le comportement

```
@property(nonatomic, getter=isZoomEnabled) BOOL  
zoomEnabled;  
@property(nonatomic, getter=isScrollEnabled) BOOL  
scrollEnabled;  
@property (nonatomic) MKMapType mapType;
```

- Afficher la localisation utilisateur

```
@property (nonatomic) BOOL showsUserLocation;
```

Changer de région

- Définir la region de la carte

```
@property (nonatomic) MKCoordinateRegion region;  
- (void)setRegion:(MKCoordinateRegion)region animated:(BOOL)animated;
```

- Centrer sa vue (sans changer le span)

```
@property (nonatomic) CLLocationCoordinate2D centerCoordinate;  
- (void)setCenterCoordinate:(CLLocationCoordinate2D)coordinate animated:(BOOL)animated;
```

Region et Span

```
typedef struct {
    CLLocationDegrees latitudeDelta;
    CLLocationDegrees longitudeDelta;
} MKCoordinateSpan;

typedef struct {
    CLLocationCoordinate2D center;
    MKCoordinateSpan span;
} MKCoordinateRegion;

// Exemple (non animé)
self.mapView.region =
MKCoordinateRegionMake(aLocation.coordinate,
MKCoordinateSpanMake(STANDARD_ZOOM_LEVEL,
STANDARD_ZOOM_LEVEL));
```

MKMapViewDelegate

- Notifications sur changement de region
- Notification sur chargement carte
- Controle des annotations
 - Apparence (defaut = )
 - Notification sur clic

Annotations

- Objet se conformant au protocole MKAnnotation

```
// Coordonée de l'annotation (obligatoire)
@property (nonatomic, readonly) CLLocationCoordinate2D
coordinate;

// Titre et sous-titre affiché dans le callout (optional)
@property (nonatomic, readonly, copy) NSString *title;
@property (nonatomic, readonly, copy) NSString *subtitle;

// Appeler si l'annotation est déplacé sur la carte
(optional)
- (void)setCoordinate:
(CLLocationCoordinate2D)newCoordinate;
```

MKAnnotations

- Obtenir la liste des annotations de la mapView (read-only!!!)

```
@property (nonatomic, readonly) NSArray *annotations;
```

- Ajouter / supprimer des annotations

- (void)addAnnotation:(id <MKAnnotation>)annotation;
- (void)addAnnotations:(NSArray *)annotations;
- (void)removeAnnotation:(id <MKAnnotation>)annotation;
- (void)removeAnnotations:(NSArray *)annotations;

- Vue par défaut :



Apparence d'une annotation

- Sous classe de MKAnnotationView
- Par défaut MKPinAnnotation

```
@property (nonatomic) MKPinAnnotationColor pinColor;  
(MKPinAnnotationColorRed, MKPinAnnotationColorGreen,  
MKPinAnnotationColorPurple)  
@property (nonatomic) BOOL animatesDrop;
```

- Possibilité de créer notre propre vue
 - (MKAnnotationView *)mapView:(MKMapView *)mapView
viewForAnnotation:(id <MKAnnotation>)annotation;

Exemple d'implémentation

```
if([annotation isKindOfClass:[MKUserLocation class]])
    return nil;

MKPinAnnotationView * view = (MKPinAnnotationView *)
[mapView
dequeueReusableCellWithIdentifier:REUSE_ID];

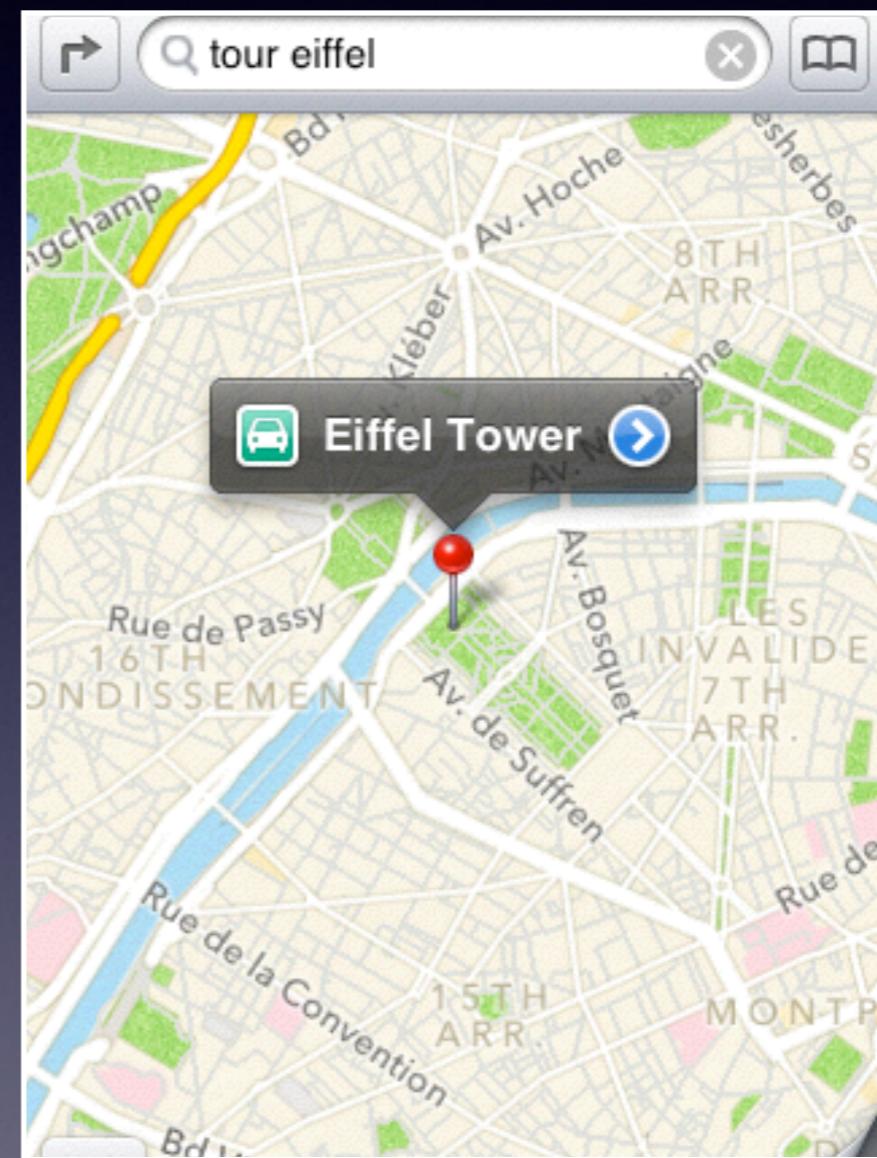
if(!view) view = [[[MKPinAnnotationView alloc]
initWithAnnotation:annotation reuseIdentifier:reuseID];

view.pinColor = MKPinAnnotationColorPurple;
view.canShowCallout = YES;

view.annotation = annotation;
return view;
```

CalloutView

- Que se passe-t-il lorsque l'on clique sur une annotation?
 - affichage d'un callout
 - Affichage du title, subtitle de l'annotation
 - Possibilité d'ajouter des accessoryViews



Personnaliser le callout

- Properties de MKAnnotationView

```
@property (retain, nonatomic) UIView  
*leftCalloutAccessoryView;  
@property (retain, nonatomic) UIView  
*rightCalloutAccessoryView;
```

- Si UIControl, le delegate est notifié lors du tap

```
- (void)mapView:(MKMapView *)mapView annotationView:  
(MKAnnotationView *)view calloutAccessoryControlTapped:  
(UIControl *)control;
```