Programmation iOS
L3 informatique

Étienne Payet

Département de mathématiques et d'informatique

Plan

Ressources

- http://developer.apple.com
- Standford CS193P (niveau L2-L3)
  http://www.stanford.edu/class/cs193p/cgi-bin/drupal/
  Vidéos sur iTunes U
- livres : il y en a beaucoup...
  - *iOS Programming : The Big Nerd Ranch Guide* (Conway & Hillegass)
  - voir à la BU
  - privilégiez ceux traitant des dernières versions d'iOS
  - Objective C 2.0

Pour pouvoir travailler

- développement : Mac Intel + OS X $\geq$ 10.6.6 (Snow Leopard) + Xcode + iOS SDK
- déploiement : Apple ID + appareil iOS
- par défaut, tout développeur a accès à Xcode, à l'iOS SDK et aux forums, peut déployer sur un appareil iOS mais ne peut pas distribuer son application sur l'App Store
- des abonnements payants sont proposés à ceux qui veulent distribuer leur application

Workflow iOS

1. Développement	2. Test	3. Publication
Xcode	Émulateur ou Appareil mobile (iPhone, iPad, iPod Touch)	App Store

Les frameworks d'iOS

Core OS

noyau OS X (BSD - Mach 3.0),
sockets, système de fichiers, ...

Les frameworks d'iOS

Core Services	services réseau, SQLite, contacts, préférences, géo-localisation, ...
Core OS	noyau OS X (BSD - Mach 3.0), sockets, système de fichiers, ...

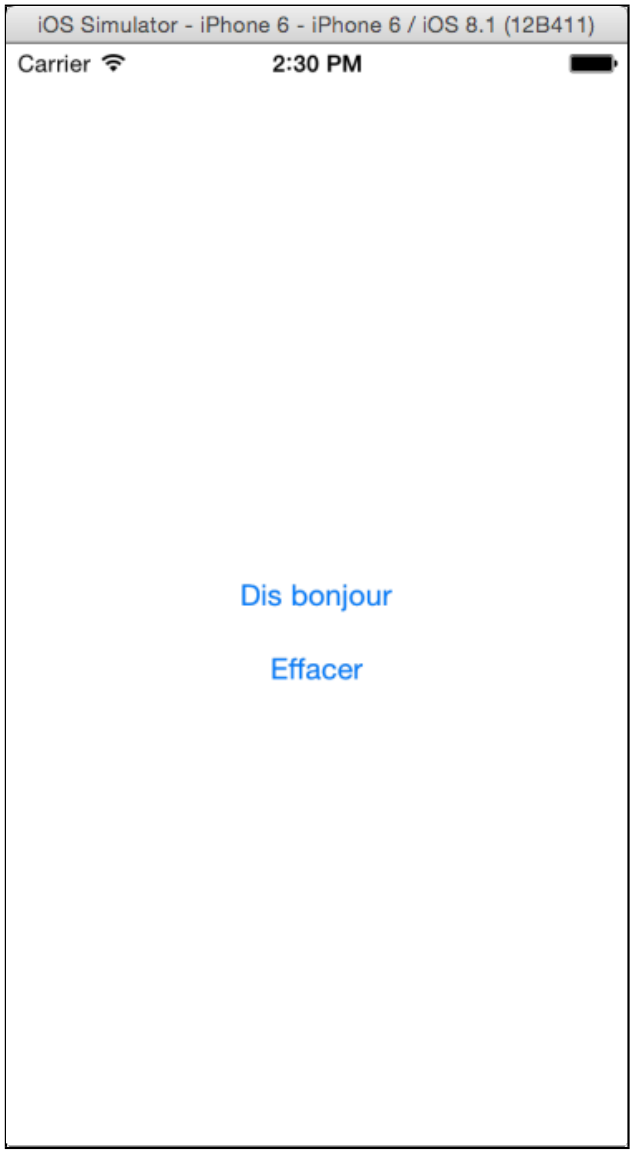
Les frameworks d'iOS

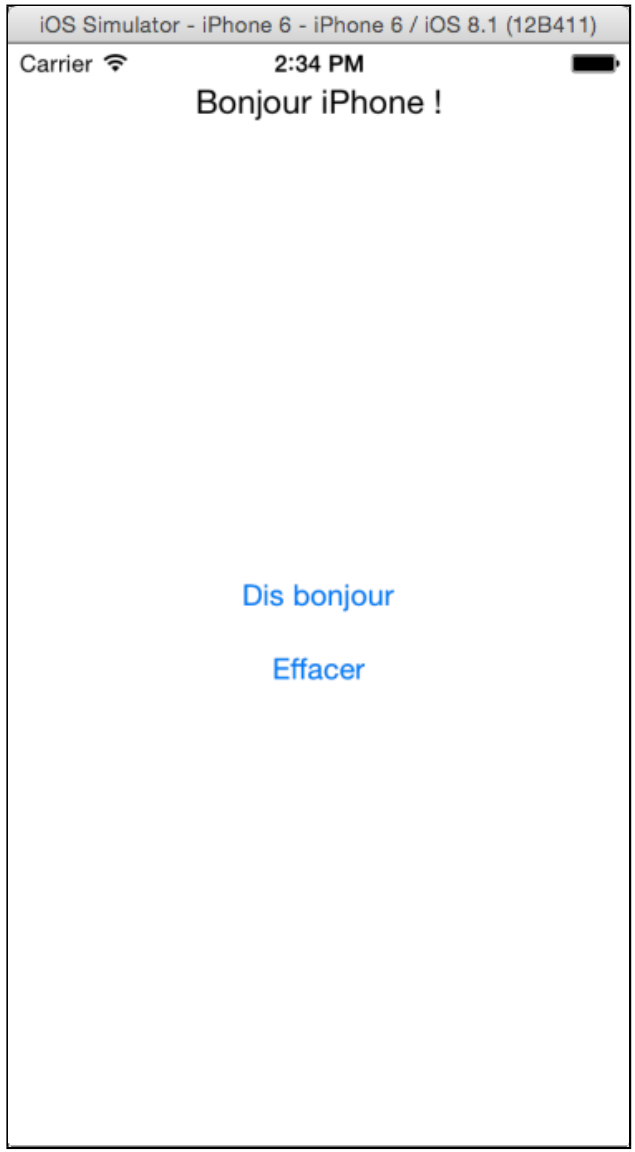
Media	PDF, JPG, PNG, TIFF, audio, vidéo, animations, OpenGL, ...
Core Services	services réseau, SQLite, contacts, préférences, géo-localisation, ...
Core OS	noyau OS X (BSD - Mach 3.0), sockets, système de fichiers, ...

Les frameworks d'iOS

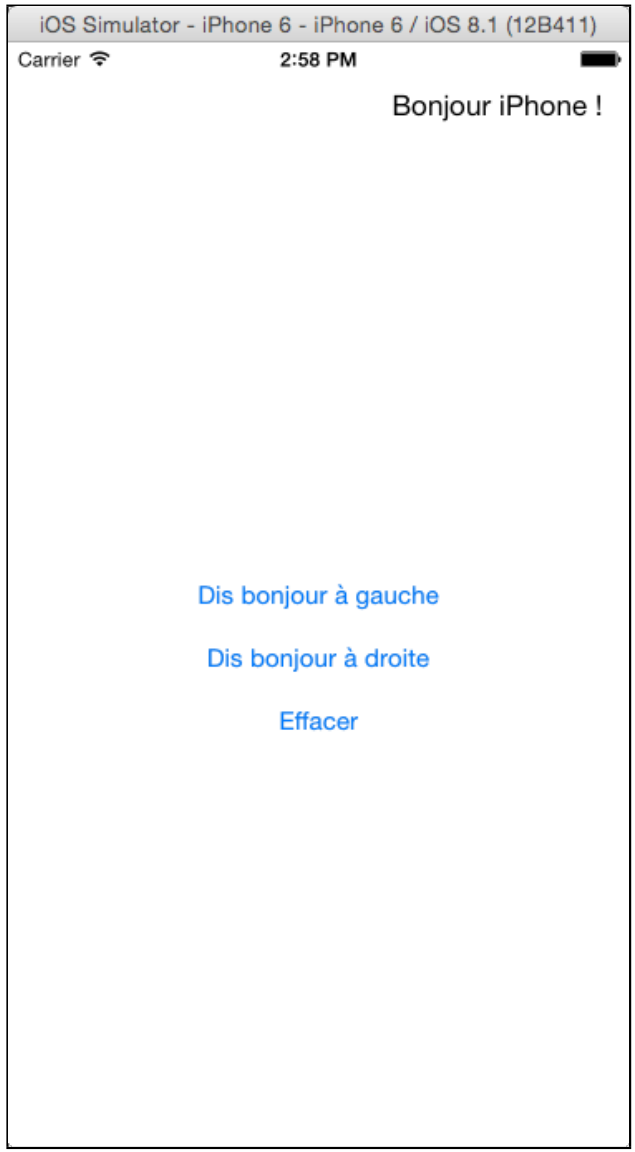
Cocoa Touch	UIKit, Foundation, MapKit, ...
Media	PDF, JPG, PNG, TIFF, audio, vidéo, animations, OpenGL, ...
Core Services	services réseau, SQLite, contacts, préférences, géo-localisation, ...
Core OS	noyau OS X (BSD - Mach 3.0), sockets, système de fichiers, ...

Exercice : *Hello World !*





Exercice : *Hello World !*



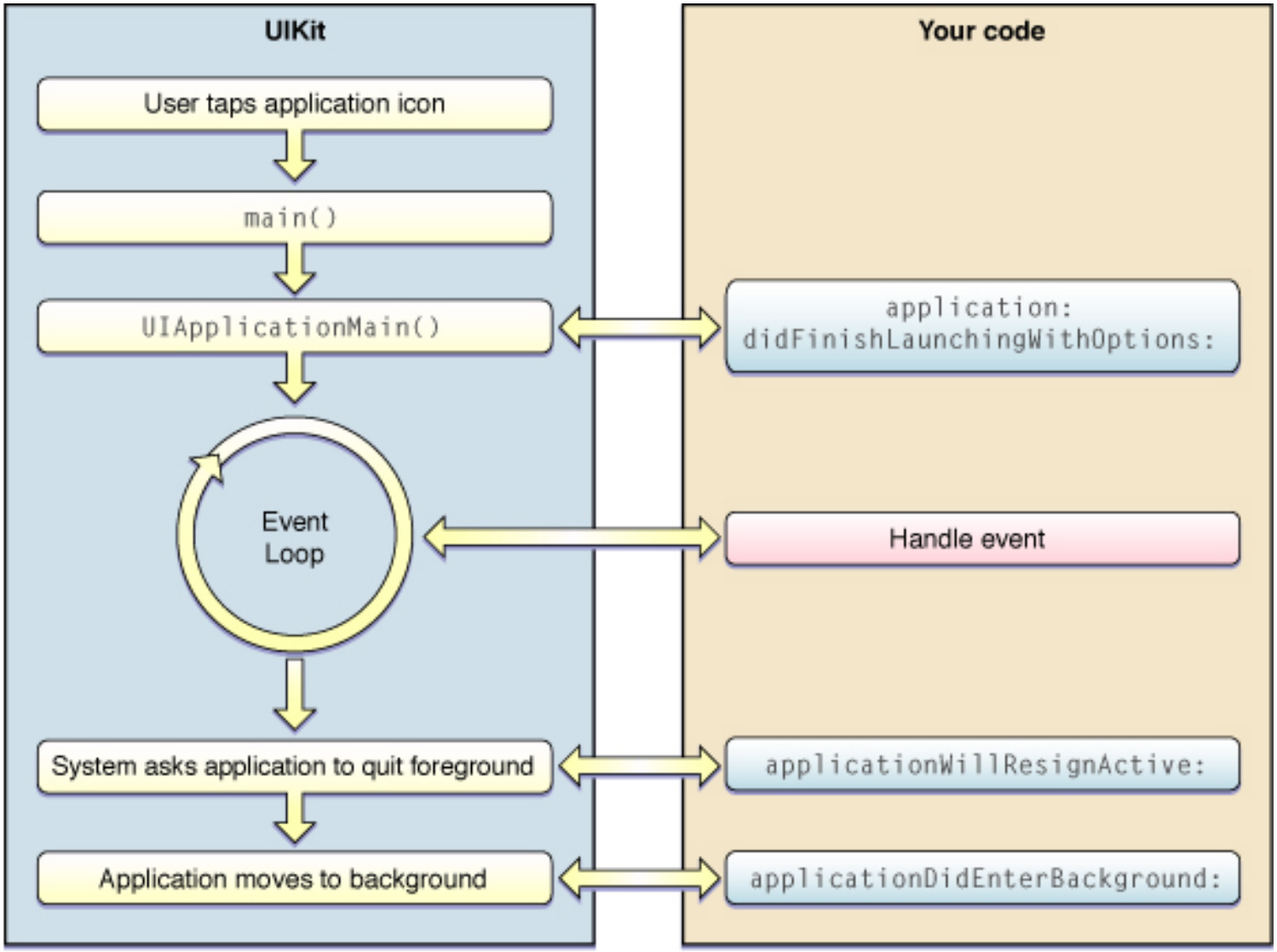
Plan

Anatomie d'une application iOS

voir ici pour une description complète

certains éléments sont présentés ci-après

Cycle de vie d'une application



La fonction `main` : transfère le contrôle à UIKit

```
#import <UIKit/UIKit.h>
#import "AppDelegate.h"

int main(int argc, char * argv[])
{
  @autoreleasepool {
    return UIApplicationMain(argc, argv, nil,
                             NSStringFromClass([AppDelegate class]));
  }
}
```

La fonction `UIApplicationMain`

- crée l'objet *application delegate*
- charge l'interface graphique à partir des fichiers Storyboard
- initialise l'application
- lance la boucle des événements

L'objet *application delegate*

- présent dans *toute* application iOS
- gère les messages système
- implémente le protocole `UIApplicationDelegate`

Le protocole UIApplicationDelegate

Ses méthodes permettent de répondre aux événements du cycle de vie :

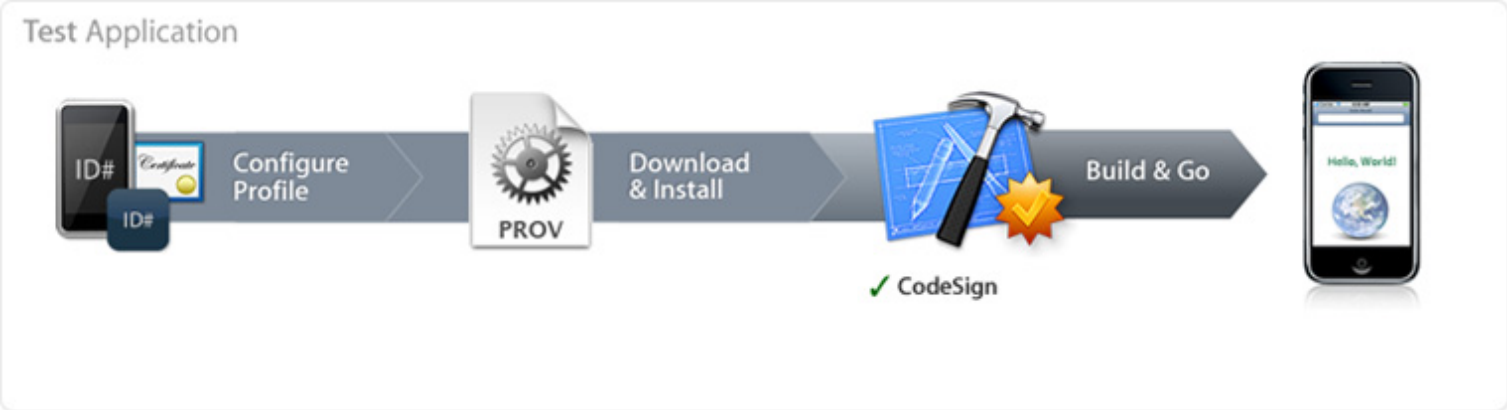
- `application:didFinishLaunchingWithOptions:`
- `applicationDidBecomeActive:`
- `applicationWillResignActive:`
- `applicationDidEnterBackground:`
- `applicationWillEnterForeground:`
- `applicationWillTerminate:`

Exercice

Repérez tous ces éléments dans votre application *Hello World !*

Plan

Processus de déploiement



Provisioning profile

- basé sur la notion d'équipe (*team*)
- peut être créé automatiquement par Xcode à partir de l'équipe associée au projet ou peut être créé par le chef d'équipe

- constitué de :
  - certificats développeurs (un ou plus)
  - iDevices IDs (un ou plus)
  - un App ID (un seul)
- à installer sur les iDevices (à partir de Xcode)

Exercice

Déployez votre application *Hello World !* sur votre iDevice.

Plan

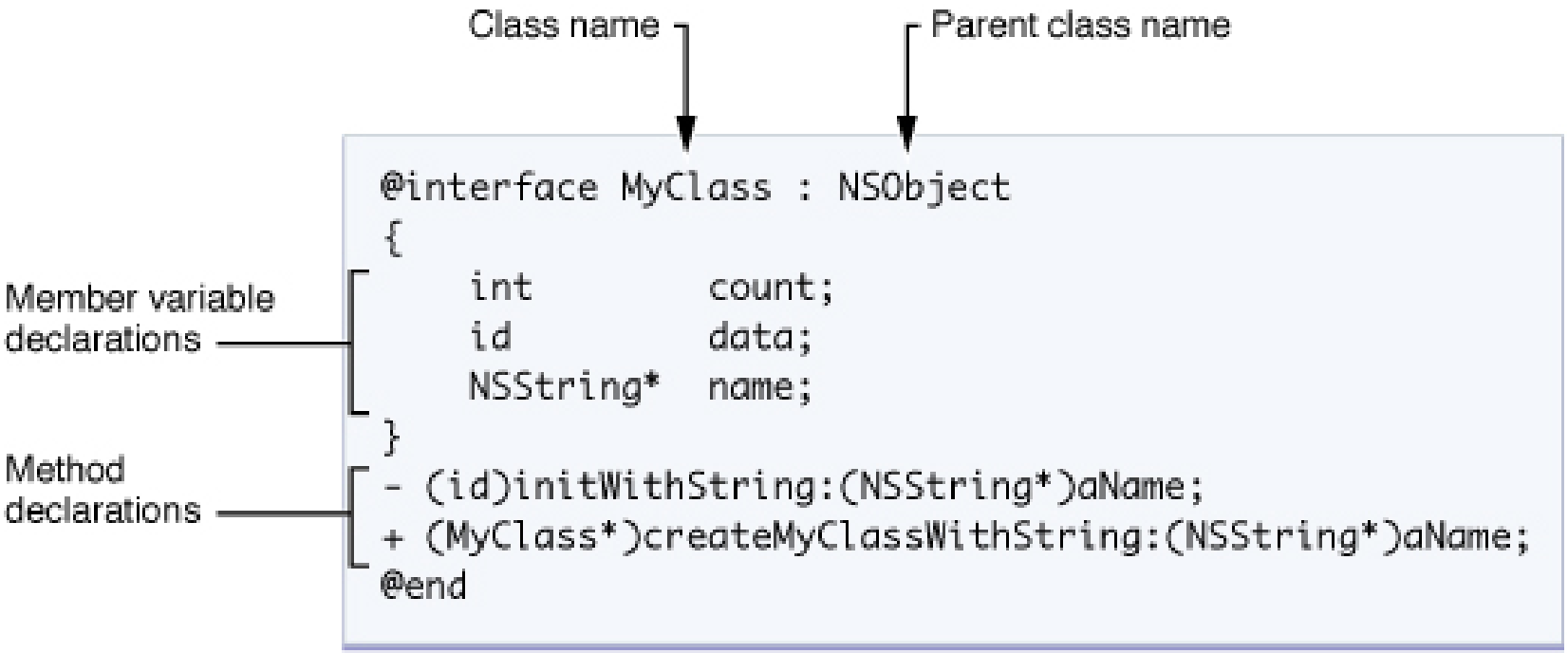
Sur-ensemble strict du C ANSI

programmation orientée objet :

- classes
- méthodes, envoi de messages
- héritage, délégation (protocoles)
- introspection
- ...

Classes : déclaration

fichier d'en-tête (`.h`)

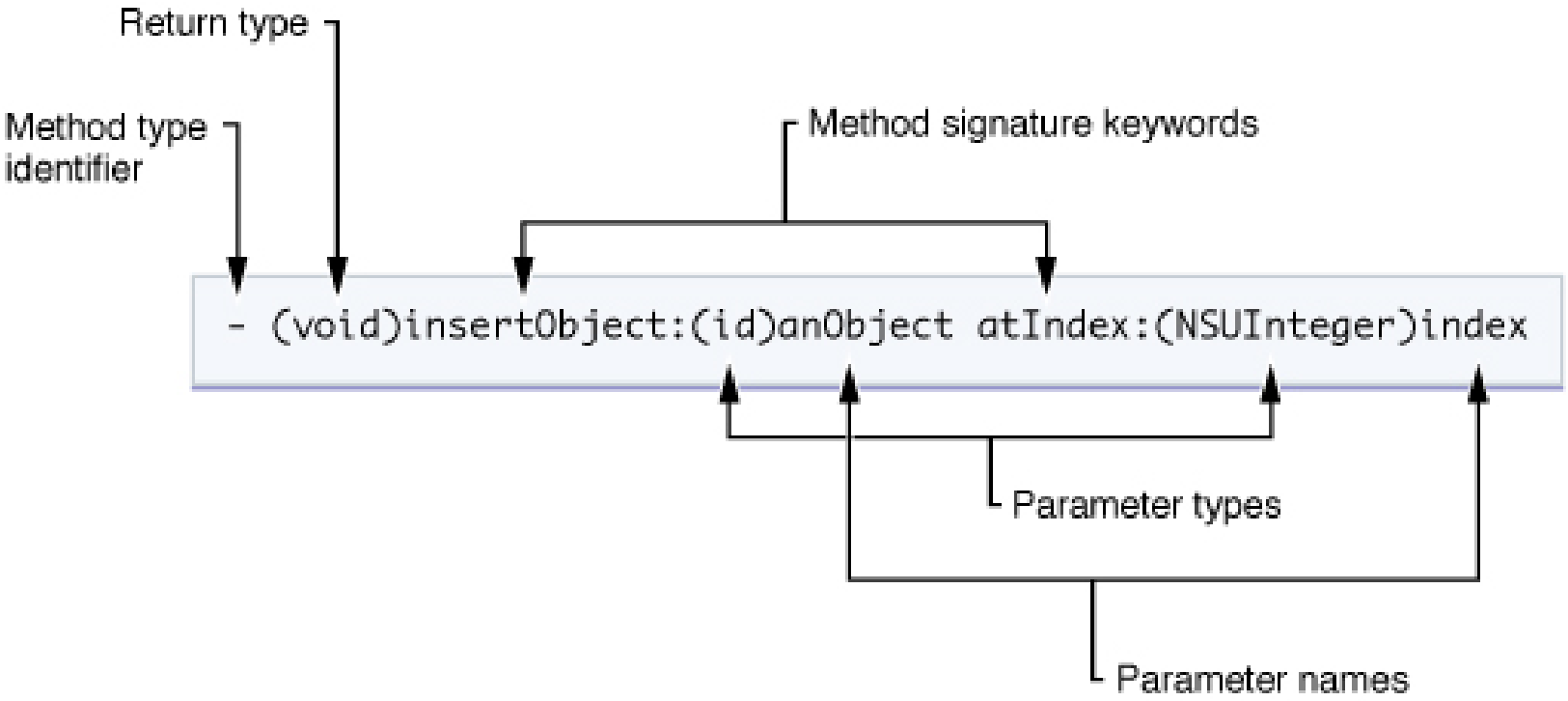


Classes : implémentation

fichier source (`.m`)

```
#import "MyClass.h"

@implementation MyClass
- (id)initWithString:(NSString *)aName {
    self = [super init];
    if (self) {
        name = [aName copy];
    }
    return self;
}

+ (MyClass *)createMyClassWithString: (NSString *)aName {
    return [[[self alloc] initWithString:aName] autorelease];
}
@end
```

Méthodes : déclaration



Méthodes : appel

envoi de messages :

- syntaxe : `[myArray insertObject:anObject atIndex:0]`

  obligatoire destinataire sélecteur paramètres

- imbrication :

```
[[myAppObject theArray]
 insertObject:[myAppObject objectToInsert]
 atIndex:0
]
```

Chaînes de caractères

- types `NSString` et `NSMutableString`
- constantes : `@"Hello World!"`
- déclaration :

```
NSString * ma_chaine = @"Hello World!";
```

Chaînes de caractères

```
NSString *meteo = @"beau";
NSString *message = [NSString stringWithFormat:@"Il fait %@", meteo];
```

```
NSLog(@"il fait %d degres", [meteo temperature]);
```

Chaînes de caractères

```
NSString *ma_chaine = @"Hello";
NSString *chaine;
chaine = [ma_chaine stringByAppendingString: @" World!"];
```

```
NSMutableString *ma_chaine = [NSMutableString string];
[ma_chaine appendString:@"Meteo : "];
[ma_chaine appendFormat:@"ciel %@ temperature %d degres",
                                    [meteo ciel], [meteo temperature]];
```

Collections

- ordonnées : `NSArray` et `NSMutableArray`
- non-ordonnées, sans doublons : `NSSet` et `NSMutableSet`
- couples {clé,valeur} : `NSDictionary` et `NSMutableDictionary`

NSArray et NSMutableArray

```
NSArray *couleurs = [NSArray arrayWithObjects:@"vert",@"bleu",nil];
NSLog(@"nombre de couleurs : %d", [couleurs count]);
NSLog(@"troisieme couleur : %@", [couleurs objectAtIndex:2]);
```

```
NSMutableArray *couleurs = [NSMutableArray array];
[couleurs addObject:@"vert"];
[couleurs addObject:@"bleu"];
[couleurs insertObject:@"jaune" atIndex:1];
[couleurs removeObjectAtIndex:0];
```

NSDictionary et NSMutableDictionary

```
NSDictionary *couleurs =
  [NSDictionary dictionaryWithObjectsAndKeys:
    @"vert",@"00FF00",
    @"bleu",@"0000FF",
    nil];
NSString *bleu = [couleurs objectForKey:@"0000FF"];
if ([couleurs objectForKey:@"FFFFFF"]) NSLog(@"le blanc existe !");
```

```
NSMutableDictionary *couleurs = [NSMutableDictionary dictionary];
[couleurs setObject:@"vert" forKey:@"00FF00"];
[couleurs removeObjectForKey:@"000000"];
[couleurs removeAllObjects];
```

Fast enumeration

peu efficace :

```
NSArray *couleurs = [NSArray arrayWithObjects:@"vert",@"bleu",nil];
int combien = [couleurs count];
for (int i = 0; i < combien; i++) {
   NSLog(@"couleur %@", [couleurs objectAtIndex:i]);
}
```

efficace :

```
for (NSString *uneCouleur in couleurs)
   NSLog(@"couleur %@", uneCouleur);
```

Gestion mémoire : création d'un objet

2 étapes :

1. allocation mémoire : `+(id)alloc`
2. initialisation de l'objet : `-(id)init`

```
Etudiant * unEtudiant = [[Etudiant alloc] init];
```

Gestion mémoire : initialisation d'un objet

- re-définition de la méthode `init` :

```
-(id)init {
    if (self = [super init]) {
        numero = 0;
        nom = @"";
    }
    return self;
}
```

- définition de méthodes `init`-like :
  - `-(id)initWithNumero:(int)unNumero;`
  - `-(id)initWithNumero:(int)unNumero nom:(NSString*)unNom;`

Gestion mémoire : destruction d'un objet

- pas de garbage-collector
- le dual de `+(id)alloc` est `-(void)dealloc`
- ne jamais appeler `dealloc`
- trouver l'équilibre allocation/désallocation

Gestion mémoire : compter les références

- chaque objet a un compteur de références :
  - tant que compteur > 0, l'objet peut vivre
  - dès que compteur ≤ 0, l'objet est détruit
- `+(id)alloc` crée un objet avec compteur $= 1$
- `-(id)retain` incrémente le compteur $(+1)$
- `-(void)release` décrémente le compteur (-1)

Gestion mémoire : trouver l'équilibre

```
Etudiant * unEtudiant = [[Etudiant alloc] initWithNumero:007];
...
[unEtudiant release];
```

```
Etudiant * unEtudiant = [[Etudiant alloc] initWithNumero:007];
...
[unEtudiant release];
[unEtudiant identite]; // CRASH !!
```

Gestion mémoire : champs d'un objet

classe `Etudiant` avec les champs d'instance `nom` et `binome` :

```
- (void)setNom:(NSString*)unNom {
    if (nom != unNom) {
      [nom release];
      nom = [unNom copy];
    }
}
- (void)setBinome:(Etudiant*)unBinome {
    if (binome != unBinome) {
      [binome release];
      binome = [unBinome retain];
    }
}
-(void)dealloc {
    [nom release];
    [binome release];
}
```

Gestion mémoire : méthode retournant un objet

```
- (NSString*)identite {
    NSString * resultat;
    resultat = [[NSString alloc] initWithFormat:@"%d-%@",numero,nom];
    [resultat release];
    return resultat;
}
```

NON : à la fin de la méthode, la chaîne de caractères créée est desallouée

Gestion mémoire : méthode retournant un objet

```
- (NSString*)identite {
    NSString * resultat;
    resultat = [[NSString alloc] initWithFormat:@"%d-%@",numero,nom];
    return resultat;
}
```

OK : l'objet qui reçoit le résultat doit se charger du `release`

Gestion mémoire : méthode retournant un objet

`autorelease` :

- indique qu'un `release` va être envoyé à l'objet ... bientôt
- équivalent à un sursis

```
- (NSString*)identite {
   NSString * resultat;
   resultat = [[NSString alloc] initWithFormat:@"%d-%@",numero,nom];
   return [resultat autorelease]; //resultat existe... pour le moment
}
```

OK : l'objet qui reçoit le résultat doit faire un `retain`

Gestion mémoire : comment savoir ?

- l'objet qui reçoit doit-il faire `release` ou `retain` ?
- **convention :**
  - si le nom de la méthode appelée contient alloc, init ou copy, l'objet qui reçoit le résultat doit se charger du `release`
  - sinon, le résultat est en `autorelease`

Propriétés

pour chaque attribut :

- remplacent la déclaration et l'implémentation des méthodes d'accès (getters/setters)
- des qualificatifs permettent de spécifier le comportement en environnement multi-thread, le mode d'accès (lecture/écriture) et la gestion mémoire

Propriétés : dans le .h

```
@interface Etudiant: NSObject

@property (nonatomic,readonly,copy) NSString * nom;
@property (nonatomic,readwrite,assign) int numero;

@end
```

Propriétés : dans le .m

```
@implementation Etudiant

@synthesize nom;
@synthesize numero;

...

@end
```

ou

```
@implementation Etudiant

@synthesize nom, numero;

...

@end
```

Propriétés : atomicité des accès

comportement en environnement multi-thread

2 qualificatifs possibles :

- `atomic` (par défaut) : performances dégradées
- `nonatomic`

Propriétés : accès à l'attribut

2 qualificatifs possibles :

- `readonly` : seule une méthode de lecture (getter) est créée
- `readwrite` (par défaut) : une méthode de lecture (getter) et une méthode d'écriture (setter) sont créées

Propriétés : gestion mémoire

3 qualificatifs possibles :

- `assign` (par défaut) : le setter réalise une affectation simple
- `retain` : le setter fait un `retain`
- `copy` : le setter crée un nouvel objet

Propriétés : gestion mémoire

assign

```
- (void)setBinome:(Etudiant*)unBinome { binome = unBinome; }
```

retain

```
- (void)setBinome:(Etudiant*)unBinome {
  if (binome != unBinome) {
    [binome release];
    binome = [unBinome retain];
  }
}
```

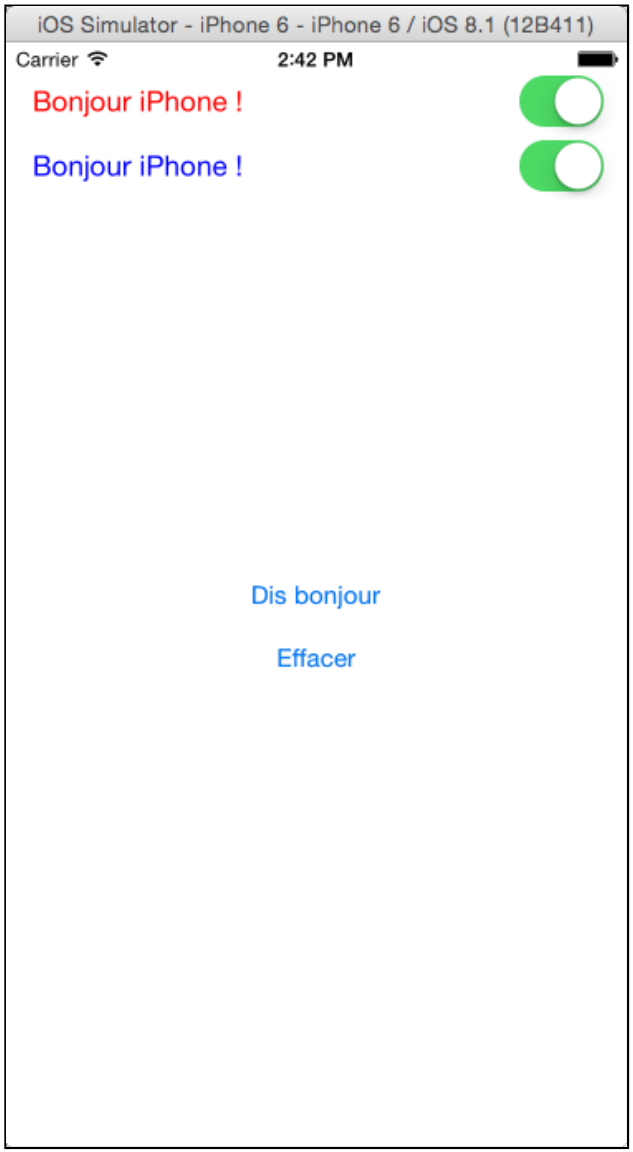
copy

```
- (void)setBinome:(Etudiant*)unBinome {
  if (binome != unBinome) {
    [binome release];
    binome = [unBinome copy];
  }
}
```

Automatic Reference Counting (ARC)

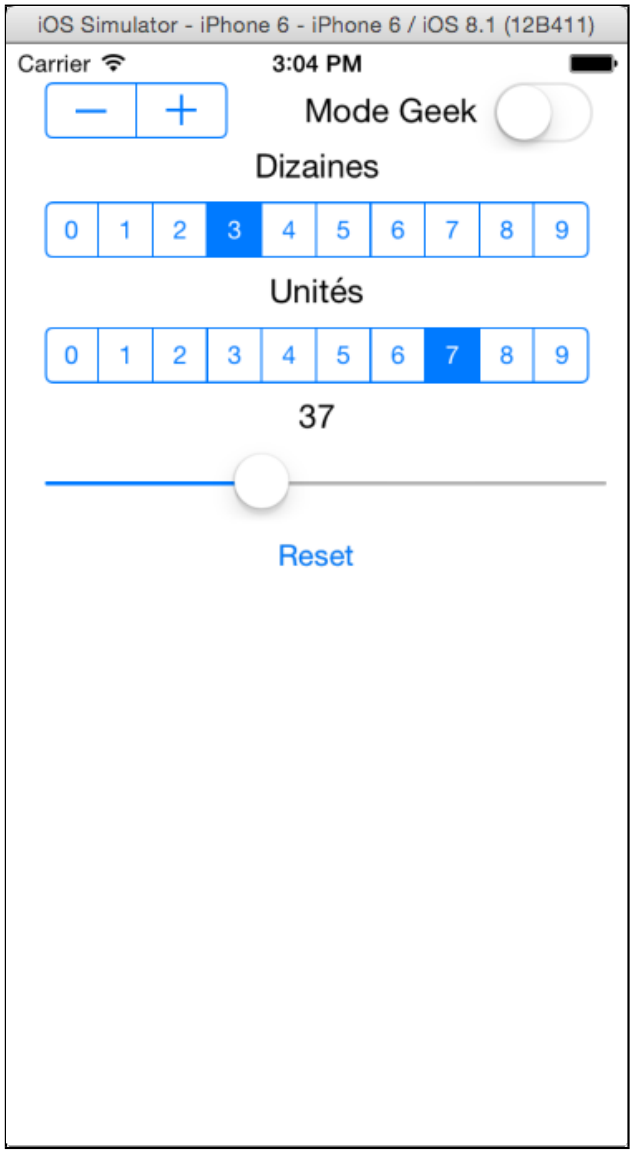
- mécanisme fourni par les dernières versions d'Xcode
- automatise la gestion des compteurs de références : ne plus utiliser `retain`, `release`, `autorelease` (le compilateur les ajoute automatiquement)
- nouveaux qualificatifs pour la gestion de la mémoire des propriétés de type objet : `strong` (par défaut) et `weak`

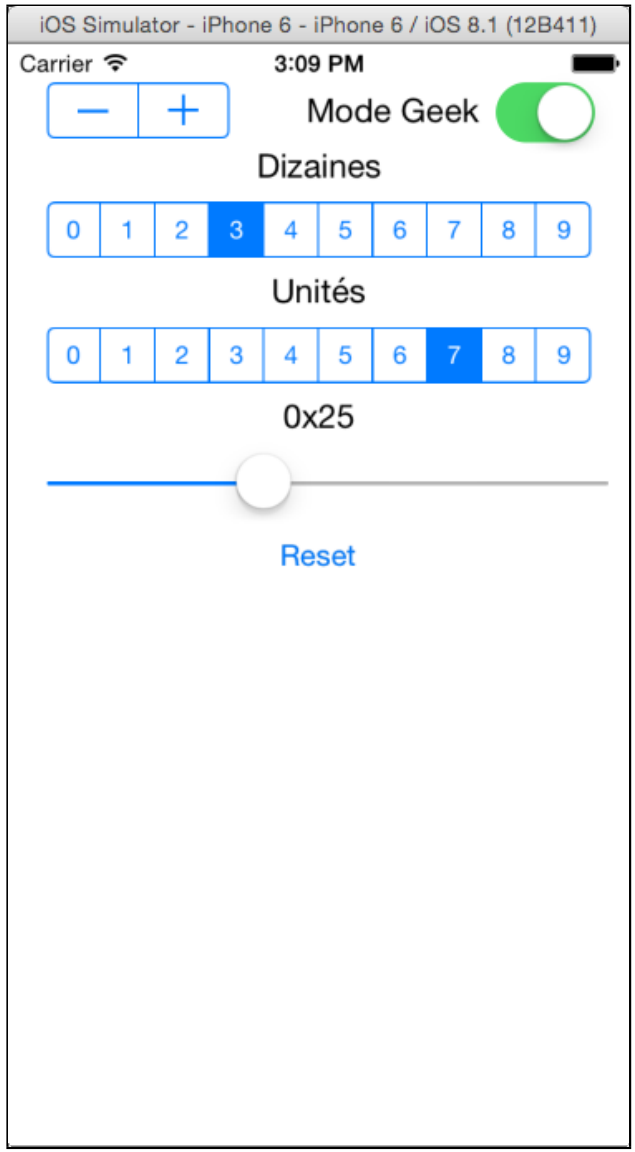
Exercice : *Hello World !*



la couleur du texte change à chaque affichage

Exercice : *Dizainier*





Plan

Fenêtre

toute application iOS a au moins une fenêtre (en général exactement une) :

- instance de `UIWindow`
- surface remplissant tout l'écran et accueillant des *vues*

Vue

mécanisme de base pour l'interaction avec l'utilisateur :

- instance de `UIView`
- rectangle où on peut dessiner, sensible aux événements
- contenue dans une fenêtre
- tous les composants graphiques (*widgets*) d'iOS sont des vues : boutons, étiquettes, zones de texte, ...

Organisation hiérarchique des vues

chaque vue :

- a une super-vue
- peut avoir aucune, une seule ou plusieurs sous-vues
- est propriétaire de ses sous-vues (elle fait un `retain` sur ses sous-vues)

Création et organisation des vues

2 possibilités :

- avec Storyboard
- dans le code :
  - `-(id)initWithFrame:(CGRect)aRect`
  - `-(void)addSubview:(UIView*)view`
  - `-(void)removeFromSuperview`
  - ...

Associer une vue à l'application

- compléter l'*application delegate* : `application:DidFinishLaunchingWithOptions`
- allocation/initialisation de la vue :

```
CGRect appWindow = [window bounds];
newView = [[MyView alloc] initWithFrame:appWindow];
```

- création de la hiérarchie :

```
[window addSubview:newView];
```

Position et taille des vues

2 propriétés de type `CGRect` :

- `frame` = rectangle de la vue, dans le système de coordonnées de la super-vue
- `bounds` = rectangle de la vue, dans le système de coordonnées de la vue elle-même

```
struct CGRect {          |   struct CGPoint {       |   struct CGSize {
  CGPoint origin;        |     CGFloat x;           |     CGFloat width;
  CGSize size;           |     CGFloat y;           |     CGFloat height;
};                       |   };                     |   };
```

Construire ses propres vues

créer une classe héritant de `UIView` :

- affichage :

  ```
  -(void)drawRect:(CGRect)rect
  ```

- capture des événements tactiles :

  ```
  -(void)touchesBegan:(NSSet*)touches withEvent:(UIEvent*)event
  -(void)touchesMoved:(NSSet*)touches withEvent:(UIEvent*)event
  -(void)touchesEnded:(NSSet*)touches withEvent:(UIEvent*)event
  ```

- forcer le rafraîchissement :

  ```
  -(void)setNeedsDisplay
  ```

Construire ses propres vues

- réagir à une secousse :
  - `-(BOOL)canBecomeFirstResponder { return YES; }`
  - dans `initWithFrame` :

    ```
    [self becomeFirstResponder];
    ```

  - toute secousse provoque l'exécution de la méthode :

    ```
    -(void)motionBegan:(UIEventSubtype)motion
             withEvent:(UIEvent*)event
    ```

Dessiner en 2D dans une vue

CoreGraphics (CG) :

- bibliothèque complète de dessin
- fonctions C, pas d'objets
- les fonctions utilisent un contexte graphique :

  `UIGraphicsGetCurrentContext()` dans `drawRect`
- formes, couleurs, polices, ...

Dessiner en 2D dans une vue

un rectangle

```
CGRect square = CGRectMake(50, 50, 250, 250);
CGContextAddRect(context, square);
CGContextStrokePath(context); // dessine le contour
```

un disque

```
CGContextAddArc(context, 200, 200, 70, 0, M_PI*2, YES);
CGContextFillPath(context); // peint le disque
```

un triangle

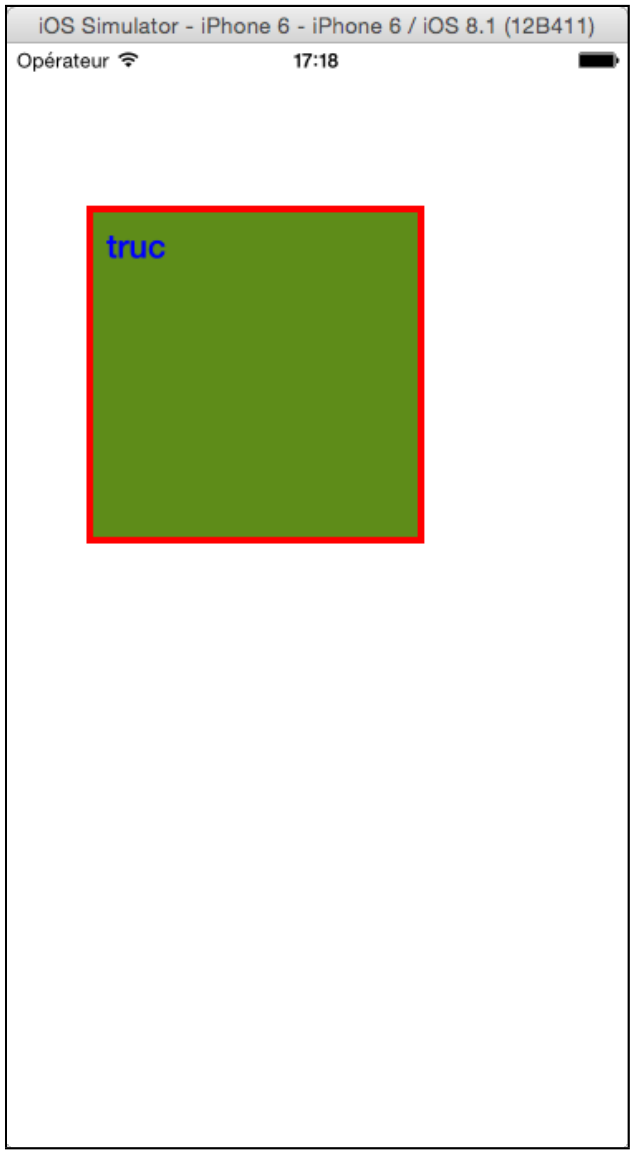
```
CGContextBeginPath(context);
CGContextMoveToPoint(context, 75, 10);
CGContextAddLineToPoint(context, 10, 150);
CGContextAddLineToPoint(context, 160, 150);
CGContextClosePath(context);
CGContextStrokePath(context);  // dessine le contour
```

Couleurs et polices

```
-(void)drawRect:(CGRect)rect {
  CGContextRef context = UIGraphicsGetCurrentContext();
  CGContextSetLineWidth(context, 4);

  [[UIColor redColor] setStroke];
  [[UIColor colorWithRed:0.3 green:0.5 blue:0 alpha:0.9] setFill];

  CGRect square = CGRectMake(50, 100, 200, 200);
  CGContextAddRect(context, square);

  CGContextDrawPath(context, kCGPathFillStroke);

  [[UIColor blueColor] setFill];
  square.origin.x += 10;  square.origin.y += 10;
  NSString *message = @"truc";
  UIFont *font = [UIFont boldSystemFontOfSize:20];
  [message drawInRect:square withFont:font];
}
```

Couleurs et polices

le code précédent donne :



Scroll et zoom : `UIScrollView` + `UIScrollViewDelegate`

```
@interface ViewController : UIViewController <UIScrollViewDelegate> {
    MyView *maVue; // la vue sur laquelle les scrolls et
                   // les zooms seront actifs
}
@property (weak, nonatomic) IBOutlet UIScrollView *maScrollView;
@end
```

dans Storyboard :

- indiquer que le délégué (`delegate`) de la *Scroll View* est le *View Controller*
- fixer les valeurs min et max du zoom

Scroll et zoom : UIScrollView + UIScrollViewDelegate

```
@implementation ViewController

- (void)viewDidLoad {
    [super viewDidLoad];
    CGRect square = CGRectMake(0, 0, 1500, 1500);
    maVue = [[MyView alloc] initWithFrame:square];
    [[self maScrollView] addSubview:maVue];
    [[self maScrollView] setContentSize:[maVue bounds].size];
    [[self maScrollView] setContentOffset:[maVue center]];
}

-(UIView*)viewForZoomingInScrollView:(UIScrollView *)scrollView {
     return maVue;
}
...
@end
```

Faire disparaître le clavier d'un TextField

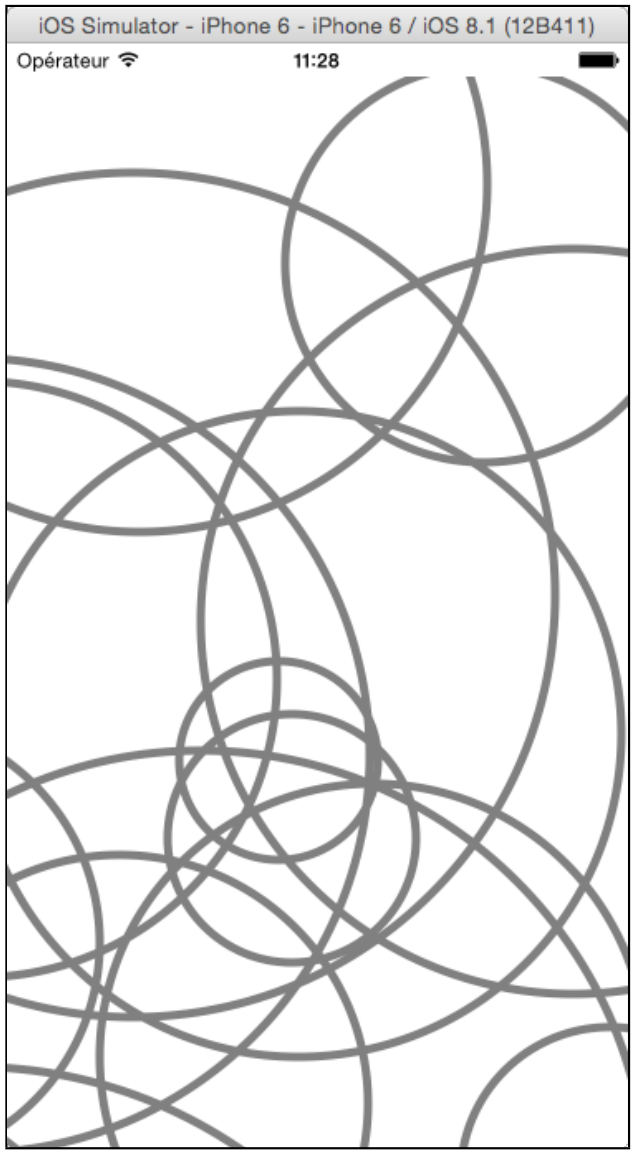
```
@interface ViewController : UIViewController <UITextFieldDelegate>
@property (weak, nonatomic) IBOutlet UITextField *monTextField;
...
@end
```

```
@implementation ViewController
...
- (BOOL)textFieldShouldReturn:(UITextField *)textField {
    if (textField == monTextField)
        [textField resignFirstResponder];
    return YES;
}
@end
```

dans Storyboard indiquer que le délégué du TextField est le ViewController

Exercice : *Cercles*

- créer une classe `CerclesView`
- compléter `drawRect` : dessiner 20 cercles gris, position au hasard, rayon croissant, épaisseur du trait : 5 pixels
- une secousse efface l'écran et dessine 20 nouveaux cercles
- scroll + zoom



Exercice : *Cercles* (variante)

Exercice : *Cercles* (variante)

- gérer le cas où l'utilisateur a laissé une zone de texte vide
- tenir compte des changements d'orientation de l'écran
- anglais par défaut + proposer une version française
  - voir le premier chapitre du cours de M1
- 2 zones de texte de même taille occupant la largeur de l'écran ?

Plan

Tables

- classe `UITableViewController`
  (sous-classe de `UIViewController`)

- présentation des données :
  - une seule colonne, plusieurs lignes (cellules)
  - découpage en sections
  - défilement vertical
  - gestion optimisée de la mémoire

Styles de présentation

UITableViewStylePlain

UITableViewStyleGrouped

Composants

- une en-tête et un pied-de-page
- des sections, chacune composée :
  - d'une en-tête et d'un pied-de-page
  - de cellules

Les données

- affichage à la demande
- seules les données affichées sont allouées
- les données proviennent d'une source (ex : un tableau, une base de données. . .)

Les méthodes à implémenter

- -tableView:cellForRowAtIndexPath:
- -numberOfSectionsInTableView:
- -tableView:numberOfRowsInSection:
- -tableView:titleForHeaderInSection:
- -tableView:titleForFooterInSection:
- ...

Exemple d'implémentation

```
-(NSInteger)numberOfSectionsInTableView:(UITableView*)tableView {
  return 2;
}

-(NSInteger)tableView:(UITableView*)tableView
                        numberOfRowsInSection:(NSInteger)section {
  if (section == 0) return 3;
  else return 1;
}

-(NSString*) tableView:(UITableView*)tableView
                      titleForHeaderInSection:(NSInteger)section {
  if (section == 0) return @"Professionnel";
  else return @"Personnel";
}
```

Affichage

attribut `tableView` (instance de `UITableView`) répond aux méthodes :

- `reloadData` (rafraîchit l'affichage)
- `insertRowsAtIndexPaths:withRowAnimation:`
- `deleteRowsAtIndexPaths:withRowAnimation:`
- `setTableHeaderView:` (en-tête de la table)
- `setTableFooterView:` (pied-de-page de la table)

Comment remplir une cellule

- implémenter `tableView:cellForRowAtIndexPath:`
- cette méthode renvoie la cellule pointée par l'*index path* fourni en argument
- *index path* = (numéro section, numéro ligne)

Exemple : afficher l'*index path* de chaque cellule

```
-(UITableViewCell*)tableView:(UITableView*)tableView
                       cellForRowAtIndexPath:(NSIndexPath*)indexPath {

  UITableViewCell *cell = ...;

  ...

  [[cell textLabel] setText:
    [NSString stringWithFormat:@"section: %d, ligne: %d",
      [indexPath section],
      [indexPath row]
    ]
  ];

  return cell;
}
```

Réutilisation des cellules

```
-(UITableViewCell*)tableView:(UITableView*)tableView
                       cellForRowAtIndexPath:(NSIndexPath*)indexPath {

  UITableViewCell *cell =
     [tableView dequeueReusableCellWithIdentifier:@"Cell"
                forIndexPath:indexPath];

  ...

  return cell;
}
```

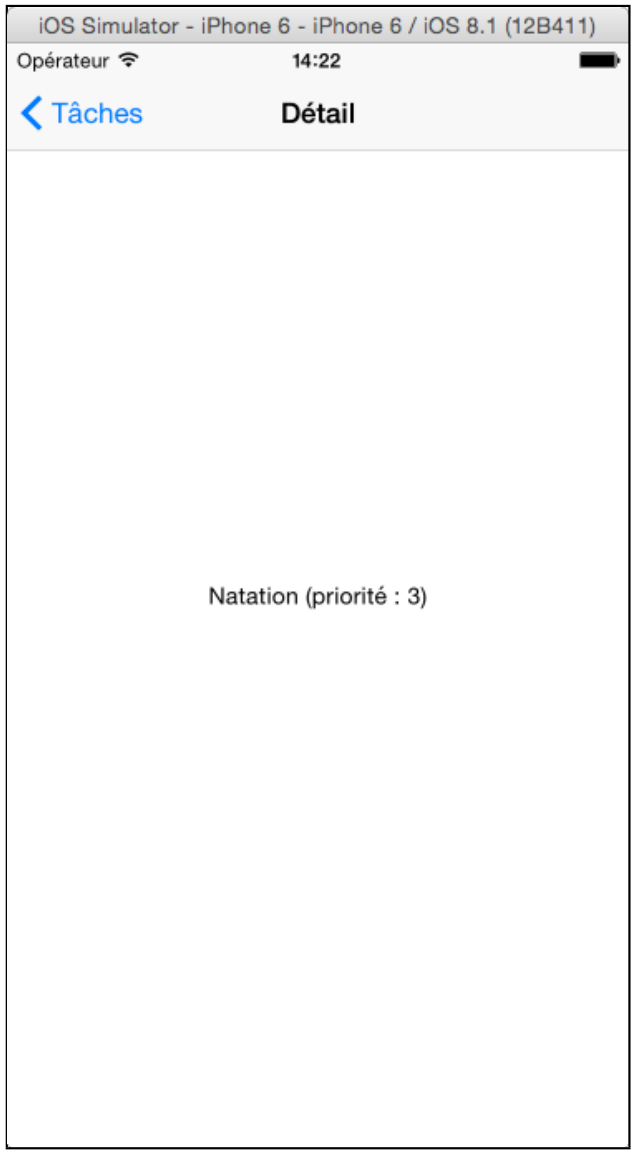
Ajout de données

```
- (void)insertNewObject:(id)sender {
  // Ajout dans la source de données :
  [self.objects insertObject:
     [[Tache alloc] initWithPriorite:0 andTitre:@"A faire"] atIndex:0];

  // Ajout dans la vue :
  NSIndexPath *indexPath = [NSIndexPath indexPathForRow:0 inSection:0];
  [self.tableView insertRowsAtIndexPaths:@[indexPath]
     withRowAnimation:UITableViewRowAnimationAutomatic];
}
```

Suppression de données

```
-(void)tableView:(UITableView*)tableView
     commitEditingStyle:(UITableViewCellEditingStyle)editingStyle
      forRowAtIndexPath:(NSIndexPath*)indexPath {

  if (editingStyle == UITableViewCellEditingStyleDelete) {
    // Suppression dans la source de données :
    [self.objects removeObjectAtIndex:indexPath.row];
    // Suppression dans la vue :
    [tableView deleteRowsAtIndexPaths:@[indexPath]
       withRowAnimation:UITableViewRowAnimationFade];
  }

  else if (editingStyle == UITableViewCellEditingStyleInsert) {
    ...
  }

}
```

Exercice : *Tâches*



Exercice : *Tâches*

- choisir le template *Master-Detail Application* lors de la création du projet
- créer une classe `Tache` avec les propriétés :
  - `NSString* titre`
  - `int priorite` ($\in [0, 4]$)
- créer un tableau de tâches initiales (voir transparent précédent)
- le bouton + ajoute une tâche de titre *À faire* et de priorité 0
- le bouton *Edit* permet de supprimer des tâches

Déplacer des données

- compléter la méthode

```
-(void)tableView:(UITableView*)tableView
      moveRowAtIndexPath:(NSIndexPath*)fromIndexPath
      toIndexPath:(NSIndexPath*)toIndexPath
```

- algorithme à implémenter :
  - trouver l'élément à déplacer
  - le supprimer de la source de données
  - l'ajouter au bon endroit dans la source de données
- l'affichage est actualisé automatiquement

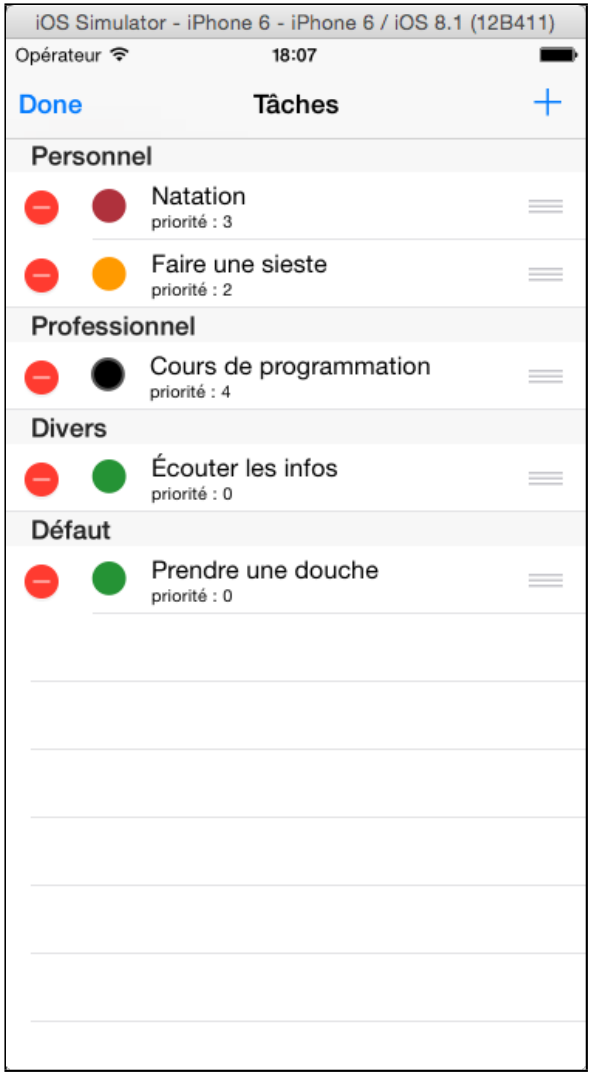
Déplacer des données

autoriser le déplacement :

```
- (BOOL)tableView:(UITableView *)tableView
         canMoveRowAtIndexPath:(NSIndexPath *)indexPath {
  return YES;
}
```

Exercice : *Tâches*

créer 4 sections



autoriser le déplacement

Plan

Contrôleur de vues

- instance de `UIViewController`
- correspond à un écran de l'application
- associé à une hiérarchie de vues
- gère la création, la présentation, la destruction de ses vues
- gère les interactions entre ses vues et d'autres objets de l'application
- ex : `UITableViewController`, `UISplitViewController`, ...

Interactions entre contrôleurs de vues

- la plupart des applications iOS sont constituées de plusieurs contrôleurs de vues entre lesquels l'utilisateur peut naviguer
- Storyboard permet de créer des segues pour modéliser ces interactions : avec un clic droit, tirer une flèche entre les contrôleurs de vues qui interagissent
- Storyboard permet d'attribuer un identifiant à un segue

Interactions entre contrôleurs de vues

dans le contrôleur de vues parent, la méthode suivante est appelée juste avant l'activation d'un segue :

```
- (void)prepareForSegue:(UIStoryboardSegue *)segue sender:(id)s {
    if ([[segue identifier] isEqualToString:@"..."]) {
        ...
    }
}
```

Affichage d'un contrôleur de vues

- présentation modale
- *Navigation Controller*
- *TabBar Controller*

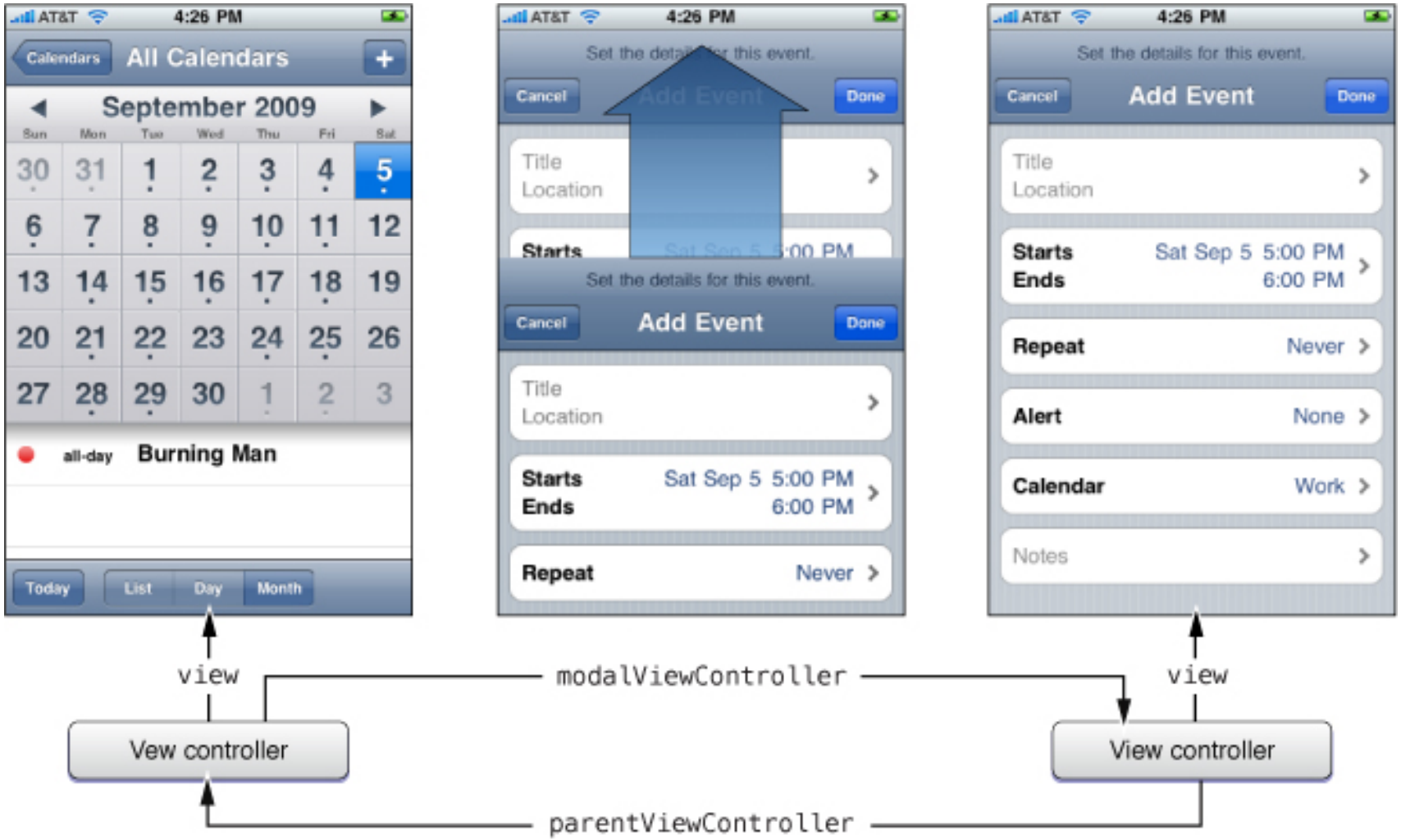
Présentation modale

utilisation : interruption temporaire du programme pour

- saisie immédiate d'informations par l'utilisateur
- affichage temporaire d'informations
- ...

affichage à partir d'un autre contrôleur de vues (le parent)

Présentation modale : exemple



Création d'une présentation modale

deux possibilités :

- dans Storyboard : créer un segue et choisir l'option *present modally*
- dans le code, dans le contrôleur de vues parent :

```
UIViewController *controller = ...;
[self presentViewController:controller
                   animated:YES completion:NULL];
```

Terminer une présentation modale : solution 1

par ex. dans un contrôleur de vues présenté modalement pour éditer une tâche et dans lequel on trouve un bouton *Sauver* :

```
-(IBAction)sauver:(id)sender {
  ...
  [monParent editionTerminee:tache]; // monParent = controleur de vue
}
```

dans le contrôleur de vues parent (par ex. un *UITableViewController*) :

```
- (void)editionTerminee:(Tache*)tache {
  ...
  [self.tableView reloadData];
  [self dismissViewControllerAnimated:YES completion:NULL];
}
```

Terminer une présentation modale : solution 2

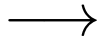
ou alors :

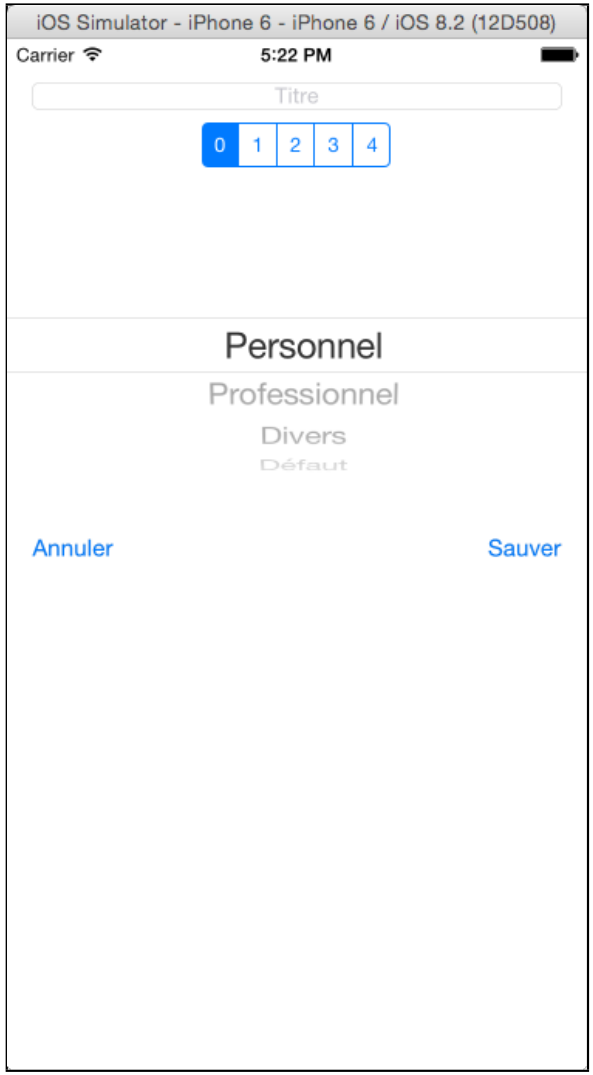
```
-(IBAction)sauver:(id)sender {
  ...
  [[self presentingViewController]
                dismissViewControllerAnimated:YES completion:NULL];
  [monParent editionTerminee:tache]; // monParent = controleur de vue
}
```

et :

```
- (void)editionTerminee:(Tache*)tache {
  ...
  [self.tableView reloadData];
}
```

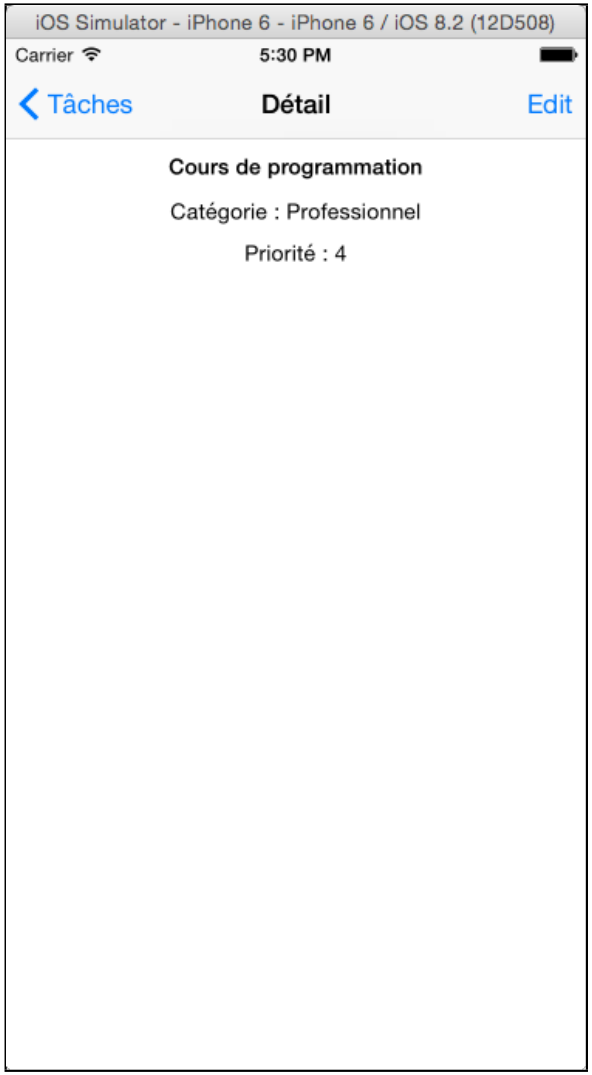
Exercice : *Tâches* avec présentations modales



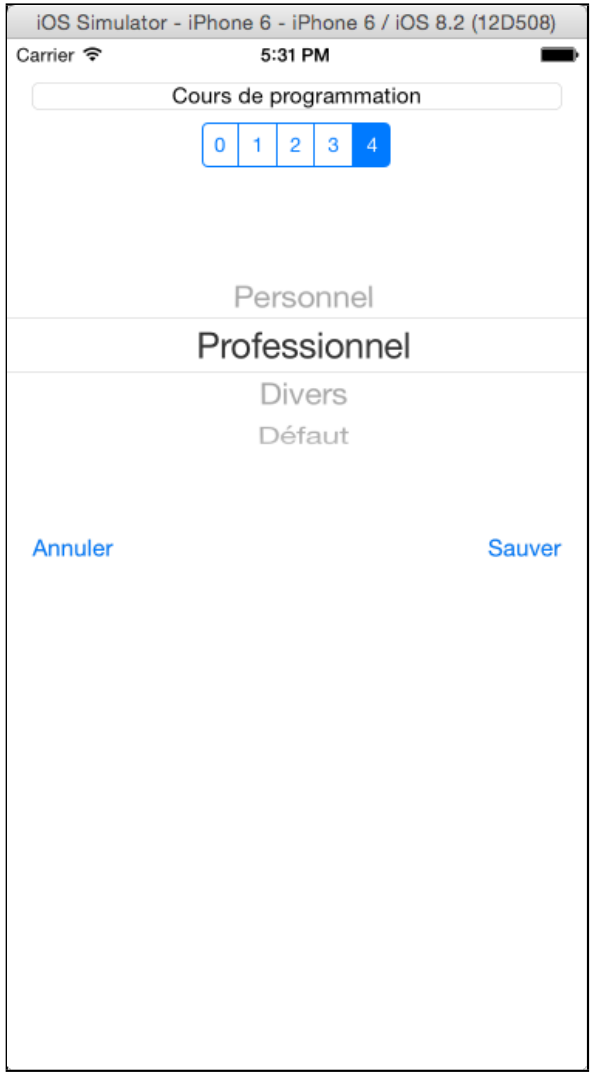


appui sur bouton +

Exercice : *Tâches* avec présentations modales



sélection d'une tâche

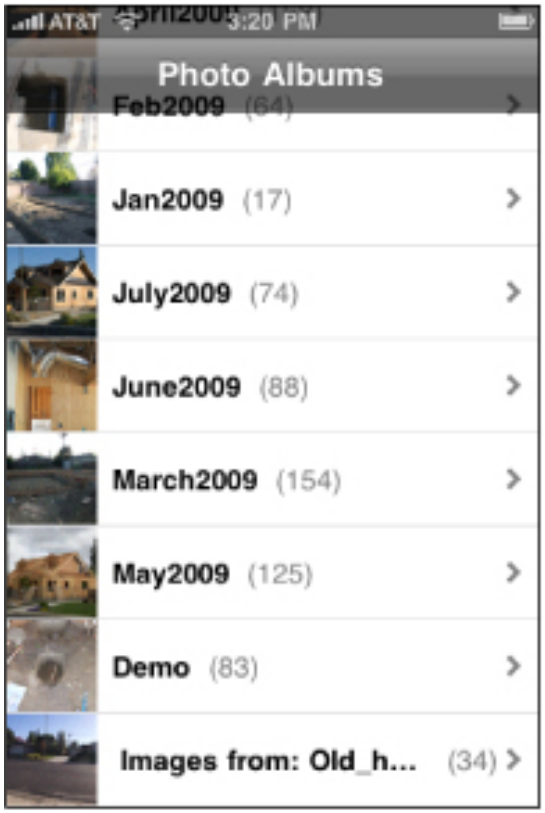


appui sur bouton Edit

Navigation Controller

- instance de `UINavigationController`
- gère une pile de contrôleurs de vues
- sommet de la pile = contrôleur de vues actif (visible)

Navigation Controller : exemple



Album list controller

Photo album controller

Photo controller

Navigation Controller : barre de navigation

- en haut de l'écran
- peut contenir :
  - le titre du contrôleur de vue courant
  - des boutons (par exemple un bouton pour revenir à l'écran précédent)

Navigation Controller : installation

- dans Storyboard ou
- dans le code :

```
UITableViewController *tableController =
  [[UITableViewController alloc]
    initWithStyle:UITableViewStylePlain];

UINavigationController *navigationController =
  [[UINavigationController alloc]
    initWithRootViewController:tableController];
```

Navigation Controller : push et pop

- ajouter un contrôleur de vues au sommet de la pile :

```
UIViewController *viewController = ...;
[navigationController
  pushViewController:viewController animated:YES];
```

- retirer un contrôleur de vues du sommet de la pile :

  iOS s'en charge (ajoute automatiquement à la barre de navigation un bouton de retour vers le contrôleur précédent)

Navigation Controller : gérer la barre de navigation

tous les détails sont dans l'objet `navigationItem` :

```
UINavigationItem *nav = [self navigationItem];

[nav setTitle:@"Truc"];

UIBarButtonItem *trucButton =
  [[UIBarButtonItem alloc]
    initWithTitle:@"truc"
            style:UIBarButtonItemStyleBordered
           target:self
           action:@selector(truc:)
  ];

[nav setLeftBarButtonItem:trucButton];
```

Navigation Controller : boutons spéciaux

- bouton *Edit* (basculer en mode édition) :

```
[[self navigationItem]
  setLeftBarButtonItem:[self editButtonItem]];
```

- bouton + :

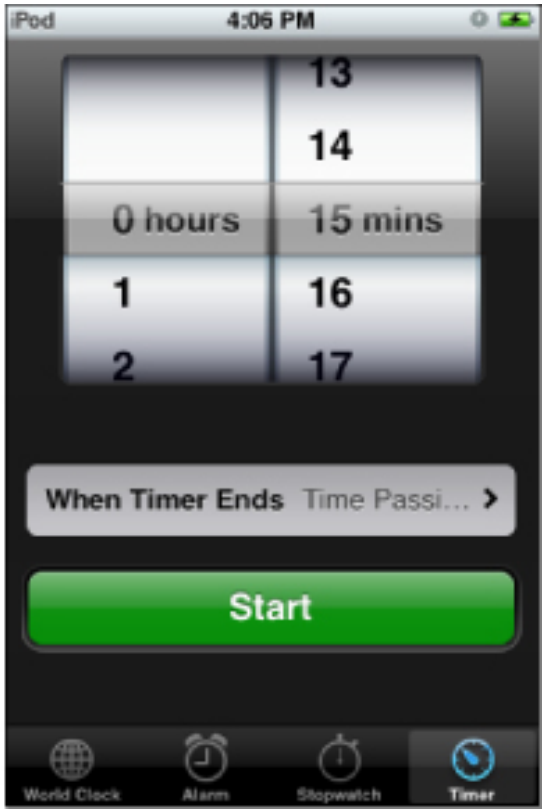
```
UIBarButtonItem *addButton =
  [[UIBarButtonItem alloc]
    initWithBarButtonSystemItem:UIBarButtonSystemItemAdd
                         target:self
                         action:@selector(addTruc)];
```

- ...

TabBar Controller

- instance de `UITabBarController`
- gère un tableau de contrôleurs de vues
- une *tab bar* en bas de l'écran contient des items, chacun associé à un contrôleur de vues du tableau
- chaque item permet de sélectionner le contrôleur de vues associé et de l'afficher

TabBar Controller : exemple



TabBar Controller : installation

- dans Storyboard ou
- dans le code :
  - création d'une instance de `UITabBarController`
  - création d'un tableau de `UIViewController`

```
NSArray *viewControllers = [NSArray arrayWithObjects:...];
```

  - ajout du tableau dans le *TabBarController*

```
[tabBarController setViewControllers:viewControllers];
```

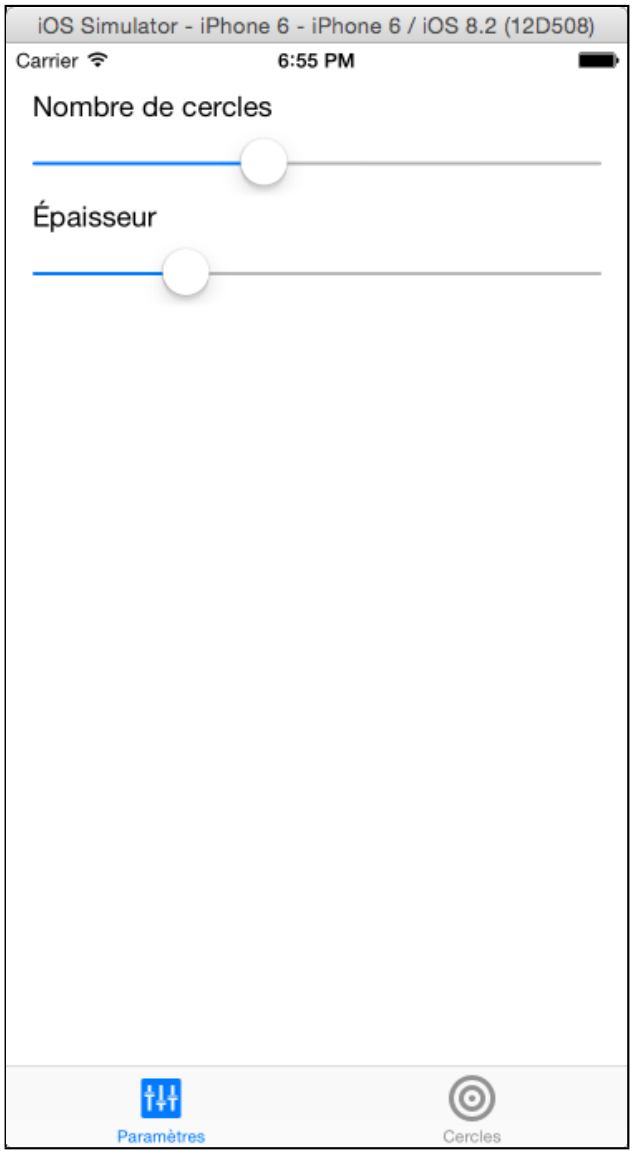
  - création de la filiation entre la fenêtre de l'application et le *TabBar Controller*

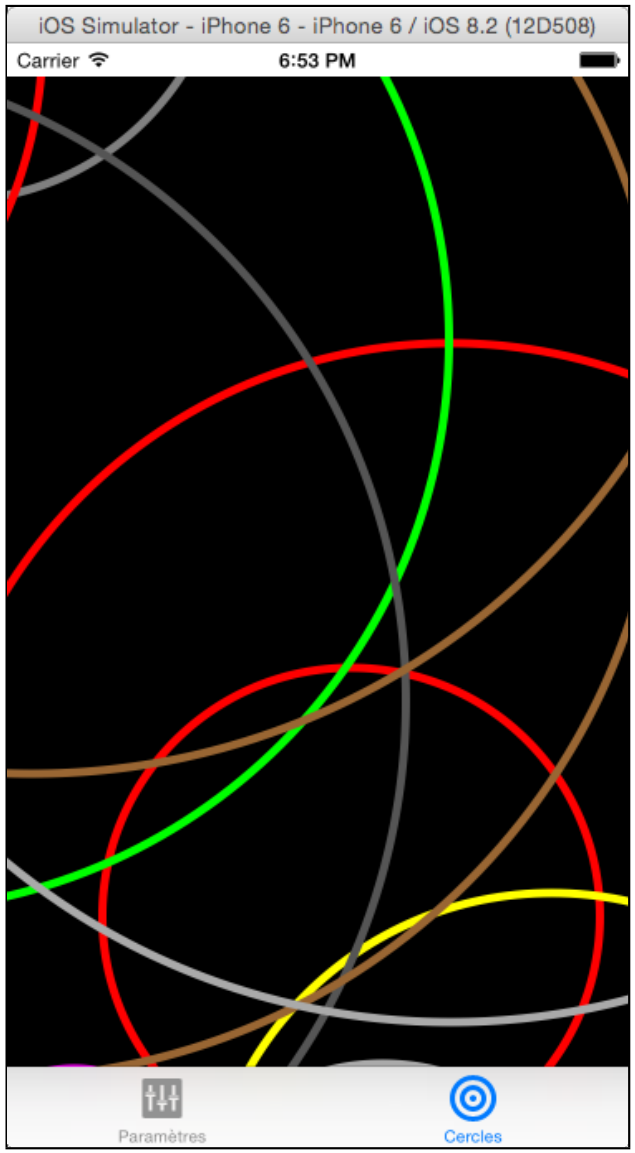
TabBar Controller : gérer la barre d'items

tous les détails sont dans l'objet `tabBarItem` :

```
UITabBarItem *bar = [self tabBarItem];
[bar setTitle:@"Parametres"];
[bar setImage:[UIImage imageNamed:@"sliders-small.png"]];
```

Exercice : *Cercles*





Plan

abstraction d'un modèle relationnel SQLite en un modèle objet :

- une table $A \leftrightarrow$ une classe A
- une ligne dans la table $A \leftrightarrow$ une instance de la classe A
- une colonne de la table $A \leftrightarrow$ une variable d'instance de la classe A

voir la documentation Apple

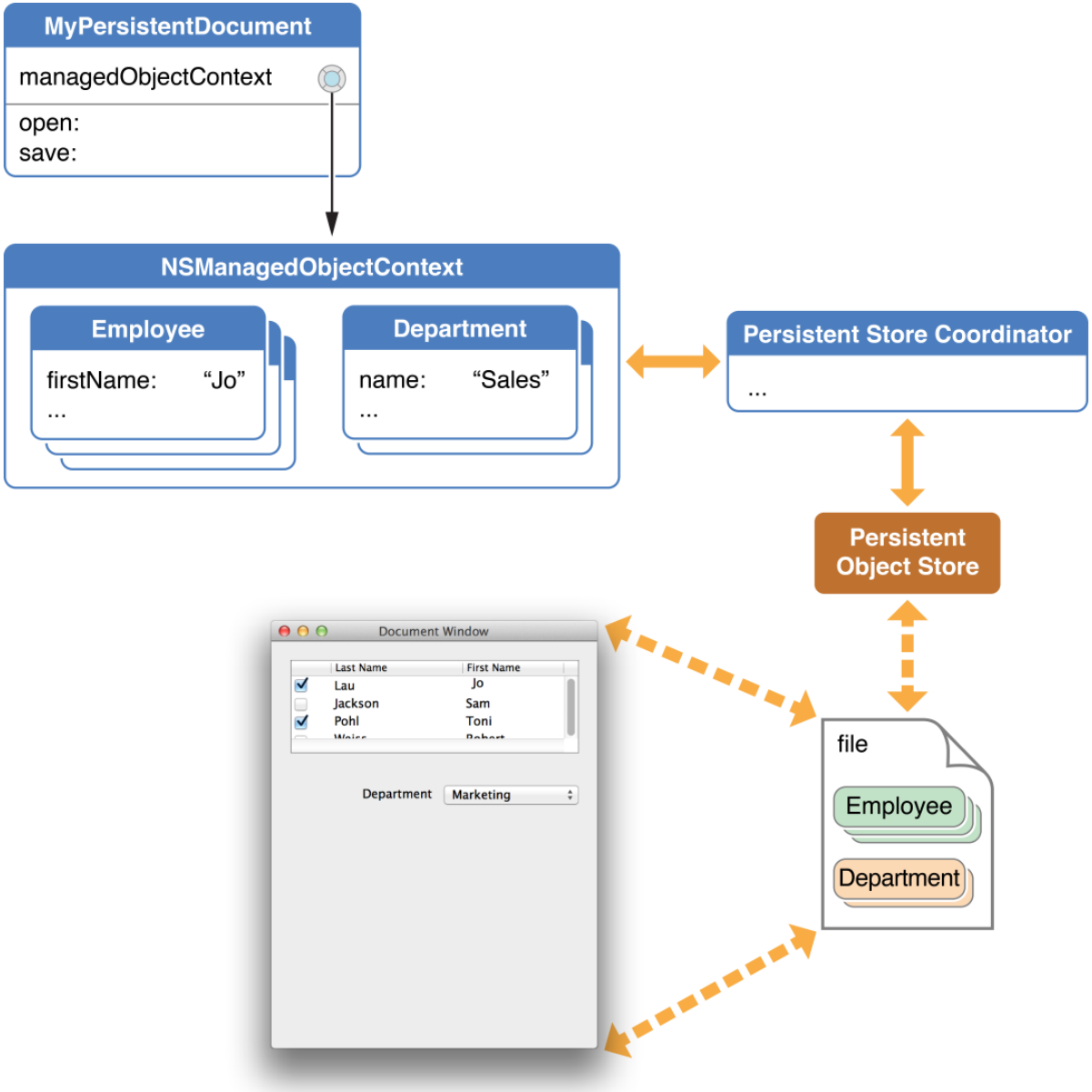
transforme les lignes des tables en objets :

- création d'un objet $\Rightarrow$ insertion d'une ligne dans la table correspondante
- modification d'un objet $\Rightarrow$ mise à jour de la ligne correspondante
- destruction d'un objet $\Rightarrow$ suppression de la ligne correspondante

- une table/classe est appelée entité
- une colonne/variable d'instance est appelée attribut
- un modèle (fichier `.xcdatamodeld`) décrit les entités, leurs attributs et les relations entre les entités

un objet correspondant à une ligne d'une table

- est une instance de la classe `NSManagedObject`
- vit dans un *contexte* : la classe `NSManagedObjectContext` se charge de l'adéquation entre la base de données SQLite et les objets (elle gère les créations, modifications, suppressions de données)
- est récupéré grâce à une *requête* (classe `NSFetchRequest`) sur un contexte


MyPersistentDocument
managedObjectContext
open:
save:
NSManagedObjectContext
Employee
firstName: "Jo"
...
Department
name: "Sales"
...
Persistent Store Coordinator
...
Persistent Object Store
Document Window
Last Name
First Name
Lau
Jo
Jackson
Sam
Pohl
Toni
Department
Marketing
file
Employee
Department

Création d'une application

sous XCode, pour créer une application utilisant *Core Data* avec affichage des données sous forme de listes :

- File → New → New Project...
- Master-Detail Application
- cocher *Use Core Data*

l'essentiel du code est créé automatiquement !

Création d'une application

création des entités et des relations sous XCode :

- cliquer sur le fichier dont le nom se termine par `.xcdatamodeld`
- deux modes de visualisation (boutons `Editor Style` en bas à droite)

Exercice : *Tâches*

rendez les tâches persistantes