Programmation iOS
L3 informatique

Étienne Payet

Département de mathématiques et d'informatique

Plan

Ressources

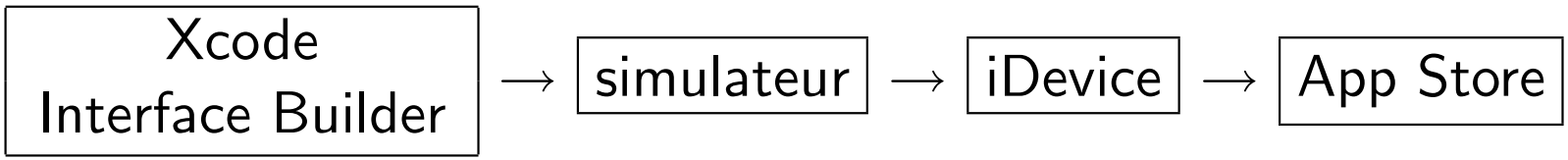
- `http://developer.apple.com`
- Standford CS193P (niveau L2-L3)
  `http://www.stanford.edu/class/cs193p/cgi-bin/drupal/`
  Vidéos sur iTunes U
- livres : il y en a beaucoup...
  - *The Big Nerd Ranch Guide*, Conway & Hillegass
  - voir à la BU
  - iOS 4 & iOS 5
  - Objective C 2.0

Pour pouvoir travailler

- développement :
  - Mac avec processeur Intel
  - Mac OS X 10.6.6 (Snow Leopard)
  - Xcode 3.2 ou Xcode 4 avec iOS SDK 4.3

- déploiement :
  - Apple ID
  - certificat développeur
  - iPod Touch, iPhone, iPad

Workflow d'Apple



Developer programs proposés par Apple

	/	université	entreprise	standard
SDK	✓	✓	✓	✓
versions bêta			✓	✓
forums	✓	✓	✓	✓
simulateur	✓	✓	✓	✓
déploiement		✓	✓	✓
distribution			✓	
App Store				✓
coût	0 $	0 $	299 $	99 $

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, ...

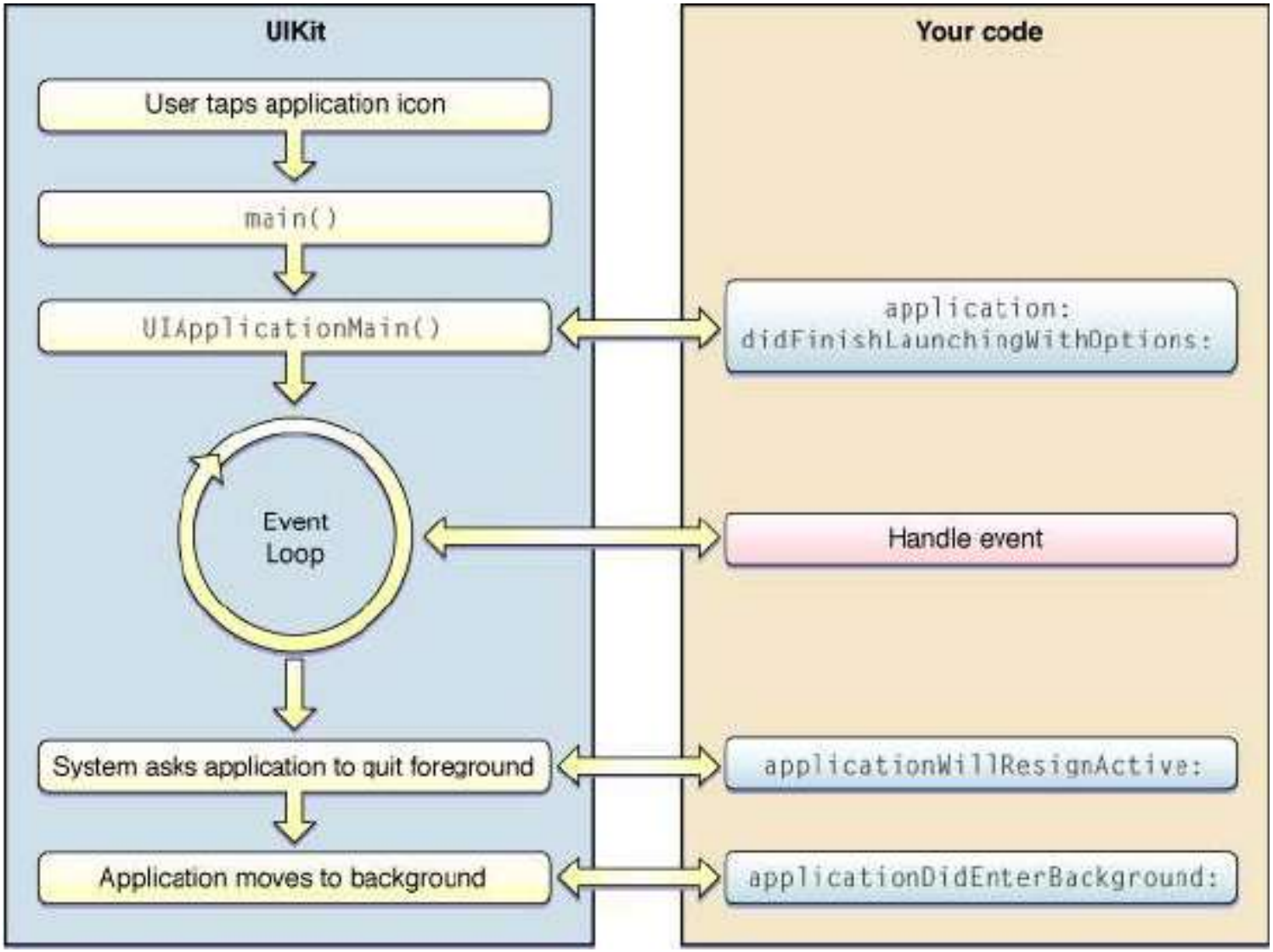
Les frameworks d'iOS

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 !*

Plan

Cycle de vie d'une application



La fonction `main` : transfère le contrôle à UIKit

```
#import <UIKit/UIKit.h>
int main(int argc, char *argv[]) {
    NSAutoreleasePool *pool = [[NSAutoreleasePool alloc] init];
    int retVal = UIApplicationMain(argc, argv, nil, nil);
    [pool release];
    return retVal;
}
```

La fonction `UIApplicationMain`

- initialise l'application
- crée l'*application delegate*
- charge le fichier `nib` principal (`MainWindow.xib`)

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

- créé par le team leader
- 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

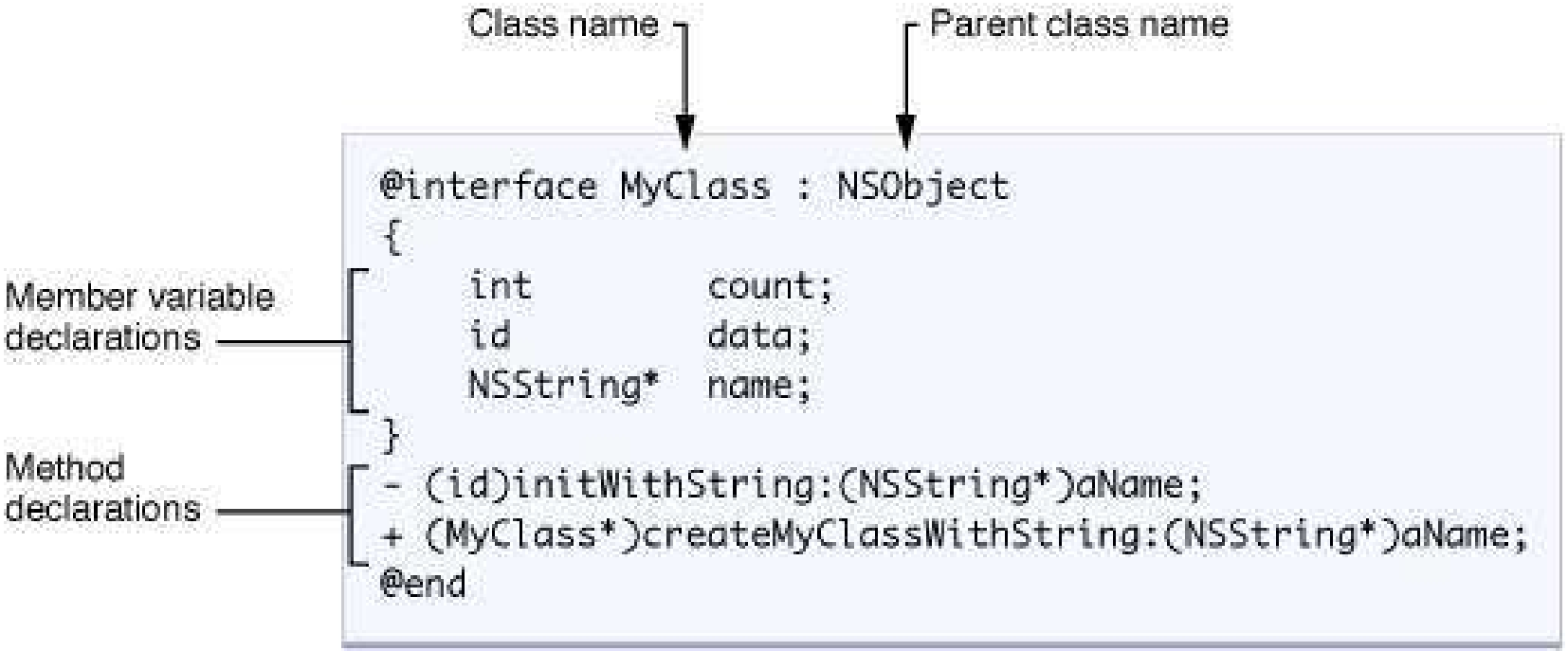
Sur-ensemble strict du C ANSI

programmation orientée objet :

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

Classes : déclaration

fichier d'en-tête (`.h`)



Classes : implémentation

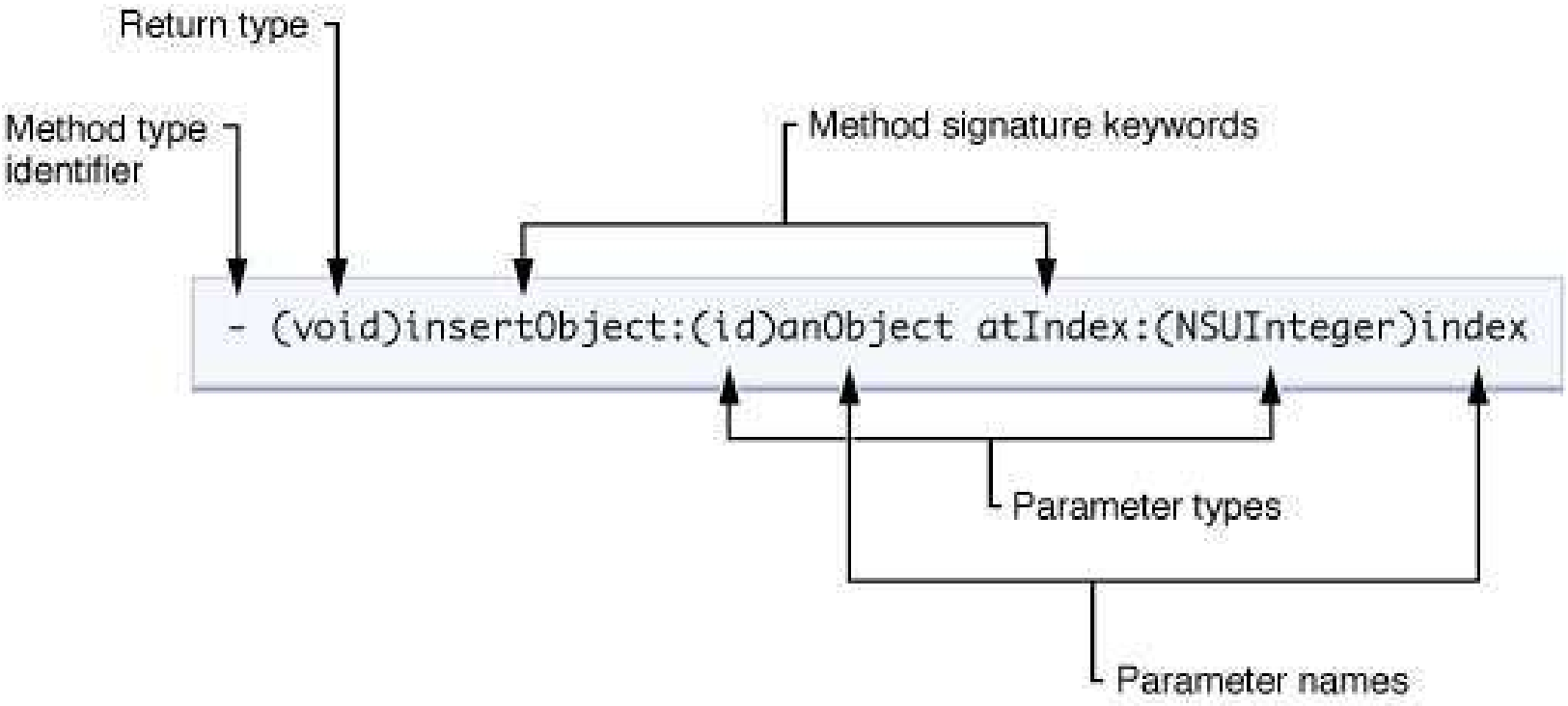
fichier source (`.m`)

```
#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 :

```
Couleur *uneCouleur;
int combien = [couleurs count];
for (int i = 0; i < combien; i++) {
   uneCouleur = [couleurs objectAtIndex:i];
   NSLog(@"couleur %@", [uneCouleur description]);
}
```

efficace :

```
for (Couleur *uneCouleur in couleurs)
   NSLog(@"couleur %@", [uneCouleur description]);
```

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)
- spécifient les accès, la gestion mémoire et le comportement en environnement multi-threads

Propriétés : dans le .h

```
@interface Etudiant: NSObject {
  NSString * nom;
  int numero;
}

@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 : accès à l'attribut

2 possibilités :

- `readonly` : le système crée une méthode de lecture (getter)
- `readwrite` :
  - par défaut
  - le système crée une méthode de lecture (getter)
  - le système crée une méthode d'écriture (setter)

Propriétés : gestion mémoire

3 possibilités pour le setter :

- `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];
  }
}
```

Propriétés : atomicité des accès

2 possibilités :

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

Exercice : *Quizz*

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

Associer une vue à l'application

- compléter l'*application delegate* : `applicationDidFinishLaunching`
- 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];
}
```

Scroll et zoom

créer une sous-classe de `UIScrollView` :

```
@interface MyScrollView : UIScrollView <UIScrollViewDelegate> {
    MyView *view; // la vue sur laquelle les scrolls et
                  // les zooms seront actifs
}
```

Scroll et zoom

```
-(id)initWithFrame:(CGRect)frame {
    if ((self = [super initWithFrame:frame])) {
        CGRect square = CGRectMake(0, 0, 1000, 2000);
        view = [[MyView alloc] initWithFrame:square];
        [self addSubview:view];
        [self setContentSize:[view bounds].size];
        [self setDelegate:self];
        [self setMaximumZoomScale:20];
        [self setMinimumZoomScale:0.2];
    }
    return self;
}

-(UIView*)viewForZoomingInScrollView:(UIScrollView *)scrollView {
     return view;
}
```

Exercice : *Cercles*

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

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`

Installation

```
-(void)applicationDidFinishLaunching:(UIApplication*)application {

  UITableViewController *myTable =
      [[UITableViewController alloc]
          initWithStyle:UITableViewStyleGrouped];

  [window addSubview:[myTable view]];

  ...
}
```

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

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)

Afficher l'*index path* dans 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 {

  static NSString *CellIdentifier = @"Cell";

  UITableViewCell *cell =
    [tableView dequeueReusableCellWithIdentifier:CellIdentifier];

  if (cell == nil) {
    cell = [[[UITableViewCell alloc]
                initWithStyle:UITableViewCellStyleSubtitle
              reuseIdentifier:CellIdentifier] autorelease];
  }

  ...
}
```

Exercice : *Tâches*

- créer une classe `Tache` :
  - `NSString* titre`
  - `int priorite` ($\in [0, 4]$)
- créer une sous-classe de `UITableViewController`:
  - les tâches d'une catégorie sont rangées dans un `NSArray`
  - un dictionnaire associe chaque catégorie à son `NSArray`
- un *tap* sur une cellule incrémente la priorité de la tâche associée

Ajout de données

exemple, une méthode appelée si appui sur un bouton :

```
-(void)addNewTask {
  NSMutableArray *tasksArray = ...;
  Task *t = [[Task alloc] initWithTitle:@"Nouvelle tache" priority:0];
  [tasksArray insertObject:t atIndex:0];
  [t release];
  NSIndexPath *indexPath = [NSIndexPath indexPathForRow:0 inSection:0];
  [tableView
    insertRowsAtIndexPaths:[NSArray arrayWithObject:indexPath]
          withRowAnimation:UITableViewRowAnimationAutomatic];
}
```

Basculer en mode *édition*


```
[[self tableView] setEditing:YES]
```

Supprimer des données en mode *édition*

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

  if (editingStyle == UITableViewCellEditingStyleDelete) {
    NSMutableArray *tasksArray = ...;
    [tasksArray removeObjectAtIndex:indexPath.row];
    [tableView
      deleteRowsAtIndexPaths:[NSArray arrayWithObject:indexPath]
            withRowAnimation:YES];
  }

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

}
```

Réordonner des données en mode *édition*

- compléter la méthode

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

- algorithme à implémenter :
  - trouver l'élément à déplacer et le retenir (`retain`)
  - le supprimer de la source de données
  - l'ajouter au bon endroit dans la source de données
  - faire un `release`
- l'affichage est actualisé automatiquement

Exercice : *Tâches*

dans l'en-tête de la table :

- bouton add pour ajouter une tâche (titre et priorité fixés)
- bouton edit pour basculer en mode *édition* et effacer ou déplacer des cellules

Plan

Contrôleur de vues

- instance de `UIViewController`
- 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`, ...

Types courants d'organisation

- *Modal View Controller*
- *Navigation Controller*
- *TabBar Controller*

Modal View Controller

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)

Modal View Controller : exemple

Afficher un *Modal View Controller*

dans le contrôleur de vues parent :

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

  EditTaskViewController *editViewcontroller = ...;
  Task *t = [[self tasksForSection:indexPath.section]
                              objectAtIndex:indexPath.row];
  editViewcontroller.editedTask = t;
  editViewcontroller.delegate = self;

  [self presentModalViewController:editViewcontroller animated:YES];
  [editViewcontroller release];
}
```

Effacer un *Modal View Controller*

dans le *Modal View Controller* :

```
-(IBAction)save:(id)sender {
  editedTask.title = titleTextField.text;
  editedTask.priority = prioritySegmentedControl.selectedSegmentIndex;
  [delegate editTask:editedTask];
}
```

dans le contrôleur de vues parent :

```
- (void)editTask:(Task*)task {
  [self.tableView reloadData];
  [self dismissModalViewControllerAnimated:YES];
}
```

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 supplémentaires

Navigation Controller : installation

```
-(void)aplicationDidFinishLaunching:(UIApplication*)application {
  ...
  UITableViewcontroller *tableController =
    [[UITableViewController alloc]
      initWithStyle:UITableViewStylePlain];

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

  [tableController release];
  [window addSubView:[navigationController view]];

  ...
}
```

Navigation Controller : push et pop

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

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

- 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 `UINavigationItem` :

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

[nav setTitle:@"Truc"];

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

[nav setLeftBarButtonItem:trucButton];
[trucButton release];
```

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)];
```

- ...

Exercice : *Tâches*

- le bouton + ajoute une tâche avec un titre et une priorité fixés
- un tap sur une cellule ouvre un *Modal View Controller* permettant de modifier la tâche associée

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 l'*application delegate* :
  - 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*

- dans chaque contrôleur de vues :
  - spécifier les propriétés (titre, image) : `[self tabBarItem]`

Exercice : *Cercles*

Plan

Core Data

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

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

Core Data

- une table/classe est appelée entité
- une colonne/variable d'instance est appelée attribut

- un objet correspondant à une ligne d'une table est une instance de la classe `NSManagedObject`

- 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)

Création d'une application

sous XCode 4, 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 4 :

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

Exercices

- tutoriel *Locations* sur `http://developer.apple.com`
- application *Tâches* : rendez les tâches persistantes