Développement d'application Nomade (iOS & Xamarin)

COURS 02 – OBJECTIVE-C & COCOA

M2 INIS – 2015

julien.mialon@univ-orleans.fr
martinsfonseca.jose@yahoo.fr

Plan du cours

- Introduction

iOS

- Introduction à Objective-C
- Structure d'une application
- Les vues
- La navigation
- Les tables
- Persistance de données

Xamarin

- Développement cross-platform
- Introduction à C#
- NuGet & Xamarin Store
- Le pattern MVVM

Objective-C & Cocoa

Objective-C

- Extension du C avec des concepts venant de Smalltalk
- Langage orienté objet
- Au départ limité à iOS et Mac OS X

- Maintenant disponible sur Windows, Linux, Android, …

Cocoa Touch

- Un ensemble de classe Objective-C
- Cocoa = Foundation + UIKit

- NeXTStep > OpenStep > Cocoa Mac > Cocoa Touch

- GNUstep & Cocotron => Projet OpenSource implémentant les API Cocoa Touch pour d’autres plateformes

Autres API utiles

- CoreData (base de données locale)
- Quartz (Dessin 2D)
- CoreAudio (traitement audio)
- …

Let’s Code

Définir une classe

- Deux fichiers :
  - Header file (fichier .h) : @interface … @end
  - Method file (fichier .m) : @implementation … @end

- Convention :
  - Le nom des classes est préfixé par les initiales du produit ou auteur.
  - => toutes les classes de « base » sont préfixées par NS (NextStep)

Header file (.h)

```
@interface Rectangle: NSObject
{
    int height;
    int width;
}

// Setters
-(void) setHeight: (int)newHeight;
-(void) setWidth: (int)newWidth;
-(void) setHeight:(int)newHeight width:(int)newWidth;

// Getters
-(int) height;
-(int) width;

@end
```

NSObject : objet racine de presque tout

Attention : ce n'est pas automatique comme en Java

Method file (.m)

```objc
#import "Rectangle.h"

@implementation Rectangle
-(void) setHeight: (int)newHeight { height = newHeight; }
-(void) setWidth: (int)newWidth { width = newWidth; }
-(void) setHeight:(int)newHeight width:(int)newWidth
{
    height = newHeight; width = newWidth;
}
-(int) height { return height; }
-(int) width { return width; }
-(NSString *) description
{
    // Redéfinition de la méthode de NSObject
    return @"Ceci est un rectangle";
}
@end
```

Définir un protocole

- Les protocoles sont l'équivalent des interfaces Java.

```
@protocol IAge <NSObject>
    - (int) age;
@end

@interface Age : NSObject <IAge>
    - (int) age;
@end
```

Type de données

- id : pointeur vers un objet, peu importe son type *(void *)*
- BOOL : booléen (*YES* ou *NO*)
- IBOutlet / IBAction : permet d'associer vos objets graphiques dans le designer avec votre code
- nil : aucune valeur
  - ⇒ *[nil method]* ne renvoie pas de NullPointerException, ça ne fait rien

Méthodes

```
-(void) setHeight:(int)newHeight width:(int)newWidth;
```

- Marqueur d'accès à la méthode
  - + ⇒ méthode de classe (static)
  - - ⇒méthode d'instance
- Type de retour *(void)*
- Nom de la méthode *setHeight*
- Paramètre de la méthode *(int)newHeight*
- Nom d'appel du second paramètre width
- Nom de la variable du second paramètre newWidth

```
[horizontalRectangle setHeight:5 width:10]
```

Appel de méthodes

- On dit aussi envoyer un message à un objet
  - *[instance method];*
  - *[instance methodWithInput: input];*
- Une méthode peut renvoyer une valeur
  - *output = [instance methodWithOutput];*
  - *output = [instance methodWithInputAndOutput: input];*
- Appel de méthode de classe
  - *id myString = [NSString string];*
  - *NSString* myString = [NSString string];*

Appel de méthodes

- Méthodes imbriquées
  - o1.function1(o2.function2(4))
- En objective-C ⇒
  - *[o1 function1: [o2 function2: 4]]*

- Plus de deux niveaux d'imbrication ?
  - *[you goto: hell]*

Simplification d’écriture

```
@interface Rectangle: NSObject
{
    int height;
    int width;
}

// Setters
-(void) setHeight: (int)newHeight;
-(void) setWidth: (int)newWidth;
-(void) setHeight:(int)newHeight width:(int)newWidth;

// Getters
-(int) height;
-(int) width;

@end
```

Simplification d’écriture

```
@interface Rectangle: NSObject

@property int height;
@property int width;

-(void) setHeight:(int)newHeight width:(int)newWidth;

@end
```

Simplification d’écriture

- Les propriétés sont déclarées avec :
  - *@property (attributs…) type nom;*
  - *@property int age;*

- Génère automatiquement :
  - Un getter : *nom* (dans l’exemple : *age*)
  - Un setter : *setNom* (dans l’exemple : *setAge*)

Attributs des propriétés

- Différents attributs sont disponibles :
  - *atomic / nonatomic* : get/set atomique dans un environnement threadé ou non => par défaut : atomic
  - *strong / weak* : indique si l'instance doit être comptée par le compteur de référence du garbage collector => par défaut : strong
  - *readwrite / readonly* : indique si la propriété est en lecture seule ou non => par défaut : readwrite
  - *copy / assign* : indique si lors de l'affectation on doit copier l'objet ou non => par défaut : assign

Implémentation des propriétés

- Deux choix :
  - implémentation automatique du get/set

    @synthetize age;
  - implémentation explicite

    - (int) age { return self.age; }

    - (void) setAge: (int)newAge { self.age = newAge; }

Utilisation des propriétés : le .

- Pour accèder aux propriétés, deux choix
  - Utiliser les get/set explicitement
    - *int age = [person age];*
    - *[person setAge:23];*
  - Utiliser le . pour un accès plus simple
    - *int age = person.age;*
    - *person.age = 23;*

Chaînes de caractères

- *NSString => chaine constante non modifiable !*
- @" ... " ⇒ constante
- NSString* string = [[NSString alloc] initWithString:@"Bonjour"]
- NSString* string = [NSString initWithString::@"Bonjour"]

- *NSMutableString => même chose que NSString mais modifiable*

Tableaux

- NSArray => tableaux de tailles constantes
- NSArray* names = [[NSArray alloc] initWithObjects:@"Albert", @"Michel", nil]
- Parcours de tableau, soit boucle for « standard » soit foreach :
  - for(NSString* name in names) { //Do Stuff… }

- NSMutableArray => tableaux avec une taille modifiable

Création des objets

- Méthodes alloc et init de NSObject
  - alloc : effectue l'allocation mémoire
  - init : initialise l'objet
  - init* : initialise l'objet avec des paramètres