Introduction au développement d'application iOS

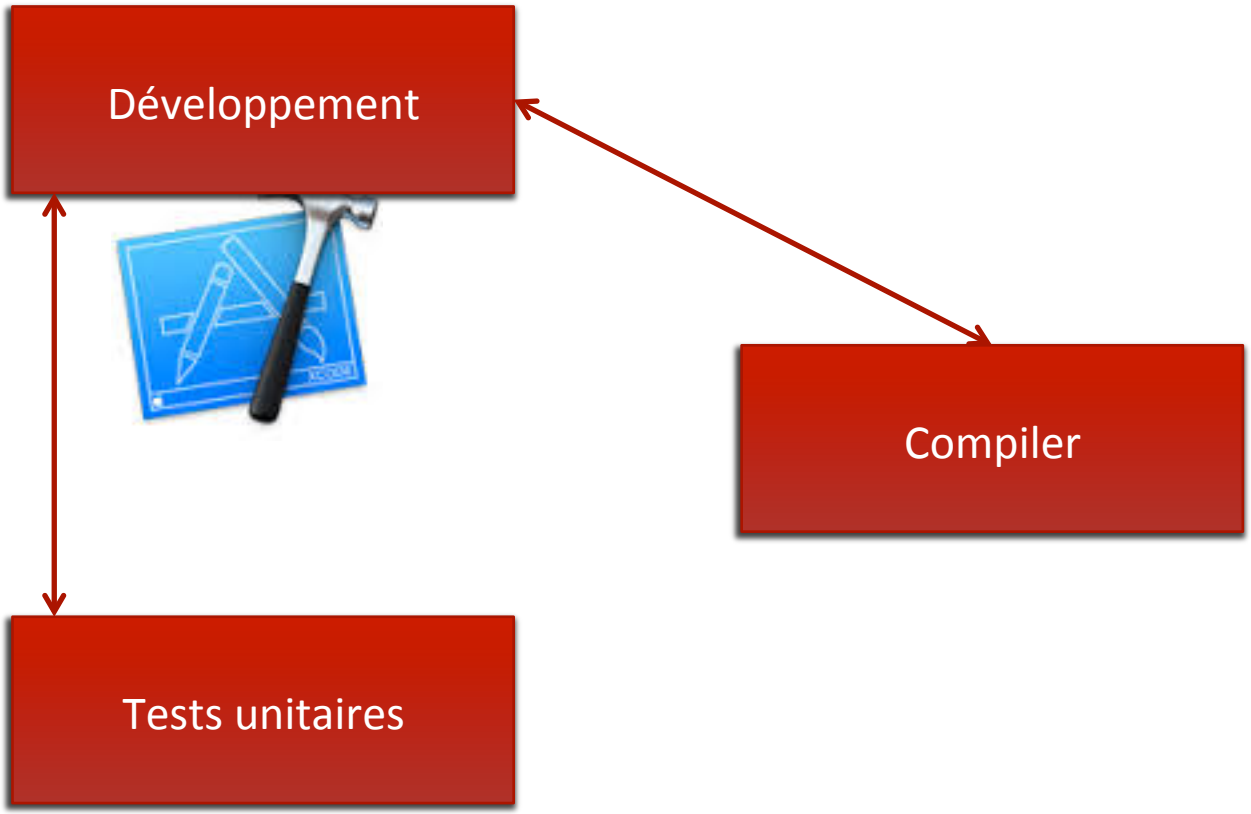
Amosse EDOUARD, PhD

Aout 2015, MBDS Haiti

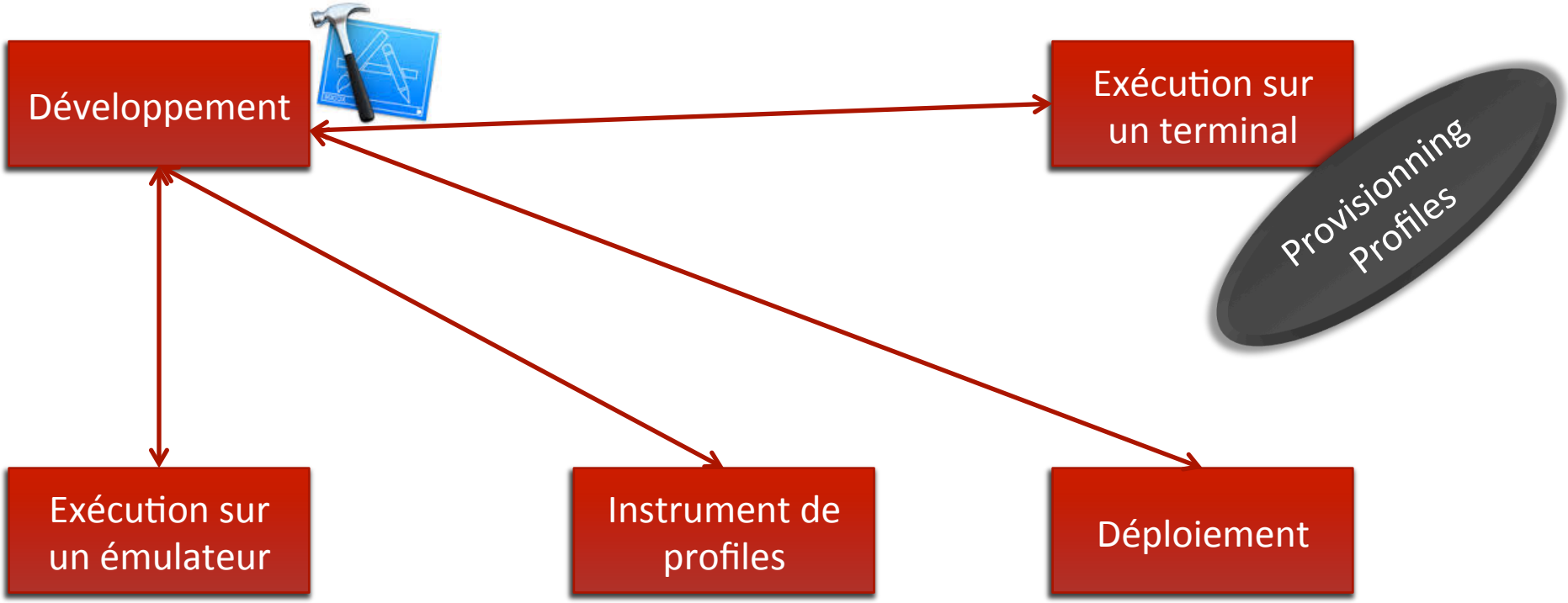
Objectifs

- Mécanique de développement d'application iOS
- Xcode, iOS SDK & Storyboard
- Guide de survie Objective-C
- Premiers exercices
  - Construction d'un premier formulaire statique

Tests et Analyses



Chaine de production d'application iOS



Xcode

- Environnement de dévloppement des applications iOS et MacOs
- Evolue au meme rythme que les versions des OS (Mac, iOS)
- Il existe toujours une possibilité de faire de l'hybride
  - Titanium
  - Phonegap
- Il est aussi possible de faire du HTML5

Principes de dépliement d'une application iOS

- Sur un émulateur
  - Gratuit
  - Fonctions limitées
    - Pas de capteur GPS
    - Pas d'apareil photo
- Sur un iPhone ou une tablette
  - Payant ☹
  - Plus compliqué ☹
  - Toutes les fonctionnalités de l'OS ☺

Les programmes de développement

	Rien	University	Enterprise	Standard
SDK	✔	✔	✔	✔
Beta	✗	✗	✔	✔
Forums	✔	✔	✔	✔
Simulateur	✔	✔	✔	✔
Déploiement	✗	✔	✔	✔
Distribution	✗	✗	✔	✗
AppStore	✗	✗	✗	✔
Coût	0$	-	299$	99$

3 techniques de déploiement

- Localement en utilisant un cable USB
  - Requiert un abonnement → Provisioning Profile
  - Possibilité d'utiliser un University program
- Via l'AppStore
  - Validation d'Apple ☹
  - Apple prélève 30% de votre CA
  - Necessite un abonnement → Distribution profile
- Via Intranet + Itunes
  - Génération d'un fichier .ipa
  - Peut etre déployé sur un serveur local
  - Certification entreprise → Distribution profile

Chaine de déploiement iOS

- Très cadrée, bien plus sécurisé que Android
- Avantages
  - Protection contre les programmes malveillants
  - Controle de qualité
- Inconvénients
  - Protection contre la concurrence
  - Mecanisme de déploiement complexe

Environnement de développement

- Xcode
  - IDE permettant de developpement des applications iOS
  - Version 6.0 (Juin 2014)
- Xcode propose un GUI + SDK
  - Graphic User Interface
    - Concevoir l'interface
  - Software Development Kit
    - Environnment intégré de développement

Xcode

- Les modules
  - StoryBoard
  - Simulateur/Debugger
  - Instrument Profiler
  - Analyse statistique
  - Gestionnaire de versions intégrés
    - Git, Subversion
  - Accès directe à la documentation

Principe de construction des interfaces

- Deux approaches
  - Une approche basée sur storyboard → Mode débutant
    - Par dessin comme au kindergarden
    - On ne me maitrise pas totalement la construction de l'interface
  - Une approche par pure programmation
    - On fait tout à la main
    - Maitrise complet du design de l'interface
    - Mais plus compliqué (réservé aux experts)

Storyboard ou le mode Kindergaten

- Interface graphique
  - Dessiner ses écrans
    - Positionnement par contraintes
  - Scénariser l'application
  - Lier les éléments de l'interface au code métier
- Génération de code "AutoLayout"
  - Introduit avec iOS 6 (2012)
  - Positionnement relatif des éléments de l'interface
    - Relations entre les éléments
    - Contraintes par rapport à l'écran

Les résolutions d'écrans

- Plusieurs types d'écrans
  - 960x640 retina 3.5" (iPode touch 3 et 4, iPhone 4/4S)
  - 1027x648 iPad 1&2 et iPad mini
  - 1136x640 retina 4" (iPod touch 5 et iPhone 5)
  - 1334x750 retina 4.7" (iPhone 6)
  - 2048x1536 retina (iPad 3/4/air, iPad mini retina)
  - 1920x1080 (iPhone 6+)
- Deux modes de comportement
  - Petits terminaux et grand terminaux
- Rotation
  - Portrait et Paysage
  - Assez bien géré par Storyboard

Les bases de Storyboard

- On va voir un exemple simple
  - Gestion des résolutions
  - Gestion de l'orientation du terminal
  - Positionnement des éléments
  - Les contraintes

Et le code …

- Objective C
- Swift

Guide de survie Objective C

- Historique
  - 1983
  - S’inspire du C
- Nous aborderons
  - Les principes du langage
  - Elements de syntaxe
  - La classe NSString

Principes d'objective-C

- Sur ensemble strict de C
  - Nouvelle syntaxe, nouveaux types : [instance method]
  - @property, @interface, @implementation, @synthesize, …
  - Id, class, selector
- Héritage simple
- Typage relativement fort par rapport à C, mécanismes d'introspection
- Classes & Instances → Objets
  - Méthodes de classes
  - Méthodes d'instances

Les méthodes

- Les instances répondent à des méthodes instances

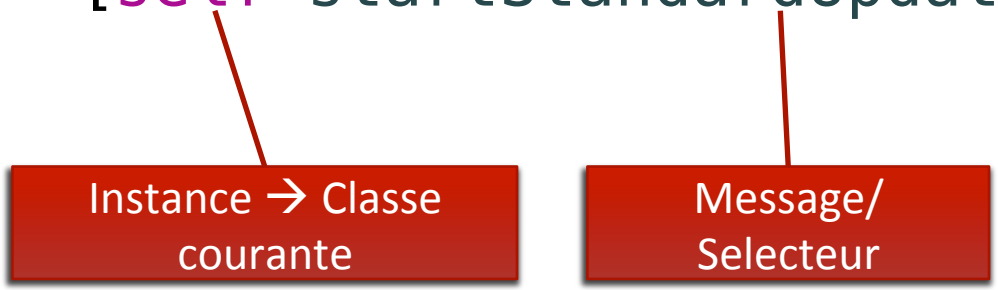
```
- (id) init;
- (float) temperature;
```

- Les classes répondent à des méthodes de classes

```
+ (id) alloc;
- (UIDevice *) currentDevice;
```

En objective-C tout est message

```
[self startStandardUpdates];
```




```
[dbManager executeQuery:query];
```

Les selecteurs

- Référencer les méthodes

```
SEL sel1 = @selector(sayHello);
SEL sel2 = @selector(sayHelloTo:grade);
```

- Utile pour l'introspection

```
- (BOOL) respondsToSelector:(SEL)aSelector;
```

Premiers éléments utiles

- NSString : Chaine de caractère
- NSLog : Afficher des informations dans la console

NSString

- Classe permettant de manipuler les chaines de charactères
- Préféré à char* de C
- Syntaxe évolué
  - En C: “C’est la pause”
  - En Objective-C : @”Hello world”
- Exemple

```
NSString *message = @"C'est la pause";
```

Manipulation des NSString

- Objets immuables → Non modifiables

```
NSString *message = @"C'est la pause";
message = @"C'est la pause, il est content";
```

- Une string peut être construite à partir d'une autre

```
NSString *completMessage = [NSString stringWithFormat:@"%@, il est content", message];
```

Opération sur les NSString

- Le selecteur (ou méthode) `stringByAppendingString`

```
NSString *completMessage = [message
stringByAppendingString:@", il est content"];
```

- Le selecteur `isEqualToString`

```
if([message isEqualToString:@"C'est la pause"])
{…}
```

- Le selecteur `containsString`

```
if([message containsString:@"C'est la pause"])
{...}
```

NSLog

- Prend une string en paramètre et l'afficher dans la console

```
NSLog (@"C'est la pause, il est content");
```

- On peut utiliser la concaténation

```
NSLog(@"%@, il est content", message);

NSLog(@"Il est %d:%dh, c'est la pause", 10,
15);
```

NSArray

- Collection ordonnée d'objets
- Deux versions
  - Immuable (non modifiable une fois instanciée)
  - Mutable (On peut modifier la liste après avoir instancié la collection)

NSArray, Exemple

```
NSArray *colors = [NSArray
arrayWithObjects:@"bleu", @"Jaune", @"Rouge",
nil];

NSLog(@"Il y a %d couleurs en tout", [colors
count]);

NSLog(@"La 3e couleur est %@", [colors
objectAtIndex:2]);
```

NSMutableArray

- Collection ordonnée et modifiable

```
//Instancier une liste mutable
NSMutableArray *mColors = [[NSMutableArray alloc] init];
//Inséerer un élément dans la liste
[mColors addObject:@"Rouge"];
//Insérer un élément à une position définie
[mColors insertObjectAtIndex:@"Vert" atIndex:2
//Enlever un élément de la liste
[mColors removeObjectAtIndex:0];
```

NSDictionnary

- Collection d'objets gérée via un système clef/valeur
- Deux versions
  - NSDictionnary → Immuable
  - NSMutableDictionnary → Mutable

NSDictionnary

```
[NSDictionary
dictionaryWithObjectsAndKeys:@"blanc",@"#FFFF
FF", @"bleu",@"0000FF",nil ];

NSLog(@"Le code de couleur du blanc est %@",
[dict valueForKey:@”#FFFFFF"]);

NSLog(@"Le code de couleur du blanc est %@",
[dict valueForKey:@"@blan"]);

NSLog(@"La clef pour la valeur blanc est %@",
[dict objectForKey:@”blanc"]);
```

```
NSMutableDictionary *mDict =
[[NSMutableDictionary alloc]init];

NSMutableDictionary *mDict =
[[NSMutableDictionary dictionary];

[mDict setObject:@"blanc" forKey:@"#FFFFFF"];

[mDict removeObjectForKey:@"#FFFFFF"];

[mDict removeAllObjects];
```

NSSet

- Collection non ordonnée (Ensemble)
- Deux types
  - NSSet → Immuable
  - NSMutableSet → Mutable

NSSet, Exemple

- Ordre fixe ou aléatoire

```
NSSet *set = [NSSet
setWithObjects:@"bleu",@"blanc", @"rouge",
nil];

NSLog(@"Une couleur au hasard %@", [set
anyObject]);
```

NSMutableSet

- Collection non ordonnée mutable

```
 NSMutableSet *mSet = [[NSMutableSet
alloc]init];

[mSet addObject:@"bleu"];

[mSet addObject:@"blanc"];

[mSet removeObject:@”blanc"];
```

Créer ses propres classes

- Deux fichiers sont necessaires
  - Interface : MaClasse.h (@interface … @end)
  - Implémentation : MaClasse .m (@implementation … @end)
- Hérite au moins de la classe NSObject
- Variables d'instances
- Comportement exprimé via des méthodes

Exemple – Classe Personne

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

@interface Person : NSObject

{

    //Variable d'instance

    int personId;

    NSString *nom;

    NSString *prenom;

} //méthodes

- (NSString *) nomComplet;

- (void) setNom :(NSString *) nom;
```

Exemple Suite - Implémentation

```
#import "Person.h”

@implementation Person

-(NSString *) nomComplet{

    return [NSString stringWithFormat:@"%@ %@",
nom,prenom];

} -(void) setNom:(NSString *)nouveauNom{

    nom =[nouveauNom copy];

}

@end
```

Getters & Setters

- Les accesseurs peuvent être générés automatiquement
- Description dans le .h
  - Défini par le mot clef : `@property`
    - Associé à un attribut d'instance
    - Spécifie les accès, la gestion de la mémoire, le comportement en environnement mutlithreadé
- Implémentation automatique dans le .m
  - Lecture : monAttribut
  - Ecriture : setMonAttribut
- Mot clef : @synthesize

Attributs et les modes d'accès

- Lecture seulement
  - Mot clef : readonly
  - Seule la méthode de lecture sera implémenté ➔ le get
- Lecture & Ecriture ***(Par défaut si rien n'est précisé)***
  - Mot clef : readwrite
  - Implémentation des méthodes de lecture et écriture
- Si pas de @synthesize, il faut implémenter les méthodes manuellement
  - Respecter les conventions de nommage

Attributs - Droits d'accès

- Accès protégé
  - Mot clef atomic
  - Gére des verrous
  - Performances dégradées
- Accès libres (mode par défaut)
  - Mot clef : nonatomic

Gestion de la mémoire

- Affectation simple mais un peu brutale (mode par défaut)
  - Mot clef : assign
- Recopie un attribut
  - Mot clef : copy
  - Création d'un nouvel objet identique au premier
- Incrémentation du pointeur de référence
  - Mot clef : retain
- Relation par rapport à l'objet parent
  - Mots clefs : strong ou weak

On peut aussi renommer les accesseurs …

- Les getter
  - getter = monAttribut
  - Désormais le getteur s'appelle monAttribut

- Les setter
  - Setter = monAttribut
  - Désormais le setter s'appelle monAttribut

Classe Person – Implémentation 2

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

@interface Person : NSObject

@property (readwrite, nonatomic, copy) NSString
*nom;

@property (readwrite, nonatomic, assign)
NSString* prenom;

-(NSString *) nomComplet;

@end
```

Classe Person – Implémentation 2

```
#import "Person.h”

@implementation Person

@synthesize nom,prenom;

-(NSString *) nomComplet{

    return [NSString stringWithFormat:@"%@ %@",
nom,prenom];

}

@end
```

Classe Person – Implémentation 2

```
#import "Person.h”

@implementation Person

@synthesize nom,prenom;

-(NSString *) nomComplet{

    return [NSString stringWithFormat:@"%@ %@",
_nom,_prenom];

}

@end
```

Classe Person – Implémentation 2

```
@interface Person : NSObject:{
    NSString *nom;
    NSString *prenom;
}
@property (readwrite, nonatomic, copy) NSString *nom;
@property (readwrite, nonatomic, assign) NSString*
prenom;
-(NSString *) nomComplet;
```

Cycle de vie d'un objet

- Depuis iOS 5
  - Le cycle de vie d'un objet est pris en charge par l'environnement
    - ARC (Automatic Reference Counting)
- Principe
  - L'environnement fait le travail à votre place
  - Il gère les compteurs de référence

Création & Déstruction d'objets

- Un objet se crée, vit et meurt
- Création
  - Allocation de mémoire
    - +(id) alloc
  - Initialisation
    - -(id) init

Création & Déstruction d'objets

- Déstruction
  - -(id) dealloc;
  - `[person dealloc];`
- Implémentation par défaut dans NSObject
  - N'oubliez pas toute object en Objective-C est un NSObject

Classe Person - Exemple

- Initialisation typique

```
Person *person = [[Person alloc]init];
```

- Les méthodes alloc et init sont définies par défaut
- On peut surcharger la méthode init

Surcharger la méthode init

```
-(id) init{
    if(self == [super init]){
       //valeur par défaut pour certains atributs
        _sexe = @"Féminin";
    }
    return self;
}
```

Plusieurs manières de surchager init

```
-(id) init
```

- Pas besoin de déclarer cette méthode dans le .h

```
-(id) initWIthNomEtPrenom : (NSString *) nom
etPrenom :(NSString *)prenom;
```

Plusieurs manières de surchager init

```
- (id) initWIthNomEtPrenom:(NSString *)nom
etPrenom:(NSString *)prenom{

    self.nom = nom;

    [self setPrenom:prenom];

    return [self init];

}
```

Et le cycle de vie …

- Si on n'utilisait pas ARC, il fallait y penser mais….
- Et s'il fallait y penser, on devrait implémenter la méthode dealloc

```
-(void) dealloc{
    [_nom dealloc];
    [_prenom dealloc];
    [super dealloc];
}
```

Un peu de pratique…

- Gestion des événements sur les vues
- Gestion de la navigation
- Traitement de formulaire