

Partie 1

Introduction à iOS

Introduction à iOS

- 1. Présentation**
- 2. Typage, pointeurs et ARC (Automatic Reference Counting)**
- 3. Messaging**
- 4. Structures de données Foundation (String & Collections).**
- 5. Construction & utilisation de vos Interfaces**
- 6. Construction & utilisation de vos Categories**
- 7. Construction & utilisation de vos Extensions**
- 8. Construction & implementation de vos Protocols**

Présentation



Présentation



APIs Bas niveau
(Très proche des composants, niveau d'abstraction quasi nul)

Accelerate Framework (langage C)

Algèbre linéaire (Matrices, vecteurs, traitement du signal, ...)

External Accessory Framework (langage C)

Gestion des composants branchés via le dock. (drivers)

Local Authentication Framework (Objective-C)

Authentification via Touch-ID

System (BSD/Darwin)

POSIX Threads, BSD Sockets, File-system,
Allocation mémoire, ...

Présentation



APIs Haut niveau (Ce que vous utiliserez dans 99% de vos développements)

Core Foundation (language C), Foundation (Objective-C)

Arrays (CFArray vs. NSArray)
Dictionaries (CFDictionary vs. NSDictionary)
Strings (CFString vs. NSString)
...

iCloud Storage

iCloud document storage: stockage de fichiers.
iCloud KV storage: stockage de petites informations concernant votre application.
iCloudKit storage: Notamment utilisé pour du contenu partagé.

Block Objects

Fonctions anonymes.

In-App Purchase

SQLite, CoreData

Core Location, Core Motion, Event Kit, Health Kit, Home Kit, ...



Présentation



APIs Haut niveau

(très utilisé dans le domaine des jeux-vidéo,
ou pour tout ce qui concerne les animations / sons joués par votre application)

Graphics Technologies

UIKit graphics, Core Graphics, Core Animation, OpenGL, Metal



Audio Technologies

Media Player, AV Foundation, OpenAL, Core Audio

Vidéo Technologies

AVKit, AV Foundation, Core Media

Présentation



APIs Haut Niveau

(Couche graphique d'iOS, utilisé également dans 99% de vos développements)

UIKit

Tout ce qui est vu par un utilisateur.

Storyboards

Manipulation et design de vos interfaces et de leur enchainement.

AutoLayout

Positionnement intelligent de vos éléments.

iAd

Intégration de publicités au sein de votre application.

Gesture Recognizers

Détection des gestes (swipe, touch, pinch, ...)

Présentation

Environnement iOS	
Languages de programmation	Objective-C, Swift
Principaux Frameworks	Fondation, UIKit
Principaux outils	   Xcode, Instruments, iOS Simulator
Design patterns	MVC, Delegate, Observer,...

Typage, pointeurs et ARC

Lorsque le type d'une variable est déterminé à la compilation, on parle de **typage fort statique** (ex: Java). Les valeurs de cette variable devront être obligatoirement de ce type.

Autrement, dans les cas où ce ne sont pas les variables qui ont un type, mais les valeurs, on parle de **typage dynamique** ou **typage latent** (ex: python).

En opposition, on parle de **typage faible** (ex: javascript) lorsque le langage admet qu'une variable puisse changer de type au cours de son existence.

Objective-C est un **langage à typage fort... statique & dynamique !**

A l'utilisation, vous devrez préciser le type de vos variables lors de leur déclaration pour toutes vos primitives et pointeurs.

Pour vos objets, le type **id** offre une detection du type de vos objets **au moment de l'execution de votre code (runtime)**.

Typage, pointeurs et ARC

Objective-C est une décoration du langage C, il hérite de l'ensemble de ses primitives. Tout ce qui fonctionne en C fonctionne en Objective-C.

Exemple des primitives hérité du C:

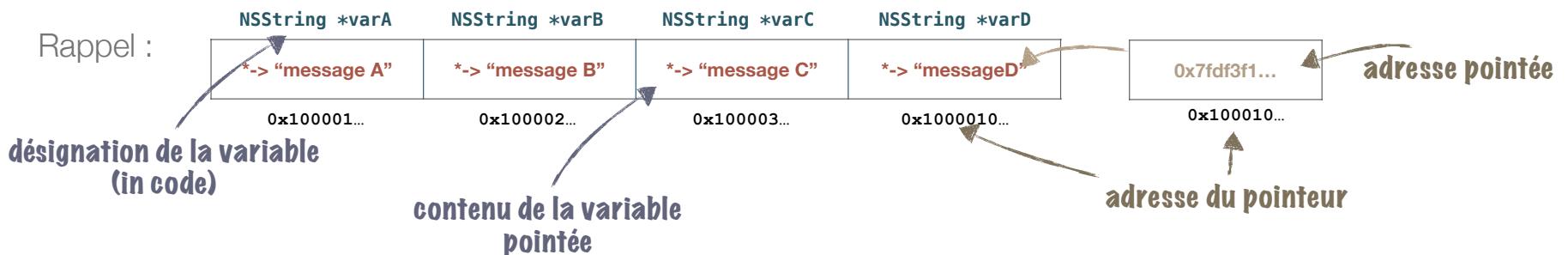
- **void, short, int, float, long, long long, double, long double, char, ...**

Liste des primitives Objective-C:

- **NSInteger & NSUInteger**
“surdéfinition” des types **int** et **unsigned int**, adaptée à l’architecture executant le code (32bits vs. 64bits).
- **Class**
Représentation d’une Classe sous forme de primitive.
- **SEL**
Structure de données permettant de stocker des **selector**. Les selectors peuvent être comparés à des pointeurs de fonction.
- **id**
Type générique. Précise au compilateur que sa valeur sera un object Objective-C.
- **nil**
Nil ne fait pas vraiment partie des primitives mais est utilisé comme valeur **null**. Dans ce cas pourquoi ne pas utiliser null?
nil est capable de recevoir des messages.

Typage, pointeurs et ARC

Objective-C est un langage orientée Objet. L'utilisation de ce paradigme se fait par l'intermédiaire de pointeurs.



```
int main(int argc, const char * argv[]) {
    @autoreleasepool {
        NSString *message = @"Hello World";

        NSLog(@"Address of var 'message': %p", &message);
        NSLog(@"Address of var's pointer: %p", message);
        NSLog(@"Content of message: %@", message);
    }
    return 0;
}

2014-11-02 22:33:07.750 ios-introduction[1210:109402] Address of var 'message': 0x7fff5fbff7c8
2014-11-02 22:33:07.750 ios-introduction[1210:109402] Address of var's pointer: 0x100001038
2014-11-02 22:33:07.750 ios-introduction[1210:109402] Content of message: Hello World
```

Typage, pointeurs et ARC

Avant l'arrivée d'**iOS 5** la gestion de la mémoire se faisait **manuellement**, par manipulation du **compteur de références**. De ce fait, les développeurs devaient maîtriser les concepts et instructions liés à la gestion de mémoire.

Explication sur le compteur de référence :

La durée de vie d'une instance correspond à son nombre de référence. Si celui-ci arrive à **0**, l'espace mémoire alloué pour cette variable est libéré.

```
int main(int argc, const char * argv[]) {
    NSArray *array = [[NSArray alloc] initWithObjects:@"message1", @"message2", nil];
    NSLog(@"%@", array);

    [array retain];
    [array release];
    [array release];
    return 0;
}
```

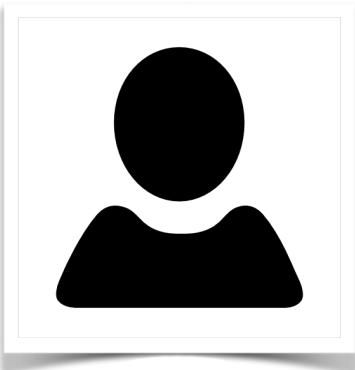
Dans cet exemple, la variable **array** a son nombre de référence initialisé à **1** lors de son allocation **[NSArray alloc]**.

On **décrémente** manuellement ce nombre via l'instruction **[array release]**;
On **incrémente** le compteur via l'instruction **[array retain]**;

Aujourd'hui **ARC** (**A**utomatic **R**eference **C**ounting) étant très performant, ces instructions ne sont plus nécessaires.
On peut comparer ce système au **garbage collector de JAVA**.

Messaging

Prenons deux instances de la classe Person.



Ed

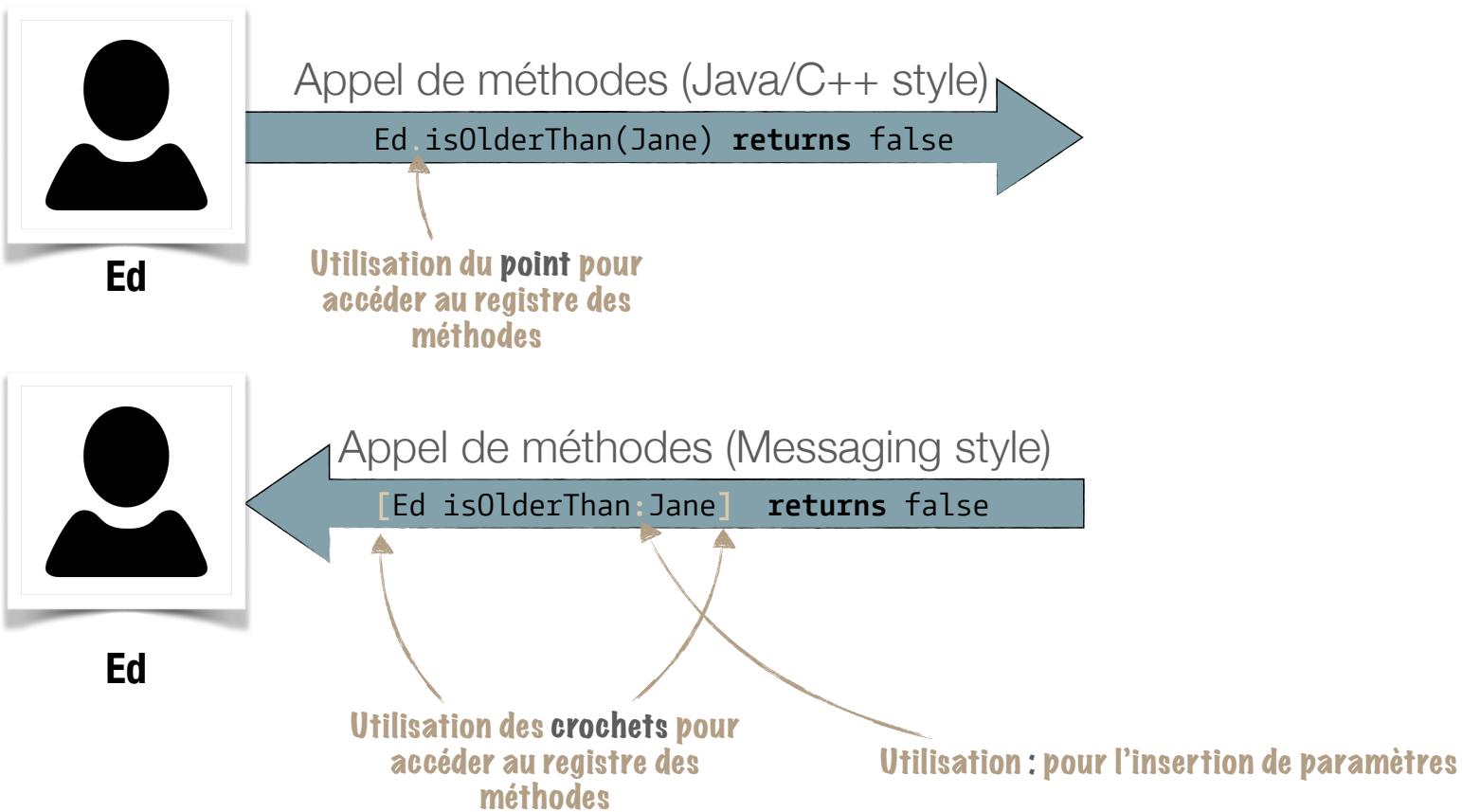
Propriétés
<ul style="list-style-type: none">• firstName : “Edward”• lastName : “Egg”• birthDate : 15/05/1987
Méthodes
<ul style="list-style-type: none">• sayHello()• saySomething()• isOlderThan(Person p)• setWeightAndHeight(int w, int h)



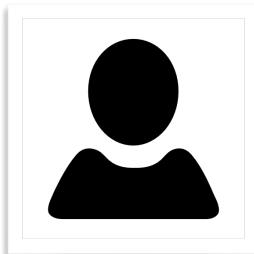
Jane

Propriétés
<ul style="list-style-type: none">• firstName : “Jane”• lastName : “To”• birthDate : 26/01/1943
Méthodes
<ul style="list-style-type: none">• sayHello()• saySomething()• isOlderThan(Person p)• setWeightAndHeight:(int, int)

Messaging



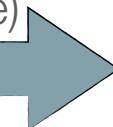
Messaging



Ed

Appel de méthodes (Java/C++ style)

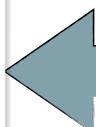
`Ed.setWeightAndHeight(100, 200)`



Ed

Appel de méthodes (Messaging style)

`[Ed setWeight:100 andHeight:200] void`



Structures de données Foundation : **NSString**

Création

```
NSString *make = @"Porsche";
NSString *model = @"911";

int year = 1999;

NSString *message = [NSString stringWithFormat:@"This car is a %@ %@ from %i.", make, model, year];
NSLog(@"%@", message);
```

Énumération

```
NSString *bank = @"Crédit Agricole";
for (int i=0; i<[bank length]; i++) {
    unichar letter = [bank characterAtIndex:i];
    NSLog(@"%@", letter, i);
```

Structures de données Foundation : **NSString**

Comparaison

```
NSString *car = @"Porsche Boxster";
if ([car isEqualToString:@"Porsche Boxster"])
    NSLog(@"That car is a Porsche Boxster");

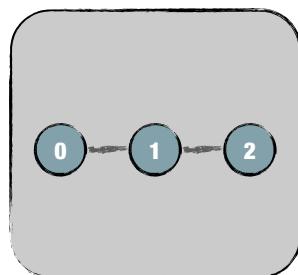
if ([car hasPrefix:@"Porsche"])
    NSLog(@"That car is a Porsche of some sort");

if ([car hasSuffix:@"Carrera"])
    NSLog(@"That car is a Carrera"); // This won't be executed
```

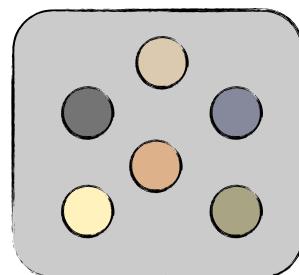
Comparaison 2 (via compare:)

```
NSString *otherCar = @"Ferrari";
NSComparisonResult result = [car compare:otherCar]; // Warning Case sensitive
if (result == NSOrderedAscending)
    NSLog(@"The letter 'P' comes before 'F'");
else if (result == NSOrderedSame)
    NSLog(@"We're comparing the same string");
else if (result == NSOrderedDescending)
    NSLog(@"The letter 'P' comes after 'F'");
```

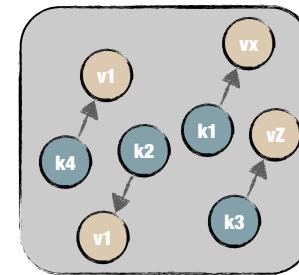
Structures de données Foundation : Collections



NSArray



NSSet



NSDictionary

Structures de données Foundation : NSSet

Création

```
NSSet *names = [NSSet setWithObjects:@"Edward", @"Jessica", @"Logan", nil];
NSLog(@"%@", names);

NSArray *namesArray = @[@"Edward", @"Jessica", @"Logan", @"Jessica"];
NSSet *removedDuplicates = [NSSet setWithArray:namesArray];
NSLog(@"%@", removedDuplicates);
```

Énumération

```
NSSet *persons = [NSSet setWithObjects:@"Sarah", @"James", @"Oz", @"Eric", @"John", nil];
NSLog(@"This NSSet has %li elements", persons.count);
for (id person in persons) {
    NSLog(@"%@", person);
}
```

Structures de données Foundation : NSSet

Comparaison

```
NSSet *japaneseCars = [NSSet setWithObjects:@"Honda", @"Nissan", @"Mitsubishi", @"Toyota", nil];
NSSet *johnsFavoriteCars = [NSSet setWithObjects:@"Honda", @"Nissan", nil];
NSSet *marysFavoriteCars = [NSSet setWithObjects:@"Toyota", @"Alfa Romeo", nil];

if ([johnsFavoriteCars isEqualToSet:japaneseCars])
    NSLog(@"John likes all the Japanese auto Carrs and no others");

if ([johnsFavoriteCars intersectsSet:japaneseCars])
    NSLog(@"John likes at least one Japanese auto Carr"); //john's favorites cars that are in japanese cars.

if ([johnsFavoriteCars isSubsetOfSet:japaneseCars])
    NSLog(@"All of the auto makers that John likes are Japanese"); //john's favorites cars are all in the japanese cars.

if ([marysFavoriteCars isSubsetOfSet:japaneseCars])
    NSLog(@"All of the auto makers that Mary likes are Japanese");
```

Structures de données Foundation : NSArray

Création

```
NSArray *movies = @[@"Guardian Of the Galaxy", @"Avengers" , @"X-men : Days of Futur Past",
@@"Batman : The Dark Knight"];
```

```
NSArray *series = [NSArray arrayWithObjects:@"Game of Thrones", @"Breaking Bad", @"House of Cards",
@@"Walking Dead", @"Californication", nil];
```

```
NSArray *videoGames = [[NSArray alloc] initWithObjects:@"Zelda: Ocarina Of Times", @"Final Fantasy 7",
@@"League Of Legends", @"BattleField 4", @"Super Mario Bross", nil];
```

```
NSLog(@"%@", movies);
NSLog(@"%@", series);
NSLog(@"%@", videoGames);
```

Structures de données Foundation : NSArray

Énumération

```
// With fast-enumeration
for (NSString *movie in movies)
    NSLog(@"movie: %@", movie);

// With traditional for loop
for (int i=0; i<videoGames.count; i++)
    NSLog(@"Video game: %@", videoGames[i]);
```

Comparaison

```
NSArray *movies = @[@"Guardian Of the Galaxy", @"Avengers" , @"X-men : Days of Futur Past",
@"Batman : The Dark Knight"];
NSArray *sameMovies = [NSArray arrayWithObjects:@"Guardian Of the Galaxy", @"Avengers" ,
@"X-men : Days of Futur Past", @"Batman : The Dark Knight", nil];

if ([movies isEqualToString:sameMovies])
    NSLog(@"Litteral and non-litteral are the same!");
```

Structures de données Foundation : NSDictionary

Création

```
// Literal syntax
NSDictionary *inventory = @{
    @"Mercedes-Benz SLK250" : [NSNumber numberWithInt:13],
    @"Mercedes-Benz E350" : [NSNumber numberWithInt:22],
    @"BMW M3 Coupe" : [NSNumber numberWithInt:19],
    @"BMW X6" : [NSNumber numberWithInt:16],
};

// Values and keys as arguments
inventory = [NSDictionary dictionaryWithObjectsAndKeys:
    [NSNumber numberWithInt:13], @"Mercedes-Benz SLK250",
    [NSNumber numberWithInt:22], @"Mercedes-Benz E350",
    [NSNumber numberWithInt:19], @"BMW M3 Coupe",
    [NSNumber numberWithInt:16], @"BMW X6", nil];

// Values and keys as arrays
NSArray *models = @[@"Mercedes-Benz SLK250", @"Mercedes-Benz E350",
    @"BMW M3 Coupe", @"BMW X6"];
NSArray *stock = @[[NSNumber numberWithInt:13],
    [NSNumber numberWithInt:22],
    [NSNumber numberWithInt:19],
    [NSNumber numberWithInt:16]];
inventory = [NSDictionary dictionaryWithObjects:stock forKeys:models];
NSLog(@"%@", inventory);
```

Structures de données Foundation : NSDictionary

Énumération

```
NSDictionary *inventory = @{
    @"Mercedes-Benz SLK250" : [NSNumber numberWithInt:13],
    @"Mercedes-Benz E350" : [NSNumber numberWithInt:22],
    @"BMW M3 Coupe" : [NSNumber numberWithInt:19],
    @"BMW X6" : [NSNumber numberWithInt:16],
};

NSLog(@"We currently have %ld models available", [inventory count]);
for (id key in inventory) {
    NSLog(@"There are %@ %@'s in stock", inventory[key], key);
}
```

Comparaison

```
NSDictionary *warehouseInventory = @{
    @"Mercedes-Benz SLK250" : [NSNumber numberWithInt:13],
    @"Mercedes-Benz E350" : [NSNumber numberWithInt:22],
    @"BMW M3 Coupe" : [NSNumber numberWithInt:19],
    @"BMW X6" : [NSNumber numberWithInt:16],
};

NSDictionary *showroomInventory = @{
    @"Mercedes-Benz SLK250" : [NSNumber numberWithInt:13],
    @"Mercedes-Benz E350" : [NSNumber numberWithInt:22],
    @"BMW M3 Coupe" : [NSNumber numberWithInt:19],
    @"BMW X6" : [NSNumber numberWithInt:16],
};

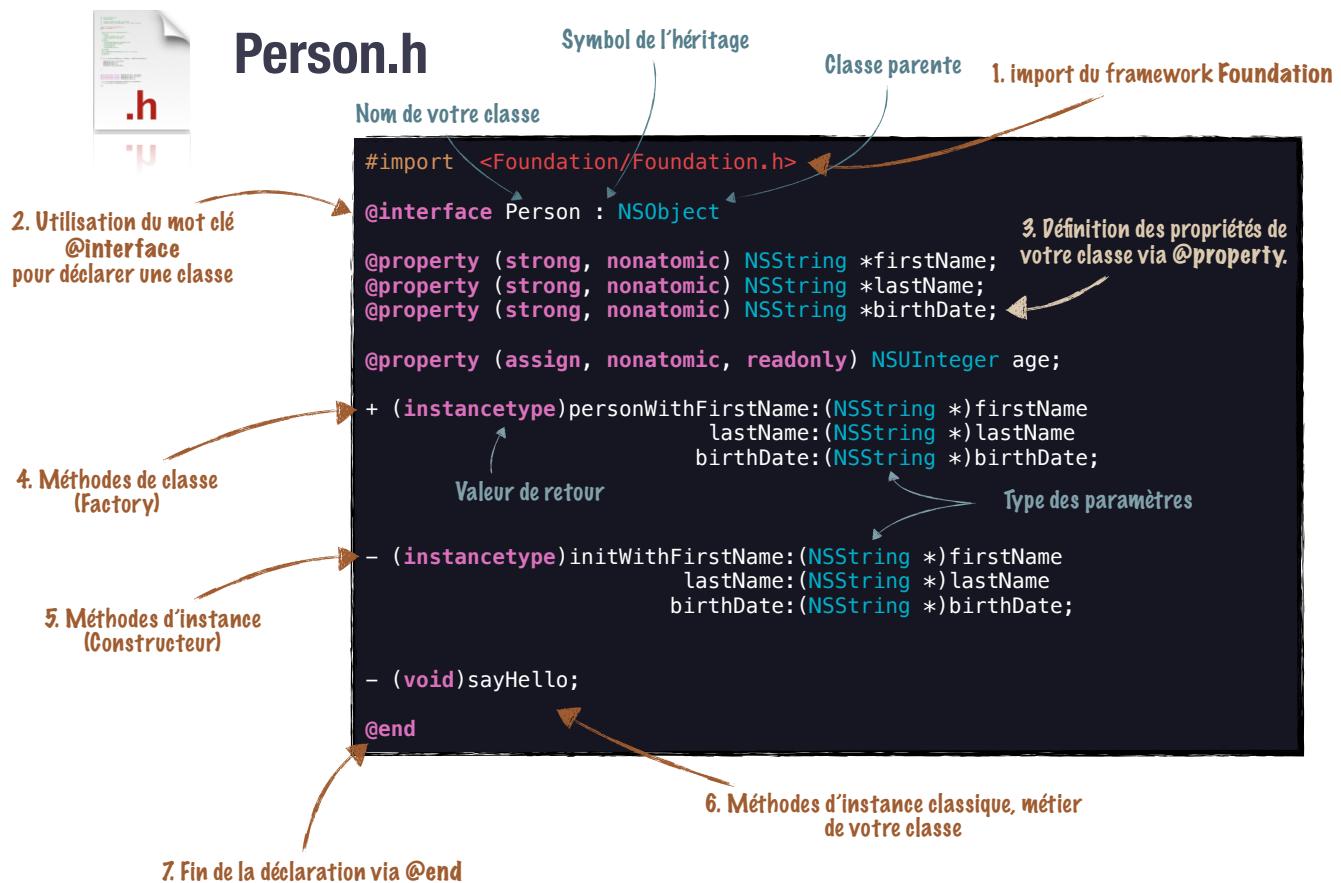
if ([warehouseInventory isEqualToDictionary:showroomInventory])
    NSLog(@"Why are we storing so many cars in our showroom?");
```

Besoin d'une information sur une classe en particulier ?

**iOS Developer Library
est là pour ça.**

“X class reference” dans google
(ex : NSString class reference)

Construction de vos interfaces



Construction de vos interfaces

Person.m

.m

1. Import de vos headers, au minimum celle de la classe que vous implémentez

2. Définition de votre implémentation par le mot clé `@implementation` + le nom de la classe que vous implémentez

```
#import "Person.h"

@implementation Person

+ (instancetype)personWithFirstName:(NSString *)firstName lastName:(NSString *)lastName birthDate:(NSString *)birthDate {
    Person *person = [[Person alloc] initWithFirstName:firstName lastName:lastName birthDate:birthDate];
    return person;
}

- (instancetype)initWithFirstName:(NSString *)firstName lastName:(NSString *)lastName birthDate:(NSString *)birthDate {
    self = [super init];
    if (self){
        self.firstName = firstName;
        self.lastName = lastName;
        self.birthDate = birthDate;
    }
    return self;
}
- (void)sayHello{
    NSLog(@"%@", @" says: Hello!", _firstName, _lastName);
}

@end
```

3. implémentation de vos méthodes

4. Fin de l'implémentation

Utilisation de vos Interfaces

1. Création

message "init" envoyé à l'instance renvoyée par la classe Person,
dont l'adresse sera stockée dans la variable person.

alloc

Méthode de **classe** de NSObject. Retourne une **nouvelle instance** de la classe utilisée.

init (message d'appel au constructeur)

Méthode **d'instance** de NSObject. Cette méthode, surdéfinie par les classes héritant de NSObject, permet l'initialisation de l'objet (ex: assignation de valeurs). Cette méthode peut être comparée au constructeur.

2. Comparaison

```
Person *p1 = [[Person alloc] init];
Person *p2 = [[Person alloc] init];

p1.firstName = p2.firstName = @"Luc";
p1.lastName = p2.lastName = @"Lamin";

NSLog(@"%@", p1 == p2);
NSLog(@"%@", [p1 isEqualToPerson:p2]);
```

• Comparaison de pointeurs

l'utilisation de l'opérateur **==** permet de voir si deux pointeurs référence la même adresse.

La notion de comparaison porte sur l'aspect atomique de la ressource. (une seule et même instance comparée)

• Comparaison de valeur

la majorité des objets en Objective-C propose des signature comme **[anObject isEqualTo... anotherObject]** ;

La notion de comparaison ici porte sur la description de votre objet. Plusieurs instances peuvent être égales.

Utilisation de vos Interfaces

3. Création depuis une factory

```
#import <Foundation/Foundation.h>
#import "Person.h"
int main(int argc, const char * argv[]) {
    @autoreleasepool {
        Person *person = [Person personWithFirstName:@"Julien" lastName:@"Sarazin" birthDate:nil];
        [person sayHello];
    }
    return 0;
}
2014-11-02 22:33:07.750 ios-introduction[1210:109402] Julien Sarazin says "Hello!".
```

factory

message paramètré envoyé à la classe Person

Pour des raisons historiques, concernant notamment la gestion mémoire manuelle, chaque classe dispose de sa propre factory (Design Pattern Factory) proposant de renvoyer des instances pré-configurées.

Avant iOS5 ces factories renvoyaient des instances “autorelease” ce qui permettait qu’elles soient désallouées automatiquement.

Reste très utilisé aujourd’hui car plus rapide que de passer par `[[alloc] init]`.

Construction de vos Extensions

Propriétés et méthodes pseudo-privées



Person.m

```
#import "Person.h"          1. Définition d'une extension dans votre fichier d'implémentation
#import "NSArray+RandomObject.h"

@interface Person ()           ◀
@property (strong, nonatomic, readonly) NSArray *_knownWords;
@end

@implementation Person

+ (instancetype)personWithFirstName:(NSString *)firstName lastName:(NSString *)lastName birthDate:(NSDate *)birthDate
{
    Person *person = [[Person alloc] initWithFirstName:firstName lastName:lastName birthDate:birthDate];
    return person;
}

- (instancetype)initWithFirstName:(NSString *)firstName lastName:(NSString *)lastName birthDate:(NSDate *)birthDate {
    self = [super init];
    if (self){
        self.firstName = firstName;
        self.lastName = lastName;
        self.birthDate = birthDate;

        _knownWords = @[@"Hello", @"Goodbye", @"I'm a Person", @"I'm not a Dog", @"U MAD Bro?"];
    }
    return self;
}

- (void)sayHello{
    NSLog(@"%@", self.firstName, self.lastName);
}
@end
```

Propriété pseudo-privée

Construction de vos Catégories



NSArray+RandomObject.h

```
#import <Foundation/Foundation.h>
@interface NSArray (RandomObject)
- (id)randomObject;
@end
```

Symboles de l'extension

1. Import de la classe que vous voulez étendre. Dans ce contexte on import fondation car NSArray appartient au Framework.

Nom de la catégorie

2. Signatures de vos méthodes

Construction de vos Catégories



NSArray+RandomObject.m

```
#import "NSArray+RandomObject.h" ← 1. Tout comme les interfaces,  
@implementation NSArray (RandomObject) vous importez vos headers.  
  
- (id)randomObject{  
    if(self.count == 0)  
        return nil; ← 2. Implementation de vos signatures  
  
    NSUInteger index = arc4random() % self.count;  
    return self[index];  
}  
  
@end
```

Utilisation de vos Catégories



Person.m

.m

W

1. Import de la catégorie que vous souhaitez utiliser.

```
#import "Person.h"
#import "NSArray+RandomObject.h" ◀

@interface Person ()
@property (strong, nonatomic, readonly) NSArray *knownWords;
@end

@implementation Person

...

- (void)saySomething{
    NSLog(@"%@", [self.knownWords randomObject]);
}
@end
```

2. Utilisation directe, comme si cela était une méthode native de votre classe.

Différences entre Catégories et Extensions

Catégories

En objective-C lorsque vous voulez ajouter des fonctionnalités à votre classe, indépendamment de son métier de base, vous pouvez utiliser des catégories.

Attention, l'utilisation des catégories **permet uniquement** d'ajouter de **nouvelles méthodes** et non de nouvelles propriétés.

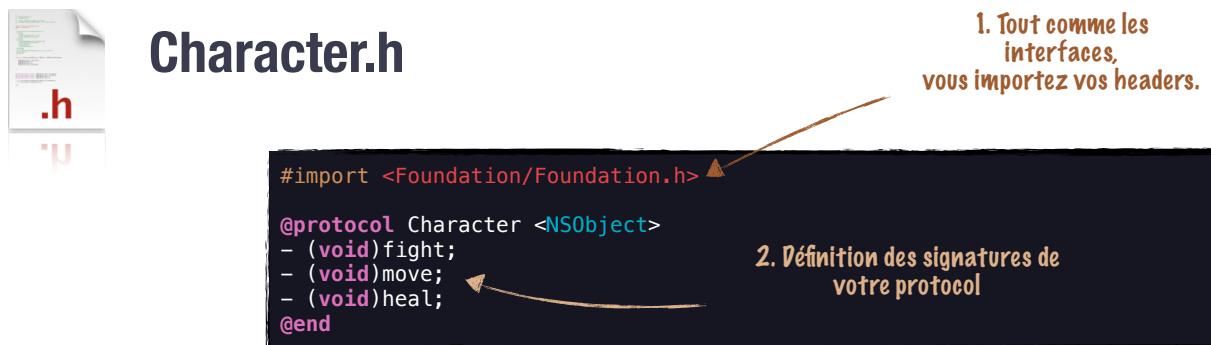
Ces comportements sont exposés **publiquement**.

Extensions

En objective-C lorsque vous désirez ajouter des **comportements privés** à vos classes cela se fait par l'intermédiaire d'extensions.

Il est très courant de voir la **présence d'une extension dans les fichiers d'implémentations** de vos classes.

Construction & implementation de vos protocoles



En Objective-C les **Protocols** peuvent être comparés aux **Interfaces Java**. Attention, certaines différences subsistent, notamment lors de leur implémentation. Il n'est pas obligatoire (mais **très fortement conseillé**), d'implémenter l'ensemble les signatures.

Ces comportements apportent le concept de **polymorphisme**.

Construction & implementation de vos protocoles



Rogue.h

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

@interface Rogue : NSObject <Character>
@end
```

Symbolise l'implémentation d'un protocol



Rogue.m

```
#import "Rogue.h"

@implementation Rogue
- (void)move{
    NSLog(@"Rogue: Moving in the shadow.");
}

- (void)heal{
    NSLog(@"Rogue: WTF ? I can't HEAL?! :/");
}

- (void)fight{
    NSLog(@"Rogue: Using dagger.");
}
@end
```

Implémentation du protocol

Construction & implementation de vos protocoles



Wizard.h

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

@interface Wizard : NSObject <Character>
@end
```

Wizard répond aux
signatures de Character



Wizard.m

```
#import "Wizard.h"

@implementation Wizard
- (void)move{
    NSLog(@"Wizard: Flying like a sir.");
}

- (void)fight{
    NSLog(@"Wizard: Casting spells.");
}

- (void)heal{
    NSLog(@"Wizard: Healing everybody with magic.");
}
@end
```

Implémentation du protocole

Construction & implementation de vos protocoles



Warrior.h

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

@interface Warrior : NSObject <Character>
@end
```

Warrior répond aux
signature de Character



Warrior.h

```
#import "Warrior.h"

@implementation Warrior
- (void)move{
    NSLog(@"Warrior: Walking.");
}

- (void)heal{
    NSLog(@"Warrior: Healing myself with a bandage.");
}

- (void)fight{
    NSLog(@"Warrior: Brwaaaaaa! Taste my sword!!!!!!");
}
@end
```

Implémentation du
protocole.

Construction & implementation de vos protocoles



VideoGame.h

```
#import <Foundation/Foundation.h>
#import "Character.h" ← Import du protocol.

@interface VideoGame : NSObject
- (void)addCharacter:(id <Character>)character; On autorise uniquement
- (void)start;           les entités de type
@end                      Character à être ajoutées
```



VideoGame.m

```
#import "VideoGame.h"

@interface VideoGame () ← Propriété privée, donc impossible
@property (strong, nonatomic) NSMutableArray *characters;
@end                   d'y accéder depuis l'extérieur

@implementation VideoGame
- (instancetype)init{
    self = [super init];
    if (self){
        self.characters = [[NSMutableArray alloc] init];
    }
    return self;
}

- (void)addCharacter:(id<Character>)character{ ← Setter avec un niveau de contrôle
    [self.characters addObject:character];       sur le type inséré
}
- (void)start{
    NSLog(@"Game started.");
    for (id character in self.characters){
        [character move];
        [character fight];
        [character heal];
    }
    NSLog(@"Game ended.");
}
@end
```

Le type id pouvant être n'importe quel type d'instance l'appel se fait sans problème.

Utilisation de vos protocoles

```
#import <Foundation/Foundation.h>
#import "VideoGame.h"
#import "Rogue.h"
#import "Warrior.h"
#import "Wizard.h"
#import "Troll.h"

int main(int argc, const char * argv[]) {
    @autoreleasepool {
        Rogue *rogue = [[Rogue alloc] init];
        Warrior *warrior = [[Warrior alloc] init];
        Wizard *wizard = [[Wizard alloc] init];

        Troll *troll = [Troll new];

        VideoGame *videoGame = [VideoGame new];
        [videoGame addCharacter:rogue];
        [videoGame addCharacter:warrior];
        [videoGame addCharacter:wizard];
        [videoGame addCharacter:troll];
        [videoGame start];
    }
    return 0;
}
```

Ajout de nos characters, sans problème.

WARNING! lorsque vous essayez d'insérer un objet qui ne respect pas le protocol demandé.

```
2014-10-28 22:05:03.119 Hello World[1575:53277] Game started.
2014-10-28 22:05:03.119 Hello World[1575:53277] Rogue: Moving in the shadow.
2014-10-28 22:05:03.120 Hello World[1575:53277] Rogue: Using dagger.
2014-10-28 22:05:03.120 Hello World[1575:53277] Rogue: WTF ? I can't HEAL?! :/
2014-10-28 22:05:03.120 Hello World[1575:53277] Warrior: Walking.
2014-10-28 22:05:03.120 Hello World[1575:53277] Warrior: Brwaaaaaaaa! Taste my sword!!!!!
2014-10-28 22:05:03.138 Hello World[1575:53277] Warrior: Healing myself with a bandage.
2014-10-28 22:05:03.138 Hello World[1575:53277] Wizard: Flying like a sir.
2014-10-28 22:05:03.138 Hello World[1575:53277] Wizard: Casting spells.
2014-10-28 22:05:03.138 Hello World[1575:53277] Wizard: Healing everybody with magic.
2014-10-28 22:05:03.138 Hello World[1575:53277] Game ended.
```

Manipulation de vos getters/setters



Person.h

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

@interface Person : NSObject
    Ajout de getter/setter (readwrite par défaut)
    @property (strong, nonatomic) NSString *firstName;
    @property (strong, nonatomic, readwrite) NSString *lastName;
    Ajout de getter only
    @property (assign, nonatomic, readonly)NSUInteger age;

    + (instancetype)personWithFirstName:(NSString *)firstName
                                    lastName:(NSString *)lastName
                               birthDate:(NSString *)birthDate;

    - (instancetype)initWithFirstName:(NSString *)firstName
                                lastName:(NSString *)lastName
                               birthDate:(NSString *)birthDate;

    - (void)sayHello;
    - (void)saySomething;
    - (BOOL)isOlderThanPerson:(Person *)person;
    - (BOOL)isEqualToString:(Person *)person;
@end
```

Manipulation de vos getters/setters



Person.h

.m

```
#import "Person.h"
#import "NSArray+RandomObject.h"

@interface Person ()
@property (strong, nonatomic, readonly) NSArray *knownWords;
@property (strong, nonatomic) NSDate *birthDate;
@property (assign, nonatomic, readwrite) NSUInteger age;
- (NSUInteger)ageFromBirthDate;
@end

@implementation Person
- (instancetype)initWithFirstName:(NSString *)firstName lastName:(NSString *)lastName birthDate:(NSString *)birthDate {
    self = [super init];
    if (self){
        self.firstName = firstName;
        self.lastName = lastName;

        NSDateFormatter *dateFormatter = [[NSDateFormatter alloc] init];
        [dateFormatter setDateFormat:@"dd/MM/yyyy"];

        self.birthDate = [dateFormatter dateFromString:birthDate];
        self.age = self.ageFromBirthDate;
    }
    _knownWords = @[@"Hello", @"Goodbye", @"I'm a Person", @"I'm not a Dog", @"U MAD Bro?"];
}
return self;
}
@end
```

Surdéfinition en private de la propriété.
Elle obtient maintenant un getter/setter

Accès directe à la propriété de votre
classe sans passer par le getter.

Fin de la Partie 1

Travaux pratiques!