

Développement iOS

LP IEM

Développement iOS

■ Développement mobile

□ Un marché récent

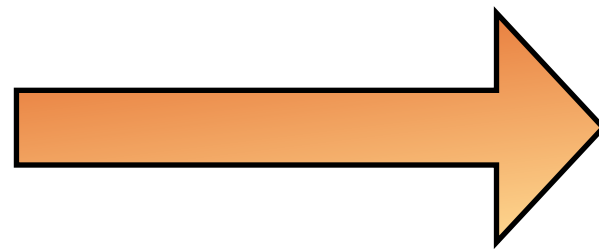
□ Sortie du premier iPhone en 2007

BlackBerry App World

Android Market
(Google play)



Nokia Ovi store



App Store

Windows market place



□ Développer sur iOS

- IDE sur OS X uniquement (Xcode et en option AppCode)
- Créer un compte Apple Developer
 - Le compte donne accès au SDK et permet d'utiliser le simulateur et de tester sur son matériel
- Programmes de développement iOS
 - Programme universitaire (ne permet pas la mise en vente des applications)
 - iPhone Developer Program (99\$/an) -> mise en vente via l'App Store
 - iPhone Developer Enterprise Program (299\$/an)
 - Permet de déployer des applications internes



☐ Développer sur iOS

- ☐ Maîtriser l'utilisation des iDevices

 - ☐ Qui n'a jamais utilisé un iPhone ?

- ☐ Travailler sous OS X

- ☐ Ergonomie

 - ☐ Human Interface Guidelines

 - ☐ Le non respect peut être une cause de rejet de l'App store

 - ☐ <https://developer.apple.com/design/human-interface-guidelines/ios/overview/themes/>



□ Les spécificités du développement sur iOS

- Contrainte de taille d'écran (<https://www.paintcodeapp.com/news/ultimate-guide-to-iphone-resolutions>)
- Ressources « limitées » en CPU et mémoire
- Fonctionnement sur batterie (les calculs importants diminuent l'autonomie)
- Une seule application est exécutée à la fois
 - nécessite des temps de démarrages et d'arrêt courts
 - à tout moment l'application peut-être interrompue par un appel
 - un fonctionnement multitâche est possible depuis la version 4 d'iOS



❑ Les spécificités du développement sur iOS

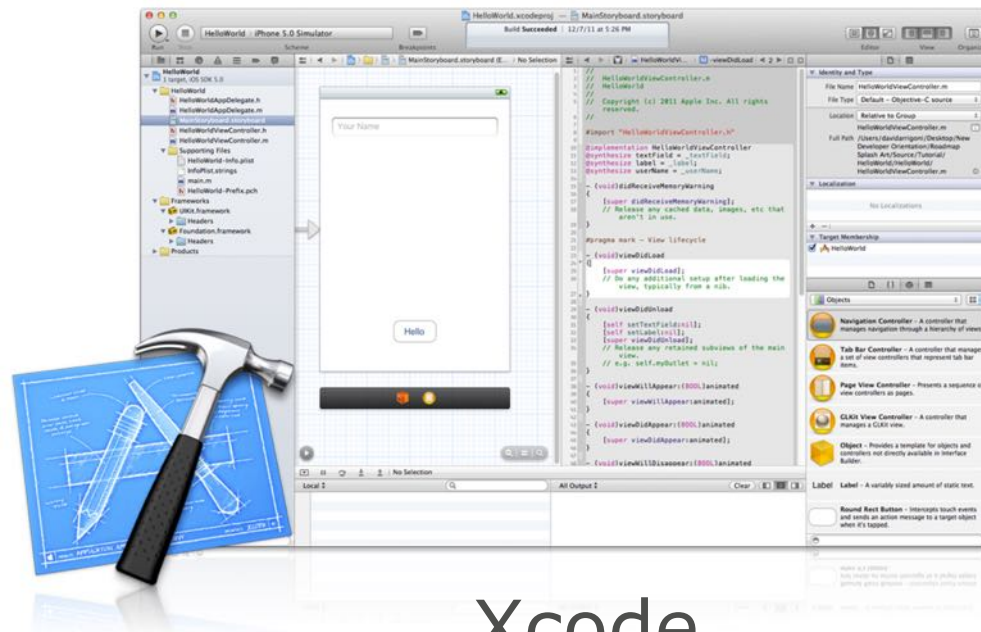
- ❑ Développement natif en Objective-C/Swift principalement (possibilité d'utiliser du C et C++)
- ❑ Prise en compte des spécificités des plateformes
 - ❑ iPad / iPhone / Apple Watch
 - ❑ Résolutions écrans différentes (Retina)
 - ❑ Plus le temps passe et plus il y a de diversités à prendre en compte
 - ❑ Il est possible de restreindre la liste des matériels sur lesquels une application pourra tourner



Développement iOS

□ Environnement de développement

Outils



Xcode

Frameworks



Foundation

Langages

```
[UIView setAnimationDidStopSelector:@selector(removeNightImage)];
```



Instruments



□ L'environnement de développement : les frameworks

- Un framework est une collection d'éléments : fichiers d'en-tête, librairies, images, sons, ... lié dynamiquement au code
- Frameworks fondamentaux
 - Foundation :
 - Définition des conventions pour la gestion de la mémoire, la mutabilité, les notifications (classe NSObject)
 - Support de la localisation de l'application (gestion des langues et des formatages propres aux pays : nombres, dates et heures,...), codage en Unicode des chaînes
 - "Object wrappers" pour les types primitifs, classes de collections, classes utilitaires pour les threads, le système de fichier, ...
 - UIKit
 - Structure d'une application (UIApplication)
 - Interface utilisateur (NSControl, NSResponder,...)

❑ Choix des versions à supporter

- ❑ Les statistiques de taux d'utilisation sont dures à trouver
 - ❑ certains développeurs donnent le pourcentage des OS utilisant leurs applications
 - ❑ biaisées car le type d'application utilisé est généralement destiné à un public technophile qui a tendance à mettre à jour vite
- ❑ des réseaux de publicité sur mobile et web mobile fournissent des statistiques
 - ❑ biaisées car souvent limitées à des régions du monde



☐ Choix des versions à supporter



https://mixpanel.com/trends/#report/ios_12

☐ Sauf si l'application développée doit toucher tous les utilisateurs (par exemple suivi de consommation opérateur téléphonique / compte bancaire / ...), cibler iOS n-1 voire iOS n si l'application cible un public plutôt technophile

☐ iOS Support Matrix : <http://iossupportmatrix.com>

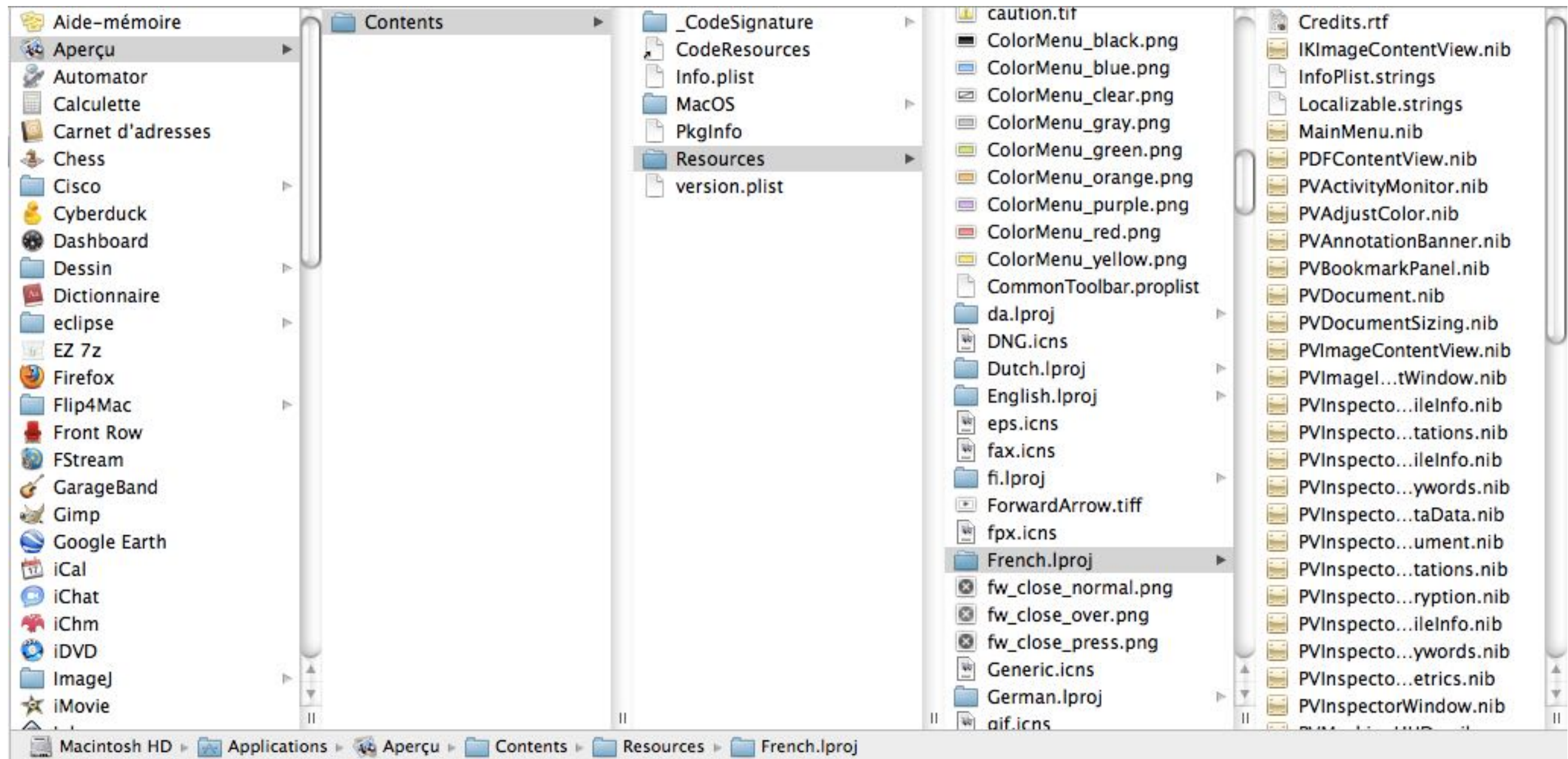


□ Anatomie d'une application iOS

- Les applications Mac et iDevice sont vues dans l'explorateur de fichier comme un seul fichier
- Ce fichier est en réalité un bundle qui contient un ensemble de fichiers avec une arborescence complète contenant :
 - Le fichier binaire avec le code de l'application
 - des meta-données sur l'application (auteur, nom du fichier d'icône, signature du code et diverses propriétés)
 - des ressources de l'application : interfaces graphiques, images, sons, fichiers d'aide, bases de données, ...



■ Anatomie d'une application (Aperçu)



Développement iOS



Objective-C

□ Le langage Objective C

- Basé sur le C ANSI
 - Toute la syntaxe du C peut-être utilisée
 - Code réparti entre :
 - Fichiers d'en-tête (.h) : déclarations publiques
 - Fichiers sources (.m) : implémentation (peuvent contenir du code en C classique et de l'Objective-C
 - Fichiers sources (.mm) : implémentation contenant du code en C++ et de l'Objective-C
- Apporte les principes de programmation orientée objet avec un minimum d'ajouts syntaxiques



□ Le langage Objective C

□ Types

□ Tous les types du C sont disponibles

□ Préférer les types redéfinis (inclus automatiquement avec Foundation.h) : SInt8, UInt8, SInt16, UInt16, SInt32, UInt32, SInt64, UInt64, Float32, Float64

□ NSInteger, NSUInteger : entiers 32 ou 64 bits selon la plateforme pour laquelle on les compile

□ Types spécifiques

□ BOOL : prend les valeurs YES ou NO

□ id : objet anonyme

□ Class : pointeur sur un objet décrivant une classe

□ SEL : sélecteur de message



□ Le langage Objective C

- Ne permet que l'héritage simple (comme Java)
 - possibilité d'implémenter plusieurs protocoles (équivalent des interfaces en Java)
- Pas de typage fort
 - le type des objets est vérifié par le runtime et non pas à la compilation
 - le compilateur émet un warning si le récepteur d'un message peut ne pas répondre à un message
- Les objets sont accessible par des variables de type pointeur (avec la syntaxe du C) :
`UIButton *unBouton;`
- Les types primitifs sont déclarés différemment :
`UInt8 age;`



□ Le langage Objective C

□ Pointeurs

<code>UInt8 u8_maVariable</code>	Déclaration d'une variable contenant un entier codé sur 8 bits
<code>UInt8 *p_monPointeur</code>	Déclaration d'une variable qui pointe sur un entier codé sur 8 bits (la variable contient une adresse mémoire)
<code>&u8_maVariable</code>	Obtention de l'adresse d'une variable
<code>*p_monPointeur</code>	Accès au contenu pointé (déréférencement du pointeur)

□ Le langage Objective C

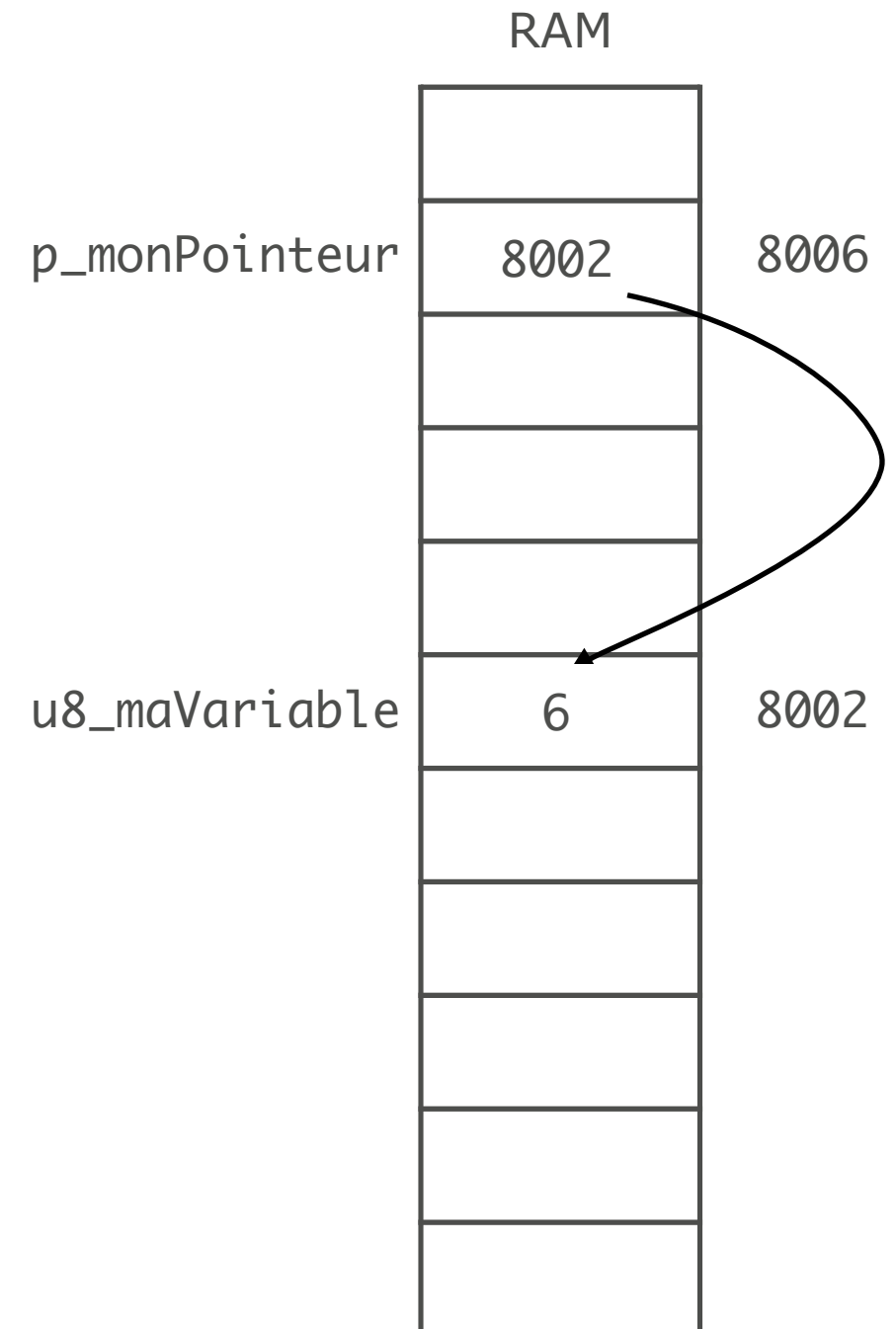
□ « Rappels » de syntaxe sur les pointeurs

➔ `UInt8 *p_monPointeur;`

`UInt8 u8_maVariable = 4;`

`p_monPointeur = &u8_maVariable;`

`*p_monPointeur = 6;`



□ Syntaxe Objective C

□ Typage statique

□ `UIButton *unBouton;`

□ La compilateur vérifie que les méthodes appelées sur un objet sont définies

□ Les variables permettant de manipuler les objets sont des pointeurs

□ Typage dynamique

□ `id unObjet;`

□ Pas d'étoile (`id` est défini comme un pointeur sur un objet dont le type est anonyme)

□ Plus général que `NSObject *unObjet`



□ Syntaxe Objective C

□ Typage dynamique

- Différence entre `id unObjet;` et `NSObject *unAutreObjet;`
 - On ne peut envoyer à `unAutreObjet` que les messages définis dans la classe `NSObject`
 - On peut envoyer à `unObjet` tous les messages possibles
 - si l'objet ne reconnaît pas le message cela provoquera une exception
 - il est possible de tester dynamiquement si un objet répond à un message donné
- Le type `id` est le format de stockage des classes de collection `NSDictionary` (tableau associatif - équivalent d'une Map Java), `NSArray` (tableau - équivalent d'une List Java) et `NSSet` (collection d'objets uniques - équivalent d'un Set Java)



□ Syntaxe Objective C - l'envoi de messages

```
[destinataire message];
```

```
[destinataire message:argument];
```

```
[destinataire message:arg1 andArg:arg2];
```

Exemples :

```
[alertTimer invalidate];
```

```
UIApplication *app = [UIApplication sharedApplication];  
[app setStatusBarHidden:YES animated:YES];
```

□ Syntaxe Objective C - l'envoi de messages

□ Equivalence avec Java

```
[alertTimer invalidate]; //Objective C
```

équivalent à :

```
alertTimer.invalidate(); //Java
```

```
[app setHidden:YES animated:YES]; //Objective C
```

équivalent à :

```
app.setHidden(true, true); //Java
```

□ Syntaxe Objective C - l'envoi de messages

□ Arguments nommés

□ Syntaxe, déroutante au premier abord, influencée par smalltalk

□ Avantages : facilite la lecture et l'écriture du code (pas besoin de se référer à la documentation pour connaître la signification des arguments)

```
(UIColor *)color = [UIColor colorWithRed:0.2 green:0.4 blue:0.8 alpha:1.0];
```

plutôt que :

```
Color color = new Color(0.2f,0.4f,0.8f,1.0f);
```

On peut envoyer un message à une classe (méthode statique)



□ Syntaxe Objective C - l'envoi de messages

□ Terminologie

□ Message expression

[receiver method: argument]

□ Message

[receiver method: argument]

□ Selector

[receiver method: argument]

```
[UIColor colorWithRed:0.2 green:0.4 blue:0.8 alpha:1.0];
```

Selector : colorWithRed:green:blue:alpha:

□ Method

Le code sélectionné par un message (dépend de la classe qui reçoit le message)



□ Syntaxe Objective C

□ Null object pointer

□ Test explicite

```
if (personne == nil) return;
```

□ Test implicite

```
if (!personne) return;
```

□ Peut être utilisé pour des affectations ou comme argument

```
personne = nil;  
[button setTarget: nil];
```

□ Message à nil ?

```
Person *motherInLaw = [[aPerson spouse] mother];
```

□ Que se passe-t-il si l'objet `aPerson` n'a pas d'épouse ?

□ La valeur de retour est `nil` ou 0 (selon la signature de la méthode appelée)



□ Syntaxe Objective C

□ Définition d'une nouvelle classe

- Nom de la classe : **Person**
- Détermination de la super-classe : **NSObject**
- Choix des variables d'instances : **nom, age**
- Choix des actions à réaliser : **voter**



□ Syntaxe Objective C

□ #import vs #include

- Préférer l'utilisation de `#import` qui permet d'assurer qu'un fichier n'est inclus qu'une seule fois

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

...

MonFichier.h

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

...

Toto.h



□ Syntaxe Objective C

- Définition d'une nouvelle classe

- Une interface publique et une implémentation privée



Fichier d'en-tête
Header file



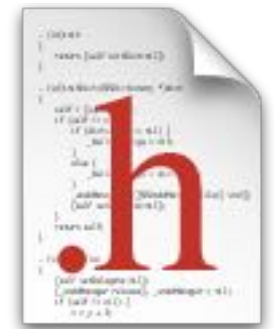
Fichier source
Implementation file



□ Syntaxe Objective C

@interface Person

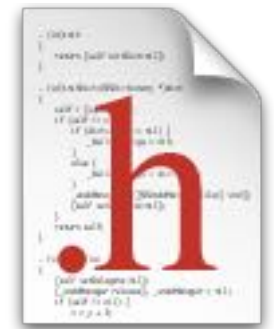
@end



Person.h

□ Syntaxe Objective C

```
@interface Person : NSObject
```



Person.h

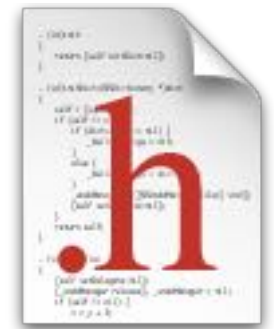
super-classe

```
@end
```

□ Syntaxe Objective C

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

```
@interface Person : NSObject
```



Person.h

```
@end
```

❑ Syntaxe Objective C

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

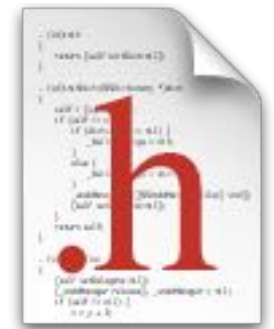
```
@interface Person : NSObject
```

```
// Variables d'instance
```

```
@property(nonatomic) NSString *name;
```

```
@property(nonatomic) UInt8 age;
```

```
@end
```



Person.h

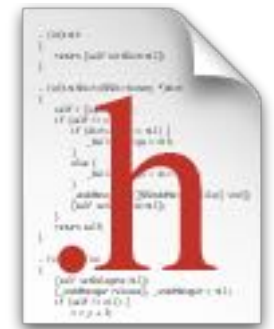
□ Syntaxe Objective C

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

@interface Person : NSObject

// Variables d'instance
@property(n nonatomic) NSString *name;
@property(n nonatomic) UInt8 age;

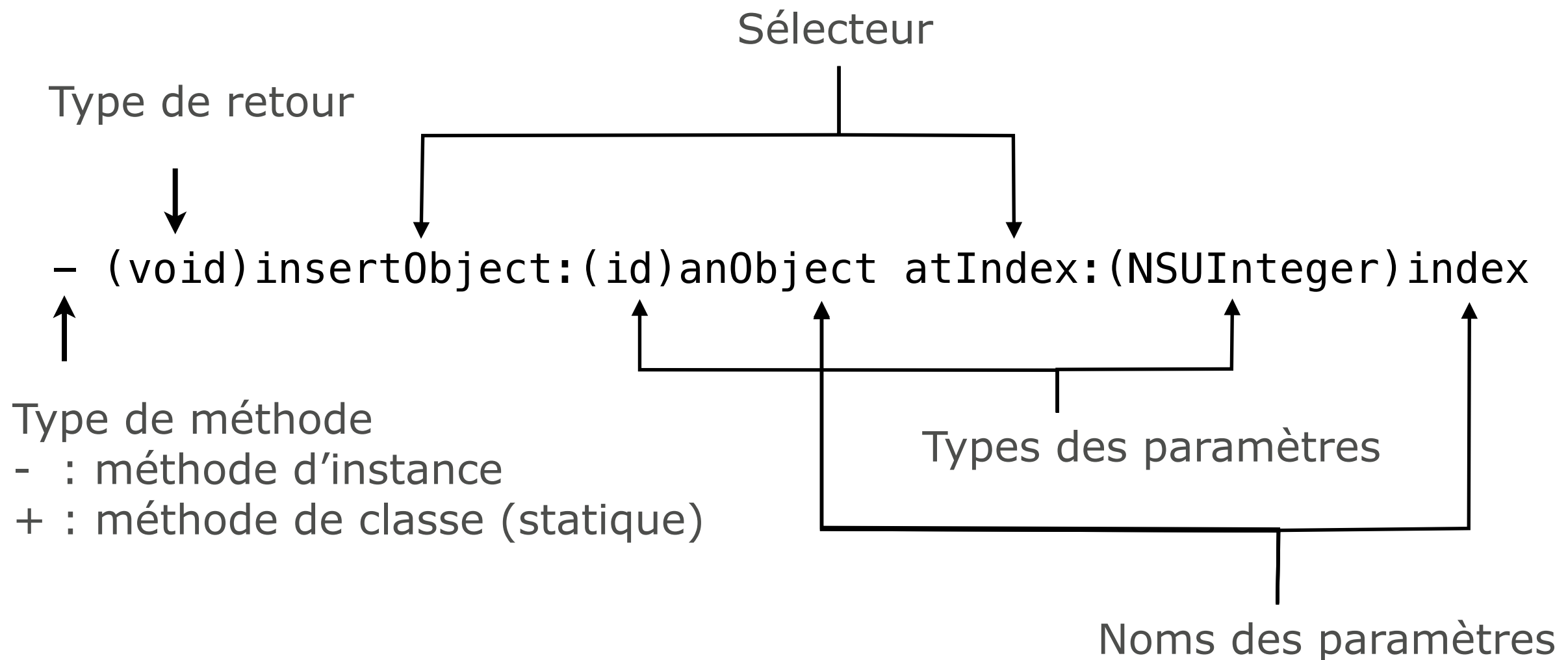
// Déclarations des méthodes
- (BOOL)canLegallyVote;
- (void)vote;
@end
```



Person.h

□ Syntaxe Objective C

□ Déclaration de méthode

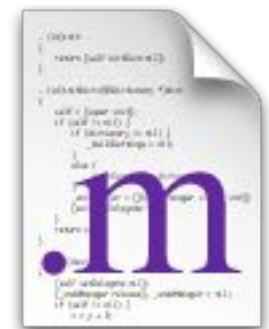


□ Syntaxe Objective C

```
#import "Person.h"
```

```
@implementation Person
```

```
@end
```



Person.m

□ Syntaxe Objective C

```
#import "Person.h"
```

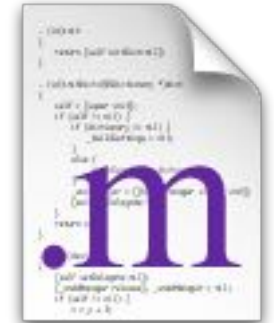
```
@implementation Person
```

```
- (BOOL)canLegallyVote {  
}
```

```
- (void)vote {
```

```
}
```

```
@end
```



Person.m

❑ Syntaxe Objective C

```
#import "Person.h"
```

```
@implementation Person
```

```
- (BOOL)canLegallyVote {  
    return ([self age] >= 18);  
}
```

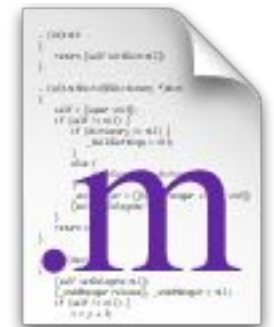
```
- (void)vote {  
    if ([self canLegallyVote]) {  
        // faire tout ce qu'il faut pour voter  
    } else {  
        NSLog(@"Je n'ai pas le droit de voter!");  
    }  
}
```

```
@end
```

Sortie console

Appel des méthodes du même objet

Constante littérale de type NSString



Person.m

□ Syntaxe Objective C

- La variable `self` désigne l'objet courant (équivalent de `this` en Java)
- Il est également possible d'appeler une méthode définie dans une super-classe en utilisant le mot clé `super`

```
- (void)faireQuelqueChose {  
    // Appel de l'implémentation de la super-classe d'abord  
    [super faireQuelqueChose];  
    //Faire le reste  
}
```



□ Syntaxe Objective C

□ Création d'un objet

- En deux étapes : allocation et initialisation

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

- Toujours utiliser ce motif car la méthode `init` retourne un objet

- Cet objet peut être d'une instance d'une sous-classe de la classe d'interface

- Erreur à ne pas commettre :

```
NSString *maChaine = [NSString alloc];  
[maChaine initWithString:@"%plouf !"];
```

la chaîne initialisée est perdue car on ne stocke pas sa référence

- Façon correcte de procéder

```
NSString *maChaine = [[NSString alloc] initWithString:@"%plouf !"];
```



■ Syntaxe Objective C

■ Création d'un objet

■ `init` ou `initWithString` ?

■ Une classe peut proposer plusieurs méthodes d'initialisation (analogie avec plusieurs constructeurs)

■ le nom de méthode commence par `init` et le type de retour est `instancetype`

■ Chaque classe possède un "designated initializer"

■ `NSObject : -(instancetype)init`

■ `UIView : -(instancetype)initWithFrame:(CGRect)aFrame`

■ `UIViewController :`
`-(instancetype)initWithNibName:(NSString *)nibName`
`bundle:(NSBundle *)nibBundle`

■ Tous les autres initialiseurs appellent le "designated initializer" qui est généralement celui qui possède le plus d'arguments

■ Syntaxe Objective C

■ Création d'un objet

■ Ajout d'un constructeur à la classe Person

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

```
@interface Person : NSObject
```

```
// Variables d'instance
```

```
@property(nonatomic) NSString *name;
```

```
@property(nonatomic) UInt8 age;
```

```
// Déclarations des méthodes
```

```
- (instancetype)initWithName:(NSString *)name age:(UInt8)age;
```

```
- (BOOL)canLegallyVote;
```

```
- (void)vote;
```

```
@end
```



Person.h

■ Syntaxe Objective C

■ Création d'un objet

■ Ajout d'un constructeur à la classe Person

```
#import "Person.h"

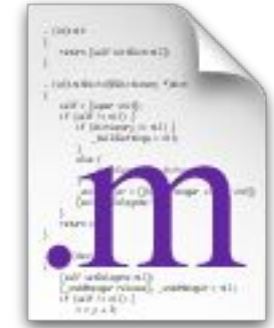
@implementation Person

- (instancetype)initWithName:(NSString *)name age:(UInt8)age
{
    self = [super init];
    if (self) {
        _age = age;
        _name = [name copy];
    }
    return self;
}

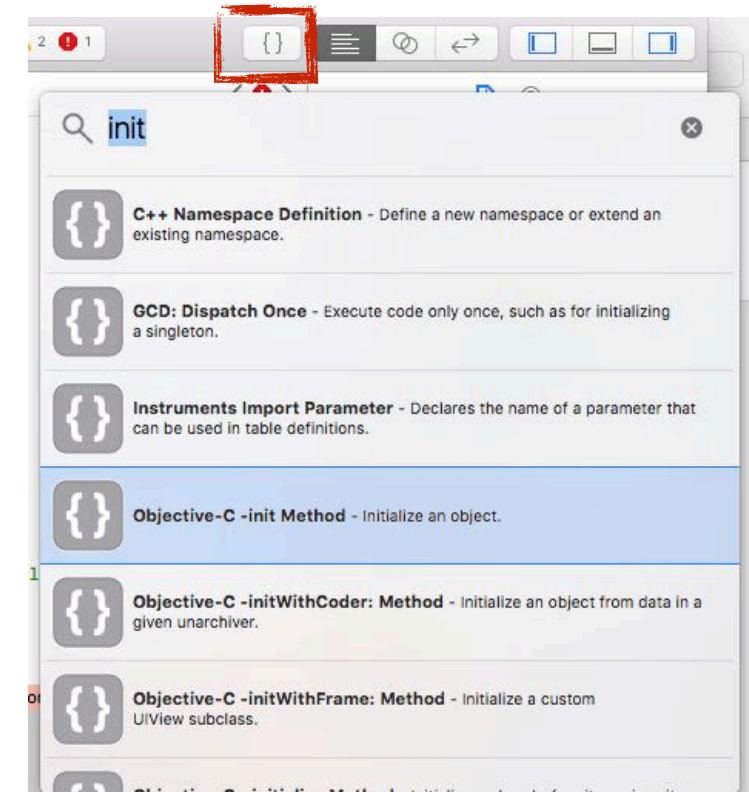
- (BOOL)canLegallyVote {
    return [self age] >= 18;
}

- (void)vote {
    if ([self canLegallyVote]) {
        // faire tout ce qu'il faut pour voter
    } else {
        NSLog(@"Je n'ai pas le droit de voter!");
    }
}

@end
```



Person.m



Utiliser la bibliothèque de snippets pour la méthode init