Lyon 1

Développement iOS

LP IEM

Développement iOS

- Développement mobile
  - Marché de l'emploi
    - Développement d'agences mobiles sur le modèle des agences web auxquelles les grands groupes sous-traitent le développement et la mise à jour de leur app
    - Développeurs indépendants
    - Développeurs "maison" pour les très grands groupes qui ont une forte présence sur l'appstore ou dont le corps de métier est le développement logiciel (ex : Lagardère Active Digital 4 développeurs iPhone, 1 développeur Android, les autres plate-formes sont sous-traitées)

Développement iOS

- Développement mobile
  - Un marché très récent
    - Sortie du premier iPhone en 2007

BlackBerry App World

Android Market
(Google play)

Nokia Ovi store

App Store

Windows market place

- Stratégie mobile pour les entreprises
  données 07/2011
  - 1 - Développement d'une web app
    - Permet de ne pas exclure de terminaux
    - Permet de faire des statistiques sur les terminaux de ses clients
  - 2 - Développement d'une application iPhone
  - 3 - Développement d'une application Android
  - Nécessité de connaitre le marché
    - Les activations plus nombreuses de terminaux Android ne se traduisent pas encore en termes d'usage (en Angleterre : 28 % d'iPhone mais 61 % des usages)
    - Problème d'Android (fragmentation : versions logicielles / matérielles)

- Coût d'une application
  données 07/2011
  - Ratios d'investissement pour le développement
    - Base 100 iOS ➪ 120 à 130 Android, 140 RIM, 140 à 150 Windows Phone
  - Application iOS de consultation de contenu ≈ 5 à 10 k€ (dépend de la complexité)
  - Applications iOS sécurisées et transactionnelles > 20 k€
    - Ex : groupe Accor (réservation hotel) : coût total 50 à 60 k€ (avec les mise à jour)
  - Jeux : 2 M$ pour Real Racing 2 !

- Développer sur iOS
  - IDE sur MAC OS X uniquement (Lion et Mountain Lion)
  - Créer un compte Apple Developper
    - Le compte donne accès au SDK et permet d'utiliser le simulateur pour la mise au point
  - Pour tester une application sur le iPhone/iPod/iPad il faut faire partie d'une équipe de développement
    - Etre rattaché à un programme universitaire comme étudiant (ne permet pas la mise en vente des applications)
    - Etre rattaché à un compte iPhone Developer Program (99$/an)
    - Etre rattaché à un compte iPhone Developer Enterprise Program (299$/an)
      - Permet de déployer des applications internes

- Développer sur iOS
  - Maitriser l'utilisation des iDevices
    - Qui n'a jamais utilisé un iPhone ?
  - Travailler sous Mac OS X (depuis Lion, il est officiellement possible de le virtualiser pour l'utiliser depuis un PC avec une machine virtuelle)
    - Qui n'a jamais utilisé Mac OS X ?

- Les spécificités du développement sur iOS
  - Contrainte de taille d'écran 320 x 480 (640 x 960 pour l'iPhone 4/4S, 1 136 x 640 iPhone 5, 5S et 5C, 1024 x 768 pour les iPads classiques, 2048 x 1536 pour les iPads retina, 1334 x 750 pour l'iPhone 6 et 1920 x 1080 pour l'iPhone 6 plus)
  - 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 interrompu par un appel
    - un fonctionnement multitâche est possible à partir de la version 4 d'iOS

- Les spécificités du développement sur iOS
  - Développement en Objective-C principalement (possibilité d'utiliser du C et C++) et maintenant en swift
  - Prise en compte des spécificités des plateformes
    - Absence d'appareil photo et de puce GPS sur certains devices
    - Ecran de taille différentes sur l'iPad et iPhone 5/5S/5C/6/6+ avec de nombreux frameworks supplémentaires
    - 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

- Environnement de développement

Outils

Xcode

Instruments

Frameworks

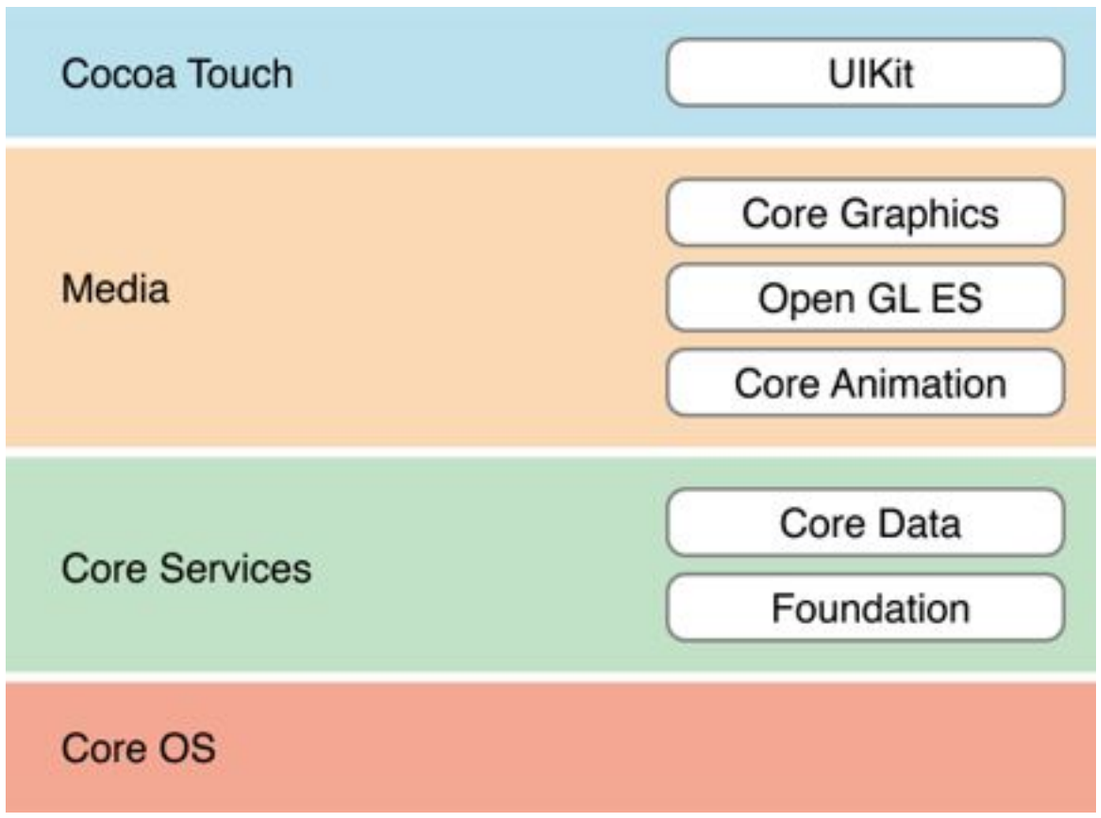
Foundation

Langages

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

- 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,...)

- L’environnement de développement : les frameworks
  - Architecture d'iOS
    - Chaque couche propose un grand nombre de frameworks
    - Documentation de référence pour s'orienter :
      - *iOS Technology Overview*



- Versions iOS - principales fonctionnalités pour le développeur
  - iPhone OS 2 : premier SDK public
  - iPhone OS 3 : Core Data (framework de persistence), Notifications Push, In App Purchase, In App email, In App map service (MapKit basé sur Google Maps), accès à la bibliothèque iPod, éditeur vidéo, Core Text, gesture recognizers, partage de fichiers, génération de PDF
  - iOS 4 : multitâche, notifications locales, accès au calendrier, programmation par blocks, Grand Central Dispatch, In App SMS composer
  - iOS 5 : iCloud, Storyboard, framework Twitter, Customisation des composants d'interface graphique, Core Image (traitement de l'image), ...
    - Sortie simultanée d'un nouveau compilateur (LLVM)et d'une mise à jour majeure de l'IDE XCode (version 4) avec une gestion automatisée de la mémoire (ARC)

- Versions iOS - principales fonctionnalités pour le développeur
  - iOS 6 : framework Facebook, Pass Kit (billets électroniques), MapKit (abandon de google maps), accès aux rappels du téléphone depuis une application, nouveau gestionnaire de mise en page des vues, ajouts de nouveaux contrôleurs de vue
    - Peu de changements majeurs mais beaucoup de petits ajouts de fonctionnalités dans de nombreux frameworks
  - iOS 7 : changement majeur de l'interface
  - iOS 8 : app extensions, Touch ID, …

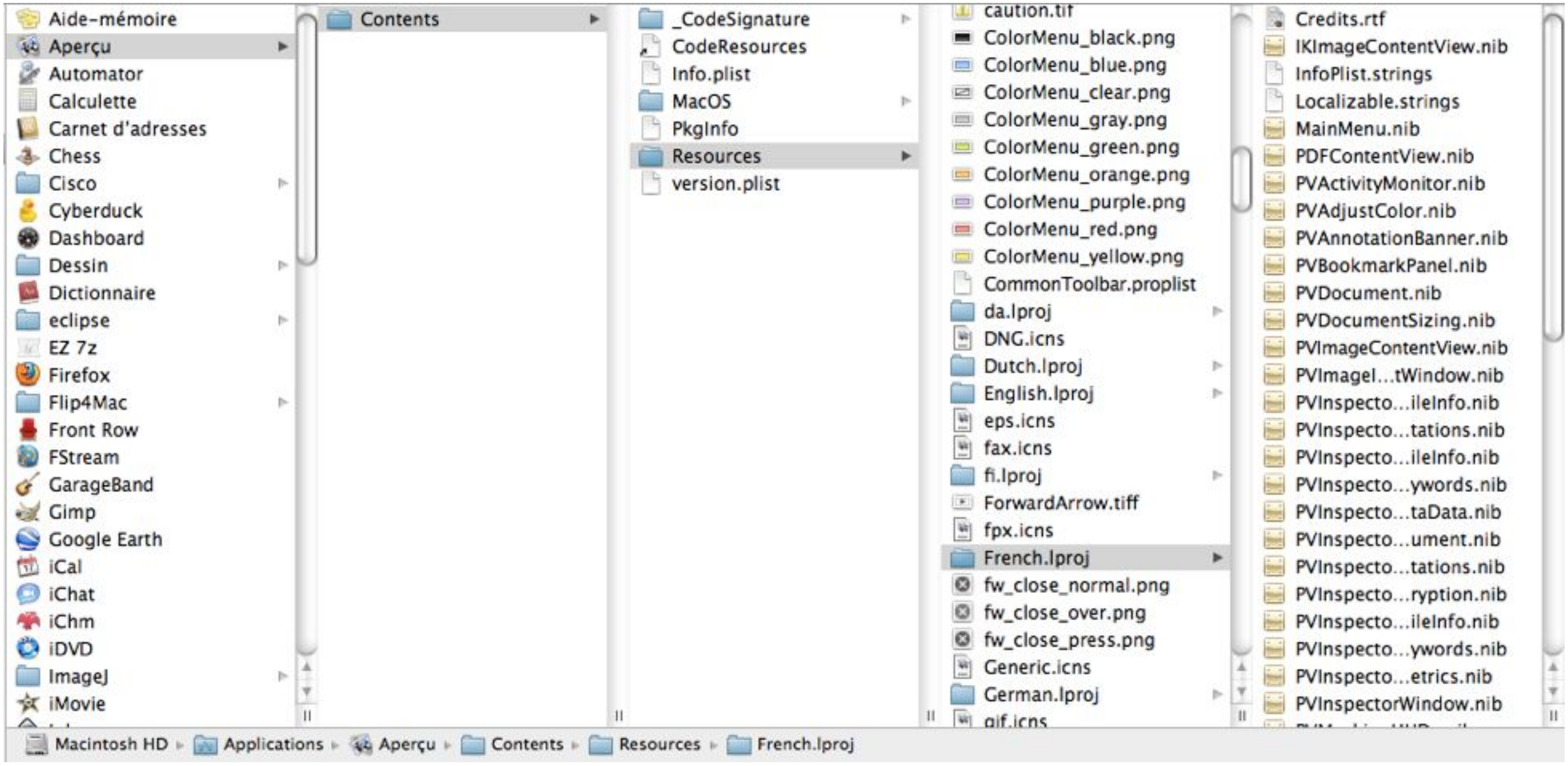
- Choix des versions à supporter
  - Les statistiques de taux d'utilisation sont dures à trouver
    - certains développeurs donnent le pourcentage des OS utilisant leurs applications
      - biaisées car le type d'application utilisé est généralement destiné à un public technophile qui a tendance à 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
  - Taux d'adoption pour iOS 4 : 50% un mois après la sortie, 70% 3 mois après et 90 % 6 mois après
  - iOS 5 est disponible sur les iPhones > 3G (2008)
  - iOS 8 (60 %), iOS 7 (35%)
  - 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

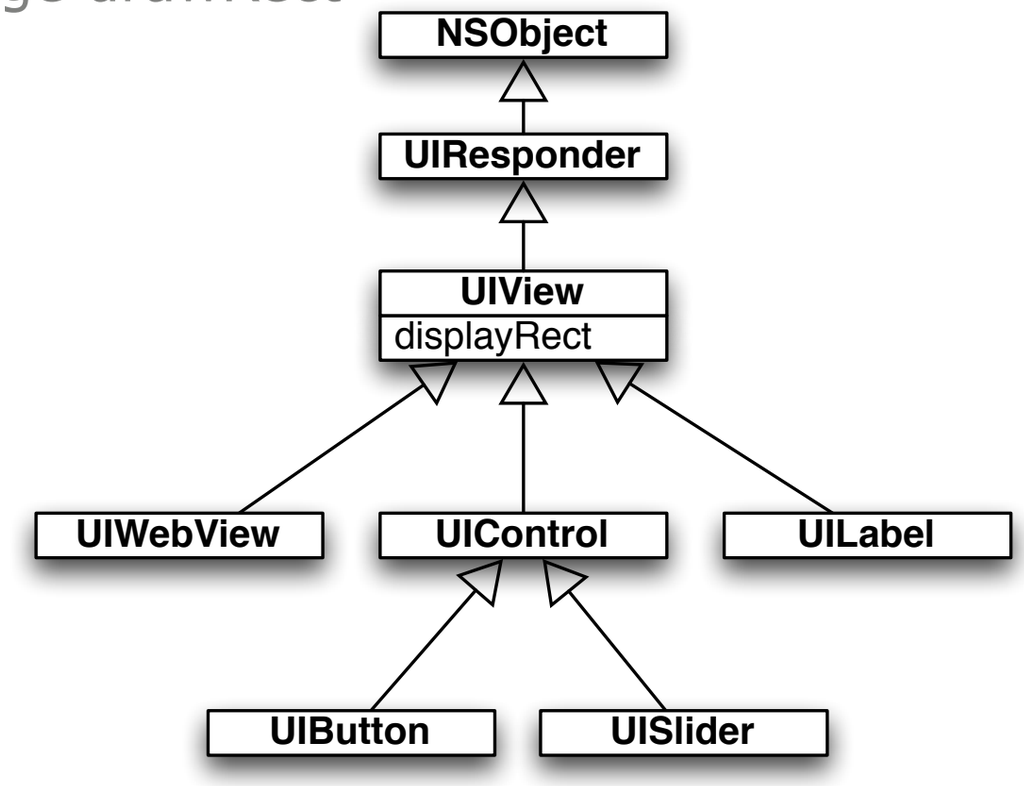
- Les applications Mac et iDevice sont vues dans l'explorateur de fichier comme un seul fichier
- Ce fichier est en réalité un bundle qui contient un ensemble de fichiers avec une arborescence complète contenant :
  - Le fichier binaire avec le code de l'application
  - des meta-données sur l'application (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)



- 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

- Programmation Orientée Objet
  - Polymorphisme
    - Possibilité de manipuler plusieurs instances de sous-classes différentes à travers l'interface définit par la super-classe commune
    - Tous les éléments d'interface graphiques (UILabel, UISlider, UITextView, UIButton,...) peuvent être manipulés en considérant qu'ils sont de type UIView ➪ ils comprennent le message drawRect



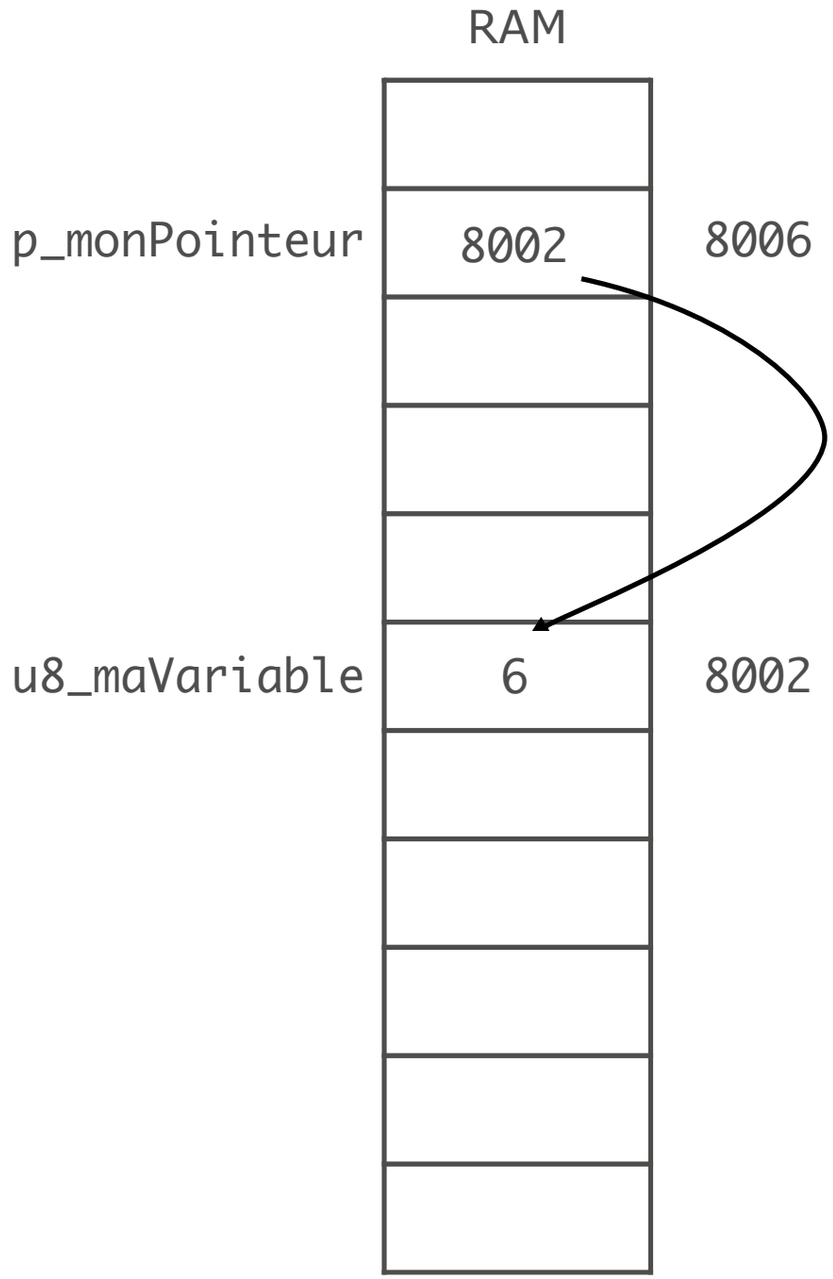
- Le langage Objective C
  - Types
    - Tous les types du C sont disponibles
      - Préférer les types redéfinis (inclus automatiquement avec Foundation.h) : SInt8, UInt8, SInt16, UInt16, SInt32, UInt32, SInt64, UInt64, Float32, Float64
      - NSInteger, NSUInteger : entiers 32 ou 64 bits selon la 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

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
      - Dynamic Binding : le code effectivement appelé est résolu par le runtime à l'exécution

- 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 reconnait 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
```

équivaut à :

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

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

équivaut à :

```
app.setStatusBarHidden(YES,YES); //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 connaitre 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, âge
    - 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

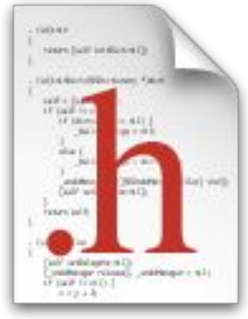
```
#import <Foundation/Foundation.h>
#import “Toto.h”

...
```

MonFichier.h

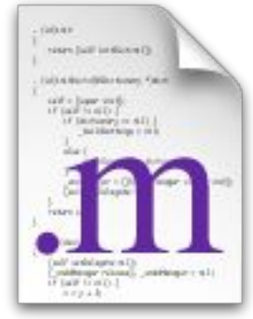
```
#import <Foundation/Foundation.h>

...
```

Toto.h

- Syntaxe Objective C
  - Définition d’une nouvelle classe
    - Une interface publique et une implémentation privée



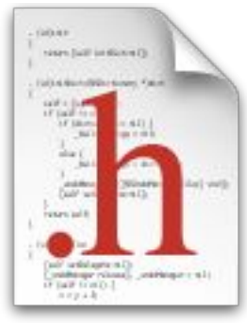
Fichier d’en-tête
Header file



Fichier source
Implementation file

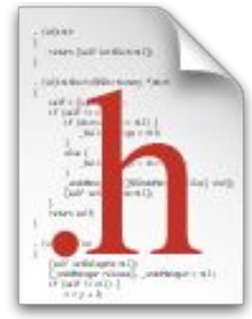
Syntaxe Objective C

```
@interface Person

@end
```



Person.h

Syntaxe Objective C



Person.h

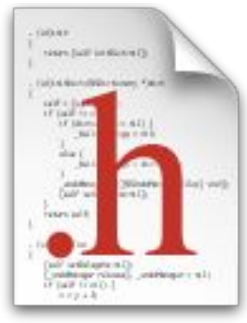
```
@interface Person : NSObject
```

super-classe

```
@end
```

Syntaxe Objective C

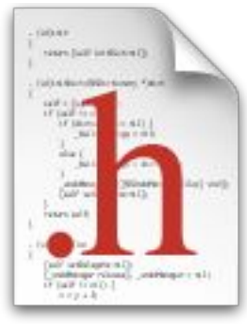
```
#import <Foundation/Foundation.h>

@interface Person : NSObject

@end
```



Person.h

Syntaxe Objective C

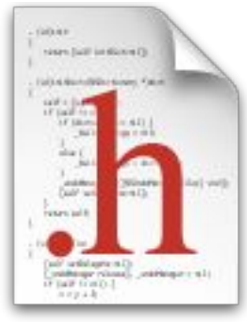
```
#import <Foundation/Foundation.h>

@interface Person : NSObject {
  // Variables d'instance
  NSString *name;
  UInt8 age;
}




@end
```



Person.h

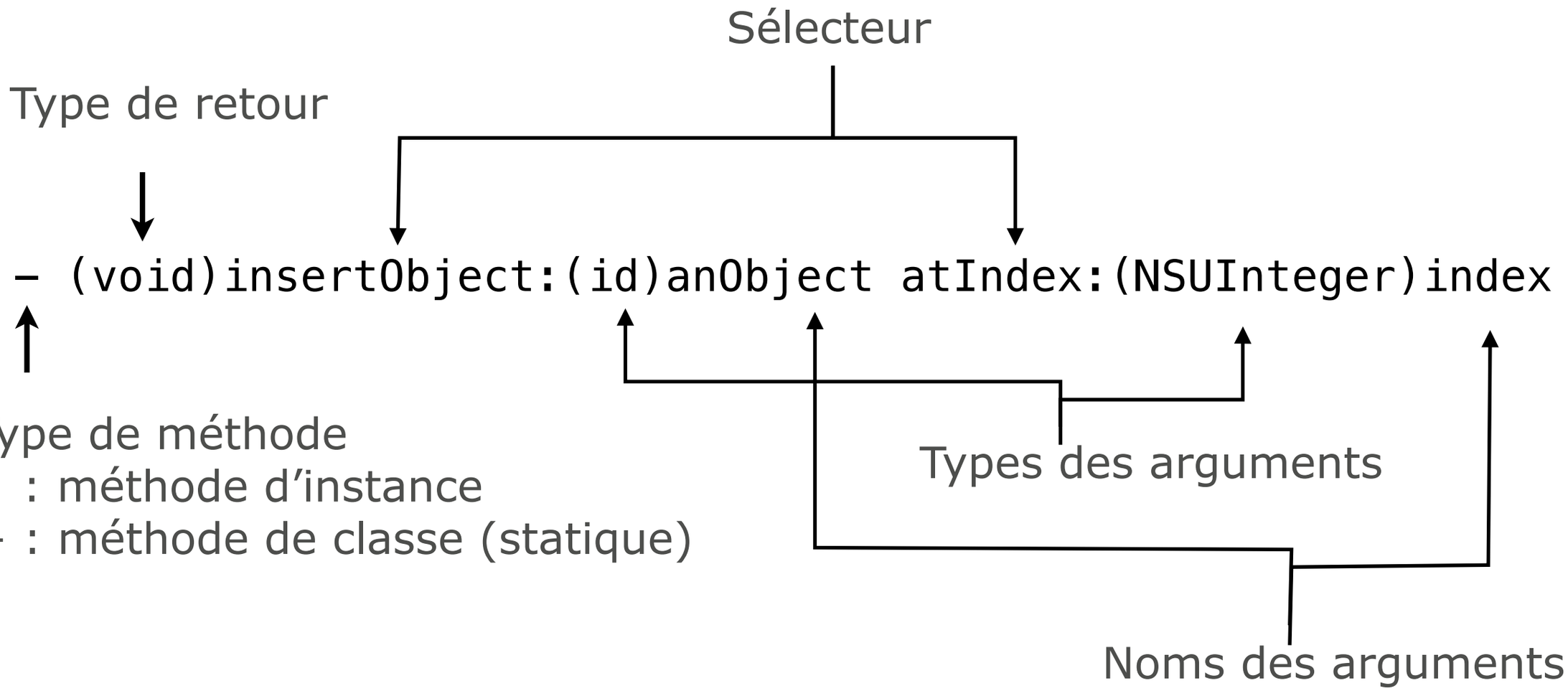
Syntaxe Objective C

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

@interface Person : NSObject {
  // Variables d'instance
  NSString *name;
  UInt8 age;
}

// Déclarations des méthodes
- (NSString *)name;
- (void)setName:(NSString *)value;
- (UInt8)age;
- (void)setAge:(UInt8)age;
- (BOOL)canLegallyVote;
- (void)vote;
@end
```



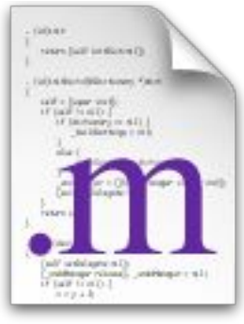
Person.h

- Syntaxe Objective C
  - Déclaration de méthode



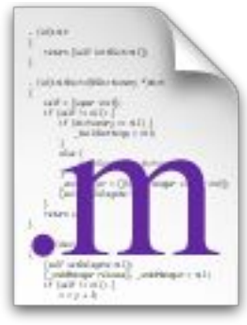
Syntaxe Objective C

```
#import "Person.h"

@implementation Person

@end
```



Person.m

Syntaxe Objective C

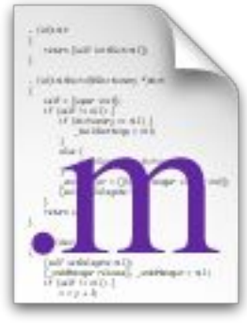


Person.m

```
#import "Person.h"

@implementation Person

- (UInt8)age {
  return age;
}

- (void)setAge:(UInt8)value {
  age = value;
}
//... autres méthodes
@end
```

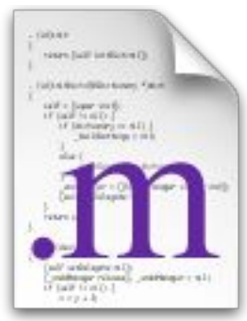
Syntaxe Objective C

```
#import "Person.h"

@implementation Person

- (BOOL)canLegallyVote {
}

- (void)vote {



}

@end
```



Person.m

Syntaxe Objective C



Person.m

```
#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
```

Appel des méthodes du même objet

Constante littérale de type NSString

Sortie console

- 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 "id"
  - Chaque classe possède un "designated initializer"
    - NSObject : -(id)init
    - NSView : -(id)initWithFrame:(NSRect)aFrame
    - UIViewController :
      - (id)initWithNibName:(NSString *)nibName bundle:(NSBundle *)nibBundle
  - Tous les autres initialiseurs appellent le "designated initializer" qui est généralement celui qui possède le plus d'arguments