eBook Gratuit

APPRENEZ

Objective-C Language

eBook gratuit non affilié créé à partir des **contributeurs de Stack Overflow.**

#objective-c

Table des matières

À propos

You can share this PDF with anyone you feel could benefit from it, downloaded the latest version from: objective-c-language

It is an unofficial and free Objective-C Language ebook created for educational purposes. All the content is extracted from Stack Overflow Documentation, which is written by many hardworking individuals at Stack Overflow. It is neither affiliated with Stack Overflow nor official Objective-C Language.

Use the content presented in this book at your own risk; it is not guaranteed to be correct nor accurate, please send your feedback and corrections to info@zzzprojects.com

Chapitre 1: Premiers pas avec le langage Objective-C

Versions

Version	Date de sortie
1.0	1983-01-01
2.0	2007-10-27
Moderne	2014-03-10

Examples

Bonjour le monde

Ce programme va afficher "Hello World!"

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

int main(int argc, char * argv[]) {
    NSLog(@"Hello World!");
}
```

`#import` est une directive de préprocesseur, qui indique que nous voulons *importer* ou inclure les informations de ce fichier dans le programme. Dans ce cas, le compilateur copiera le contenu de `Foundation.h` dans le framework `Foundation` en haut du fichier. La principale différence entre #import et #include est que #import est suffisamment "intelligent" pour ne pas retraiter les fichiers qui ont déjà été inclus dans d'autres #includes.

La documentation du langage C explique la fonction `main` .

La fonction `NSLog()` imprimera la chaîne fournie à la console, ainsi que certaines informations de débogage. Dans ce cas, nous utilisons un littéral de chaîne Objective-C: `@"Hello World!"` . En C, vous écrivez ceci comme `"Hello World!"` Cependant, le Framework de base d'Apple ajoute la classe `NSString` , qui fournit de nombreuses fonctionnalités utiles et est utilisée par NSLog. La façon la plus simple de créer une instance de `NSString` est la suivante: `@" string content here "` .

> Techniquement, NSLog () fait partie du framework Foundation d'Apple et ne fait pas réellement partie du langage Objective-C. Cependant, le cadre de la Fondation est omniprésent dans la programmation d'Objective-C. Étant donné que Foundation Framework n'est pas open-source et ne peut pas être utilisé en dehors du développement d'Apple, il existe des alternatives open-source au framework associées à OPENStep et GNUStep .

Compiler le programme

En supposant que nous voulions compiler notre programme Hello World, constitué d'un seul fichier `hello.m` , la commande pour compiler l'exécutable est la suivante:

```
clang -framework Foundation hello.m -o hello
```

Ensuite, vous pouvez l'exécuter:

```
./hello
```

Cela va sortir:

```
Hello World!
```

Les options sont les suivantes:

- `-framework` : Spécifie un framework à utiliser pour compiler le programme. Comme ce programme utilise Foundation, nous incluons le framework Foundation.
- `-o` : Cette option indique à quel fichier nous aimerions sortir notre programme. Dans notre cas `hello` . S'il n'est pas spécifié, la valeur par défaut est `a.out` .

Lire Premiers pas avec le langage Objective-C en ligne: https://riptutorial.com/fr/objective-c/topic/199/premiers-pas-avec-le-langage-objective-c

Chapitre 2: Analyse XML

Examples

Analyse XML

```
<?xml version="1.0" encoding="UTF-8"?>
<GeocodeResponse>
 <status>OK</status>
 <result>
  <type>premise</type>
  <formatted_address>4201 Oak Lawn Ave, Dallas, TX 75219, USA</
      formatted_address>
  <address_component>
   <long_name>4201</long_name>
   <short_name>4201</short_name>
   <type>street_number</type>
  </address_component>
  <address_component>
   <long_name>Oak Lawn Avenue</long_name>
   <short_name>Oak Lawn Ave</short_name>
   <type>route</type>
  </address_component>
  <address_component>
   <long_name>Oak Lawn</long_name>
   <short_name>Oak Lawn</short_name>
   <type>neighborhood</type>
   <type>political</type>
  </address_component>
  <address_component>
   <long_name>Dallas</long_name>
   <short_name>Dallas</short_name>
   <type>locality</type>
   <type>political</type>
  </address_component>
 </result>
</GeocodeResponse>
```

Nous allons analyser les données de balises mises en évidence via `NSXMLParser`

Nous avons déclaré peu de propriétés comme suit

```
@property(nonatomic, strong)NSMutableArray *results;
@property(nonatomic, strong)NSMutableString *parsedString;
@property(nonatomic, strong)NSXMLParser *xmlParser;

//Fetch xml data
NSURLSession *session=[NSURLSession sessionWithConfiguration:[NSURLSessionConfiguration
```

```
defaultSessionConfiguration]];

NSURLSessionDataTask *task=[session dataTaskWithRequest:[NSURLRequest requestWithURL:[NSURL
URLWithString:YOUR_XMLURL]] completionHandler:^(NSData * _Nullable data, NSURLResponse *
_Nullable response, NSError * _Nullable error) {

self.xmlParser=[[NSXMLParser alloc] initWithData:data];
self.xmlParser.delegate=self;
if([self.xmlParser parse]){
   //If parsing completed successfully

        NSLog(@"%@",self.results);

}

}];

[task resume];
```

Ensuite, nous définissons le `NSXMLParserDelegate`

```
- (void)parser:(NSXMLParser *)parser didStartElement:(NSString *)elementName
namespaceURI:(nullable NSString *)namespaceURI qualifiedName:(nullable NSString *)qName
attributes:(NSDictionary<NSString *, NSString *> *)attributeDict{

    if([elementName isEqualToString:@"GeocodeResponse"]){
        self.results=[[NSMutableArray alloc] init];
    }

    if([elementName isEqualToString:@"formatted_address"]){
        self.parsedString=[[NSMutableString alloc] init];
    }

}


- (void)parser:(NSXMLParser *)parser foundCharacters:(NSString *)string{

    if(self.parsedString){
        [self.parsedString appendString:[string
stringByTrimmingCharactersInSet:[NSCharacterSet whitespaceAndNewlineCharacterSet]]];
    }


}

- (void)parser:(NSXMLParser *)parser didEndElement:(NSString *)elementName
namespaceURI:(nullable NSString *)namespaceURI qualifiedName:(nullable NSString *)qName{

    if([elementName isEqualToString:@"formatted_address"]){
        [self.results addObject:self.parsedString];

        self.parsedString=nil;
    }

}
```

Lire Analyse XML en ligne: https://riptutorial.com/fr/objective-c/topic/8048/analyse-xml

Chapitre 3: Blocs

Syntaxe

- // déclare comme variable locale:

  returnType (^ blockName) (parameterType1, parameterType2, ...) = ^ returnType (argument1, argument2, ...) {...};

- // déclare en tant que propriété:

  @property (nonatomique, copie, nullité) returnType (^ blockName) (parameterTypes);

- // Déclarez comme paramètre de méthode:

  - (void) someMethodThatTakesABlock: (returnType (^ nullité) (parameterTypes)) blockName;

- // Déclarer comme argument à un appel de méthode:

  [someObject someMethodThatTakesABlock: ^ returnType (paramètres) {...}];

- // déclare en tant que typedef:

  typedef returnType (^ TypeName) (parameterTypes);

  TypeName blockName = ^ returnType (paramètres) {...};

- // Déclarer une fonction C renvoyer un objet bloc:

  BLOCK_RETURN_TYPE (^ nom_fonction (paramètres de fonction)) (BLOCK_PARAMETER_TYPE);

Remarques

Les blocs sont spécifiés par la spécification de langage pour les blocs C, Objective-C, C ++ et Objective-C ++.

En outre, l'ABI des blocs est défini par la spécification d'implémentation de bloc .

Examples

Blocs en tant que paramètres de méthode

```
- (void)methodWithBlock:(returnType (^)(paramType1, paramType2, ...))name;
```

Définir et assigner

Un bloc qui effectue l'ajout de deux nombres à double précision, affectés à l' `addition` variables:

```
double (^addition)(double, double) = ^double(double first, double second){
    return first + second;
};
```

Le bloc peut ensuite être appelé comme ceci:

```
double result = addition(1.0, 2.0); // result == 3.0
```

Blocs en tant que propriétés

```
@interface MyObject : MySuperclass

@property (copy) void (^blockProperty)(NSString *string);

@end
```

Lors de l'attribution, puisque `self` conserve `blockProperty` , block ne doit pas contenir de référence forte à self. Ces références fortes mutuelles sont appelées "cycle de rétention" et empêchent la libération de l'un ou l'autre des objets.

```
__weak __typeof(self) weakSelf = self;
self.blockProperty = ^(NSString *string) {
    // refer only to weakSelf here.  self will cause a retain cycle
};
```

C'est très improbable, mais `self` peut être désalloué à l'intérieur du bloc, quelque part pendant l'exécution. Dans ce cas, `weakSelf` devient `nil` et tous les messages ne présentent aucun effet souhaité. Cela pourrait laisser l'application dans un état inconnu. Cela peut être évité en conservant `weakSelf` un `__strong weakSelf` avec un ivar `__strong` lors de l'exécution du bloc et en `__strong` par la suite.

```
__weak __typeof(self) weakSelf = self;
self.blockProperty = ^(NSString *string) {
    __strong __typeof(weakSelf) strongSelf = weakSelf;
    // refer only to strongSelf here.
    // ...
    // At the end of execution, clean up the reference
    strongSelf = nil;
};
```

Bloc Typedefs

```
typedef double (^Operation)(double first, double second);
```

Si vous déclarez un type de bloc en tant que typedef, vous pouvez utiliser le nouveau nom de type

au lieu de la description complète des arguments et des valeurs de retour. Cela définit `Operation` comme un bloc qui prend deux doubles et retourne un double.

Le type peut être utilisé pour le paramètre d'une méthode:

```
- (double)doWithOperation:(Operation)operation
                    first:(double)first
                   second:(double)second;
```

ou comme type de variable:

```
Operation addition = ^double(double first, double second){
    return first + second;
};

// Returns 3.0
[self doWithOperation:addition
                first:1.0
               second:2.0];
```

Sans le typedef, c'est beaucoup plus compliqué:

```
- (double)doWithOperation:(double (^)(double, double))operation
                    first:(double)first
                   second:(double)second;

double (^addition)(double, double) = // ...
```

Blocs en tant que variables locales

```
returnType (^blockName)(parameterType1, parameterType2, ...) = ^returnType(argument1,
argument2, ...) {...};

float (^square)(float) = ^(float x) {return x*x;};

square(5); // resolves to 25
square(-7); // resolves to 49
```

Voici un exemple sans retour et sans paramètres:

```
NSMutableDictionary *localStatus;
void (^logStatus)() = ^(void){ [MYUniversalLogger logCurrentStatus:localStatus]};

// Insert some code to add useful status information
// to localStatus dictionary

logStatus(); // this will call the block with the current localStatus
```

Lire Blocs en ligne: https://riptutorial.com/fr/objective-c/topic/540/blocs

Chapitre 4: BOOL / bool / Boolean / NSCFBoolean

Examples

BOOL / Boolean / bool / NSCFBoolean

1. bool est un type de données défini dans C99.
2. Les valeurs booléennes sont utilisées dans les conditions, par exemple les instructions if ou while, pour effectuer une logique ou une répétition de manière conditionnelle. Lors de l'évaluation d'une instruction conditionnelle, la valeur 0 est considérée comme «fausse», tandis que toute autre valeur est considérée comme «vraie». Étant donné que NULL et nil sont définis comme 0, les instructions conditionnelles sur ces valeurs inexistantes sont également évaluées comme «false».
3. BOOL est un type Objective-C défini en tant que char signé avec les macros YES et NO pour représenter true et false

De la définition dans objc.h:

```
#if (TARGET_OS_IPHONE && __LP64__)  ||  TARGET_OS_WATCH
typedef bool BOOL;
#else
typedef signed char BOOL;
// BOOL is explicitly signed so @encode(BOOL) == "c" rather than "C"
// even if -funsigned-char is used.
#endif

#define YES ((BOOL)1)
#define NO  ((BOOL)0)
```

4. NSCFBoolean est une classe privée du cluster de classes NSNumber. C'est un pont vers le type CFBooleanRef, qui est utilisé pour encapsuler les valeurs booléennes des listes de propriétés et des collections Core Foundation. CFBoolean définit les constantes kCFBooleanTrue et kCFBooleanFalse. Étant donné que CFNumberRef et CFBooleanRef sont des types différents dans Core Foundation, il est logique qu'ils soient représentés par différentes classes de pontage dans NSNumber.

BOOL VS Booléen

BOOL

- Les frameworks Objective-C d'Apple et la plupart des utilisations de code Objective-C / Cocoa
  BOOL.
- Utilisez BOOL dans objective-C lorsque vous traitez avec des API CoreFoundation

Booléen

- Boolean est un ancien mot-clé Carbon, défini comme un caractère non signé.

Lire BOOL / bool / Boolean / NSCFBoolean en ligne: https://riptutorial.com/fr/objective-c/topic/7267/bool---bool---boolean---nscfboolean

Chapitre 5: Catégories

Syntaxe

- @interface ClassName (categoryName) // ClassName est la classe à étendre
- // Déclarations de méthode et de propriété
- @fin

Remarques

Pour éviter les conflits de noms de méthodes, il est recommandé d'utiliser des préfixes (comme `xyz_` dans l'exemple). Si des méthodes portant le même nom existent, il est impossible de définir celle qui sera utilisée dans le runtime.

Examples

Catégorie simple

Interface et implémentation d'une catégorie simple sur NSArray, nommée Filter, avec une méthode unique qui filtre les nombres.

Il est recommandé d'ajouter un préfixe (`PF`) à la méthode pour éviter de remplacer les futures méthodes `NSArray` .

```
@interface NSArray (PFFilter)

- (NSArray *)pf_filterSmaller:(double)number;

@end

@implementation NSArray (PFFilter)

- (NSArray *)pf_filterSmaller:(double)number
{
    NSMutableArray *result = [NSMutableArray array];
    for (id val in self)
    {
        if ([val isKindOfClass:[NSNumber class] && [val doubleValue] >= number)
        {
            [result addObject:val];
        }
    }
    return [result copy];
}

@end
```

Déclaration d'une méthode de classe

Fichier d'en-tête `UIColor+XYZPalette.h` :

```
@interface UIColor (XYZPalette)

+(UIColor *)xyz_indigoColor;

@end
```

et implémentation `UIColor+XYZPalette.m` :

```
@implementation UIColor (XYZPalette)

+(UIColor *)xyz_indigoColor
{
    return [UIColor colorWithRed:75/255.0f green:0/255.0f blue:130/255.0f alpha:1.0f];
}

@end
```

Ajout d'une propriété avec une catégorie

Des propriétés peuvent être ajoutées avec des catégories en utilisant des objets associés, une fonctionnalité du runtime Objective-C.

Notez que la déclaration de propriété de `retain, nonatomic` correspond au dernier argument de `objc_setAssociatedObject` . Voir Joindre un objet à un autre objet existant pour obtenir des explications.

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

@interface UIViewController (ScreenName)

@property (retain, nonatomic) NSString *screenName;

@end

@implementation UIViewController (ScreenName)

@dynamic screenName;

- (NSString *)screenName {
    return objc_getAssociatedObject(self, @selector(screenName));
}

- (void)setScreenName:(NSString *)screenName {
    objc_setAssociatedObject(self, @selector(screenName), screenName,
OBJC_ASSOCIATION_RETAIN_NONATOMIC);
}

@end
```

Conforme au protocole

Vous pouvez ajouter des protocoles aux classes standard pour étendre leurs fonctionnalités:

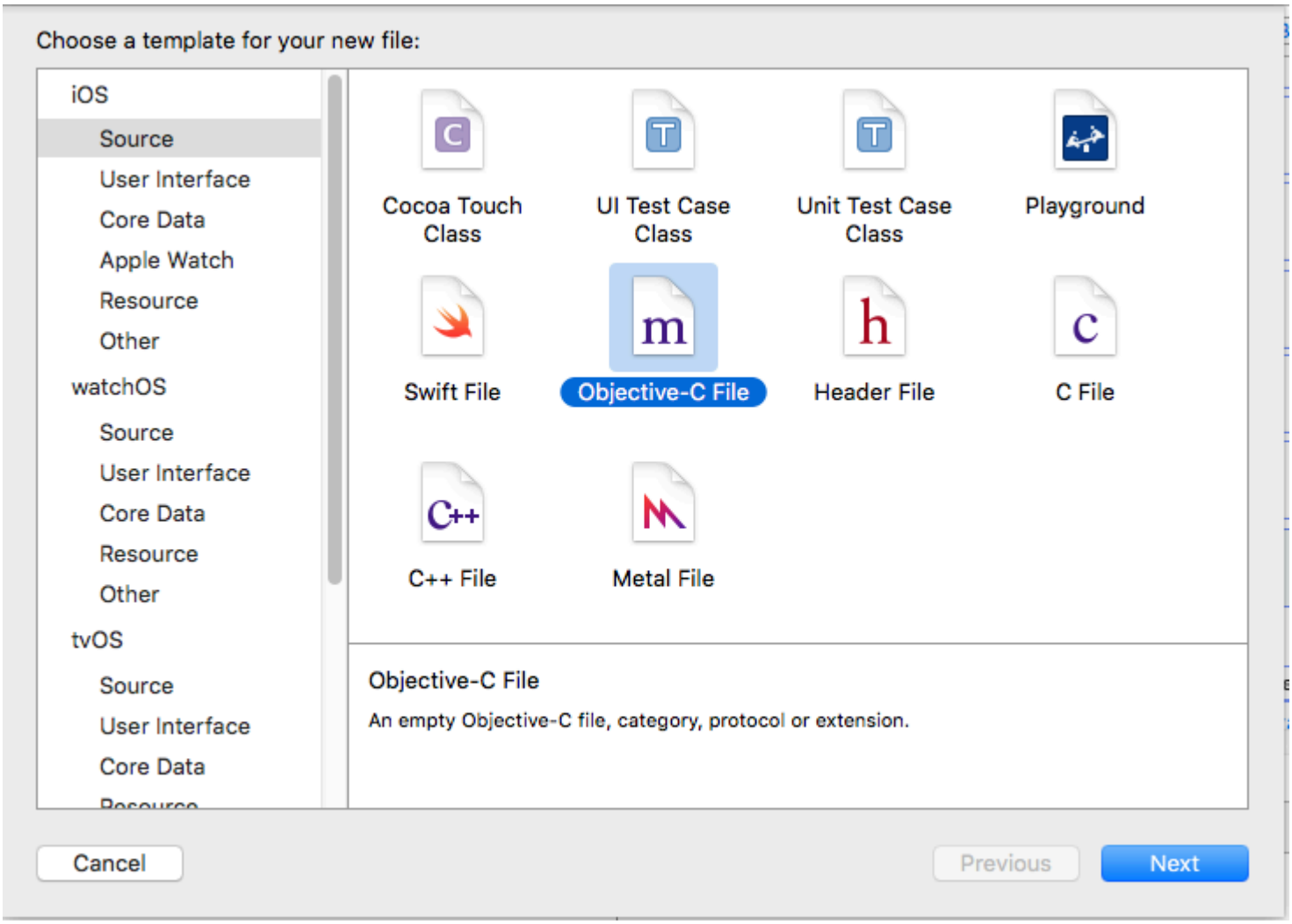
```
@protocol EncodableToString <NSObject>
- (NSString *)toString;
@end

@interface NSDictionary (XYZExtended) <EncodableToString>
@end

@implementation NSDictionary (XYZExtended)
- (NSString *)toString {
    return self.description;
}
@end
```

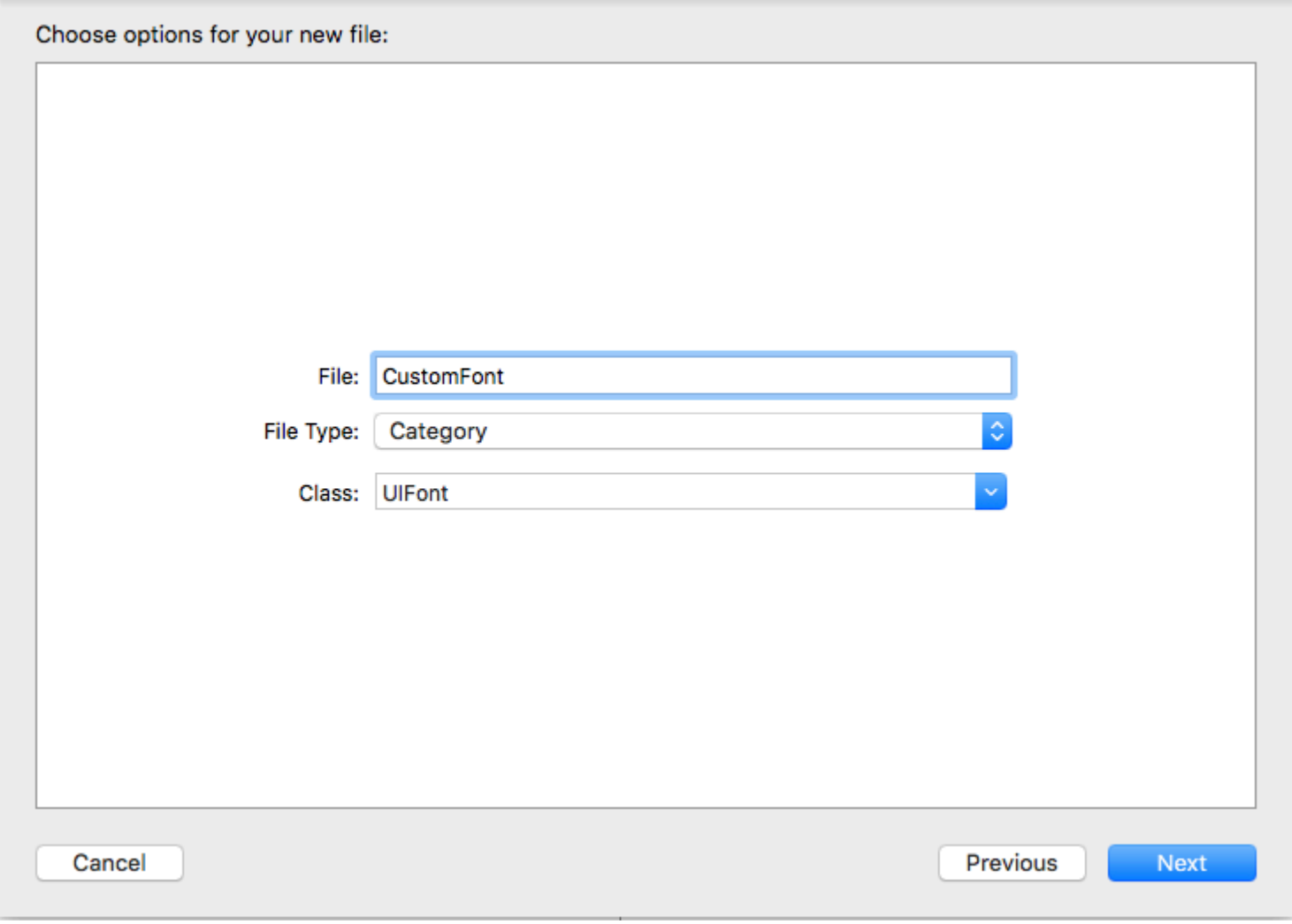
où `XYZ` préfixe de votre projet

Créer une catégorie sur XCode

Les catégories permettent d'ajouter des fonctionnalités supplémentaires à un objet sans sous-classer ni modifier l'objet réel.

Par exemple, nous voulons définir des polices personnalisées. Créons une catégorie qui ajoute des fonctionnalités à la classe `UIFont` . Ouvrez votre projet XCode, cliquez sur `File` -> `New` -> `File` et choisissez le `Objective-C file` , cliquez sur Suivant. Entrez le nom de votre catégorie, puis "CustomFont". "


Choose a template for your new file:
iOS
Source
User Interface
Core Data
Apple Watch
Resource
Other
watchOS
Source
User Interface
Core Data
Resource
Other
tvOS
Source
User Interface
Core Data
Cocoa Touch Class
UI Test Case Class
Unit Test Case Class
Playground
Swift File
Objective-C File
Header File
C File
C++ File
Metal File
Objective-C File
An empty Objective-C file, category, protocol or extension.
Cancel
Previous
Next



Déclarez la méthode de la catégorie: -

Cliquez sur "UIFont + CustomFonts.h" pour afficher le fichier d'en-tête de la nouvelle catégorie. Ajoutez le code suivant à l'interface pour déclarer la méthode.

```
@interface UIFont (CustomFonts)

+(UIFont *)productSansRegularFontWithSize:(CGFloat)size;

@end
```

Maintenant, implémentez la méthode de la catégorie: -

Cliquez sur "UIFont + CustomFonts.m" pour afficher le fichier d'implémentation de la catégorie. Ajoutez le code suivant pour créer une méthode définissant la police ProductSansRegular.

```
+(UIFont *)productSansRegularFontWithSize:(CGFloat)size{

    return [UIFont fontWithName:@"ProductSans-Regular" size:size];

}
```

Importez votre catégorie

```
#import "UIFont+CustomFonts.h"
```

Maintenant, définissez la police d'étiquette

```
[self.label setFont:[UIFont productSansRegularFontWithSize:16.0]];
```

Lire Catégories en ligne: https://riptutorial.com/fr/objective-c/topic/550/categories

Chapitre 6: Classes et Objets

Syntaxe

- Cat * cat = [[Cat alloc] init]; // Créer un objet chat de type Cat
- Dog * dog = [[Dog alloc] init]; // Créer un objet chien de type Chien
- NSObject * someObject = [NSObject alloc]; [someObject init]; // **ne fais pas ça**
- XYZObject * object = [XYZObject new]; // Utilise new pour créer des objets si AUCUN argument n'est requis pour l'initialisation
- NSString * someString = @ "Bonjour, Monde!"; // Créer un NSString avec *une syntaxe littérale*
- NSNumber * myFloat = @ 3.14f; // Un autre exemple pour créer un NSNumber en utilisant une syntaxe littérale
- NSNumber * myInt = @ (84/2); // Crée un objet en utilisant une expression encadrée

Examples

Création de classes avec des valeurs d'initialisation

```
#import <Foundation/Foundation.h>
@interface Car:NSObject {
    NSString *CarMotorCode;
    NSString *CarChassisCode;
}

- (instancetype)initWithMotorValue:(NSString *) motorCode
andChassisValue:(NSInteger)chassisCode;
- (void) startCar;
- (void) stopCar;

@end

@implementation Car

- (instancetype)initWithMotorValue:(NSString *) motorCode
andChassisValue:(NSInteger)chassisCode{
    CarMotorCode = motorCode;
    CarChassisCode = chassisCode;
    return self;
}

- (void) startCar {...}
- (void) stopCar {...}

@end
```

La méthode `initWithMotorValue: type andChassisValue: type` sera utilisée pour initialiser les objets Car.

Classe Singleton

Qu'est-ce qu'une classe Singleton?

Une classe singleton renvoie la même instance, quel que soit le nombre de fois qu'une application le demande. Contrairement à une classe régulière, un objet singleton fournit un point d'accès global aux ressources de sa classe.

Quand utiliser les classes Singleton?

Les singletons sont utilisés dans des situations où ce point de contrôle unique est souhaitable, par exemple avec des classes offrant des services généraux ou des ressources.

Comment créer des classes Singleton

Tout d'abord, créez un nouveau fichier et sous-classez-le à partir de `NSObject` . Nommez-le, nous utiliserons `CommonClass` ici. Xcode va maintenant générer des fichiers CommonClass.h et CommonClass.m pour vous.

Dans votre fichier `CommonClass.h` :

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

@interface CommonClass : NSObject {
}
+ (CommonClass *)sharedObject;
@property NSString *commonString;
@end
```

Dans votre fichier `CommonClass.m` :

```
#import "CommonClass.h"

@implementation CommonClass

+ (CommonClass *)sharedObject {
    static CommonClass *sharedClass = nil;
    static dispatch_once_t onceToken;
    dispatch_once(&onceToken, ^{
        sharedClass = [[self alloc] init];
    });
    return sharedClass;
}

- (id)init {
    if (self = [super init]) {
        self.commonString = @"this is string";
    }
    return self;
}

@end
```

Comment utiliser les classes Singleton

La classe Singleton que nous avons créée précédemment sera accessible de n'importe où dans le

projet tant que vous avez importé le fichier `CommonClass.h` dans le module concerné. Pour modifier et accéder aux données partagées dans la classe Singleton, vous devrez accéder à l'objet partagé de cette classe accessible à l'aide de la méthode `sharedObject` , comme suit:

```
[CommonClass sharedObject]
```

Pour lire ou modifier les éléments de la classe partagée, procédez comme suit:

```
NSString *commonString = [[CommonClass sharedObject].commonString; //Read the string in
singleton class

NSString *newString = @"New String";
[CommonClass sharedObject].commonString = newString;//Modified the string in singleton class
```

Le type de retour "instancetype"

Objective-C prend en charge un type spécial appelé `instancetype qui ne peut être utilisé que comme type renvoyé par une méthode. Il évalue à la classe de l'objet récepteur.

Considérons la hiérarchie de classes suivante:

```
@interface Foo : NSObject

- (instancetype)initWithString:(NSString *)string;

@end

@interface Bar : Foo
@end
```

Lorsque `[[Foo alloc] initWithString:@"abc"]` est appelé, le compilateur peut en déduire que le type de retour est `Foo *` . La classe `Bar` dérivée de `Foo` mais n'a pas remplacé la déclaration de l'initialiseur. Cependant, grâce à `instancetype` , le compilateur peut déduire que `[[Bar alloc] initWithString:@"xyz"]` renvoie une valeur de type `Bar *` .

Considérez le type de retour de `-[Foo initWithString:]` étant à la place `Foo *` : si vous appelez `[[Bar alloc] initWithString:]` , le compilateur en `[[Bar alloc] initWithString:]` qu'un `Foo *` est renvoyé, et non une `Bar *` développeur. Le type d' `instancetype` résolu ce problème.

Avant l'introduction de l' `instancetype` , les initialiseurs, les méthodes statiques telles que les accesseurs singleton et les autres méthodes souhaitant renvoyer une instance de la classe réceptrice devaient renvoyer un `id` . Le problème est que `id` signifie *"un objet de tout type"* . Le compilateur ne peut donc pas détecter que `NSString *wrong = [[Foo alloc] initWithString:@"abc"];` attribue une variable avec un type incorrect.

En raison de ce problème, les **initialiseurs doivent toujours utiliser `instancetype` au lieu de `id`** comme valeur de retour.

Spécification de génériques

Vous pouvez améliorer vos propres classes avec des *génériques* comme `NSArray` ou `NSDictionary` .

```
@interface MyClass<__covariant T>

@property (nonnull, nonatomic, strong, readonly) NSArray<T>* allObjects;

- (void) addObject:(nonnull T)obj;

@end
```

Différence entre allocation et initialisation

Dans la plupart des langages orientés objet, l'allocation de mémoire pour un objet et son initialisation est une opération atomique:

```
// Both allocates memory and calls the constructor
MyClass object = new MyClass();
```

En Objective-C, ce sont des opérations distinctes. Les méthodes de classe `alloc` (et son frère historique `allocWithZone:` font que le runtime Objective-C réserve la mémoire requise et l'efface. À l'exception de quelques valeurs internes, toutes les propriétés et variables sont définies sur 0 / `NO` / `nil` .

L'objet est alors déjà "valide" mais nous voulons toujours appeler une méthode pour configurer réellement l'objet, que nous appelons un *initialiseur* . Celles-ci servent le même objectif que les *constructeurs* dans d'autres langues. Par convention, ces méthodes commencent par `init` . Du point de vue du langage, ce ne sont que des méthodes normales.

```
// Allocate memory and set all properties and variables to 0/NO/nil.
MyClass *object = [MyClass alloc];
// Initialize the object.
object = [object init];

// Shorthand:
object = [[MyClass alloc] init];
```

Lire Classes et Objets en ligne: https://riptutorial.com/fr/objective-c/topic/758/classes-et-objets

Chapitre 7: Continuer et casser!

Examples

Déclaration de continuation et de rupture

L'instruction continue dans le langage de programmation Objective-C fonctionne un peu comme l'instruction break. Au lieu de forcer la terminaison, continuez à forcer la prochaine itération de la boucle, en ignorant tout code intermédiaire.

Pour la boucle for, l'instruction continue entraîne l'exécution des parties conditionnelles et d'incrémentation de la boucle. Pendant ce temps, les boucles while, l'instruction continue provoque le passage du contrôle de programme aux tests conditionnels.

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

int main ()
{
   /* local variable definition */
   int a = 10;

   /* do loop execution */
   do
   {
      if( a == 15)
      {
         /* skip the iteration */
         a = a + 1;
         continue;
      }
      NSLog(@"value of a: %d\n", a);
      a++;

   }while( a < 20 );

   return 0;
}
```

Sortie:

```
2013-09-07 22:20:35.647 demo[29998] value of a: 10
2013-09-07 22:20:35.647 demo[29998] value of a: 11
2013-09-07 22:20:35.647 demo[29998] value of a: 12
2013-09-07 22:20:35.647 demo[29998] value of a: 13
2013-09-07 22:20:35.647 demo[29998] value of a: 14
2013-09-07 22:20:35.647 demo[29998] value of a: 16
2013-09-07 22:20:35.647 demo[29998] value of a: 17
2013-09-07 22:20:35.647 demo[29998] value of a: 18
2013-09-07 22:20:35.647 demo[29998] value of a: 19
```

Reportez-vous à ce lien pour plus d'informations.

Lire Continuer et casser! en ligne: https://riptutorial.com/fr/objective-c/topic/8709/continuer-et-casser-

Chapitre 8: Déclarer la méthode de classe et la méthode d'instance

Introduction

La méthode d'instance est une méthode spécifique à certaines classes. Les méthodes d'instance sont déclarées et définies, suivies du symbole - (moins).

Les méthodes de classe peuvent être appelées par le nom de classe lui-même. Les méthodes de classe sont déclarées et définies à l'aide du signe + (plus).

Syntaxe

1. - (void) testInstanceMethod; // Les méthodes de classe déclarent avec le signe "+"
2. (void) classMethod; // méthodes d'instance déclarent avec le signe "-"

Examples

Comment déclarer une méthode de classe et une méthode d'instance.

les méthodes d'instance utilisent une instance d'une classe.

```
@interface MyTestClass : NSObject

- (void)testInstanceMethod;

@end
```

Ils pourraient alors être utilisés comme ça:

```
MyTestClass *object = [[MyTestClass alloc] init];
[object testInstanceMethod];
```

La méthode de classe peut être utilisée avec uniquement le nom de la classe.

```
@interface MyClass : NSObject

+ (void)aClassMethod;

@end
```

Ils pourraient alors être utilisés comme ça:

```
[MyClass aClassMethod];
```

les méthodes de classe sont les méthodes pratiques de nombreuses classes Foundation telles que [NSString's + stringWithFormat:] ou NSArray + arrayWithArray

Lire Déclarer la méthode de classe et la méthode d'instance en ligne: https://riptutorial.com/fr/objective-c/topic/8214/declarer-la-methode-de-classe-et-la-methode-d-instance

Chapitre 9: Enregistrement

Syntaxe

- NSLog (@ "text to log"); // Journal de texte de base
- NSLog (@ "data:% f -% .2f", myFloat, anotherFloat); // Enregistrement du texte, y compris les nombres flottants.
- NSLog (@ "data:% i", myInteger); // Enregistrement de texte incluant un nombre entier.
- NSLog (@ "data:% @", myStringOrObject); // Enregistrement de texte faisant référence à une autre chaîne ou à un objet dérivé de NSObject.

Remarques

Pour la journalisation de différents types d'objets et de types de données, reportez-vous à: Objective-C, Spécificateurs de format

Examples

Enregistrement

```
NSLog(@"Log Message!");
NSLog(@"NSString value: %@", stringValue);
NSLog(@"Integer value: %d", intValue);
```

Le premier argument de `NSLog` est un `NSString` contenant le format du message de journal. Les autres paramètres sont utilisés comme valeurs pour remplacer les spécificateurs de format.

Le formatage fonctionne exactement comme `printf` , à l'exception du spécificateur de format supplémentaire `%@` pour un objet Objective-C arbitraire. Ce:

```
NSLog(@"%@", object);
```

est équivalent à:

```
NSLog(@"%s", [object description].UTF8String);
```

NSLog vs printf

```
NSLog(@"NSLog message");
printf("printf message\n");
```

Sortie:

```
2016-07-16 08:58:04.681 test[46259:1244773] NSLog message
```

```
printf message
```

`NSLog` la date, l'heure, le nom du processus, l'ID du processus et l'ID du thread en plus du message de journal. `printf` que le message.

`NSLog` nécessite un `NSString` et ajoute automatiquement une nouvelle ligne à la fin. `printf` nécessite une chaîne C et n'ajoute pas automatiquement une nouvelle ligne.

`NSLog` envoie la sortie à `stderr` , `printf` envoie la sortie à `stdout` .

Certains `format-specifiers` dans `printf` vs `NSLog` sont différents. Par exemple, lorsque vous incluez une chaîne imbriquée, les différences suivantes sont à prévoir:

```
NSLog(@"My string: %@", (NSString *)myString);
printf("My string: %s", [(NSString *)myString UTF8String]);
```

Format de sortie NSLog

```
NSLog(@"NSLog message");
```

Le message qui est imprimé en appelant `NSLog` a le format suivant lorsqu'il est affiché dans Console.app:

Rendez-vous amoureux	Temps	Nom du programme	ID du processus		ID de fil	Message
2016-07-16	08:58:04.681	test	[46259	:	1244773]	NSLog message

Enregistrement des valeurs variables

Vous ne devriez pas appeler `NSLog` sans une chaîne de format littérale comme celle-ci:

```
NSLog(variable);    // Dangerous code!
```

Si la variable n'est pas une `NSString` , le programme se `NSLog` , car `NSLog` attend une `NSString` .

Si la variable est une `NSString` , elle fonctionnera à moins que votre chaîne ne contienne un `%` . `NSLog` analysera la séquence `%` tant que spécificateur de format, puis lira une valeur de mémoire de la pile, provoquant un blocage ou même exécutant du code arbitraire .

Au lieu de cela, faites toujours le premier argument comme spécificateur de format, comme ceci:

```
NSLog(@"%@", anObjectVariable);
NSLog(@"%d", anIntegerVariable);
```

Le message vide n'est pas imprimé

Lorsque `NSLog` est invité à imprimer une chaîne vide, il omet complètement le journal.

```
NSString *name = @"";
NSLog(@"%@", name);  // Resolves to @""
```

Le code ci-dessus n'imprimera **rien** .

Il est recommandé de préfixer les journaux avec des étiquettes:

```
NSString *name = @"";
NSLog(@"Name: %@", name);  // Resolves to @"Name: "
```

Le code ci-dessus sera imprimé:

```
2016-07-21 14:20:28.623 App[87711:6153103] Name:
```

Suppression des enregistrements de journal des versions de publication

Les messages imprimés à partir de `NSLog` sont affichés sur Console.app même dans la version de publication de votre application, ce qui n'a aucun sens pour les impressions qui ne sont utiles que pour le débogage. Pour résoudre ce problème, vous pouvez utiliser cette macro pour la journalisation du débogage au lieu de `NSLog` .

```
#ifdef DEBUG
#define DLog(...) NSLog(__VA_ARGS__)
#else
#define DLog(...)
#endif
```

Utiliser:

```
NSString *value = @"value 1";
DLog(@"value = %@", value);
// little known fact: programmers look for job postings in Console.app
NSLog(@"We're hiring!");
```

Dans les versions de débogage, `DLog` appellera `NSLog` . En version release, `DLog` ne fera rien.

Utiliser __FUNCTION __

```
NSLog(@"%s %@",__FUNCTION__, @"etc etc");
```

Insère le nom de la classe et de la méthode dans la sortie:

```
2016-07-22 12:51:30.099 loggingExample[18132:2971471] -[ViewController viewDidLoad] etc etc
```

Type NSLog et BOOL

Il n'y a pas de spécificateur de format pour imprimer le type booléen en utilisant NSLog. Une façon d'imprimer une valeur booléenne consiste à la convertir en chaîne.

```
BOOL boolValue = YES;
NSLog(@"Bool value %@", boolValue ? @"YES" : @"NO");
```

Sortie:

```
2016-07-30 22:53:18.269 Test[4445:64129] Bool value YES
```

Une autre façon d'imprimer une valeur booléenne consiste à la convertir en entier, en obtenant une sortie binaire (1 = oui, 0 = non).

```
BOOL boolValue = YES;
NSLog(@"Bool value %i", boolValue);
```

Sortie:

```
2016-07-30 22:53:18.269 Test[4445:64129] Bool value 1
```

Consignation des métadonnées NSLog

```
NSLog(@"%s %d %s, yourVariable: %@", __FILE__, __LINE__, __PRETTY_FUNCTION__, yourVariable);
```

Enregistre le fichier, le numéro de ligne et les données de fonction avec les variables que vous souhaitez consigner. Cela peut rendre les lignes de journal beaucoup plus longues, en particulier avec les noms de fichier et de méthode détaillés, mais cela peut aider à accélérer les diagnostics d'erreur.

Vous pouvez également emballer ceci dans une macro (stockez ceci dans un Singleton ou où vous en aurez le plus besoin);

```
#define ALog(fmt, ...) NSLog((@"%s [Line %d] " fmt), __PRETTY_FUNCTION__, __LINE__,
##__VA_ARGS__);
```

Ensuite, lorsque vous souhaitez vous connecter, appelez simplement

```
ALog(@"name: %@", firstName);
```

Ce qui vous donnera quelque chose comme;

```
-[AppDelegate application:didFinishLaunchingWithOptions:] [Line 27] name: John
```

Enregistrement en ajoutant à un fichier

NSLog est bon, mais vous pouvez également vous connecter en ajoutant un fichier à la place, en

utilisant le code suivant:

```
NSFileHandle* fh = [NSFileHandle fileHandleForWritingAtPath:path];
if ( !fh ) {
    [[NSFileManager defaultManager] createFileAtPath:path contents:nil attributes:nil];
    fh = [NSFileHandle fileHandleForWritingAtPath:path];
}
if ( fh ) {
    @try {
        [fh seekToEndOfFile];
        [fh writeData:[self dataUsingEncoding:enc]];
    }
    @catch (...) {
    }
    [fh closeFile];
}
```

Lire Enregistrement en ligne: https://riptutorial.com/fr/objective-c/topic/724/enregistrement

Chapitre 10: Entier aléatoire

Examples

Entier de base aléatoire

La fonction `arc4random_uniform()` est la manière la plus simple d'obtenir des nombres entiers aléatoires de haute qualité. Selon le manuel:

> arc4random_uniform (upper_bound) renverra un nombre aléatoire uniformément distribué inférieur à upper_bound.
>
> arc4random_uniform () est recommandé sur les constructions comme " arc4random ()% upper_bound " car il évite le "biais modulo" lorsque la limite supérieure n'est pas une puissance de deux.

```
uint32_t randomInteger = arc4random_uniform(5); // A random integer between 0 and 4
```

Entier aléatoire dans une plage

Le code suivant illustre l'utilisation de `arc4random_uniform()` pour générer un entier aléatoire compris entre 3 et 12:

```
uint32_t randomIntegerWithinRange = arc4random_uniform(10) + 3; // A random integer between 3
and 12
```

Cela permet de créer une plage car `arc4random_uniform(10)` renvoie un entier compris entre 0 et 9. L'ajout de 3 à cet entier aléatoire produit une plage comprise entre `0 + 3` et `9 + 3` .

Lire Entier aléatoire en ligne: https://riptutorial.com/fr/objective-c/topic/1573/entier-aleatoire

Chapitre 11: Énumération rapide

Examples

Énumération rapide d'un NSArray

Cet exemple montre comment utiliser l'énumération rapide pour traverser un NSArray.

Lorsque vous avez un tableau, tel que

```
NSArray *collection = @[@"fast", @"enumeration", @"in objc"];
```

Vous pouvez utiliser la syntaxe `for ... in` pour parcourir chaque élément du tableau, en commençant automatiquement par le premier à l'index `0` et en vous arrêtant avec le dernier élément:

```
for (NSString *item in collection) {
    NSLog(@"item: %@", item);
}
```

Dans cet exemple, la sortie générée ressemblerait à

```
// item: fast
// item: enumeration
// item: in objc
```

Énumération rapide d'un NSArray avec index.

Cet exemple montre comment utiliser l'énumération rapide pour traverser un NSArray. De cette façon, vous pouvez également suivre l'index de l'objet en cours lors de la traversée.

Supposons que vous ayez un tableau,

```
NSArray *weekDays = @[@"Monday", @"Tuesday", @"Wednesday", @"Thursday", @"Friday",
@"Saturday", @"Sunday"];
```

Maintenant, vous pouvez parcourir le tableau comme ci-dessous,

```
[weekDays enumerateObjectsUsingBlock:^(id obj, NSUInteger idx, BOOL *stop) {

    //... Do your usual stuff here

    obj  // This is the current object
    idx  // This is the index of the current object
    stop // Set this to true if you want to stop

}];
```

Lire Énumération rapide en ligne: https://riptutorial.com/fr/objective-c/topic/5583/enumeration-rapide

Chapitre 12: Enums

Syntaxe

- typedef NS_ENUM (type, name) {...} - *type* est le type d'énumération et *name* est le nom de l'énumération. les valeurs sont dans "...". Cela crée un enum de base et un type pour aller avec; les programmes comme Xcode supposent qu'une variable avec le type enum a l'une des valeurs enum

Examples

Définir un enum

Les énumérations sont définies par la syntaxe ci-dessus.

```
typedef NS_ENUM(NSUInteger, MyEnum) {
    MyEnumValueA,
    MyEnumValueB,
    MyEnumValueC,
};
```

Vous pouvez également définir vos propres valeurs brutes sur les types d'énumération.

```
typedef NS_ENUM(NSUInteger, MyEnum) {
    MyEnumValueA = 0,
    MyEnumValueB = 5,
    MyEnumValueC = 10,
};
```

Vous pouvez également spécifier la première valeur et tous les éléments suivants l'utiliseront avec un incrément:

```
typedef NS_ENUM(NSUInteger, MyEnum) {
    MyEnumValueA = 0,
    MyEnumValueB,
    MyEnumValueC,
};
```

Les variables de cette énumération peuvent être créées par `MyEnum enumVar = MyEnumValueA`.

Déclaration de typedef enum dans Objective-C

Un enum déclare un ensemble de valeurs ordonnées - le typedef ajoute simplement un nom pratique à ceci. Le 1er élément est 0 etc.

```
 typedef enum {
        Monday=1,
        Tuesday,
```

```
        Wednesday

    } WORKDAYS;

    WORKDAYS today = Monday;//value 1
```

Conversion de C ++ std :: vector vers un tableau Objective-C

De nombreuses bibliothèques C ++ utilisent des énumérations et renvoient / reçoivent des données à l'aide de vecteurs contenant des énumérations. Comme les énumérations C ne sont pas des objets Objective-C, les collections Objective-C ne peuvent pas être utilisées directement avec les énumérations C. L'exemple ci-dessous traite de cela en utilisant une combinaison de NSArray et de génériques et un objet wrapper pour le tableau. De cette manière, la collection peut être explicite sur le type de données et il n'ya pas de soucis concernant les fuites de mémoire possibles avec les tableaux C Les objets Objective-C sont utilisés.

Voici l'objet équivalent C enum & Objective-C:

```
typedef enum
{
  Error0 = 0,
  Error1 = 1,
  Error2 = 2
} MyError;

@interface ErrorEnumObj : NSObject

@property (nonatomic) int intValue;

+ (instancetype) objWithEnum:(MyError) myError;
- (MyError) getEnumValue;

@end

@implementation ErrorEnumObj

+ (instancetype) objWithEnum:(MyError) error
{
    ErrorEnumObj * obj = [ErrorEnumObj new];
    obj.intValue = (int)error;
    return obj;
}

- (MyError) getEnumValue
{
    return (MyError)self.intValue;
}

@end
```

Et voici une utilisation possible dans Objective-C ++ (le NSArray résultant peut être utilisé dans les fichiers Objective-C uniquement car aucun C ++ n'est utilisé).

```
class ListenerImpl : public Listener
{
```

```
public:
    ListenerImpl(Listener* listener) : _listener(listener) {}
    void onError(std::vector<MyError> errors) override
    {
        NSMutableArray<ErrorEnumObj *> * array = [NSMutableArray<ErrorEnumObj *> new];
        for (auto&& myError : errors)
        {
            [array addObject:[ErrorEnumObj objWithEnum:myError]];
        }
        [_listener onError:array];
    }

private:
    __weak Listener* _listener;
}
```

Si ce type de solution doit être utilisé sur plusieurs énumérations, la création de EnumObj (déclaration et implémentation) peut être effectuée à l'aide d'une macro (pour créer un modèle comme une solution).

Lire Enums en ligne: https://riptutorial.com/fr/objective-c/topic/1461/enums

Chapitre 13: Environnement d'exécution de bas niveau

Remarques

Pour utiliser le runtime Objective-C, vous devez l'importer.

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

Examples

Joindre un objet à un autre objet existant (association)

Il est possible d'attacher un objet à un objet existant comme s'il y avait une nouvelle propriété. Cela s'appelle l' *association* et permet d'étendre des objets existants. Il peut être utilisé pour fournir du stockage lors de l'ajout d'une propriété via une extension de classe ou pour ajouter des informations supplémentaires à un objet existant.

L'objet associé est automatiquement libéré par le runtime une fois que l'objet cible est libéré.

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

// "Key" for association. Its value is never used and doesn't
// matter. The only purpose of this global static variable is to
// provide a guaranteed unique value at runtime: no two distinct
// global variables can share the same address.
static char key;

id target = ...;
id payload = ...;
objc_setAssociateObject(target, &key, payload, OBJC_ASSOCIATION_RETAIN);
// Other useful values are OBJC_ASSOCIATION_COPY
// and OBJ_ASSOCIATION_ASSIGN

id queryPayload = objc_getAssociatedObject(target, &key);
```

Augmenter les méthodes en utilisant Method Swizzling

Le runtime Objective-C vous permet de modifier l'implémentation d'une méthode à l'exécution. Ceci est appelé *méthode swizzling* et est souvent utilisé pour échanger les implémentations de deux méthodes. Par exemple, si les méthodes `foo` et `bar` sont échangées, l'envoi du message `foo` va maintenant exécuter l'implémentation de `bar` et vice versa.

Cette technique peut être utilisée pour augmenter ou "patcher" les méthodes existantes que vous ne pouvez pas éditer directement, telles que les méthodes des classes fournies par le système.

Dans l'exemple suivant, la méthode `-[NSUserDefaults synchronize]` est augmentée pour imprimer

l'heure d'exécution de l'implémentation d'origine.

IMPORTANT: beaucoup de gens essaient de faire des swizzling en utilisant `method_exchangeImplementations` . Cependant, cette approche est dangereuse si vous devez appeler la méthode que vous remplacez, car vous l'appellerez avec un sélecteur différent de celui attendu. Par conséquent, votre code peut se révéler étrange et inattendu, en particulier si plusieurs utilisateurs manipulent un objet de cette manière. Au lieu de cela, vous devriez toujours faire un swizzling en utilisant `setImplementation` conjointement avec une fonction C, vous permettant d'appeler la méthode avec le sélecteur d'origine.

```
#import "NSUserDefaults+Timing.h"
#import <objc/runtime.h> // Needed for method swizzling

static IMP old_synchronize = NULL;

static void new_synchronize(id self, SEL _cmd);

@implementation NSUserDefaults(Timing)

+ (void)load
{
    Method originalMethod = class_getInstanceMethod([self class], @selector(synchronize:));
    IMP swizzleImp = (IMP)new_synchronize;
    old_synchronize = method_setImplementation(originalMethod, swizzleImp);
}
@end

static void new_synchronize(id self, SEL _cmd);
{
    NSDate *started;
    BOOL returnValue;

    started = [NSDate date];

    // Call the original implementation, passing the same parameters
    // that this function was called with, including the selector.
    returnValue = old_synchronize(self, _cmd);


    NSLog(@"Writing user defaults took %f seconds.", [[NSDate date]
timeIntervalSinceDate:started]);

    return returnValue;
}

@end
```

Si vous devez modifier une méthode qui prend des paramètres, vous devez simplement les ajouter en tant que paramètres supplémentaires à la fonction. Par exemple:

```
static IMP old_viewWillAppear_animated = NULL;
static void new_viewWillAppear_animated(id self, SEL _cmd, BOOL animated);

...

Method originalMethod = class_getClassMethod([UIViewController class],
@selector(viewWillAppear:));
```

```
IMP swizzleImp = (IMP)new_viewWillAppear_animated;
old_viewWillAppear_animated = method_setImplementation(originalMethod, swizzleImp);

...

static void new_viewWillAppear_animated(id self, SEL _cmd, BOOL animated)
{
    ...

    old_viewWillAppear_animated(self, _cmd, animated);

    ...
}
```

Méthodes d'appel directement

Si vous avez besoin d'appeler une méthode Objective-C à partir du code C, vous avez deux façons: d'utiliser `objc_msgSend` ou d'obtenir le pointeur de fonction d'implémentation de méthode (`IMP`) et de l'appeler.

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

@implementation Example

- (double)negate:(double)value {
    return -value;
}

- (double)invert:(double)value {
    return 1 / value;
}

@end

// Calls the selector on the object. Expects the method to have one double argument and return
a double.
double performSelectorWithMsgSend(id object, SEL selector, double value) {
    // We declare pointer to function and cast `objc_msgSend` to expected signature.
    // WARNING: This step is important! Otherwise you may get unexpected results!
    double (*msgSend)(id, SEL, double) = (typeof(msgSend)) &objc_msgSend;

    // The implicit arguments of self and _cmd need to be passed in addition to any explicit
arguments.
    return msgSend(object, selector, value);
}

// Does the same as the above function, but by obtaining the method's IMP.
double performSelectorWithIMP(id object, SEL selector, double value) {
    // Get the method's implementation.
    IMP imp = class_getMethodImplementation([self class], selector);

    // Cast it so the types are known and ARC can work correctly.
    double (*callableImp)(id, SEL, double) = (typeof(callableImp)) imp;

    // Again, you need the explicit arguments.
    return callableImp(object, selector, value);
}
```

```
int main() {
    Example *e = [Example new];

    // Invoke negation, result is -4
    double x = performSelectorWithMsgSend(e, @selector(negate:), 4);

    // Invoke inversion, result is 0.25
    double y = performSelectorWithIMP(e, @selector(invert:), 4);
}
```

`objc_msgSend` fonctionne en obtenant l'IMP pour la méthode et en l'appelant. Les `IMP` pour les dernières méthodes appelées sont mises en cache, donc si vous envoyez un message Objective-C dans une boucle très serrée, vous pouvez obtenir des performances acceptables. Dans certains cas, la mise en cache manuelle de l'IMP peut donner des performances légèrement meilleures, même s'il s'agit d'une optimisation de dernier recours.

Lire Environnement d'exécution de bas niveau en ligne: https://riptutorial.com/fr/objective-c/topic/1180/environnement-d-execution-de-bas-niveau

Chapitre 14: Expédition Grand Central

Introduction

Grand Central Dispatch (GCD) Dans iOS, Apple propose deux méthodes multitâches: les frameworks Grand Central Dispatch (GCD) et NSOperationQueue. Nous en discuterons ici à propos de GCD. GCD est un moyen léger de représenter des unités de travail qui seront exécutées simultanément. Vous ne planifiez pas ces unités de travail; le système prend en charge la planification pour vous. L'ajout de dépendance entre les blocs peut être un casse-tête. L'annulation ou la suspension d'un bloc crée un travail supplémentaire pour vous en tant que développeur!

Examples

Qu'est-ce que la dépêche centrale?

Qu'est-ce que la concurrence?

- Faire plusieurs choses en même temps.
- Profitant du nombre de cœurs disponibles dans les CPU multicœurs.
- Exécution de plusieurs programmes en parallèle.

Objectifs de la concurrence

- Programme en cours d'exécution en arrière-plan sans courir le processeur.
- Définissez les tâches, définissez les règles et laissez le système prendre la responsabilité de les exécuter.
- Améliorez la réactivité en vous assurant que le thread principal est libre de répondre aux événements des utilisateurs.

RÉPARTITIONS

Expédition centralisée centralisée - les files d'attente de répartition nous permettent d'exécuter des blocs de code arbitraires de manière asynchrone ou synchrone. Toutes les files d'attente de répartition sont les premières entrées.

Lire Expédition Grand Central en ligne: https://riptutorial.com/fr/objective-c/topic/8280/expedition-grand-central

Chapitre 15: Gestion de la mémoire

Examples

Comptage automatique des références

Avec le comptage automatique des références (ARC), le compilateur insère des instructions de `retain` , de `release` et d' `autorelease` lorsque cela est nécessaire, ce qui vous `autorelease` de les écrire vous-même. Il écrit également des méthodes de `dealloc` pour vous.

Le programme exemple de Manual Memory Management ressemble à ceci avec ARC:

```
@interface MyObject : NSObject {
    NSString *_property;
}
@end

@implementation MyObject
@synthesize property = _property;

- (id)initWithProperty:(NSString *)property {
    if (self = [super init]) {
        _property = property;
    }
    return self;
}

- (NSString *)property {
    return property;
}

- (void)setProperty:(NSString *)property {
    _property = property;
}

@end
```

```
int main() {
    MyObject *obj = [[MyObject alloc] init];

    NSString *value = [[NSString alloc] initWithString:@"value"];
    [obj setProperty:value];

    [obj setProperty:@"value"];
}
```

Vous pouvez toujours remplacer la méthode dealloc pour nettoyer les ressources non gérées par ARC. Contrairement à l'utilisation de la gestion manuelle de la mémoire, vous n'appelez pas `[super dealloc]` .

```
-(void)dealloc {
   //clean up
}
```

Références fortes et faibles

Moderne

Une référence faible ressemble à l'une de ces:

```
@property (weak) NSString *property;
NSString *__weak variable;
```

Si vous avez une référence faible à un objet, alors sous le capot:

- Vous ne le retenez pas.
- Quand il est désalloué, chaque référence sera automatiquement mise à `nil`

Les références d'objet sont toujours fortes par défaut. Mais vous pouvez explicitement spécifier qu'ils sont forts:

```
@property (strong) NSString *property;
NSString *__strong variable;
```

Une référence forte signifie que, même si cette référence existe, vous conservez l'objet.

Gestion de la mémoire manuelle

Voici un exemple de programme écrit avec gestion manuelle de la mémoire. Vous ne devriez vraiment pas écrire votre code comme celui-ci, à moins que vous ne puissiez pas utiliser ARC (comme si vous deviez prendre en charge 32 bits). L'exemple évite la notation `@property` pour illustrer comment vous deviez écrire des getters et des setters.

```
@interface MyObject : NSObject {
    NSString *_property;
}
@end

@implementation MyObject
@synthesize property = _property;

- (id)initWithProperty:(NSString *)property {
    if (self = [super init]) {
        // Grab a reference to property to make sure it doesn't go away.
        // The reference is released in dealloc.
        _property = [property retain];
    }
    return self;
}

- (NSString *)property {
    return [[property retain] autorelease];
}

- (void)setProperty:(NSString *)property {
    // Retain, then release. So setting it to the same value won't lose the reference.
    [property retain];
```

```
    [_property release];
    _property = property;
}

- (void)dealloc {
    [_property release];
    [super dealloc]; // Don't forget!
}

@end
```

```
int main() {
    // create object
    // obj is a reference that we need to release
    MyObject *obj = [[MyObject alloc] init];

    // We have to release value because we created it.
    NSString *value = [[NSString alloc] initWithString:@"value"];
    [obj setProperty:value];
    [value release];

    // However, string constants never need to be released.
    [obj setProperty:@"value"];
    [obj release];
}
```

Règles de gestion de la mémoire lors de l'utilisation du comptage manuel des références.

Ces règles s'appliquent uniquement si vous utilisez le comptage manuel des références!

1. **Vous possédez un objet que vous créez**

   En appelant une méthode dont le nom commence par `alloc` , `new` , `copy` ou `mutableCopy` . Par exemple:

   ```
   NSObject *object1 = [[NSObject alloc] init];
   NSObject *object2 = [NSObject new];
   NSObject *object3 = [object2 copy];
   ```

   Cela signifie que vous êtes responsable de la libération de ces objets lorsque vous en avez fini avec eux.

2. **Vous pouvez prendre possession d'un objet à l'aide de keep**

   Pour prendre possession d'un objet que vous appelez la méthode de conservation.

   Par exemple:

   ```
   NSObject *object = [NSObject new]; // object already has a retain count of 1
   [object retain]; // retain count is now 2
   ```

   Cela n'a de sens que dans certaines situations rares.

Par exemple, lorsque vous implémentez un accesseur ou une méthode init pour prendre possession:

```
- (void)setStringValue:(NSString *)stringValue {
    [_privateStringValue release]; // Release the old value, you no longer need it
    [stringValue retain]; // You make sure that this object does not get deallocated
outside of your scope.
    _privateStringValue = stringValue;
}
```

3. **Lorsque vous n'en avez plus besoin, vous devez renoncer à la propriété d'un objet que vous possédez.**

```
NSObject* object = [NSObject new]; // The retain count is now 1
[object performAction1]; // Now we are done with the object
[object release]; // Release the object
```

4. **Vous ne devez pas renoncer à la propriété d'un objet que vous ne possédez pas**

   Cela signifie que lorsque vous n'avez pas pris possession d'un objet, vous ne le libérez pas.

5. **Autoreleasepool**

   Le pool d'autorélease est un bloc de code qui libère tous les objets du bloc qui ont reçu un message d'autorisation automatique.

   Exemple:

```
@autoreleasepool {
    NSString* string = [NSString stringWithString:@"We don't own this object"];
}
```

   Nous avons créé une chaîne sans en prendre possession. La méthode `NSString stringWithString:` doit s'assurer que la chaîne est correctement désallouée une fois qu'elle n'est plus nécessaire. Avant que la méthode ne retourne, la chaîne nouvellement créée appelle la méthode autorelease pour qu'il ne soit pas nécessaire qu'elle prenne possession de la chaîne.

   Voici comment la `stringWithString:` est implémentée:

```
+ (NSString *)stringWithString:(NSString *)string {
    NSString *createdString = [[NSString alloc] initWithString:string];
    [createdString autorelease];
    return createdString;
}
```

   Il est nécessaire d'utiliser les blocs autoreleasepool car vous avez parfois des objets que vous ne possédez pas (la quatrième règle ne s'applique pas toujours).

   Le comptage automatique des références prend automatiquement en charge les règles afin que vous n'ayez pas à le faire.

Lire Gestion de la mémoire en ligne: https://riptutorial.com/fr/objective-c/topic/2751/gestion-de-la-memoire

Chapitre 16: Héritage

Syntaxe

1. @interface nom-classe-dérivé: nom-classe-base

Examples

La voiture est héritée du véhicule

Considérons un **véhicule** de classe de base et sa **voiture** de classe dérivée comme suit:

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

@interface Vehicle : NSObject

{
    NSString *vehicleName;
    NSInteger vehicleModelNo;
}

- (id)initWithName:(NSString *)name andModel:(NSInteger)modelno;
- (void)print;
@end

@implementation Vehicle

- (id)initWithName:(NSString *)name andModel:(NSInteger)modelno{
    vehicleName = name;
    vehicleModelNo = modelno;
    return self;
}

- (void)print{
    NSLog(@"Name: %@", vehicleName);
    NSLog(@"Model: %ld", vehicleModelNo);
}

@end

@interface Car : Vehicle

{
    NSString *carCompanyName;
}

- (id)initWithName:(NSString *)name andModel:(NSInteger)modelno
  andCompanyName:(NSString *)companyname;
- (void)print;

@end


@implementation Car
```

```
- (id)initWithName:(NSString *)name andModel:(NSInteger) modelno
  andCompanyName: (NSString *) companyname
  {
    vehicleName = name;
    vehicleModelNo = modelno;
    carCompanyName = companyname;
    return self;
}
- (void)print
{
    NSLog(@"Name: %@", vehicleName);
    NSLog(@"Model: %ld", vehicleModelNo);
    NSLog(@"Company: %@", carCompanyName);
}

@end

int main(int argc, const char * argv[])
{
    NSAutoreleasePool * pool = [[NSAutoreleasePool alloc] init];
    NSLog(@"Base class Vehicle Object");
    Vehicle *vehicle = [[Vehicle alloc]initWithName:@"4Wheeler" andModel:1234];
    [vehicle print];
    NSLog(@"Inherited Class Car Object");
    Car *car = [[Car alloc]initWithName:@"S-Class"
    andModel:7777 andCompanyName:@"Benz"];
    [car print];
    [pool drain];
    return 0;
}
```

Lorsque le code ci-dessus est compilé et exécuté, il produit le résultat suivant:

2016-09-29 18: 21: 03.561 Héritage [349: 303] Objet véhicule de classe de base

2016-09-29 18: 21: 03.563 Héritage [349: 303] Nom: 4Wheeler

2016-09-29 18: 21: 03.563 Héritage [349: 303] Modèle: 1234

2016-09-29 18: 21: 03.564 Héritage [349: 303] Objet de classe hérité

2016-09-29 18: 21: 03.564 Héritage [349: 303] Nom: Classe S

2016-09-29 18: 21: 03.565 Héritage [349: 303] Modèle: 7777

2016-09-29 18: 21: 03.565 Héritage [349: 303] Société: Benz

Lire Héritage en ligne: https://riptutorial.com/fr/objective-c/topic/7117/heritage

Chapitre 17: Indice

Examples

Des indices avec NSArray

Les indices peuvent être utilisés pour simplifier la récupération et la définition d'éléments dans un tableau. Étant donné le tableau suivant

```
NSArray *fruit = @[@"Apples", @"Bananas", @"Cherries"];
```

Cette ligne

```
[fruit objectAtIndex: 1];
```

Peut être remplacé par

```
fruit[1];
```

Ils peuvent également être utilisés pour définir un élément dans un tableau mutable.

```
NSMutableArray *fruit = [@[@"Apples", @"Bananas", @"Cherries"] mutableCopy];
fruit[1] = @"Blueberries";
NSLog(@"%@", fruit[1]); //Blueberries
```

Si l'index de l'indice est égal au compte du tableau, l'élément sera ajouté au tableau.

Des indices répétés peuvent être utilisés pour accéder aux éléments des tableaux imbriqués.

```
NSArray *fruit = @[@"Apples", @"Bananas", @"Cherries"];
NSArray *vegetables = @[@"Avocado", @"Beans", @"Carrots"];
NSArray *produce = @[fruit, vegetables];

NSLog(@"%@", produce[0][1]); //Bananas
```

Des indices avec NSDictionary

Les indices peuvent également être utilisés avec NSDictionary et NSMutableDictionary. Le code suivant:

```
NSMutableDictionary *myDictionary = [@{@"Foo": @"Bar"} mutableCopy];
[myDictionary setObject:@"Baz" forKey:@"Foo"];
NSLog(@"%@", [myDictionary objectForKey:@"Foo"]); // Baz
```

Peut être raccourci à:

```
NSMutableDictionary *myDictionary = [@{@"Foo": @"Bar"} mutableCopy];
myDictionary[@"Foo"] = @"Baz";
NSLog(@"%@", myDictionary[@"Foo"]); // Baz
```

Indice personnalisé

Vous pouvez ajouter un indice à vos propres classes en appliquant les méthodes requises.

Pour les index indexés (comme les tableaux):

```
- (id)objectAtIndexedSubscript:(NSUInteger)idx
- (void)setObject:(id)obj atIndexedSubscript:(NSUInteger)idx
```

Pour indexer les clés (comme les dictionnaires):

```
- (id)objectForKeyedSubscript:(id)key
- (void)setObject:(id)obj forKeyedSubscript:(id <NSCopying>)key
```

Lire Indice en ligne: https://riptutorial.com/fr/objective-c/topic/3825/indice

Chapitre 18: La gestion des erreurs

Syntaxe

- NSAssert (condition, fmtMessage, *arg1, arg2, ...*) (les arguments en italique sont facultatifs) - Affirme que la *condition est* évaluée à une valeur vraie. Si ce n'est pas le cas, l'assertion générera une exception (NSAssertionException), avec le format *fmtMessage* avec les *arguments* fournis.

Examples

Affirmer

```
@implemenetation Triangle

...

-(void)setAngles:(NSArray *)_angles {
    self.angles = _angles;

    NSAssert((self.angles.count == 3), @"Triangles must have 3 angles. Array '%@' has %i",
self.angles, (int)self.angles.count);

    CGFloat angleA = [self.angles[0] floatValue];
    CGFloat angleB = [self.angles[1] floatValue];
    CGFloat angleC = [self.angles[2] floatValue];
    CGFloat sum = (angleA + angleB + angleC);
    NSAssert((sum == M_PI), @"Triangles' angles must add up to pi radians (180°). This
triangle's angles add up to %f radians (%f°)", (float)sum, (float)(sum * (180.0f / M_PI)));
}
```

Ces assertions s'assurent que vous ne donnez pas un triangle d'angles incorrects, en lançant une exception si vous le faites. S'ils ne lançaient pas une exception, le triangle, n'étant pas du tout un triangle, pourrait provoquer des bogues dans un code ultérieur.

Gestion des erreurs et des exceptions avec try catch block

Les exceptions représentent des bogues au niveau du programmeur, comme essayer d'accéder à un élément de tableau qui n'existe pas.

Les erreurs sont des problèmes au niveau de l'utilisateur, comme essayer de charger un fichier qui n'existe pas. Parce que des erreurs sont attendues lors de l'exécution normale d'un programme.

Exemple:

```
    NSArray *inventory = @[@"Sam",
                           @"John",
                           @"Sanju"];
```

```
int selectedIndex = 3;
@try {
    NSString * name = inventory[selectedIndex];
    NSLog(@"The selected Name is: %@", name);
} @catch(NSException *theException) {
    NSLog(@"An exception occurred: %@", theException.name);
    NSLog(@"Here are some details: %@", theException.reason);
} @finally {
    NSLog(@"Executing finally block");
}
```

SORTIE:

> Une exception est survenue: NSRangeException
>
> Voici quelques détails: *** - [__ NSArrayI objectAtIndex:]: index 3 au-delà des limites [0 .. 2]
>
> Exécuter finalement le bloc

Lire La gestion des erreurs en ligne: https://riptutorial.com/fr/objective-c/topic/1459/la-gestion-des-erreurs

Chapitre 19: Les méthodes

Syntaxe

- `-` ou `+` : Le type de méthode. Instance ou classe?
- (): Où le type de retour va. Utilisez le vide si vous ne voulez rien retourner!
- Suivant est le nom de la méthode. Utilisez camelCase et rendez le nom facile à retenir et à comprendre.
- Si votre méthode a besoin de paramètres, le moment est venu! Le premier paramètre vient juste après le nom de la fonction comme ceci `:(type)parameterName` . Tous les autres paramètres se font de cette façon `parameterLabel:(type)parameterName`
- Que fait votre méthode? Mettez tout ici, dans les accolades {}!

Examples

Paramètres de méthode

Si vous voulez transmettre des valeurs à une méthode lorsqu'elle est appelée, vous utilisez des paramètres:

```
- (int)addInt:(int)intOne toInt:(int)intTwo {
  return intOne + intTwo;
}
```

Les deux points (`:`) sépare le paramètre du nom de la méthode.

Le type de paramètre est entre parenthèses `(int)` .

Le nom du paramètre suit le type de paramètre.

Créer une méthode de base

Voici comment créer une méthode de base qui enregistre "Hello World" sur la console:

```
- (void)hello {
  NSLog(@"Hello World");
}
```

Le `-` au début dénote cette méthode comme une méthode d'instance.

Le `(void)` indique le type de retour. Cette méthode ne renvoie rien, donc vous entrez `void` .

Le 'salut' est le nom de la méthode.

Tout dans le `{}` est le code exécuté lorsque la méthode est appelée.

Valeurs de retour

Lorsque vous souhaitez renvoyer une valeur à partir d'une méthode, vous devez indiquer le type que vous souhaitez renvoyer dans le premier ensemble de parenthèses.

```
- (NSString)returnHello {
  return @"Hello World";
}
```

La valeur que vous voulez renvoyer va après le mot-clé `return` ;

Méthodes de classe

Une méthode de classe est appelée sur la classe à laquelle appartient la méthode, pas une instance de celle-ci. Ceci est possible car les classes Objective-C sont également des objets. Pour désigner une méthode comme méthode de classe, changez le `-` en un `+` :

```
+ (void)hello {
  NSLog(@"Hello World");
}
```

Méthodes d'appel

Appel d'une méthode d'instance:

```
[classInstance hello];

 @interface Sample
 -(void)hello; // exposing the class Instance method
 @end

 @implementation Sample
     -(void)hello{
        NSLog(@"hello");
      }
 @end
```

Appel d'une méthode d'instance sur l'instance actuelle:

```
[self hello];

@implementation Sample

     -(void)otherMethod{
       [self hello];
     }

     -(void)hello{
        NSLog(@"hello");
      }
```

```
 @end
```

Appeler une méthode qui prend des arguments:

```
[classInstance addInt:1 toInt:2];

 @implementation Sample
     -(void)add:(NSInteger)add to:(NSInteger)to
        NSLog(@"sum = %d",(add+to));
      }
 @end
```

Appeler une méthode de classe:

```
[Class hello];

 @interface Sample
 +(void)hello; // exposing the class method
 @end

 @implementation Sample
     +(void)hello{
        NSLog(@"hello");
      }
 @end
```

Méthodes d'instance

Une méthode d'instance est une méthode disponible sur une instance particulière d'une classe, après l'instanciation de l'instance:

```
MyClass *instance = [MyClass new];
[instance someInstanceMethod];
```

Voici comment vous en définissez un:

```
@interface MyClass : NSObject

- (void)someInstanceMethod; // "-" denotes an instance method

@end

@implementation MyClass

- (void)someInstanceMethod {
    NSLog(@"Whose idea was it to have a method called \"someInstanceMethod\"?");
}

@end
```

Passage du paramètre valeur par passage

En passant par la valeur du paramètre passant à une méthode, la valeur réelle du paramètre est

copiée dans la valeur du paramètre formel. La valeur réelle du paramètre ne changera donc pas après le retour de la fonction appelée.

```
@interface SwapClass : NSObject

-(void) swap:(NSInteger)num1 andNum2:(NSInteger)num2;

@end

@implementation SwapClass

-(void) num:(NSInteger)num1 andNum2:(NSInteger)num2{
    int temp;
    temp = num1;
    num1 = num2;
    num2 = temp;
}
@end
```

Appeler les méthodes:

```
NSInteger a = 10, b =20;
SwapClass *swap = [[SwapClass alloc]init];
NSLog(@"Before calling swap: a=%d,b=%d",a,b);
[swap num:a andNum2:b];
NSLog(@"After calling swap: a=%d,b=%d",a,b);
```

Sortie:

```
2016-07-30 23:55:41.870 Test[5214:81162] Before calling swap: a=10,b=20
2016-07-30 23:55:41.871 Test[5214:81162] After calling swap: a=10,b=20
```

Pass par référence paramètre passant

En passant par référence de paramètre passant à une méthode, l'adresse du paramètre réel est passée au paramètre formel. La valeur réelle du paramètre sera donc modifiée après le retour de la fonction appelée.

```
@interface SwapClass : NSObject

-(void) swap:(int)num1 andNum2:(int)num2;

@end

@implementation SwapClass

-(void) num:(int*)num1 andNum2:(int*)num2{
    int temp;
    temp = *num1;
    *num1 = *num2;
    *num2 = temp;
}
@end
```

Appeler les méthodes:

```
int a = 10, b =20;
SwapClass *swap = [[SwapClass alloc]init];
NSLog(@"Before calling swap: a=%d,b=%d",a,b);
[swap num:&a andNum2:&b];
NSLog(@"After calling swap: a=%d,b=%d",a,b);
```

Sortie:

```
2016-07-31 00:01:47.067 Test[5260:83491] Before calling swap: a=10,b=20
2016-07-31 00:01:47.070 Test[5260:83491] After calling swap: a=20,b=10
```

Lire Les méthodes en ligne: https://riptutorial.com/fr/objective-c/topic/1457/les-methodes

Chapitre 20: Macros prédéfinies

Introduction

ANSI C définit un certain nombre de macros. Bien que chacun soit disponible pour la programmation, les macros prédéfinies ne doivent pas être directement modifiées.

Syntaxe

1. **DATE** La date actuelle sous la forme d'un littéral de caractère au format "MMM JJ AAAA"
2. **TIME** L'heure actuelle sous la forme d'un littéral de caractère au format "HH: MM: SS"
3. **FILE** Il contient le nom de fichier actuel en tant que chaîne littérale.
4. **LINE** Il contient le numéro de ligne actuel sous forme de constante décimale.
5. **STDC** Défini comme 1 lorsque le compilateur est conforme à la norme ANSI.

Examples

Macros prédéfinies

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

int main()
{
   NSLog(@"File :%s\n", __FILE__ );
   NSLog(@"Date :%s\n", __DATE__ );
   NSLog(@"Time :%s\n", __TIME__ );
   NSLog(@"Line :%d\n", __LINE__ );
   NSLog(@"ANSI :%d\n", __STDC__ );

   return 0;
}
```

Lorsque le code ci-dessus dans un fichier main.m est compilé et exécuté, il produit le résultat suivant:

```
2013-09-14 04:46:14.859 demo[20683] File :main.m
2013-09-14 04:46:14.859 demo[20683] Date :Sep 14 2013
2013-09-14 04:46:14.859 demo[20683] Time :04:46:14
2013-09-14 04:46:14.859 demo[20683] Line :8
2013-09-14 04:46:14.859 demo[20683] ANSI :1
```

Lire Macros prédéfinies en ligne: https://riptutorial.com/fr/objective-c/topic/8254/macros-predefinies

Chapitre 21: Multi-Threading

Examples

Créer un fil simple

Le moyen le plus simple de créer un thread est d'appeler un sélecteur "en arrière-plan". Cela signifie qu'un nouveau thread est créé pour exécuter le sélecteur. L'objet récepteur peut être n'importe quel objet, pas seulement `self` , mais il doit répondre au sélecteur donné.

```
- (void)createThread {
    [self performSelectorInBackground:@selector(threadMainWithOptionalArgument:)
                           withObject:someObject];
}

- (void)threadMainWithOptionalArgument:(id)argument {
    // To avoid memory leaks, the first thing a thread method needs to do is
    // create a new autorelease pool, either manually or via "@autoreleasepool".
    @autoreleasepool {
        // The thread code should be here.
    }
}
```

Créer des threads plus complexes

L'utilisation d'une sous-classe de `NSThread` permet la mise en œuvre de threads plus complexes (par exemple, pour permettre de transmettre plus d'arguments ou d'encapsuler toutes les méthodes d'assistance associées dans une classe). En outre, l'instance `NSThread` peut être enregistrée dans une propriété ou une variable et peut être interrogée sur son état actuel (qu'il soit toujours en cours d'exécution).

La classe `NSThread` prend en charge une méthode appelée `cancel` qui peut être appelée à partir de n'importe quel thread, qui définit ensuite la propriété `cancelled` sur `YES` de manière sûre. L'implémentation du thread peut interroger (et / ou observer) la propriété `cancelled` et quitter sa méthode `main` . Cela peut être utilisé pour arrêter normalement un thread de travail.

```
// Create a new NSThread subclass
@interface MyThread : NSThread

// Add properties for values that need to be passed from the caller to the new
// thread. Caller must not modify these once the thread is started to avoid
// threading issues (or the properties must be made thread-safe using locks).
@property NSInteger someProperty;

@end

@implementation MyThread

- (void)main
{
    @autoreleasepool {
```

```
        // The main thread method goes here
        NSLog(@"New thread. Some property: %ld", (long)self.someProperty);
    }
}

@end


MyThread *thread = [[MyThread alloc] init];
thread.someProperty = 42;
[thread start];
```

Stockage local

Chaque thread a accès à un dictionnaire mutable local au thread actuel. Cela permet de mettre en cache les informations de manière simple sans avoir besoin de les verrouiller, car chaque thread a son propre dictionnaire de mutabilité dédié:

```
NSMutableDictionary *localStorage = [NSThread currentThread].threadDictionary;
localStorage[someKey] = someValue;
```

Le dictionnaire est automatiquement libéré à la fin du thread.

Lire Multi-Threading en ligne: https://riptutorial.com/fr/objective-c/topic/3350/multi-threading

Chapitre 22: NSArray

Syntaxe

- NSArray * mots; // Déclarer un tableau immuable
- NSMutableArray * mots; // Déclarer un tableau mutable
- NSArray * words = [NSArray arrayWithObjects: @ "one", @ "two", nil]; // Syntaxe d'initialisation de tableau
- NSArray * words = @ [@ "list", @ "de", @ "mots", @ 123, @ 3.14]; // Déclaration des littéraux de tableau
- NSArray * stringArray = [NSArray arrayWithObjects: [NSMutableArray array], [NSMutableArray array], [NSMutableArray array], nil]; // Création de tableaux multidimentionnels

Examples

Création de tableaux

Création de tableaux immuables:

```
NSArray *myColors = [NSArray arrayWithObjects: @"Red", @"Green", @"Blue", @"Yellow", nil];

// Using the array literal syntax:
NSArray *myColors = @[@"Red", @"Green", @"Blue", @"Yellow"];
```

Pour les tableaux mutables, voir NSMutableArray .

Déterminer le nombre d'éléments dans un tableau

```
NSArray *myColors = [NSArray arrayWithObjects: @"Red", @"Green", @"Blue", @"Yellow", nil];
NSLog (@"Number of elements in array = %lu", [myColors count]);
```

Accès aux éléments

```
NSArray *myColors = @[@"Red", @"Green", @"Blue", @"Yellow"];
// Preceding is the preferred equivalent to [NSArray arrayWithObjects:...]
```

Obtenir un seul article

La méthode `objectAtIndex:` fournit un objet unique. Le premier objet d'un `NSArray` est l'index 0. Comme un `NSArray` peut être homogène (avec différents types d'objets), le type de retour est `id` ("any object"). (Un `id` peut être assigné à une variable de n'importe quel autre type d'objet.) Il est important de noter que `NSArray` ne peut contenir que des objets. Ils ne peuvent pas contenir de valeurs comme `int` .

```
NSUInteger idx = 2;
NSString *color = [myColors objectAtIndex:idx];
// color now points to the string @"Green"
```

Clang fournit une meilleure syntaxe d'indice dans le cadre de sa fonctionnalité de littéraux de tableau :

```
NSString *color = myColors[idx];
```

Les deux lancent une exception si l'index transmis est inférieur à 0 ou supérieur à `count - 1` .

Premier et dernier article

```
NSString *firstColor = myColors.firstObject;
NSString *lastColor = myColors.lastObject;
```

Les `firstObject` et `lastObject` sont des propriétés calculées et renvoient à `nil` plutôt que de planter pour des tableaux vides. Pour les tableaux à élément unique, ils renvoient le même objet. Bien que la méthode `firstObject` n'ait pas été introduite dans `NSArray` avant iOS 4.0.

```
NSArray *empty = @[]
id notAnObject = empty.firstObject;    // Returns `nil`
id kaboom = empty[0];    // Crashes; index out of bounds
```

Filtrage de tableaux avec des prédicats

```
NSArray *array = [NSArray arrayWithObjects:@"Nick", @"Ben", @"Adam", @"Melissa", nil];

NSPredicate *aPredicate = [NSPredicate predicateWithFormat:@"SELF beginswith[c] 'a'"];
NSArray *beginWithA = [array filteredArrayUsingPredicate:bPredicate];
     // beginWithA contains { @"Adam" }.

NSPredicate *ePredicate = [NSPredicate predicateWithFormat:@"SELF contains[c] 'e'"];
[array filterUsingPredicate:ePredicate];
    // array now contains { @"Ben", @"Melissa" }
```

Plus à propos

NSPredicate :

Doc Apple: NSPredicate

Conversion de NSArray en NSMutableArray pour permettre la modification

```
NSArray *myColors = [NSArray arrayWithObjects: @"Red", @"Green", @"Blue", @"Yellow", nil];

// Convert myColors to mutable
NSMutableArray *myColorsMutable = [myColors mutableCopy];
```

Tri du tableau avec des objets personnalisés

Comparer la méthode

Soit vous implémentez une méthode de comparaison pour votre objet:

```
- (NSComparisonResult)compare:(Person *)otherObject {
    return [self.birthDate compare:otherObject.birthDate];
}

NSArray *sortedArray = [drinkDetails sortedArrayUsingSelector:@selector(compare:)];
```

NSSortDescriptor

```
NSSortDescriptor *sortDescriptor;
sortDescriptor = [[NSSortDescriptor alloc] initWithKey:@"birthDate"
                                             ascending:YES];
NSArray *sortDescriptors = [NSArray arrayWithObject:sortDescriptor];
NSArray *sortedArray = [drinkDetails sortedArrayUsingDescriptors:sortDescriptors];
```

Vous pouvez facilement trier par plusieurs clés en en ajoutant plusieurs dans le tableau. L'utilisation de méthodes de comparaison personnalisées est également possible. Regardez la documentation .

Blocs

```
NSArray *sortedArray;
sortedArray = [drinkDetails sortedArrayUsingComparator:^NSComparisonResult(id a, id b) {
    NSDate *first = [(Person*)a birthDate];
    NSDate *second = [(Person*)b birthDate];
    return [first compare:second];
}];
```

Performance

Les `-compare:` et les méthodes basées sur les blocs seront en général beaucoup plus rapides qu'en utilisant `NSSortDescriptor` car ce dernier repose sur KVC. Le principal avantage de la méthode `NSSortDescriptor` est qu'elle permet de définir l'ordre de tri en utilisant des données plutôt que du code, ce qui facilite la configuration, par exemple, pour que les utilisateurs puissent trier un `NSTableView` en cliquant sur la ligne d'en-tête.

Conversion entre les ensembles et les tableaux

```
NSSet *set = [NSSet set];
NSArray *array = [NSArray array];

NSArray *fromSet = [set allObjects];
```

```
NSSet *fromArray = [NSSet setWithArray:array];
```

Inverser un tableau

```
NSArray *reversedArray = [myArray.reverseObjectEnumerator allObjects];
```

En boucle à travers

```
NSArray *myColors = @[@"Red", @"Green", @"Blue", @"Yellow"];

// Fast enumeration
// myColors cannot be modified inside the loop
for (NSString *color in myColors) {
    NSLog(@"Element %@", color);
}

// Using indices
for (NSUInteger i = 0; i < myColors.count; i++) {
    NSLog(@"Element %d = %@", i, myColors[i]);
}

// Using block enumeration
[myColors enumerateObjectsUsingBlock:^(id obj, NSUInteger idx, BOOL * stop) {
    NSLog(@"Element %d = %@", idx, obj);

    // To abort use:
    *stop = YES
}];

// Using block enumeration with options
[myColors enumerateObjectsWithOptions:NSEnumerationReverse usingBlock:^(id obj, NSUInteger
idx, BOOL * stop) {
    NSLog(@"Element %d = %@", idx, obj);
}];
```

Utiliser des génériques

Pour plus de sécurité, nous pouvons définir le type d'objet que contient le tableau:

```
NSArray<NSString *> *colors = @[@"Red", @"Green", @"Blue", @"Yellow"];
NSMutableArray<NSString *> *myColors = [NSMutableArray arrayWithArray:colors];
[myColors addObject:@"Orange"]; // OK
[myColors addObject:[UIColor purpleColor]]; // "Incompatible pointer type" warning
```

Il convient de noter que cela est vérifié uniquement au moment de la compilation.

Enumérer à l'aide de blocs

```
NSArray *myColors = @[@"Red", @"Green", @"Blue", @"Yellow"];
[myColors enumerateObjectsUsingBlock:^(id obj, NSUInteger idx, BOOL *stop) {
    NSLog(@"enumerating object %@ at index %lu", obj, idx);
}];
```

En définissant le paramètre d' `stop` sur `YES` vous pouvez indiquer qu'une autre énumération n'est pas nécessaire. pour ce faire, il suffit de définir `&stop = YES` .

NSEnumerationOptions
Vous pouvez énumérer le tableau en sens inverse et / ou simultanément:

```
[myColors enumerateObjectsWithOptions:NSEnumerationConcurrent | NSEnumerationReverse
                              usingBlock:^(id obj, NSUInteger idx, BOOL *stop) {
                                  NSLog(@"enumerating object %@ at index %lu", obj, idx);
                              }];
```

Sous-ensemble énumérant du tableau

```
NSIndexSet *indexSet = [NSIndexSet indexSetWithIndexesInRange:NSMakeRange(1, 1)];
[myColors enumerateObjectsAtIndexes:indexSet
                            options:kNilOptions
                         usingBlock:^(id obj, NSUInteger idx, BOOL *stop) {
                                NSLog(@"enumerating object %@ at index %lu", obj, idx);
                            }];
```

Comparaison de tableaux

Les tableaux peuvent être comparés en **termes d'** égalité avec la méthode **isEqualToArray:** qui renvoie **YES** lorsque les deux tableaux ont le même nombre d'éléments et que chaque paire **réussit** une comparaison **isEqual :** .

```
NSArray *germanMakes = @[@"Mercedes-Benz", @"BMW", @"Porsche",
                       @"Opel", @"Volkswagen", @"Audi"];
NSArray *sameGermanMakes = [NSArray arrayWithObjects:@"Mercedes-Benz",
                          @"BMW", @"Porsche", @"Opel",
                          @"Volkswagen", @"Audi", nil];

if ([germanMakes isEqualToArray:sameGermanMakes]) {
    NSLog(@"Oh good, literal arrays are the same as NSArrays");
}
```

La chose importante est que chaque paire doit passer le test isEqual: Pour les objets personnalisés, cette méthode doit être implémentée. Elle existe dans le protocole NSObject.

Ajouter des objets à NSArray

```
NSArray *a = @[@1];
a = [a arrayByAddingObject:@2];
a = [a arrayByAddingObjectsFromArray:@[@3, @4, @5]];
```

Ces méthodes sont optimisées pour recréer le nouveau tableau très efficacement, généralement sans avoir à détruire le tableau d'origine ou même à allouer plus de mémoire.

Lire NSArray en ligne: https://riptutorial.com/fr/objective-c/topic/736/nsarray

Chapitre 23: NSArray

Examples

Création d'instances NSArray

```
NSArray *array1 = [NSArray arrayWithObjects:@"one", @"two", @"three", nil];
NSArray *array2 = @[@"one", @"two", @"three"];
```

Tri des tableaux

Les méthodes les plus flexibles pour trier un tableau sont la méthode sortArrayUsingComparator:. Ceci accepte un **bloc ^ NSComparisonResult (id obj1, id obj2)** .

```
 Return Value            Description
 NSOrderedAscending      obj1 comes before obj2
 NSOrderedSame           obj1 and obj2 have no order
 NSOrderedDescending     obj1 comes after obj2
```

Exemple:

```
   NSArray *categoryArray = @[@"Apps", @"Music", @"Songs",
                      @"iTunes", @"Books", @"Videos"];

    NSArray *sortedArray = [categoryArray sortedArrayUsingComparator:
^NSComparisonResult(id obj1, id obj2) {
    if ([obj1 length] < [obj2 length]) {
        return NSOrderedAscending;
    } else if ([obj1 length] > [obj2 length]) {
        return NSOrderedDescending;
    } else {
        return NSOrderedSame;
    }
  }];

 NSLog(@"%@", sortedArray);
```

Filtrer NSArray et NSMutableArray

```
NSMutableArray *array =
    [NSMutableArray arrayWithObjects:@"Ken", @"Tim", @"Chris", @"Steve",@"Charlie",@"Melissa",
nil];

NSPredicate *bPredicate =
    [NSPredicate predicateWithFormat:@"SELF beginswith[c] 'c'"];
NSArray *beginWithB =
    [array filteredArrayUsingPredicate:bPredicate];
// beginWith "C" contains { @"Chris", @"Charlie" }.

NSPredicate *sPredicate =
    [NSPredicate predicateWithFormat:@"SELF contains[c] 'a'"];
```

```
[array filterUsingPredicate:sPredicate];
// array now contains { @"Charlie", @"Melissa" }
```

Lire NSArray en ligne: https://riptutorial.com/fr/objective-c/topic/1181/nsarray

Chapitre 24: NSAttributedString

Examples

Création d'une chaîne comportant un crénage personnalisé (espacement des lettres) editshare

`NSAttributedString` (et son frère modifiable `NSMutableAttributedString`) vous permet de créer des chaînes complexes dans leur apparence pour l'utilisateur.

Une application courante consiste à utiliser cette option pour afficher une chaîne et ajouter un crénage / espacement des lettres personnalisé.

Cela se ferait comme suit (où label est un `UILabel`), donnant un crénage différent pour le mot "kerning"

```
NSMutableAttributedString *attributedString;
attributedString = [[NSMutableAttributedString alloc] initWithString:@"Apply kerning"];
[attributedString addAttribute:NSKernAttributeName value:@5 range:NSMakeRange(6, 7)];
[label setAttributedText:attributedString];
```

Créer une chaîne avec du texte barré

```
NSMutableAttributedString *attributeString = [[NSMutableAttributedString alloc]
initWithString:@"Your String here"];
[attributeString addAttribute:NSStrikethroughStyleAttributeName
                    value:@2
                    range:NSMakeRange(0, [attributeString length])];
```

Utilisation de l'énumération sur des attributs dans une chaîne et souligné la partie de la chaîne

```
 NSMutableDictionary *attributesDictionary = [NSMutableDictionary dictionary];
 [attributesDictionary setObject:[UIFont systemFontOfSize:14] forKey:NSFontAttributeName];
 //[attributesDictionary setObject:[UIColor redColor] forKey:NSForegroundColorAttributeName];
 NSMutableAttributedString *attributedString = [[NSMutableAttributedString
alloc]initWithString:@"Google www.google.com link" attributes:attributesDictionary];

 [attributedString enumerateAttribute:(NSString *) NSFontAttributeName
                              inRange:NSMakeRange(0, [attributedString length])

options:NSAttributedStringEnumerationLongestEffectiveRangeNotRequired
                           usingBlock:^(id value, NSRange range, BOOL *stop) {
                               NSLog(@"Attribute: %@, %@", value, NSStringFromRange(range));
                              }];

  NSMutableAttributedString *attributedStr = [[NSMutableAttributedString alloc]
initWithString:@"www.google.com "];

  [attributedString addAttribute:NSUnderlineStyleAttributeName
```

```
                        value:[NSNumber numberWithInt:NSUnderlineStyleDouble]
                        range:NSMakeRange(7, attributedStr.length)];

  [attributedString addAttribute:NSForegroundColorAttributeName
                        value:[UIColor blueColor]
                        range:NSMakeRange(6,attributedStr.length)];

  _attriLbl.attributedText = attributedString;//_attriLbl (of type UILabel) added in
storyboard
```

Sortie:

Google www.google.com link

Comment vous créez une chaîne attribuée à trois couleurs.

```
 NSMutableAttributedString * string = [[NSMutableAttributedString alloc]
initWithString:@"firstsecondthird"];
[string addAttribute:NSForegroundColorAttributeName value:[UIColor redColor]
range:NSMakeRange(0,5)];
[string addAttribute:NSForegroundColorAttributeName value:[UIColor greenColor]
range:NSMakeRange(5,6)];
[string addAttribute:NSForegroundColorAttributeName value:[UIColor blueColor]
range:NSMakeRange(11,5)];
```

Plage: commencer à terminer la chaîne

Ici, nous avons la première chaîne de troisième rang, donc nous avons d'abord défini la plage (0,5), de sorte que du premier caractère au cinquième caractère, elle sera affichée en vert.

Lire NSAttributedString en ligne: https://riptutorial.com/fr/objective-c/topic/3725/nsattributedstring

Chapitre 25: NSCache

Examples

NSCache

Vous l'utilisez de la même manière que vous utiliseriez NSMutableDictionary. La différence est que lorsque NSCache détecte une pression de mémoire excessive (c.-à-d. Qu'il met trop de valeurs en cache), il libère certaines de ces valeurs pour faire de la place.

Si vous pouvez recréer ces valeurs à l'exécution (en les téléchargeant depuis Internet, en effectuant des calculs, etc.), NSCache peut répondre à vos besoins. Si les données ne peuvent pas être recréées (par exemple, elles sont sensibles au facteur temps, etc.), vous ne devez pas les stocker dans un NSCache, car elles y seront détruites.

Lire NSCache en ligne: https://riptutorial.com/fr/objective-c/topic/8257/nscache

Chapitre 26: NSCalendar

Examples

Informations locales sur le système

`+currentCalendar` renvoie le calendrier logique pour l'utilisateur actuel.

```
    NSCalendar *calender = [NSCalendar currentCalendar];
    NSLog(@"%@",calender);
```

`+autoupdatingCurrentCalendar` renvoie le calendrier logique actuel pour l'utilisateur actuel.

```
NSCalendar *calender = [NSCalendar autoupdatingCurrentCalendar];
NSLog(@"%@",calender);
```

Initialiser un calendrier

`- initWithCalendarIdentifier:` Initialise un objet NSCalendar nouvellement alloué pour le calendrier spécifié par un identifiant donné.

```
NSCalendar *calender = [[NSCalendar alloc]initWithCalendarIdentifier:@"gregorian"];
NSLog(@"%@",calender);
```

`- setFirstWeekday:` définit l'index du premier jour de la semaine pour le récepteur.

```
NSCalendar *calender = [NSCalendar autoupdatingCurrentCalendar];
[calender setFirstWeekday:1];
NSLog(@"%d",[calender firstWeekday]);
```

`- setLocale:` définit les paramètres régionaux du récepteur.

```
NSCalendar *calender = [NSCalendar autoupdatingCurrentCalendar];
[calender setLocale:[NSLocale currentLocale]];
NSLog(@"%@",[calender locale]);
```

`- setMinimumDaysInFirstWeek:` définit le nombre minimum de jours dans la première semaine du destinataire.

```
NSCalendar *calender = [NSCalendar autoupdatingCurrentCalendar];
[calender setMinimumDaysInFirstWeek:7];
NSLog(@"%d",[calender minimumDaysInFirstWeek]);
```

`- setTimeZone:` définit le fuseau horaire du récepteur.

```
NSCalendar *calender = [NSCalendar autoupdatingCurrentCalendar];
[calender setTimeZone:[NSTimeZone timeZoneForSecondsFromGMT:0]];
```

```
NSLog(@"%@",[calender timeZone]);
```

Calculs calendaires

`- components:fromDate:` Retourne un objet NSDateComponents contenant une date donnée décomposée en composants spécifiés

```
NSCalendar *calender = [NSCalendar autoupdatingCurrentCalendar];
[calender setTimeZone:[NSTimeZone timeZoneForSecondsFromGMT:0]];
NSLog(@"%@",[calender components:NSCalendarUnitDay fromDate:[NSDate date]]);
NSLog(@"%@",[calender components:NSCalendarUnitYear fromDate:[NSDate date]]);
NSLog(@"%@",[calender components:NSCalendarUnitMonth fromDate:[NSDate date]]);
```

`- components:fromDate:toDate:options:` Retourne, en tant qu'objet NSDateComponents à l'aide de composants spécifiés, la différence entre deux dates fournies.

```
 NSCalendar *calender = [NSCalendar autoupdatingCurrentCalendar];
[calender setTimeZone:[NSTimeZone timeZoneForSecondsFromGMT:0]];
NSLog(@"%@",[calender components:NSCalendarUnitYear fromDate:[NSDate
dateWithTimeIntervalSince1970:0] toDate:[NSDate dateWithTimeIntervalSinceNow:18000]
options:NSCalendarWrapComponents]);
```

`- dateByAddingComponents:toDate:options:` Retourne un nouvel objet NSDate représentant le temps absolu calculé en ajoutant des composants donnés à une date donnée.

```
NSCalendar *calender = [NSCalendar autoupdatingCurrentCalendar];
NSDateComponents *dateComponent = [[NSDateComponents alloc]init];
[dateComponent setYear:10];
NSLog(@"%@",[calender dateByAddingComponents:dateComponent toDate:[NSDate
dateWithTimeIntervalSinceNow:0] options:NSCalendarWrapComponents] );
```

`- dateFromComponents:` renvoie un nouvel objet NSDate représentant le temps absolu calculé à partir de composants donnés.

```
NSCalendar *calender = [NSCalendar autoupdatingCurrentCalendar];
NSDateComponents *dateComponent = [[NSDateComponents alloc]init];
[dateComponent setYear:2020];
NSLog(@"%@",[calender dateFromComponents:dateComponent]);
```

Lire NSCalendar en ligne: https://riptutorial.com/fr/objective-c/topic/2903/nscalendar

Chapitre 27: NSData

Examples

Créer

De NSString:

```
NSString *str = @"Hello world";
NSData *data = [str dataUsingEncoding:NSUTF8StringEncoding];
```

De Int:

```
int i = 1;
NSData *data = [NSData dataWithBytes: &i length: sizeof(i)];
```

Vous pouvez également utiliser les méthodes suivantes:

```
+ dataWithContentsOfURL:
+ dataWithContentsOfURL:options:error:
+ dataWithData:
- initWithBase64EncodedData:options:
- initWithBase64EncodedString:options:
- initWithBase64Encoding:
- initWithBytesNoCopy:length:
- initWithBytesNoCopy:length:deallocator:
- initWithBytesNoCopy:length:freeWhenDone:
- initWithContentsOfFile:
- initWithContentsOfFile:options:error:
- initWithContentsOfMappedFile:
- initWithContentsOfURL:
- initWithContentsOfURL:options:error:
- initWithData:
```

Obtenez NSData lengh

```
NSString *filePath = [[NSFileManager defaultManager] pathForRessorce: @"data" ofType:@"txt"];
NSData *data = [NSData dataWithContentsOfFile:filePath];
int len = [data length];
```

Encodage et décodage d'une chaîne à l'aide de NSData Base64

Codage

```
 //Create a Base64 Encoded NSString Object
 NSData *nsdata = [@"iOS Developer Tips encoded in Base64"
dataUsingEncoding:NSUTF8StringEncoding];

 // Get NSString from NSData object in Base64
```

```
 NSString *base64Encoded = [nsdata base64EncodedStringWithOptions:0];
 // Print the Base64 encoded string
 NSLog(@"Encoded: %@", base64Encoded);
```

Décodage:

```
// NSData from the Base64 encoded str
NSData *nsdataFromBase64String = [[NSData alloc]initWithBase64EncodedString:base64Encoded
options:0];

// Decoded NSString from the NSData
NSString *base64Decoded = [[NSString alloc] initWithData:nsdataFromBase64String
encoding:NSUTF8StringEncoding];
NSLog(@"Decoded: %@", base64Decoded);
```

Chaîne NSData et hexadécimale

Obtenir NSData à partir d'une chaîne hexadécimale

```
+ (NSData *)dataFromHexString:(NSString *)string
{
    string = [string lowercaseString];
    NSMutableData *data= [NSMutableData new];
    unsigned char whole_byte;
    char byte_chars[3] = {'\0','\0','\0'};
    int i = 0;
    int length = (int) string.length;
    while (i < length-1) {
        char c = [string characterAtIndex:i++];
        if (c < '0' || (c > '9' && c < 'a') || c > 'f')
            continue;
        byte_chars[0] = c;
        byte_chars[1] = [string characterAtIndex:i++];
        whole_byte = strtol(byte_chars, NULL, 16);
        [data appendBytes:&whole_byte length:1];
    }
    return data;
}
```

Obtenir une chaîne hexadécimale à partir de données:

```
+ (NSString *)hexStringForData:(NSData *)data
{
    if (data == nil) {
        return nil;
    }

    NSMutableString *hexString = [NSMutableString string];

    const unsigned char *p = [data bytes];

    for (int i=0; i < [data length]; i++) {
        [hexString appendFormat:@"%02x", *p++];
    }

    return hexString;
```

```
}
```

Lire NSData en ligne: https://riptutorial.com/fr/objective-c/topic/1532/nsdata

Chapitre 28: NSDate

Remarques

`NSDate` est un type de valeur très simple, représentant un moment exact du temps universel. Il ne contient pas d'informations sur les fuseaux horaires, l'heure d'été, les calendriers ou les paramètres régionaux.

`NSDate` est vraiment un wrapper immuable autour d'un `NSTimeInterval` qui est un `double` . Il n'y a pas de sous-classe mutable, comme avec d'autres types de valeurs dans Foundation.

Examples

Créer un NSDate

La classe `NSDate` fournit des méthodes pour créer des objets `NSDate` correspondant à une date et une heure données. Un `NSDate` peut être initialisé à l'aide de l'initialiseur désigné, qui:

> Retourne un objet `NSDate` initialisé par rapport à 00:00:00 UTC le 1er janvier 2001 d'un nombre de secondes donné.

```
NSDate *date = [[NSDate alloc] initWithTimeIntervalSinceReferenceDate:100.0];
```

`NSDate` fournit également un moyen simple de créer un `NSDate` égal à la date et à l'heure actuelles:

```
NSDate *now = [NSDate date];
```

Il est également possible de créer un `NSDate` un nombre de secondes donné à partir de la date et de l'heure actuelles:

```
NSDate *tenSecondsFromNow = [NSDate dateWithTimeIntervalSinceNow:10.0];
```

Comparaison de date

Il y a 4 méthodes pour comparer `NSDate` dans Objective-C:

- `- (BOOL)isEqualToDate:(NSDate *)anotherDate`
- `- (NSDate *)earlierDate:(NSDate *)anotherDate`
- `- (NSDate *)laterDate:(NSDate *)anotherDate`
- `- (NSComparisonResult)compare:(NSDate *)anotherDate`

Considérez l'exemple suivant en utilisant 2 dates, `NSDate date1 = July 7, 2016` et `NSDate date2 = July 2, 2016` :

```
NSDateComponents *comps1 = [[NSDateComponents alloc]init];
comps.year = 2016;
```

```
comps.month = 7;
comps.day = 7;

NSDateComponents *comps2 = [[NSDateComponents alloc]init];
    comps.year = 2016;
    comps.month = 7;
    comps.day = 2;

NSDate* date1 = [calendar dateFromComponents:comps1]; //Initialized as July 7, 2016
NSDate* date2 = [calendar dateFromComponents:comps2]; //Initialized as July 2, 2016
```

Maintenant que les `NSDate` sont créés, ils peuvent être comparés:

```
if ([date1 isEqualToDate:date2]) {
    //Here it returns false, as both dates are not equal
}
```

Nous pouvons également utiliser les `earlierDate:` et `laterDate:` de la classe `NSDate` :

```
NSDate *earlierDate = [date1 earlierDate:date2];//Returns the earlier of 2 dates. Here
earlierDate will equal date2.
NSDate *laterDate = [date1 laterDate:date2];//Returns the later of 2 dates. Here laterDate
will equal date1.
```

Enfin, nous pouvons utiliser la `NSDate compare:` :

```
NSComparisonResult result = [date1 compare:date2];
    if (result == NSOrderedAscending) {
        //Fails
        //Comes here if date1 is earlier than date2. In our case it will not come here.
    }else if (result == NSOrderedSame){
        //Fails
        //Comes here if date1 is the same as date2. In our case it will not come here.
    }else{//NSOrderedDescending

        //Succeeds
        //Comes here if date1 is later than date2. In our case it will come here
    }
```

Convertir NSDate composé d'heures et de minutes (uniquement) en un NSDate complet

Il y a beaucoup de cas où l'on a créé un NSDate à partir d'un format heure et minute, à savoir: 08:12

L'inconvénient de cette situation est que votre NSDate est presque complètement "nu" et que vous devez créer: jour, mois, année, seconde et fuseau horaire pour que cet objet "joue" avec d'autres types de NSDate.

Pour l'exemple, supposons que hourAndMinute est le type NSDate composé du format heure et minute:

```
NSDateComponents *hourAndMintuteComponents = [calendar components:NSCalendarUnitHour |
```

```
NSCalendarUnitMinute
                                                        fromDate:hourAndMinute];
NSDateComponents *componentsOfDate = [[NSCalendar currentCalendar]
components:NSCalendarUnitDay | NSCalendarUnitMonth | NSCalendarUnitYear
                                                                fromDate:[NSDate date]];

NSDateComponents *components = [[NSDateComponents alloc] init];
[components setDay: componentsOfDate.day];
[components setMonth: componentsOfDate.month];
[components setYear: componentsOfDate.year];
[components setHour: [hourAndMintuteComponents hour]];
[components setMinute: [hourAndMintuteComponents minute]];
[components setSecond: 0];
[calendar setTimeZone: [NSTimeZone defaultTimeZone]];

NSDate *yourFullNSDateObject = [calendar dateFromComponents:components];
```

Maintenant, votre objet est le contraire total d'être "nu".

Conversion de NSDate en NSString

Si ww a un objet NSDate, et que nous voulons le convertir en NSString. Il existe différents types de chaînes de dates. Comment pouvons-nous faire cela? C'est très simple. Juste 3 étapes.

1. Créer un objet NSDateFormatter

```
NSDateFormatter *dateFormatter = [[NSDateFormatter alloc] init];
```

2. Définissez le format de date dans lequel vous souhaitez que votre chaîne.

```
dateFormatter.dateFormat = @"yyyy-MM-dd 'at' HH:mm";
```

3. Maintenant, obtenez la chaîne formatée

```
NSDate *date = [NSDate date]; // your NSDate object
NSString *dateString = [dateFormatter stringFromDate:date];
```

Cela donnera quelque chose comme ceci: `2001-01-02 at 13:00`

Remarque:

La création d'une instance `NSDateFormatter` est une opération coûteuse. Il est donc recommandé de la créer une fois et de la réutiliser si possible.

Lire NSDate en ligne: https://riptutorial.com/fr/objective-c/topic/1981/nsdate

Chapitre 29: NSDictionary

Examples

Créer

```
NSDictionary *dict = [[NSDictionary alloc] initWithObjectsAndKeys:@"value1", @"key1",
@"value2", @"key2", nil];
```

ou

```
NSArray *keys = [NSArray arrayWithObjects:@"key1", @"key2", nil];
NSArray *objects = [NSArray arrayWithObjects:@"value1", @"value2", nil];
NSDictionary *dictionary = [NSDictionary dictionaryWithObjects:objects
                                                       forKeys:keys];
```

ou en utilisant la syntaxe littérale appropriée

```
NSDictionary *dict = @{@"key": @"value", @"nextKey": @"nextValue"};
```

NSDictionary vers NSArray

```
NSDictionary *myDictionary = [[NSDictionary alloc] initWithObjectsAndKeys:@"value1", @"key1",
@"value2", @"key2", nil];
NSArray *copiedArray = myDictionary.copy;
```

Obtenir des clés:

```
NSArray *keys = [myDictionary allKeys];
```

Obtenir des valeurs:

```
NSArray *values = [myDictionary allValues];
```

NSDictionary à NSData

```
NSDictionary *myDictionary = [[NSDictionary alloc] initWithObjectsAndKeys:@"value1", @"key1",
@"value2", @"key2", nil];
NSData *myData = [NSKeyedArchiver archivedDataWithRootObject:myDictionary];
```

Chemin de réserve:

```
NSDictionary *myDictionary = (NSDictionary*) [NSKeyedUnarchiver
unarchiveObjectWithData:myData];
```

NSDictionary à JSON

```
NSDictionary *myDictionary = [[NSDictionary alloc] initWithObjectsAndKeys:@"value1", @"key1",
@"value2", @"key2", nil];

NSMutableDictionary *mutableDictionary = [myDictionary mutableCopy];
NSData *data = [NSJSONSerialization dataWithJSONObject:myDictionary
options:NSJSONWritingPrettyPrinted error:nil];
NSString *jsonString = [[NSString alloc] initWithData:data encoding:NSUTF8StringEncoding];
```

Enumération par blocs

L'énumération des dictionnaires vous permet d'exécuter un bloc de code sur chaque paire clé-valeur de dictionnaire à l'aide de la méthode `enumerateKeysAndObjectsUsingBlock:(void (^)(id key, id obj, BOOL *stop))block`

Exemple:

```
NSDictionary stockSymbolsDictionary = @{
                                        @"AAPL": @"Apple",
                                        @"GOOGL": @"Alphabet",
                                        @"MSFT": @"Microsoft",
                                        @"AMZN": @"Amazon"
                                      };
NSLog(@"Printing contents of dictionary via enumeration");
[stockSymbolsDictionary enumerateKeysAndObjectsUsingBlock:^(id key, id obj, BOOL *stop) {
    NSLog(@"Key: %@, Value: %@", key, obj);
}];
```

Énumération rapide

`NSDictionary` peut être énuméré en utilisant une énumération rapide, tout comme les autres types de collection:

```
NSDictionary stockSymbolsDictionary = @{
                                      @"AAPL": @"Apple",
                                      @"GOOGL": @"Alphabet",
                                      @"MSFT": @"Microsoft",
                                      @"AMZN": @"Amazon"
                                    };

for (id key in stockSymbolsDictionary)
{
    id value = dictionary[key];
    NSLog(@"Key: %@, Value: %@", key, value);
}
```

`NSDictionary` étant intrinsèquement non ordonné, l'ordre des clés de la boucle for n'est pas garanti.

Lire NSDictionary en ligne: https://riptutorial.com/fr/objective-c/topic/847/nsdictionary

Chapitre 30: NSDictionary

Syntaxe

- @{ valeur clé, ... }
- [NSDictionary dictionaryWithObjectsAndKeys: value, key, ..., nil];
- dict [*clé*] = *valeur* ;
- id *value* = dict [*clé*];

Remarques

La classe NSDictionary déclare l'interface programmatique aux objets qui gèrent des associations immuables de clés et de valeurs. Utilisez cette classe ou sa sous-classe NSMutableDictionary lorsque vous avez besoin d'un moyen pratique et efficace pour récupérer des données associées à une clé arbitraire. NSDictionary crée des dictionnaires statiques et NSMutableDictionary crée des dictionnaires dynamiques. (Pour plus de commodité, le terme dictionnaire fait référence à toute instance de l'une de ces classes sans spécifier son appartenance à une classe exacte.)

Une paire clé-valeur dans un dictionnaire est appelée une entrée. Chaque entrée se compose d'un objet qui représente la clé et d'un second objet qui correspond à la valeur de cette clé. Dans un dictionnaire, les clés sont uniques. C'est-à-dire que deux clés d'un même dictionnaire ne sont pas égales (comme déterminé par isEqual :). En général, une clé peut être n'importe quel objet (à condition qu'elle soit conforme au protocole NSCopying — voir ci-dessous), mais notez que lors de l'utilisation du codage, la clé doit être une chaîne (voir Principes fondamentaux du codage). Ni une clé ni une valeur ne peuvent être nulles; Si vous devez représenter une valeur nulle dans un dictionnaire, vous devez utiliser NSNull.

NSDictionary est «sans frais ponté» avec son homologue Core Foundation, CFDictionaryRef. Reportez-vous à la section Pontage sans frais pour plus d'informations sur les passerelles sans frais.

Examples

Créer des littéraux

```
NSDictionary *inventory = @{
    @"Mercedes-Benz SLK250" : @(13),
    @"BMW M3 Coupe" : @(self.BMWM3CoupeInventory.count),
    @"Last Updated" : @"Jul 21, 2016",
    @"Next Update"  : self.nextInventoryUpdateString
};
```

Création à l'aide de dictionaryWithObjectsAndKeys:

```
NSDictionary *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];
```

`nil` doit être passé comme dernier paramètre en tant que sentinelle indiquant la fin.

Il est important de se rappeler que lors de l'instanciation des dictionnaires, les valeurs passent en premier et les clés en second. Dans l'exemple ci-dessus, les chaînes sont les clés et les nombres sont les valeurs. Le nom de la méthode reflète également ceci: `dictionaryWithObjectsAndKeys` . Bien que cela ne soit pas incorrect, la manière la plus moderne d'instancier les dictionnaires (avec les littéraux) est préférable.

Créer à l'aide de plists

```
NSString *pathToPlist = [[NSBundle mainBundle] pathForResource:@"plistName"
    ofType:@"plist"];
NSDictionary *plistDict = [[NSDictionary alloc] initWithContentsOfFile:pathToPlist];
```

Définition d'une valeur dans NSDictionary

Il existe plusieurs façons de définir l'objet d'une clé dans un NSDictionary, correspondant aux manières dont vous obtenez une valeur. Par exemple, pour ajouter une lamborghini à une liste de voitures

la norme

```
[cars setObject:lamborghini forKey:@"Lamborghini"];
```

Tout comme n'importe quel autre objet, appelez la méthode de NSDictionary qui définit un objet d'une clé, `objectForKey:` Veillez à ne pas confondre ceci avec `setValue:forKey:` c'est pour une chose complètement différente, Key Value Coding

Sténographie

```
cars[@"Lamborghini"] = lamborghini;
```

C'est la syntaxe que vous utilisez pour les dictionnaires dans la plupart des autres langages, tels que C #, Java et Javascript. C'est beaucoup plus pratique que la syntaxe standard, et sans doute plus lisible (surtout si vous codez dans ces autres langages), mais bien sûr, ce n'est pas *standard* . Il est également disponible uniquement dans les nouvelles versions d'Objective C

Obtenir une valeur de NSDictionary

Il existe plusieurs façons d'obtenir un objet à partir d'un NSDictionary avec une clé. Par exemple,

pour obtenir une lamborghini d'une liste de voitures

la norme

```
Car * lamborghini = [cars objectForKey:@"Lamborghini"];
```

Tout comme n'importe quel autre objet, appelez la méthode de NSDictionary qui vous donne un objet pour une clé, `objectForKey:` Veillez à ne pas confondre ceci avec `valueForKey:` c'est pour une chose complètement différente, Key Value Coding

Sténographie

```
Car * lamborghini = cars[@"Lamborghini"];
```

C'est la syntaxe que vous utilisez pour les dictionnaires dans la plupart des autres langages, tels que C #, Java et Javascript. C'est beaucoup plus pratique que la syntaxe standard, et sans doute plus lisible (surtout si vous codez dans ces autres langages), mais bien sûr, ce n'est pas *standard* . Il est également disponible uniquement dans les nouvelles versions d'Objective C

Vérifiez si NSDictionary a déjà une clé ou non

Objectif c:

```
   //this is the dictionary you start with.
 NSDictionary *dict = [NSDictionary dictionaryWithObjectsAndKeys:@"name1", @"Sam",@"name2",
@"Sanju",nil];

//check if the dictionary contains the key you are going to modify.  In this example, @"Sam"
if (dict[@"name1"] != nil) {
    //there is an entry for Key name1
}
else {
    //There is no entry for name1
}
```

Lire NSDictionary en ligne: https://riptutorial.com/fr/objective-c/topic/875/nsdictionary

Chapitre 31: NSJSONSerialization

Syntaxe

- (id) Options de données JSONObjectWithData: (NSData *): (NSJSONReadingOptions) erreur d'option: (NSError * _Nullable *) erreur

Paramètres

Opérateur	La description
Les données	Un objet de données contenant des données JSON
opter	Options pour lire les données JSON et créer les objets Foundation.
Erreur	Si une erreur survient, contient au retour un objet NSError qui décrit le problème.

Remarques

NSJSONSerialization est disponible dans iOS 5.0 et versions ultérieures Un objet pouvant être converti en JSON doit avoir les propriétés suivantes:

- L'objet de niveau supérieur est un NSArray ou NSDictionary.
- Tous les objets sont des instances de NSString, NSNumber, NSArray, NSDictionary ou NSNull.
- Toutes les clés de dictionnaire sont des instances de NSString.
- Les nombres ne sont pas NaN ou l'infini.

Examples

Analyse JSON utilisant NSJSONSerialization Objective c

```
NSError *e = nil;
NSString *jsonString = @"[{\"id\": \"1\", \"name\":\"sam\"}]";
NSData *data = [jsonString dataUsingEncoding:NSUTF8StringEncoding];

NSArray *jsonArray = [NSJSONSerialization JSONObjectWithData: data options:
NSJSONReadingMutableContainers error: &e];

if (!jsonArray) {
    NSLog(@"Error parsing JSON: %@", e);
```

```
} else {
    for(NSDictionary *item in jsonArray) {
        NSLog(@"Item: %@", item);
    }
}
```

Sortie:

```
Item: {
    id = 1;
    name = sam;
}
```

Exemple 2: Utilisation du contenu de l'url:

```
//Parsing:

NSData *data = [NSData dataWithContentsOfURL:@"URL HERE"];
NSError *error;
NSDictionary *json = [NSJSONSerialization JSONObjectWithData:data options:kNilOptions
error:&error];
NSLog(@"json :%@",json);
```

Exemple de réponse:

```
json: {
    MESSAGE = "Test Message";
    RESPONSE =(
                {
            email = "test@gmail.com";
            id = 15;
            phone = 1234567890;
            name = Staffy;
        }
    );
    STATUS = SUCCESS;
}


 NSMutableDictionary *response = [[[json valueForKey:@"RESPONSE"]
objectAtIndex:0]mutableCopy];
 NSString *nameStr = [response valueForKey:@"name"];
 NSString *emailIdStr = [response valueForKey:@"email"];
```

Lire NSJSONSerialization en ligne: https://riptutorial.com/fr/objective-c/topic/2587/nsjsonserialization

Chapitre 32: NSMutableArray

Examples

Ajouter des éléments

```
NSMutableArray *myColors;
myColors = [NSMutableArray arrayWithObjects: @"Red", @"Green", @"Blue", @"Yellow", nil];
[myColors addObject: @"Indigo"];
[myColors addObject: @"Violet"];

//Add objects from an NSArray
NSArray *myArray = @[@"Purple",@"Orange"];
[myColors addObjectsFromArray:myArray];
```

Insérer des éléments

```
NSMutableArray *myColors;
int i;
int count;
myColors = [NSMutableArray arrayWithObjects: @"Red", @"Green", @"Blue", @"Yellow", nil];
[myColors insertObject: @"Indigo" atIndex: 1];
[myColors insertObject: @"Violet" atIndex: 3];
```

Supprimer des éléments

Supprimer à un index spécifique:

```
[myColors removeObjectAtIndex: 3];
```

Supprimez la première instance d'un objet spécifique:

```
[myColors removeObject: @"Red"];
```

Supprimez toutes les instances d'un objet spécifique:

```
[myColors removeObjectIdenticalTo: @"Red"];
```

Supprimer tous les objets:

```
[myColors removeAllObjects];
```

Supprimer le dernier objet:

```
[myColors removeLastObject];
```

Tri des tableaux

```
NSMutableArray *myColors = [NSMutableArray arrayWithObjects: @"red", @"green", @"blue",
@"yellow", nil];
NSArray *sortedArray;
sortedArray = [myColors sortedArrayUsingSelector:@selector(localizedCaseInsensitiveCompare:)];
```

Déplacer un objet vers un autre index

Déplacer le *bleu* au début du tableau:

```
NSMutableArray *myColors = [NSMutableArray arrayWithObjects: @"Red", @"Green", @"Blue",
@"Yellow", nil];

NSUInteger fromIndex = 2;
NSUInteger toIndex = 0;

id blue = [[[self.array objectAtIndex:fromIndex] retain] autorelease];
[self.array removeObjectAtIndex:fromIndex];
[self.array insertObject:blue atIndex:toIndex];
```

`myColors` est maintenant `[@"Blue", @"Red", @"Green", @"Yellow"]` .

Filtrer le contenu d'un tableau avec un prédicat

Utilisation de **filterUsingPredicate: évalue** un prédicat donné par rapport au contenu des tableaux et renvoie les objets correspondants.

Exemple:

```
      NSMutableArray *array = [NSMutableArray array];
      [array setArray:@[@"iOS",@"macOS",@"tvOS"]];
      NSPredicate *predicate = [NSPredicate predicateWithFormat:@"SELF beginswith[c] 'i'"];
      NSArray *resultArray = [array filteredArrayUsingPredicate:predicate];
      NSLog(@"%@",resultArray);
```

Création d'un NSMutableArray

`NSMutableArray` peut être initialisé comme un tableau vide comme ceci:

```
NSMutableArray *array = [[NSMutableArray alloc] init];
// or
NSMutableArray *array2 = @[].mutableCopy;
// or
NSMutableArray *array3 = [NSMutableArray array];
```

`NSMutableArray` peut être initialisé avec un autre tableau comme celui-ci:

```
NSMutableArray *array4 = [[NSMutableArray alloc] initWithArray:anotherArray];
// or
NSMutableArray *array5 = anotherArray.mutableCopy;
```

Lire NSMutableArray en ligne: https://riptutorial.com/fr/objective-c/topic/2008/nsmutablearray

Chapitre 33: NSMutableDictionary

Paramètres

objets	clés
Un tableau contenant les valeurs du nouveau dictionnaire.	CellAn array contenant les clés du nouveau dictionnaire. Chaque clé est copiée et la copie est ajoutée au dictionnaire.

Examples

Exemple NSMutableDictionary

+ dictionaryWithCapacity:

Crée et retourne un dictionnaire mutable, en lui donnant initialement suffisamment de mémoire allouée pour contenir un nombre donné d'entrées.

```
NSMutableDictionary *dict =  [NSMutableDictionary dictionaryWithCapacity:1];
NSLog(@"%@",dict);
```

- init

Initialise un dictionnaire mutable nouvellement attribué.

```
NSMutableDictionary *dict =  [[NSMutableDictionary alloc] init];
NSLog(@"%@",dict);
```

+ dictionaryWithSharedKeySet:

Crée un dictionnaire mutable optimisé pour traiter un ensemble de clés connu.

```
id sharedKeySet = [NSDictionary sharedKeySetForKeys:@[@"key1", @"key2"]]; // returns
NSSharedKeySet
NSMutableDictionary *dict = [NSMutableDictionary dictionaryWithSharedKeySet:sharedKeySet];
dict[@"key1"] = @"Easy";
dict[@"key2"] = @"Tutorial";
//We can an object thats not in the shared keyset
dict[@"key3"] = @"Website";
NSLog(@"%@",dict);
```

SORTIE

```
{
    key1 = Eezy;
    key2 = Tutorials;
```

```
    key3 = Website;
}
```

Ajout d'entrées à un dictionnaire Mutable

- setObject: forKey:

Ajoute une paire clé-valeur donnée au dictionnaire.

```
NSMutableDictionary *dict =  [NSMutableDictionary dictionary];
[dict setObject:@"Easy" forKey:@"Key1"];
NSLog(@"%@",dict);
```

SORTIE

```
{
    Key1 = Eezy;
}
```

- setObject: forKeyedSubscript:

Ajoute une paire clé-valeur donnée au dictionnaire.

```
NSMutableDictionary *dict =  [NSMutableDictionary dictionary];
[dict setObject:@"Easy" forKeyedSubscript:@"Key1"];
NSLog(@"%@",dict);
```

SORTIE {Key1 = Facile; }

Supprimer des entrées d'un dictionnaire Mutable

- removeObjectForKey:

Supprime une clé donnée et sa valeur associée du dictionnaire.

```
NSMutableDictionary *dict =  [NSMutableDictionary
dictionaryWithDictionary:@{@"key1":@"Easy",@"key2": @"Tutorials"}];
[dict removeObjectForKey:@"key1"];
NSLog(@"%@",dict);
```

SORTIE

```
{
    key2 = Tutorials;
}
```

- removeAllObjects

Vide le dictionnaire de ses entrées.

```
NSMutableDictionary *dict =  [NSMutableDictionary
dictionaryWithDictionary:@{@"key1":@"Eezy",@"key2": @"Tutorials"}];
[dict removeAllObjects];
NSLog(@"%@",dict);
```

SORTIE

```
{
}
```

- removeObjectsForKeys:

Supprime des entrées de dictionnaire spécifiées par les éléments d'un tableau donné.

```
NSMutableDictionary *dict =  [NSMutableDictionary
dictionaryWithDictionary:@{@"key1":@"Easy",@"key2": @"Tutorials"}];
[dict removeObjectsForKeys:@[@"key1"]];
NSLog(@"%@",dict);
```

SORTIE

```
{
    key2 = Tutorials;
}
```

Lire NSMutableDictionary en ligne: https://riptutorial.com/fr/objective-c/topic/3103/nsmutabledictionary

Chapitre 34: NSObject

Introduction

`NSObject` est la classe racine de `Cocoa` . Toutefois, le langage `Objective-C` ne définit pas de classes de racine du tout par `Cocoa` , le Framework d'Apple. Cette classe racine de la plupart des hiérarchies de classes Objective-C le système d'exécution et la capacité à se comporter comme des objets Objective-C.

Cette classe a toutes les propriétés de base de l'objet de classe `Objective'C` comme:

soi.

classe (nom de la classe).

superclasse (superclasse de la classe actuelle).

Syntaxe

- soi
- superclasse
- init
- allouer
- Nouveau
- est égal
- isKindOfClass
- isMemberOfClass
- la description

Examples

NSObject

`@interface NSString : NSObject` (`NSObject` est une classe de base de la classe NSString).

Vous pouvez utiliser les méthodes ci-dessous pour l'allocation d'une classe de chaîne:

```
- (instancetype)init

+ (instancetype)new

+ (instancetype)alloc
```

Pour copier un objet:

```
- (id)copy;
```

```
- (id)mutableCopy;
```

Pour comparer des objets:

```
- (BOOL)isEqual:(id)object
```

Pour obtenir la super-classe de la classe actuelle:

```
superclass
```

Pour vérifier quel type de classe est-ce?

```
- (BOOL)isKindOfClass:(Class)aClass
```

Quelques propriétés des classes NON-ARC:

```
- (instancetype)retain OBJC_ARC_UNAVAILABLE;

- (oneway void)release OBJC_ARC_UNAVAILABLE;

- (instancetype)autorelease OBJC_ARC_UNAVAILABLE;

- (NSUInteger)retainCount
```

Lire NSObject en ligne: https://riptutorial.com/fr/objective-c/topic/9355/nsobject

Chapitre 35: NSPredicate

Syntaxe

- Opérateur CONTAINS: Permet de filtrer des objets avec un sous-ensemble correspondant.

```
NSPredicate *filterByName = [NSPredicate predicateWithFormat:@"self.title CONTAINS[cd]
%@",@"Tom"];
```

- LIKE: Son filtre de comparaison simple.

```
NSPredicate *filterByNameCIS = [NSPredicate predicateWithFormat:@"self.title LIKE[cd]
%@",@"Tom and Jerry"];
```

- = operator: retourne tous les objets avec la valeur de filtre correspondante.

```
NSPredicate *filterByNameCS = [NSPredicate predicateWithFormat:@"self.title = %@",@"Tom and
Jerry"];
```

- Opérateur IN: Permet de filtrer des objets avec un jeu de filtres spécifique.

```
NSPredicate *filterByIds = [NSPredicate predicateWithFormat:@"self.id IN %@",@[@"7CDF6D22-
8D36-49C2-84FE-E31EECCECB79", @"7CDF6D22-8D36-49C2-84FE-E31EECCECB76"]];
```

- Opérateur NOT IN: il vous permet de trouver des objets Inverse avec un ensemble spécifique.

```
NSPredicate *filterByNotInIds = [NSPredicate predicateWithFormat:@"NOT (self.id IN
%@)",@[@"7CDF6D22-8D36-49C2-84FE-E31EECCECB79", @"7CDF6D22-8D36-49C2-84FE-E31EECCECB76"]];
```

Remarques

Pour plus de détails, lisez NSPredicate dans la documentation Apple.

Examples

Filtrer par nom

```
NSArray *array = @[
                    @{
                        @"id": @"7CDF6D22-8D36-49C2-84FE-E31EECCECB71",
                        @"title": @"Jackie Chan Strike Movie",
                        @"url": @"http://abc.com/playback.m3u8",
                        @"thumbnailURL": @"http://abc.com/thumbnail.png",
                        @"isMovie" : @1
                    },
                    @{
                        @"id": @"7CDF6D22-8D36-49C2-84FE-E31EECCECB72",
                        @"title": @"Sherlock homes",
                        @"url": @"http://abc.com/playback.m3u8",
                        @"thumbnailURL": @"http://abc.com/thumbnail.png",
                        @"isMovie" : @0
```

```
                    },
                    @{
                        @"id": @"7CDF6D22-8D36-49C2-84FE-E31EECCECB73",
                        @"title": @"Titanic",
                        @"url": @"http://abc.com/playback.m3u8",
                        @"thumbnailURL": @"http://abc.com/thumbnail.png",
                        @"isMovie" : @1
                    },
                    @{
                        @"id": @"7CDF6D22-8D36-49C2-84FE-E31EECCECB74",
                        @"title": @"Star Wars",
                        @"url": @"http://abc.com/playback.m3u8",
                        @"thumbnailURL": @"http://abc.com/thumbnail.png",
                        @"isMovie" : @1
                    },
                    @{
                        @"id": @"7CDF6D22-8D36-49C2-84FE-E31EECCECB75",
                        @"title": @"Pokemon",
                        @"url": @"http://abc.com/playback.m3u8",
                        @"thumbnailURL": @"http://abc.com/thumbnail.png",
                        @"isMovie" : @0
                    },
                    @{
                        @"id": @"7CDF6D22-8D36-49C2-84FE-E31EECCECB76",
                        @"title": @"Avatar",
                        @"url": @"http://abc.com/playback.m3u8",
                        @"thumbnailURL": @"http://abc.com/thumbnail.png",
                        @"isMovie" : @1
                    },
                    @{
                        @"id": @"7CDF6D22-8D36-49C2-84FE-E31EECCECB77",
                        @"title": @"Popey",
                        @"url": @"http://abc.com/playback.m3u8",
                        @"thumbnailURL": @"http://abc.com/thumbnail.png",
                        @"isMovie" : @1
                    },
                    @{
                        @"id": @"7CDF6D22-8D36-49C2-84FE-E31EECCECB78",
                        @"title": @"Tom and Jerry",
                        @"url": @"http://abc.com/playback.m3u8",
                        @"thumbnailURL": @"http://abc.com/thumbnail.png",
                        @"isMovie" : @1
                    },
                    @{
                        @"id": @"7CDF6D22-8D36-49C2-84FE-E31EECCECB79",
                        @"title": @"The wolf",
                        @"url": @"http://abc.com/playback.m3u8",
                        @"thumbnailURL": @"http://abc.com/thumbnail.png",
                        @"isMovie" : @1
                    }
                    ];

// *** Case Insensitive comparision with excate title match ***
NSPredicate *filterByNameCIS = [NSPredicate predicateWithFormat:@"self.title LIKE[cd]
%@",@"Tom and Jerry"];
NSLog(@"Filter By Name(CIS) : %@",[array filteredArrayUsingPredicate:filterByNameCIS]);
```

Trouver des films sauf les identifiants donnés

```
// *** Find movies except given ids ***
NSPredicate *filterByNotInIds = [NSPredicate predicateWithFormat:@"NOT (self.id IN
%@)",@[@"7CDF6D22-8D36-49C2-84FE-E31EECCECB79", @"7CDF6D22-8D36-49C2-84FE-E31EECCECB76"]];
NSLog(@"Filter movies except given Ids : %@",[array
filteredArrayUsingPredicate:filterByNotInIds]);
```

Trouver tous les objets de type movie

```
// *** Find all the objects which is of type movie, Both the syntax are valid ***
NSPredicate *filterByMovieType = [NSPredicate predicateWithFormat:@"self.isMovie = %@",@1];
// OR
//NSPredicate *filterByMovieType = [NSPredicate predicateWithFormat:@"self.isMovie =
%@",[NSNumber numberWithBool:YES]];
NSLog(@"Filter By Movie Type : %@",[array filteredArrayUsingPredicate:filterByMovieType]);
```

Trouver les identifiants d'objet distinct du tableau

```
// *** Find Distinct object ids of array ***
NSLog(@"Distinct id : %@",[array valueForKeyPath:@"@distinctUnionOfObjects.id"]);
```

Trouver des films avec des identifiants spécifiques

```
// *** Find movies with specific ids ***
NSPredicate *filterByIds = [NSPredicate predicateWithFormat:@"self.id IN %@",@[@"7CDF6D22-
8D36-49C2-84FE-E31EECCECB79", @"7CDF6D22-8D36-49C2-84FE-E31EECCECB76"]];
NSLog(@"Filter By Ids : %@",[array filteredArrayUsingPredicate:filterByIds]);
```

Comparaison insensible à la casse avec correspondance exacte du titre

```
// *** Case Insensitive comparison with exact title match ***
NSPredicate *filterByNameCIS = [NSPredicate predicateWithFormat:@"self.title LIKE[cd]
%@",@"Tom and Jerry"];
NSLog(@"Filter By Name(CIS) : %@",[array filteredArrayUsingPredicate:filterByNameCIS]);
```

Sensible à la casse avec correspondance exacte du titre

```
// *** Case sensitive with exact title match ***
NSPredicate *filterByNameCS = [NSPredicate predicateWithFormat:@"self.title = %@",@"Tom and
Jerry"];
NSLog(@"Filter By Name(CS) : %@",[array filteredArrayUsingPredicate:filterByNameCS]);
```

Comparaison insensible à la casse avec le sous-ensemble correspondant

```
// *** Case Insensitive comparison with matching subset ***
NSPredicate *filterByName = [NSPredicate predicateWithFormat:@"self.title CONTAINS[cd]
%@",@"Tom"];
NSLog(@"Filter By Containing Name : %@",[array filteredArrayUsingPredicate:filterByName]);
```

Lire NSPredicate en ligne: https://riptutorial.com/fr/objective-c/topic/2004/nspredicate

Chapitre 36: NSRegularExpression

Syntaxe

- NSRegularExpression * regex = [NSRegularExpression regularExpressionWithPattern: options PATTERN: erreur OPTIONS: ERROR];
- NSArray <NSTextCheckingResult *> * results = [regex matchesInString: options STRING: plage OPTIONS: RANGE_IN_STRING];
- NSInteger numberOfMatches = [regex numberOfMatchesInString: options STRING: plage OPTIONS: RANGE_IN_STRING];

Examples

Trouver tous les nombres dans une chaîne

```
NSString *testString = @"There are 42 sheep and 8672 cows.";
NSError *error = nil;
NSRegularExpression *regex = [NSRegularExpression regularExpressionWithPattern:@"(\\d+)"

options:NSRegularExpressionCaseInsensitive
                                                                        error:&error];

NSArray *matches = [regex matchesInString:testString
                                  options:0
                                    range:NSMakeRange(0, testString.length)];

for (NSTextCheckingResult *matchResult in matches) {
    NSString* match = [testString substringWithRange:matchResult.range];
    NSLog(@"match: %@", match);
}
```

La sortie sera la `match: 42` et la `match: 8672` .

Vérifier si une chaîne correspond à un motif

```
NSString *testString1 = @"(555) 123-5678";
NSString *testString2 = @"not a phone number";

NSError *error = nil;
NSRegularExpression *regex = [NSRegularExpression regularExpressionWithPattern:@"^\\(\\d{3}\\)
\\d{3}\\-\\d{4}$"

options:NSRegularExpressionCaseInsensitive error:&error];

NSInteger result1 = [regex numberOfMatchesInString:testString1 options:0 range:NSMakeRange(0,
testString1.length)];
NSInteger result2 = [regex numberOfMatchesInString:testString2 options:0 range:NSMakeRange(0,
testString2.length)];

NSLog(@"Is string 1 a phone number? %@", result1 > 0 ? @"YES" : @"NO");
NSLog(@"Is string 2 a phone number? %@", result2 > 0 ? @"YES" : @"NO");
```

La sortie indique que la première chaîne est un numéro de téléphone et que la seconde ne l’est pas.

Lire NSRegularExpression en ligne: https://riptutorial.com/fr/objective-c/topic/2484/nsregularexpression

Chapitre 37: NSSortDescriptor

Examples

Trié par des combinaisons de NSSortDescriptor

```
NSArray *aryFName = @[ @"Alice", @"Bob", @"Charlie", @"Quentin" ];
NSArray *aryLName = @[ @"Smith", @"Jones", @"Smith", @"Alberts" ];
NSArray *aryAge = @[ @24, @27, @33, @31 ];

//Create a Custom class with properties for firstName & lastName of type NSString *,
//and age, which is an NSUInteger.

NSMutableArray *aryPerson = [NSMutableArray array];
[firstNames enumerateObjectsUsingBlock:^(id obj, NSUInteger idx, BOOL *stop) {
    Person *person = [[Person alloc] init];
    person.firstName = [aryFName objectAtIndex:idx];
    person.lastName = [aryLName objectAtIndex:idx];
    person.age = [aryAge objectAtIndex:idx];
    [aryPerson addObject:person];
}];

NSSortDescriptor *firstNameSortDescriptor = [NSSortDescriptor
sortDescriptorWithKey:@"firstName"
                                           ascending:YES
                                           selector:@selector(localizedStandardCompare:)];

NSSortDescriptor *lastNameSortDescriptor = [NSSortDescriptor sortDescriptorWithKey:@"lastName"
                                          ascending:YES
                                          selector:@selector(localizedStandardCompare:)];

NSSortDescriptor *ageSortDescriptor = [NSSortDescriptor sortDescriptorWithKey:@"age"
                                      ascending:NO];

NSLog(@"By age: %@", [aryPerson sortedArrayUsingDescriptors:@[ageSortDescriptor]]);
// "Charlie Smith", "Quentin Alberts", "Bob Jones", "Alice Smith"


NSLog(@"By first name: %@", [aryPerson
sortedArrayUsingDescriptors:@[firstNameSortDescriptor]]);
// "Alice Smith", "Bob Jones", "Charlie Smith", "Quentin Alberts"


NSLog(@"By last name, first name: %@", [aryPerson
sortedArrayUsingDescriptors:@[lastNameSortDescriptor, firstNameSortDescriptor]]);
// "Quentin Alberts", "Bob Jones", "Alice Smith", "Charlie Smith"
```

Lire NSSortDescriptor en ligne: https://riptutorial.com/fr/objective-c/topic/5833/nssortdescriptor

Chapitre 38: NSString

Introduction

La classe *NSString* fait partie du framework Foundation pour travailler avec des chaînes (séries de caractères). Il comprend également des méthodes pour comparer, rechercher et modifier des chaînes.

Remarques

Pour imbriquer divers types d'objets et types de données dans NSStrings, reportez-vous à: Object-C, spécificateurs de format

Examples

Création

Simple:

```
NSString *newString = @"My String";
```

De plusieurs chaînes:

```
NSString *stringOne = @"Hello";
NSString *stringTwo = @"world";
NSString *newString = [NSString stringWithFormat:@"My message: %@ %@",
                     stringOne, stringTwo];
```

Utilisation de la chaîne Mutable

```
NSString *stringOne = @"Hello";
NSString *stringTwo = @"World";
NSMutableString *mutableString = [NSMutableString new];
[mutableString appendString:stringOne];
[mutableString appendString:stringTwo];
```

De NSData:

Lors de l'initialisation à partir de `NSData` , un codage explicite doit être fourni car `NSString` n'est pas en mesure de deviner comment les caractères sont représentés dans le flux de données brutes. L'encodage le plus courant de nos jours est UTF-8, qui est même une exigence pour certaines données telles que JSON.

Évitez d'utiliser `+[NSString stringWithUTF8String:]` car il attend une chaîne C explicitement terminée par NULL, qui `-[NSData bytes]` ne fournit *pas* .

```
NSString *newString = [[NSString alloc] initWithData:myData encoding:NSUTF8StringEncoding];
```

De NSArray:

```
NSArray *myArray = [NSArray arrayWithObjects:@"Apple", @"Banana", @"Strawberry", @"Kiwi",
nil];
NSString *newString = [myArray componentsJoinedByString:@" "];
```

Longueur de chaine

NSString a une propriété `length` pour obtenir le nombre de caractères.

```
NSString *string = @"example";
NSUInteger length = string.length;       // length equals 7
```

Comme dans l' exemple de fractionnement , gardez à l'esprit que `NSString` utilise UTF-16 pour représenter les caractères. La longueur est en fait juste le nombre d'unités de code UTF-16. Cela peut différer de ce que l'utilisateur perçoit comme des personnages.

Voici quelques cas qui pourraient être surprenants:

```
@"é".length == 1   // LATIN SMALL LETTER E WITH ACUTE (U+00E9)
@"é".length == 2   // LATIN SMALL LETTER E (U+0065) + COMBINING ACUTE ACCENT (U+0301)
@"❤".length == 2  // HEAVY BLACK HEART (U+2764) + VARIATION SELECTOR-16 (U+FE0F)
@"".length == 4  // REGIONAL INDICATOR SYMBOL LETTER I (U+1F1EE) + REGIONAL INDICATOR SYMBOL
LETTER T (U+1F1F9)
```

Pour obtenir le nombre de caractères perçus par l'utilisateur, appelés techniquement " grappes de graphèmes ", vous devez parcourir la chaîne avec `-enumerateSubstringsInRange:options:usingBlock:` et conserver un compte. Ceci est démontré dans une réponse de Nikolai Ruhe sur Stack Overflow .

Changement de Cas

Pour convertir une chaîne en majuscule, utilisez la chaîne `uppercaseString` :

```
NSString *myString = @"Emphasize this";
NSLog(@"%@", [myString uppercaseString]; // @"EMPHASIZE THIS"
```

Pour convertir une chaîne en minuscule, utilisez la chaîne `lowercaseString` :

```
NSString *myString = @"NORMALIZE this";
NSLog(@"%@", [myString lowercaseString]; // @"normalize this"
```

Pour mettre en majuscule le premier caractère de chaque mot d'une chaîne, utilisez `capitalizedString` :

```
NSString *myString = @"firstname lastname";
NSLog(@"%@", [myString capitalizedString]); // @"Firstname Lastname"
```

Comparaison de chaînes

Les chaînes sont comparées pour l'égalité en utilisant `isEqualToString:`

L'opérateur `==` teste simplement l'identité de l'objet et ne compare pas les valeurs logiques des objets, il ne peut donc pas être utilisé:

```
NSString *stringOne = @"example";
NSString *stringTwo = [stringOne mutableCopy];

BOOL objectsAreIdentical = (stringOne == stringTwo);          // NO
BOOL stringsAreEqual = [stringOne isEqualToString:stringTwo]; // YES
```

L'expression `(stringOne == stringTwo)` teste si les adresses mémoire des deux chaînes sont identiques, ce qui n'est généralement pas ce que nous voulons.

Si les variables de chaîne peuvent être `nil` vous devez également prendre en compte ce cas:

```
BOOL equalValues = stringOne == stringTwo || [stringOne isEqualToString:stringTwo];
```

Cette condition renvoie `YES` lorsque les chaînes ont des valeurs égales ou que les deux sont `nil` .

Pour commander deux chaînes par ordre alphabétique, utilisez `compare`

```
NSComparisonResult result = [firstString compare:secondString];
```

`NSComparisonResult` peut être:

- `NSOrderedAscending` : La première chaîne vient avant la deuxième chaîne.
- `NSOrderedSame` : les chaînes sont égales.
- `NSOrderedDescending` : la deuxième chaîne vient avant la première chaîne.

Pour comparer l'égalité entre deux chaînes, utilisez `isEqualToString:`

```
BOOL result = [firstString isEqualToString:secondString];
```

Pour comparer avec la chaîne vide (`@""`), mieux vaut utiliser la `length` .

```
BOOL result = string.length == 0;
```

Rejoindre un tableau de chaînes

Pour combiner un `NSArray` de `NSString` dans un nouveau `NSString` :

```
NSArray *yourWords = @[@"Objective-C", @"is", @"just", @"awesome"];
NSString *sentence = [yourWords componentsJoinedByString:@" "];

// Sentence is now: @"Objective-C is just awesome"
```

Encodage et décodage

```
// decode
NSString *string = [[NSString alloc] initWithData:utf8Data
                                         encoding:NSUTF8StringEncoding];

// encode
NSData *utf8Data = [string dataUsingEncoding:NSUTF8StringEncoding];
```

Certains encodages supportés sont:

- `NSASCIIStringEncoding`
- `NSUTF8StringEncoding`
- `NSUTF16StringEncoding` (== `NSUnicodeStringEncoding`)

Notez que `utf8Data.bytes` n'inclut pas de caractère NULL `utf8Data.bytes` , nécessaire pour les chaînes C. Si vous avez besoin d'une chaîne C, utilisez `UTF8String` :

```
const char *cString = [string UTF8String];
printf("%s", cString);
```

Scission

Vous pouvez diviser une chaîne en un tableau de parties, divisé par **un caractère séparateur** .

```
NSString * yourString = @"Stack,Exchange,Network";
NSArray * yourWords = [yourString componentsSeparatedByString:@","];
// Output: @[@"Stack", @"Exchange", @"Network"]
```

Si vous devez diviser sur un ensemble de **plusieurs délimiteurs différents** , utilisez `-[NSString componentsSeparatedByCharactersInSet:]` .

```
NSString * yourString = @"Stack Overflow+Documentation/Objective-C";
NSArray * yourWords = [yourString componentsSeparatedByCharactersInSet:
                       [NSCharacterSet characterSetWithCharactersInString:@"+/"]];
// Output: @[@"Stack Overflow", @"Documentation", @"Objective-C"]`
```

Si vous devez diviser une chaîne en **caractères individuels** , parcourez la longueur de la chaîne et convertissez chaque caractère en une nouvelle chaîne.

```
NSMutableArray * characters = [[NSMutableArray alloc] initWithCapacity:[yourString length]];
for (int i = 0; i < [myString length]; i++) {
    [characters addObject: [NSString stringWithFormat:@"%C",
                                      [yourString characterAtIndex:i]];
}
```

Comme dans l' exemple de longueur , gardez à l'esprit qu'un "caractère" est une unité de code UTF-16, pas nécessairement ce que l'utilisateur considère comme un caractère. Si vous utilisez cette boucle avec `@"🇵🇷"` , vous verrez qu'elle est divisée en quatre parties.

Pour obtenir une liste des caractères perçus par l'utilisateur, utilisez `-enumerateSubstringsInRange:options:usingBlock:`

```
NSMutableArray * characters = [NSMutableArray array];
[yourString enumerateSubstringsInRange:(NSRange){0, [yourString length]}
                               options:NSStringEnumerationByComposedCharacterSequences
                            usingBlock:^(NSString * substring, NSRange r, NSRange s, BOOL *
b){
                                [characters addObject:substring];
                            }];
```

Cela préserve les grappes de graphèmes comme le drapeau italien en tant que sous-chaîne unique.

Recherche d'une sous-chaîne

Pour rechercher si une chaîne contient une sous-chaîne, procédez comme suit:

```
NSString *myString = @"This is for checking substrings";
NSString *subString = @"checking";

BOOL doesContainSubstring = [myString containsString:subString];  // YES
```

Si vous ciblez iOS 7 ou OS X 10.9 (ou antérieur):

```
BOOL doesContainSubstring = ([myString rangeOfString:subString].location != NSNotFound);  //
YES
```

Travailler avec des chaînes C

Pour convertir `NSString` en `const char` use `-[NSString UTF8String]` :

```
NSString *myNSString = @"Some string";
const char *cString = [myNSString UTF8String];
```

Vous pouvez également utiliser `-[NSString cStringUsingEncoding:]` si votre chaîne est codée avec autre chose que UTF-8.

Pour le chemin inverse, utilisez `-[NSString stringWithUTF8String:]` :

```
const *char cString = "Some string";
NSString *myNSString = [NSString stringWithUTF8String:cString];
myNSString = @(cString); // Equivalent to the above.
```

Une fois que vous avez le caractère `const char *` , vous pouvez l'utiliser avec un tableau de `chars` :

```
printf("%c\n", cString[5]);
```

Si vous souhaitez modifier la chaîne, faites une copie:

```
char *cpy = calloc(strlen(cString)+1, 1);
strncpy(cpy, cString, strlen(cString));
// Do stuff with cpy
free(cpy);
```

Suppression d'espaces de début et de fin

```
NSString *someString = @"   Objective-C Language  \n";
NSString *trimmedString = [someString stringByTrimmingCharactersInSet:[NSCharacterSet
whitespaceAndNewlineCharacterSet]];
//Output will be - "Objective-C Language"
```

La méthode stringByTrimmingCharactersInSet renvoie une nouvelle chaîne créée en supprimant des deux extrémités des caractères String contenus dans un jeu de caractères donné.

Nous pouvons également supprimer uniquement les espaces ou les lignes

```
// Removing only WhiteSpace
NSString *trimmedWhiteSpace = [someString stringByTrimmingCharactersInSet:[NSCharacterSet
whitespaceCharacterSet]];
//Output will be - "Objective-C Language  \n"

// Removing only NewLine
NSString *trimmedNewLine = [someString stringByTrimmingCharactersInSet:[NSCharacterSet
newlineCharacterSet]];
//Output will be - "   Objective-C Language  "
```

Mise en forme

Le formatage `NSString` prend en charge toutes les chaînes de format disponibles sur la fonction `printf` ANSI-C. Le seul ajout apporté par le langage est le symbole `%@` utilisé pour formater tous les objets Objective-C.

Il est possible de formater des entiers

```
int myAge = 21;
NSString *formattedAge = [NSString stringWithFormat:@"I am %d years old", my_age];
```

Ou tout objet sous-classé de NSObject

```
NSDate *now = [NSDate date];
NSString *formattedDate = [NSString stringWithFormat:@"The time right now is: %@", now];
```

Pour une liste complète des spécificateurs de format, voir: Objective-C, spécificateurs de format, syntaxe

Inverser un objectif NSString-C

```
// myString is "hi"
```

```
NSMutableString *reversedString = [NSMutableString string];
NSInteger charIndex = [myString length];
while (charIndex > 0) {
    charIndex--;
    NSRange subStrRange = NSMakeRange(charIndex, 1);
    [reversedString appendString:[myString substringWithRange:subStrRange]];
}
NSLog(@"%@", reversedString); // outputs "ih"
```

Lire NSString en ligne: https://riptutorial.com/fr/objective-c/topic/832/nsstring

Chapitre 39: NSTextAttachment

Syntaxe

1. NSTextAttachment * attachmentName = [[NSTextAttachment alloc] init];

Remarques

`NSTextAttachment` objets `NSTextAttachment` sont utilisés par le `NSAttributedString` classes `NSAttributedString` tant que valeurs des attributs de pièce jointe. Les objets que vous créez avec cette classe sont appelés objets de pièce jointe de texte, ou lorsqu'aucune confusion ne peut en résulter, en tant que pièces jointes de texte ou simplement pièces jointes.

Examples

Exemple NSTextAttachment

```
NSTextAttachment *attachment = [[NSTextAttachment alloc] init];
attachment.image = [UIImage imageNamed:@"imageName"];
attachment.bounds = CGRectMake(0, 0, 35, 35);
NSAttributedString *attachmentString = [NSAttributedString
attributedStringWithAttachment:attachment];
```

Lire NSTextAttachment en ligne: https://riptutorial.com/fr/objective-c/topic/7143/nstextattachment

Chapitre 40: NSTimer

Examples

Créer une minuterie

Cela créera une minuterie pour appeler la méthode `doSomething` sur `self` en `5.0` secondes.

```
[NSTimer scheduledTimerWithTimeInterval:5.0
         target:self
         selector:@selector(doSomething)
         userInfo:nil
         repeats:NO];
```

Si vous `repeats` paramètre `repeats` sur `false/NO` , vous souhaiterez que la minuterie ne se déclenche qu'une seule fois. Si nous définissons cette valeur sur `true/YES` , elle se déclenche toutes les cinq secondes jusqu'à ce qu'elle soit invalidée manuellement.

Invalider une minuterie

```
[timer invalidate];
timer = nil;
```

Cela empêchera la minuterie de se déclencher. **Doit être appelé à partir du thread dans lequel le minuteur a été créé,** voir les notes d'Apple :

> Vous devez envoyer ce message à partir du thread sur lequel la minuterie a été installée. Si vous envoyez ce message à partir d'un autre thread, la source d'entrée associée à la minuterie peut ne pas être supprimée de sa boucle d'exécution, ce qui peut empêcher la sortie du thread correctement.

Si vous définissez la valeur `nil` , vous pourrez ensuite vérifier si elle fonctionne ou non.

```
if(timer) {
    [timer invalidate];
    timer = nil;
}

//Now set a timer again.
```

Lancer manuellement une minuterie

```
[timer fire];
```

L'appel de la méthode d' `fire` provoque l'exécution par NSTimer de la tâche qu'il aurait normalement effectuée sur une planification.

Dans un **minuteur non répétitif** , cela invalidera automatiquement le minuteur. En d'autres termes, le fait d'appeler le `fire` avant l'intervalle de temps ne produira qu'une seule invocation.

Dans un compte à **rebours** , cela invoquera simplement l'action sans interrompre le programme habituel.

Stocker des informations dans la minuterie

Lors de la création d'une minuterie, vous pouvez définir le paramètre `userInfo` pour inclure les informations que vous souhaitez transmettre à la fonction que vous appelez avec la minuterie.

En prenant un minuteur comme paramètre dans cette fonction, vous pouvez accéder à la propriété `userInfo` .

```
NSDictionary *dictionary = @{
                              @"Message" : @"Hello, world!"
                             }; //this dictionary contains a message
[NSTimer scheduledTimerWithTimeInterval:5.0
     target:self
     selector:@selector(doSomething)
     userInfo:dictionary
     repeats:NO]; //the timer contains the dictionary and later calls the function

...

- (void) doSomething:(NSTimer*)timer{
    //the function retrieves the message from the timer
    NSLog("%@", timer.userInfo["Message"]);
}
```

Lire NSTimer en ligne: https://riptutorial.com/fr/objective-c/topic/3773/nstimer

Chapitre 41: NSURL

Examples

Créer

De NSString:

```
NSString *urlString = @"https://www.stackoverflow.com";
NSURL *myUrl = [NSURL URLWithString: urlString];
```

Vous pouvez également utiliser les méthodes suivantes:

```
- initWithString:
+ URLWithString:relativeToURL:
- initWithString:relativeToURL:
+ fileURLWithPath:isDirectory:
- initFileURLWithPath:isDirectory:
+ fileURLWithPath:
- initFileURLWithPath:
 Designated Initializer
+ fileURLWithPathComponents:
+ URLByResolvingAliasFileAtURL:options:error:
+ URLByResolvingBookmarkData:options:relativeToURL:bookmarkDataIsStale:error:
- initByResolvingBookmarkData:options:relativeToURL:bookmarkDataIsStale:error:
+ fileURLWithFileSystemRepresentation:isDirectory:relativeToURL:
- getFileSystemRepresentation:maxLength:
- initFileURLWithFileSystemRepresentation:isDirectory:relativeToURL:
```

Comparer NSURL

```
NSString *urlString = @"https://www.stackoverflow.com";

NSURL *myUrl = [NSURL URLWithString: urlString];
NSURL *myUrl2 = [NSURL URLWithString: urlString];

if ([myUrl isEqual:myUrl2]) return YES;
```

Modification et conversion d'une URL de fichier avec suppression et ajout d'un chemin

1. URLByDeletingPathExtension:

Si le récepteur représente le chemin d'accès racine, cette propriété contient une copie de l'URL d'origine. Si l'URL a plusieurs extensions de chemin, seule la dernière est supprimée.

2. URLByAppendingPathExtension:

Renvoie une nouvelle URL créée en ajoutant une extension de chemin à l'URL d'origine.

Exemple:

```
    NSUInteger count = 0;
        NSString *filePath = nil;
        do {
            NSString *extension = ( NSString *)UTTypeCopyPreferredTagWithClass((
CFStringRef)AVFileTypeQuickTimeMovie, kUTTagClassFilenameExtension);
            NSString *fileNameNoExtension = [[asset.defaultRepresentation.url
URLByDeletingPathExtension] lastPathComponent];//Delete is used
            NSString *fileName = [NSString stringWithFormat:@"%@-%@-%u",fileNameNoExtension ,
AVAssetExportPresetLowQuality, count];
            filePath = NSTemporaryDirectory();
            filePath = [filePath stringByAppendingPathComponent:fileName];//Appending is used
            filePath = [filePath stringByAppendingPathExtension:extension];
            count++;

        } while ([[NSFileManager defaultManager] fileExistsAtPath:filePath]);

        NSURL *outputURL = [NSURL fileURLWithPath:filePath];
```

Lire NSURL en ligne: https://riptutorial.com/fr/objective-c/topic/1187/nsurl

Chapitre 42: NSUrl envoie une demande de publication

Examples

Demande POST simple

```
// Create the request.
NSMutableURLRequest *request = [NSMutableURLRequest requestWithURL:[NSURL
URLWithString:@"http://google.com"]];

// Specify that it will be a POST request
request.HTTPMethod = @"POST";

// This is how we set header fields
[request setValue:@"application/xml; charset=utf-8" forHTTPHeaderField:@"Content-Type"];

// Convert your data and set your request's HTTPBody property
NSString *stringData = @"some data";
NSData *requestBodyData = [stringData dataUsingEncoding:NSUTF8StringEncoding];
request.HTTPBody = requestBodyData;

// Create url connection and fire request
NSURLConnection *conn = [[NSURLConnection alloc] initWithRequest:request delegate:self];
```

Simple Post Request With Timeout

```
// Create the request.
NSMutableURLRequest *request = [NSMutableURLRequest requestWithURL:[NSURL
URLWithString:@"http://google.com"]];

// Specify that it will be a POST request
request.HTTPMethod = @"POST";

// Setting a timeout
request.timeoutInterval = 20.0;
// This is how we set header fields
[request setValue:@"application/xml; charset=utf-8" forHTTPHeaderField:@"Content-Type"];

// Convert your data and set your request's HTTPBody property
NSString *stringData = @"some data";
NSData *requestBodyData = [stringData dataUsingEncoding:NSUTF8StringEncoding];
request.HTTPBody = requestBodyData;

// Create url connection and fire request
NSURLConnection *conn = [[NSURLConnection alloc] initWithRequest:request delegate:self];
```

Lire NSUrl envoie une demande de publication en ligne: https://riptutorial.com/fr/objective-c/topic/7243/nsurl-envoie-une-demande-de-publication

Chapitre 43: NSUserDefaults

Examples

Exemple simple

Par exemple:

POUR ÉCONOMISER:

```
  NSUserDefaults *prefs = [NSUserDefaults standardUserDefaults];

  // saving an NSString
  [prefs setObject:txtUsername.text forKey:@"userName"];
  [prefs setObject:txtPassword.text forKey:@"password"];

  [prefs synchronize];
```

POUR RÉCUPÉRER

```
  NSUserDefaults *prefs = [NSUserDefaults standardUserDefaults];

  // getting an NSString
 NSString *savedUsername = [prefs stringForKey:@"userName"];
 NSString *savedPassword = [prefs stringForKey:@"password"];
```

Effacer NSUserDefaults

```
NSString *appDomain = [[NSBundle mainBundle] bundleIdentifier];
[[NSUserDefaults standardUserDefaults] removePersistentDomainForName:appDomain];
```

Lire NSUserDefaults en ligne: https://riptutorial.com/fr/objective-c/topic/10713/nsuserdefaults

Chapitre 44: Objectif moderne-C

Examples

Littéraux

L'objectif moderne C permet de réduire la quantité de code nécessaire à l'initialisation de certains types courants. Cette nouvelle méthode est très similaire à la façon dont les objets NSString sont initialisés avec des chaînes constantes.

NSNumber

Ancienne manière:

```
NSNumber *number = [NSNumber numberWithInt:25];
```

Manière moderne:

```
NSNumber *number = @25;
```

Remarque: vous pouvez également stocker `BOOL` valeurs dans `NSNumber` objets à l' aide `@YES` , `@NO` ou `@(someBoolValue)` ;

NSArray

Ancienne manière:

```
NSArray *array = [[NSArray alloc] initWithObjects:@"One", @"Two", [NSNumber numberWithInt:3],
@"Four", nil];
```

Manière moderne:

```
NSArray *array = @[@"One", @"Two", @3, @"Four"];
```

NSDictionary

Ancienne manière:

```
NSDictionary *dictionary = [NSDictionary dictionaryWithObjectsAndKeys: array, @"Object",
[NSNumber numberWithFloat:1.5], @"Value", @"ObjectiveC", @"Language", nil];
```

Manière moderne:

```
NSDictionary *dictionary = @{@"Object": array, @"Value": @1.5, @"Language": @"ObjectiveC"};
```

Indice de conteneur

Dans la syntaxe Objective C moderne, vous pouvez obtenir des valeurs à partir des conteneurs `NSArray` et `NSDictionary` aide de l' `NSDictionary` conteneur.

Ancienne manière:

```
NSObject *object1 = [array objectAtIndex:1];
NSObject *object2 = [dictionary objectForKey:@"Value"];
```

Manière moderne:

```
NSObject *object1 = array[1];
NSObject *object2 = dictionary[@"Value"];
```

Vous pouvez également insérer des objets dans des tableaux et définir des objets pour les clés dans des dictionnaires de manière plus propre:

Ancienne manière:

```
// replacing at specific index
[mutableArray replaceObjectAtIndex:1 withObject:@"NewValue"];
// adding a new value to the end
[mutableArray addObject:@"NewValue"];

[mutableDictionary setObject:@"NewValue" forKey:@"NewKey"];
```

Manière moderne:

```
mutableArray[1] = @"NewValue";
mutableArray[[mutableArray count]] = @"NewValue";

mutableDictionary[@"NewKey"] = @"NewValue";
```

Lire Objectif moderne-C en ligne: https://riptutorial.com/fr/objective-c/topic/9486/objectif-moderne-c

Chapitre 45: Propriétés

Syntaxe

- @property (*optional_attributes, ...*) *identifiant de type* ;
- *identificateur* de @synthesize = *optional_backing_ivar;*
- *identifiant* dynamique

Paramètres

Attribut	La description
`atomic`	*Implicite.* Permet la synchronisation dans les méthodes d'accès synthétisées.
`nonatomic`	Désactive la synchronisation dans les méthodes d'accesseur synthétisées.
`readwrite`	*Implicite.* Synthétise getter, setter et back ivar.
`readonly`	Synthétise uniquement la méthode getter et le support ivar, qui peuvent être assignés directement.
`getter=` *nom*	Spécifie le nom de la méthode getter, implicite est `propertyName` .
`setter=` *nom*	Spécifie le nom de la méthode setter, l'implicite est `setPropertyName:` Colon `:` doit faire partie du nom.
`strong`	*Implicite pour les objets sous ARC* . La sauvegarde ivar est synthétisée en utilisant `__strong` , ce qui empêche la désallocation de l'objet référencé.
`retain`	Synonyme de `strong` .
`copy`	Même chose que `strong` , mais le composeur synthétisé appelle également `-copy` sur la nouvelle valeur.
`unsafe_unretained`	*Implicite, sauf pour les objets sous ARC.* La sauvegarde ivar est synthétisée à l'aide de `__unsafe_unretained` , qui (pour obejcts) se traduit par un pointeur en suspens une fois que l'objet référencé est désalloué.
`assign`	Synonyme de `unsafe_unretained` . Convient aux types sans objet.
`weak`	Le backar ivar est synthétisé en utilisant `__weak` , donc la valeur sera annulée une fois que l'objet référencé sera désalloué.
`class`	Les accesseurs de propriétés sont synthétisés en tant que méthodes de

Attribut	La description
	classe, au lieu de méthodes d'instance. Aucun stockage de sauvegarde n'est synthétisé.
`nullable`	La propriété accepte les valeurs `nil` . Principalement utilisé pour les ponts rapides.
`nonnull`	La propriété n'accepte pas les valeurs `nil` . Principalement utilisé pour les ponts rapides.
`null_resettable`	La propriété accepte les valeurs `nil` dans setter, mais ne renvoie jamais de valeurs `nil` partir de getter. Votre implémentation personnalisée de getter ou setter doit garantir ce comportement. Principalement utilisé pour les ponts rapides.
`null_unspecified`	*Implicite.* La propriété ne spécifie pas le traitement des valeurs `nil` . Principalement utilisé pour les ponts rapides.

Examples

Quelles sont les propriétés?

Voici un exemple de classe qui comporte deux variables d’instance, sans utiliser les propriétés:

```
@interface TestClass : NSObject {
    NSString *_someString;
    int _someInt;
}

-(NSString *)someString;
-(void)setSomeString:(NSString *)newString;

-(int)someInt;
-(void)setSomeInt:(NSString *)newInt;

@end


@implementation TestClass

-(NSString *)someString {
    return _someString;
}

-(void)setSomeString:(NSString *)newString {
    _someString = newString;
}

-(int)someInt {
    return _someInt;
}

-(void)setSomeInt:(int)newInt {
```

```
    _someInt = newInt;
}

@end
```

C'est un code assez standard pour créer une variable d'instance simple. Vous devez créer la variable d'instance et créer des méthodes d'accès qui ne font rien sauf définir ou renvoyer la variable d'instance. Ainsi, avec Objective-C 2.0, Apple a introduit des propriétés qui génèrent automatiquement tout ou partie du code standard.

Voici la classe ci-dessus réécrite avec les propriétés:

```
@interface TestClass

@property NSString *someString;
@property int someInt;

@end


@implementation testClass

@end
```

Une propriété est une variable d'instance associée à des getters et des setters générés automatiquement. Pour une propriété appelée `someString` , le getter et le setter sont appelés `someString` et `setSomeString:` respectivement. Le nom de la variable d'instance est, par défaut, le nom de la propriété préfixée par un trait de soulignement (la variable d'instance pour `someString` est appelée `_someString` , mais elle peut être remplacée par une directive `@synthesize` dans la section `@implementation` :

```
@synthesize someString=foo;    //names the instance variable "foo"
@synthesize someString;    //names it "someString"
@synthesize someString=_someString;        //names it "_someString"; the default if
                                           //there is no @synthesize directive
```

On peut accéder aux propriétés en appelant les getters et les setters:

```
[testObject setSomeString:@"Foo"];
NSLog(@"someInt is %d", [testObject someInt]);
```

On peut également y accéder en utilisant la notation par points:

```
testObject.someString = @"Foo";
NSLog(@"someInt is %d", testObject.someInt);
```

Getters et régleurs personnalisés

Les getters et setters de propriétés par défaut peuvent être remplacés:

```
@interface TestClass
```

```
@property NSString *someString;

@end

@implementation TestClass

// override the setter to print a message
- (void)setSomeString:(NSString *)newString {
    NSLog(@"Setting someString to %@", newString);
    // Make sure to access the ivar (default is the property name with a _
    // at the beginning) because calling self.someString would call the same
    // method again leading to an infinite recursion
    _someString = newString;
}

- (void)doSomething {
    // The next line will call the setSomeString: method
    self.someString = @"Test";
}

@end
```

Cela peut être utile pour fournir, par exemple, une initialisation différée (en remplaçant le getter pour définir la valeur initiale s'il n'a pas encore été défini):

```
- (NSString *)someString {
    if (_someString == nil) {
        _someString = [self getInitialValueForSomeString];
    }
    return _someString;
}
```

Vous pouvez également créer une propriété qui calcule sa valeur dans le getter:

```
@interface Circle : NSObject

@property CGPoint origin;
@property CGFloat radius;
@property (readonly) CGFloat area;

@end

@implementation Circle

- (CGFloat)area {
    return M_PI * pow(self.radius, 2);
}

@end
```

Propriétés provoquant des mises à jour

Cet objet, `Shape` possède une `image` propriété qui dépend de `numberOfSides` et de `sideWidth` . Si l'un d'eux est défini, l' `image` doit être recalculée. Mais le recalcul est probablement long et ne doit être effectué qu'une seule fois si les deux propriétés sont définies. Ainsi, la `Shape` permet de définir les

deux propriétés et de ne recalculer qu'une seule fois. Cela se fait en définissant directement la propriété ivars.

En `Shape.h`

```
@interface Shape {
    NSUInteger numberOfSides;
    CGFloat sideWidth;

    UIImage * image;
}

// Initializer that takes initial values for the properties.
- (instancetype)initWithNumberOfSides:(NSUInteger)numberOfSides withWidth:(CGFloat)width;

// Method that allows to set both properties in once call.
// This is useful if setting these properties has expensive side-effects.
// Using a method to set both values at once allows you to have the side-
// effect executed only once.
- (void)setNumberOfSides:(NSUInteger)numberOfSides andWidth:(CGFloat)width;

// Properties using default attributes.
@property NSUInteger numberOfSides;
@property CGFloat sideWidth;

// Property using explicit attributes.
@property(strong, readonly) UIImage * image;

@end
```

En `Shape.m`

```
@implementation AnObject

// The variable name of a property that is auto-generated by the compiler
// defaults to being the property name prefixed with an underscore, for
// example "_propertyName". You can change this default variable name using
// the following statement:
// @synthesize propertyName = customVariableName;

- (id)initWithNumberOfSides:(NSUInteger)numberOfSides withWidth:(CGFloat)width {
    if ((self = [self init])) {
       [self setNumberOfSides:numberOfSides andWidth:width];
    }

    return self;
}

- (void)setNumberOfSides:(NSUInteger)numberOfSides {
    _numberOfSides = numberOfSides;

    [self updateImage];
}

- (void)setSideWidth:(CGFloat)sideWidth {
    _sideWidth = sideWidth;

    [self updateImage];
}
```

```
- (void)setNumberOfSides:(NSUInteger)numberOfSides andWidth:(CGFloat)sideWidth {
    _numberOfSides = numberOfSides;
    _sideWidth = sideWidth;

    [self updateImage];
}

// Method that does some post-processing once either of the properties has
// been updated.
- (void)updateImage {
    ...
}

@end
```

Lorsque des propriétés sont affectées à (à l'aide de `object.property = value`), la méthode setter `setProperty:` est appelée. Ce setter, même s'il est fourni par `@synthesize` , peut être remplacé, comme c'est le cas pour `numberOfSides` et `sideWidth` . Toutefois, si vous définissez directement le paramètre ivar d'une propriété (via la `property` si l'objet est self ou `object->property`), il n'appelle pas getter ou setter, ce qui vous permet de faire des choses comme plusieurs jeux de propriétés contourner les effets secondaires causés par le passeur.

Lire Propriétés en ligne: https://riptutorial.com/fr/objective-c/topic/1818/proprietes

Chapitre 46: Protocoles

Examples

Définition de protocole de base

Définir un nouveau protocole:

```
@protocol NewProtocol

- (void)protocolMethod:(id)argument;

- (id)anotherMethod;

@end
```

Méthodes optionnelles et obligatoires

Par défaut, toutes les méthodes déclarées dans un protocole sont obligatoires. Cela signifie que toute classe conforme à ce protocole doit implémenter ces méthodes.

Il est également possible de déclarer *des* méthodes *optionnelles* . Ces méthodes ne peuvent être implémentées que si nécessaire.

Vous marquez des méthodes facultatives avec la directive `@optional` .

```
@protocol NewProtocol
- (void)protocolMethod:(id)argument;
@optional
- (id)anotherMethod;
@end
```

Dans ce cas, seule une autre `anotherMethod` est marquée comme facultative; les méthodes sans la directive `@optional` sont supposées être obligatoires.

La directive `@optional` s'applique aux méthodes suivantes, jusqu'à la fin de la définition du protocole ou jusqu'à ce qu'une autre directive soit trouvée.

```
@protocol NewProtocol
- (void)protocolMethod:(id)argument;
@optional
- (id)anotherMethod;
- (void)andAnotherMethod:(id)argument;
@required
- (void)lastProtocolMethod;
@end
```

Ce dernier exemple définit un protocole avec deux méthodes facultatives et deux méthodes requises.

Conforme aux protocoles

La syntaxe suivante indique qu'une classe adopte un protocole, en utilisant des crochets.

```
@interface NewClass : NSObject <NewProtocol>
...
@end
```

Cela signifie que toute instance de NewClass répondra aux méthodes déclarées dans son interface mais fournira également une implémentation pour toutes les méthodes requises de `NewProtocol` .

Il est également possible pour une classe de se conformer à plusieurs protocoles en les séparant par une virgule.

```
@interface NewClass : NSObject <NewProtocol, AnotherProtocol, MyProtocol>
...
@end
```

Comme lorsque vous vous conformez à un seul protocole, la classe doit implémenter chaque méthode requise de chaque protocole et chaque méthode facultative que vous choisissez d'implémenter.

Déclarations avant

Il est possible de déclarer le nom du protocole sans méthodes:

```
@protocol Person;
```

utilisez votre code (définition de classe, etc.):

```
@interface World : NSObject
@property (strong, nonatomic) NSArray<id<some>> *employees;
@end
```

et plus tard définir la méthode du protocole quelque part dans votre code:

```
@protocol Person
- (NSString *)gender;
- (NSString *)name;
@end
```

C'est utile lorsque vous n'avez pas besoin de connaître les détails des protocoles avant d'importer ce fichier avec la définition du protocole. Ainsi, votre fichier d'en-tête de classe reste clair et contient uniquement les détails de la classe.

Vérification de l'existence d'implémentations de méthodes facultatives

```
if ([object respondsToSelector:@selector(someOptionalMethodInProtocol:)])
{
    [object someOptionalMethodInProtocol:argument];
}
```

Vérifiez que le protocole est conforme

Retourne un booléen indiquant si la classe est conforme au protocole:

```
[MyClass conformsToProtocol:@protocol(MyProtocol) ];
```

Lire Protocoles en ligne: https://riptutorial.com/fr/objective-c/topic/448/protocoles

Chapitre 47: Protocoles et délégués

Remarques

Les protocoles et les *délégués* sont deux concepts liés mais différents:

Un *protocole* est une interface à laquelle une classe peut se conformer, ce qui signifie que la classe implémente les méthodes répertoriées.

Un *délégué* est généralement un objet anonyme conforme à un protocole.

L'application de *Délégué* appelée *Délégation* est un modèle de conception.

À une extrémité , nous avons le concept *d'héritage* qui crée un couplage étroit entre la sous - classe et sa superclasse alors que modèle de conception *délégation* constitue une alternative pour éviter ce couplage serré à l' aide que nous pouvons créer une relation beaucoup plus souple basée sur les objets *délégués* anonymes.

Examples

Mise en œuvre des protocoles et du mécanisme de délégation.

Supposons que vous ayez deux vues `ViewA` et `ViewB`

L'instance de `ViewB` est créée dans `ViewA` , de sorte que `ViewA` puisse envoyer un message à `ViewB's` instance `ViewB's` , mais pour que l'inverse se produise, nous devons implémenter la délégation (de sorte que l'utilisation de `ViewB's` instance de délégué `ViewB's` puisse envoyer un message à `ViewA`)

Suivez ces étapes pour implémenter la délégation

1. Dans `ViewB` créez le protocole en tant que

```
 @protocol ViewBDelegate

-(void) exampleDelegateMethod;

 @end
```

2. Déclarez le délégué dans la classe d'expéditeur

```
 @interface ViewB : UIView
 @property (nonatomic, weak) id< ViewBDelegate > delegate;
 @end
```

3. Adoptez le protocole en classe ViewA

```
@interfac ViewA: UIView < ViewBDelegate >
```

4. Définir le délégué

```
-(void) anyFunction
{
    // create Class ViewB's instance and set the delegate
    [viewB setDelegate:self];
}
```

5. Implémenter la méthode delegate dans la classe `ViewA`

```
-(void) exampleDelegateMethod
{
    // will be called by Class ViewB's instance
}
```

6. Utilisez la méthode dans la classe `ViewB` pour appeler la méthode déléguée en tant que

```
-(void) callDelegateMethod
{
    [delegate exampleDelegateMethod];
    //assuming the delegate is assigned otherwise error
}
```

Lire Protocoles et délégués en ligne: https://riptutorial.com/fr/objective-c/topic/7832/protocoles-et-delegues

Chapitre 48: Singletons

Introduction

Assurez-vous de lire ce fil (Qu'est-ce qui est si mauvais sur les singletons?) Avant de l'utiliser.

Examples

Utiliser Grand Central Dispatch (GCD)

GCD garantira que votre singleton ne sera instancié qu'une seule fois, même s'il est appelé depuis plusieurs threads. Insérez ceci dans n'importe quelle classe pour une instance singleton appelée `shared` .

```
+ (instancetype)shared {

    // Variable that will point to the singleton instance. The `static`
    // modifier makes it behave like a global variable: the value assigned
    // to it will "survive" the method call.
    static id _shared;

    static dispatch_once_t _onceToken;
    dispatch_once(&_onceToken, ^{

        // This block is only executed once, in a thread-safe way.
        // Create the instance and assign it to the static variable.
        _shared = [self new];
    });

    return _shared;
}
```

Créer la classe Singleton et l'empêcher d'avoir plusieurs instances utilisant alloc / init.

Nous pouvons créer la classe Singleton de telle sorte que les développeurs soient obligés d'utiliser l'instance partagée (objet singleton) au lieu de créer leurs propres instances.

```
@implementation MySingletonClass

+ (instancetype)sharedInstance
{
    static MySingletonClass *_sharedInstance = nil;
    static dispatch_once_t oncePredicate;
    dispatch_once(&oncePredicate, ^{
        _sharedInstance = [[self alloc] initClass];
    });

    return _sharedInstance;
}
```

```
-(instancetype)initClass
{
    self = [super init];
    if(self)
    {
        //Do any additional initialization if required
    }
    return self;
}

- (instancetype)init
{
    @throw [NSException exceptionWithName:@"Not designated initializer"
                                   reason:@"Use [MySingletonClass sharedInstance]"
                                 userInfo:nil];
    return nil;
}
@end



/*Following line will throw an exception
  with the Reason:"Use [MySingletonClass sharedInstance]"
  when tried to alloc/init directly instead of using sharedInstance */
MySingletonClass *mySingletonClass = [[MySingletonClass alloc] init];
```

Créer Singleton et l'empêcher également d'avoir plusieurs instances utilisant alloc / init, new.

```
//MySingletonClass.h
@interface MYSingletonClass : NSObject

+ (instancetype)sharedInstance;

-(instancetype)init NS_UNAVAILABLE;

-(instancetype)new NS_UNAVAILABLE;

@end

//MySingletonClass.m

@implementation MySingletonClass

+ (instancetype)sharedInstance
{
    static MySingletonClass *_sharedInstance = nil;
    static dispatch_once_t oncePredicate;
    dispatch_once(&oncePredicate, ^{
        _sharedInstance = [[self alloc]init];
    });

    return _sharedInstance;
}
-(instancetype)init
{
    self = [super init];
    if(self)
    {
```

```
        //Do any additional initialization if required
    }
    return self;
}
@end
```

Lire Singletons en ligne: https://riptutorial.com/fr/objective-c/topic/2834/singletons

Chapitre 49: Spécificateurs de format

Introduction

Les spécificateurs de format sont utilisés dans Objective-c pour implanter des valeurs d'objet dans une chaîne.

Syntaxe

- %@ //Chaîne
- % d // Entier 32 bits signé
- % D // Entier 32 bits signé
- % u // Entier non signé de 32 bits
- % U // Entier non signé de 32 bits
- % x // Entier 32 bits non signé au format hexadécimal minuscule
- % X // Entier 32 bits non signé au format hexadécimal UPPERCASE
- % o // Entier non signé de 32 bits au format octal
- % O // Entier non signé de 32 bits au format octal
- % f // nombre à virgule flottante 64 bits
- % F // nombre à virgule flottante 64 bits imprimé en notation décimale
- % e // Nombre à virgule flottante de 64 bits au format de notation scientifique minuscule
- % E // nombre à virgule flottante 64 bits au format de notation scientifique UPPERCASE
- % g // cas particulier% e qui utilise% f lorsque moins de 4 figures sont disponibles, sinon% e
- % G // cas particulier% E qui utilise% f lorsque moins de 4 figures sont disponibles, sinon% E
- % c // caractère non signé de 8 bits
- % C // Unité de code UTF-16 16 bits
- % s // Chaîne UTF8
- % S // variante de 16 bits de% s
- % p // Pointeur vide en format hexadécimal minuscule avec '0x' en tête
- % zx // cas spécial% p qui supprime le premier '0x' (à utiliser avec la distribution sans type)
- % un nombre à virgule flottante // 64 bits en notation scientifique avec «0x» et un chiffre hexadécimal avant le point décimal à l'aide d'un «p» pour noter l'exposant.
- % Un nombre à virgule flottante // 64 bits en notation scientifique avec «0x» et un chiffre hexadécimal avant le point décimal à l'aide d'un «P» pour noter l'exposant.

Remarques

En raison de la nature des spécificateurs de format, si vous souhaitez inclure le symbole de pourcentage (%) dans votre chaîne, vous devez y échapper en utilisant un deuxième symbole de pourcentage.

Exemple:

```
int progress = 45;//percent
NSString *progressString = [NSString stringWithFormat:@"Progress: %i%%", (int)progress];

NSLog(progressString);//logs "Progress: 45%"
```

Aucun spécificateur de format pour le type BOOL n'existe.

Les solutions à usage commun comprennent:

```
BOOL myBool = YES;
NSString *boolState = [NSString stringWithFormat:@"BOOL state: %@", myBool?@"true":@"false"];

NSLog(boolState);//logs "true"
```

Qui utilise un opérateur ternaire pour lancer une chaîne équivalente.

```
BOOL myBool = YES;
NSString *boolState = [NSString stringWithFormat:@"BOOL state: %i", myBool];

NSLog(boolState);//logs "1" (binary)
```

Qui utilise un moulage (int) pour implanter un équivalent binaire.

Examples

Exemple entier -% i

```
int highScore = 57;
NSString *scoreBoard = [NSString stringWithFormat:@"HighScore: %i", (int)highScore];

NSLog(scoreBoard);//logs "HighScore: 57"
```

Lire Spécificateurs de format en ligne: https://riptutorial.com/fr/objective-c/topic/9048/specificateurs-de-format

Chapitre 50: Structs

Syntaxe

- typedef struct { *typeA propertyA* ; *typeB propertyB* ; ...} *StructName*

Remarques

Dans Objective C, vous devriez presque toujours utiliser un objet au lieu d'une structure. Cependant, il existe encore des cas où l'utilisation d'une structure est meilleure, comme par exemple:

- Lorsque vous allez créer et détruire de nombreuses valeurs du type (struct), vous avez donc besoin de bonnes performances et d'une faible utilisation de la mémoire
    - Les structures sont plus rapides à créer et à utiliser car lors de l'appel d'une méthode sur un objet, la méthode doit être déterminée à l'exécution.
    - Les structures prennent moins de taille car les objets ont une propriété supplémentaire `isa` , qui contient leur classe
- Lorsque la valeur n'a que quelques propriétés et une petite taille totale (prenez `CGSize` ; elle a 2 flottants de 4 octets chacun, elle peut donc prendre 8 octets), et va être utilisée beaucoup premier point)
- Lorsque vous pouvez utiliser des unions ou des champs de bits et, plus important encore, vous avez besoin de *la taille enregistrée* car vous avez besoin d'une petite quantité de mémoire (liée au premier point)
- Lorsque vous voulez *vraiment* stocker un tableau à l'intérieur de la structure, puisque les objets Objective-C ne peuvent pas stocker directement les tableaux C. Cependant, notez que vous pouvez toujours "indirectement" obtenir un tableau dans un objet Objective-C en en faisant une référence (par exemple, `type *` à la place du `type[]` C-array `type[]`)
- Lorsque vous devez communiquer avec un autre code, tel qu'une bibliothèque, cela est codé en C; Les structures sont complètement implémentées en C mais les objets ne sont pas

Examples

CGPoint

Un bon exemple de struct est `CGPoint` ; c'est une valeur simple qui représente un point à deux dimensions. Il a 2 propriétés, `x` et `y` , et peut être écrit comme

```
typedef struct {
    CGFloat x;
    CGFloat y;
} CGPoint;
```

Si vous avez déjà utilisé le développement d'applications Objective-C pour Mac ou iOS, vous avez certainement rencontré `CGPoint` . `CGPoint` maintient la position de presque tout à l'écran, des vues et des contrôles aux objets d'un jeu, en passant par les changements de dégradé. Cela signifie que les `CGPoint` sont beaucoup utilisés. Cela est encore plus vrai avec des jeux très performants. Ces jeux ont tendance à avoir beaucoup d'objets et tous ces objets ont besoin de positions. Ces positions sont souvent soit `CGPoint` s, soit un autre type de structure qui transmet un point (comme un point tridimensionnel pour les jeux 3D).

Des points comme `CGPoint` peuvent facilement être représentés comme des objets, comme

```
@interface CGPoint {
    CGFloat x;
    CGFloat y;
}

... //Point-related methods (e.g. add, isEqualToPoint, etc.)

@property(nonatomic, assign)CGFloat x;
@property(nonatomic, assign)CGFloat y;

@end

@implementation CGPoint

@synthesize x, y;

...

@end
```

Cependant, si `CGPoint` était utilisé de cette manière, la création et la manipulation de points prendraient beaucoup plus de temps. Dans les programmes plus petits et plus rapides, cela ne causerait pas vraiment de différence, et dans ce cas, il serait correct ou même préférable d'utiliser des points d'objet. Mais dans les grands programmes où les points sont souvent utilisés, l'utilisation d'objets en tant que points peut vraiment nuire aux performances, ce qui ralentit le programme et gaspille également de la mémoire, ce qui pourrait forcer le programme à planter.

Définition d'une structure et accès aux membres de la structure

Le format de l'instruction struct est le suivant:

```
struct [structure tag]
{
   member definition;
   member definition;
   ...
   member definition;
} [one or more structure variables];
```

Exemple: déclarez la structure ThreeFloats:

```
   typedef struct {
```

```
    float x, y, z;
} ThreeFloats;

@interface MyClass
- (void)setThreeFloats:(ThreeFloats)threeFloats;
- (ThreeFloats)threeFloats;
@end
```

Envoyer une instance de MyClass au message valueForKey: avec le paramètre @ "threeFloats" invoquera la méthode MyClass threeFloats et renverra le résultat encapsulé dans une NSValue.

Lire Structs en ligne: https://riptutorial.com/fr/objective-c/topic/3792/structs

Chapitre 51: Test unitaire en utilisant Xcode

Remarques

Dépendances :

- Si l'application utilise des bibliothèques tierces ou des pods de cacao, ces bibliothèques ou ces modules doivent également être installés pour le test.
- La classe de test (Test Suit) étend XCTestCase.

Soyez brossé avant de commencer:

- Toutes les classes de test ont deux méthodes dans setUp & tearDown.
- setUp s'exécute avant chaque test et déchirure après chaque test.
- Les cas de test s'exécutent par ordre alphabétique.
- Dans le cadre du développement piloté par les tests, il est préférable de créer des données de test factices en premier.
- Les méthodes de cas de test commencent par le mot clé "test".
- Les méthodes de test n'acceptent aucun paramètre et ne renvoient aucune valeur.

Annexe:

Il existe plusieurs autres méthodes pour comparer le résultat attendu et le résultat réel d'une opération. Certaines de ces méthodes sont énumérées ci-dessous:

- XCTAssertNil (expression, commentaire) génère un échec si expression! = Nil.
- XCTAssertNotNil (expression, commentaire) génère un échec si expression = nil.
- XCTAssert (expression, commentaire) génère un échec si l'expression == false.
- XCTAssertTrue (expression, commentaire) génère un échec si l'expression == false.
- XCTAssertFalse (expression, commentaire) génère un échec si expression! = False.
- XCTAssertEqualObjects (expression1, expression2, comment) génère un échec si expression1 n'est pas égal à expression2.
- XCTAssertEqualObjects (expression1, expression2, comment) génère un échec si expression1 est égal à expression2.
- XCTAssertNotEqual (expression1, expression2, comment) génère un échec si expression1 == expression2.
- XCTAssertEqual (expression1, expression2, comment) génère un échec si expression1! = Expression2.
- XCTAssertGreaterThanOrEqual (expression1, expression2, comment) génère un échec lorsque (expression1 <expression2).

Examples

Tester un bloc de code ou une méthode:

- Importez la classe qui contient la méthode à tester.
- Effectuez l'opération avec des données factices.
- Comparez maintenant le résultat de l'opération avec le résultat attendu.

```
- (void)testReverseString{
NSString *originalString = @"hi_my_name_is_siddharth";
NSString *reversedString = [self.someObject reverseString:originalString];
NSString *expectedReversedString = @"htrahddis_si_eman_ym_ih";
XCTAssertEqualObjects(expectedReversedString, reversedString, @"The reversed string did not
match the expected reverse");
}
```

> **Envoyez les données factices à la méthode à tester si nécessaire, puis comparez les résultats attendus et réels.**

Test du bloc de code asynchrone:

```
- (void)testDoSomethingThatTakesSomeTime{
XCTestExpectation *completionExpectation = [self expectationWithDescription:@"Long method"];
[self.someObject doSomethingThatTakesSomeTimesWithCompletionBlock:^(NSString *result) {
    XCTAssertEqualObjects(@"result", result, @"Result was not correct!");
    [completionExpectation fulfill];
}];
[self waitForExpectationsWithTimeout:5.0 handler:nil];
}
```

- Transmettez les données factices à la méthode à tester si nécessaire.
- Le test sera suspendu ici, exécutant la boucle d'exécution, jusqu'à ce que le délai d'attente soit atteint ou que toutes les attentes soient satisfaites.
- Le délai d'attente correspond au délai prévu pour la réponse du bloc asynchrone.

Mesure de la performance d'un bloc de code:

1. Pour les méthodes synchrones:

```
- (void)testPerformanceReverseString {
    NSString *originalString = @"hi_my_name_is_siddharth";
    [self measureBlock:^{
        [self.someObject reverseString:originalString];
    }];
}
```

2. Pour les méthodes asynchrones:

```
- (void)testPerformanceOfAsynchronousBlock {
   [self measureMetrics:@[XCTPerformanceMetric_WallClockTime] automaticallyStartMeasuring:YES
forBlock:^{

    XCTestExpectation *expectation = [self
expectationWithDescription:@"performanceTestWithResponse"];

    [self.someObject doSomethingThatTakesSomeTimesWithCompletionBlock:^(NSString *result) {
        [expectation fulfill];
    }];
    [self waitForExpectationsWithTimeout:5.0 handler:^(NSError *error) {
    }];
}];
}
```

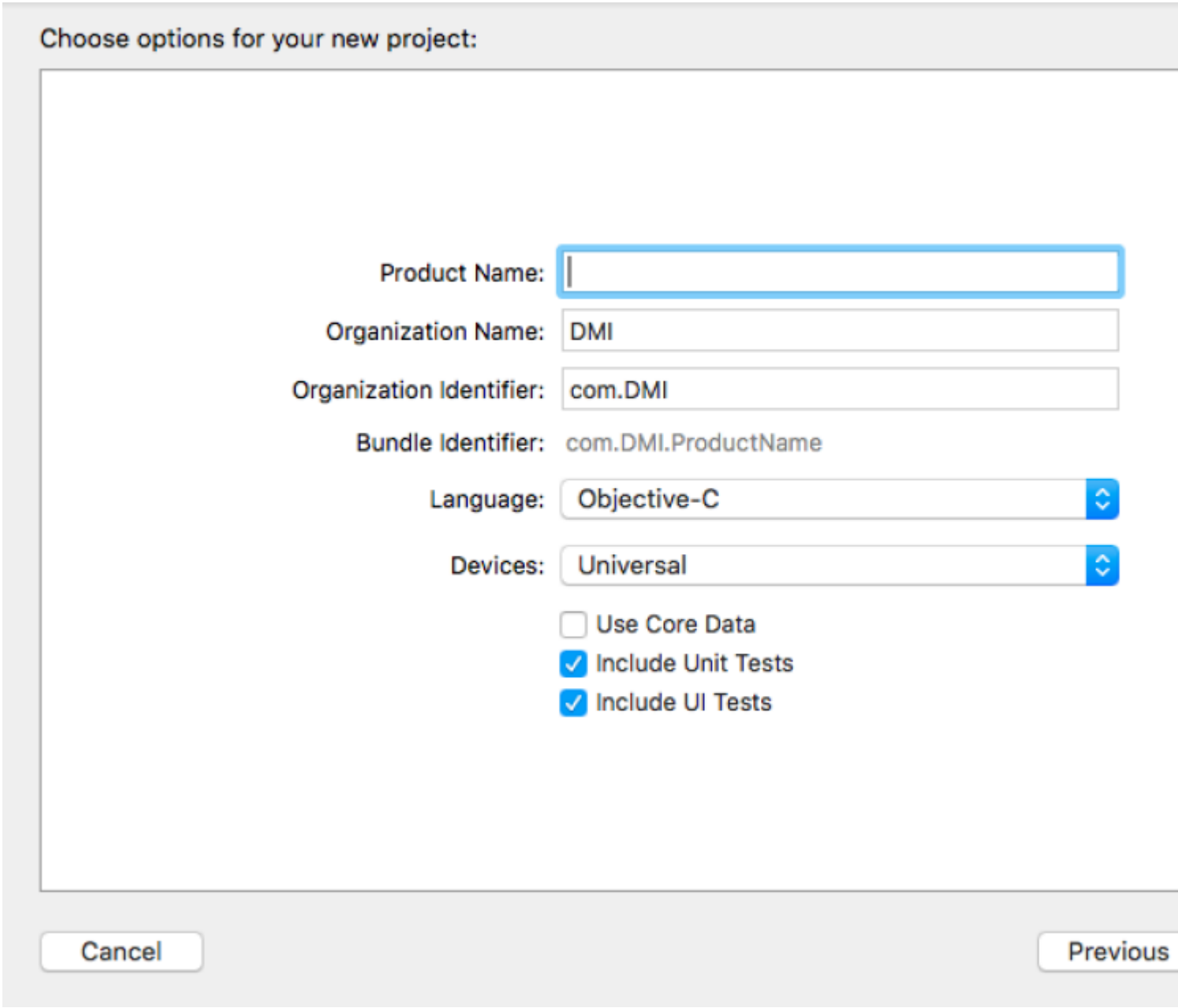
- Ce bloc de mesure de performance est exécuté 10 fois de suite, puis la moyenne est calculée et, sur la base de cette performance moyenne, le résultat est créé et la référence est acceptée pour une évaluation ultérieure.
- Le résultat de la performance est comparé aux résultats des tests précédents et à la ligne de base avec un écart-type maximal personnalisable.

Exécution de combinaisons d'essai:

Exécutez tous les tests en choisissant Produit> Test. Cliquez sur l'icône Test Navigator pour afficher l'état et les résultats des tests. Vous pouvez ajouter une cible de test à un projet (ou ajouter une classe à un test) en cliquant sur le bouton Ajouter (plus) dans le coin inférieur gauche du navigateur de test. Pour afficher le code source d'un test particulier, sélectionnez-le dans la liste de tests. Le fichier s'ouvre dans l'éditeur de code source.

Remarque:

Assurez-vous que la case Inclure le cas de test unitaire est cochée lors de la création d'un nouveau projet, comme indiqué ci-dessous:



Lire Test unitaire en utilisant Xcode en ligne: https://riptutorial.com/fr/objective-c/topic/5160/test-unitaire-en-utilisant-xcode

Chapitre 52: Types de données de base

Syntaxe

- BOOL havePlutonium = YES; // Assigment direct
- BOOL fastEnough = (car.speedInMPH> = 88); // Expression de comparaison
- BOOL fluxCapacitorActive = (havePlutonium && fastEnough); // Expression booléenne
-
- id somethingWicked = [witchesCupboard lastObject]; // Récupère un objet non typé
- id powder = prepareWickedIngredient (quelquechose) // Passe et retour
- if ([ingrédient isKindOfClass: [classe Toad]]) {// Tester le type d'exécution

Examples

BOOL

Le type `BOOL` est utilisé pour les valeurs booléennes dans Objective-C. Il a deux valeurs, `YES` et `NO` , contrairement aux valeurs les plus courantes "true" et "false".

Son comportement est simple et identique à celui du langage C.

```
BOOL areEqual = (1 == 1);    // areEqual is YES
BOOL areNotEqual = !areEqual    // areNotEqual is NO
NSCAssert(areEqual, "Mathematics is a lie");    // Assertion passes

BOOL shouldFlatterReader = YES;
if (shouldFlatterReader) {
    NSLog(@"Only the very smartest programmers read this kind of material.");
}
```

Un `BOOL` est une primitive et ne peut donc pas être stocké directement dans une collection Foundation. Il doit être enveloppé dans un `NSNumber` . Clang fournit une syntaxe spéciale pour ceci:

```
NSNumber * yes = @YES;    // Equivalent to [NSNumber numberWithBool:YES]
NSNumber * no = @NO;    // Equivalent to [NSNumber numberWithBool:NO]
```

L'implémentation `BOOL` est directement basée sur C's, en ce sens qu'il s'agit d'un typedef du type `bool` standard C99. Les valeurs `YES` et `NO` sont définies respectivement pour `__objc_yes` et `__objc_no` . Ces valeurs spéciales sont des instructions intégrées au compilateur introduites par Clang, qui sont traduites en `(BOOL)1` et `(BOOL)0` . S'ils ne sont pas disponibles, `YES` et `NO` sont définis directement sous forme de nombre entier. Les définitions se trouvent dans l'entête d'exécution Objective-C objc.h

id

`id` est le pointeur d'objet générique, un type Objective-C représentant "n'importe quel objet". Une

instance de toute classe Objective-C peut être stockée dans une variable `id` . Un `id` et tout autre type de classe peuvent être assignés sans conversion:

```
id anonymousSurname = @"Doe";
NSString * surname = anonymousSurname;
id anonymousFullName = [NSString stringWithFormat:@"%@, John", surname];
```

Cela devient pertinent lors de la récupération d'objets à partir d'une collection. Les types de retour des méthodes comme `objectAtIndex:` sont `id` pour exactement cette raison.

```
DataRecord * record = [records objectAtIndex:anIndex];
```

Cela signifie également qu'une méthode ou un paramètre de fonction saisi comme `id` peut accepter n'importe quel objet.

Lorsqu'un objet est saisi en tant `id` , tout message connu peut lui être transmis: la méthode dispatch ne dépend pas du type de compilation.

```
NSString * extinctBirdMaybe =
               [anonymousSurname stringByAppendingString:anonymousSurname];
```

Un message auquel l'objet ne répond pas réellement provoquera une exception à l'exécution, bien sûr.

```
NSDate * nope = [anonymousSurname addTimeInterval:10];
// Raises "Does not respond to selector" exception
```

Protéger contre les exceptions.

```
NSDate * nope;
if([anonymousSurname isKindOfClass:[NSDate class]]){
    nope = [anonymousSurname addTimeInterval:10];
}
```

Le type d' `id` est défini dans objc.h

```
typedef struct objc_object {
    Class isa;
} *id;
```

SEL

Les sélecteurs sont utilisés comme identificateurs de méthode dans Objective-C.

Dans l'exemple ci-dessous, il y a deux sélecteurs. `new` et `setName:`

```
Person* customer = [Person new];
[customer setName:@"John Doe"];
```

Chaque paire de parenthèses correspond à un message envoyé. Sur la première ligne, nous envoyons un message contenant le `new` sélecteur à la classe `Person` et sur la deuxième ligne, nous envoyons un message contenant le `setName:` selector et une chaîne. Le destinataire de ces messages utilise le sélecteur pour rechercher la bonne action à effectuer.

La plupart du temps, la transmission de messages à l'aide de la syntaxe entre parenthèses est suffisante, mais vous devez parfois travailler avec le sélecteur lui-même. Dans ces cas, le type `SEL` peut être utilisé pour contenir une référence au sélecteur.

Si le sélecteur est disponible à la compilation, vous pouvez utiliser `@selector()` pour obtenir une référence.

```
SEL s = @selector(setName:);
```

Et si vous avez besoin de trouver le sélecteur à l'exécution, utilisez NSSelectorFromString.

```
SEL s NSSelectorFromString(@"setName:");
```

Lorsque vous utilisez NSSelectorFromString, veillez à envelopper le nom du sélecteur dans un NSString.

Il est couramment utilisé pour vérifier si un délégué implémente une méthode facultative.

```
if ([self.myDelegate respondsToSelector:@selector(doSomething)]) {
    [self.myDelegate doSomething];
}
```

IMP (pointeur d'implémentation)

IMP est un type C faisant référence à l'implémentation d'une méthode, également appelée pointeur d'implémentation. C'est un pointeur sur le début d'une implémentation de méthode.

Syntaxe:

```
id (*IMP)(id, SEL, …)
```

IMP est défini par:

```
typedef id (*IMP)(id self,SEL _cmd,…);
```

Pour accéder à cet IMP, le message **«methodForSelector»** peut être utilisé.

Exemple 1:

```
IMP ImpDoSomething = [myObject methodForSelector:@selector(doSomething)];
```

La méthode adressée par l'IMP peut être appelée en déréférencant l'IMP.

```
ImpDoSomething(myObject, @selector(doSomething));
```

Donc, ces appels sont égaux:

```
 myImpDoSomething(myObject, @selector(doSomething));
[myObject doSomething]
[myObject performSelector:mySelector]
[myObject performSelector:@selector(doSomething)]
[myObject performSelector:NSSelectorFromString(@"doSomething")];
```

Exemple: 2:

```
SEL otherWaySelector = NSSelectorFromString(@"methodWithFirst:andSecond:andThird:");

IMP methodImplementation  = [self methodForSelector:otherWaySelector];

result = methodImplementation( self,
                          betterWaySelector,
                          first,
                          second,
                          third );

NSLog(@"methodForSelector : %@", result);
```

Ici, nous appelons [NSObject methodForSelector qui nous renvoie un pointeur sur la fonction C qui implémente réellement la méthode, que nous pouvons appeler ensuite directement.

NSInteger et NSUInteger

Le NSInteger est juste un typedef pour un int ou un long selon l'architecture. La même chose vaut pour un NSUInteger qui est un typedef pour les variantes non signées. Si vous vérifiez le NSInteger, vous verrez ce qui suit:

```
#if __LP64__ || (TARGET_OS_EMBEDDED && !TARGET_OS_IPHONE) || TARGET_OS_WIN32 ||
NS_BUILD_32_LIKE_64
typedef long NSInteger;
typedef unsigned long NSUInteger;
#else
typedef int NSInteger;
typedef unsigned int NSUInteger;
#endif
```

La différence entre un signe signé et un signe unsigned int ou long est qu'un entier signé ou long peut contenir des valeurs négatives. L'intervalle de l'int est -2 147 483 648 à 2 147 483 647 tandis que le unsigned int a une plage de 0 à 4 294 967 295. La valeur est doublée parce que le premier bit n'est plus utilisé pour dire que la valeur est négative ou pas. Pour une longue et NSInteger sur les architectures 64 bits, la plage est beaucoup plus large.

La plupart des méthodes fournies par Apple renvoient un entier NS (U) au-dessus de l'int normal. Vous recevrez un avertissement si vous essayez de le convertir en un entier normal, car vous perdrez en précision si vous exécutez une architecture 64 bits. Ce n'est pas le cas dans la plupart des cas, mais il est plus facile d'utiliser NS (U) Integer. Par exemple, la méthode count sur un

tableau renvoie un NSUInteger.

```
NSNumber *iAmNumber = @0;

NSInteger iAmSigned = [iAmNumber integerValue];
NSUInteger iAmUnsigned = [iAmNumber unsignedIntegerValue];

NSLog(@"%ld", iAmSigned); // The way to print a NSInteger.
NSLog(@"%lu", iAmUnsigned); // The way to print a NSUInteger.
```

Tout comme un BOOL, le NS (U) Integer est un type de données primitif, vous devez donc parfois l'envelopper dans un NSNumber, vous pouvez utiliser le @ avant le nombre entier pour le lancer comme ci-dessus et le récupérer en utilisant les méthodes ci-dessous. Mais pour le convertir en NSNumber, vous pouvez également utiliser les méthodes suivantes:

```
[NSNumber numberWithInteger:0];
[NSNumber numberWithUnsignedInteger:0];
```

Lire Types de données de base en ligne: https://riptutorial.com/fr/objective-c/topic/4564/types-de-donnees-de-base

Chapitre 53: Valeur clé Codage / Valeur clé Observation

Examples

Exemple le plus commun de codage de valeur réelle

Key Value Coding est intégré à **NSObject en** utilisant le protocole **NSKeyValueCoding** .

Qu'est-ce que cela signifie?

Cela signifie que tout objet id est capable d'appeler la méthode valueForKey et ses différentes variantes comme valueForKeyPath, etc.

Cela signifie également que tout objet id peut invoquer la méthode setValue et ses différentes variantes.

Exemple:

```
id obj = [[MyClass alloc] init];
id value = [obj valueForKey:@"myNumber"];

int myNumberAsInt = [value intValue];
myNumberAsInt = 53;
[obj setValue:@(myNumberAsInt) forKey:@"myNumber"];
```

Des exceptions:

L'exemple ci-dessus suppose que MyClass possède une propriété NSNumber appelée myNumber. Si myNumber n'apparaît pas dans la définition de l'interface MyClass, une exception NSUndefinedKeyException peut être déclenchée sur les deux lignes 2 et 5 - communément appelée:

```
this class is not key value coding-compliant for the key myNumber.
```

Pourquoi c'est si puissant:

Vous pouvez écrire du code qui peut accéder dynamiquement aux propriétés d'une classe, sans avoir besoin d'interface pour cette classe. Cela signifie qu'une vue de table peut afficher les valeurs de toutes les propriétés d'un objet dérivé de NSObject, à condition que ses noms de propriété soient fournis dynamiquement à l'exécution.

Dans l'exemple ci-dessus, le code peut aussi bien fonctionner sans que MyClass soit disponible et que ID type obj soit disponible pour le code d'appel.

Observation de la valeur clé

Mettre en place l'observation de la valeur clé.

Dans ce cas, nous voulons observer le `contentOffset` sur un objet que notre observateur possède

```
//
// Class to observe
//
@interface XYZScrollView: NSObject
@property (nonatomic, assign) CGPoint contentOffset;
@end

@implementation XYZScrollView
@end


//
// Class that will observe changes
//
@interface XYZObserver: NSObject
@property (nonatomic, strong) XYZScrollView *scrollView;
@end

@implementation XYZObserver

// simple way to create a KVO context
static void *XYZObserverContext = &XYZObserverContext;


// Helper method to add self as an observer to
// the scrollView's contentOffset property
- (void)addObserver {

    // NSKeyValueObservingOptions
    //
    // - NSKeyValueObservingOptionNew
    // - NSKeyValueObservingOptionOld
    // - NSKeyValueObservingOptionInitial
    // - NSKeyValueObservingOptionPrior
    //
    // can be combined:
    // (NSKeyValueObservingOptionNew | NSKeyValueObservingOptionOld)

    NSString *keyPath = NSStringFromSelector(@selector(contentOffset));
    NSKeyValueObservingOptions options = NSKeyValueObservingOptionNew;

    [self.scrollView addObserver: self
                      forKeyPath: keyPath
                         options: options
                         context: XYZObserverContext];
}

- (void)observeValueForKeyPath:(NSString *)keyPath ofObject:(id)object
change:(NSDictionary<NSString *,id> *)change context:(void *)context {

    if (context == XYZObserverContext) { // check the context

        // check the keyPath to see if it's any of the desired keyPath's.
        // You can observe multiple keyPath's
        if ([keyPath isEqualToString: NSStringFromSelector(@selector(contentOffset))]) {
```

```
            // change dictionary keys:
            // - NSKeyValueChangeKindKey
            // - NSKeyValueChangeNewKey
            // - NSKeyValueChangeOldKey
            // - NSKeyValueChangeIndexesKey
            // - NSKeyValueChangeNotificationIsPriorKey

            // the change dictionary here for a CGPoint observation will
            // return an NSPoint, so we can take the CGPointValue of it.
            CGPoint point = [change[NSKeyValueChangeNewKey] CGPointValue];

            // handle point
        }

    } else {

        // if the context doesn't match our current object's context
        // we want to pass the observation parameters to super
        [super observeValueForKeyPath: keyPath
                             ofObject: object
                               change: change
                              context: context];
    }
}

// The program can crash if an object is not removed as observer
// before it is dealloc'd
//
// Helper method to remove self as an observer of the scrollView's
// contentOffset property
- (void)removeObserver {
    NSString *keyPath = NSStringFromSelector(@selector(contentOffset));
    [self.scrollView removeObserver: self forKeyPath: keyPath];
}

@end
```

Interrogation des données KVC

```
if ([[dataObject objectForKey:@"yourVariable"] isEqualToString:"Hello World"]) {
    return YES;
} else {
    return NO;
}
```

Vous pouvez interroger les valeurs stockées à l'aide de KVC rapidement et facilement, sans devoir les extraire ou les convertir en variables locales.

Opérateurs de collecte

Les opérateurs de collecte peuvent être utilisés dans un chemin de clé KVC pour effectuer une opération sur une propriété de type «collection-type» (`NSArray` , `NSSet` et similaire). Par exemple, une opération courante consiste à compter les objets dans une collection. Pour ce faire, vous utilisez l' *opérateur de collecte* `@count` :

```
self.array = @[@5, @4, @3, @2, @1];
NSNumber *count = [self.array valueForKeyPath:@"@count"];
NSNumber *countAlt = [self valueForKeyPath:@"array.@count"];
// count == countAlt == 5
```

Bien que cela soit complètement redondant ici (nous aurions pu simplement accéder à la propriété `count`), cela *peut* être utile à l'occasion, bien que cela soit rarement nécessaire. Il existe cependant des opérateurs de collecte beaucoup plus utiles, à savoir `@max` , `@min` , `@sum` , `@avg` et la famille `@unionOf` . Il est important de noter que ces opérateurs requièrent *également* un chemin de clé séparé *suivant* l'opérateur pour fonctionner correctement. Voici une liste de ceux-ci et le type de données avec lesquelles ils travaillent:

Opérateur	Type de données
`@count`	(aucun)
`@max`	`NSNumber` , `NSDate` , `int` (et apparenté), etc.
`@min`	`NSNumber` , `NSDate` , `int` (et apparenté), etc.
`@sum`	`NSNumber` , `int` (et apparenté), `double` (et connexe), etc.
`@avg`	`NSNumber` , `int` (et apparenté), `double` (et connexe), etc.
`@unionOfObjects`	`NSArray` , `NSSet` , etc.
`@distinctUnionOfObjects`	`NSArray` , `NSSet` , etc.
`@unionOfArrays`	`NSArray<NSArray*>`
`@distinctUnionOfArrays`	`NSArray<NSArray*>`
`@distinctUnionOfSets`	`NSSet<NSSet*>`

`@max` et `@min` respectivement la valeur la plus élevée ou la plus basse d'une propriété des objets de la collection. Par exemple, regardez le code suivant:

```
// “Point” class used in our collection
@interface Point : NSObject

@property NSInteger x, y;

+ (instancetype)pointWithX:(NSInteger)x y:(NSInteger)y;

@end

...

self.points = @[[Point pointWithX:0 y:0],
                [Point pointWithX:1 y:-1],
                [Point pointWithX:5 y:-6],
                [Point pointWithX:3 y:0],
                [Point pointWithX:8 y:-4],
```

```
];

NSNumber *maxX = [self valueForKeyPath:@"points.@max.x"];
NSNumber *minX = [self valueForKeyPath:@"points.@min.x"];
NSNumber *maxY = [self valueForKeyPath:@"points.@max.y"];
NSNumber *minY = [self valueForKeyPath:@"points.@min.y"];

NSArray<NSNumber*> *boundsOfAllPoints = @[maxX, minX, maxY, minY];

...
```

Avec seulement 4 lignes de code et une base pure, avec la puissance des opérateurs de collection Key-Value Coding, nous avons pu extraire un rectangle qui encapsule tous les points de notre tableau.

Il est important de noter que ces comparaisons sont effectuées en invoquant la méthode `compare:` sur les objets. Par conséquent, si vous souhaitez rendre votre propre classe compatible avec ces opérateurs, vous devez implémenter cette méthode.

`@sum` vous pouvez probablement le deviner, `@sum` ajoute toutes les valeurs d'une propriété.

```
@interface Expense : NSObject

@property NSNumber *price;

+ (instancetype)expenseWithPrice:(NSNumber *)price;

@end

...

self.expenses = @[[Expense expenseWithPrice:@1.50],
                  [Expense expenseWithPrice:@9.99],
                  [Expense expenseWithPrice:@2.78],
                  [Expense expenseWithPrice:@9.99],
                  [Expense expenseWithPrice:@24.95]
];

NSNumber *totalExpenses = [self valueForKeyPath:@"expenses.@sum.price"];
```

Ici, nous avons utilisé `@sum` pour trouver le prix total de toutes les dépenses dans le tableau. Si nous voulons plutôt trouver le prix moyen que nous payons pour chaque dépense, nous pouvons utiliser `@avg` :

```
NSNumber *averagePrice = [self valueForKeyPath:@"expenses.@avg.price"];
```

Enfin, il y a la famille `@unionOf` . Il y a cinq opérateurs différents dans cette famille, mais ils fonctionnent tous essentiellement de la même façon, avec seulement de petites différences entre eux. Tout d'abord, il y a `@unionOfObjects` qui renverra un tableau des propriétés des objets d'un tableau:

```
// See "expenses" array above

NSArray<NSNumber*> *allPrices = [self valueForKeyPath:
```

```
    @"expenses.@unionOfObjects.price"];

// Equal to @[ @1.50, @9.99, @2.78, @9.99, @24.95 ]
```

`@distinctUnionOfObjects` fonctionne de la même manière que `@unionOfObjects` , mais il supprime les doublons:

```
NSArray<NSNumber*> *differentPrices = [self valueForKeyPath:
    @"expenses.@distinctUnionOfObjects.price"];

// Equal to @[ @1.50, @9.99, @2.78, @24.95 ]
```

Et enfin, les trois derniers opérateurs de la famille `@unionOf` vont aller plus loin et renvoient un tableau de valeurs trouvées pour une propriété contenue dans des tableaux imbriqués en double:

```
NSArray<NSArray<Expense*,Expense*>*> *arrayOfArrays =
    @[
        @[ [Expense expenseWithPrice:@19.99],
           [Expense expenseWithPrice:@14.95],
           [Expense expenseWithPrice:@4.50],
           [Expense expenseWithPrice:@19.99]
         ],

        @[ [Expense expenseWithPrice:@3.75],
           [Expense expenseWithPrice:@14.95]
         ]
     ];

// @unionOfArrays
NSArray<NSNumber*> allPrices = [arrayOfArrays valueForKeyPath:
    @"@unionOfArrays.price"];
// Equal to @[ @19.99, @14.95, @4.50, @19.99, @3.75, @14.95 ];

// @distinctUnionOfArrays
NSArray<NSNumber*> allPrices = [arrayOfArrays valueForKeyPath:
    @"@distinctUnionOfArrays.price"];
// Equal to @[ @19.99, @14.95, @4.50, @3.75 ];
```

Celui qui manque dans cet exemple est `@distinctUnionOfSets` , mais cela fonctionne exactement comme `@distinctUnionOfArrays` , mais fonctionne avec et renvoie `NSSet` s à la place (il n'y a pas de version non `distinct` car dans un ensemble, chaque objet doit être distinct de toute façon).

Et c'est tout! Les opérateurs de collecte peuvent être très puissants s'ils sont utilisés correctement et peuvent éviter d'avoir à parcourir des fichiers inutilement.

Une dernière remarque: vous pouvez également utiliser les opérateurs de collecte standard sur les tableaux de `NSNumber` s (sans accès aux propriétés supplémentaires). Pour ce faire, vous accédez à la pseudo-propriété `self` qui renvoie simplement l'objet:

```
NSArray<NSNumber*> *numbers = @[@0, @1, @5, @27, @1337, @2048];

NSNumber *largest = [numbers valueForKeyPath:@"@max.self"];
NSNumber *smallest = [numbers valueForKeyPath:@"@min.self"];
NSNumber *total = [numbers valueForKeyPath:@"@sum.self"];
```

```
NSNumber *average = [numbers valueForKeyPath:@"@avg.self"];
```

Lire Valeur clé Codage / Valeur clé Observation en ligne: https://riptutorial.com/fr/objective-c/topic/556/valeur-cle-codage---valeur-cle-observation

Crédits

S. No	Chapitres	Contributeurs
1	Premiers pas avec le langage Objective-C	Ali Riahipour, Community, connor, insys, J F, Jeff Wolski, Josh Brown, Josh Caswell, Niraj, StrAbZ, tbodt
2	Analyse XML	iphonic, Stephen Leppik
3	Blocs	BB9z, connor, danh, Fantini, insys, J F, Jeff Wolski, Johannes Fahrenkrug, Josh Caswell, Kote, Lito, Oliver Atkinson, Seán Labastille, tbodt, Yevhen Dubinin
4	BOOL / bool / Boolean / NSCFBoolean	Md. Ibrahim Hassan, user459460
5	Catégories	atroutt, DarkDust, Ekta Padaliya, Faran Ghani, Håvard, insys, Mykola Denysyuk, Orlando, Paulo Fierro, phi, WMios
6	Classes et Objets	Byron, DarkDust, HCarrasko, Jens Meder, Josh Caswell, NSNoob, ok404, RamenChef, tbodt
7	Continuer et casser!	Md. Ibrahim Hassan
8	Déclarer la méthode de classe et la méthode d'instance	Ruby, user459460
9	Enregistrement	Albert Renshaw, Chris Prince, connor, Daniel Bocksteger, DarkDust, DavidA, HariKrishnan.P, HCarrasko, Iulian Onofrei, Jason McDermott, Josh Caswell, Kornel, Nicolas Miari, NobodyNada, Peter N Lewis, Samet DEDE, Tapan Prakash, tbodt, Thomas Tempelmann, Tricertops, WMios
10	Entier aléatoire	Josh Caswell, jsondwyer
11	Énumération rapide	ff10, shuvo
12	Enums	Daniel Bocksteger, DavidA, Doc, Doron Yakovlev-Golani, lostInTransit, Mikhail Larionov, user459460
13	Environnement d'exécution de bas niveau	connor, DarkDust, dgatwood, Grady Player, Josh Caswell, orta, tbodt, Tricertops
14	Expédition Grand	user459460

	Central	
15	Gestion de la mémoire	James P, ok404, Tamarous, tbodt, Tricertops
16	Héritage	BKO
17	Indice	connor, tbodt
18	La gestion des erreurs	Doc, Sujania
19	Les méthodes	Caleb Kleveter, Jon Schneider, Joshua, jsondwyer, mrtnf, Sujania, Tapan Prakash, tbodt
20	Macros prédéfinies	user459460
21	Multi-Threading	DarkDust, jsondwyer
22	NSArray	animuson, AnthoPak, Bharath, DarkDust, Ekta Padaliya, Evan, HCarrasko, Irfan, j.f., James P, Jeff Wolski, Johannes Fahrenkrug, Jon Schneider, Joost, Josh Caswell, Joshua, jsondwyer, Losiowaty, mrtnf, mszaro, Muhammad Zohaib Ehsan, njuri, NSNoob, Paulo Fierro, Ruby, tbodt
23	NSAttributedString	Patrick, Sujania, user459460
24	NSCache	user459460
25	NSCalendar	byJeevan, connor
26	NSData	Patrick, Sujania, WMios
27	NSDate	jsondwyer, Nikolai Ruhe, Patrick, Sujania, WMios
28	NSDictionary	connor, Fantini, insys, Kevin Stewart, Mykola Denysyuk, Patrick, Sujania
29	NSJSONSerialization	connor, Sujania
30	NSMutableArray	aniket.ghode, animuson, DavidA, HCarrasko, J F, Joost, Ruby, Spidy, Sujania, tbodt, william205
31	NSMutableDictionary	Ravi Dhorajiya
32	NSObject	CodeChanger
33	NSPredicate	Dipen Panchasara
34	NSRegularExpression	Johannes Fahrenkrug

35	NSSortDescriptor	4444, Rahul
36	NSString	Albert Renshaw, animuson, AnthoPak, Cœur, DarkDust, Darshan Kunjadiya, David Mangon, il Malvagio Dottor Prosciutto, James P, Jeff Wolski, Johnny Rockex, Jon Schneider, Josh Caswell, Joshua, jsondwyer, Md. Ibrahim Hassan, Nikolai Ruhe, NSNoob, Orlando, Patrick, Paulo Fierro, RamenChef, Ruby, Srinivasan Saivenkat, Sunil Sharma, tbodt, Tricertops, ViratA, WMios
37	NSTextAttachment	BKO
38	NSTimer	Arc676, DarkDust, Hemang, Jason McDermott, Patrick, Ruby
39	NSURL	Patrick, Sujania
40	NSUrl envoie une demande de publication	Md. Ibrahim Hassan
41	NSUserDefaults	Adriana Carelli
42	Objectif moderne-C	pckill
43	Propriétés	DarkDust, dgatwood, Doc, J F, Nef10, NobodyNada, tbodt, Tricertops
44	Protocoles	Håvard, insys, jsondwyer, Mykola Denysyuk, Patrick, RamenChef, StrAbZ, tbodt, Tricertops
45	Protocoles et délégués	RamenChef, Sanjay Mohnani
46	Singletons	Amit Kalghatgi, Andrew Hoos, connor, Daniel Bocksteger, DarkDust, Glenn Smith, Itachi, pckill, Peter DeWeese
47	Spécificateurs de format	Albert Renshaw
48	Structs	Doc, Sujania
49	Test unitaire en utilisant Xcode	ajmccall, Amon, Siddharth Sunil
50	Types de données de base	connor, Josh Caswell, Muhammad Zohaib Ehsan, ok404, Sietse, sudo, Sujania, user459460
51	Valeur clé Codage / Valeur clé Observation	Cory Wilhite, insys, Jason McDermott, Nirav Bhatt, ThatsJustCheesy, WMios