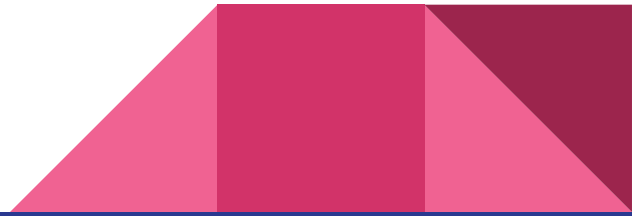
Introducción a Objective-C

Contenido

- Primeros pasos con Objective-c
- Breve especificación del lenguaje
- Peculiaridades del lenguaje
- **Programación orientada a objetos en Objective-C**
- Ejemplos



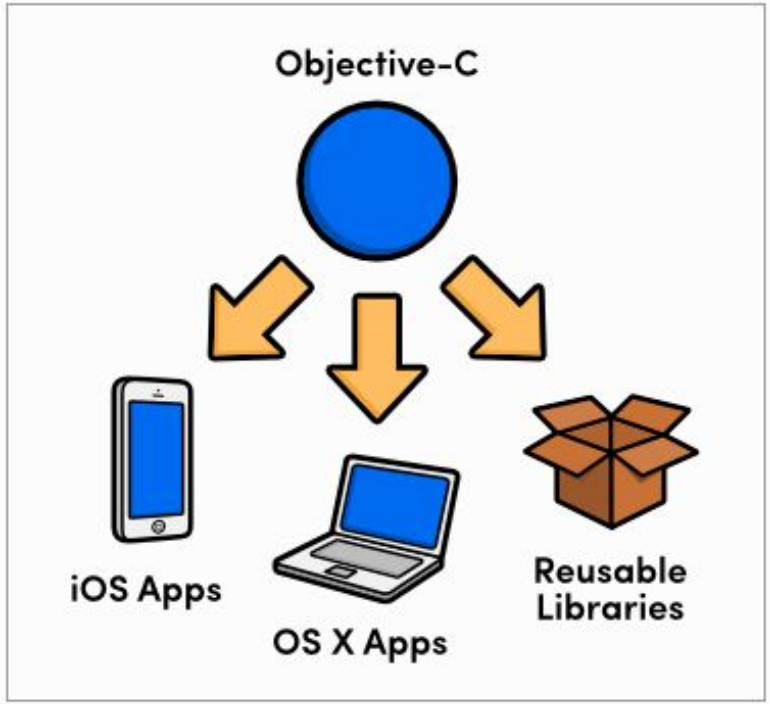
Primeros pasos

Objective-C

En esta sección veremos una introducción al lenguaje, un poco de historia y las respuestas a algunas preguntas como ¿Por qué nació? ¿Se usa hoy en día? y ¿Cómo será el futuro del lenguaje?
Adicionalmente, los ejemplos básicos y los primeros programas con Objective-C

Introducción

- Objective-C *es* el lenguaje de programación nativo para los sistemas operativos OS X y iOS de Apple.



¿Por qué existe Objective-C? Un poco de historia

Necesidad de un lenguaje de programación orientado a objetos **eficiente.**

- 1980’s
- Programación estructurada: Código espagueti.
- Smalltalk: Programación orientada a objetos y solución a muchos problemas.
- Smalltalk: Máquina virtual, bajo rendimiento.
- **Nace Objective-C**

¿Por qué existe Objective-C? Un poco de historia

Popularización a través de la compañía **NeXT** de Steve Jobs

- 1988: La compañía NeXT licencia Objective-C y desarrolló nuevas librerías.
- Objective-C fue bastante usado como herramienta de programación junto con las librerías creadas por NeXT.

¿Por qué existe Objective-C? Un poco de historia

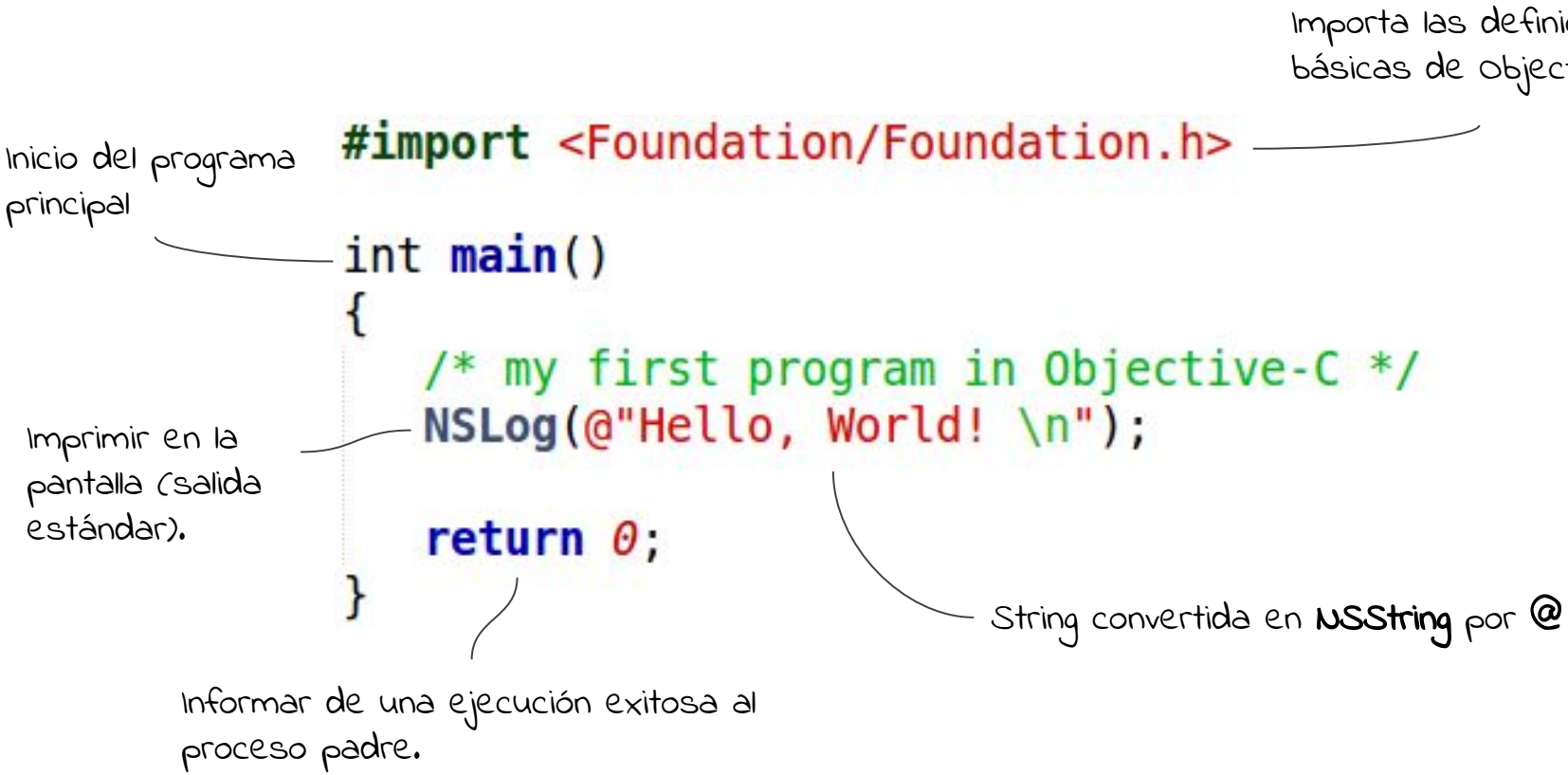
¿Se usa hoy en día?

- Desde el año 1996 y hasta 2014 fue el lenguaje de programación nativo para el desarrollo en iOS y OS X
- Actualmente es usado. Sin embargo, Apple está haciendo un proceso de transición con su nuevo lenguaje **Swift.**

Principales características de Objective-C

- **Orientado a objetos**
- Compilado
- Basado en C y Smalltalk
  - Todo código C es compilable en Objective-C
  - Se pueden usar las librerías de C dentro de Objective-C

¡Hola Mundo!


Importa las definiciones básicas de Objective-C.
Inicio del programa principal
#import <Foundation/Foundation.h>
int main()
{
/* my first program in Objective-C */
NSLog(@"Hello, World! \n");
return 0;
}
Imprimir en la pantalla (salida estándar).
String convertida en NSString por @
Informar de una ejecución exitosa al proceso padre.

Breve especificación del lenguaje

Objective-C

En esta sección vamos a echar un vistazo a cómo funciona Objective C en cuanto a las librerías que ofrece y el manejo de la memoria. Además ¿Qué significa NS?

Frameworks

- Foundation

Define los tipos de datos orientados a objetos básicos como strings, arreglos, diccionarios, etc.
NSString
NSNumber
NSDictionary
NSArray

Frameworks

- UIKit
- AppKit
- CoreData
- MediaPlayer
- AVFoundation
- QuartzCore
- CoreGraphics

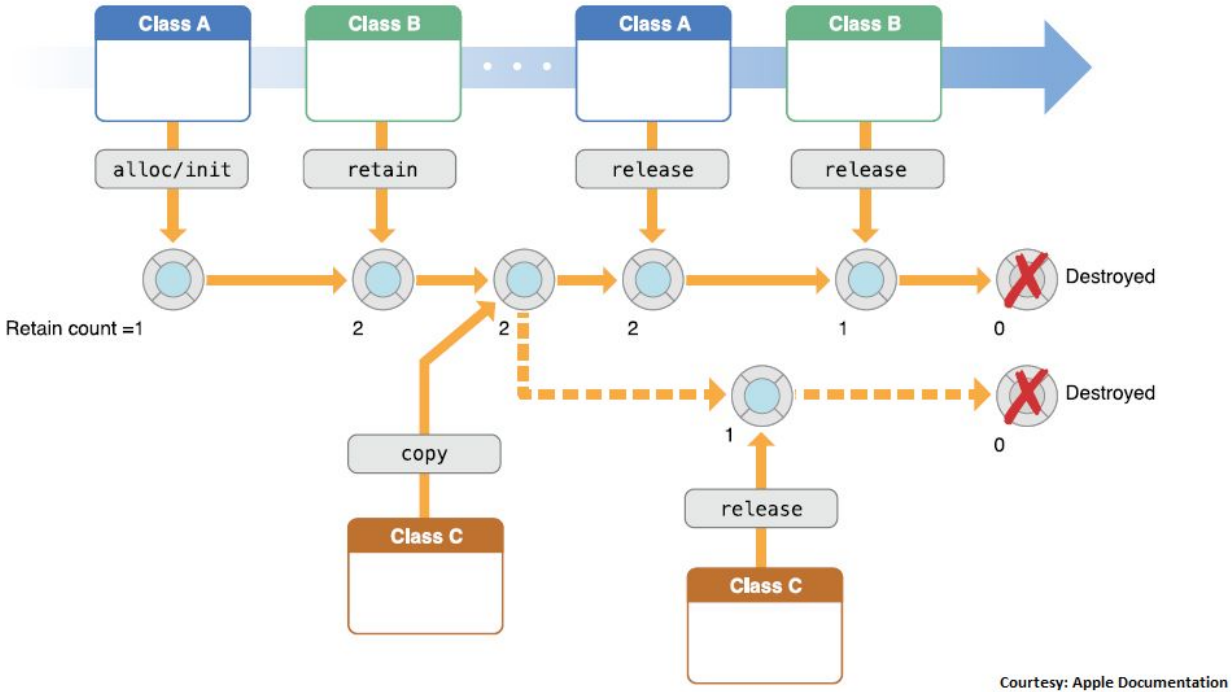
¿Qué significa NS?

Se utiliza como prefijo de la mayoría de las clases del SDK de IOS y MAC, en “honor” al sistema operativo que utilizaban las computadoras NeXT, ***NeXTStep***.

Manejo de memoria

1. Manual Retain-Release (MRR)
2. Automatic reference counting(ARC)

Manual Retain-Release



Courtesy: Apple Documentation

Manual Retain-Release

```objc
@implementation SampleClass
...
- (void)dealloc {
  NSLog(@"Object deallocated");
  [super dealloc];
}
@end

int main() {
   SampleClass *sampleClass = [[SampleClass alloc]init];
   [sampleClass sampleMethod];
   NSLog(@"Retain Count after initial allocation: %d",
   [sampleClass retainCount]);
   [sampleClass retain];
   NSLog(@"Retain Count after retain: %d", [sampleClass retainCount]);
   [sampleClass release];
   NSLog(@"Retain Count after release: %d", [sampleClass retainCount]);
   [sampleClass release];
   NSLog(@"SampleClass dealloc will be called before this");
   sampleClass = nil;
   return 0;
}
```

Libera el espacio de memoria usado por el objeto

Creamos un objeto de nuestra propiedad alloc <-> new

Consulta el valor del contador de referencias del objeto

Aumenta el contador de referencias

Decrementa el contador de referencias, cuando se vuelve cero, llama **dealloc**

Manual Retain-Release

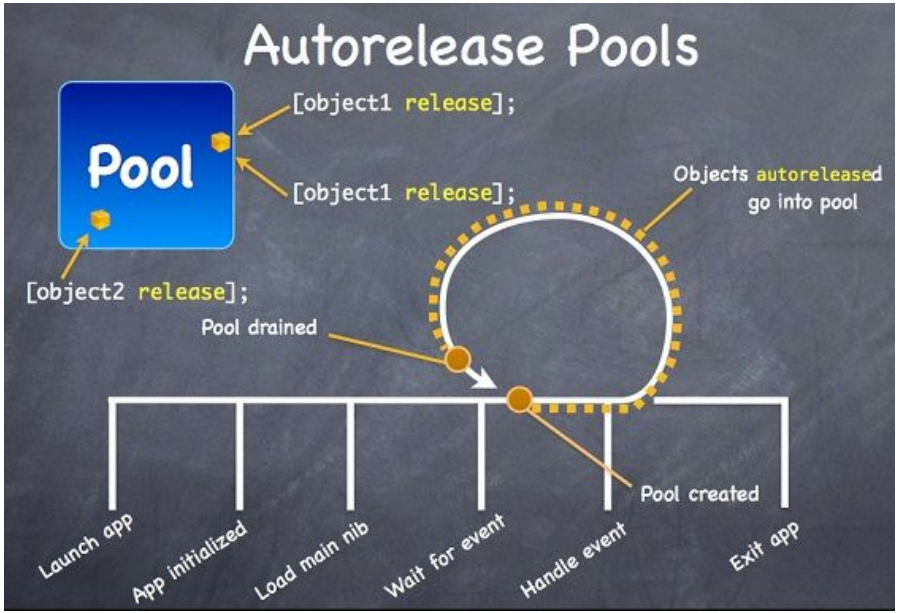
```
NSString *cad = [NSString stringWithString:@"Hola"];
[cad release]
```

Creamos un objeto que **NO** es de nuestra propiedad

Generaría error, pues el objeto no es de nuestra propiedad

Automatic Reference Counting (ARC)

Es el manipulador de memoria que nos evita hacer uso de ***release.***



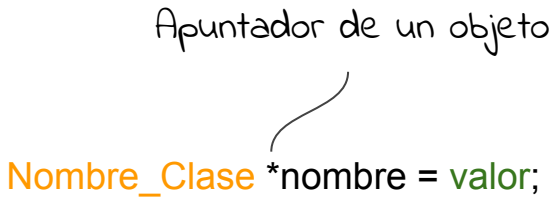
Peculiaridades del lenguaje

Objective-C

Si objective C es un lenguaje orientado a objetos, ¿como se deben manejar los datos primitivos? ¿Y qué pasa con los contenedores de datos (arreglos, estructuras, etc.)? ¿Cómo se manejan las funciones?

Apuntadores para objetos

Cuando se quiere crear un objeto se requiere hacerlo mediante un puntero, de la siguiente manera :



¿Por qué no trabajar con el valor directamente, en vez del apuntador?

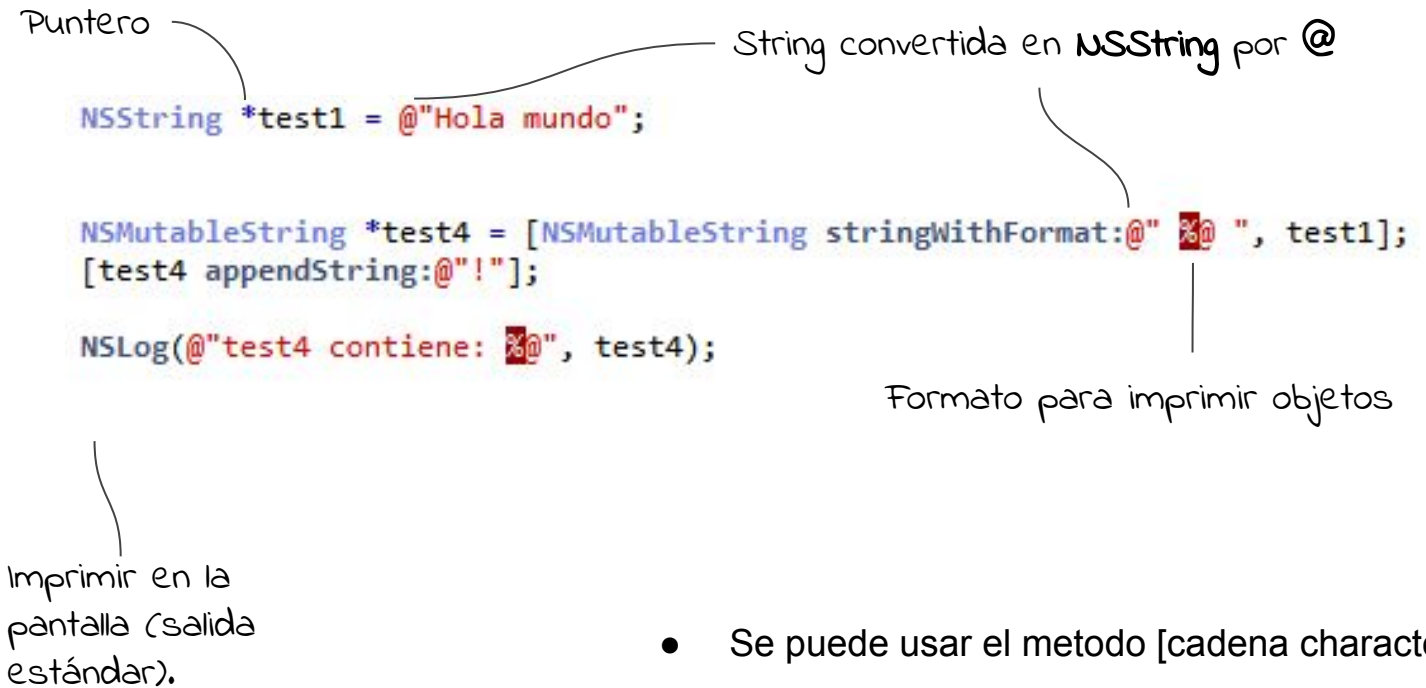
Funciones

```
- (return_type) method_name:( argumentType1 )argumentName1
  joiningArgument2:( argumentType2 )argumentName2 ... joiningArgumentn:( argumentTypen )argumentNamen {
   body of the function
}
```

```
-( id ) inicializarConPrecio: (  NSNumber* ) precioInicial yPeso: ( NSNumber* ) pesoInicial {
    precioBase = precioInicial;
    color = @"blanco";
    consumoEnergetico = [ NSNumber numberWithChar: 'F' ];
    peso = pesoInicial;
    return self;
}
```

Clases Envolventes

NSString Y NSMutableString



- Se puede usar el metodo [cadena characterAtIndex: pos]

Clases Envolventes

NSNumber

```
NSNumber *A = [NSNumber numberWithBool:NO];
NSNumber *B = [NSNumber numberWithChar:'z'];
NSNumber *C = [NSNumber numberWithUnsignedChar:255];
NSNumber *D = [NSNumber numberWithShort:32767];
NSNumber *E = [NSNumber numberWithUnsignedShort:65535];
NSNumber *F = [NSNumber numberWithInt:2147483647];
NSNumber *G = [NSNumber numberWithUnsignedInt:4294967295];
NSNumber *H = [NSNumber numberWithLong:9223372036854775807];
NSNumber *I = [NSNumber numberWithUnsignedLong:18446744073709551615];
NSNumber *J = [NSNumber numberWithFloat:26.99f];
NSNumber *K = [NSNumber numberWithDouble:26.99];
```

```
NSNumber *A = @NO;
NSNumber *B = @'z';
NSNumber *C = @2147483647;
NSNumber *D = @4294967295U;
NSNumber *E = @9223372036854775807L;
NSNumber *F = @26.99F;
NSNumber *G = @26.99;
double x = 24.0;
NSNumber *result = @(x * .15);
NSLog(@"%.2f", [result doubleValue]);
```

¿Redundancia? para qué definir esto si ya existen los datos primitivos

Clases Envolventes

NSNumber

```
NSNumber *comoInt = [NSNumber numberWithInt:42];
float comoFloat = [comoInt floatValue];
NSLog(@"%.2f", comoFloat);

NSString *comoString = [comoInt stringValue];
NSLog(@"%@", comoString);
```

No olvidar los punteros

Castear usando el método value

Un casteo fácil usando los métodos de NSNumber

Clases Envolventes

NSNumber

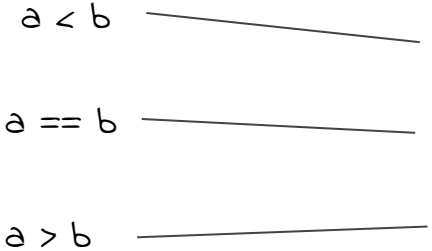
Comparación por referencias

Comparación por valor

```
NSNumber *anInt = @27;
NSNumber *sameInt = @27U;
// Comparacion de direcciones
if (anInt == sameInt) {
    NSLog(@"Son los mismos objetos");
}
// Comparacion de valores
if ([anInt isEqualToNumber:sameInt]) {
    NSLog(@"Tienen el mismo valor");
}
```

Clases Envolventes

NSNumber




```
NSNumber *anInt = @27;
NSNumber *anotherInt = @42;
NSComparisonResult result = [anInt compare:anotherInt];
if (result == NSOrderedAscending) {
NSLog(@"27 < 42");
} else if (result == NSOrderedSame) {
NSLog(@"27 == 42");
} else if (result == NSOrderedDescending) {
NSLog(@"27 > 42");
}
```

Colecciones

NSArray y NSMutableArray

```objc
#import <Foundation/Foundation.h>
int main() {
    NSAutoreleasePool *pool = [NSAutoreleasePool new];
    NSArray *arr = [NSArray arrayWithObjects: @"Colombia", @"1", [NSNumber numberWithInt:22], @"Lenguajes", nil];
    NSMutableArray *myArray = [NSMutableArray new];
    int i, size = [arr count];
    for(i = 0; i < size; i++) {
        NSLog(@"posicion %d = %@ ", i, [arr objectAtIndex: i]);
        [myArray addObject: [arr objectAtIndex: i]];
    }

    [myArray removeObject: @"1"];
    [myArray removeObjectAtIndex: 0];
    size = [myArray count];
    NSLog(@"\n");
    for(i = 0; i < size; i++) {
        NSLog(@"posicion %d = %@ ", i, [myArray objectAtIndex: i]);
    }
    if ([arr isEqualToArray:myArray]) {
        NSLog(@"Wow, ambos arreglos son identicos");
    }
    [pool drain];
return 0;
}
```

Poner null al final de una declaración explícita

Obtener el tamaño de un arreglo

Agregar elementos a un arreglo

Borrar elementos de un arreglo

Acceder a una posición

Comparar valores de dos arreglos

Colecciones

NSSet y NSMutableSet

```objc
NSSet *americanMakes = [NSSet setWithObjects:@"Chrysler", @"Ford", @"General Motors", nil];
NSArray *japaneseMakes = @[@"Honda", @"Mazda", @"Mitsubishi", @"Honda"];
// Toma los valores repetidos y los deja 1 sola vez
NSSet *uniqueMakes = [NSSet setWithArray:japaneseMakes];


NSSet *models = [NSSet setWithObjects:@"Civic", @"Accord", @"Odyssey", @"Pilot", @"Fit", nil];
NSLog(@"El set tiene %li elements", [models count]);
for (id item in models) {
    NSLog(@"%@", item);
}
```

id es un tipo de dato genérico

No olvidar poner null al final de la definición explícita

Colecciones

NSSet y NSMutableSet

```
NSSet *japaneseMakes = [NSSet setWithObjects:@"Honda", @"Nissan", @"Mitsubishi", @"Toyota", nil];
NSSet *johnsFavoriteMakes = [NSSet setWithObjects:@"Honda", nil];
NSSet *marysFavoriteMakes = [NSSet setWithObjects:@"Toyota", @"Alfa Romeo", nil];
if ([johnsFavoriteMakes isEqualToSet:japaneseMakes]) {
    // Se compara si el contenido de John es igual al de los japoneses
    NSLog(@"John comparte todo lo de los japoneses");
}
if ([johnsFavoriteMakes intersectsSet:japaneseMakes]) {
    // Existe un elemento en com´un entre los japoneses y John
    NSLog(@"John tiene algo en com´un con los japoneses");
}
if ([johnsFavoriteMakes isSubsetOfSet:japaneseMakes]) {
    // Se mira si John es un subconjunto de los japoneses
    NSLog(@"john est´a incluido en los japoneses");
}
if ([marysFavoriteMakes isSubsetOfSet:japaneseMakes]) {
    // Se mira si Mary es un subconjunto de los japoneses
    NSLog(@"Mary est´a incluida en los japoneses");
}
```

Colecciones

NSSet y NSMutableSet

```
NSSet *selectedMakes = [NSSet setWithObjects:@"Maserati", @"Porsche", nil];
if ([selectedMakes containsObject:@"Maserati"]) {
    NSLog(@"Maserati esta en selectedMakes");
}
NSString *result = [selectedMakes member:@"Maserati"];
if (result != nil) {
    NSLog(@"%@ es uno de los SelectedMakes", result);
}
```

Arroja un valor booleano

Arroja el apuntador

Colecciones

NSDictionary y NSMutableDictionary

```
// Valores y claves como argumentos
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];
// Valores y claves como arreglos
NSArray *models = @[@"Mercedes-Benz SLK250", @"Mercedes-Benz E350",
@"BMW M3 Coupe", @"BMW X6"];
NSArray *stock = @[[NSNumber numberWithInt:13],
[NSNumber numberWithInt:22],
[NSNumber numberWithInt:19],
[NSNumber numberWithInt:16]];
inventory = [NSDictionary dictionaryWithObjects:stock forKeys:models];
```

No olvide el nil al final

Colecciones

NSDictionary y NSMutableDictionary

```
// Modifica un valor ya existente
[diccionario setObject:@15 forKey:@"Audi TT"];
// Borra un elemento del diccionario
[diccionario removeObjectForKey:@"Audi A7"];
// Agregar una nueva llave y un valor al diccionario
diccionario[@"Audi R8 GT"] = @17;
```

```
for (id key in inventory) {
    NSLog(@"There are %@ %@'s in stock", [inventory objectForKey:key], key);
}
```

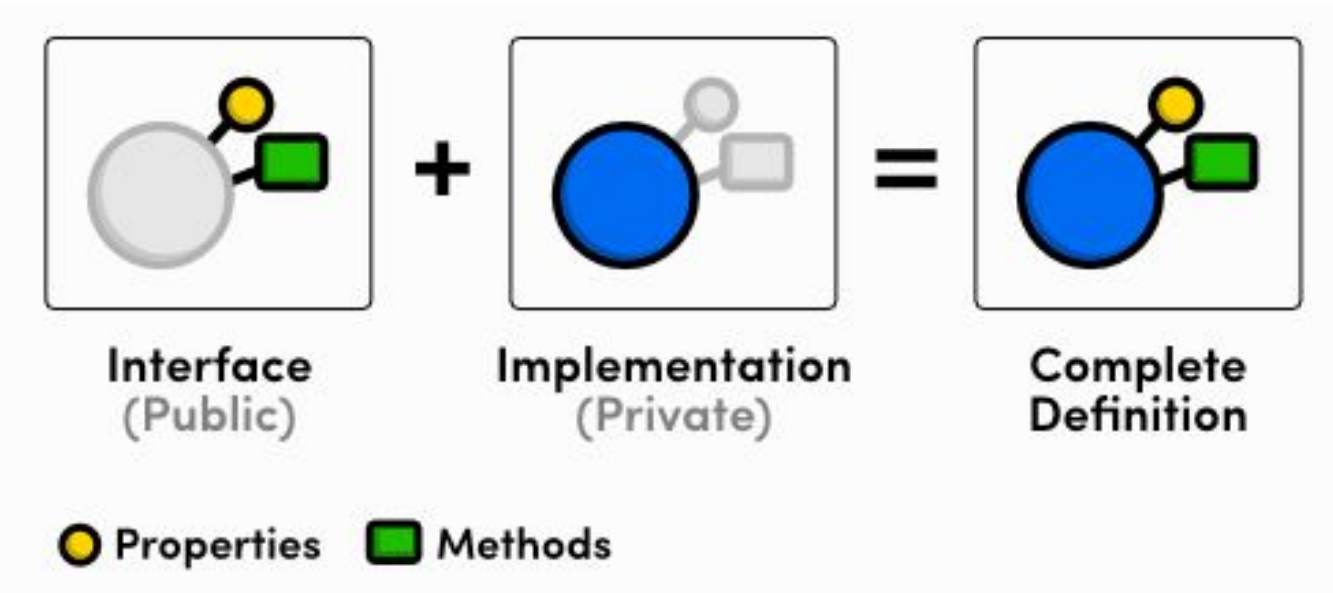
```
if ([diccionario1 isEqualToDictionary:diccionario2]) {
    NSLog(@"Ambos diccionarios son iguales");
}
```

Programación orientada a objetos

Objective-C

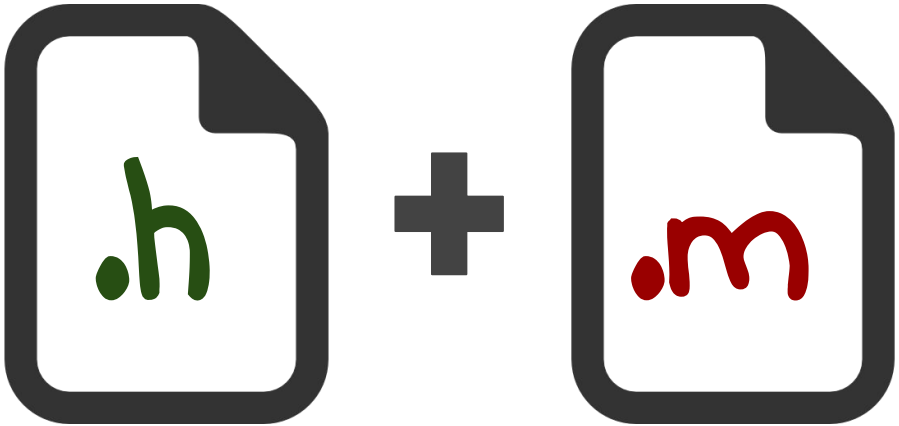
En esta sección veremos la manera de hacer programación orientada a objetos en Objective-C.
Aprenderemos a definir clases, protocolos, crear y caracterizar los objetos y entender tanto la sintaxis como la semántica de esta parte del lenguaje

Entendiendo las clases de Objective-C.



Importante: La nomenclatura de JAVA no significa lo mismo en objective-C

Entendiendo las clases de Objective-C.

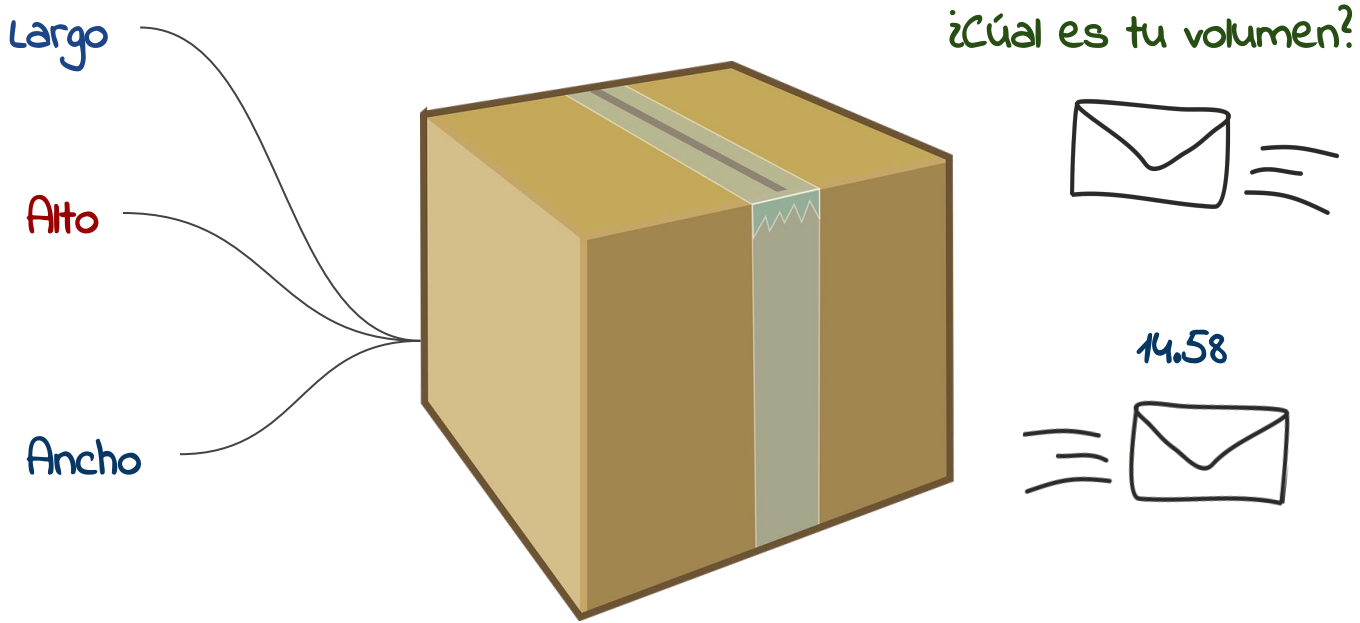


Las interfaces se escriben y almacenan en ficheros con extensión **.h**

Las implementaciones se definen en ficheros **.m**

Estos dos ficheros en conjunto forman una definición de una **clase** objective-C

Entendiendo las clases de Objective-C.



Interfaz (Box.h)

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

@interface Box:NSObject
{
    double length;   // Length of a box
    double breadth;  // Breadth of a box
    double height;   // Height of a box
}
@property(nonatomic, readwrite) double height; // Property

-(double) volume;

@end
```

Importa las definiciones básicas de Objective-C.

Definición de una interfaz con clase base NSObject.

Variables de instancia. protegidas por defecto

Toma la variable de instancia (ivar) y la convierte en propiedad con opciones de lectura y escritura (getters/setters).

Método de instancia

Implementación (Box.m)

```
# import "Box.h"

@implementation Box

@synthesize height;

-(id)init
{
    self = [super init];
    length = 1.0;
    breadth = 1.0;
    return self;
}

-(double) volume
{
    return length*breadth*height;
}

@end
```

Importa la definición de la interfaz de nuestra caja

Comienza la definición de la implementación

Sintetiza (crea) los getters y/o setters de las **propiedades** especificadas y acuerdo con la definición de la interfaz.

Constructor: Este método crea un objeto, inicializa sus valores y retorna la dirección de memoria en donde este fue creado.

Es la definición **concreta** (implementación) del método volumen declarado en la interfaz

Prueba (BoxMain2.m)

```
# import "Box.h"

int main(){
    NSAutoreleasePool * pool;
    pool = [[NSAutoreleasePool alloc] init];

    // Create box1 object of type Box
    Box *box1 = [[Box alloc]init];

    // Store the volume of a box here
    double volume = 0.0;

    // box 1 specification
    box1.height = 5.0;

    // volume of box 1
    volume = [box1 volume];
    NSLog(@"Volume of Box1 : %f", volume);

    [pool drain];
    return 0;
}
```

objeto que ayuda a el manejo de memoria

Instanciando un objeto de la clase Box

Acceso a la **propiedad** height a través de un setter

Mensaje al objeto caja requiriendo el volumen de ella.

Salida estándar con formato

Liberar la memoria

Compilación

Compilador GCC

Es necesario compilar la implementación y el archivo de prueba en el **mismo comando** si se intenta compilar BoxMain2.m por separado se obtendrá un error.

Nombre del ejecutable

```
gcc $(gnustep-config --objc-flags) Box.m BoxMain2.m $(gnustep-config --base-libs) -o BoxMain2
```

flags (opciones de configuración) para que gcc reconozca nuestro código como de Objective-C

librerías para el proceso de linkado

Nota: El comando de compilación puede diferir si te encuentras en el sistema operativo windows. Para más detalle sobre esto dirígete al tutorial escrito.

Ejecución

Con este comando ejecutaremos el programa

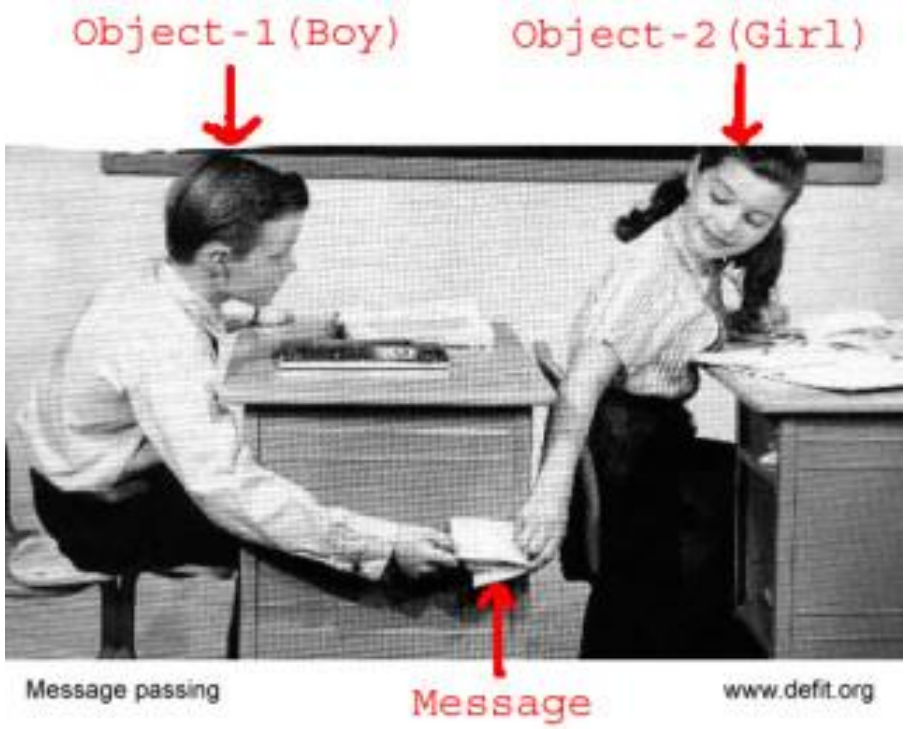
```
./BoxMain2
```

```
2016-05-14 11:24:51.230 BoxMain2[8558] Volume of Box1 : 5.000000
```

Resultado del programa

Paso de mensajes

Estrictamente hablando, Objective-C **no posee métodos** que se invocan a sobre los objetos **directamente** sino que el sistema está totalmente basado en paso de mensajes



Paso de mensajes vs Invocación de métodos

Paso de mensajes (Objective-C, Smalltalk)

- El método puede o no **existir** en la definición de la clase pero esto no causa que el programa deje de funcionar
- Existe una mayor **flexibilidad** en el diseño
- Los objetos pueden o no responder al mensaje o **redirigir** el mensaje a otro objeto
- Existe una mayor posibilidad de **errores** en el código

Invocación de métodos (Java, C++)

- La firma del método debe ser definido **explícitamente** en la clase pero este puede o no cargarse dinámicamente y cambiar en tiempo de ejecución.
- Es más **eficiente** invocar un método que enviar un mensaje.

Encapsulamiento

Características:

- Protección de la información.
- Uso de ***@property*** para mayor facilidad
- Especificación explícita de los modificadores de acceso para cada una de las variables: ***@public***, ***@private***.

Encapsulamiento

```
@interface Adder : NSObject{
    NSInteger total;
}

- (id)initWithInitialNumber:(NSInteger)initialNumber;
- (void)addNumber:(NSInteger)newNumber;
- (NSInteger)getTotal;

@end
```

Declaración de la variable total por defecto protected

Sólo podremos modificar el valor de total mediante este mensaje

Sólo podremos acceder al valor de total mediante este mensaje

Encapsulamiento

Creación del objeto, e inicialización con el valor 10

Modificación de la variable sin tener acceso directo a la variable

```
@implementation Adder
-(id)initWithInitialNumber:(NSInteger)initialNumber{
    total = initialNumber;
    return self;
}
- (void)addNumber:(NSInteger)newNumber{
    total = total + newNumber;
}
- (NSInteger)getTotal{
    return total;
}
@end

int main(int argc, const char * argv[]){
    NSAutoreleasePool * pool = [[NSAutoreleasePool alloc] init];
    Adder *adder = [[Adder alloc]initWithInitialNumber:10];
    [adder addNumber:5];
    [adder addNumber:4];
    NSLog(@"The total is %ld",[adder getTotal]);
    [pool drain];
    return 0;
}
```

Herencia

- Reusar código.
- Rápida codificación.
- Código más entendible.

```
@interface derived-class: base-class
```

Herencia

```
@interface Person : NSObject{
    NSString *personName;
    NSInteger personAge;
}
- (id)initWithName:(NSString *)name andAge:(NSInteger)age;
- (void)print;
@end

@implementation Person
- (id)initWithName:(NSString *)name andAge:(NSInteger)age{
    personName = name;
    personAge = age;
    return self;
}
- (void)print{
    NSLog(@"Name: %@", personName);
    NSLog(@"Age: %ld", personAge);
}
@end
```

Herencia

```
@interface Employee : Person{
    NSString *employeeEducation;
}
- (id)initWithName:(NSString *)name andAge:(NSInteger)age
  andEducation:(NSString *)education;
- (void)print;
@end


@implementation Employee
- (id)initWithName:(NSString *)name andAge:(NSInteger)age andEducation: (NSString *)education{
    [ super initWithName: name andAge: age ]
    employeeEducation = education;
    return self;
}
- (void)print{
    [ super print ]
    NSLog(@"Education: %@", employeeEducation);
}

@end
```

Polimorfismo

Sobrecarga y Polimorfismo

Sobrecarga

- Mismo nombre
- Diferentes parámetros
- Diferentes tipos
- Misma clase
- Tiempo de compilación

Polimorfismo

- Mismo nombre
- Mismos parámetros
- Mismos tipos
- Diferente clase (Herencia)
- Tiempo de ejecución

Polimorfismo

Sobrecarga y Polimorfismo

Sobrecarga

```
public class A {
      public void Metodo() {
         Metodo(0);
      }

      public void Medodo(int Valor) {
          //Aqui se escribe todo el código.
      }
}
```

Polimorfismo

```
public abstract class Figura {
    public abstract void Dibujar();
}
public class Triangulo : Figura {
    public override void Dibujar()      {
        //Aqui dibujar un Triangulo
    }
}
public class Cuadrado : Figura {
    public override void Dibujar() {
        //Aqui dibujar un Cuadrado
    }
}
```

Polimorfismo

Método al cual aplicaremos polimorfismo

```
// .h
@interface Shape : NSObject {
    float area;
}

- (void)printArea;
- (void)calculateArea;
@end
//.m
@implementation Shape

- (void)printArea {
    NSLog(@"The area is %f", area);
}

- (void)calculateArea{}

@end
```

Polimorfismo

```objc
@interface Square : Shape {
    float length;
}

- (id)initWithSide:(float)side;
- (void)calculateArea;

@end

@implementation Square

- (id)initWithSide:(float)side {
    length = side;
    return self;
}

- (void)calculateArea {
    area = length * length;
}

- (void)printArea {
    NSLog(@"The area of square is %f", area);
}

@end
```

Métodos a los que aplicamos polimorfismo

```objc
@interface Rectangle : Shape {
    float length;
    float breadth;
}

- (id)initWithLength:(float)rLength andBreadth:(float)rBreadth;

@end

@implementation Rectangle

- (id)initWithLength:(float)rLength andBreadth:(float)rBreadth {
    length = rLength;
    breadth = rBreadth;
    return self;
}

- (void)calculateArea {
    area = length * breadth;
}

@end
```

Polimorfismo

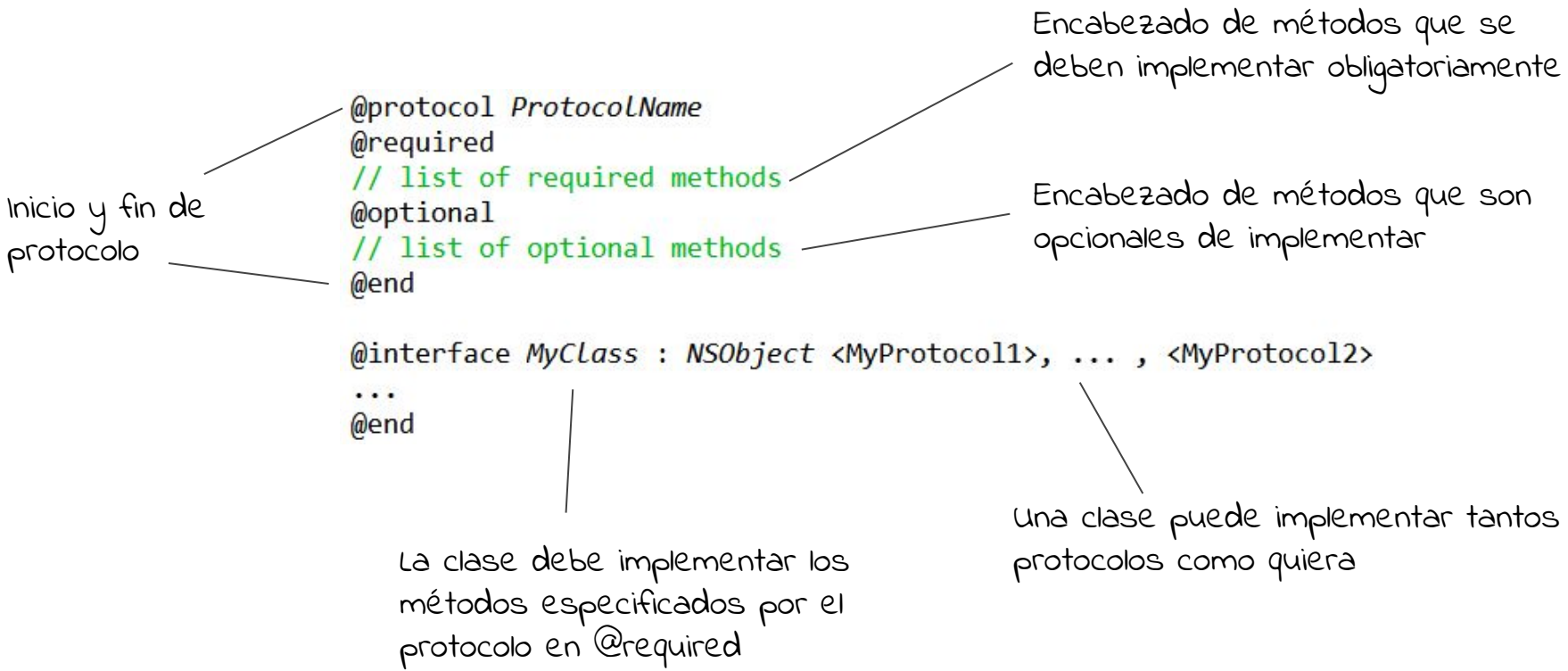
```
int main() {
    NSAutoreleasePool * pool = [[NSAutoreleasePool alloc] init];

    Shape *square = [[Square alloc]initWithSide:10.0];
    [square calculateArea];
    [square printArea];
    Shape *rect = [[Rectangle alloc] initWithLength:10.0 andBreadth:5.0];
    [rect calculateArea];
    [rect printArea];

    [pool drain];
    return 0;
}
```

Métodos con polimorfismo

Protocolos


@protocol ProtocolName
@required
// list of required methods
@optional
// list of optional methods
@end
@interface MyClass : NSObject <MyProtocol1>, ... , <MyProtocol2>
...
@end
Encabezado de métodos que se deben implementar obligatoriamente
Encabezado de métodos que son opcionales de implementar
Inicio y fin de protocolo
La clase debe implementar los métodos especificados por el protocolo en @required
Una clase puede implementar tantos protocolos como quiera

Protocolos

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

@protocol PrintProtocolDelegate
@required
- (void)processCompleted;
@optional
- (void)printError;
@end

@interface SampleClass:NSObject <PrintProtocolDelegate>

- (void)startAction;

@end

@implementation SampleClass

- (void)startAction{
    NSLog(@"Start action!!");
    [self processCompleted];
}

-(void)processCompleted{
    NSLog(@"Printing Process Completed");
}

@end
```

No es necesario implementar este método, es opcional

```objc
int main()
{
    NSAutoreleasePool * pool = [[NSAutoreleasePool alloc] init];
    SampleClass *sampleClass = [[SampleClass alloc]init];
    [sampleClass startAction];
    [pool drain];
    return 0;
}
```

Categorías

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

@interface NSString(MyAdditions)

+(NSString *)getCopyRightString;

@end

@implementation NSString(MyAdditions)

+(NSString *)getCopyRightString{
    return @"Copyright TutorialsPoint.com 2013";
}

@end


int main()
{

   NSAutoreleasePool * pool = [[NSAutoreleasePool alloc] init];
   NSString *copyrightString = [NSString getCopyRightString];
   NSLog(@"Accessing Category: %@",copyrightString);
   [pool drain];
    return 0;
}
```

Agregar una categoría a NSString

Agrega un nuevo comportamiento a la clase NSString

Ejemplos

Objective-C

En esta sección podremos ver el potencial de Objective C y podremos dar respuesta a la pregunta: ¿Realmente vale la pena aprender este lenguaje?

Descripción

Clase película

Clase juego

¿Está prestado?

Título

Género

Tiempo estimado de juego

Compañía desarrolladora

Mensajes que deben responder

Préstate

Devuélvete

¿Está prestado?

Conclusiones

Objective-C

- El lenguaje, como cualquier otro, está lejos de ser perfecto, y ciertamente tiene algunas características que lo hacen parecer extraño, sobre todo desde el punto de vista sintáctico.
- La ventaja más evidente es que los programas Objective-C pueden hacer uso de infinidad de librerías escritas en C, como por ejemplo sqlite y OpenGL, dos ejemplos de uso notable en aplicaciones para iOS especialmente.
- De no ser porque Apple está en transición hacia Swift, aprender Objective-C sería muy recomendable pero hoy en día es más recomendable empezar con Swift como una evolución de Objective-C

Referencias

[1] http://rypress.com/tutorials/objective-c/index Tutorial Objective-C

[2] https://es.wikipedia.org/wiki/Objective-C Objective-C

[3] https://es.wikipedia.org/wiki/Swift_%28lenguaje_de_programaci%C3%B3n%29 Swift

[4] Anotaciones en el capsulamiento de Objetive-c http://stackoverflow.com/questions/2255861/property-and-retain-assign-copy-nonatomic-in-objective-c