

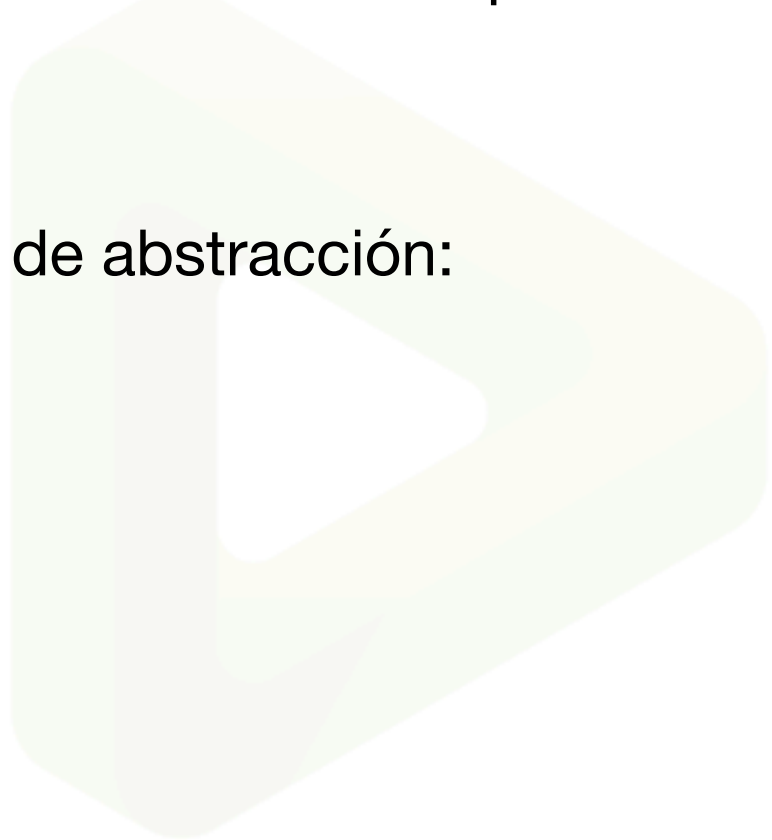
# Tema 1

## Introducción a iOS SDK y Objective-C

# Introducción a iOS



# Introducción a iOS

- Es el sistema operativo utilizado en los dispositivos iPhone, iPad y iPod Touch.
  - Dispone de cuatro capas de abstracción:
    - Core OS
    - Core Services
    - Media
    - Cocoa Touch
- 

# Introducción a iOS

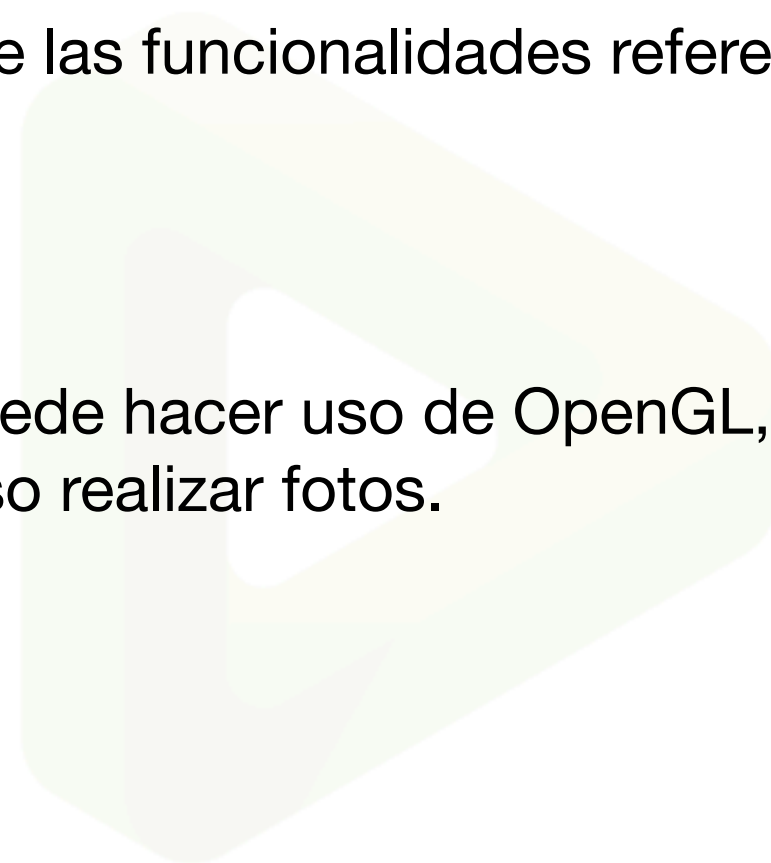
- La capa Core OS es la base del sistema operativo.
- Contiene las características de bajo nivel sobre las que se construye el resto.
- Entre sus funciones destaca la gestión de memoria, la gestión de redes y la comunicación con elementos hardware.

# Introducción a iOS

- La capa Core Services contiene la mayoría de servicios fundamentales que las aplicaciones utilizan.
- Sus características principales son el acceso a la agenda, a las cuentas del dispositivo y la gestión de bases de datos.

# Introducción a iOS

- La capa Media comprende las funcionalidades referentes al audio, vídeo y gráficos.
- Mediante esta capa se puede hacer uso de OpenGL, reproducción de audio y vídeo, animaciones e incluso realizar fotos.



# Introducción a iOS

- La capa Cocoa Touch es la capa de más alto nivel.
- Comprende funciones como el acelerómetro, las notificaciones, los controles táctiles o los mapas.

# Novedades en iOS 6



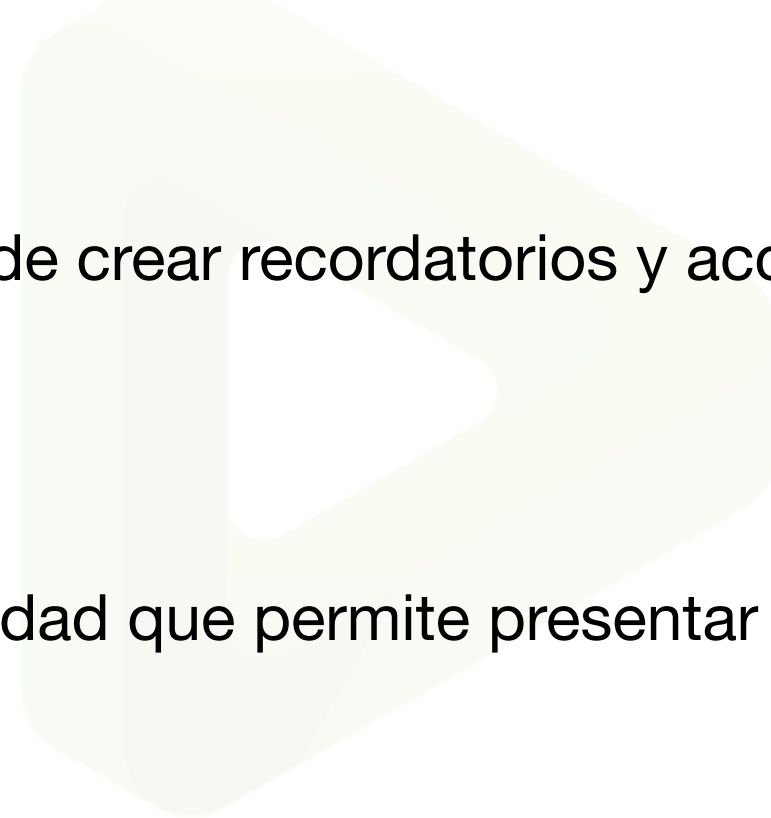


# Novedades en iOS 6

- Mapas: nuevo mapa creado por Apple, con la posibilidad de crear apps que permitan crear rutas y guiar al usuario.
- Social Framework: integración con redes sociales como Twitter y Facebook.
- Pass Kit: tecnología para la integración con Passbook.

# Novedades en iOS 6

- Game Center: actualización con nuevas características.
- Recordatorios: posibilidad de crear recordatorios y acceder a los recordatorios del usuario.
- Collection Views: funcionalidad que permite presentar la información en forma de cuadrícula.

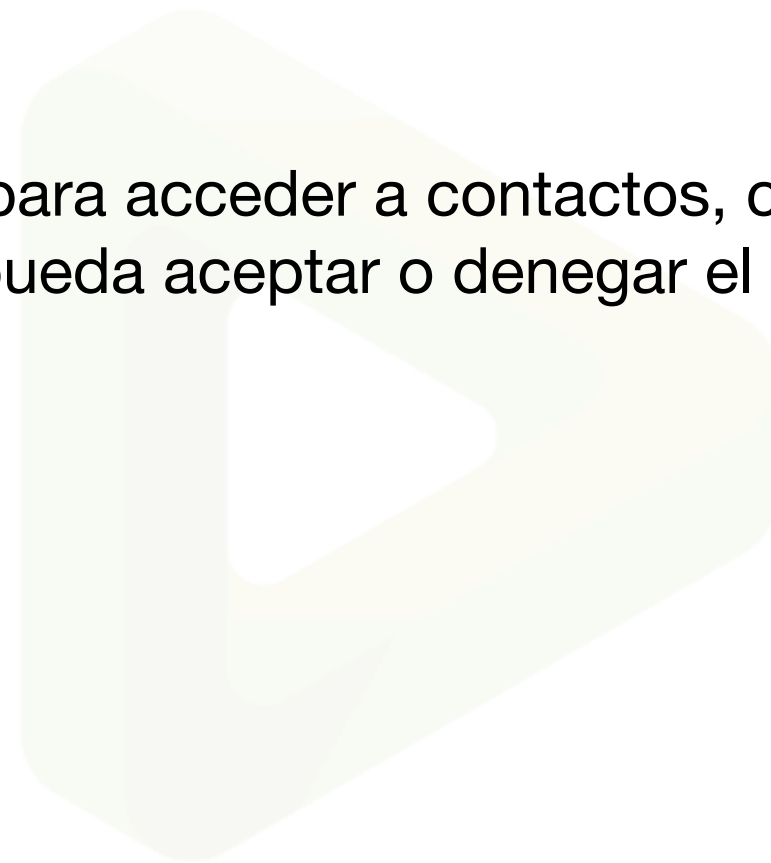


# Novedades en iOS 6

- Preservación del estado: posibilidad para restaurar el estado de la aplicación en el que se encontraba la última vez que se usó en caso de cerrarse.
- Auto layout: conjunto de reglas utilizadas para la creación de interfaces.

# Novedades en iOS 6

- Privacidad: posibilidad para acceder a contactos, calendario y recordatorios de forma que el usuario pueda aceptar o denegar el acceso a dicha información.



# Desarrollo en iOS

# Desarrollo en iOS

- Herramientas:
  - Xcode: es el entorno de desarrollo integrado utilizado para el desarrollo de aplicaciones.
  - Simulador: herramienta capaz de simular un iPhone o iPad y que permite probar nuestras aplicaciones sin el uso de un dispositivo real.
  - Instruments: herramienta de análisis capaz de monitorizar la aplicación

# Desarrollo en iOS

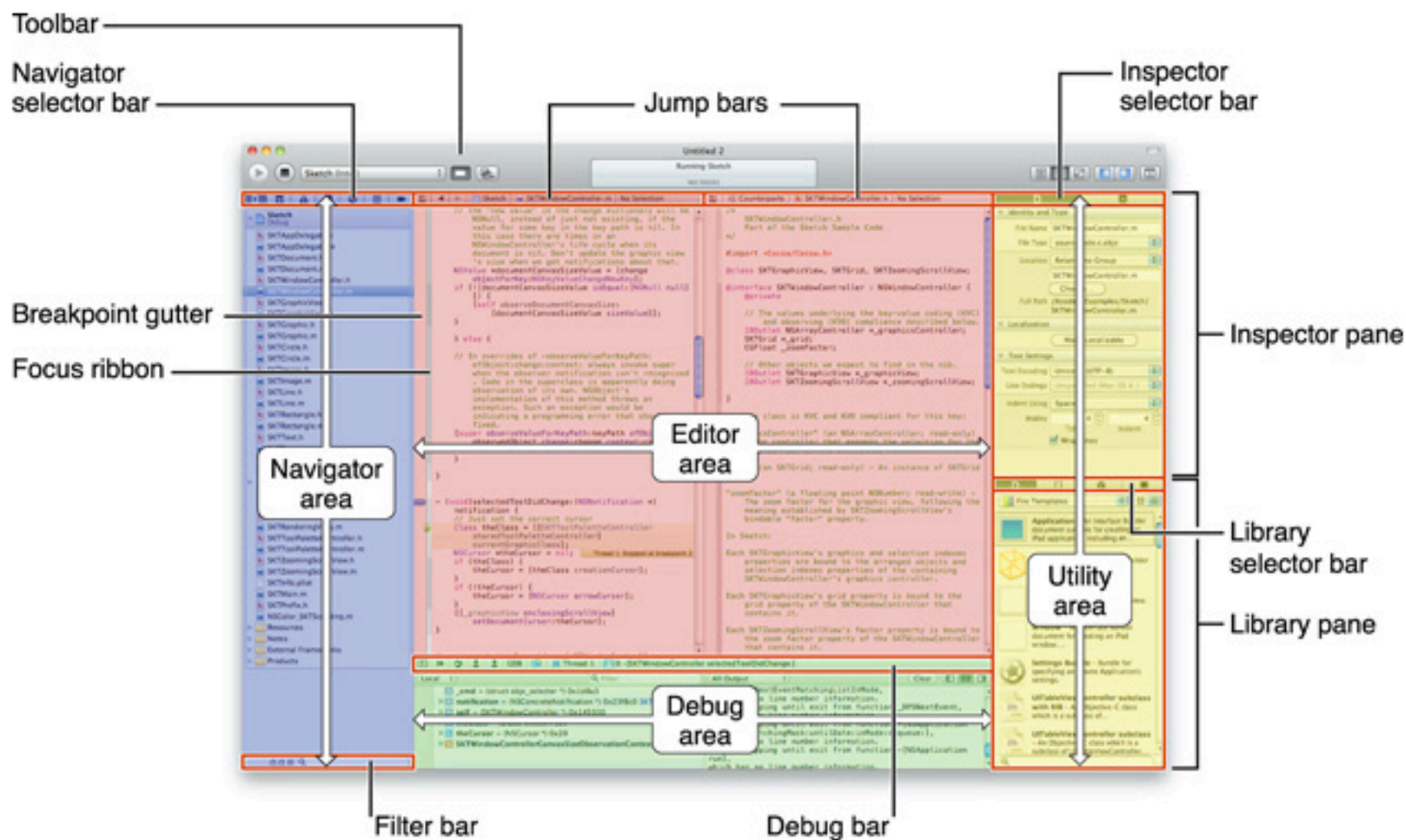
- Objective-C: es el lenguaje de programación empleado.
- Frameworks: Foundation, UIKit y resto de frameworks en iOS.
- Patrones de diseño: Principalmente MVC, delegación y target-action.



# Xcode 4.5

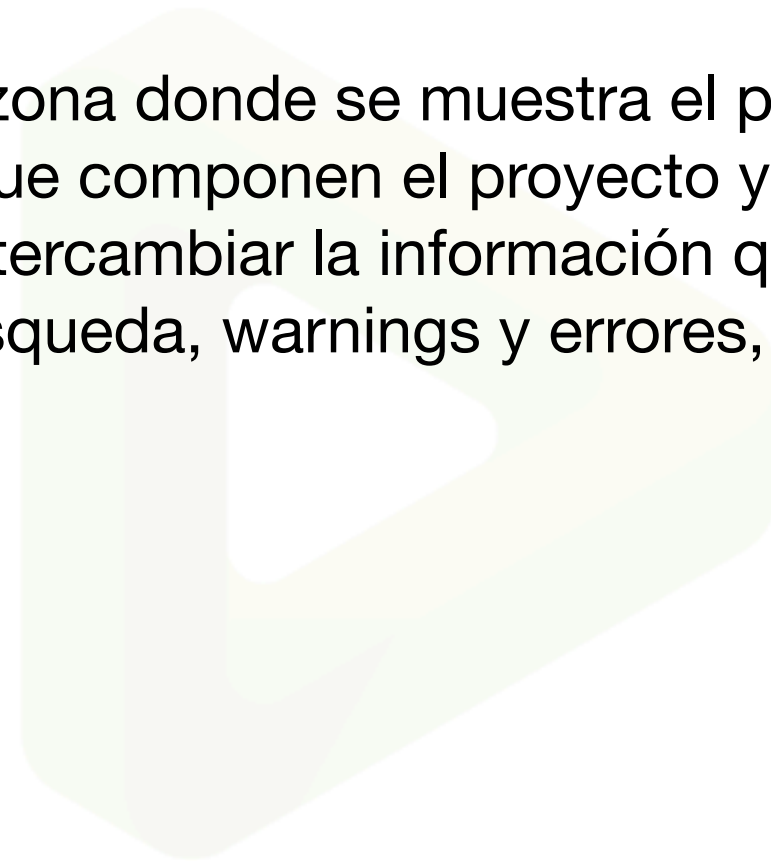


# Xcode 4.5



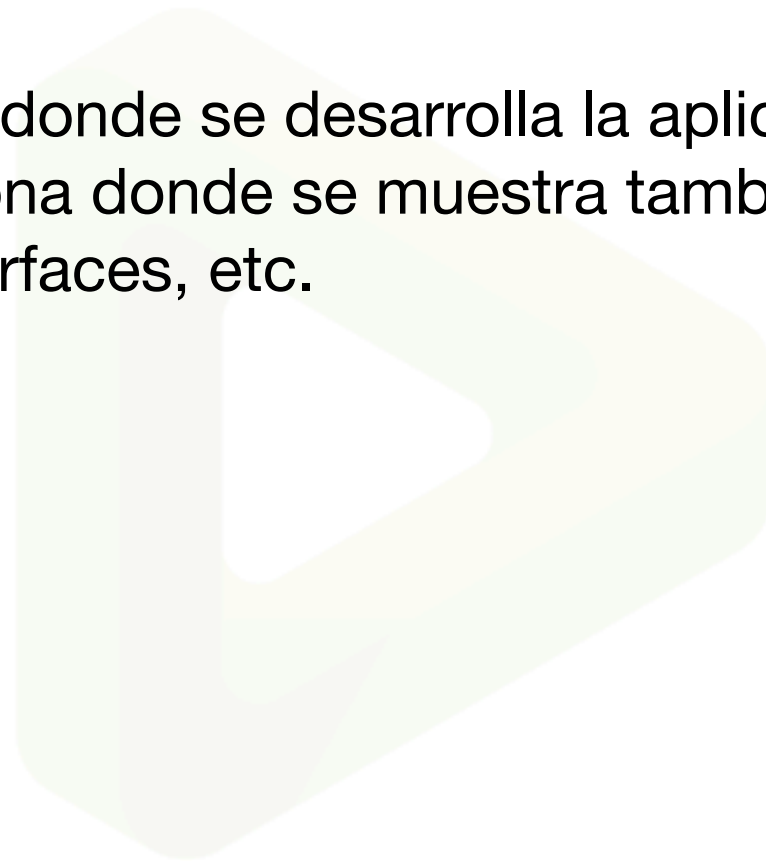
# Xcode 4.5

- **Navigation Area:** Es la zona donde se muestra el proyecto o workspace. Permite ver los ficheros que componen el proyecto y dispone de varias pestañas que permiten intercambiar la información que se muestra entre ficheros del proyecto, búsqueda, warnings y errores, breakpoints, etc.



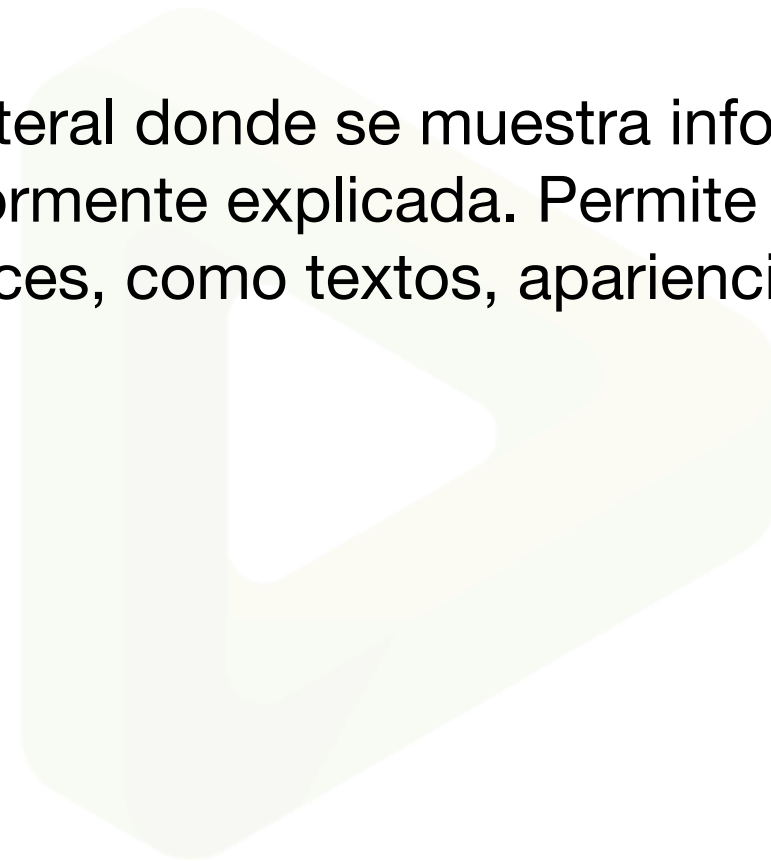
# Xcode 4.5

- **Editor area:** es la zona donde se desarrolla la aplicación y se escribe el código. Además, es la zona donde se muestra también el contenido de otros ficheros como plists, interfaces, etc.



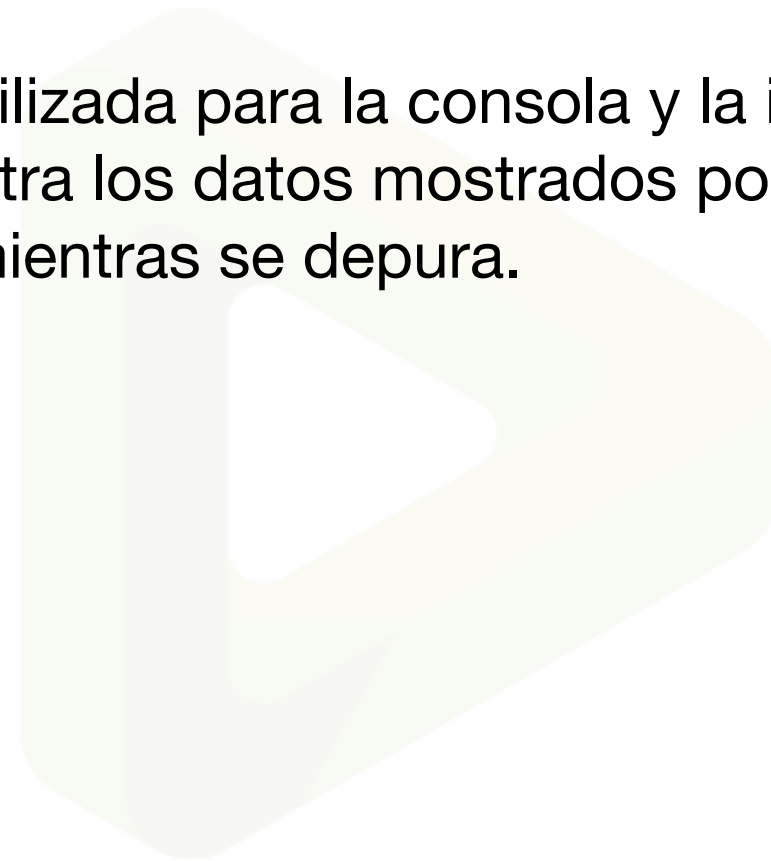
# Xcode 4.5

- **Utility area:** es una zona lateral donde se muestra información de los datos mostrados en la zona anteriormente explicada. Permite modificar también información sobre las interfaces, como textos, apariencia de los elementos, etc.



# Xcode 4.5

- **Debug area:** es la zona utilizada para la consola y la información a la hora de depurar la aplicación. Muestra los datos mostrados por consola, así como valores de las variables utilizadas mientras se depura.



# Novedades en Xcode

## 4.5

# Novedades en Xcode 4.5

- Uso de LLDB como depurador
- Auto layout

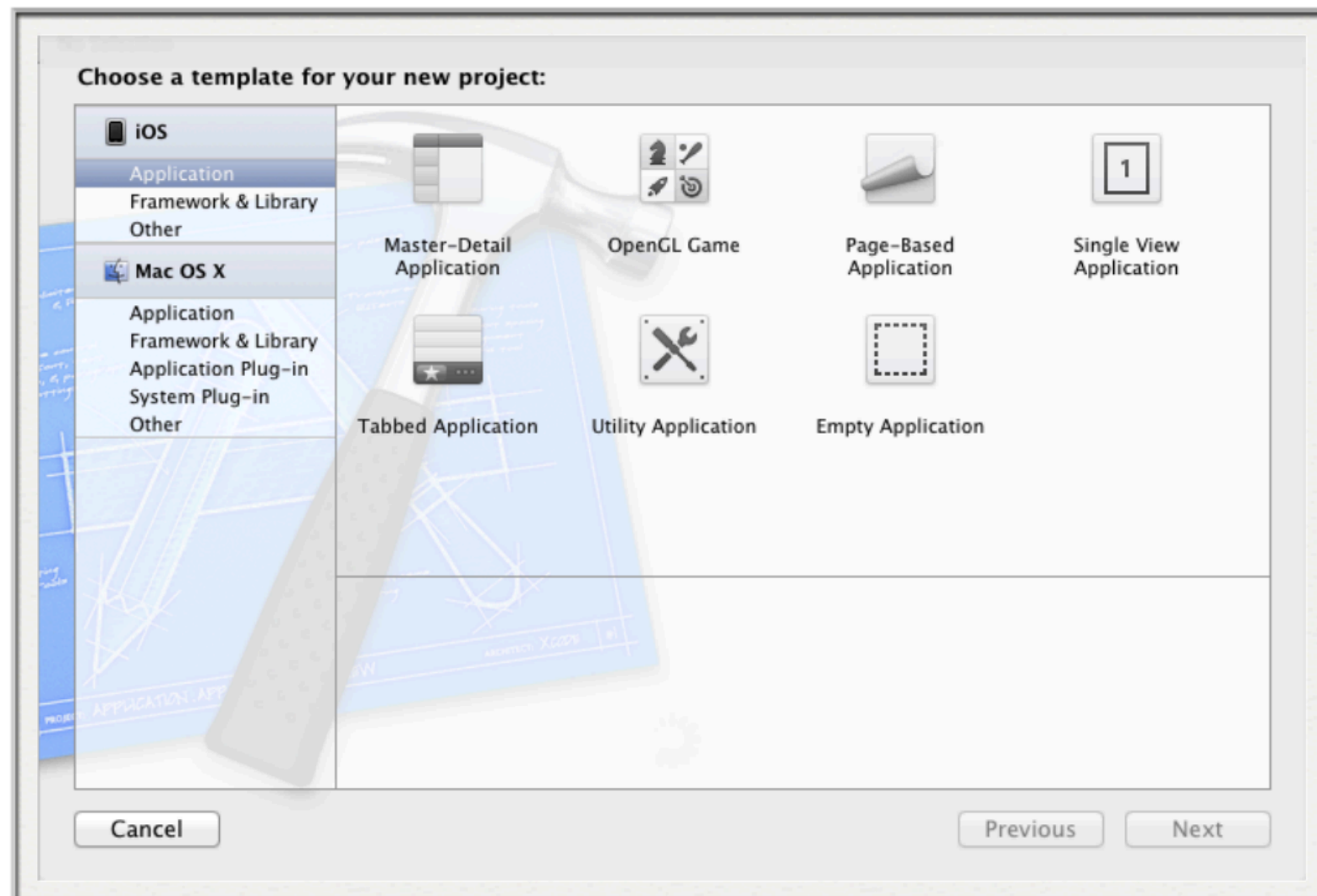


# Tipos de proyecto





# Tipos de proyecto



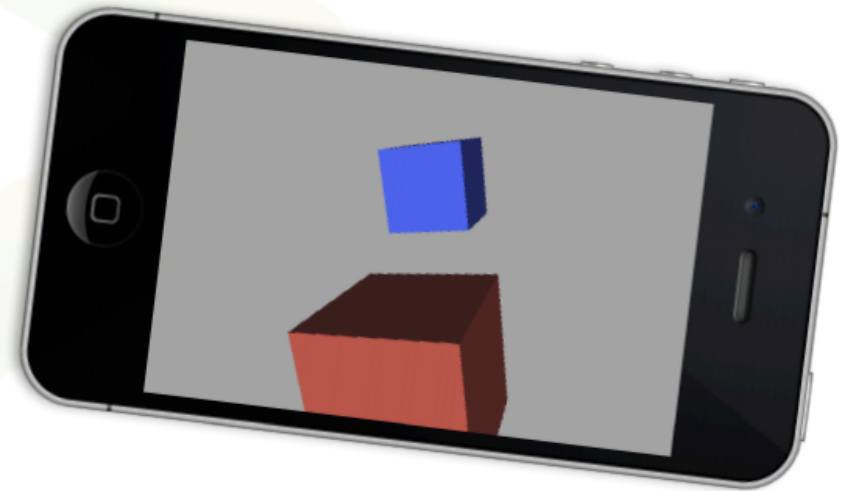
# Tipos de proyecto

- **Master-Detail Application:** permite iniciar un proyecto con dos vistas, una llamada maestro que consiste en una lista y otra llamada detalle, que muestra la información del elemento seleccionado en la lista. En iPad este tipo de aplicaciones se crean con una vista dividida, donde a la izquierda aparece la vista maestro y a la derecha la vista detalle.



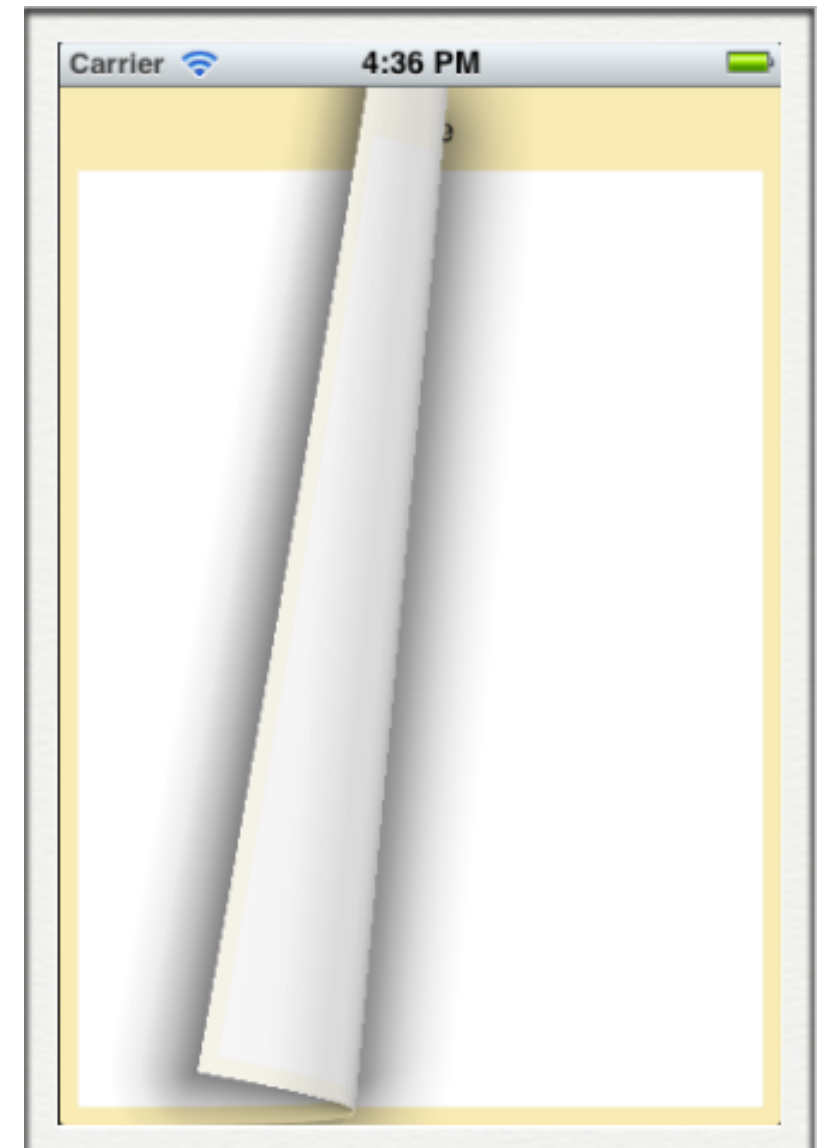
# Tipos de proyecto

- **OpenGL Game:** aplicación orientada al uso de OpenGL ES mediante el cual se representan objetos e imágenes que pueden ser animados.



# Tipos de proyecto

- **Page-Based Application:** aplicaciones que hacen uso de un controlador que simula el uso de páginas. Orientado especialmente a aplicaciones de tipo libro o revista.



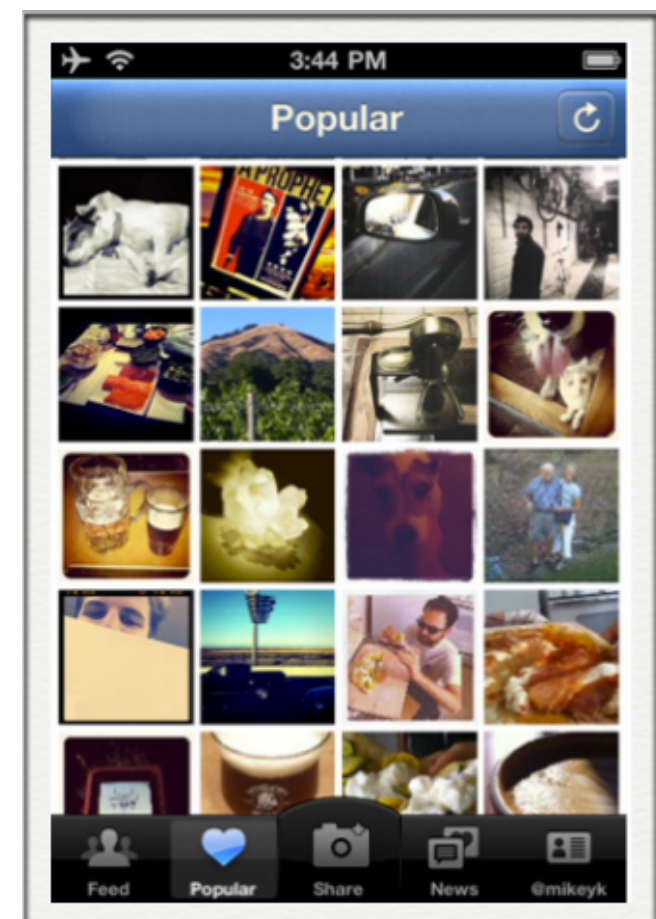
# Tipos de proyecto

- **Single View Application:** tipo básico para un proyecto con una única vista y controlador desde el cual iniciar el proyecto.



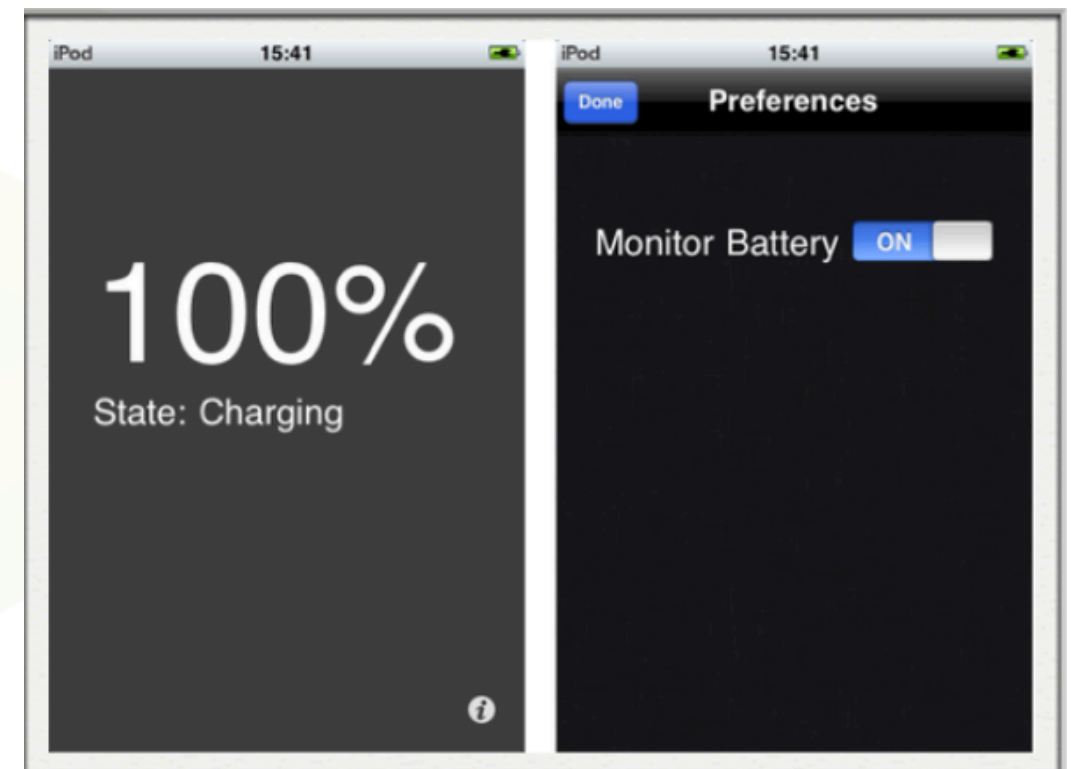
# Tipos de proyecto

- **Tabbed Application:** aplicaciones que hacen uso de pestañas para mostrar distintas vistas.



# Tipos de proyecto

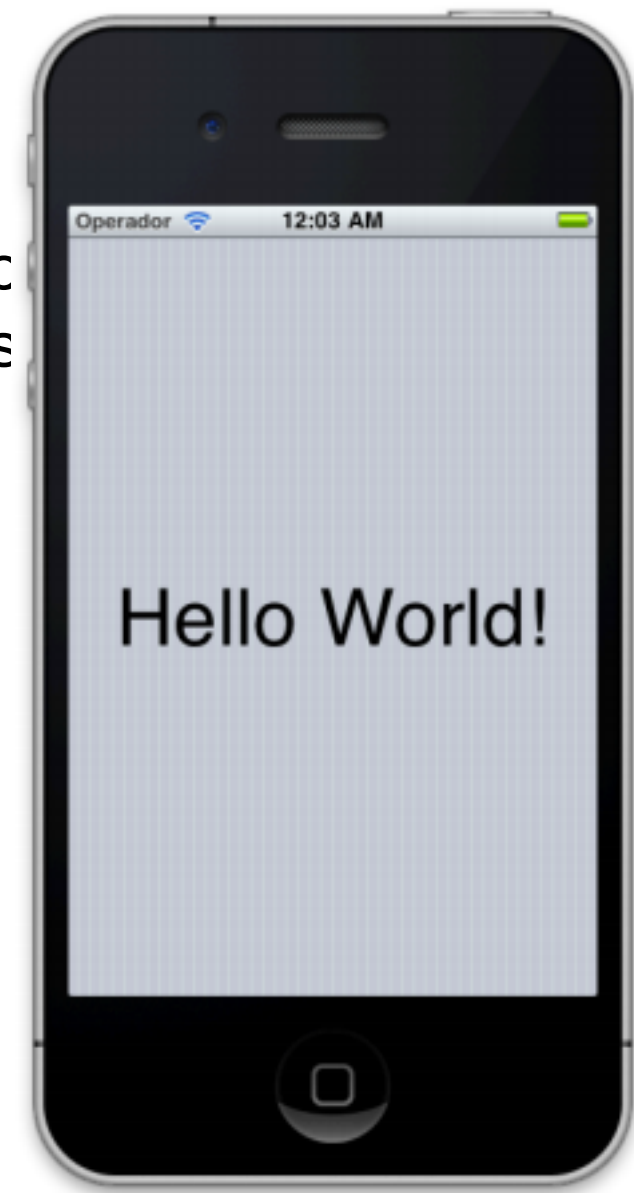
- **Utility Application:** proyecto iniciado con dos vistas, una principal y una alternativa. En iPhone las vistas se intercambian mientras que en iPad la vista alternativa se muestra como popover.





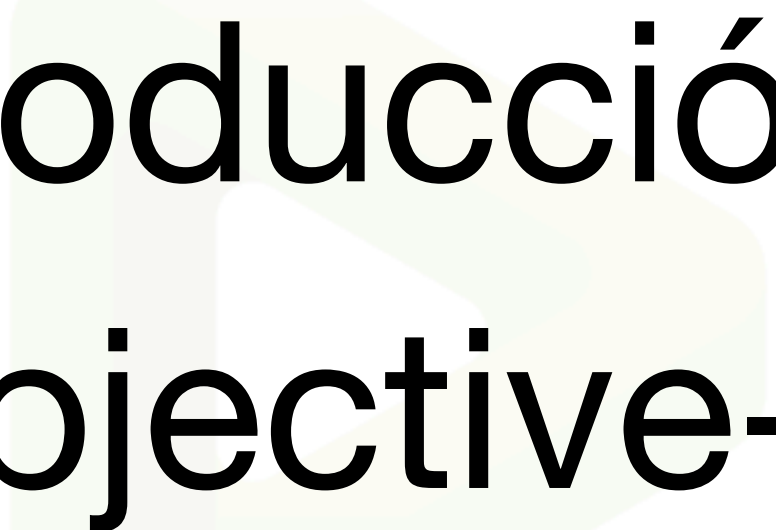
# Tipos de Proyecto

- **Empty Application:** Es el tipo más básico de los existentes. Es el adecuado si ninguno de los otros se adapta a nuestros requerimientos.





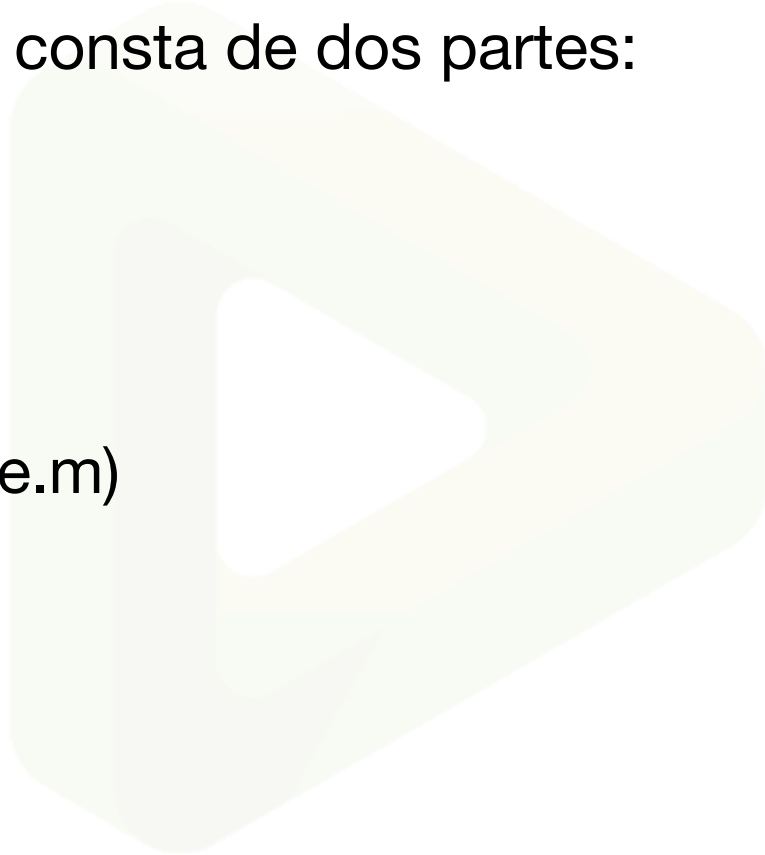
# Introducción a Objective-C



# Introducción a Objective-C

- Es el lenguaje de programación empleado.
- Es un lenguaje de alto nivel y orientado a objetos.
- Es una capa por encima del lenguaje C y utiliza una sintaxis heredada de Smalltalk llamada paso de mensajes.

# Introducción a Objective-C

- La creación de clases consta de dos partes:
    - Interfaz (Clase.h)
    - Implementación (Clase.m)
- 

# Introducción a Objective-C

- Ejemplo de interfaz:

```
@interface Persona : NSObject
{
    NSString *name;
    NSString *surname;
}

-(NSString *)name;
-(NSString *)surname;

-(void)setName:(NSString *)inputName;
-(void)setSurName:(NSString*)inputSurname;

+(void)helloWorld;

@end
```

# Introducción a Objective-C

- Ejemplo de implementación

```
#import "Persona.h"

@implementation Persona

-(NSString *)name
{
    return name;
}

-(NSString *)surname
{
    return surname;
}

-(void)setName:(NSString *)inputName
{
    name=inputName;
}

+(void)helloWorld
{
    NSLog(@"Hello world!!!");
}

@end
```

# Introducción a Objective-C

- El método `init` es el método utilizado para la inicialización de los objetos.

Su estructura básica es la siguiente:

```
- (id)init {  
    self = [super init]  
    if (self) {  
        // Inicio de las variables del objeto  
    }  
    return self;  
}
```

# Introducción a Objective-C

- El método `dealloc` es utilizado para liberar la memoria de los objetos. Puesto que actualmente se utiliza ARC y por tanto la gestión de la memoria no es manual este método sólo se usa en casos especiales.

Un ejemplo de implementación con gestión manual de memoria sería el siguiente:

```
- (void)dealloc {  
    // Liberar memoria de nuestros objetos  
    [super dealloc];  
}
```

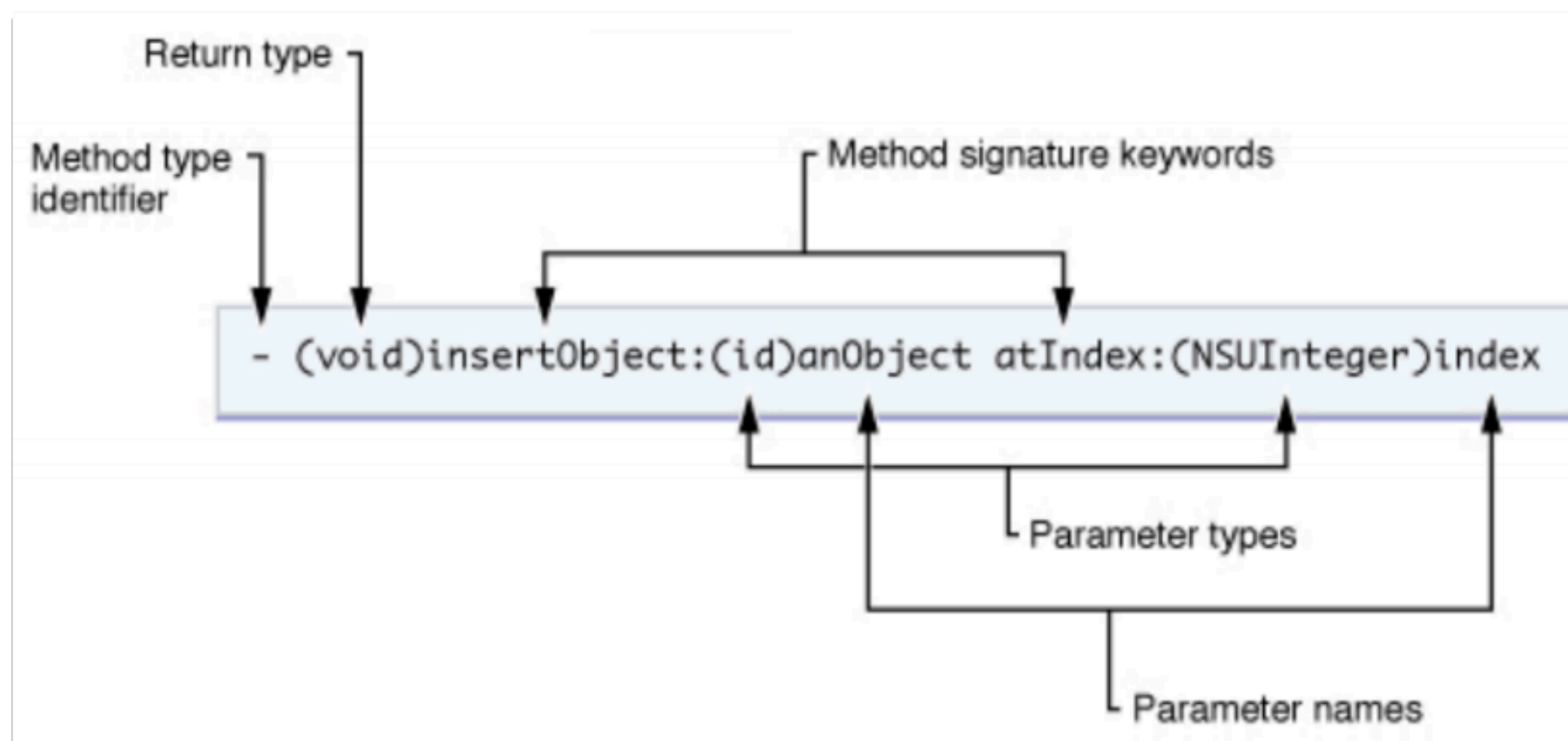
# Objective-C





# Objective-C: Declaración de métodos

- La declaración de métodos se realiza con la siguiente estructura:



# Objective-C: Declaración de métodos

- Method type identifier: existen dos tipos, de instancia, representado por el signo - y de clase, representado por el signo +. El primero se utiliza con variable inicializadas, el segundo no necesita variables.
- Return type: tipo de objeto que devolverá el método. En caso de no devolver nada se utiliza void.

# Objective-C: Declaración de métodos

- Method signature keywords: nombre del método. A diferencia de otros lenguajes, el nombre del método intercala los parámetros que recibe.
- Parameter types y parameter names: Tipo y nombre de los parámetros. Van separados del nombre del método mediante dos puntos.

# Objective-C: Llamadas a métodos

- Para realizar una llamada a un método existen distintas alternativas:

- Sin parámetros: `[objet method];`
- Con un parámetro: `[objet methodWithInput:input];`
- Con varios parámetros:

`[objet methodWithInput1:input1 andInput2:input2];`

- Con valor de retorno:

`output = [objet methodWithOutput];`

- Con valor de retorno y parámetro:

`output = [objet methodWithOutputAndInput:input];`

# Objective-C: Métodos

- Algunos ejemplos:

```
- (int)multiplica:(int)a por:(int)b {  
    return a*b;  
}  
- (NSString *)cadenaConResultado:(int)resultado {  
    return [NSString stringWithFormat:@"El resultado es %d", resultado];  
}  
int resultado = [self multiplica:2 por:3];  
NSString *cadena = [self cadenaConResultado:resultado];
```

# Objective-C: Tipado

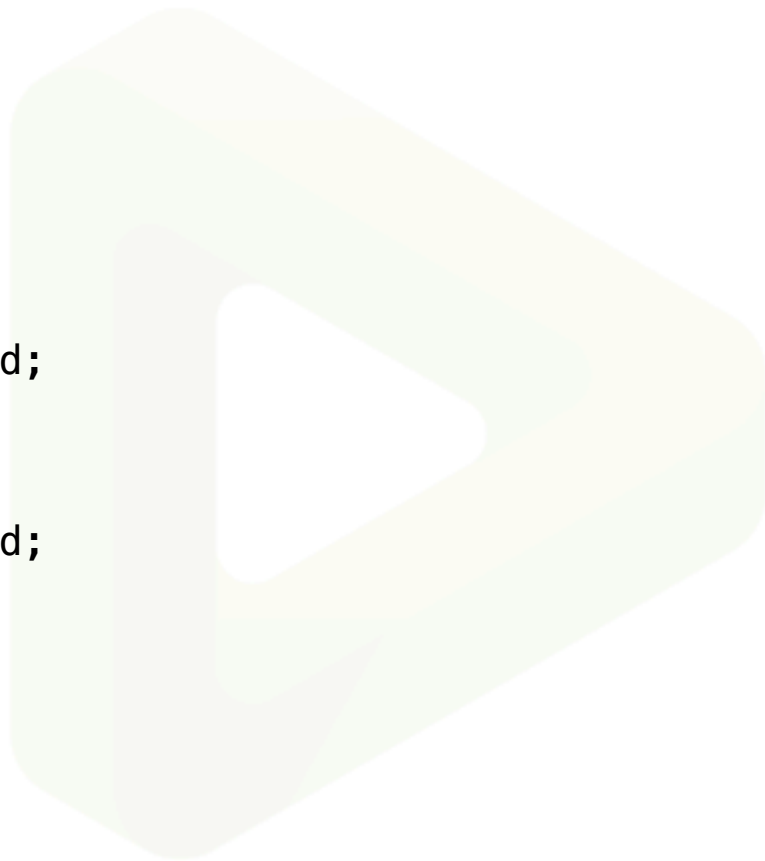
- El tipado existente en Objective-C puede ser tanto fuerte como débil.
- `UIButton *button` es un ejemplo de tipado fuerte.
- `id button` es un ejemplo de tipado débil.

# Objective-C: Protocolos

- Los protocolos en Objective-C equivalen a las interfaces en Java o las clases virtuales en C++.
- Los protocolos se utilizan para garantizar la implementación de métodos en los objetos.
- Declaran métodos que deben ser implementados por las clases que adopten el protocolo.

# Objective-C: Protocolos

```
@protocol MyProtocol  
- (void)requiredMethod;  
  
@optional  
- (void)anOptionalMethod;  
- (void)anotherOptionalMethod;  
  
@required  
- (void)anotherRequiredMethod;
```





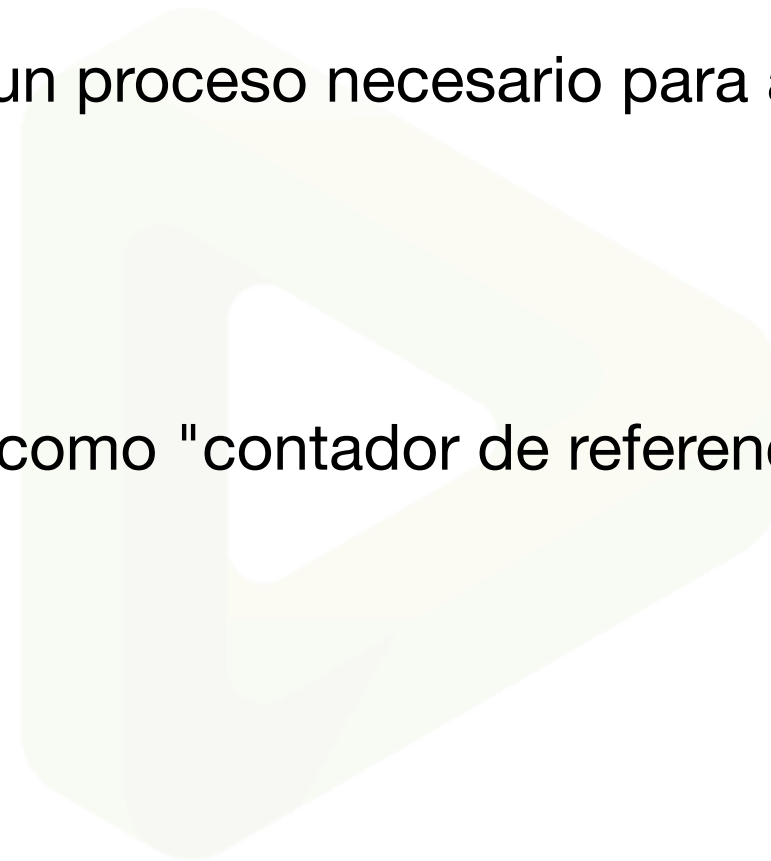
# Objective-C: Protocolos

- Para añadir un protocolo a una clase bastaría con escribir el nombre del protocolo entre los signos <> en la interfaz de la clase

```
@interface Object : NSObject <Protocol>
```

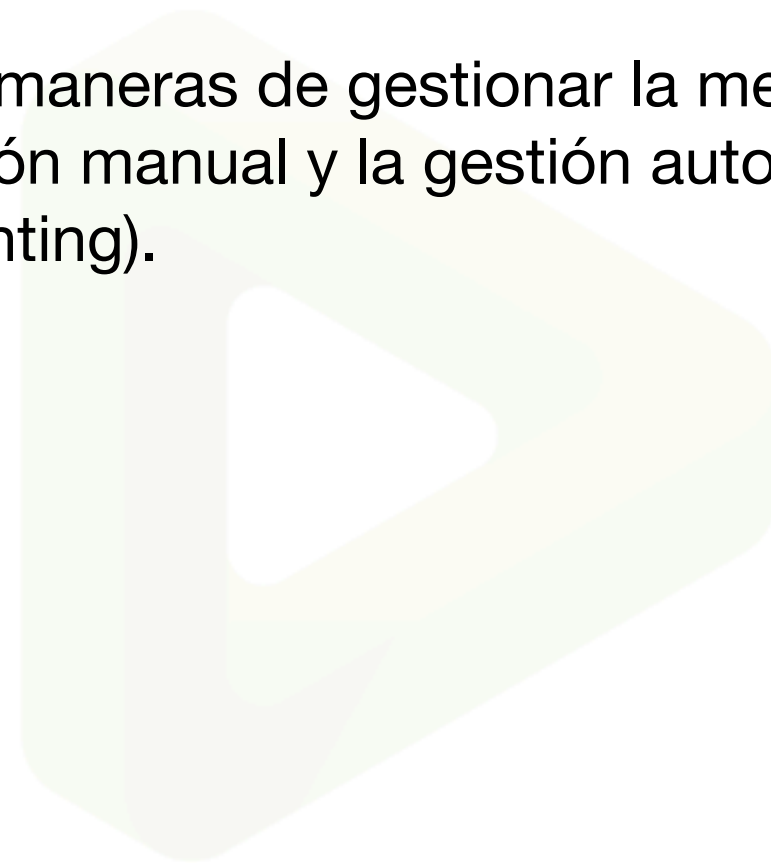
# Objective-C: Gestión de memoria

- La gestión de memoria es un proceso necesario para asegurarnos que el uso de los recursos es adecuado.
- En Objective-C se conoce como "contador de referencias".



# Objective-C: Gestión de memoria

- Actualmente existen dos maneras de gestionar la memoria en Objective-C. La primera de ellas es la gestión manual y la gestión automática, llamada ARC (Automatic Reference Counting).



# Objective-C: Gestión de memoria manual

- En Objective-C se dispone de un contador de referencias que indica el número de objetos que son propietarios de otro.
- En caso de que el contador sea 0 el objeto se libera y deja de estar accesible por lo que si intentamos acceder a él se producirá un error.
- Al crear un objeto estamos indicando que se aumente el contador por lo que al dejar de usarlo debemos de restar uno al contador.

# Objective-C: Gestión de memoria manual

- Alloc: se utiliza para inicializar la memoria de un objeto dependiendo de su tipo.

```
NSNumber *number = [[NSNumber alloc] initWithInt:2];
```

- En este momento, el objeto tendría a 1 el contador de referencia interno.

# Objective-C: Gestión de memoria manual

- Retain: se utiliza para aumentar en uno el contador de referencia.

```
NSNumber *number = [[NSNumber alloc] initWithInt:2];  
[number retain];
```

- En este momento el objeto tendría el valor 2 en su contador.

# Objective-C: Gestión de memoria manual

- Release: se utiliza para decrementar el contador.

```
NSNumber *number = [[NSNumber alloc] initWithInt:2];  
[number retain];  
[number release];
```

- El contador sería uno, puesto que se dispone de alloc y retain que suman dos, pero se ha realizado un release que decrementa el contador en uno.

# Objective-C: Gestión de memoria manual

- Autorelease: De forma similar a release, decrementa en uno el contador pero no se realiza inmediatamente.
- Especialmente útil en los métodos que devuelven un objeto que acaban de crear.

```
-(NSNumber *)crearNumero {  
    return [[[NSNumber alloc] initWithInt:2] autorelease];  
}
```

- El método crea el objeto pero no necesita retenerlo porque no va a utilizarlo, por lo que para permitir que el objeto devuelto pueda ser retenido en otra parte del código se utiliza autorelease para liberarlo pero no de manera inmediata.



# Objective-C: ARC

- Mediante ARC se eliminan los conceptos retain, release y autorelease. En su lugar disponemos de dos tipos de atributos: strong y weak.
- Strong indica que existe una relación de propiedad o fuerte.
- Weak indica que existe una relación de no-propiedad o débil.

# Objective-C: ARC

- A diferencia de la gestión manual, no debemos preocuparnos de realizar retain o release. Tampoco hay problema de acceder a un objeto que se haya liberado puesto que ARC se encarga de no liberarlo si se va a utilizar.
- El único caso donde debemos pensar con estos términos es al declarar properties.

# Objective-C: Conversión a ARC

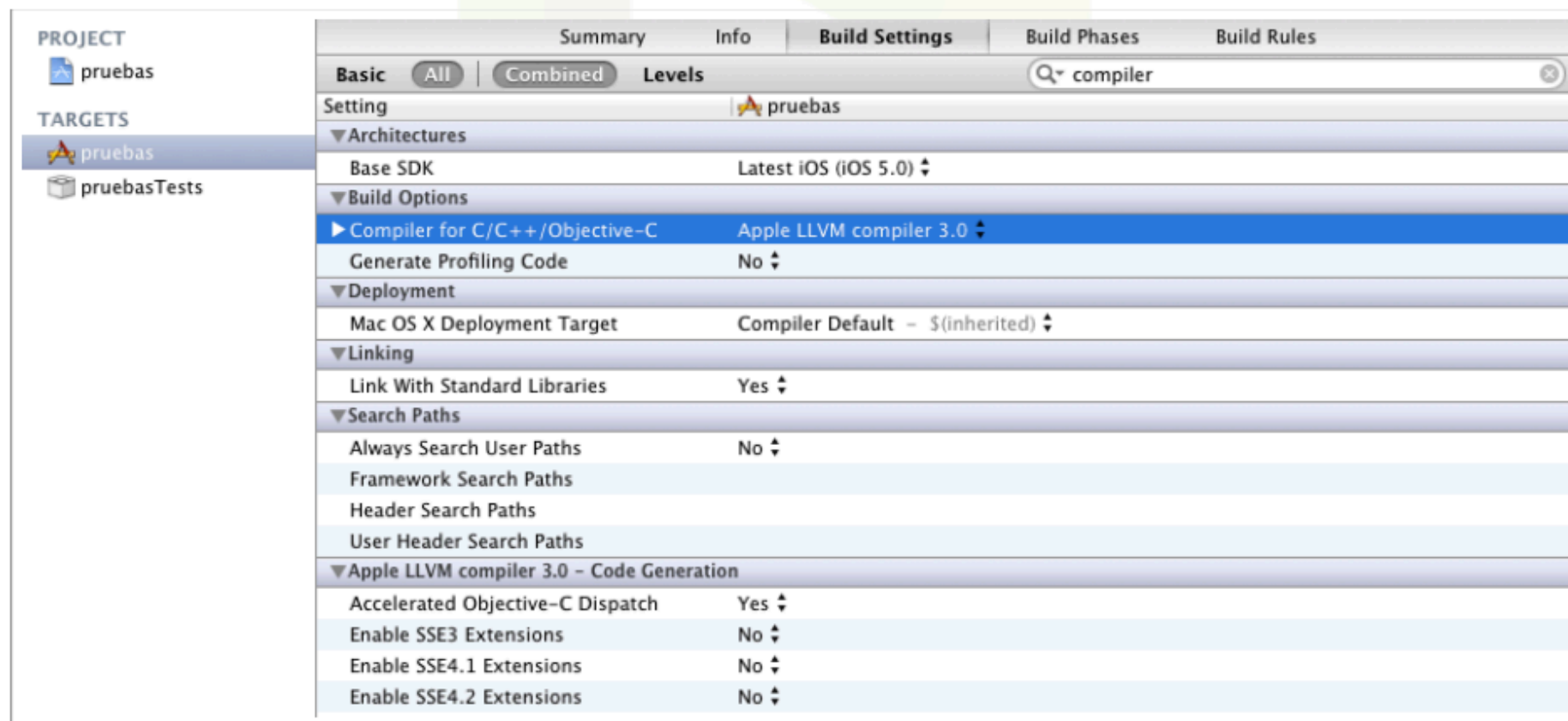
- Para migrar proyectos anteriores a iOS 5 y adaptarlos a la nueva gestión de memoria hay que tener en cuenta una serie de requisitos:
- Para realizar la conversión de un proyecto de iOS es necesario disponer de Snow Leopard o de Lion, en caso de necesitar realizar la conversión a aplicaciones de escritorio es necesario disponer de Lion.
- Hay que realizarse una copia de seguridad del proyecto antes de realizar la conversión.

# Objective-C: Conversión a ARC

Pasos a seguir:

1.- Project settings/Build Settings/Build Options

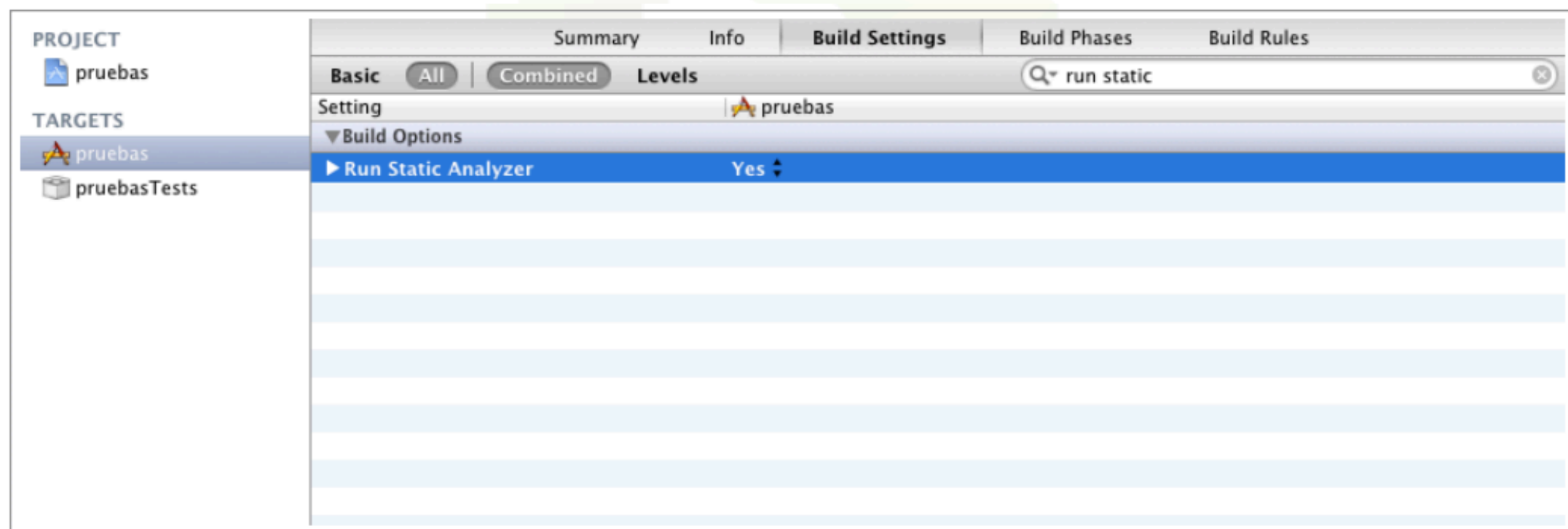
- Cambiamos el tipo de compilador por “Apple LLVM compiler 3.0”



# Objective-C: Conversión a ARC

## 2.- Project settings/Build Settings/Build Options

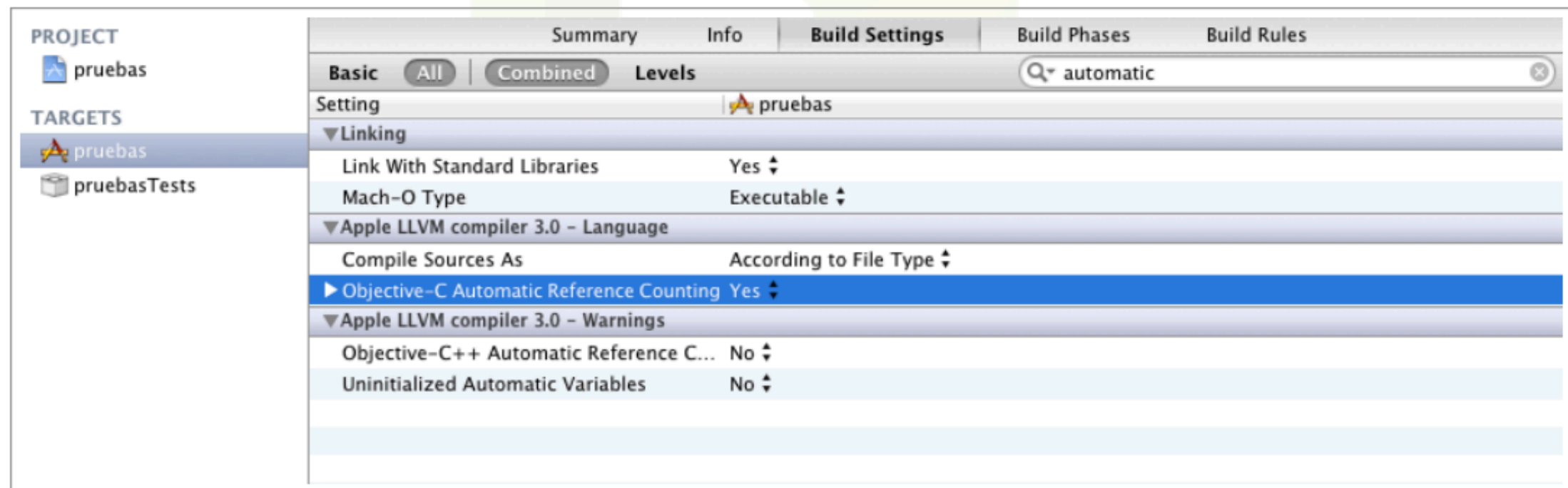
- Habilitamos “Run Static Analyzer”



# Objective-C: Conversión a ARC

## 3.- Project settings/Build Settings/Apple LLVM compiler

- Habilitamos “Objective-C Automatic Reference Counting”



# Objective-C: Conversión a ARC

4.- Compilamos la aplicación.

- En caso de que existan errores deberemos revisarlos. para corregirlos deberemos editar nuestro proyecto de la siguiente forma:

- Edit/Refactor/Convert to Objective-C ARC.

- Nos mostrará los ficheros de nuestro proyecto que debemos convertir a ARC.

- Deberemos corregir todos los warnings de la aplicación para adaptarla a ARC.

# Objective-C: Properties

- Las properties o propiedades permiten generar métodos de acceso para los atributos de clase.
- Se definen en la interfaz mediante @property.
- La declaración de los atributos dependerá del tipo de gestión de memoria que decidamos usar.



# Objective-C: Properties

- En caso de usar gestión manual podremos declarar las properties como retain.
- En caso contrario, utilizaremos strong y weak.
- Un ejemplo de property sería el siguiente:

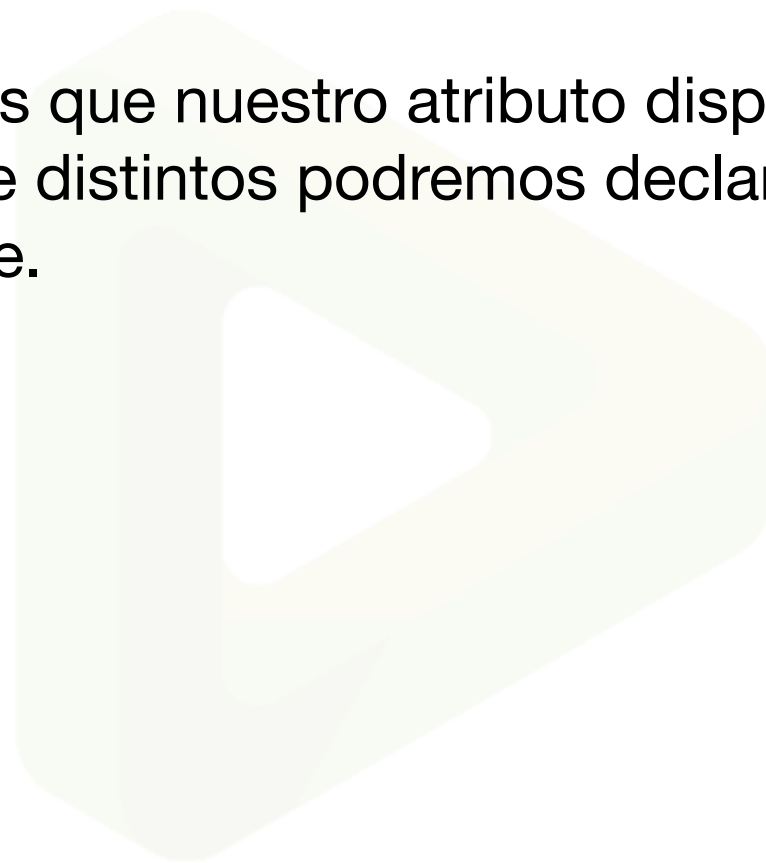
```
@property (retain) NSString *name;
```

# Objective-C: Properties

- En el ejemplo, declaramos el atributo name con retain, de forma que el setter hará retain sobre el parámetro de entrada.
- Anteriormente era necesario declarar los atributos, las properties y además realizar @synthesize de nuestros atributos en la implementación. Con las últimas versiones podemos declarar únicamente la property.
- Por ejemplo @property (strong) NSString \*name creará el método setName, name para obtenerlo y podremos acceder a la variable mediante \_name.

# Objective-C: Properties

- En caso de que queramos que nuestro atributo disponga de unos nombres de setter y getter y de variable distintos podremos declarar la variable y realizar el synthesize correspondiente.



# Objective-C: Log

- Es posible que nuestra aplicación muestre mensajes por consola. Es especialmente útil a la hora de depurar la aplicación y encontrar qué está fallando.
- Basta con utilizar la función NSLog de la siguiente manera:
- `NSLog(@"La fecha actual es %@", [NSDate date]);`

# Framework Foundation



# Framework Foundation - NSObject

- NSObject es la clase base de los objetos existentes en iOS.
- Implementa los métodos de gestión de memoria de los objetos.
- Dispone de métodos importante y utilizados habitualmente como description o performSelector:withObject:..

# Framework Foundation - NSString

- La clase NSString se utiliza para el manejo de cadenas de texto Unicode.
- Dispone de multitud de métodos para trabajar con ficheros, URLs, combinar cadenas, dividirlos, encontrar caracteres o subcadenas, etc.
- Usando la instrucción @"texto" se crea un objeto NSString.

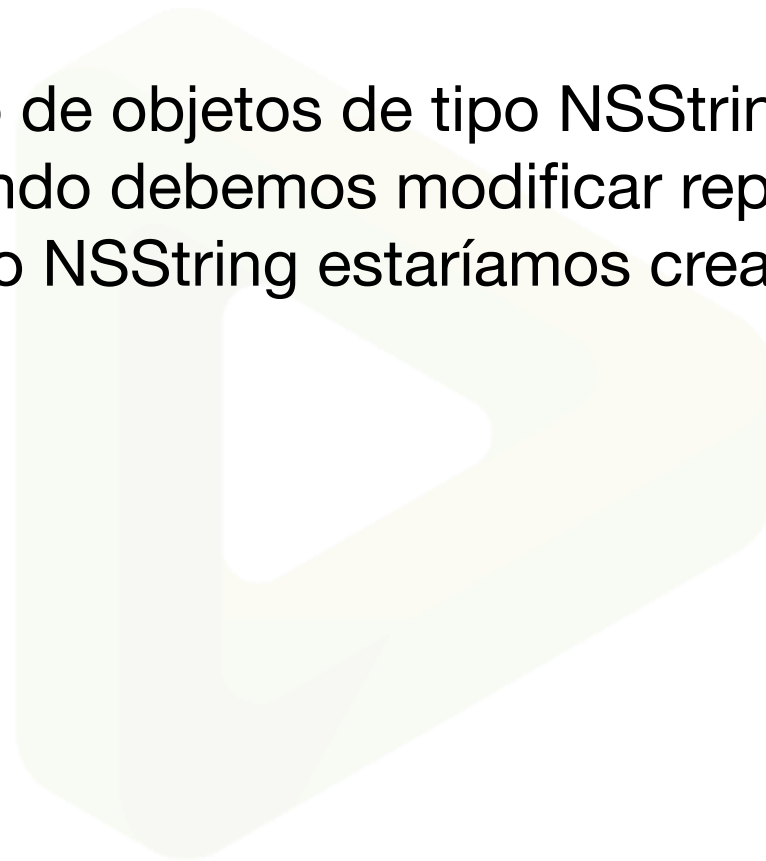
# Framework Foundation - NSString

- NSMutableString es la versión mutable de NSString.
- Cuando creamos un objeto NSString no podemos modificar su cadena, únicamente podemos crear una nueva.
- Mediante NSMutableString podemos modificar cadenas, añadir caracteres o borrarlos.



# Framework Foundation - NSString

- Lo habitual suele ser el uso de objetos de tipo NSString. El objeto NSMutableString es útil cuando debemos modificar repetidas veces una cadena, puesto que de usar un objeto NSString estaríamos creando numerosos objetos.

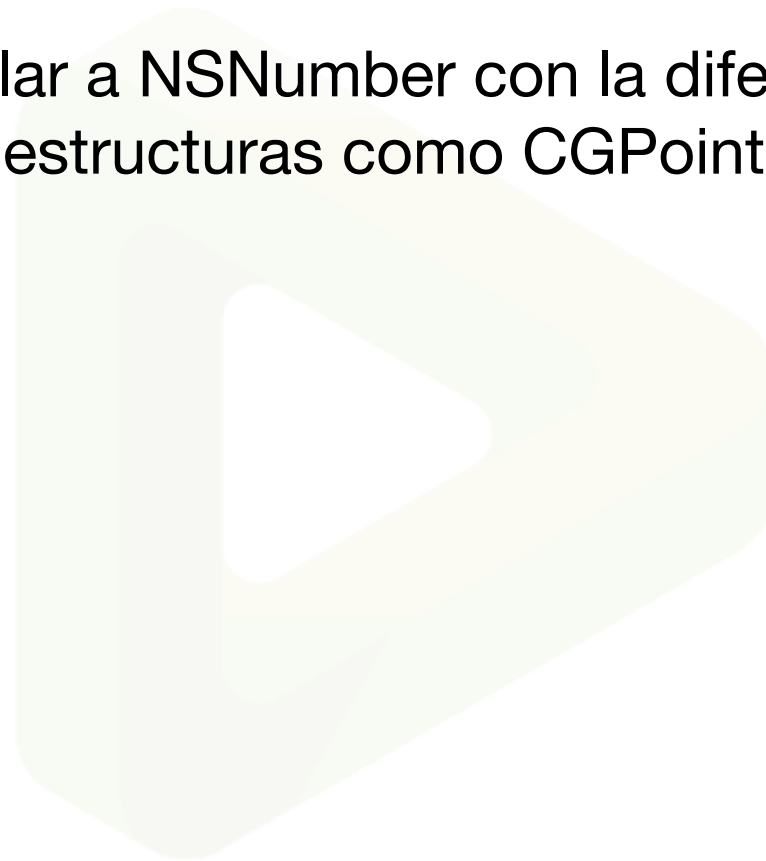


# Framework Foundation - NSNumber

- La clase NSNumber permite encapsular tipos básicos como int o float.
- Es realmente útil a la hora de almacenar tipos primitivos en objetos como arrays o diccionarios.
- Dispone de multitud de métodos para iniciar objetos así como métodos para comparar objetos de tipo NSNumber.

# Framework Foundation - NSValue

- El tipo NSValue es similar a NSNumber con la diferencia de que permite crear objetos a partir de estructuras como CGPoint, que contiene dos float, x e y.

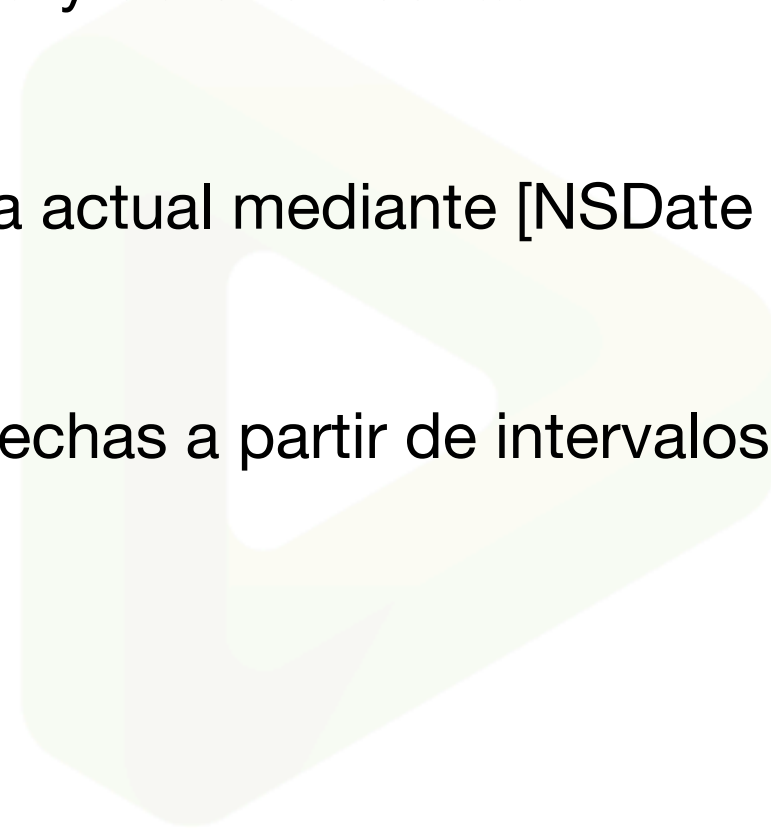


# Framework Foundation - NSData

- NSData es un tipo de objeto que permite el encapsulamiento de bytes.
- Dispone de una versión mutable, NSMutableData.
- Mediante NSData podemos encapsular bytes y leer y escribir contenido de ficheros y URLs.

# Framework Foundation - NSDate

- NSDate permite almacenar y obtener fechas.
- Podemos obtener la fecha actual mediante [NSDate date].
- También podemos crear fechas a partir de intervalos de tiempo desde otra fecha.



# Framework Foundation - NSArray

- NSArray es la clase utilizada para almacenar objetos de forma ordenada.
- Dispone de una versión mutable, NSMutableArray, por lo que si queremos un array en el que podamos insertar y eliminar objetos debemos emplear el tipo mutable.
- Tiene métodos para comprobar si contiene cierto objeto y obtener el índice, obtener el último objeto o el objeto en un índice concreto, así como ordenarlo

# Framework Foundation - NSMutableSet

- NSMutableSet permite almacenar un conjunto de objetos no ordenados.
- Es posible la creación de un NSMutableSet en caso de que queramos añadir y eliminar elementos.
- Al igual que NSArray dispone de métodos para recuperar objetos, comprobar si existen y comparación entre objetos de tipo NSMutableSet.

# Framework Foundation - NSDictionary

- La clase NSDictionary permite almacenar pares clave valor, donde la clave suele ser de tipo NSString y el valor el objeto que deseemos guardar.
- Al igual que con NSSet y NSArray, existe la versión mutable, NSMutableDictionary.
- Existen métodos para obtener las claves, los valores, obtener el objeto almacenado en una clave, etc.



# Framework Foundation - NSMutableOrderedSet

- La clase NSMutableOrderedSet permiten almacenar un conjunto de objetos de forma ordenada. Su uso es una mezcla de NSArray y NSMutableSet.
- Se recomienda su uso cuando los objetos a almacenar deben estar ordenados y es importante conocer si un objeto pertenece al conjunto o no, puesto que es más rápido comprobarlo en un conjunto que en un array.
- Dispone de una versión mutable NSMutableOrderedSet y tiene métodos para comprobar si contiene cierto objeto, así como para conocer el índice de un objeto o recuperarlo según su índice.

# Framework Foundation - NSURL

- La clase NSURL permite trabajar con URLs y los recursos a los que referencian.
- En conjunto con otros tipos como NSString o NSData cuyos métodos stringWithContentsOfURL: y dataWithContentsOfURL: reciben como parámetro un objeto de tipo NSURL podemos obtener el contenido al que referencia cierta URL.

# Framework Foundation - NSJSONSerialization

- Los objetos de tipo NSJSONSerialization permiten convertir ficheros JSON a objetos Foundation y viceversa.
- Los objetos obtenidos al convertir un JSON son de tipo NSString, NSNumber, NSArray, NSDictionary y NSNull.
- Los objetos de mayor nivel son de tipo NSArray o NSDictionary.
- Las claves son de tipo NSString.

# Framework Foundation - NSXMLParser

- Los objetos NSXMLParser permiten parsear ficheros XMLs.
- Junto a NSXMLParserDelegate se pueden obtener los objetos adecuados al recibir un XML.
- Al contrario que con un fichero JSON, al parsear un XML somos nosotros quienes debemos obtener los valores y crear los objetos necesarios.