



Desarrollo iOS 6

iPhone & iPad



Fernando Rodríguez

fernando@agbo.biz

@frr149



Desarrollo para iOS

Herramientas y Tecnologías



Copyright © 2012 AGBO Business Architecture S.L. Todos los derechos reservados. www.agbo.biz

Tecnologías

- Desarrollo Nativo: Objective C y Cocoa
- RubyMotion
- Lua
- PhoneGap



Desarrollo Nativo

- Objective C y Cocoa: la alternativa de Apple
- Su conocimiento es fundamental para desarrollar para iOS
- Las herramientas son gratis
- Necesitas un Mac



Ventajas

- Es el idioma nativo de iOS y todas las novedades están disponibles primero para él.
- Tiene una enorme demanda laboral en estos momentos.
- Conocimiento con futuro: a mediados de 2012 Objective C ya es el tercer lenguaje más usado del mundo, a punto de superar a Java.



RubyMotion

- Se programa en Ruby, siguiendo las mismas normas y hábitos de ROR.
- Poca demanda.
- Tienes que tener una buena base de Objective C y Cocoa.
- Bueno si eres muy fan ROR



Lua & Corona o Moai

- Muy orientado a juegos
- Lenguaje muy sencillo
- No adecuado para aplicaciones complejas, especialmente si no son juegos
- Bueno para empezar si no tienes NINGUNA experiencia previa de programación
- Hay que saber algo de Objective C y Cocoa



PhoneGap

- Interfaz con html 5 y CSS
- Funcionalidad con Javascript
- Para apps triviales que necesitan ejecutarse en otras plataformas también.
- Poco rendimiento





Copyright © 2012 AGBO Business Architecture S.L. Todos los derechos reservados. www.agbo.biz

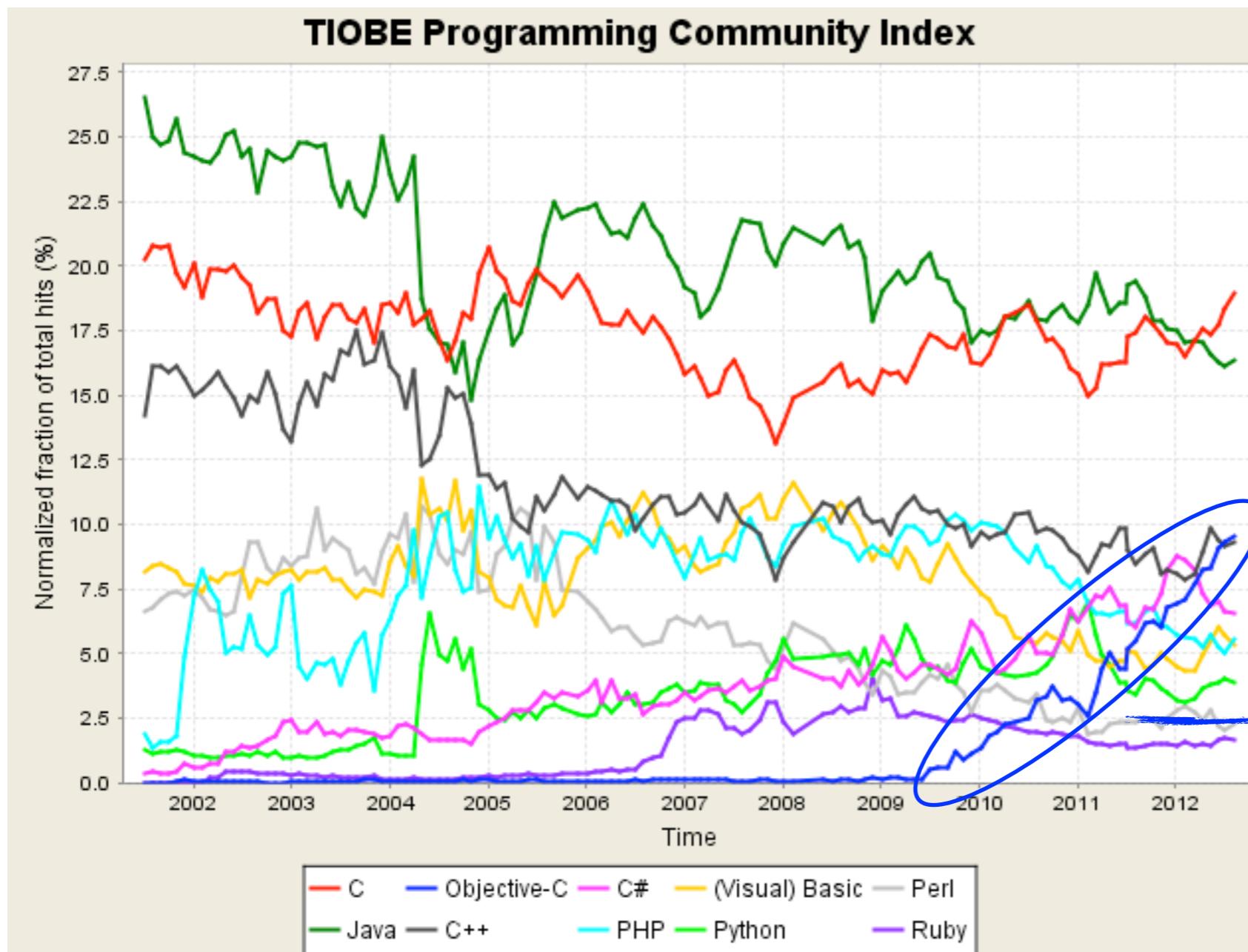
Desarrollo Nativo

Cocoa Touch & Objective C



Copyright © 2012 AGBO Business Architecture S.L. Todos los derechos reservados. www.agbo.biz

Objective C & Cocoa



Mientras los demás lenguajes se mantienen más o menos estables, Objective C tiene un crecimiento exponencial desde el 2009 y es ya el 3º más usado.

Herramientas

- Entorno de desarrollo: Xcode
- Lenguaje: [display [setColor: [UIColor redColor]]];
- Frameworks: Foundation, UIKit, Core Data, Core Location, MapKit, etc...
- Patrones de diseño: MVC (Model View Controller), Delegate



Un Vistazo al Sistema Operativo

iOS



- Es un Unix BSD con Mach 3
- Mismo núcleo que OSX
- Podemos usar todas las librerías estándar de Unix y C

Capas del SO

iOS

Objective C

Librerías de Clases

Multimedia

Infraestructura Básica

Sistema Operativo



Capas del SO

iOS

Objective C

Librerías de Clases

Multimedia

Infraestructura Básica

Sistema Operativo

BSD UNIX

Acceso a librerías
estándar via C



Capas del SO

iOS

Objective C

Librerías de Clases

Multimedia

Infraestructura Básica

Sistema Operativo

Colecciones
Acceso a ficheros
Grand Central
Dispatch
Orientado a
Objeto

Capas del SO

iOS

Objective C

Librerías de Clases

Multimedia

Infraestructura Básica

Sistema Operativo

Core Audio
Core Image
AVFoundation
Grabación y
reproducción de
Audio y Video



Capas del SO

iOS

Objective C

Librerías de Clases

Multimedia

Infraestructura Básica

Sistema Operativo

Vistas (Controles)
MVC
Delegados
Gestos
Animaciones



Capas del SO

iOS

Objective C

Librerías de Clases

Multimedia

Infraestructura Básica

Sistema Operativo

Orientado a
Objeto
Basado en C
Inspirado en
Smalltalk
Dinámico

Los Orígenes

Objective C y Cocoa



Copyright © 2012 AGBO Business Architecture S.L. Todos los derechos reservados. www.agbo.biz

Origen de Objective C y Cocoa



NeXT Cube firmado por Tim Berners-Lee

- Inventado en los 80 por Brad Cox y Tom Love, uniendo C y la filosofía de Smalltalk.
- Jobs lo licenció y fundó NeXT.
- Cuando Apple compró NeXT en el 96, NextStep pasó a llamarse OSX 6 Cocoa.
- El nombre Cocoa es una broma con Java.

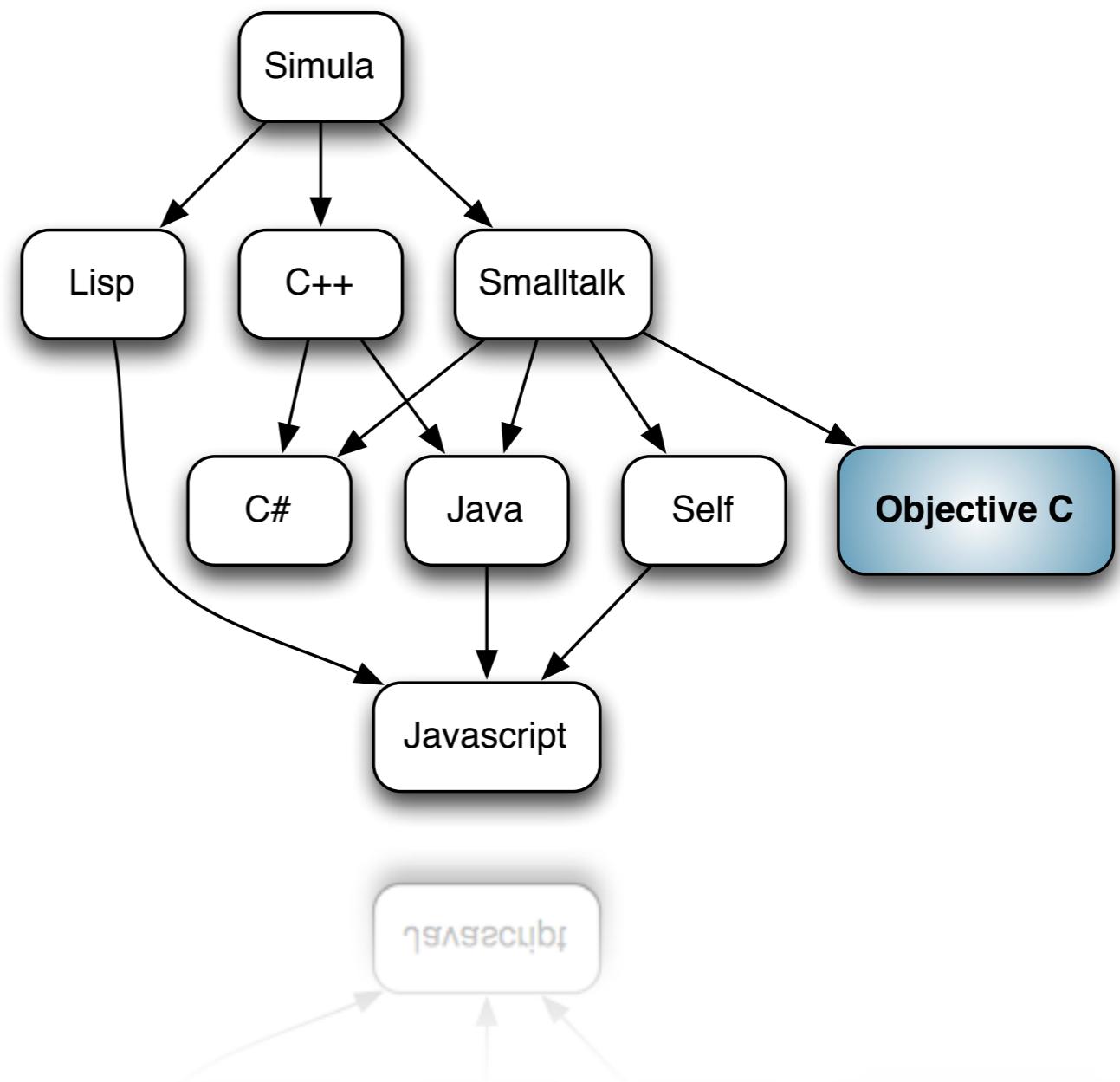
Objective C

Características y Filosofía



Lenguajes Orientados a Objeto

- Hay varias familias de lenguajes
- Cada una representa una “filosofía”
- Algunos son “puros”, otros son “híbridos”



Lenguajes Estrictos vs Dinámicos



vs



Estrictos
Java, C++

Dinámicos
Objective C, Ruby, Python

Introducción a Objective C



Objective C

A language that doesn't affect the way you think about programming, is not worth knowing.

Alan Perlis

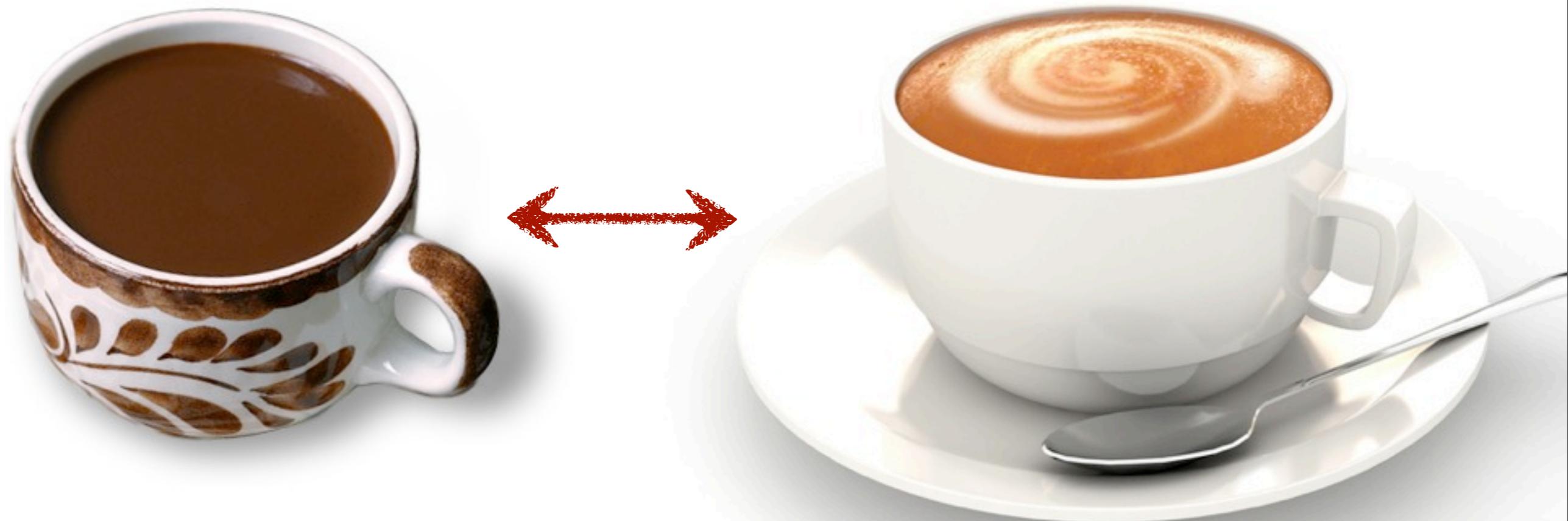
- Basado en C estándar
- Todos los “extras” a C vienen precedidos de un @ o entre corchetes.
- Es dinámico
- Los objetos envían y reciben mensajes



Envío de Mensajes

- Java: “Ejecutamos el método `toString()` del objeto x”
- Objective C: “Al objeto x, le envíamos el mensaje `description`”

Objective C vs Java



Objective C vs Java

- No tiene recolector de basura...pero hay algo parecido llamado ARC.

Objective C vs Java

- No tiene recolector de basura...pero hay algo parecido llamado ARC.
- En el fondo, todos los métodos son públicos.

Objective C vs Java

- No tiene recolector de basura...pero hay algo parecido llamado ARC.
- En el fondo, todos los métodos son públicos.
- No hay clases abstractas de forma explícita.

Objective C vs Java

- No tiene recolector de basura...pero hay algo parecido llamado ARC.
- En el fondo, todos los métodos son públicos.
- No hay clases abstractas de forma explícita.
- No es común usar excepciones.

Objective C vs Java

- No tiene recolector de basura...pero hay algo parecido llamado ARC.
- En el fondo, todos los métodos son públicos.
- No hay clases abstractas de forma explícita.
- No es común usar excepciones.
- Las clases van en dos ficheros .h y .m

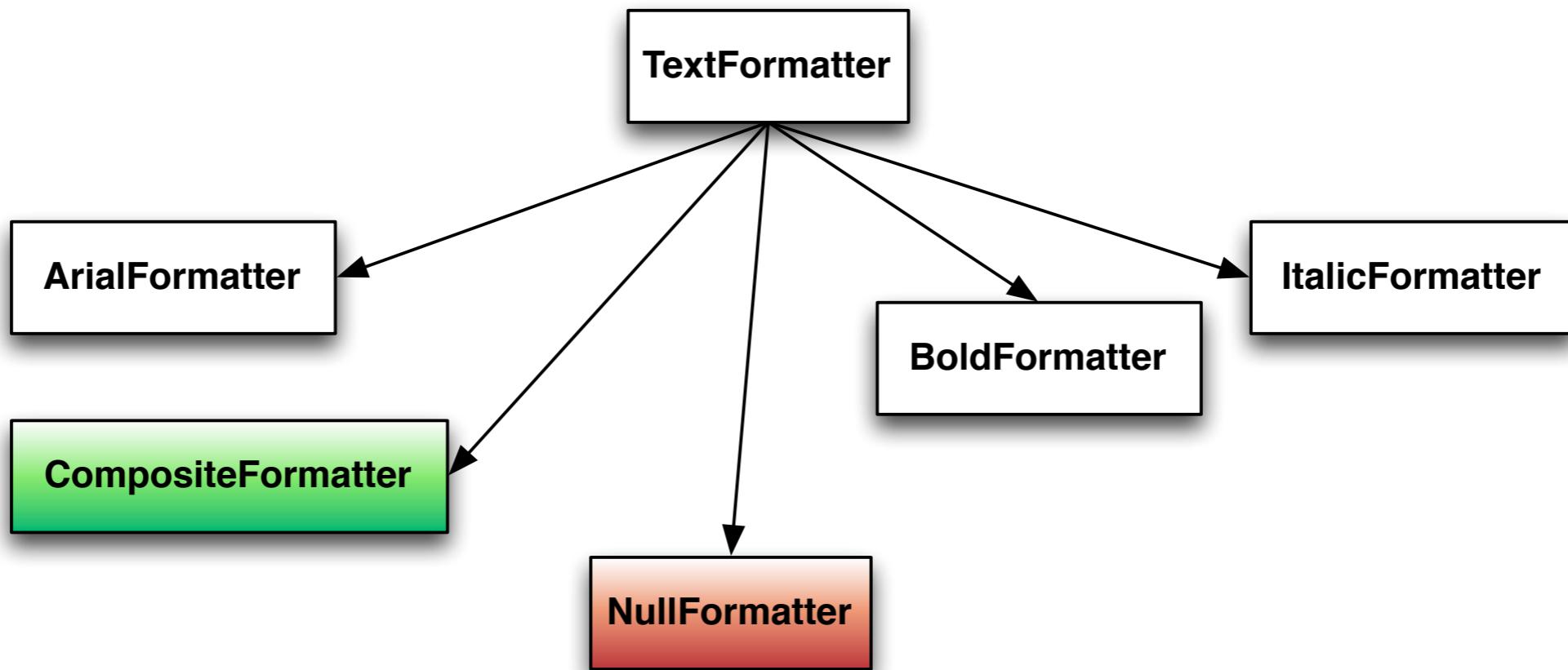
Objective C vs Java

- No tiene recolector de basura...pero hay algo parecido llamado ARC.
- En el fondo, todos los métodos son públicos.
- No hay clases abstractas de forma explícita.
- No es común usar excepciones.
- Las clases van en dos ficheros .h y .m
- Se pueden enviar mensajes a nil (null). De hecho, es muy común.

nil: el objeto nulo

- Toda variable de objeto que no está inicializada, vale nil.
- Cuando mandamos un mensaje a nil...no pasa nada.

Null Object Pattern



Objective C

Sintaxis



Sintaxis de Objective C

- Es rara...
- ...pero diseñada para niños de 5 años. ;-)
- Procura simular una frase en Inglés
- Los nombres de los métodos son largos y autodescriptivos
- En envío de mensajes se hace entre corchetes.



Sintaxis de Objective C

Java

```
cafetera.cafeCorto();
```

Objective C

```
[cafetera cafeCorto];
```

Receptor

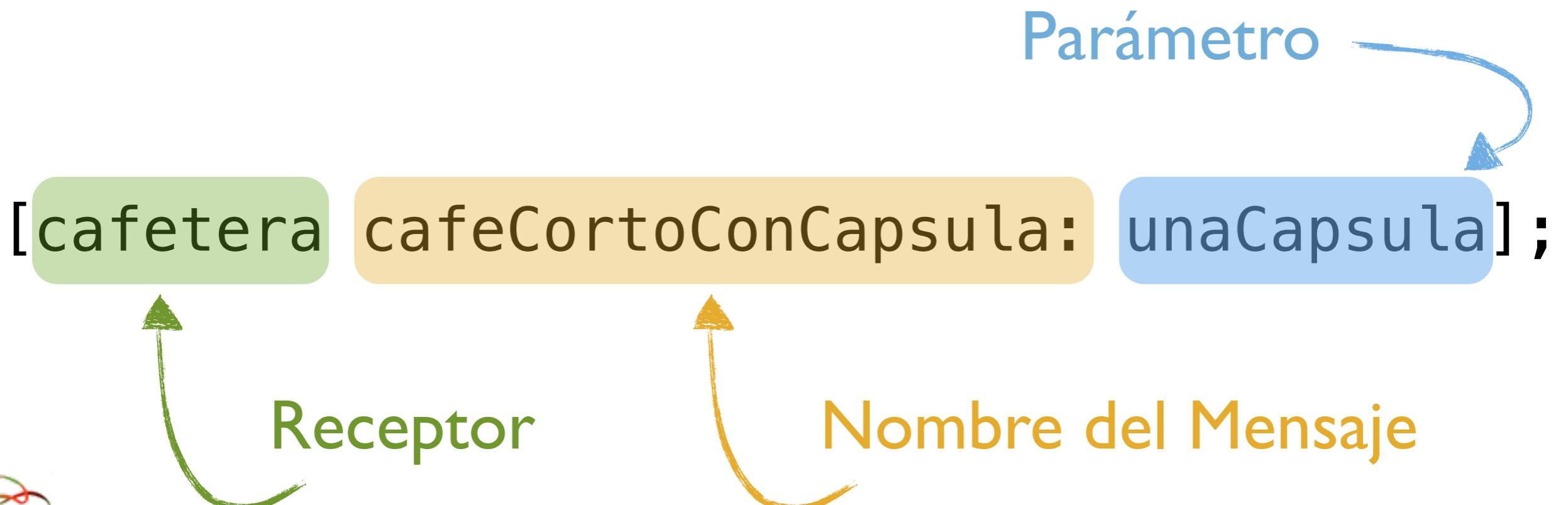


Nombre del Mensaje



Sintaxis de Objective C

cafetera.cafeCorto(capsula);



Sintaxis de Objective C

En C++ o Java, un método para dibujar un círculo podría tener la siguiente sintaxis:

```
this.drawCircle(100, 100, 25, RED, BLUE);
```

Para entender lo que hace, tendríamos que buscar su definición o comentarlo:

```
// Los dos primeros parámetros son el centro  
// (x e y), el tercer es el radio, el cuarto  
// es el color del borde y el último el color  
// del relleno  
this.drawCircle(100, 100, 25, RED, BLUE);
```



Sintaxis de Objective C

Python intenta resolver este problema ofreciendo la posibilidad de pasar los parámetros por nombre:

```
class Canvas:  
    def drawCircle(self, x, y, radius, strokeColor, fillColor):  
        pass  
  
f = Canvas()  
f.drawCircle(radius = 25, x=100, y = 100,  
             strokeColor = red, fillColor = white)
```



Sintaxis en Objective C

```
// Definición
-(void) drawCircleAtCenterX: (float) x
                      andY: (float) y
                      withRadius: (float) radius
                      withStrokeColor: (UIColor *) strokeColor
                      filledWithColor: (UIColor *) fillColor;

// uso
[self drawCircleAtCenterX:100.0
                      andY:100.0
                      withRadius:25.0
                      withStrokeColor:[UIColor redColor]
                      filledWithColor:[UIColor whiteColor]];
```

Nombre de los Mensajes

drawCircleAtCenterX:andY:withRadius:withStrokeColor:filledWithColor:fillColor:



OneMoreThing!

En Objective C SIEMPRE se usa el “camelCase”: la convención del camello:

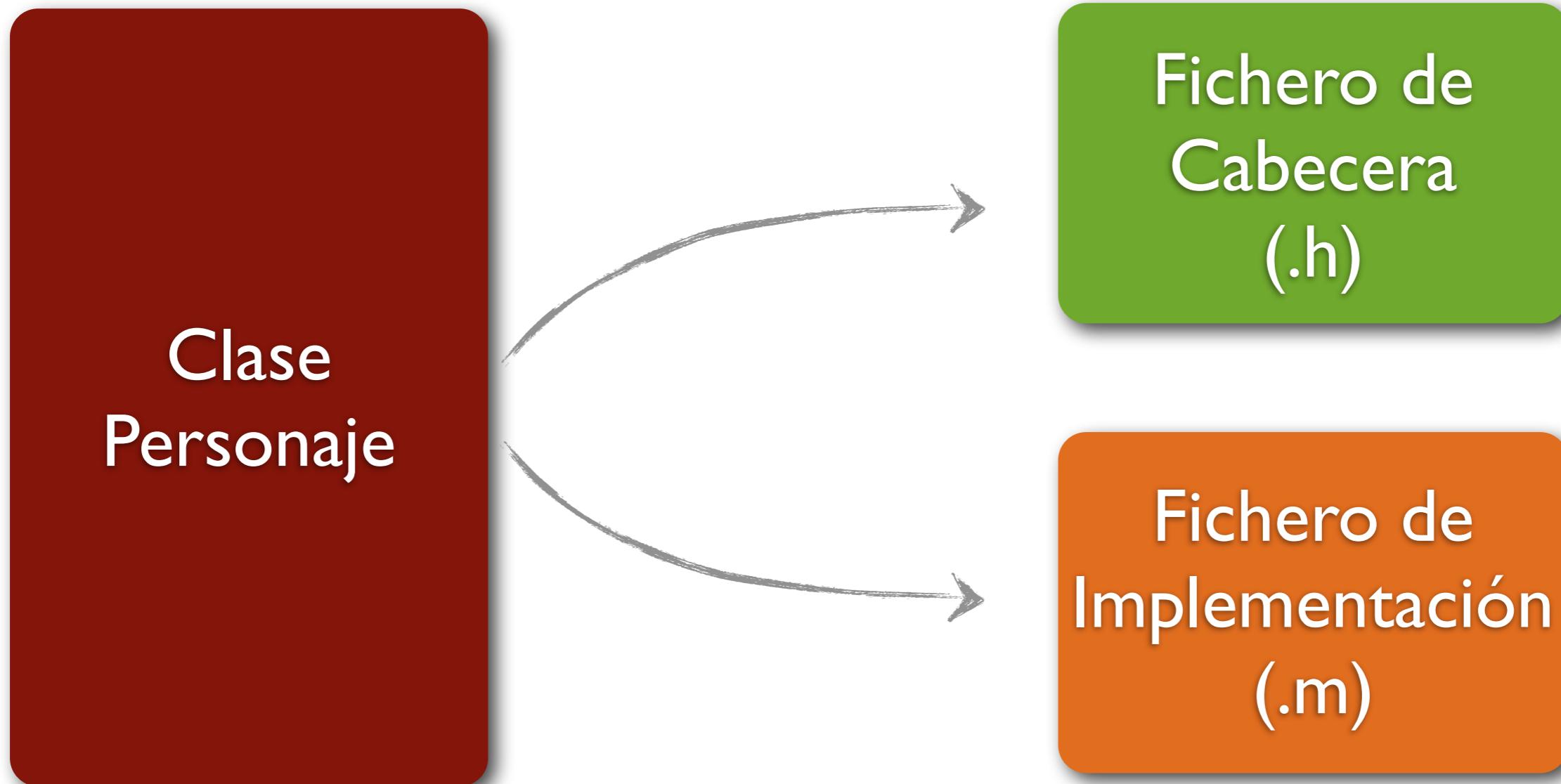


1. Los nombres se escriben siempre juntos, separando las palabras con mayúsculas.
2. Todo empieza por minúsculas, excepto el nombre de las clases.
3. El lenguaje es sensible a las mayúsculas y minúsculas

Objective C

Definición de una Clase

Definición de una Clase



Personaje.h

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

@interface Personaje : NSObject

@end
```



Personaje.m

```
#import "Personaje.h"

@implementation Personaje

@end
```



Creación de Objetos

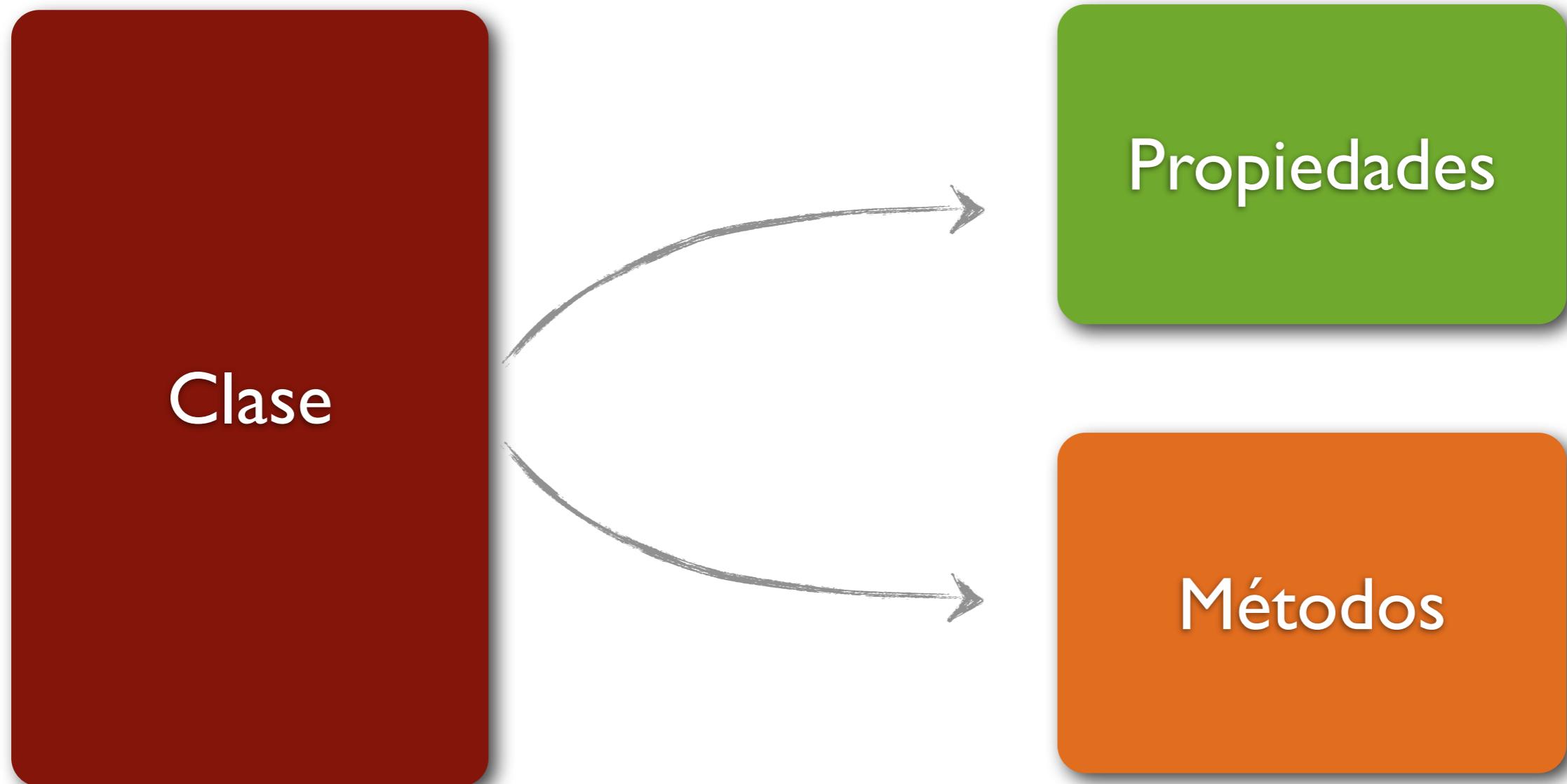
I.0

```
// anakin es un puntero a un objeto de  
// tipo personaje ¡Que no cunda el pánico!
```

```
// Para crear un objeto de tipo Personaje,  
// mandamos a la clase el mensaje new y  
// éste nos devuelve el objeto  
Personaje *anakin = [Personaje new];
```



Atributos de una Clase



Propiedades

@property

- Representan atributos estáticos de un objetos.
- Generan automáticamente la variable de instancia y los métodos accesores (getter y setter) a dicha variable.

Ejemplo de @property

Una propiedad de tipo int llamada year

```
@property int year;  
  
// Genera por nosotros una variable de  
// instancia de tipo int y llamada  
// _year.  
  
// Además, nos genera los dos métodos accesores:  
// getter  
-(int) year;  
  
// y setter:  
-(void) setYear:(int)year;
```



Setter y Getter

- El getter siempre tiene el mismo nombre que la propiedad. Nunca lleva “get” delante.
- Podemos dar información extra al compilador sobre cómo queremos que genere esos métodos.
- Podemos proporcionar nuestro propio getter o setter.



Ejemplo de Uso

Lectura y escritura de la propiedad year

```
int newYear = [self year]; // el getter (leemos un valor)  
[self setYear:1942]; // el setter (asignamos un valor)
```



Sintaxis del Punto

```
// Sintaxis habitual
int newYear = [self year]; // el getter (leemos un valor)
[self setYear:1942];       // el setter (asignamos un valor)

// Sintaxis del punto
int newYear = self.year;   // Accedo al getter (leo el valor)
self.year = 1942;          // Accedo al setter (asigno valor)
```

[] VS .





Copyright © 2012 AGBO Business Architecture S.L. Todos los derechos reservados.



Copyright © 2012 AGBO Business Architecture S.L. Todos los derechos reservados. www.agbo.biz

Creación de Objetos

alloc | init e Inicializadores Propios



El Duo Dinámico: alloc & init

Enviar new es lo mismo que enviar alloc seguido de init

```
StarWarsCharacter *darthVader = [StarWarsCharacter new];  
  
// Lo más común es hacer  
StarWarsCharacter *darthVader = [[StarWarsCharacter alloc] init];
```



Inicializadores Propios

- `initWithFirstName:lastName:age:`



Inicializadores Propios

- Siempre empiezan por initWith
- La W ha de ser mayúscula (¡Objective C es sensible a mayúsculas!)



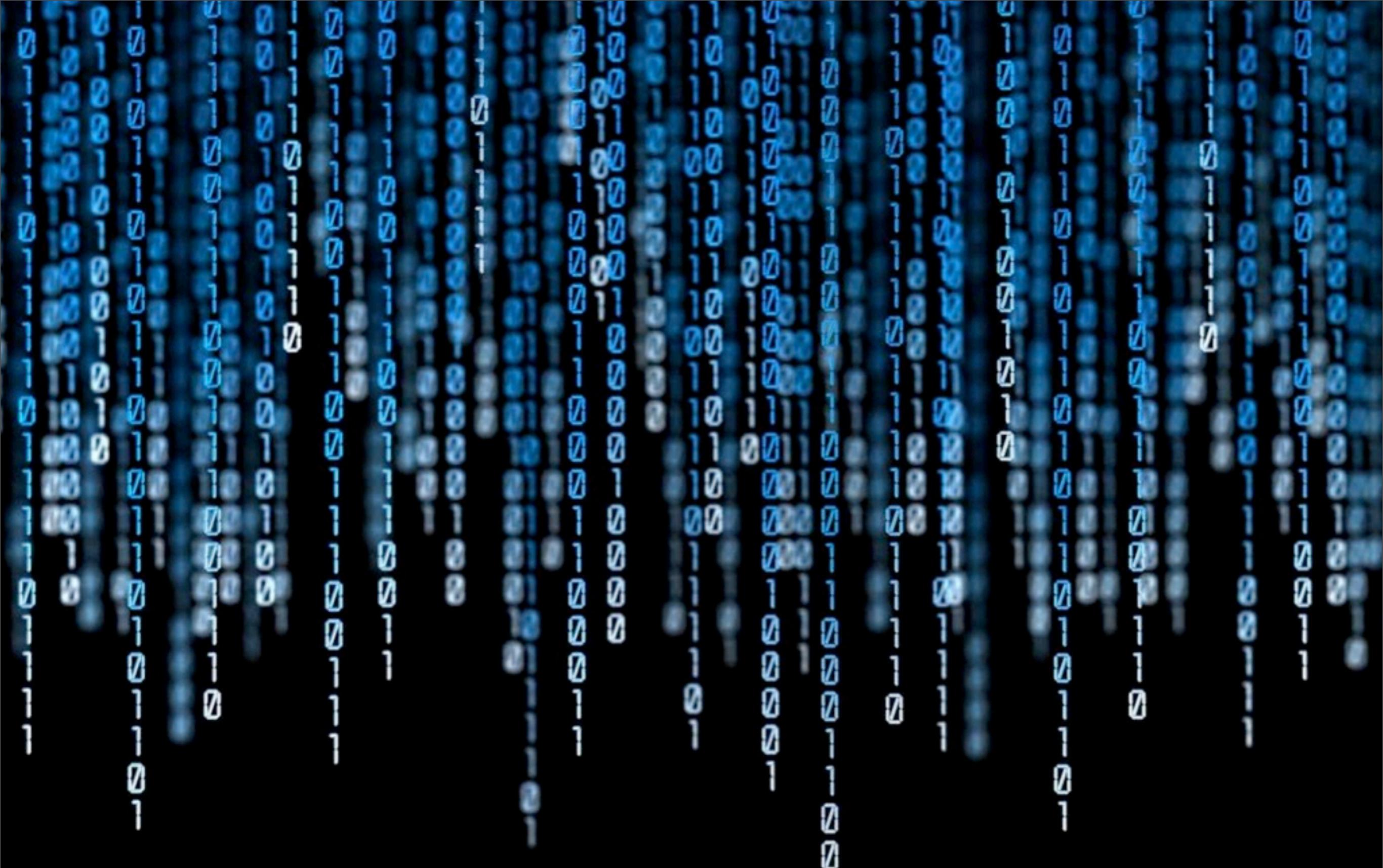


Copyright © 2012 AGBO Business Architecture S.L. Todos los derechos reservados.

Inicializadores Designado y de Conveniencia



- En Objective C, “Paco” se dice “Inicializador Designado”.
- Es el que trabaja de verdad.
- Los demás dependen de él.



Copyright © 2012 AGBO Business Architecture S.L. Todos los derechos reservados.

Constructores de Conveniencia

- Métodos de clase que hacen alloc, y initWithXXX por nosotros.
- Es la forma más común de crear instancias de clases de Cocoa.
- Su nombre siempre empieza por el nombre de la clase.

Construcción de Conveniencia

- Los de NSString empiezan todos por string: stringWithFormat: y & Cia
- Los de NSArray empiezan por array: arrayWithObjects:
- Etc



Colecciones



Colecciones en Cocoa

- Clases que contienen a otras.
- Interfaz siempre muy sencilla (ocultan la complejidad interna)
- Vienen en dos versiones: mutables e inmutables.
- **NSString, NSArray y NSDictionary** son las más comunes.



Inmutabilidad

- La mayoría de los objetos en Cocoa son *inmutables*: una vez creados ya no se pueden cambiar.
- Esto reduce el número de bugs que se pueden producir con alteraciones imprevistas en estructuras de datos.
- La mayoría de las veces no necesitamos modificar los objetos en tiempo de ejecución.



NSString

- Inmutable
- Se crea con constructores de conveniencia como stringWithFormat:
- Es una lista de caracteres UNICODE
- Representación literal:@”Hola Mundo”



NSMutableString

- Versión mutable.
- Se suele crear a partir de una inmutable con `mutableCopy`
- Tiene métodos para alterar el contenido, como `appendString:` y otros.



Usos Habituales

```
NSString *persona = @"Fernando";  
  
// Concatenado: Se "concatena" con stringWithFormat:  
  
NSString *concatenado = [NSString stringWithFormat:  
                        @"iHola %@", persona];  
NSLog(@"%@", concatenado);  
  
// Conversión Número / Cadena  
NSString *precio = @"29.99";  
float p = [precio floatValue];  
  
// Conversión a mutable  
NSMutableString *mut = [@"Hola" mutableCopy];  
[mut appendString:@" mundo"];
```

NSArray

- Acepta cualquier objeto (no es tipado).
- Sólo acepta objetos.
- Para guardar números (no objetos) usamos la clase `NSNumber` que empaqueta a los números.
- Para guardar nil, usamos `NSNull` (similar a `NSNumber`).

Literales para NSArray

```
// Creamos un NSArray de la forma estándar.  
// El nil final indica que hemos terminado de meter objetos  
NSArray *lista = [NSArray arrayWithObjects:  
    @"uno",  
    @"dos",  
    [NSNumber numberWithInt:344499],  
    nil];  
  
// Mediante literales  
NSArray *lista = @[@"uno", @"dos", @344499];
```

Iteración sobre un NSArray

```
// Estilo C
for (int i = 0; i < lista.count; i++) {
    NSLog(@"El objeto en %d es %@", i,
          [lista objectAtIndex:i]);
}

// Estilo "foreach"
for (id element in lista)
    NSLog(@"El objeto es %@", element);
}

// La más cómoda: con bloques.
// La veremos luego. Paciencia, joven padawan.
```

NSMutableArray

- Versión mutable.
- Se suele crear a partir de una inmutable con `mutableCopy`
- Tiene métodos para alterar el contenido, como `insertObject:atIndex:` y otros.

Usos Habituales

```
NSMutableArray *vector = [@[@"uno", @"dos", @344499]mutableCopy];  
[vector addObject:@42]; // se añade al final  
[vector insertObject:@[@"Last Samurai",@"Inception"]  
 atIndex:0]; // los índices empiezan en 0  
[vector removeLastObject];
```

NSDictionary

- Almacena parejas de objetos: una clave y un valor.
- Similar a un map en C++ o un hashmap en Java.
- Normalmente la clave es un **NSString**, pero puede ser cualquier objeto.



Literales para NSDictionary

```
// Estilo antiguo
NSDictionary *dict = [NSDictionary dictionaryWithObjectsAndKeys:
    @{@"uno", [NSNumber numberWithLong:1]},
    @{@"primos", @[@2, @3, @5, @7]},
    nil];
```



```
// Nuevo
NSDictionary *d = @{@"Zimmer" : @"Inception",
    @"Williams" : @"Star Wars"};
```

Iteración por NSDictionary

```
// Antiguo
NSArray *keys = [d allKeys];
for (int i = 0; i < keys.count; i++) {
    id key = [keys objectAtIndex:i];
    NSLog(@"La clave %@ se corresponde con el objeto %@", key, [d objectForKey:key]);
}

// Nuevo
for (id key in d) {
    NSLog(@"El clave %@ se corresponde con el objeto %@", key, [d objectForKey:key]);
}

// Más cómodo: con bloques
// Lo veremos luego.
```



Usos Comunes

```
NSMutableDictionary *d = @{@"Zimmer" : @"Inception",
                           @"Williams" : @"Star Wars"}
                           mutableCopy];

// Añadir
[d setObject:@"Sakamoto" forKey:@"Black Rain"];

// Recuperar
id value = [d objectForKey:@"Black Rain"];

// Comprobar existencia
// No hay un método específico. Se intenta
// recuperar y si no está, nos devuelve nil (falso)
if ([d objectForKey:@"Barry"] == nil) {
    NSLog(@"No soundtracks from John Barry");
}
```





Copyright © 2012 AGBO Business Architecture S.L. Todos los derechos reservados.

Introducción a UIKit

MVC, Target / Action, Delegate, Notifications



Qué vamos a aprender

- Organizar nuestras clases según el patrón MVC.
- Usar el patrón del delegate para comunicar objetos.
- Usar el patrón de target/action para responder a eventos
- Crear la interfaz gráfica



MVC

Controlador

Modelo

Vista



MVC

Controlador

Repartimos todos nuestras clases en 3 equipos

Modelo

Vista



MVC

Controlador

Modelo: el alma de tu aplicación

Modelo

Vista



MVC

Controlador

Controlador: Presenta el modelo al usuario

Modelo

Vista



MVC

Controlador

Vista: la tropa de esclavos que usa el controlador

Modelo

Vista



Ejemplo de Vistas

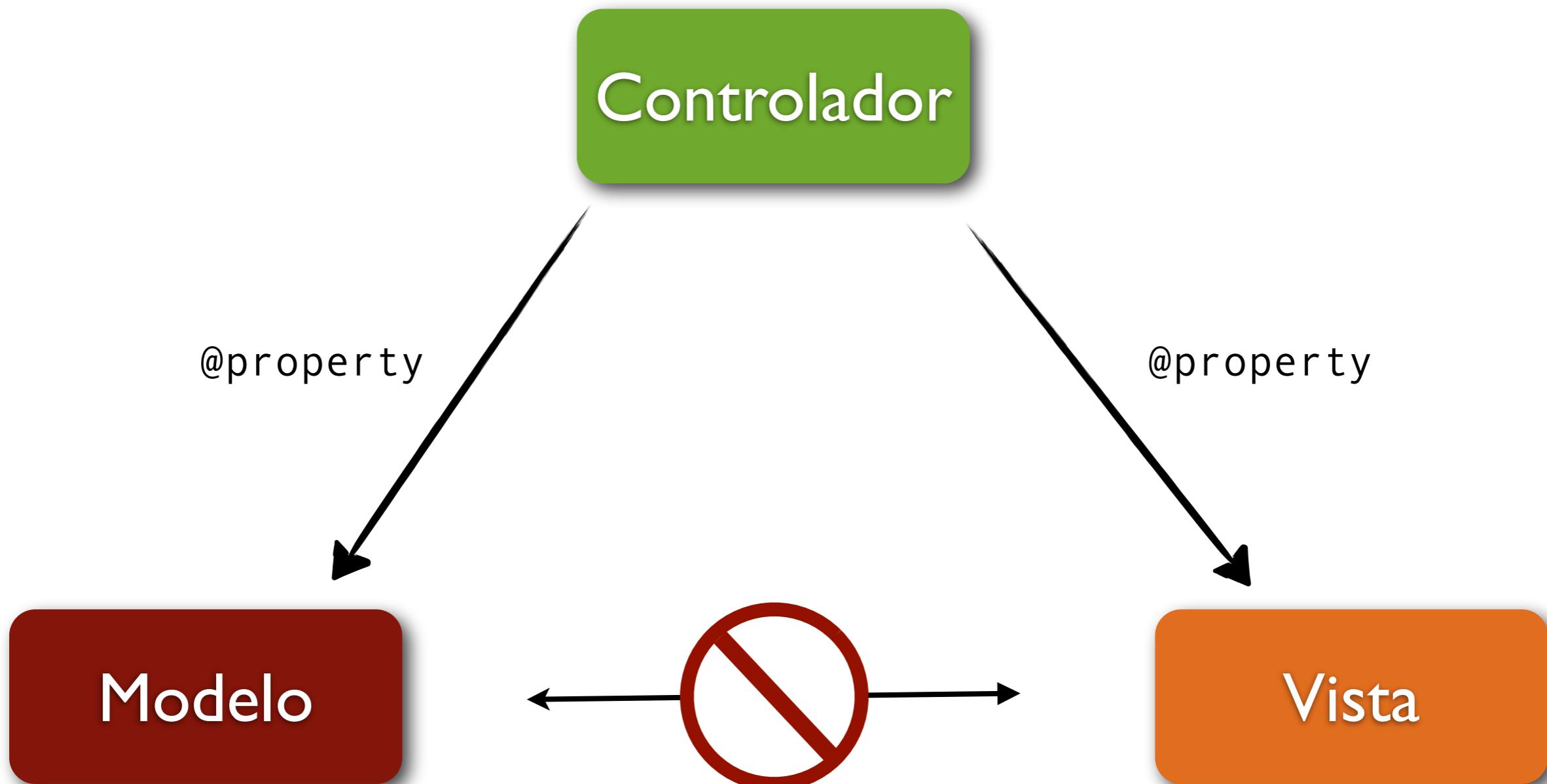


Ejemplo de Vistas



6 vistas en total

Comunicación entre los equipos



Comunicación Vista Controlador

Target Action & Delegate



Comunicación Vista Controlador

- ¿Puede la vista tener algo que decirle al controlador?



Comunicación Vista Controlador

- Son un botón (UIButton) y han hecho clic sobre mi.
- Soy una UIWebView y han hecho clic sobre un enlace. ¿Lo puedo cargar?
- Soy una UIWebView y voy a cargar un enlace.
- Soy una UIWebView y acabo de cargar un enlace. Que lo sepas.



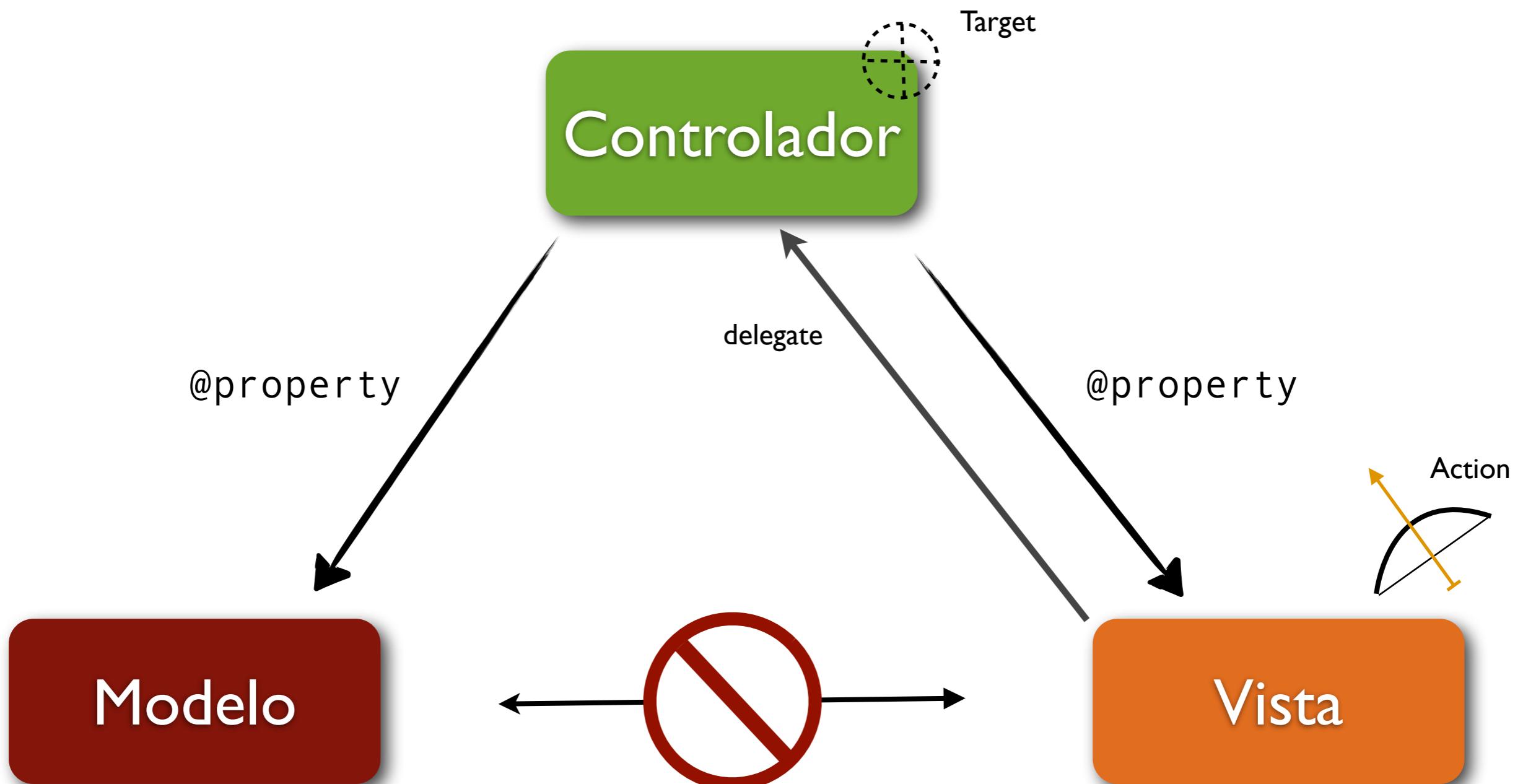
Comunicación Vista Controlador

Hay dos formas de comunicarse:

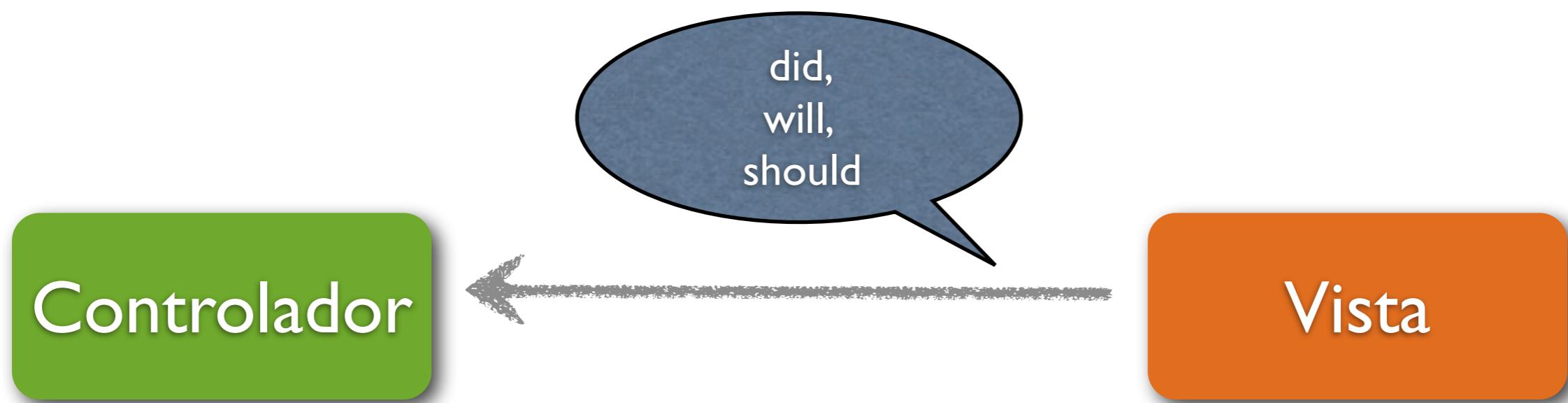
- Target/Action: cuando ocurra algo interesante, al objeto Target le envías el mensaje Action .
- Delegate: cuando necesites más información sobre cómo dibujarte o reaccionar ante las acciones del usuario, me pides más información.



Comunicación entre los equipos



Delegate



El controlador es el delegado (ayudante) de la vista.

La vista envía mensajes informando de lo que hace y pidiendo ayuda cuando hace falta.

Ejemplo de Delegate

(1) webView:shouldStartLoadWithRequest:navigationType:

Controlador

(2) YES

(3) webViewDidFinishLoad:



Comunicación Modelo Controlador

¿Puede el modelo tener algo que decir al controlador?



Comunicación Modelo Controlador

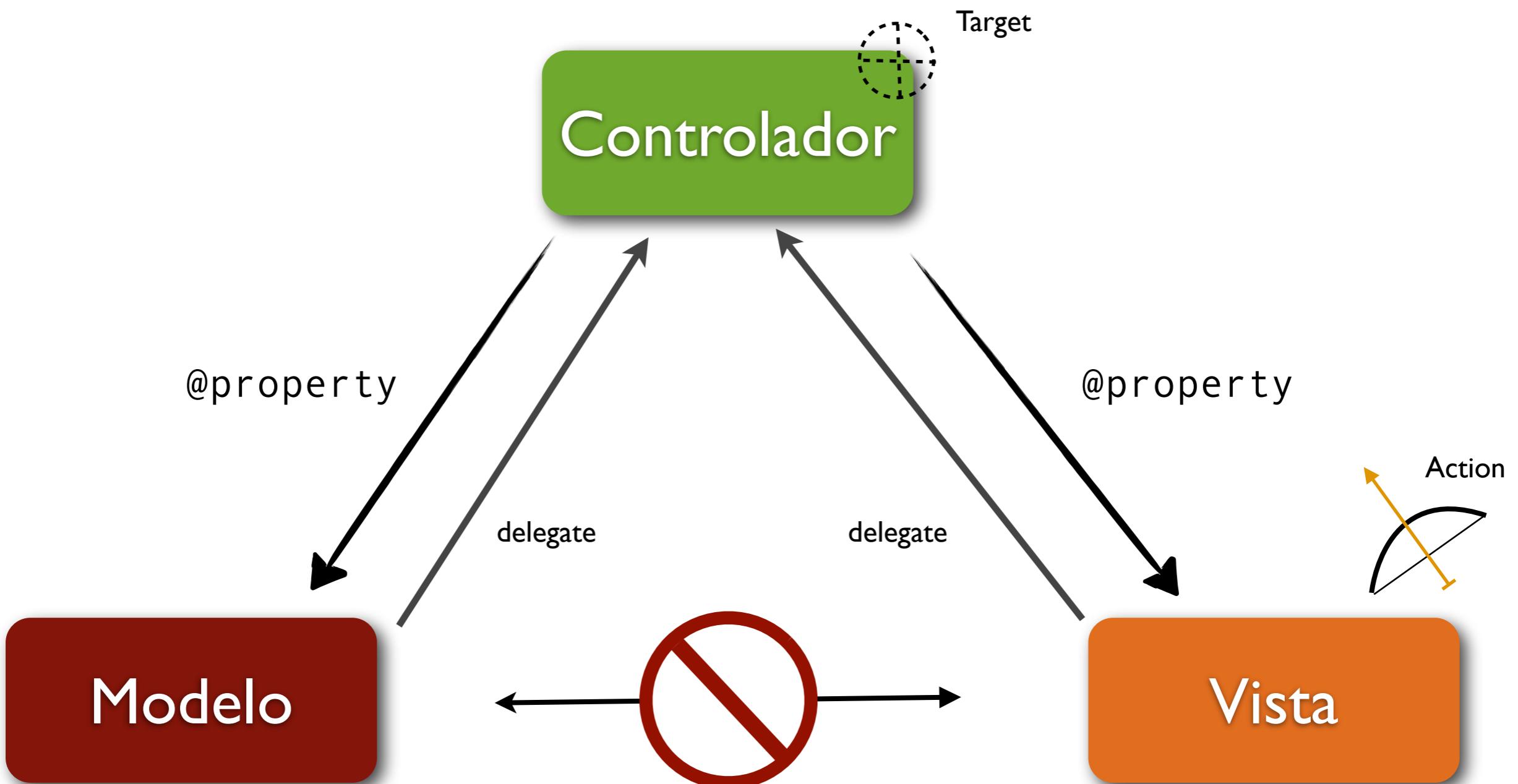
- Toda modificación que sufra el modelo debe de ser notificada al controlador.
- Por ejemplo, en un programa de bolsa, si el valor de la acción cambia, tendrá que notificar al controlador.

Comunicación Modelo Controlador

- La forma más común de comunicación modelo-controlador es mediante notificaciones. Las veremos más adelante.
- De momento, usaremos el delegado, es decir, el controlador será delegado del modelo.



Resumen MVC



Qué hemos aprendido

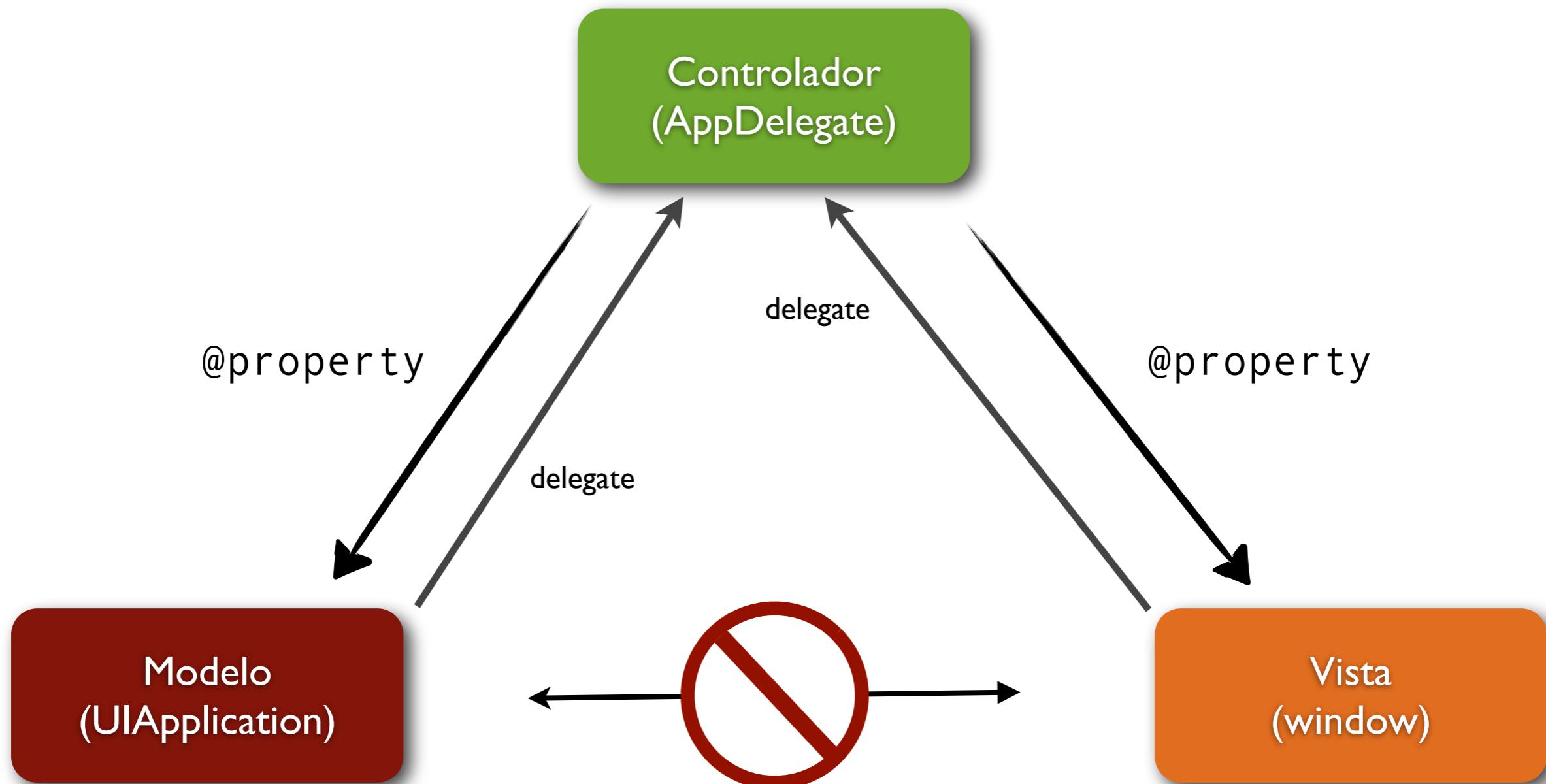
- Dividir nuestras clases según el patrón del MVC.
- Cuando usar el delegate y el target/action para comunicar objetos





Copyright © 2012 AGBO Business Architecture S.L. Todos los derechos reservados.

Empty Application





Copyright © 2012 AGBO Business Architecture S.L. Todos los derechos reservados.

Los II mandamientos de las propiedades (β 1)

- ◆ I: Para los no-objetos: @property (nonatomic)
- ◆ II: Para los objetos: @property (nonatomic, strong)





Copyright © 2012 AGBO Business Architecture S.L. Todos los derechos reservados.

Los III mandamientos de las propiedades (β 2)

- ◆ I: Para los no-objetos:
@property (nonatomic)
- ◆ II: Para los IBOutlets:
@property (weak, nonatomic)
- ◆ III: Para los objetos:
@property (strong, nonatomic)





Copyright © 2012 AGBO Business Architecture S.L. Todos los derechos reservados.

Qué hemos aprendido

- Cómo arranca una app
- Los principales ficheros de una app
- Crear varios MVCs
- Crear inicializadores propios
- Responder a eventos
- Usar el delegate
- Usar Target Action





Copyright © 2012 AGBO Business Architecture S.L. Todos los derechos reservados. www.agbo.biz

Combinar Distintos MVCs

Combinadores



Complejidad Manteniendo la Sencillez

Controlling complexity is the essence of computer programming.
Brian Kernigan



Sofá Mah Jong de Hans Hopfer para Roche Bobois

Combinando Elementos Sencillos



Sofá Mah Jong de Hans Hopfer para Roche Bobois



Combinadores de MVCs

- Hay varios y el funcionamiento suele ser similar.
- UITabBarController
- UINavigationController
- UISplitViewController
- UITableViewcontroller



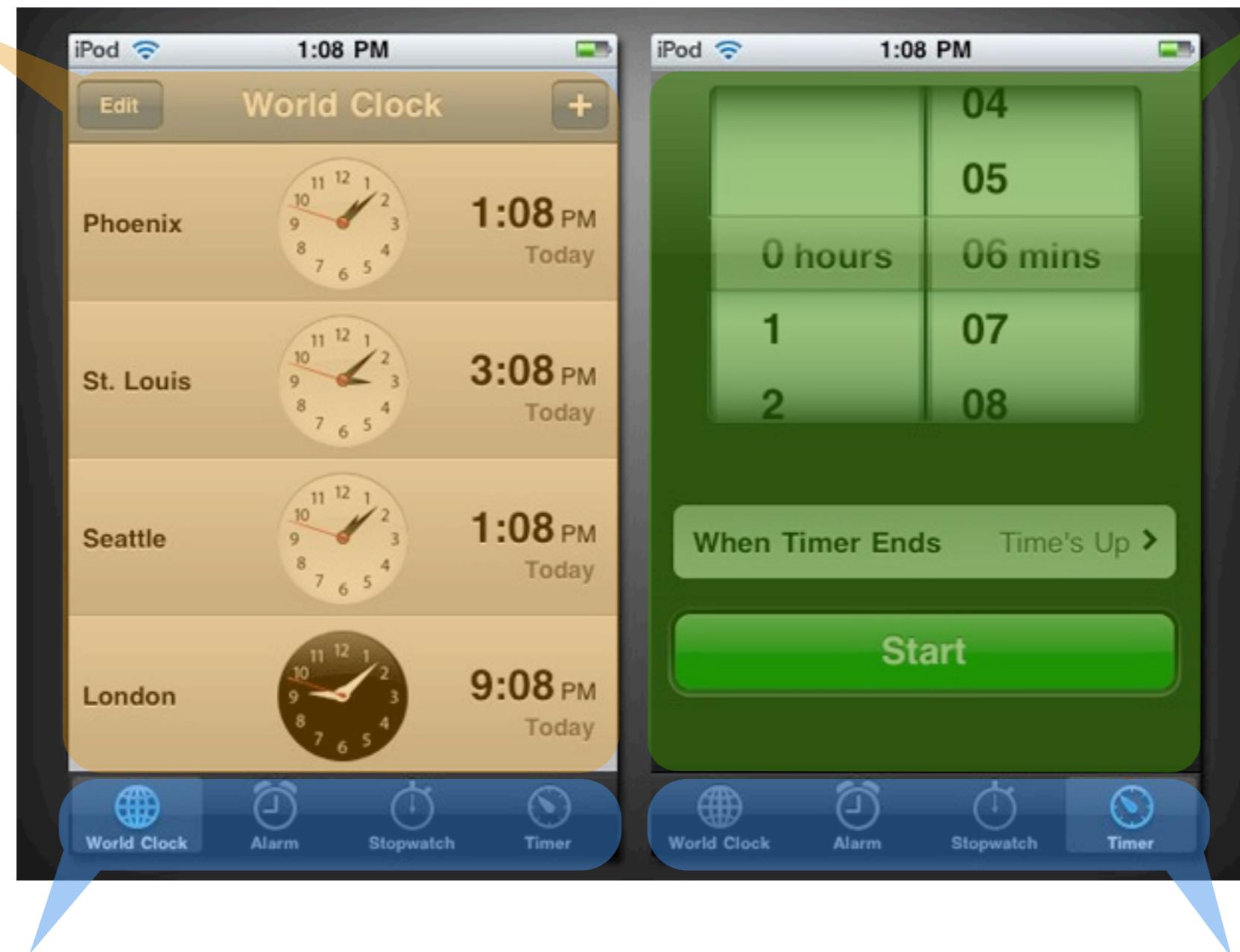
UITabBarController



UITabBarController

Un controlador con su vista y su modelo

Otro controlador con su vista y su modelo



El TabBar contiene a distintos MVCs y nos permite verlos de uno en uno mediante la barra de botones inferior.

Copyright © 2012 AGBO Business Architecture S.L. Todos los derechos reservados. www.agbo.biz

¿De donde salen los botones?



¿Cual es tu tabBarItem?
y si es nil,
¿Cual es tu title?

Uso de UITabBarController

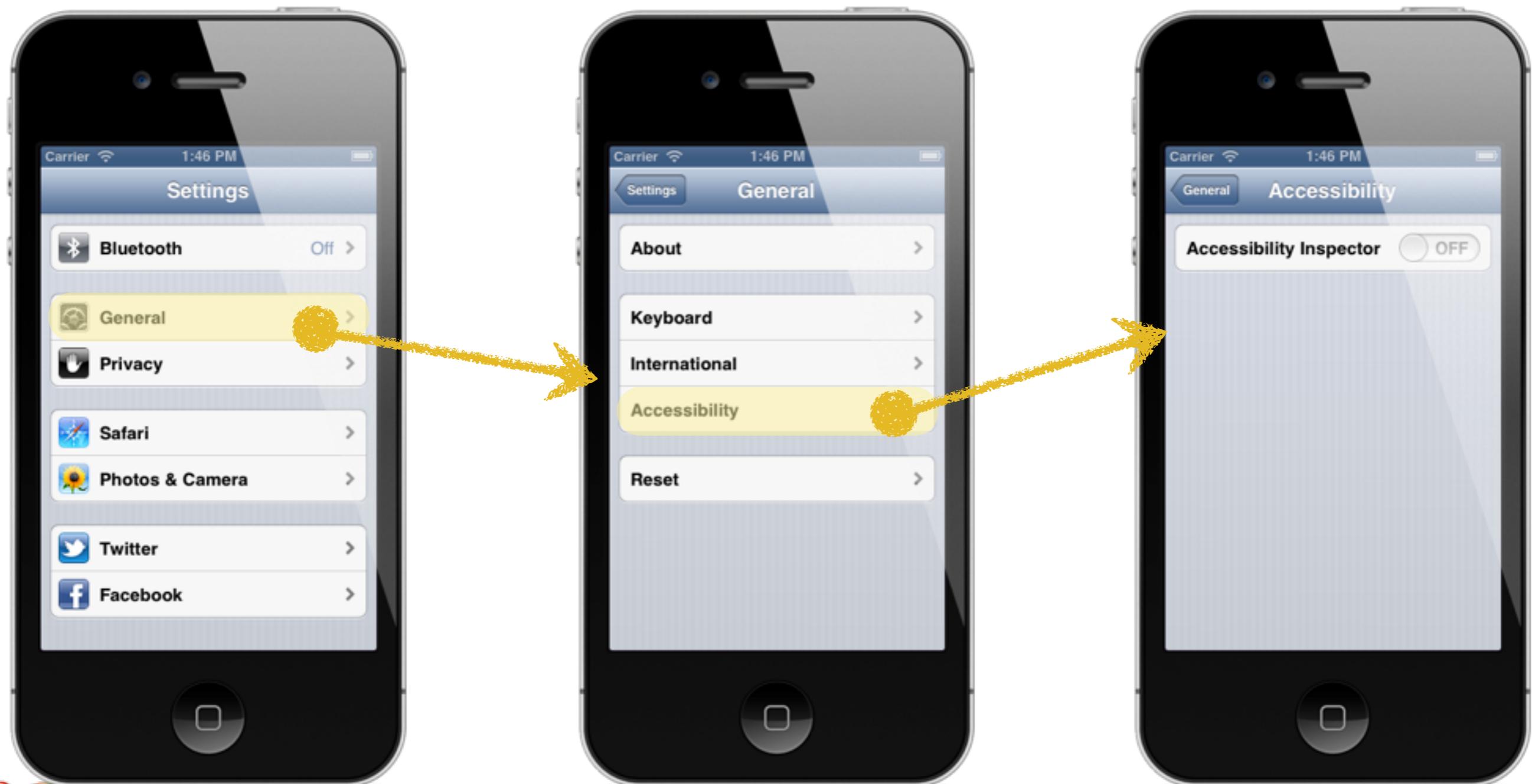
Se crea con alloc/init y se asigna un array de
UIViewController's a su propiedad viewControllers

```
// Creamos el TabBar
UITabBarController *tabVC = [[UITabBarController alloc] init];

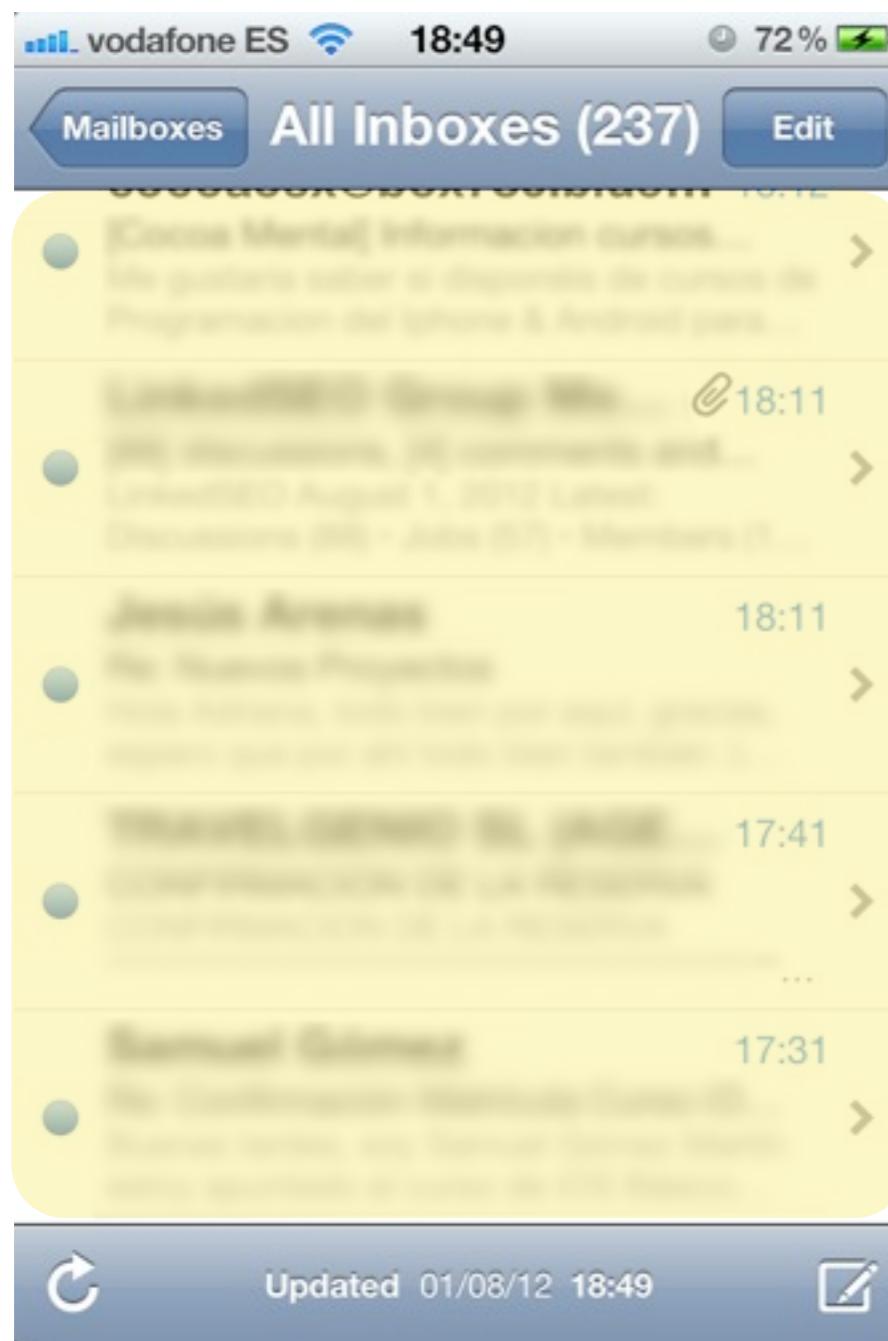
// Asignamos el array de controladores que ha de combinar
[tabVC setViewControllers:@[worldClockVC,
                           alarmVC,
                           stopWatchVC,
                           timerVC ]];
```



UINavigationController

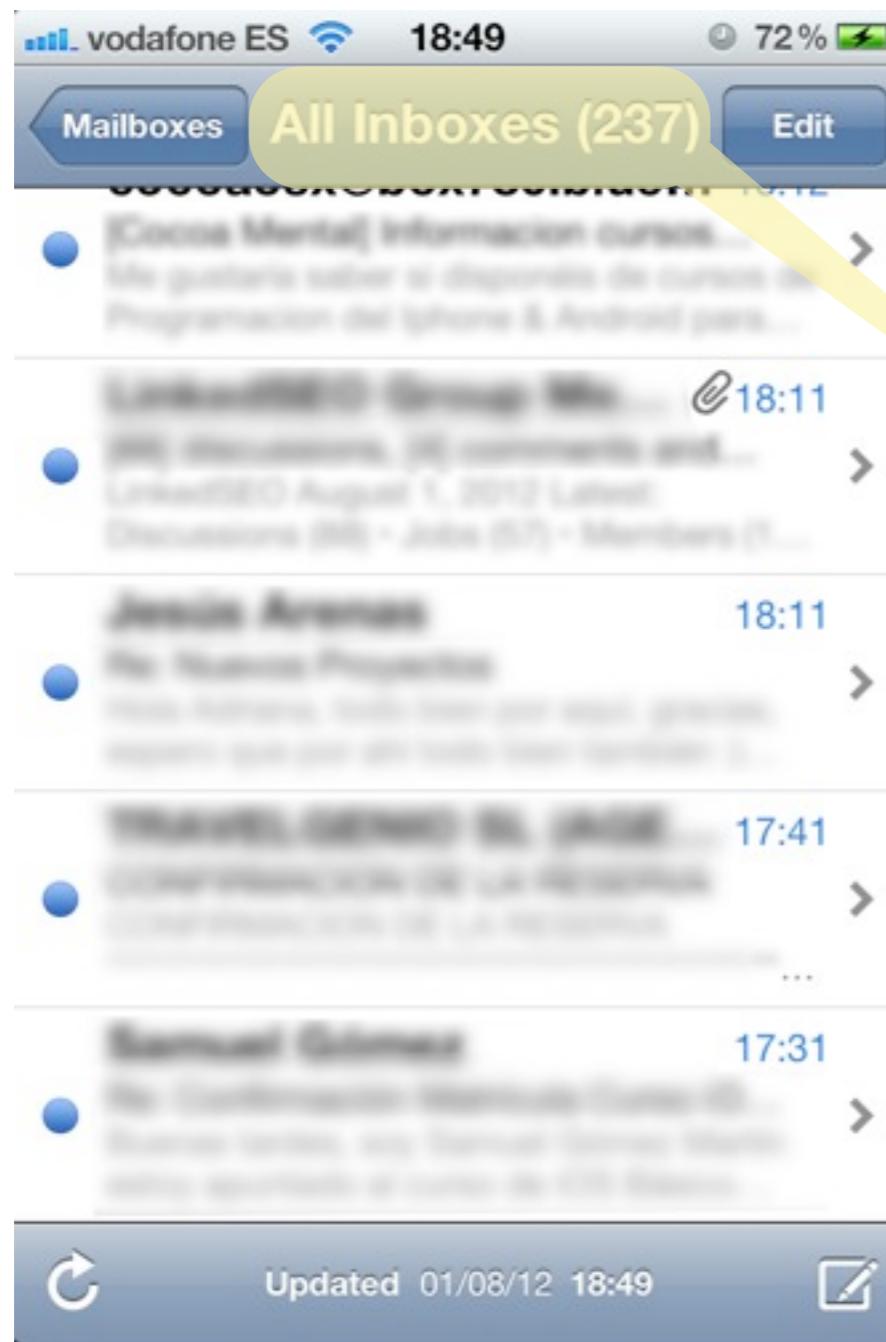


Anatomía de un UINavigationController



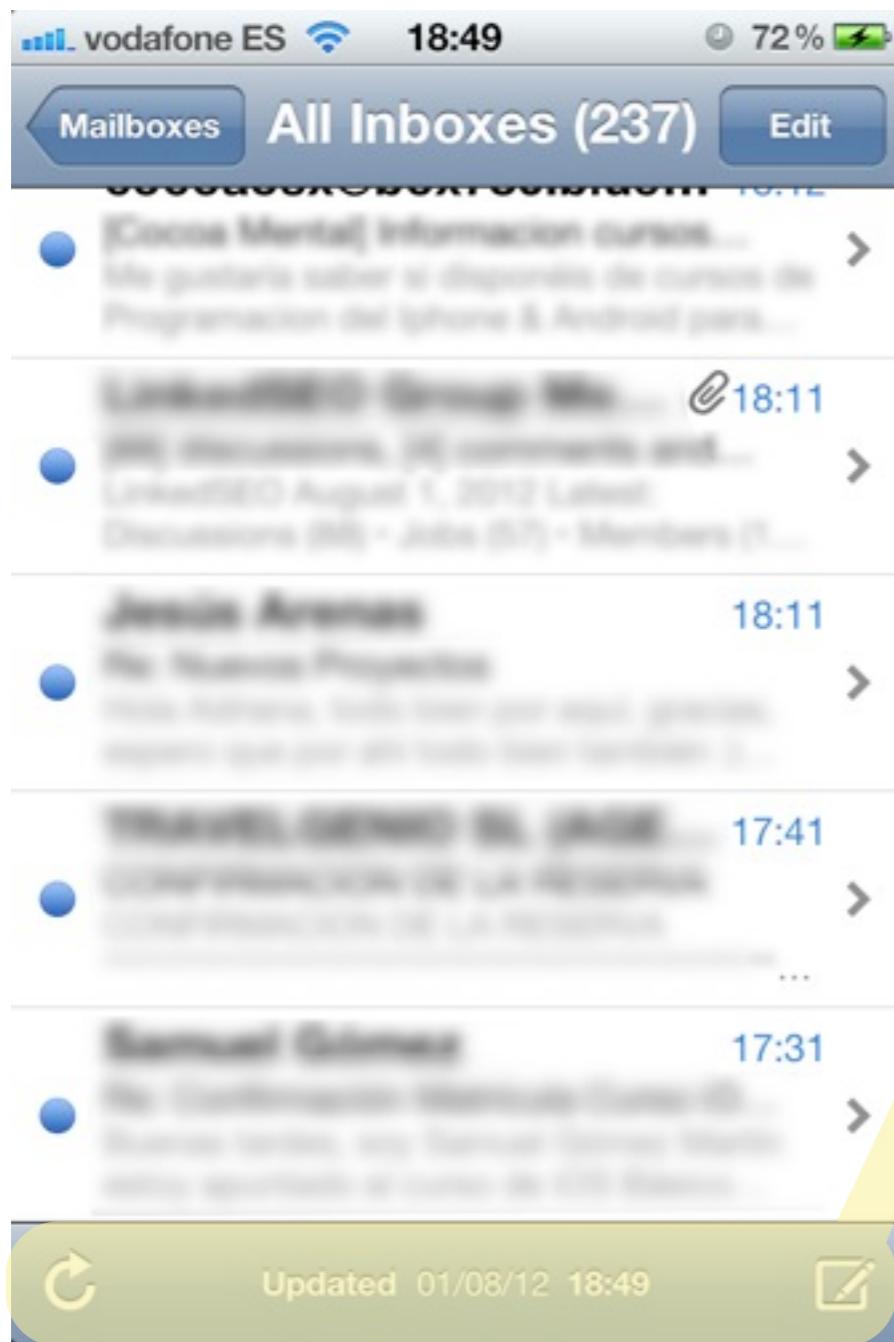
Vista del MVC superior

Anatomía de un UINavigationController



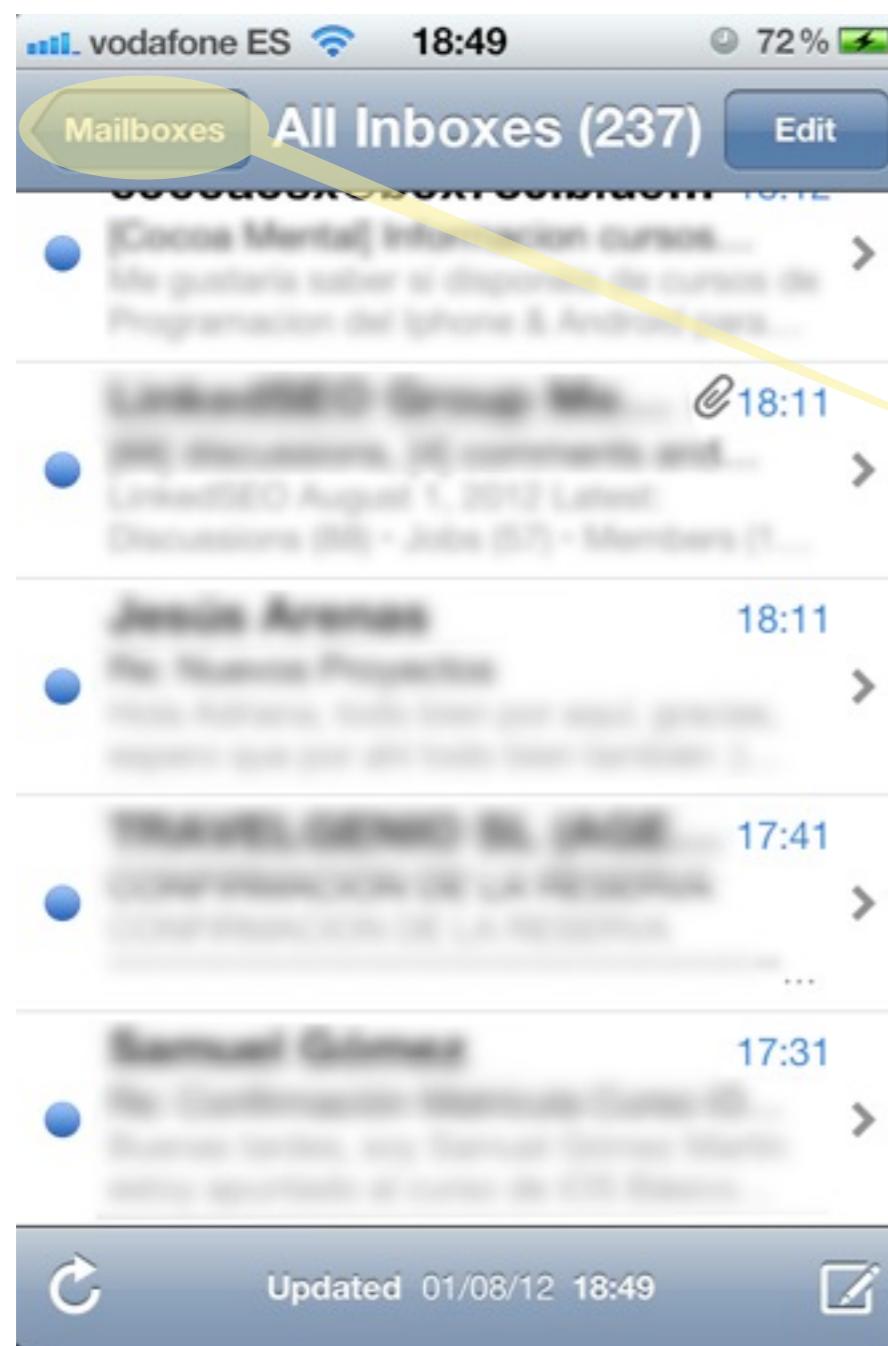
NSString obtenida de la propiedad title del controlador superior.

Anatomía de un UINavigationController



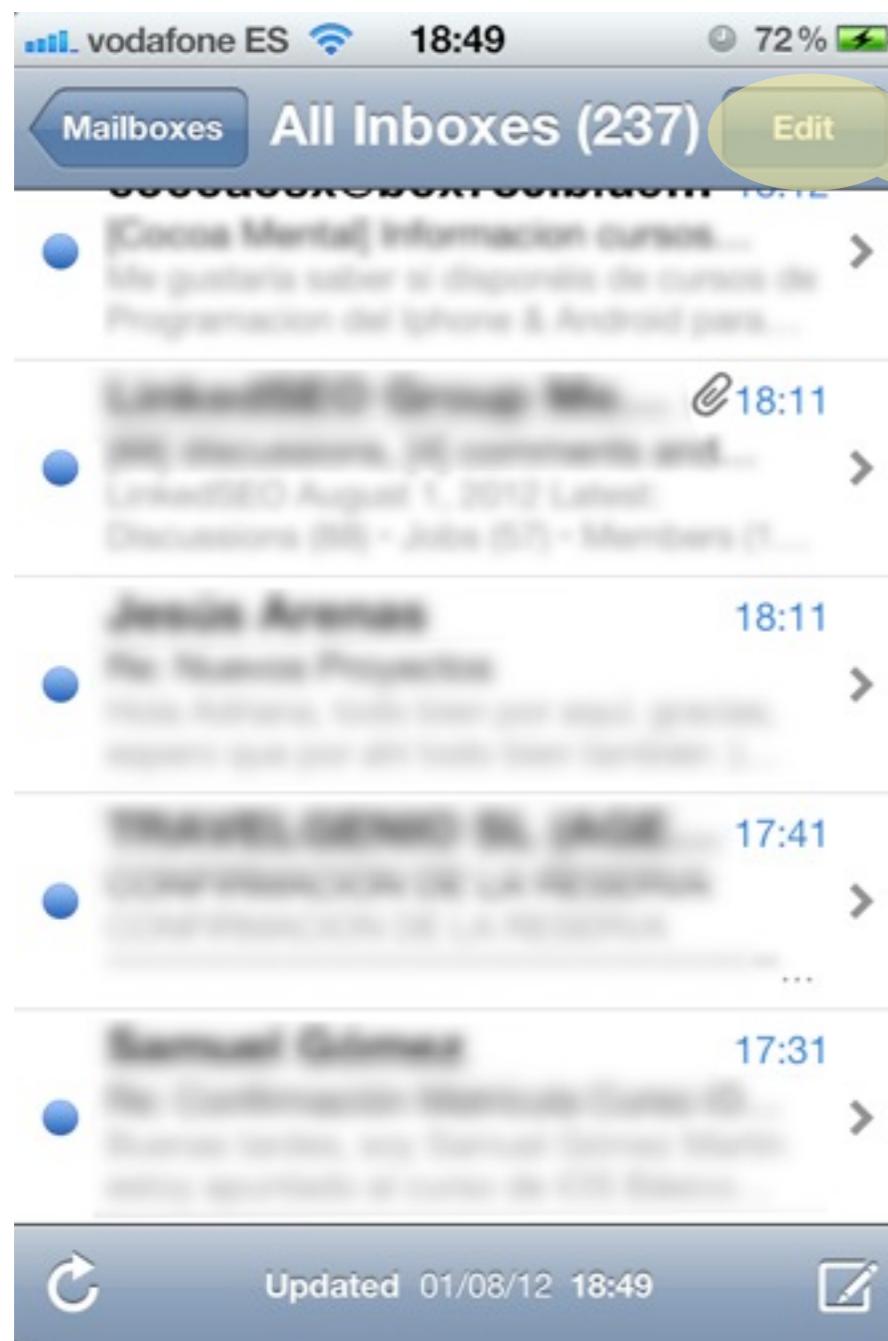
NSArray de UIBarButtonItem
obtenido de la propiedad
toolbarItems del controlador
superior.

Anatomía de un UINavigationController



NSString obtenida de la
propiedad title del controlador
inmediatamente por debajo.

Anatomía de un UINavigationController



Podemos añadir botones a la izquierda, derecha e incluso el centro de la barra de navegación.

Uso de un UINavigationController

- Se crea con alloc / init o alloc / initWithRootViewController:
- pushViewController:animated:
- popViewControllerAnimated:
- popToRootViewControllerAnimated:

```
UINavigationController *navVC = [[UINavigationController alloc]  
initWithRootViewController:wineVC];
```



Configuración

- Los controladores que están dentro de un Navigation, lo pueden configurar
- navigationController
- navigationItem





Copyright © 2012 AGBO Business Architecture S.L. Todos los derechos reservados.

TabBar como Combinador

- Los dos controladores tienen relación jerárquica.
- Deberíamos estar usando un **UINavigationController**





Copyright © 2012 AGBO Business Architecture S.L. Todos los derechos reservados.

UINavigationController como Combinador

- Necesitamos poder mostrar más de un modelo.



Combinadores dentro de Combinadores



- Necesitamos un NavigationController que contiene a un CharacterController para cada personaje.
- Todos esos NavigationControllers estarán dentro de un TabBarController.

Combinadores dentro de Combinadores

UITabBarController

NavigationVC

CharacterVC

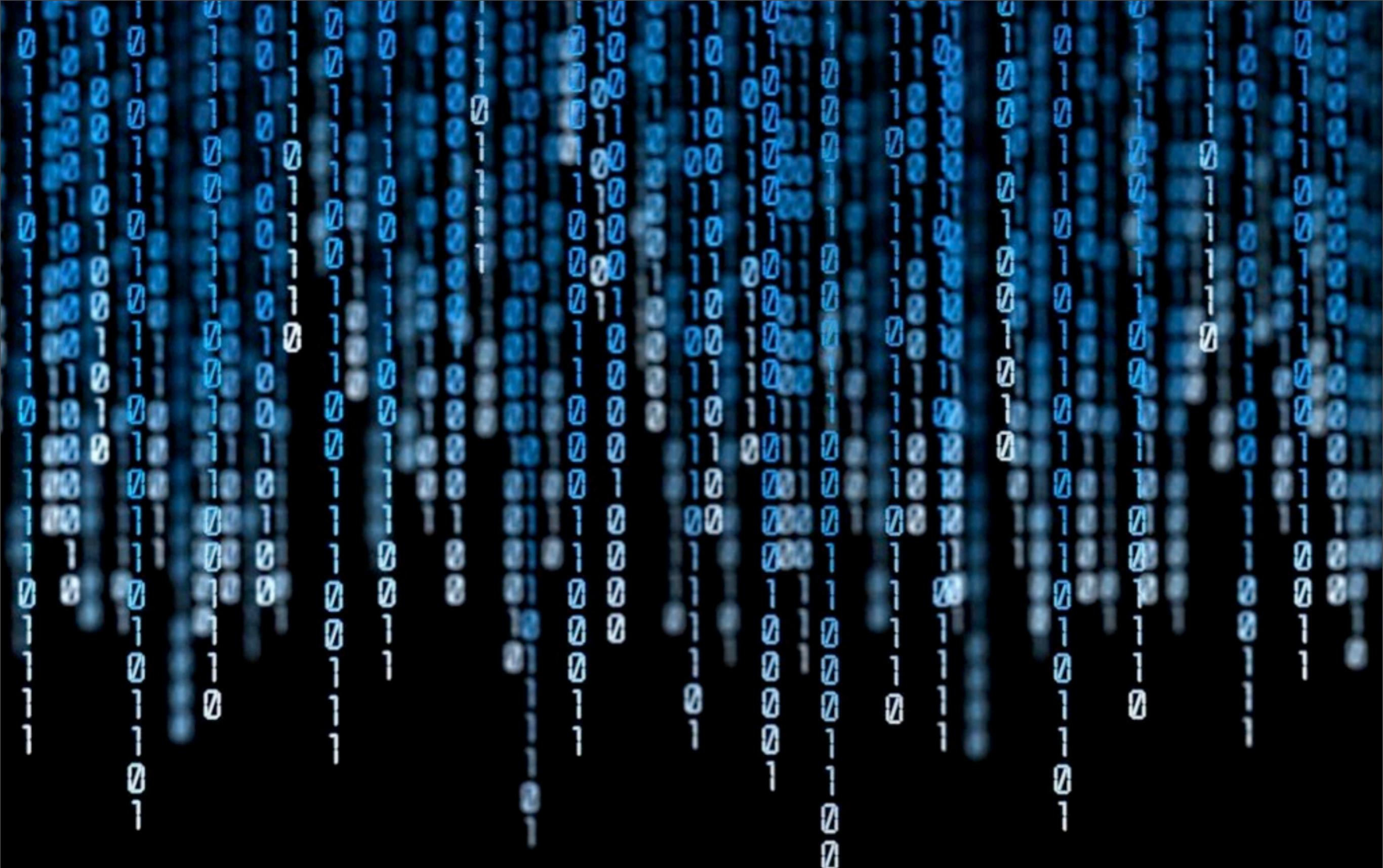
NavigationVC

CharacterVC

NavigationVC

CharacterVC





Copyright © 2012 AGBO Business Architecture S.L. Todos los derechos reservados.



Copyright © 2012 AGBO Business Architecture S.L. Todos los derechos reservados. www.agbo.biz

Interfaz para una APP de iPad

UISplitViewController & UITableViewcontroller

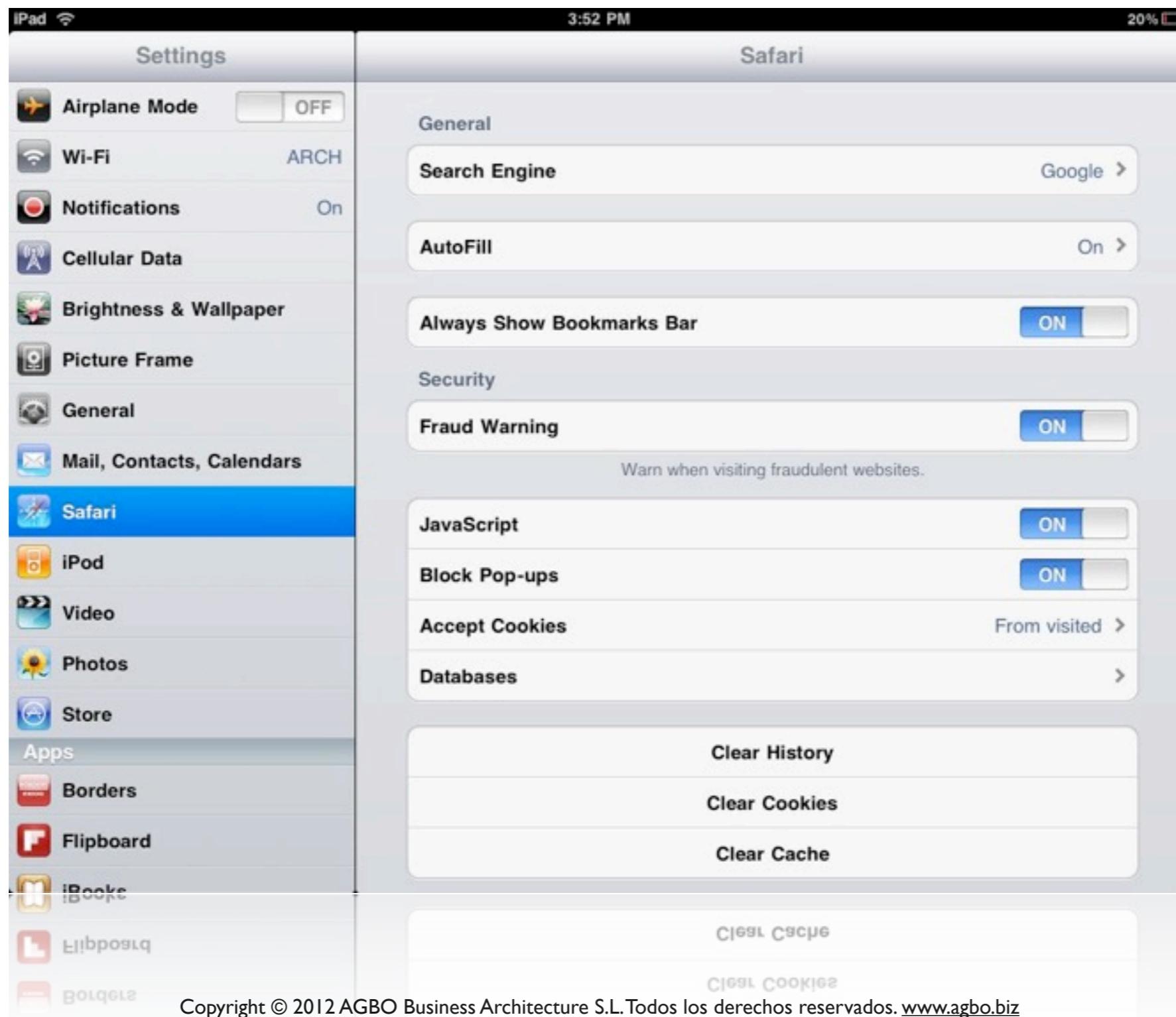


Introducción a UISplitViewController

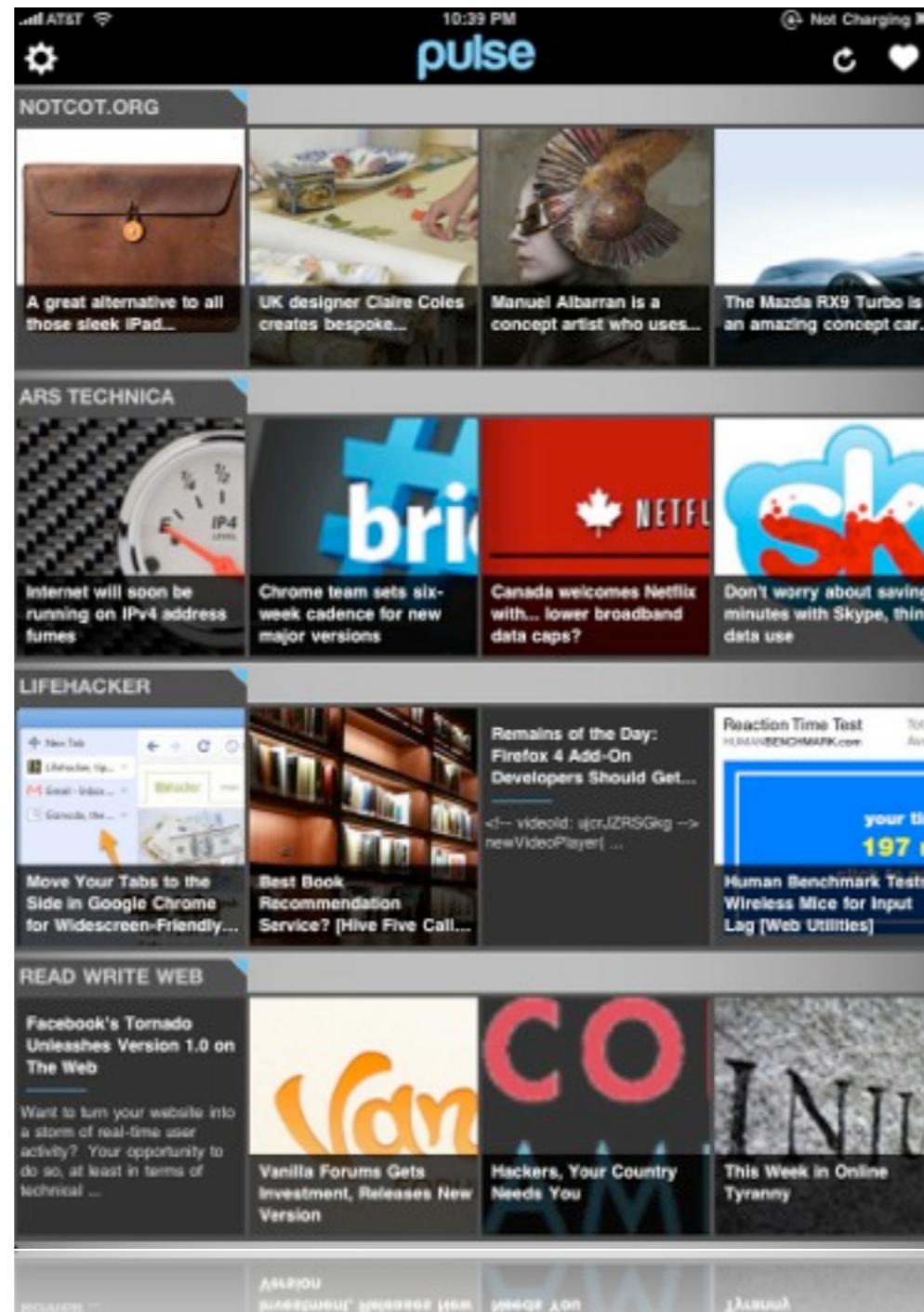
Específico del iPad



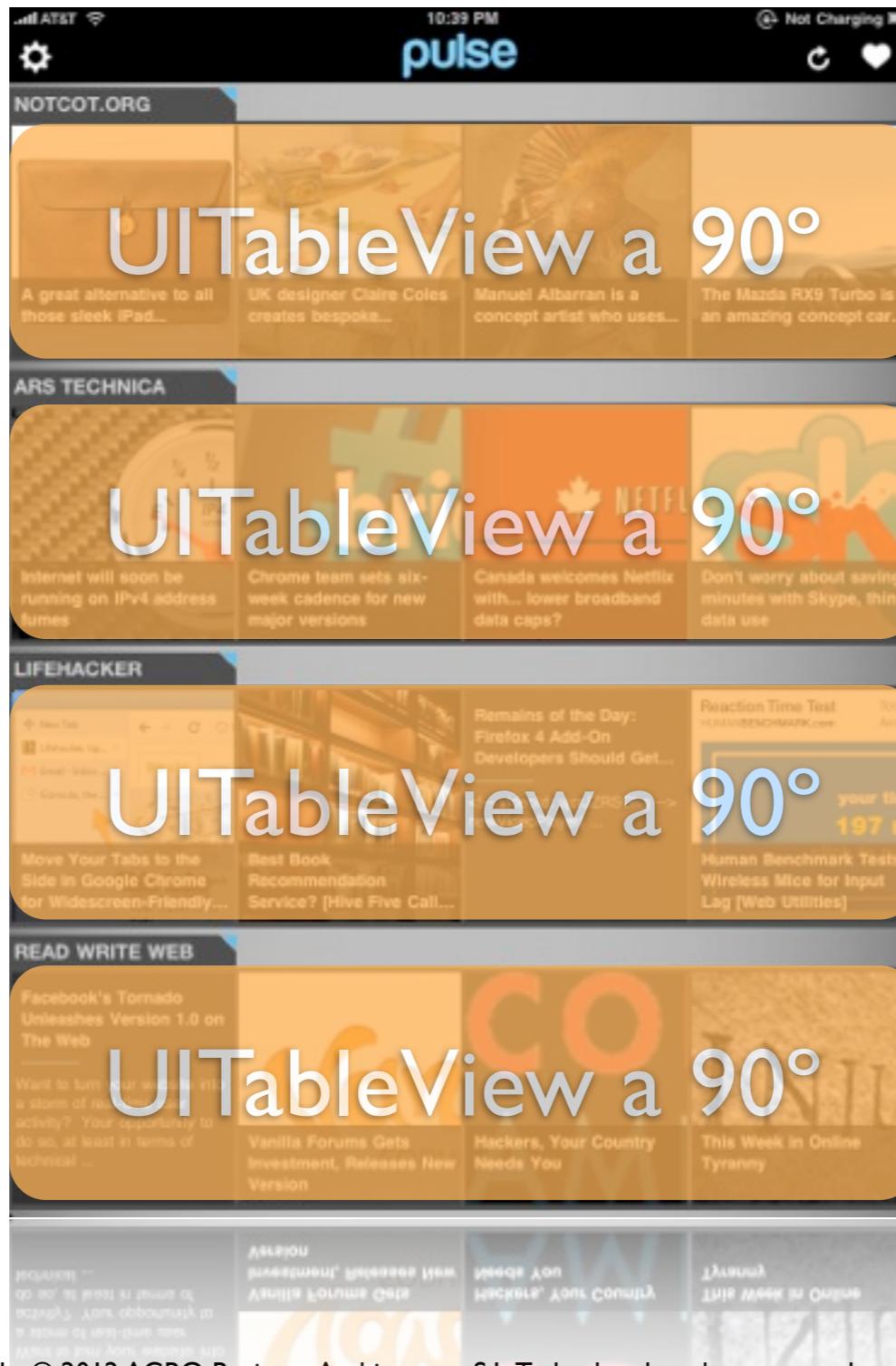
UITableViewController



UITableViewController



UITableViewController



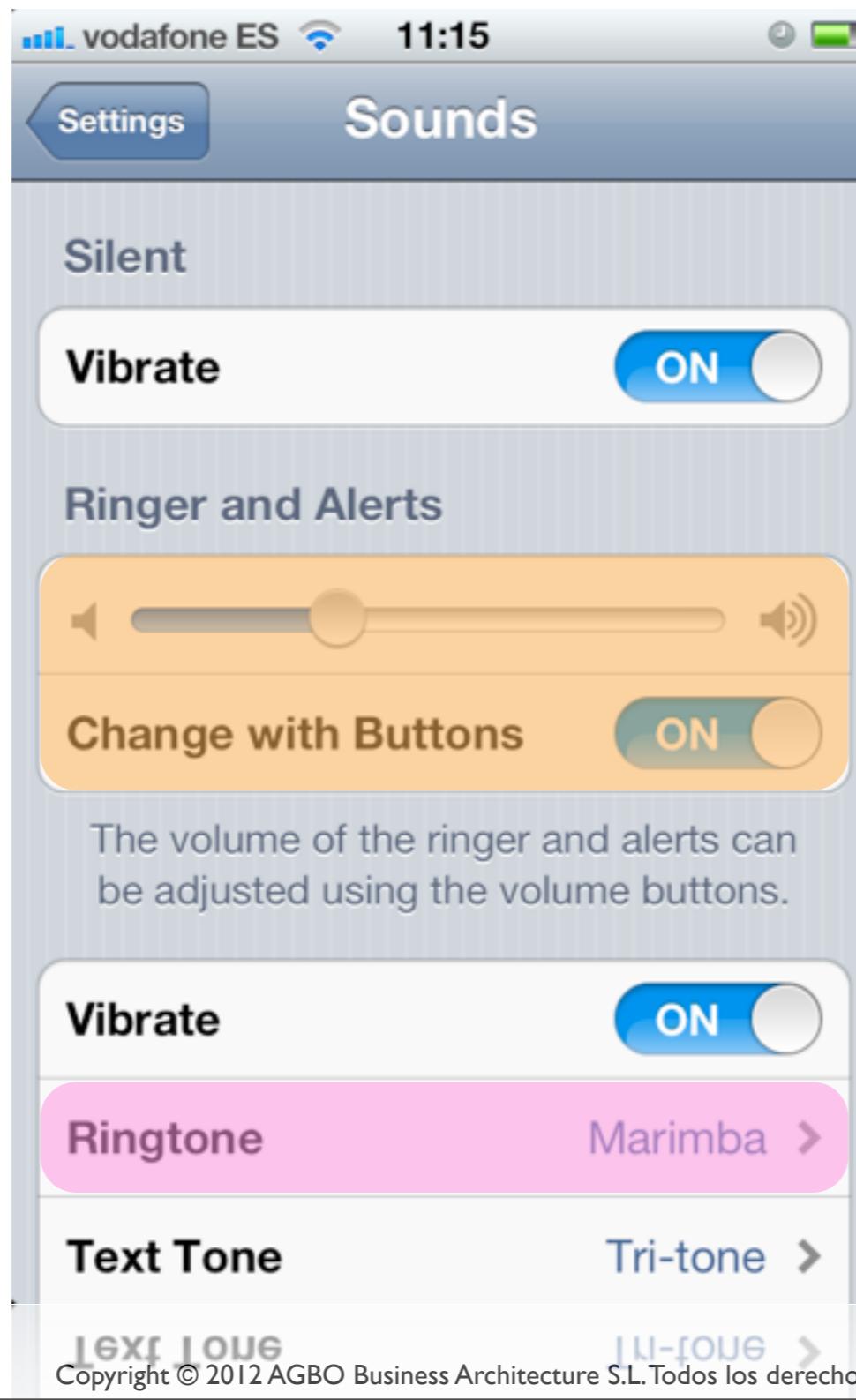
UITableViewController

- UITableViewDelegate (will, did, should)
- UITableViewDataSource (nos pide los datos)



Anatomía de una tabla

Sección



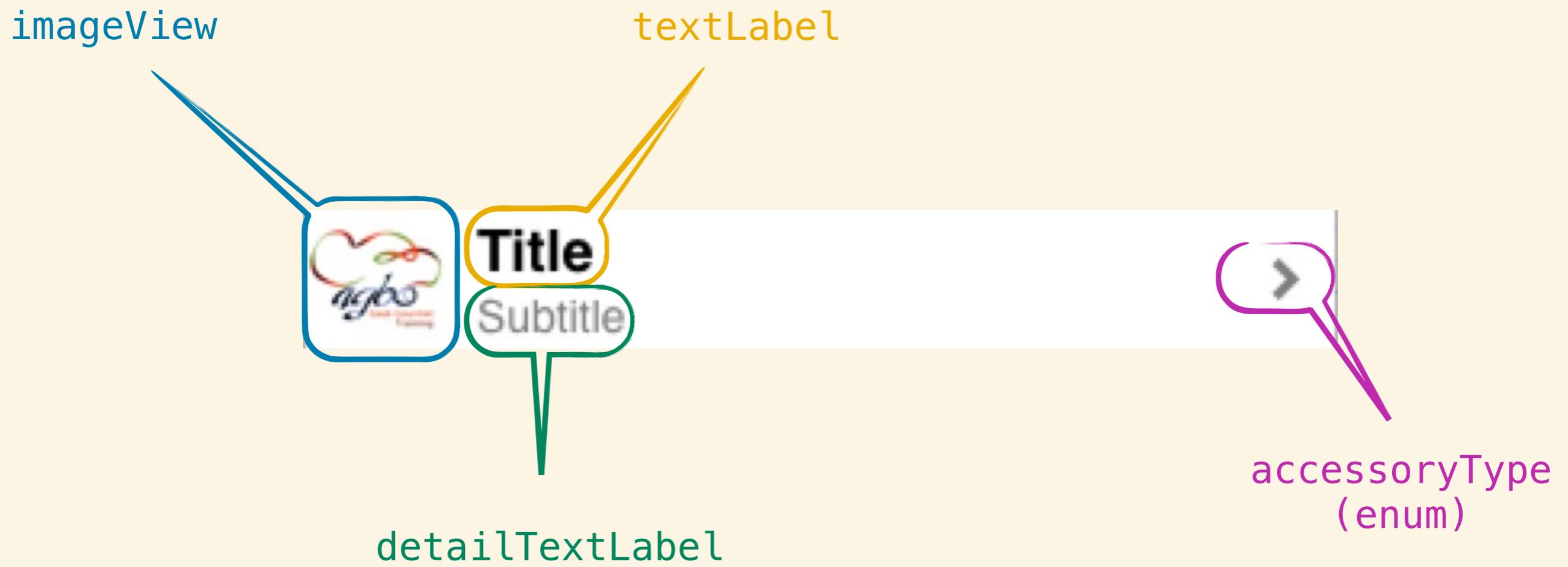
↑ Cabecera
Celdas personalizadas
↓ Pie
UITableViewCell

UITableViewCell

- Al igual que las tablas, se crean con un estilo.
- Hay 4 estilos estandar de celda.
- Se pueden crear celdas personalizadas



Anatomía de una celda



UITableViewCell

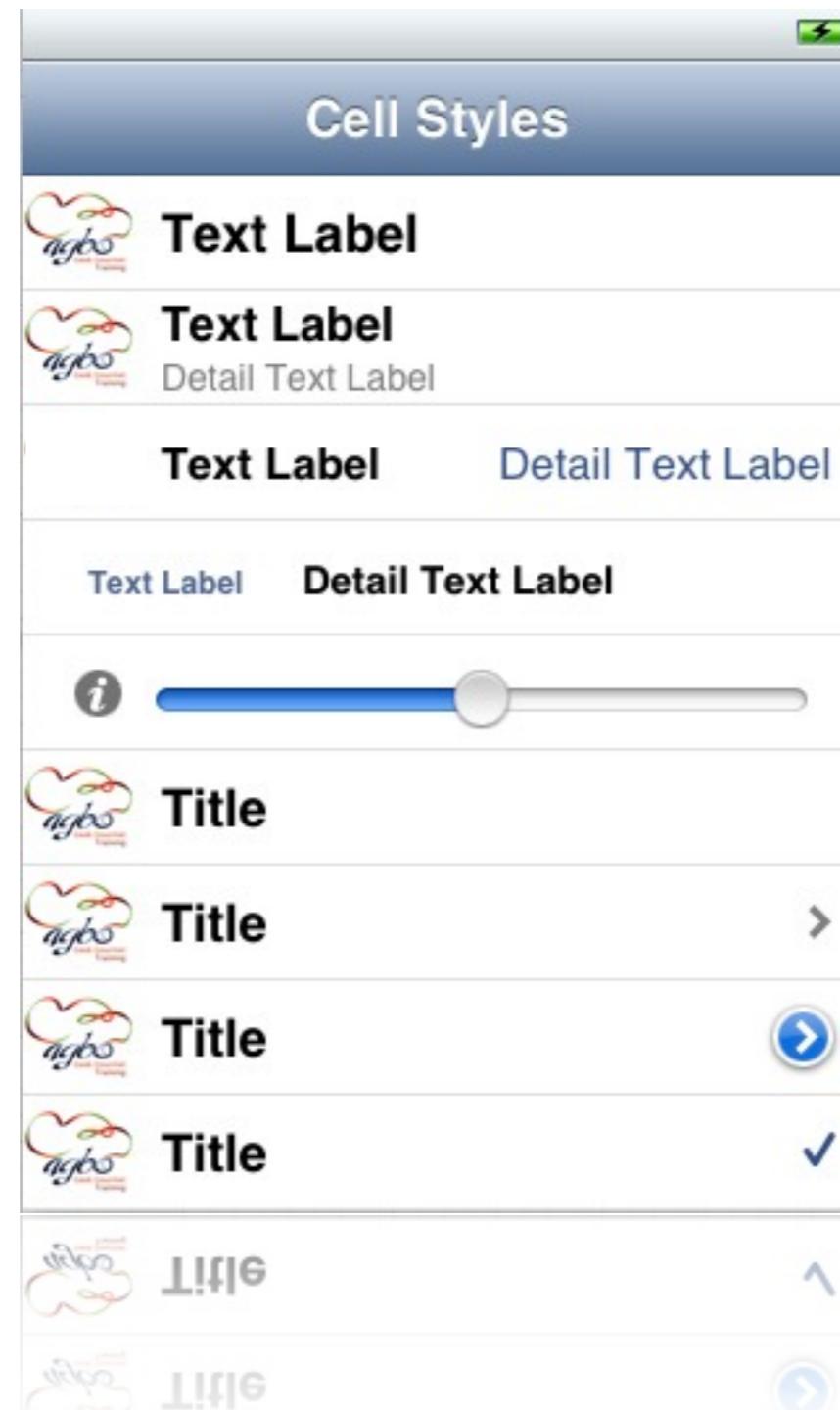
[UITableViewCellStyleDefault](#)

[UITableViewCellStyleSubtitle](#)

[UITableViewCellStyleValue1](#)

[UITableViewCellStyleValue2](#)

Custom cell



[UITableViewCellAccessoryNone](#)

[UITableViewCellAccessoryDisclosureIndicator](#)

[UITableViewCellAccessoryDetailDisclosureButton](#)

[UITableViewCellAccessoryCheckmark](#)



Uso de accessoryView



- Podemos sustituir el accessory type por una vista cualquiera, usando la propiedad accessoryView:

```
// Create the UISwitch
UISwitch *networking =[[UISwitch alloc] initWithFrame:CGRectMakeZero];

// Configure it
[networking setOn:YES]; // Or NO, of course!

// Connect it to a method
[networking addTarget:self
                  action:@selector(changeNetworking:)
forControlEvents:UIControlEventValueChanged];

// Add it as the accessoryView
[cell setAccessoryView:networking];
```

Personalización Avanzada

- Podemos poner cualquier vista que cubra por completo la celda, usando la propiedad `contentView`. Es el camino más sencillo.
- También podemos subclasicar `UITableViewCell` o crear su interfaz con Interface Builder.



Copyright © 2012 AGBO Business Architecture S.L. Todos los derechos reservados. www.agbo.biz

Tablas y Modelos

Un Modelo para modelarlos a todos



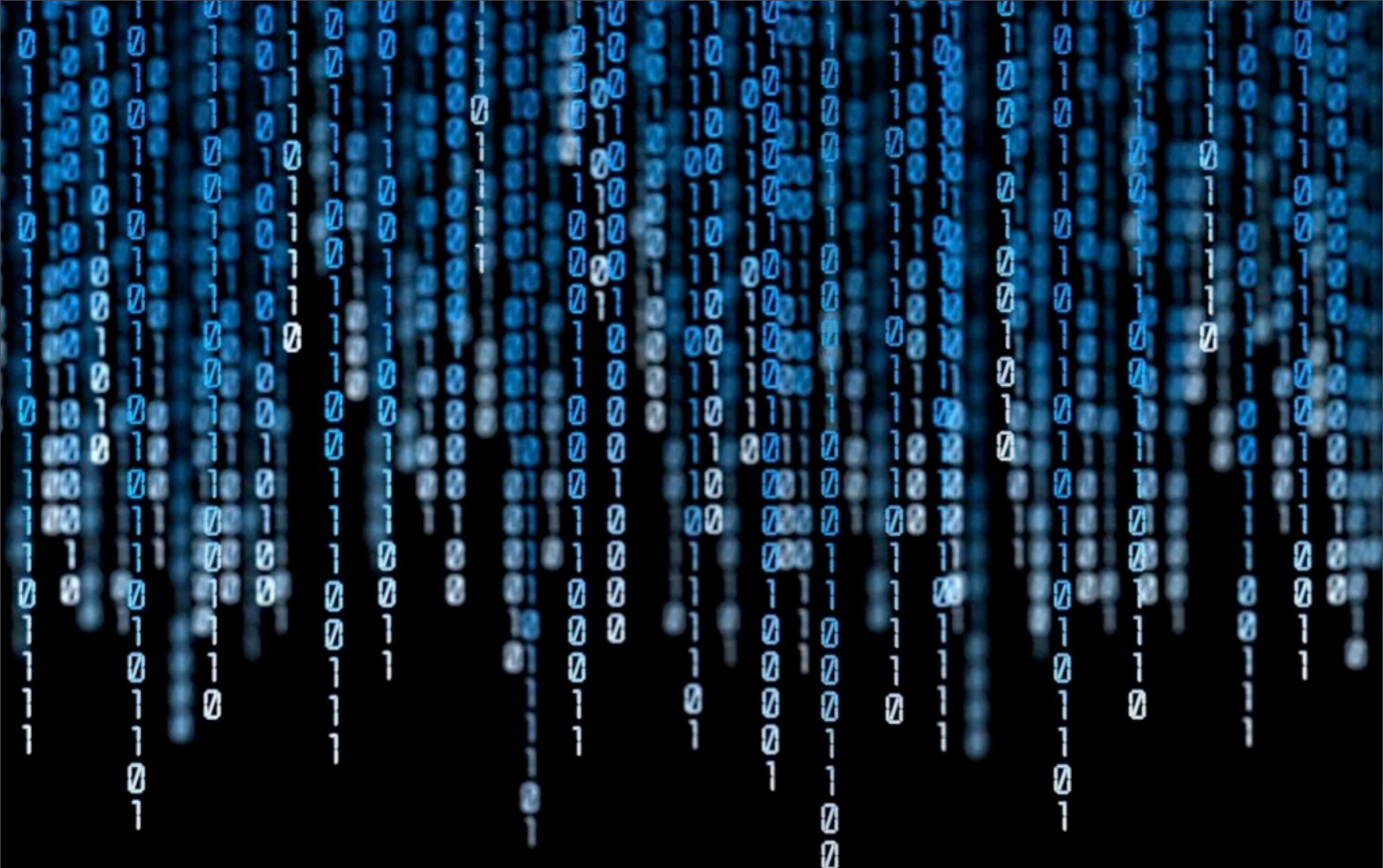
Modelo de Modelos

- El antiguo modelo representa a un personaje.
- Ahora necesitamos representar un conjunto de personajes.
- Necesitamos un modelo que representa un conjunto de modelos de personajes.





Copyright © 2012 AGBO Business Architecture S.L. Todos los derechos reservados.



Copyright © 2012 AGBO Business Architecture S.L. Todos los derechos reservados.

UISplitViewController al Detalle



Creación

- Como un UITabBarController, con alloc / init
- Se asigna un array de dos controladores a la propiedad viewControllers.

```
// The Split View Controller
UISplitViewController *splitVC = [[UISplitViewController alloc] init];

// The view controllers, usually inside a Navigation Controller
// so we have a navigation bar (split view only provides the split
UINavigationController *wineNav = [[UINavigationController alloc]
                                    initWithRootViewController:wineVC];
UINavigationController *wineCellarNav = [[UINavigationController alloc]
                                         initWithRootViewController:wineCellarVC];

// Add both to the SplitView
[splitVC setViewControllers:@[wineCellarNav, wineNav]];
```



Cambio de Orientación

- Vertical: Saca el controlador de la derecha, lo mete en un PopOver y lo “cuelga” de un botón.
- Horizontal: Saca el controlador que estaba en el PopOver y lo mete a la derecha
- Todo esto se notifica al delegado

UISplitViewController Delegate

El controlador de la derecha ha de ser el delegado del SplitView e implementar los siguientes métodos.

```
- (void)splitViewController:(UISplitViewController *)svc  
    willHideViewController:(UIViewController *)aViewController  
    withBarButtonItem:(UIBarButtonItem *)barButtonItem  
    forPopoverController:(UIPopoverController *)pc  
  
    // when going to portrait mode  
  
}  
  
- (void) splitViewController:(UISplitViewController *)svc  
    willShowViewController:(UIViewController *)aViewController  
    invalidatingBarButtonItem:(UIBarButtonItem *)barButtonItem  
  
    // When going to landscape mode  
  
}
```

Asegurarse que de verdad
es la izquierda!!!

Poner un diagrama en que
se especifique quién es
delegado de quién.



Esquema de Delegados



El de la derecha es
siempre delegado del
UISplitViewController

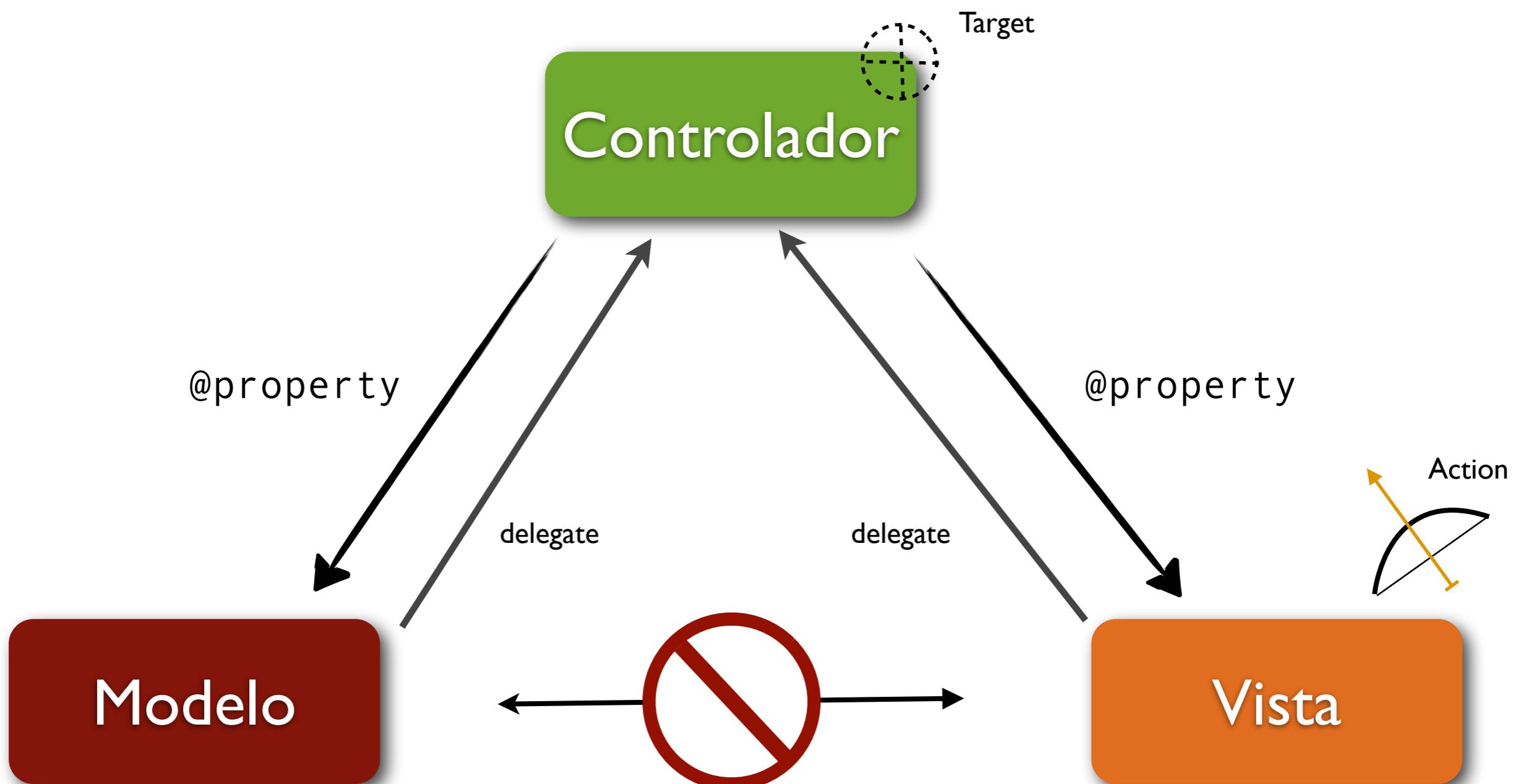


Copyright © 2012 AGBO Business Architecture S.L. Todos los derechos reservados.

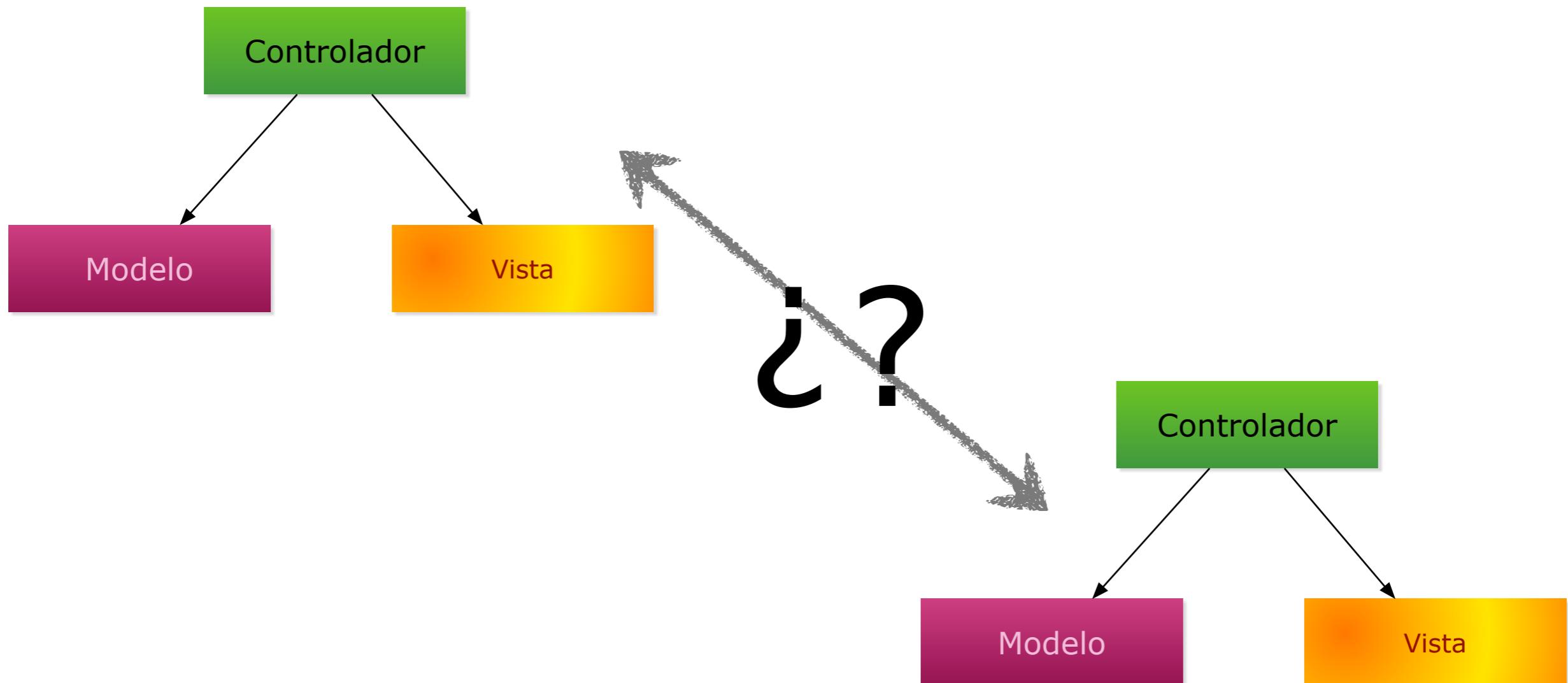


Copyright © 2012 AGBO Business Architecture S.L. Todos los derechos reservados. www.agbo.biz

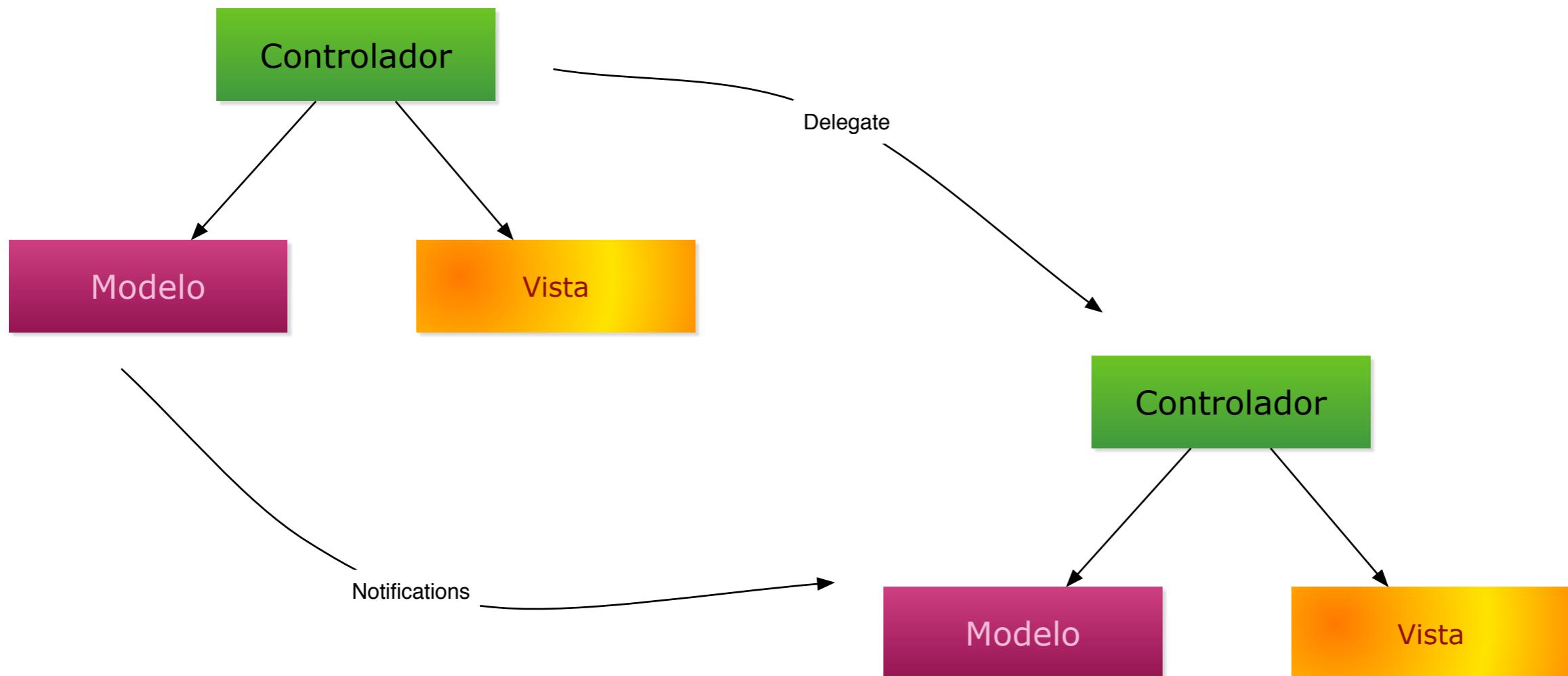
Comunicación entre los equipos



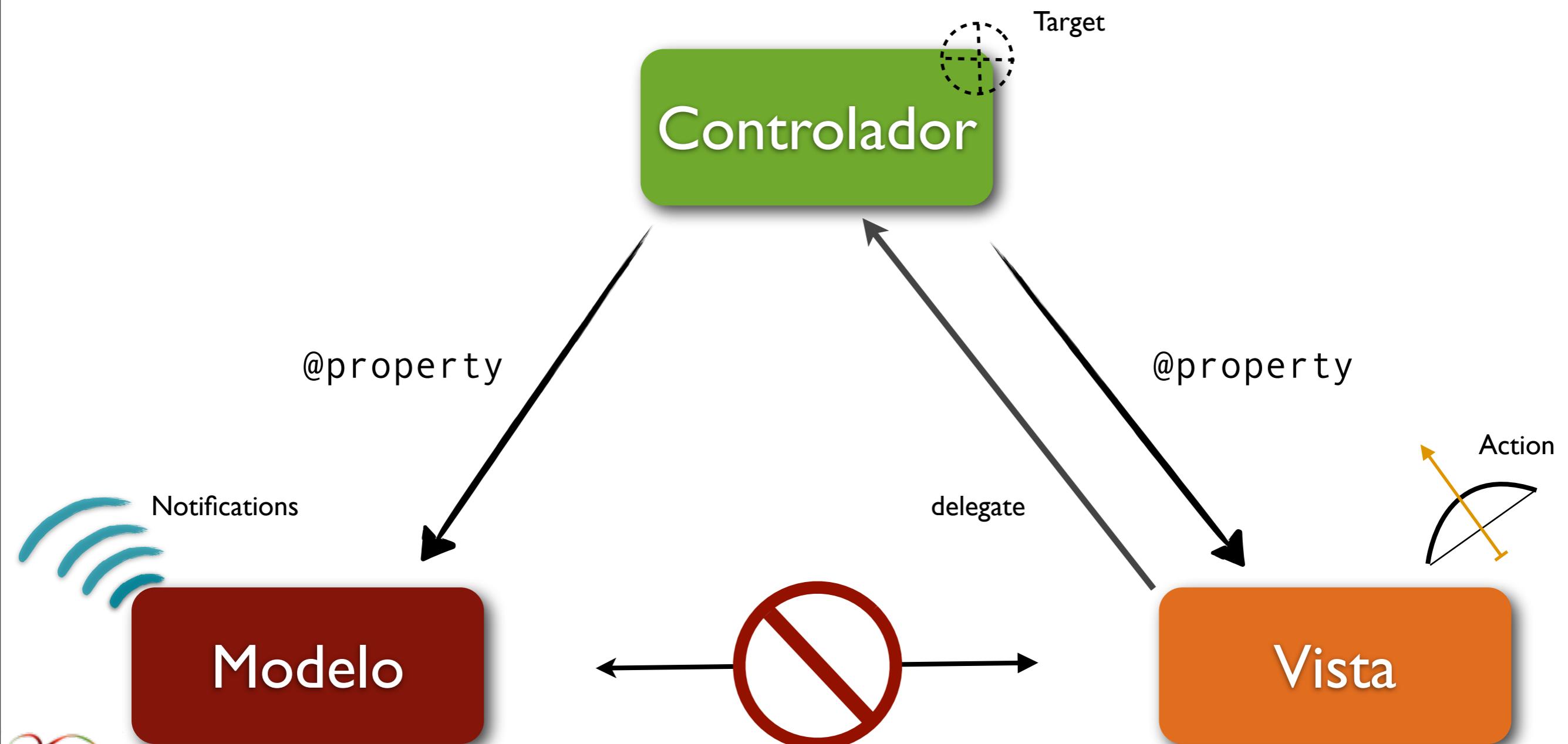
Comunicación entre distintos MVCs



Comunicación entre distintos MVCs



Comunicación entre los equipos



Definir un Protocolo

Es mucho menos común y se hace en un fichero de cabecera:

```
@protocol Foo  
  
-(void) doSomething: (NSString *) withThis;  
  
@optional  
-(id) getSomething;  
  
@required  
-(void) bar;  
  
@end
```



¿Implementas el Protocolo X? conformsToProtocol:

Se le puede preguntar a un objeto si implementa un determinado protocolo:

```
if ([s conformsToProtocol:@protocol(UIWebViewDelegate)]) {  
    // do something  
  
} else {  
    // do something else  
}
```

Esquema de Delegados



El de la derecha es siempre delegado del UISplitViewController y del de la izquierda



Copyright © 2012 AGBO Business Architecture S.L. Todos los derechos reservados.



Copyright © 2012 AGBO Business Architecture S.L. Todos los derechos reservados. www.agbo.biz



Modificadores de @property

Los IV mandamientos de las propiedades (1.0)

- ◆ I: Para los no-objetos:
 @property (nonatomic)
- ◆ II: Para los NSString:
 @property(nonatomic, copy)
- ◆ III: Para los IBOutlets y los delegates:
 @property (weak, nonatomic)
- ◆ IV: Para todo lo demás:
 MasterCard
 @property (strong, nonatomic)



Copy

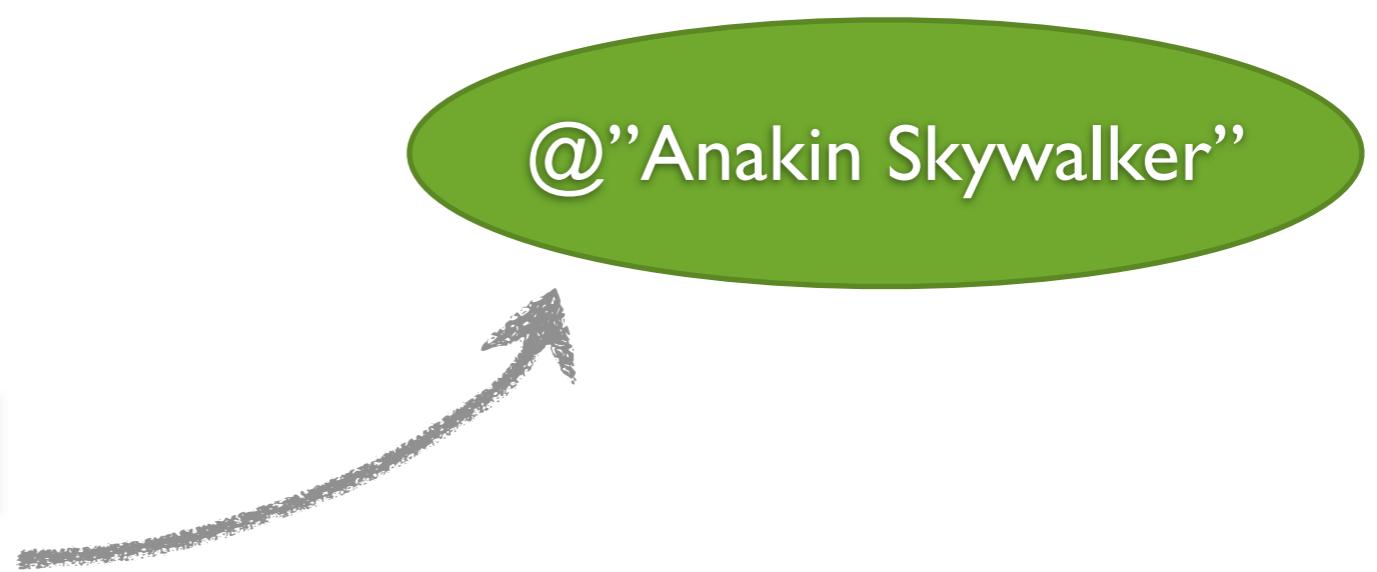
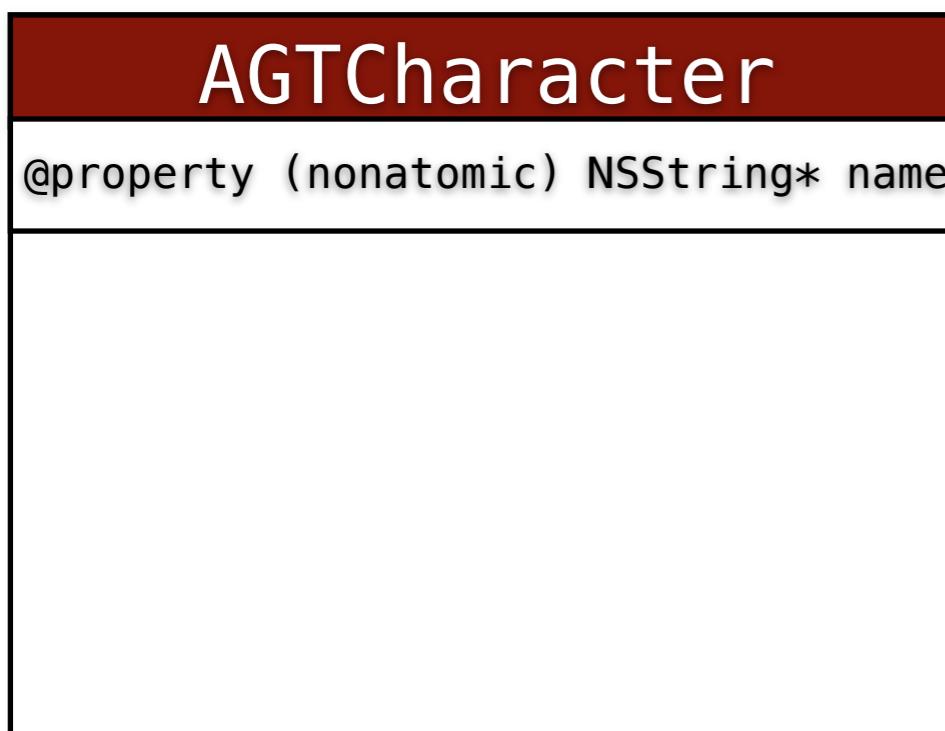


Alias: la madre de todos los bugs

Alias

NSMutableString

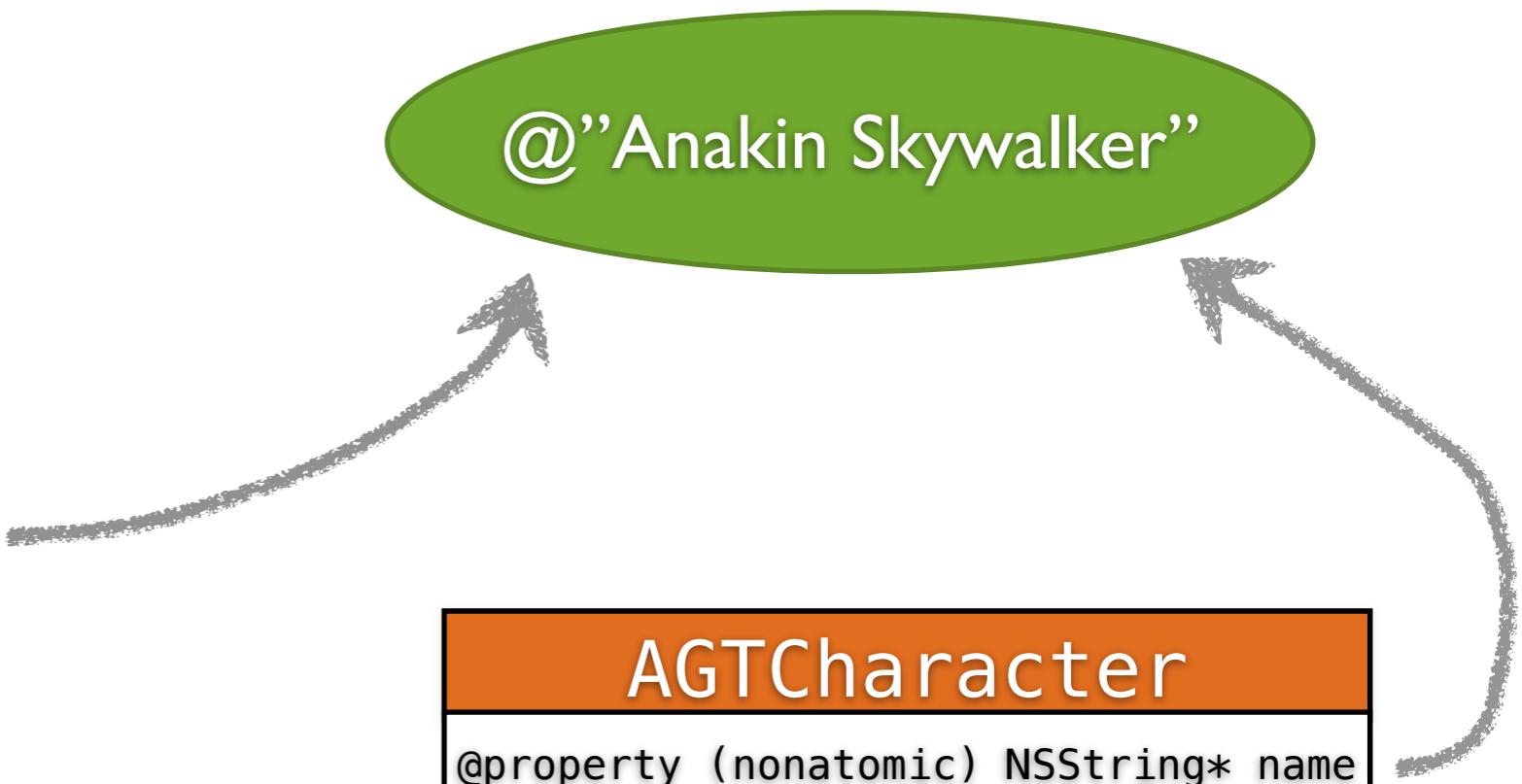
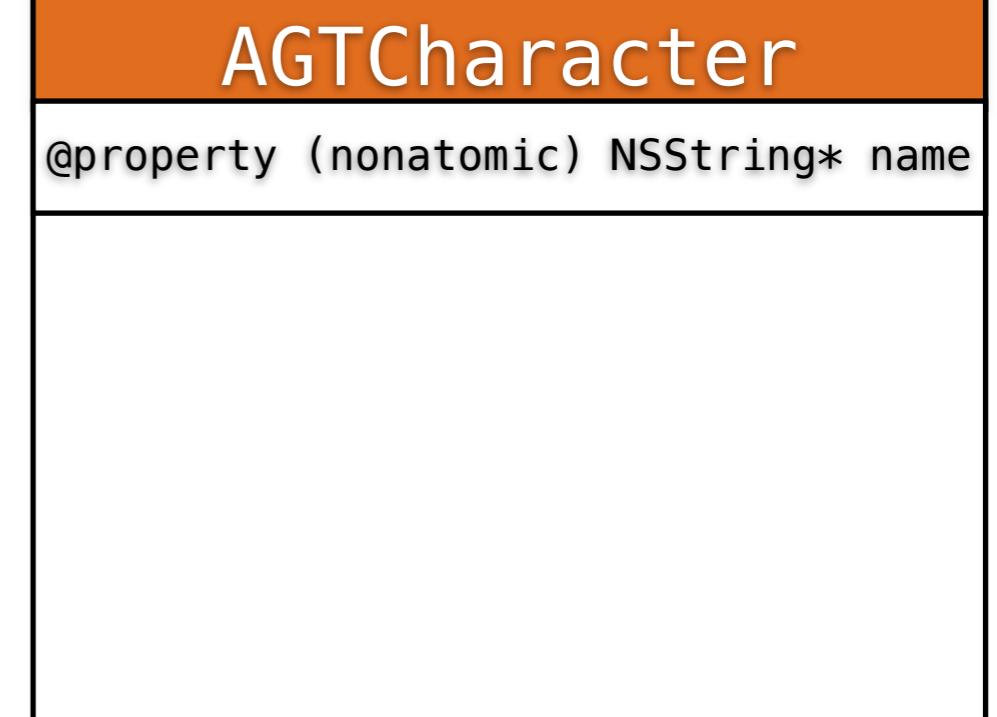
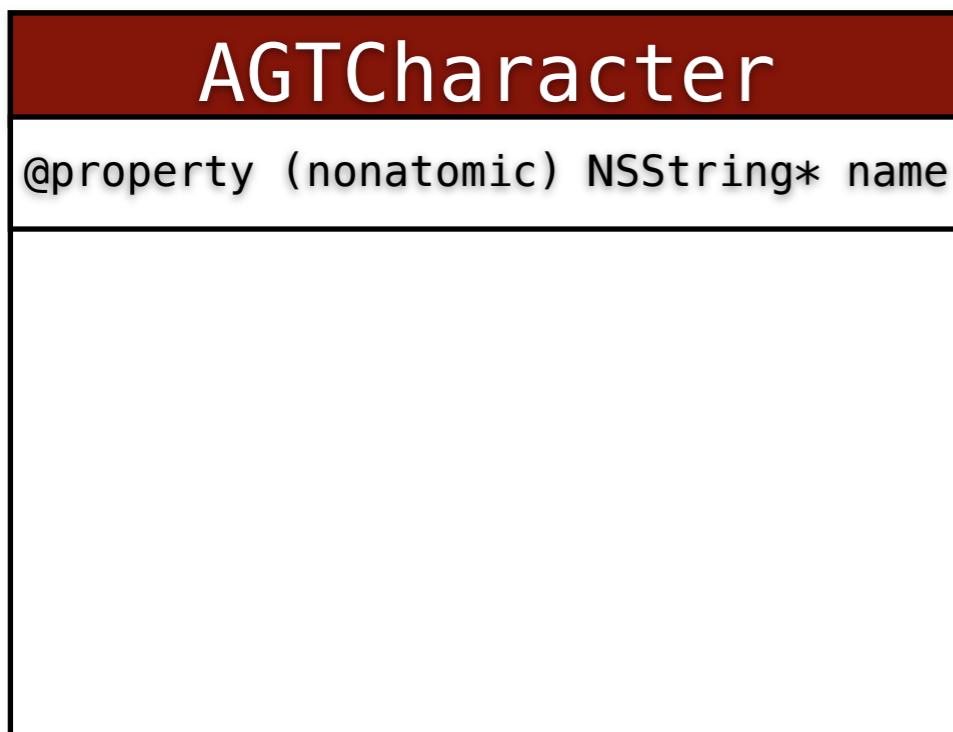
@”Anakin Skywalker”



Alias

NSMutableString

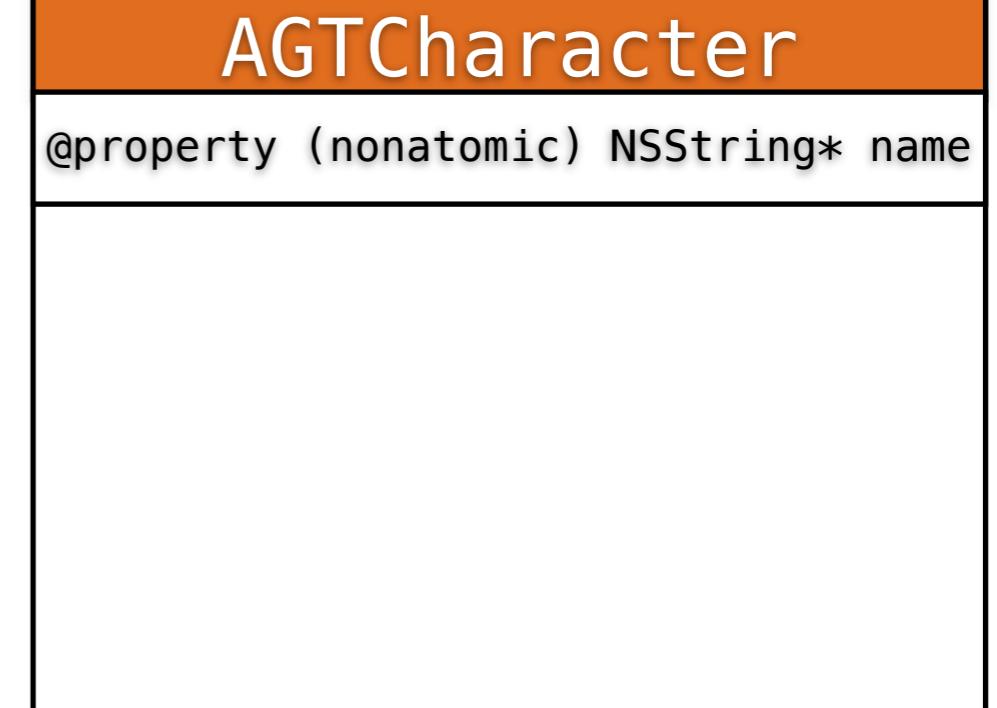
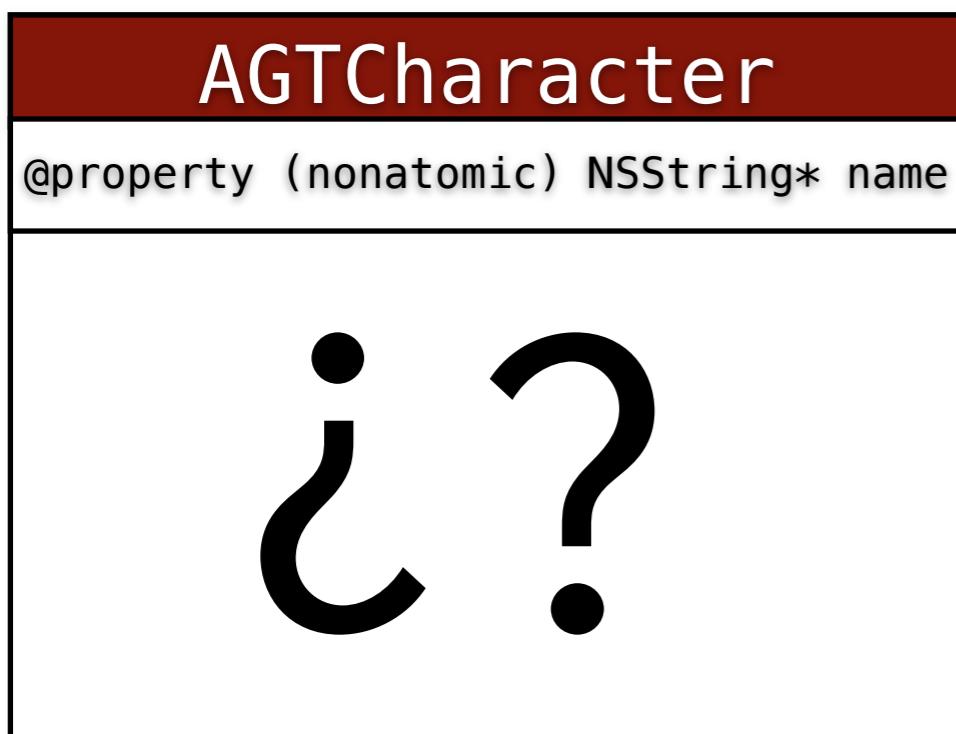
@”Anakin Skywalker”



Alias

NSMutableString

@"Manolo Skywalker"



Copy



- El setter hace una copia privada del objeto, a la cual nadie más tiene acceso.
- Tiene sentido para cualquier clase que tenga una subclase mutable.
- Por rendimiento, solo lo hacemos con `NSString`.

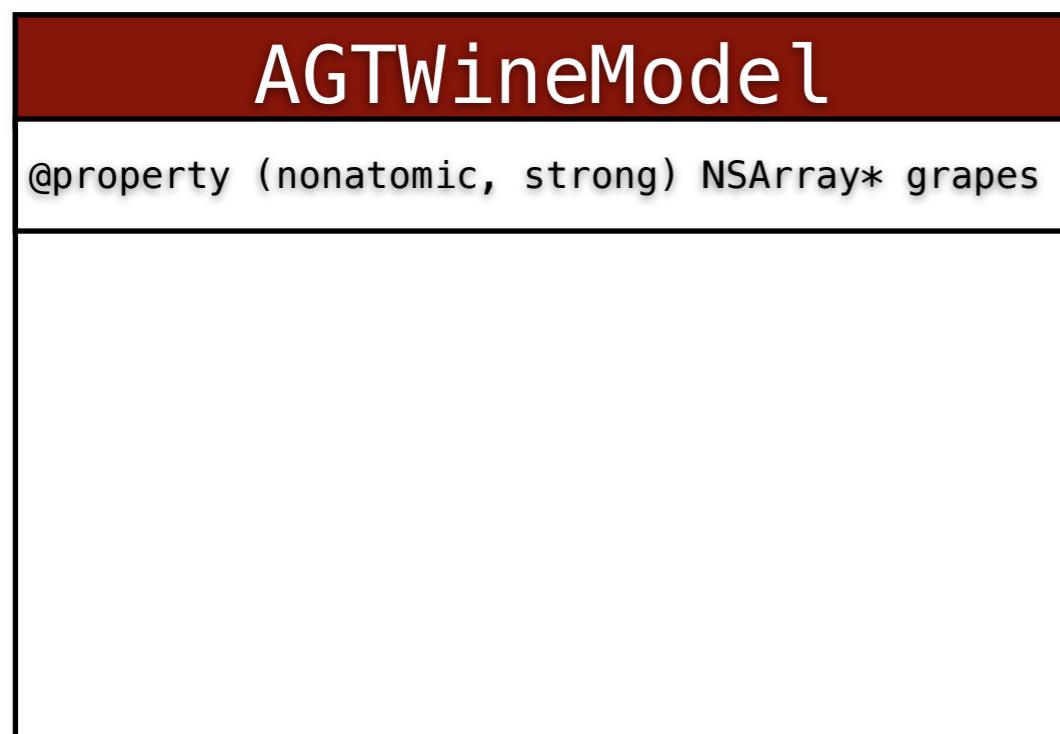
Strong & Weak

- Una `@property` crea una unión entre dos objetos.
- Strong y weak determinan el tipo de relación.
- Está relacionado con la gestión de memoria que hace ARC.

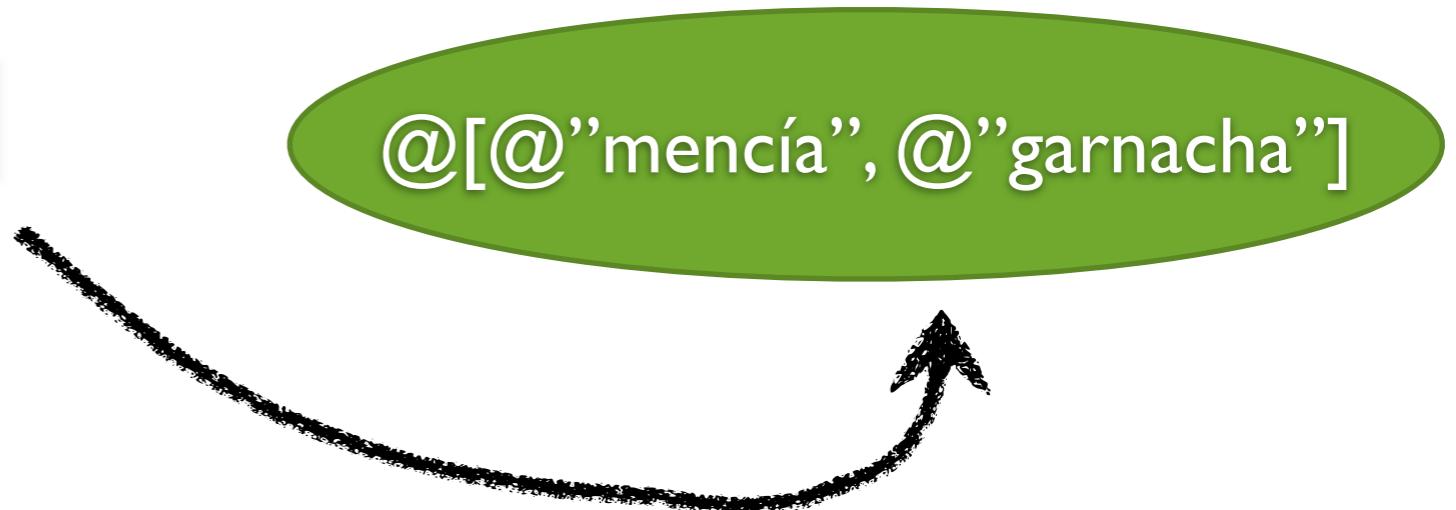


Strong

NSArray

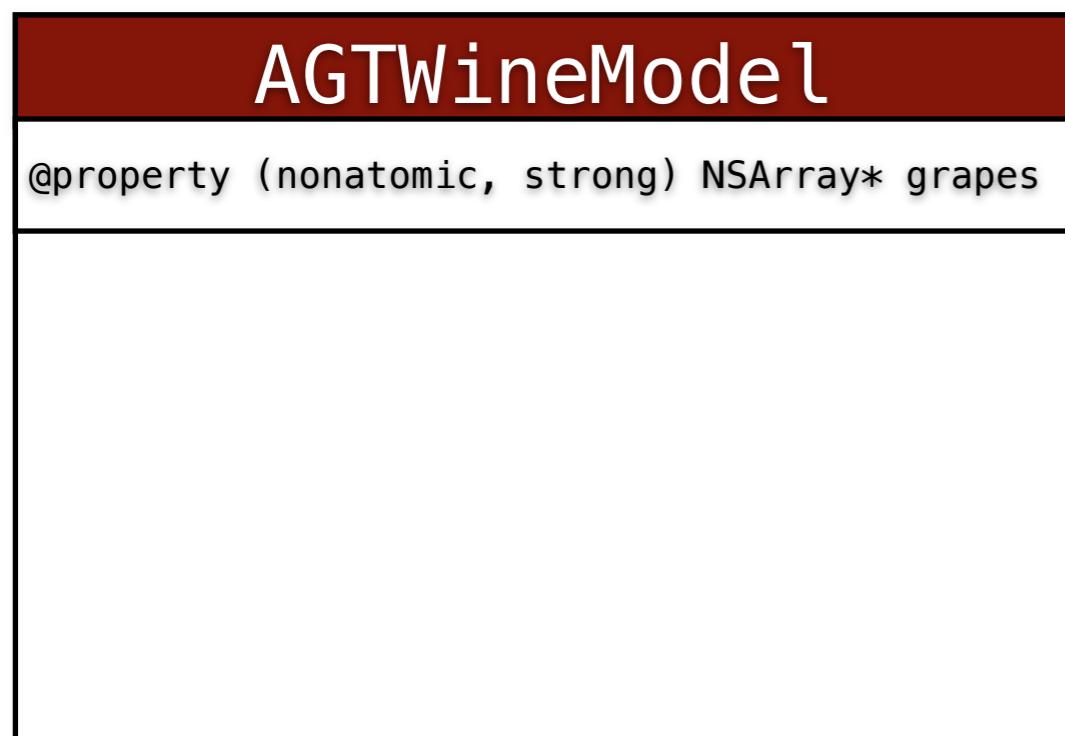


[@[@”mencía”, @”garnacha”]]



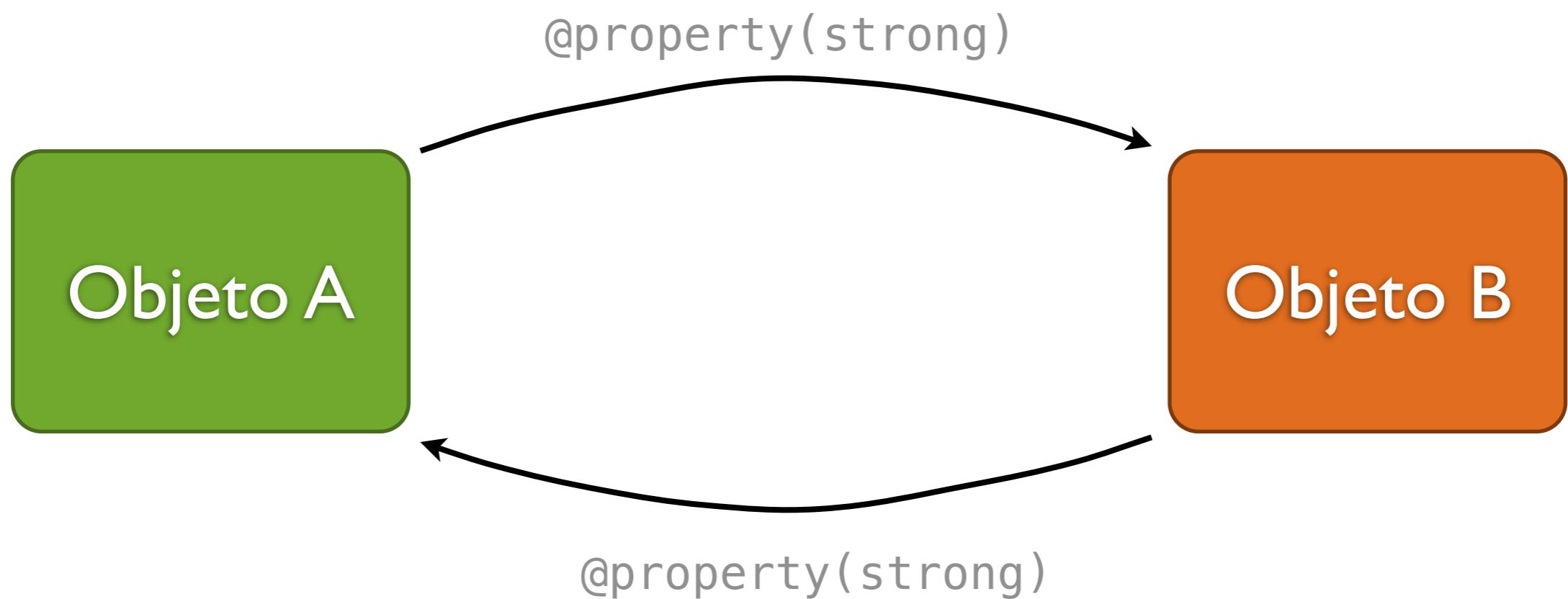
Weak

NSArray

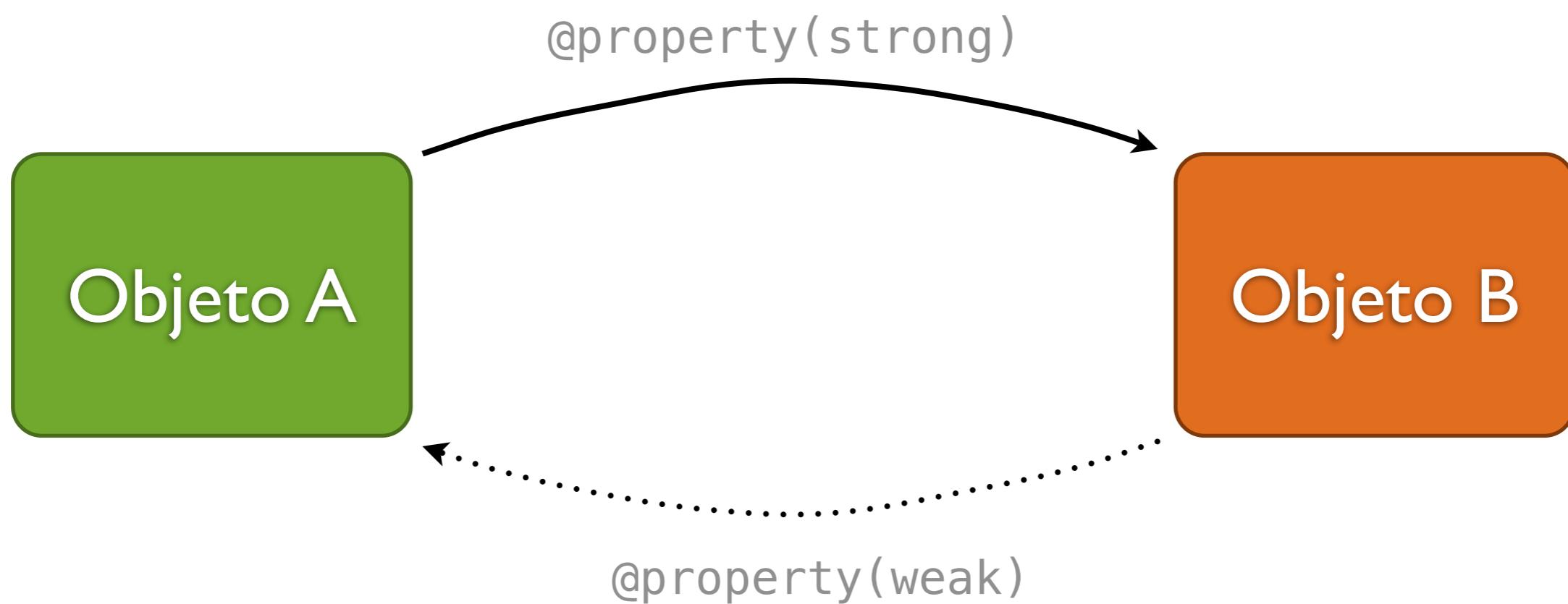


@[@”mencía”, @”garnacha”]

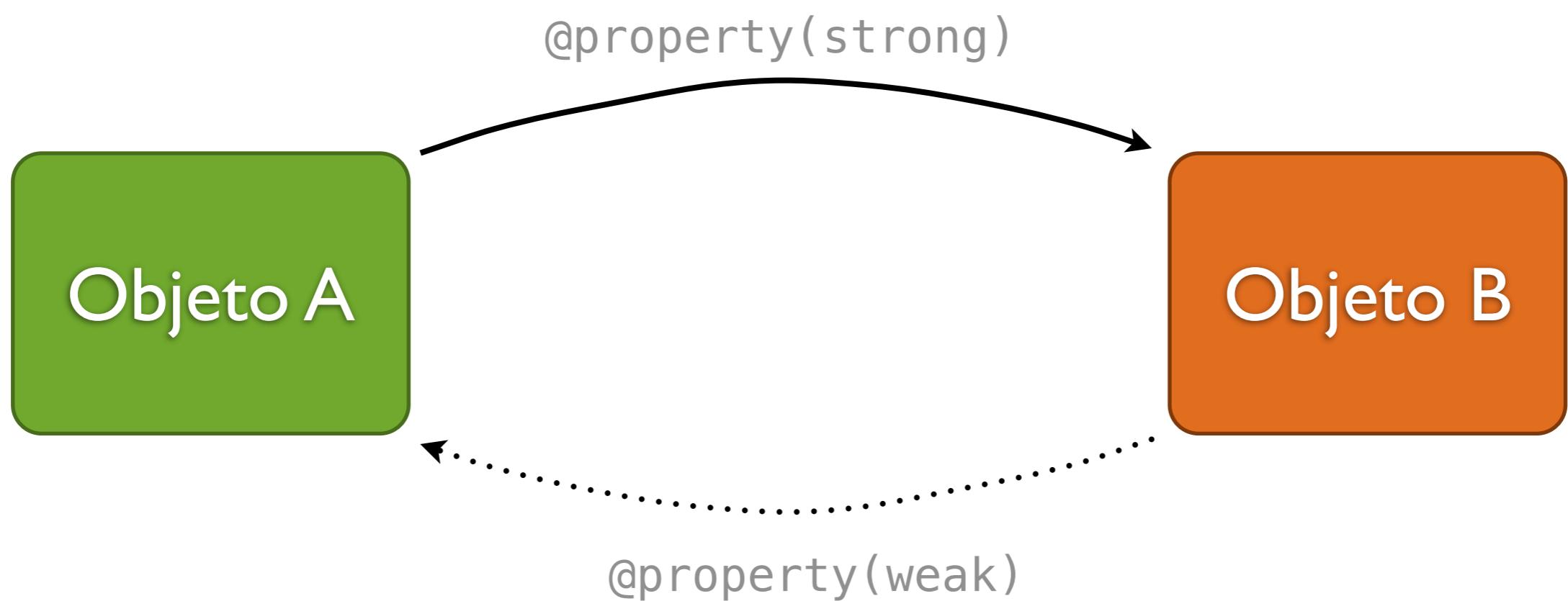
Ciclos de Objetos



Ciclos de Objetos



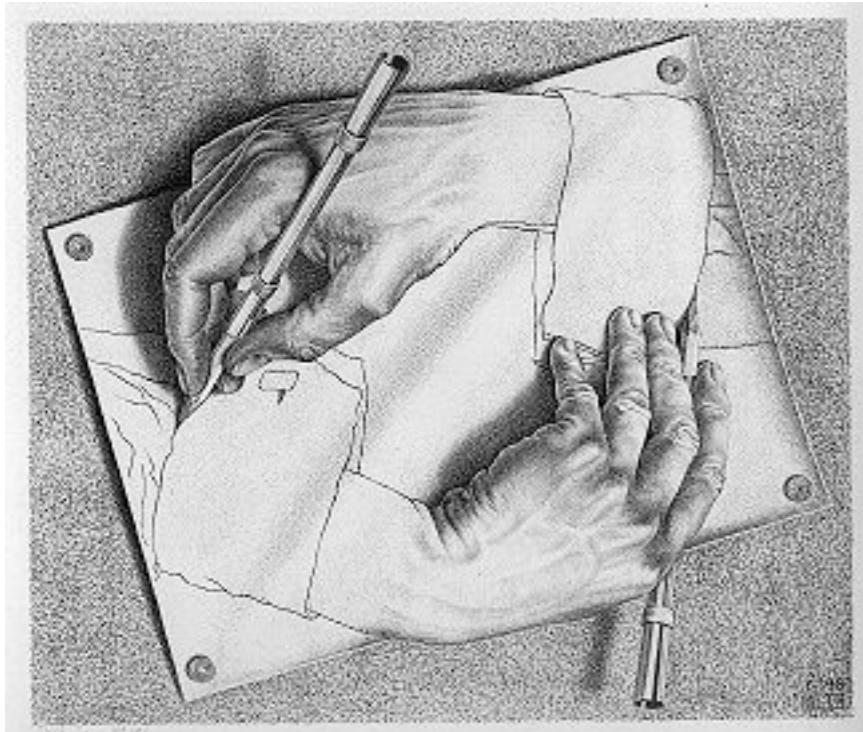
Ciclos de Objetos





Copyright © 2012 AGBO Business Architecture S.L. Todos los derechos reservados. www.agbo.biz

Ciclos de #import



- A.h contiene #import “B.h”
- B.h contiene #import “A.h”
- El compilador se vuelve loco.

Mandamientos de #import

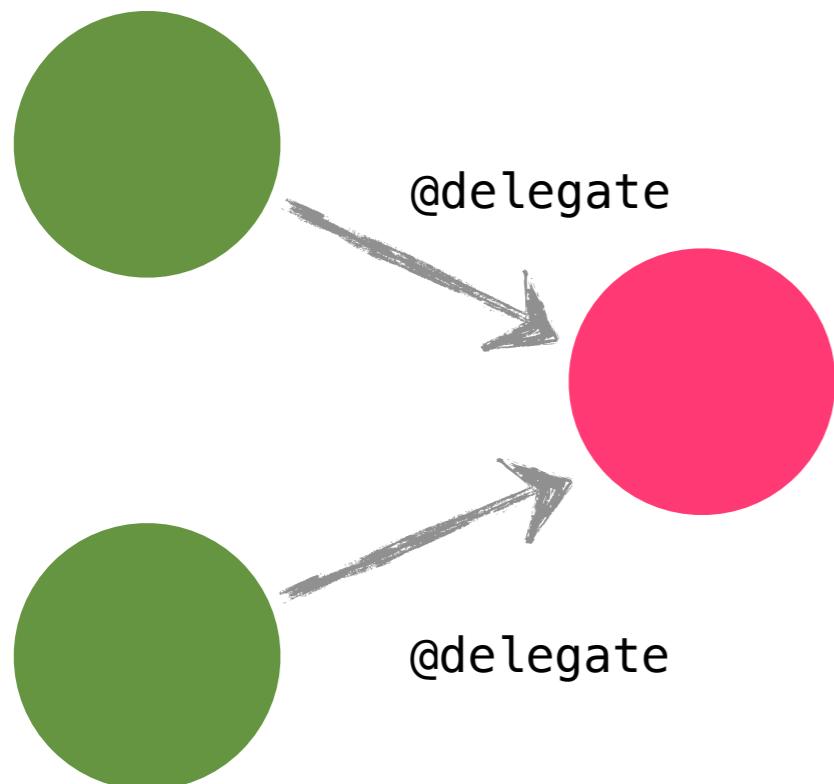
- ◆ I: Para librerías de sistema: no importa donde importes.
- ◆ II: Para clases que creaste tú:
 - ◆ @class en el .h
 - ◆ #import en el .m



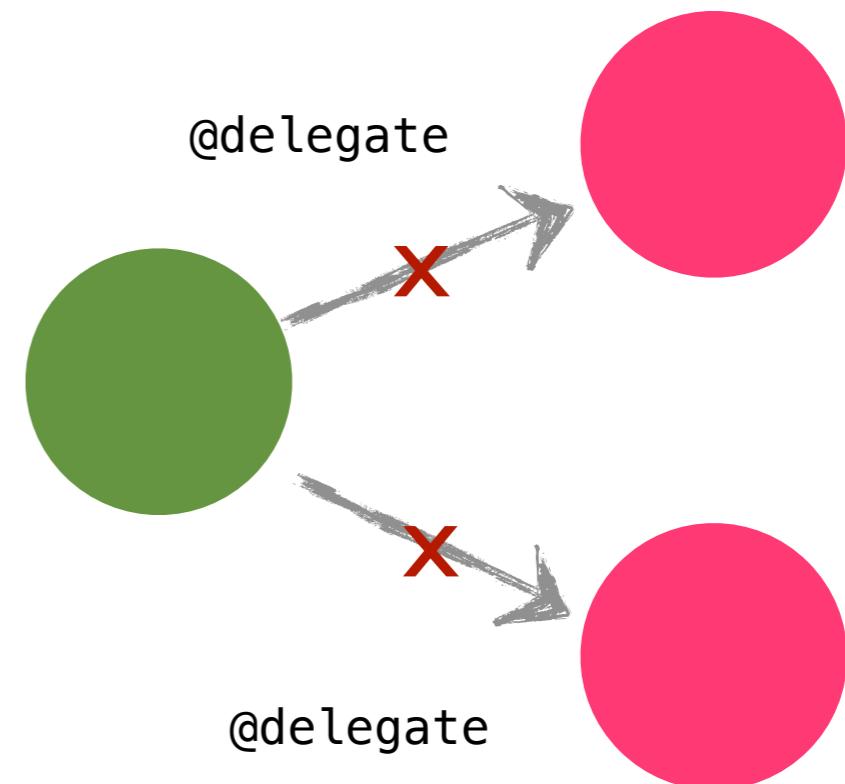


Copyright © 2012 AGBO Business Architecture S.L. Todos los derechos reservados. www.agbo.biz

Limitaciones del Delegate

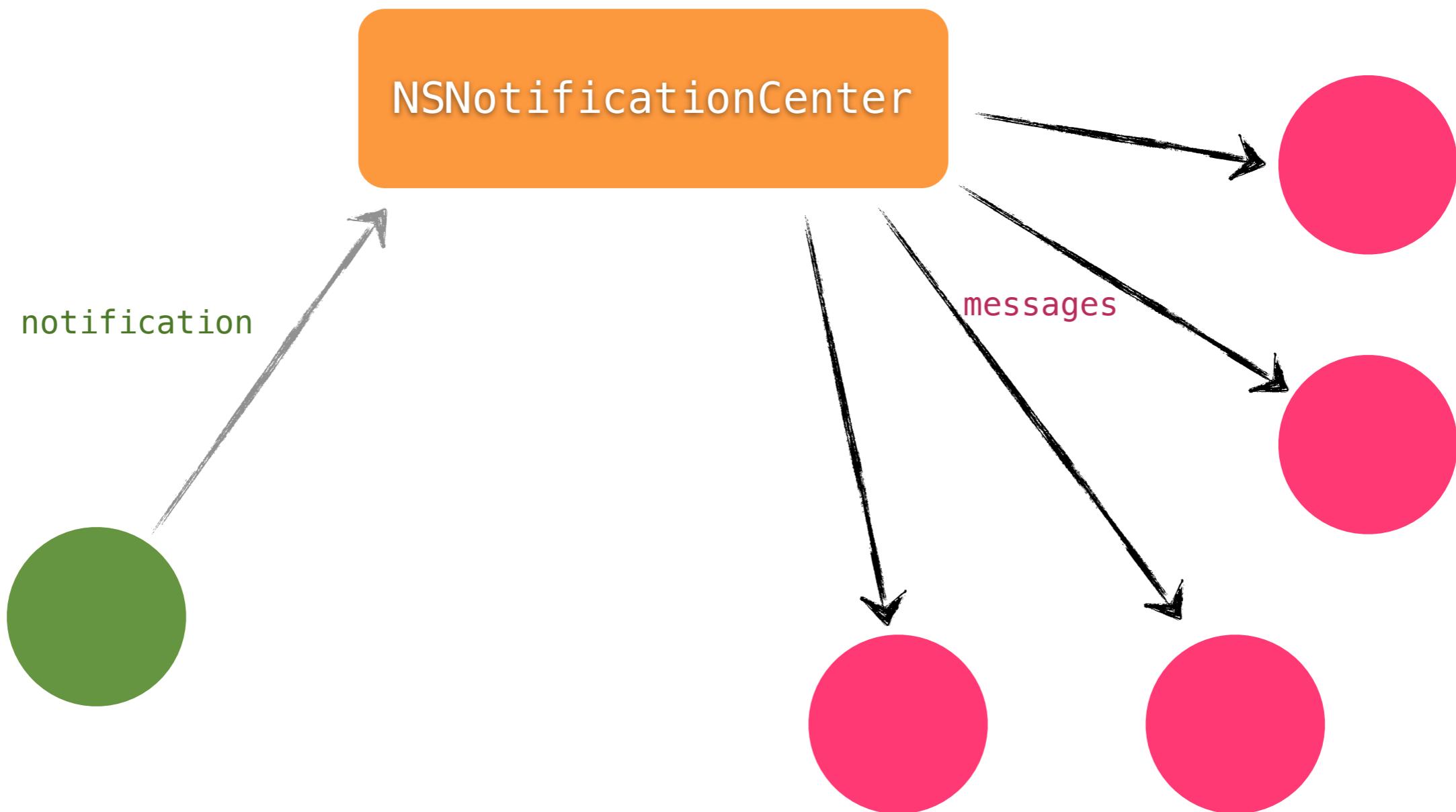


Un objeto puede ser delegado de más de un objeto



Un objeto NO puede tener más de un delegado

Notificaciones



Alta en NSNotificationCenter

- En qué notificación estamos interesados
- De quién
- Qué mensaje queremos que nos manden cuando se produzca la notificación



Alta en NSNotificationCenter

- Queremos ser avisados cuando se produzca la notificación `currentCharacterHasChanged`
- Cuando eso ocurra, queremos que el `NSNotificationCenter` nos envíe el mensaje `onCharacterChange:`
- Quiero ser notificado independientemente de quien mande el mensaje

```
NSNotificationCenter *center = [NSNotificationCenter defaultCenter];
[center addObserver:self
    selector:@selector(onCharacterChange:)
    name:@"currentCharacterHasChanged"
    object:nil];
```



Baja de NSNotificationCenter

Es importantísimo darse de baja. La última opción es en dealloc.

```
NSNotificationCenter *center = [NSNotificationCenter defaultCenter];
[center removeObserver:self];
```



Envío

- Creamos un objeto NSNotification.
 - Enviamos el mensaje postNotification: al NSNotificationCenter.

```
// Any extra info that should be included with the
// notification
NSDictionary *extraInfo = [NSDictionary dictionary];

// Create the notification
NSNotification *note = [NSNotification notificationWithName:@"currentCharacterHasChanged"
                                                 object:self
                                               userInfo:extraInfo];

// Post it
[[NSNotificationCenter defaultCenter] postNotification:note];
```

Recepción

- Implementamos el método que llamará NSNotificationCenter

```
-(void) onCharacterChange: (NSNotification *)aNotification {  
    // Update the views to keep in sync with the model  
}
```





Copyright © 2012 AGBO Business Architecture S.L. Todos los derechos reservados.



Copyright © 2012 AGBO Business Architecture S.L. Todos los derechos reservados. www.agbo.biz

Apps Universales



- Una app que funciona tanto en iPhone como iPad.
- Vamos a adaptar nuestra app de iPad para el iPhone.

Lo que cambia

- `rootViewController`
 - iPad: un `UISplitViewController`
 - iPhone: `UINavigationController`
- El comportamiento cuando el usuario toca sobre una celda de la tabla.



Detectar el Dispositivo

- No se detecta el dispositivo, sino el hardware que nos interesa.
- En nuestro caso, queremos saber si estamos en pantalla grande (tableta) o pequeña (iPhone o iPod)



Detectar la Pantalla

```
if ([[UIDevice currentDevice]
      userInterfaceIdiom] == UIUserInterfaceIdiomPad) {
    // Pantalla grande
}else{
    // Pantalla pequeña
}
```





Copyright © 2012 AGBO Business Architecture S.L. Todos los derechos reservados. www.agbo.biz

Persistencia Sencilla

NSUserDefaults



NSUserDefaults

Persistir Pequeñas Cantidades de Datos



NSUserDefaults

- Se usa para persistir pequeñas cantidades de datos, como preferencias de usuario.
- NSString, NSNumber, NSDate, NSArray y NSDictionary.
- Es un NSDictionary que automágicamente guarda su contenido en disco.



Patrón de Uso de NSUserDefaults

- Al arrancar la app, nos aseguramos de que estén presentes valores por defecto.
- A lo largo de la app, vamos leyendo o modificando los valores.
- Aunque guarda en disco de forma automática, es recomendable forzar un guardado con el método `synchronize`.



Creación de NSUserDefaults

- Con el método de clase `standardUserDefaults`

```
NSUserDefaults *defaults = [NSUserDefaults standardUserDefaults];
```

Valor por Defecto

Se asigna en `application:didFinishLaunchingWithOptions:`

```
- (BOOL)application:(UIApplication *)application  
didFinishLaunchingWithOptions:(NSDictionary *)launchOptions  
{  
  
    NSUserDefaults *defaults = [NSUserDefaults standardUserDefaults];  
  
    // comprobamos que no hay nada en la dirección  
    if (![defaults objectForKey:@"dirección"]) {  
  
        // Si no hay nada, lo añadimos  
        [defaults setObject:@"1 Infinite Loop, Cupertino"  
                      forKey:@"dirección"];  
  
        // Aunque se guarda automágicamente a cada x segundos  
        // es prudente guardar después de añadir algo  
        [defaults synchronize];  
    }  
  
    // A partir de aquí siempre que se acceda a la clave  
    // dirección, sabremos que al menos hay el valor por  
    // defecto  
}
```



Lectura y Escritura

Igual que un NSDictionary

```
// Leemos un valor  
NSString *dir = [defaults objectForKey:@"dirección"];  
  
// Escribimos un valor  
[defaults setObject:@"13, Rue del Percebe"  
                 forKey:@"dirección"];  
  
// ¡Guardamos, que Murphy acecha!  
[defaults synchronize];
```



Error Común



Todo lo que devuelve `NSUserDefaults` es Inmutable, ¡aunque tú lo hayas guardado como Mutable!

Error Común

```
NSUserDefaults *defaults = [NSUserDefaults standardUserDefaults];  
  
// Creo un array mutable con algunos nombres  
NSMutableArray *starWars = [NSMutableArray arrayWithObjects:  
    @"Anakin Skywalker",  
    @"Obi Wan Kenobi", nil];  
  
// Lo guardo en NSUserDefaults  
[defaults setValue:starWars  
    forKey:@"nombres"];  
  
// Lo recupero e intento añadir algo:  
// me da un error, porque en vez de un NSMutableArray  
// me devuelve iun NSArray (no es modificable)!  
starWars = [defaults objectForKey:@"nombres"];  
[starWars addObject:@"Frodo Bolson"]; // ierror! Se cae la app. :-(  
  
// He de crear una copia mutable (con mutableCopy)  
// para poder hacer moficiaciones  
starWars = [[defaults objectForKey:@"nombres"] mutableCopy];  
[starWars addObject:@"Minch Yoda"]; // ahora sí que funciona :-)
```





Copyright © 2012 AGBO Business Architecture S.L. Todos los derechos reservados.



Copyright © 2012 AGBO Business Architecture S.L. Todos los derechos reservados. www.agbo.biz

Persistencia Intermedia

Sistema de Archivos y Sandbox



Sistema de Archivos

- Es un sistema de ficheros Unix
- La seguridad es muy estricta
- Toda App está encerrada en una “Sandbox”



La Sandbox

Una jaula en la que está encerrada tu App



- Por seguridad: nadie puede sobreescribir tus datos.
- Por privacidad: nadie puede leer tus datos.
- Por higiene: cuando tu App es eliminada, no deja rastros en el sistema.

Principales Carpetas de la Sandbox

- App bundle: tiene tus binarios, imágenes, etc. Es de solo lectura.
- Documentos: para guardar datos permanentes creados por el usuario
- Caches: Datos temporales (no se hace copia de seguridad con iTunes)
- Otros: ver NSSearchPathDirectory en la documentación

Manejo de Archivos y Directorios

Se usan 3 clases

- **NSFileManager**
- **NSString**
- **NSData**



Obtener la Ruta a las Carpetas del Sandbox

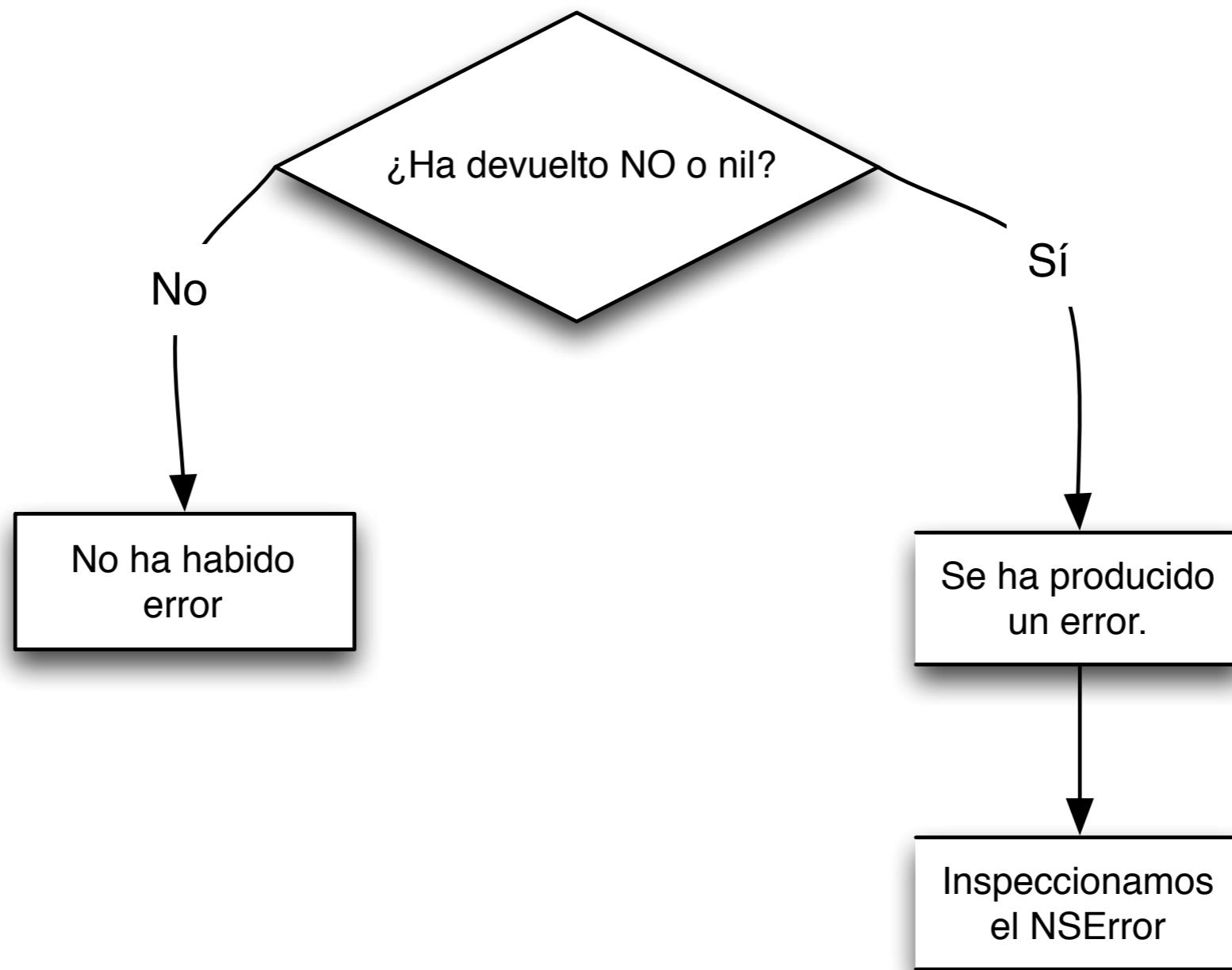
```
// Obtenemos una instancia de NSFileManager  
NSFileManager *fm = [NSFileManager defaultManager];  
  
// El método URLsForDirectory:inDomains:  
// devuelve un NSArray de NSURLs, pero  
// solo nos interesa el último  
NSArray *urls = [fm  
    URLsForDirectory:NSDocumentDirectory  
    inDomains:NSUTFUserDomainMask];  
  
// La forma estándar de acceder a recursos locales  
// o remotos es mediante un NSURL  
NSURL *url = [urls lastObject];  
  
// Si queremos acceder a un fichero, añadimos  
// su nombre a la ruta  
url = [url URLByAppendingPathComponent:@"MyFile"];
```



Gestión de errores en Cocoa

- Todo método que pueda generar un error tiene dos características:
 - Devuelve un BOOL o un objeto
 - Acepta un NSError* por referencia.
- Lo que determina si se ha producido un error es el *valor de retorno*.

Gestión de Errores



Lectura y Escritura con NSString

```
// Escritura  
[@"hola" writeToURL:atomically:encoding:error:]  
  
// Lectura  
[NSString stringWithContentsOfURL:usedEncoding:error:]
```



Lectura y Escritura con NSData

```
// Escritura
NSData *data = [NSData data];
[data writeToURL:options:error:];

// Lectura
[NSData dataWithContentsOfURL:options:error:];
```



Error Común



Lo que indica el error es el valor de retorno, ¡no que haya algo en el puntero a NSError!



Copyright © 2012 AGBO Business Architecture S.L. Todos los derechos reservados.



Copyright © 2012 AGBO Business Architecture S.L. Todos los derechos reservados. www.agbo.biz

Avisos Modales



UIActionSheet

```
[UIActionSheet alloc] initWithTitle:  
                           delegate:  
                           cancelButtonTitle:  
                           destructiveButtonTitle:  
                           otherButtonTitles:, nil];
```

```
[action showInView:self.view];
```



UIAlert

```
UIAlertView *alert = [[[UIAlertView alloc]
    initWithTitle:@"Timeout"
    message:@"Fallo al conectar"
    delegate:self
    cancelButtonTitle:@"Cancelar"
    otherButtonTitles:@"Reintentar", nil];
[alert show];
```





Copyright © 2012 AGBO Business Architecture S.L. Todos los derechos reservados.



Copyright © 2012 AGBO Business Architecture S.L. Todos los derechos reservados. www.agbo.biz