ETSIT UPM

dit UPM

Desarrollo de Apps para iOS
View Controllers

IWEB,LSWC 2013-2014
Santiago Pavón

ver: 2014-03-10

Una Pantalla

- Quiero hacer una clase que represente una pantalla de mi aplicación.
  - Toda la pantalla en un iPhone, parte de la de iPad,…
- Será una clase controladora (MVC) que:
  - mostrará una vista (formada por una jerarquía de views) con los datos del modelo.
  - actualizará el modelo según se interactúe con la vista.
- Que me avise cuando haya problemas de memoria.
  - Me permita liberar los objetos que no necesite.
  - Más tarde los reconstruiré otra vez, cuando los vuelva a necesitar.
- Que me avise cuando la pantalla vaya a hacerse visible, o se haya hecho visible.
  - Y al revés: cuando vaya a dejar de ser visible, o ya no sea visible.
- Que me ayude a presentar correctamente mi vista cuando giro el terminal.
- Que me ayude en la gestión de la navegación:
  - presentar otras pantallas, poder volver a las pantallas anteriores, tener pestañas para elegir que pantalla veo, etc…
- y más cosas.

Esto lo podemos programar nosotros

o

podemos usar la clase

`UIViewController`

que ya lo hace

La Clase UIViewController

- Es la **clase base** para crear nuestras propias clases VC, es decir, nuestras propias pantallas.
  - Crearemos **clases derivadas** para añadir la lógica de nuestra aplicación.
    - Sobreescribiendo o añadiendo propiedades y métodos.
- Proporciona la parte controladora (**C**) del patrón MVC.
- Proporciona una vista (**V**) vacía a la que añadiremos nuestra jerarquía de views.
  - La jerarquía de views se puede crear programáticamente, o
  - cargando el GUI desde un fichero storyboard o XIB.
    - Definiremos IBOutlets y IBActions para enganchar las propiedades y métodos entre el código y el diseño gráfico.
- **No** proporciona el modelo (**M**).
- Esta clase base nos proporciona también muchos métodos y propiedades para realizar las tareas descritas anteriormente: gestión de memoria, rotaciones, navegación, transiciones, etc...
  - Sobreescribiremos estos métodos para adaptarlos a nuestras necesidades.

La Clase UIViewController

- Al diseñar nuestra aplicación, diseñaremos todas las pantallas que la forman.
  - cada una de estas pantallas se encargará de una tarea

- Cada pantalla será un objeto de la clase **UIViewController**.
  - o mejor dicho, de una clase derivada de ésta.

- Para navegar entre las distintas pantallas usaremos controladores de navegación, controladores de pestañas, vistas modales.
  - Los estudiaremos en otros temas.

Nota: en un iPad una pantalla puede mostrar varias UIVC simultáneamente.
Nota: también podemos mostrar varios UIVC en un iPhone usando vistas contenedoras.

Crear Ficheros VC .h y .m

- ¿Cómo se crean los ficheros .h y .m de una subclase de UIViewController?
  - A mano desde cero.
  - o usando una plantilla de Xcode:

    Xcode **>** *menú File* **>** *New* **>** *File* **>**
    iOS + Cocoa Touch **>** *Objective-C class* **>**
    Poner nombre a la clase **>** *Subclass of UIViewController* **>**
    Con o sin XIB para el GUI **>**
    ¿Para iPad? **>**
    Seleccionar Group (subdirectorio) donde se crearán los ficheros **>**
    Marcar Targets (dependencias de compilación)

Crear un Proyecto

- Para crear un proyecto que sólo contiene una pantalla, es decir, una única clase UIViewController:

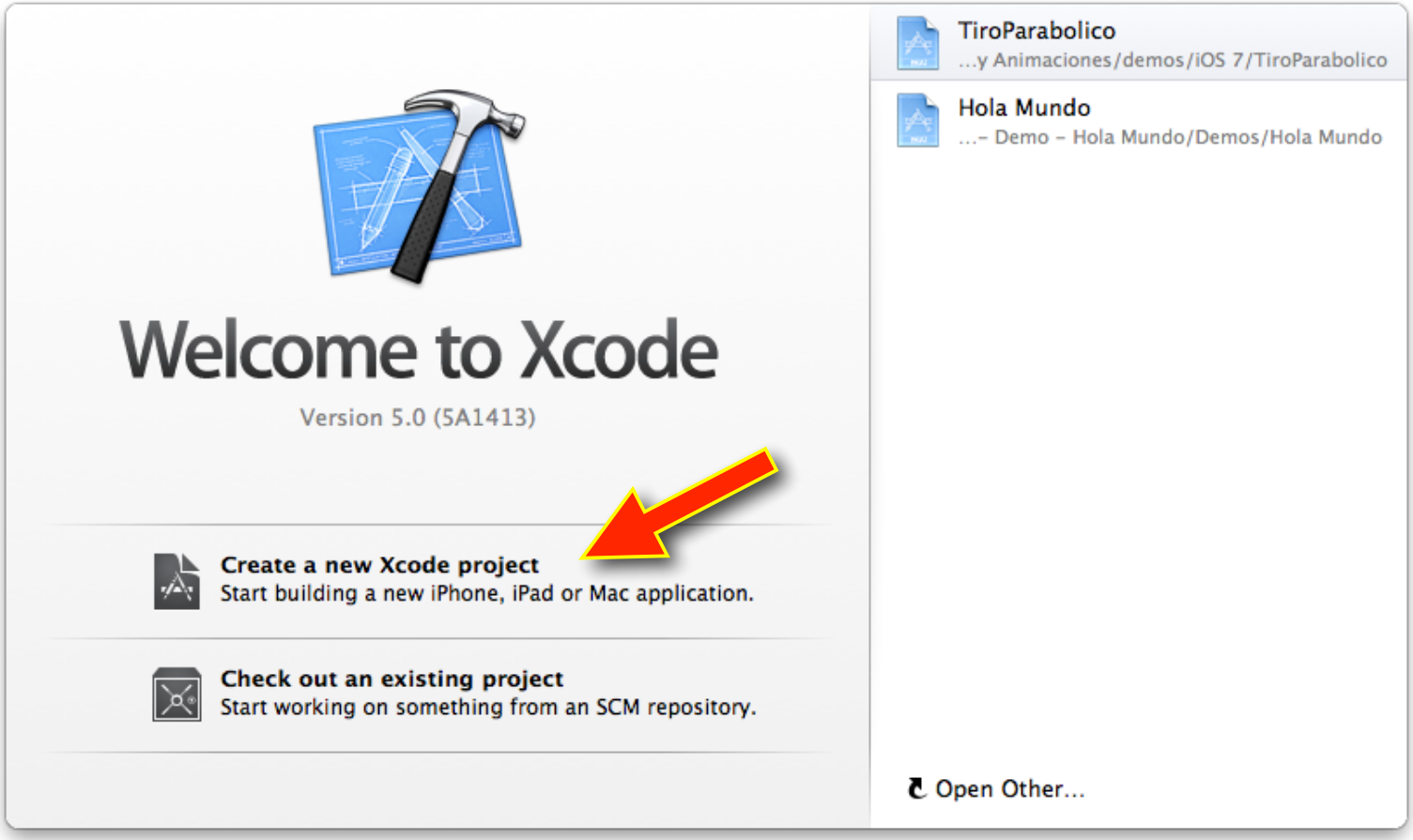
  Xcode **>** *menú File* **>** *New* **>** *Project* **>**
  iOS + Application **>** *Single View Application* **>**
  Poner nombre al proyecto, a la organización, etc… **>**
  Seleccionar subdirectorio donde se creará el proyecto

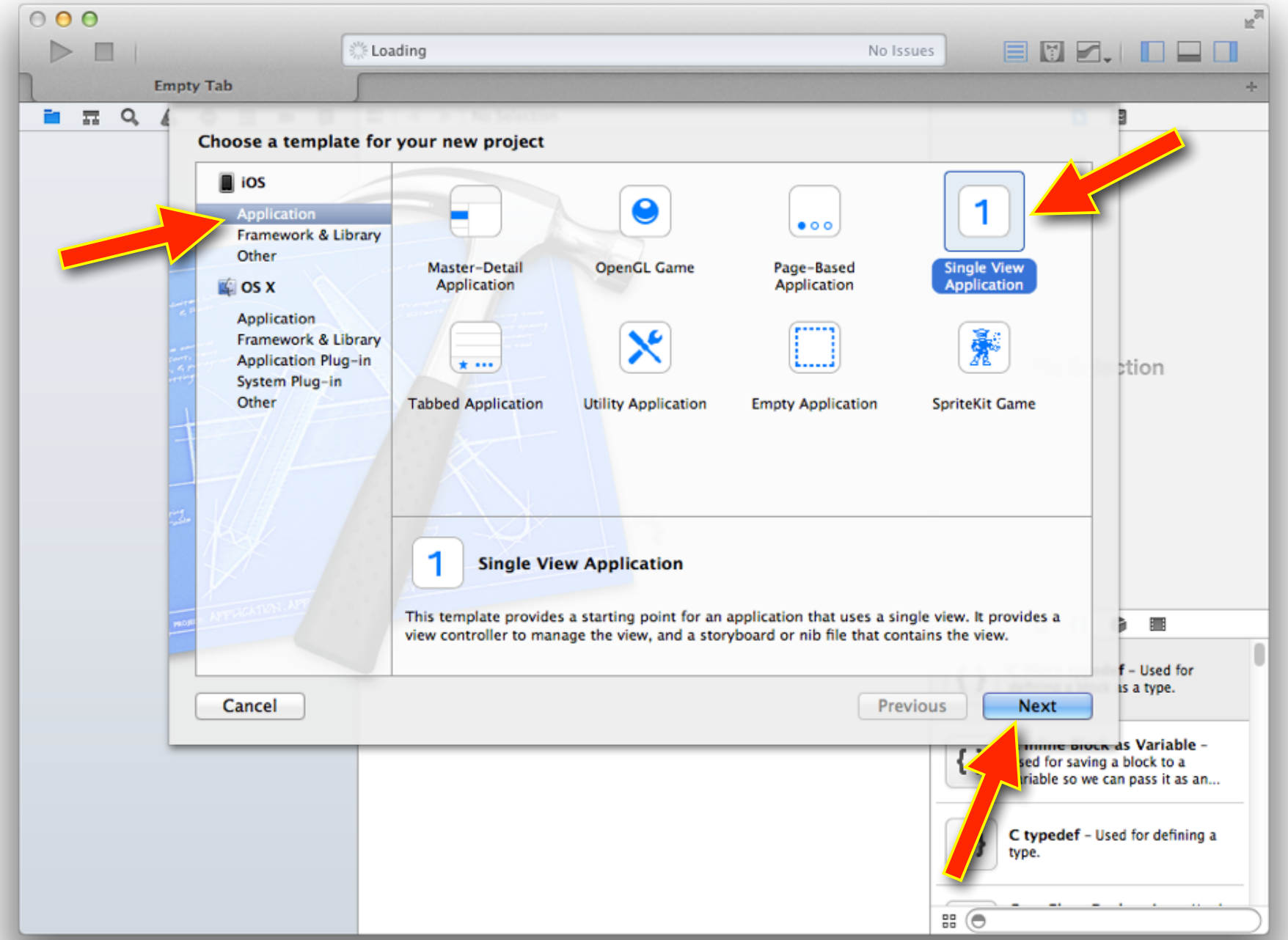
- Se crean los ficheros del proyecto (variará según opciones seleccionadas):

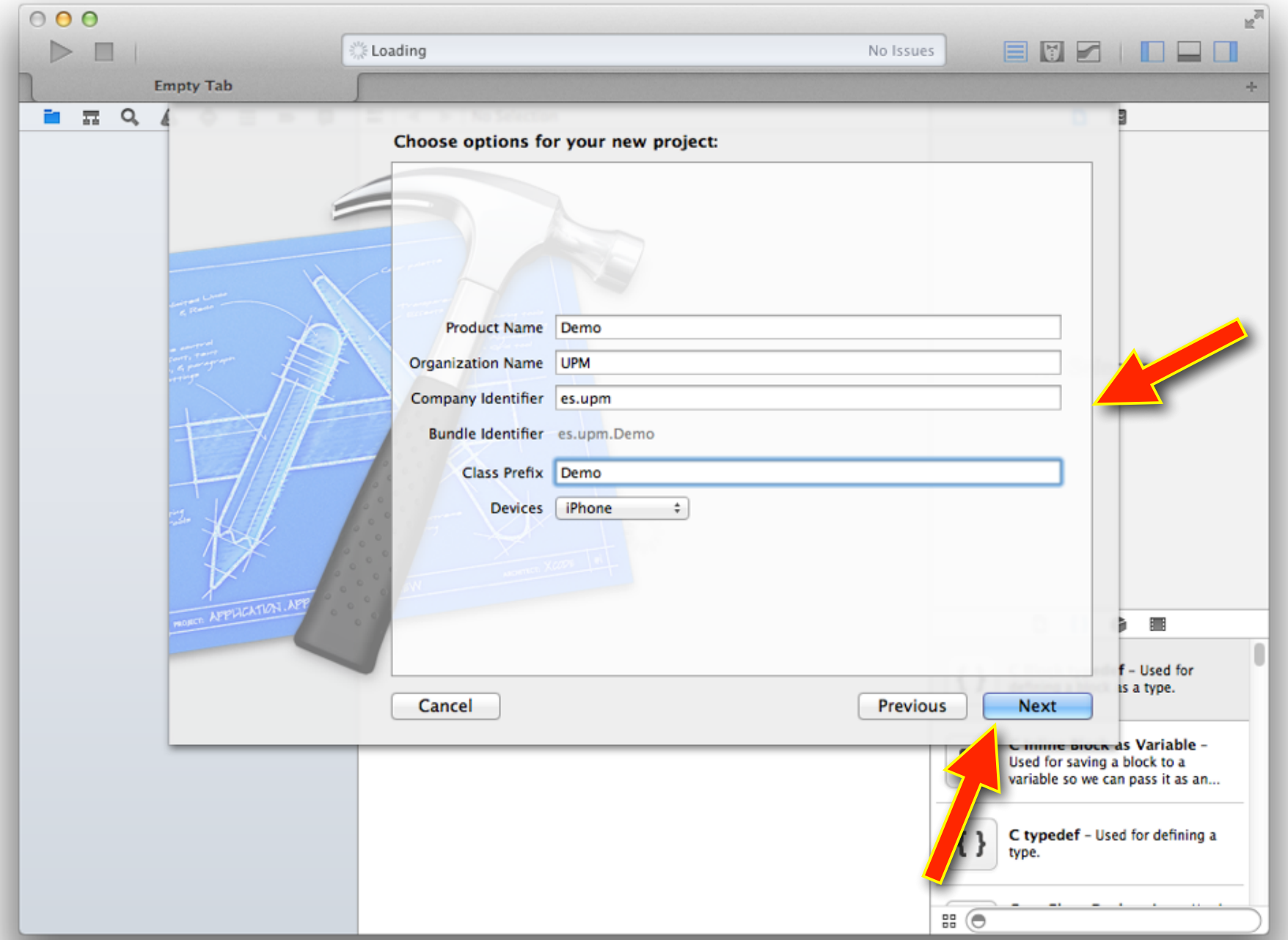
  main.m
  Nombre-**Info.plist**
  *Prefijo***AppDelegate.h** y **.m**
  *Prefijo***ViewController.h** y **.m**
  Main.storyboard
  Images.xcassets
  . . .

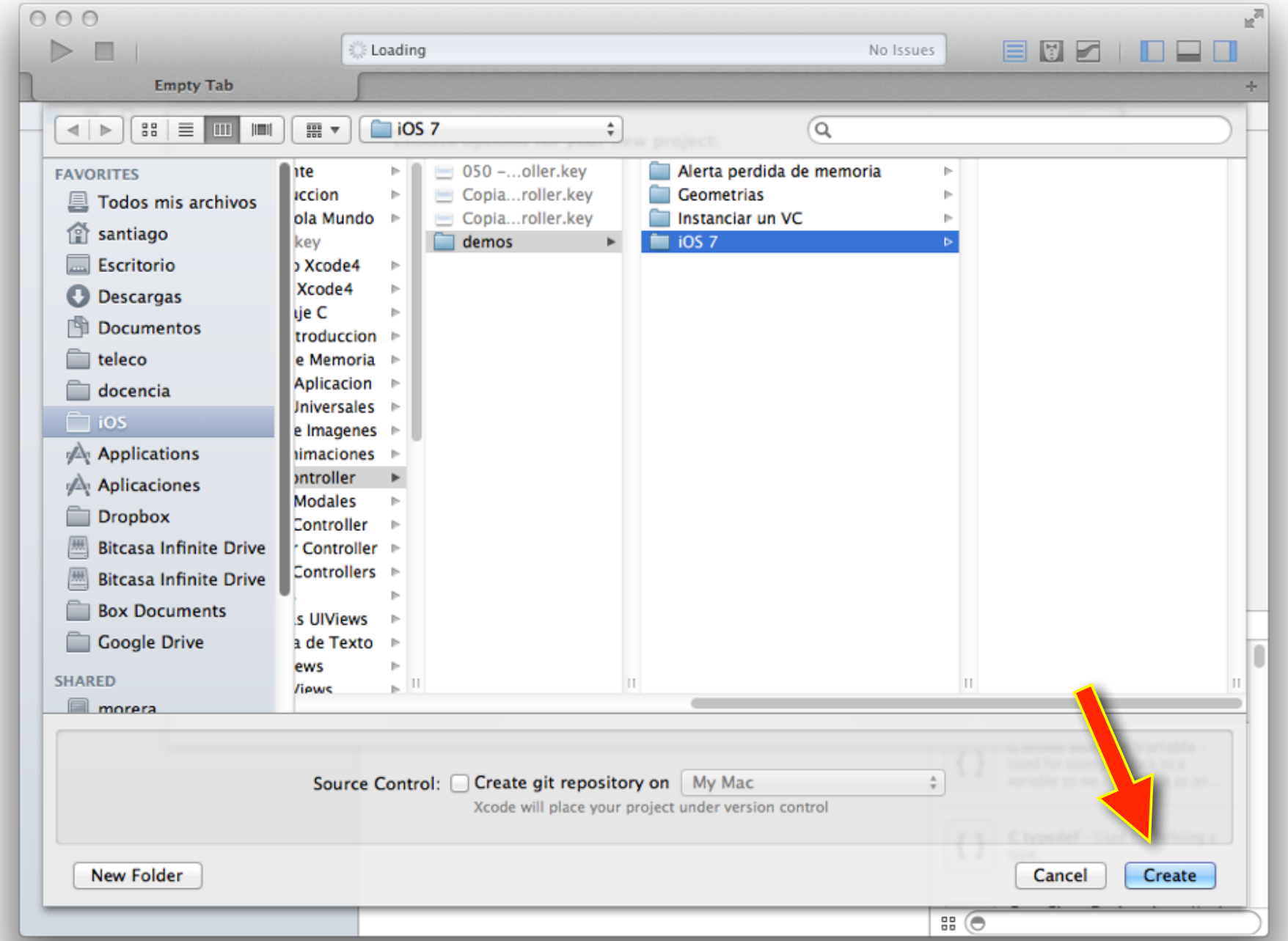
DEMO

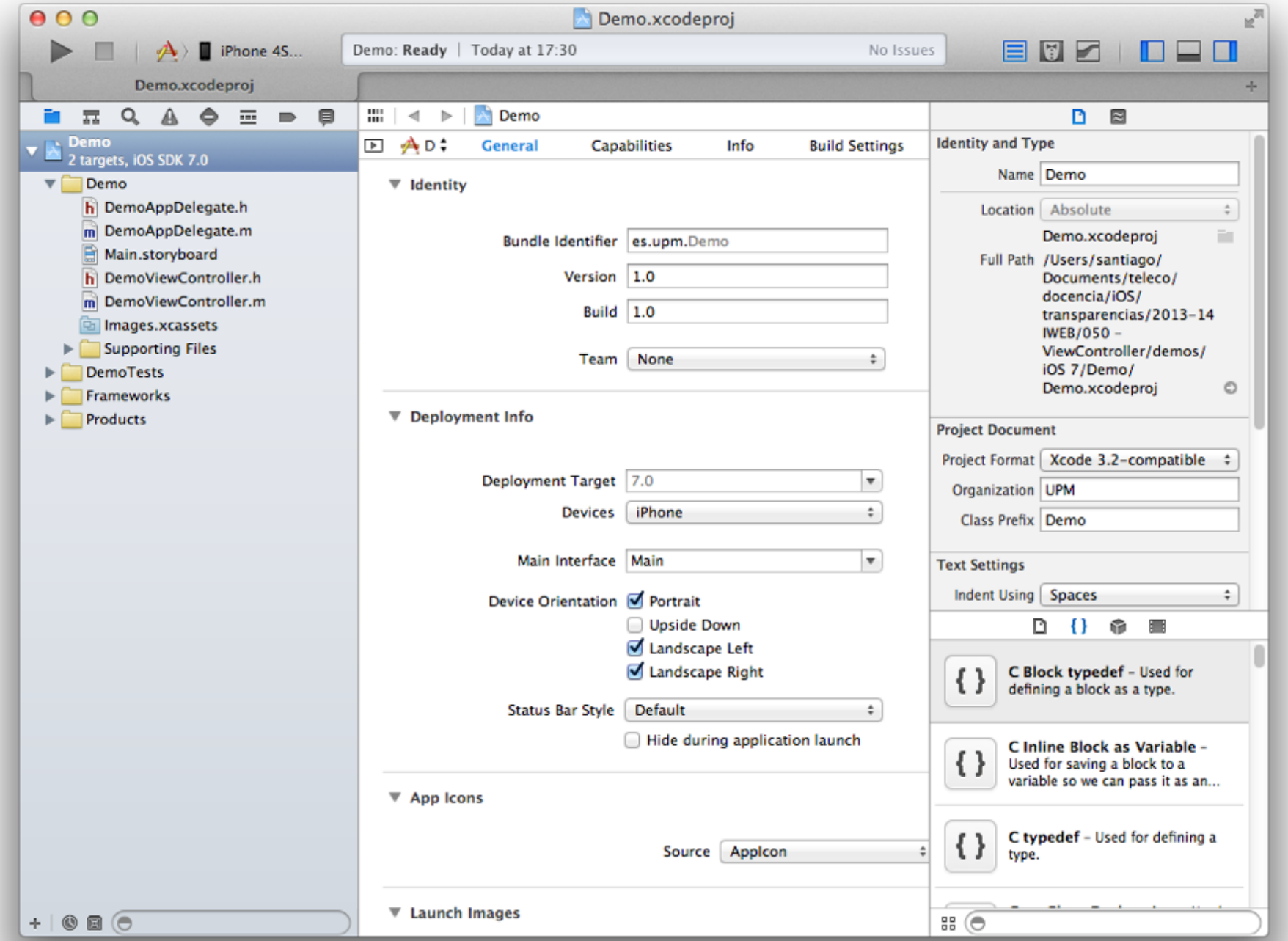
- Crear un proyecto con la plantilla Single View Application llamado Demo

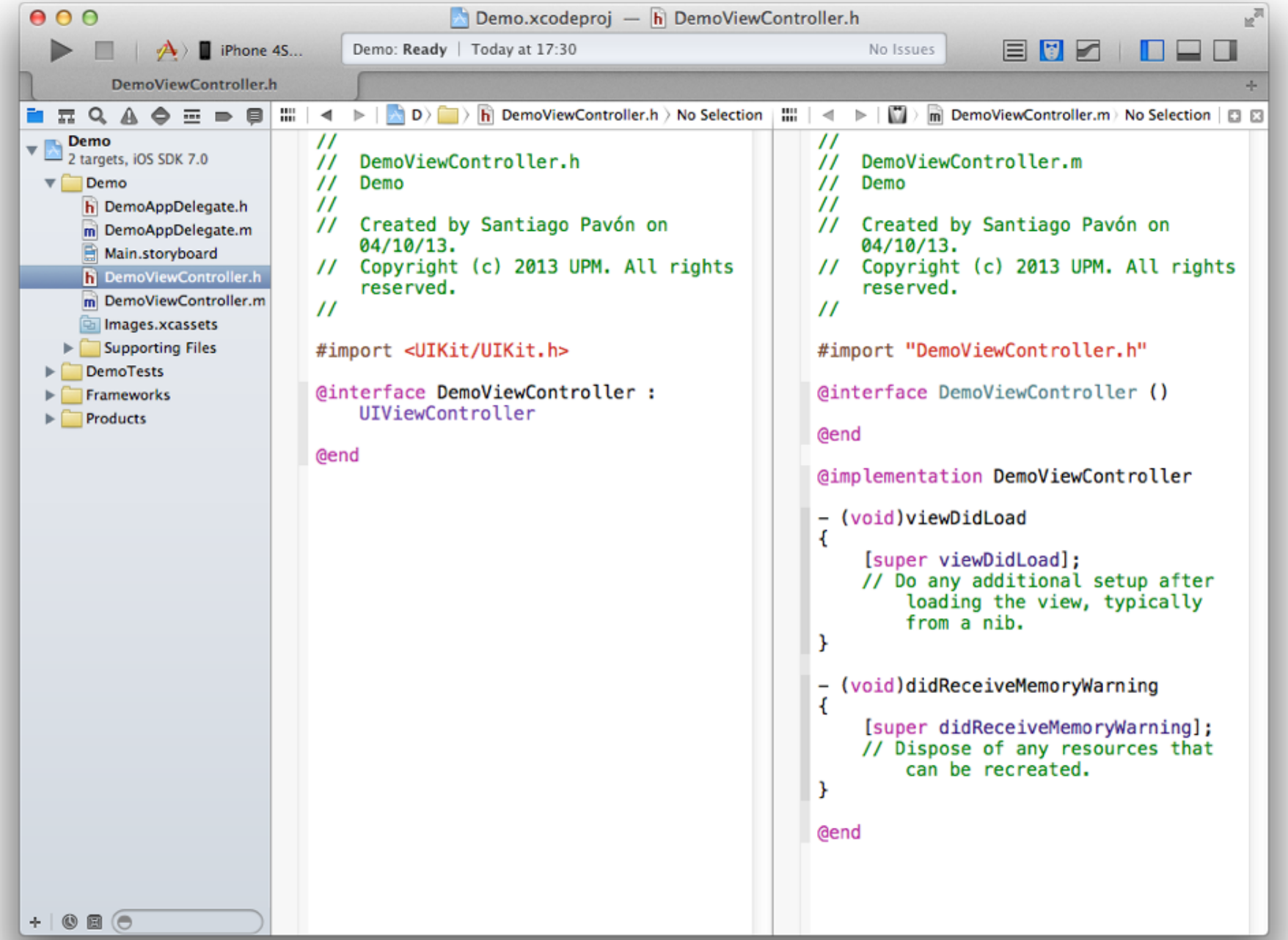


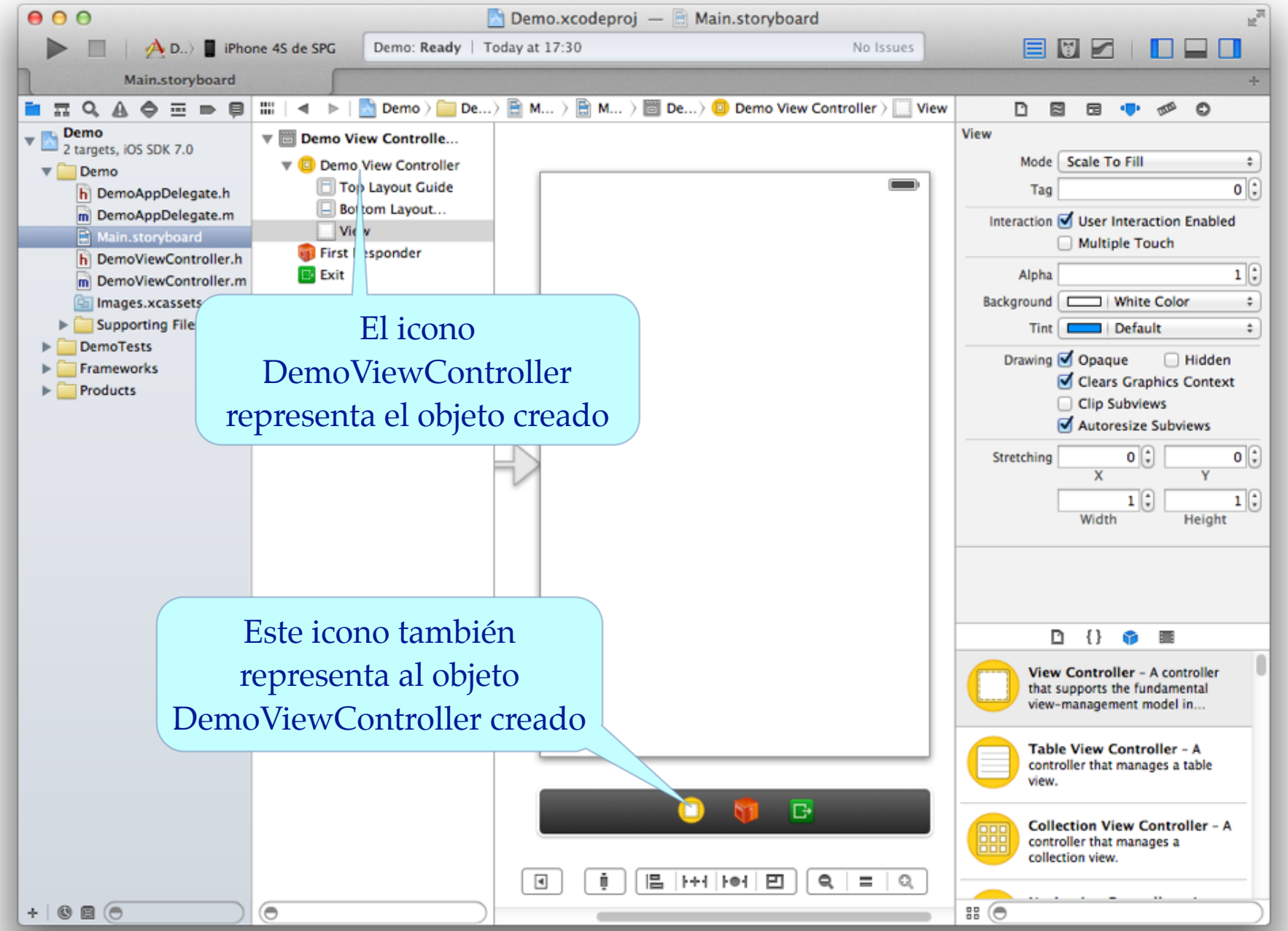


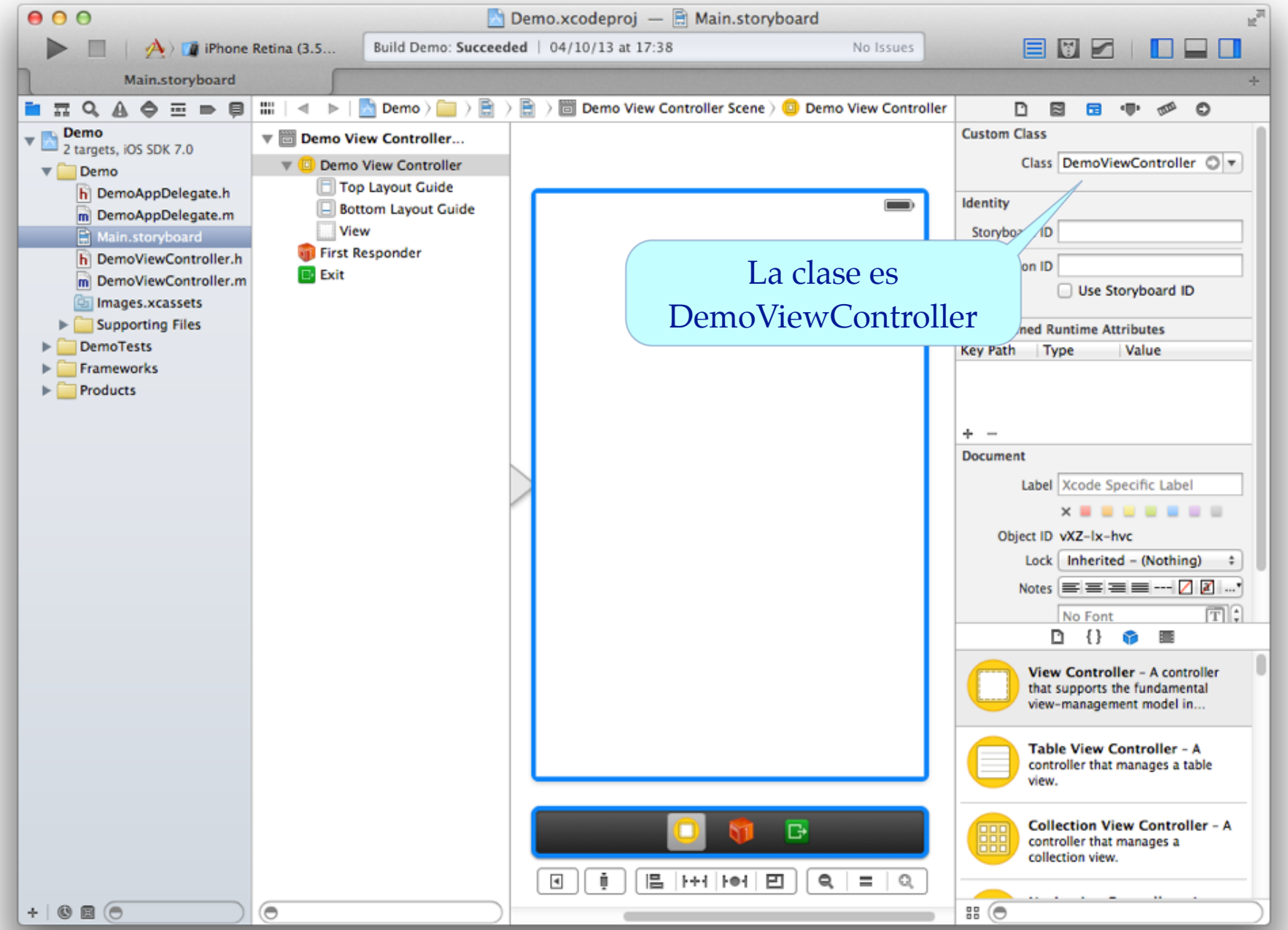


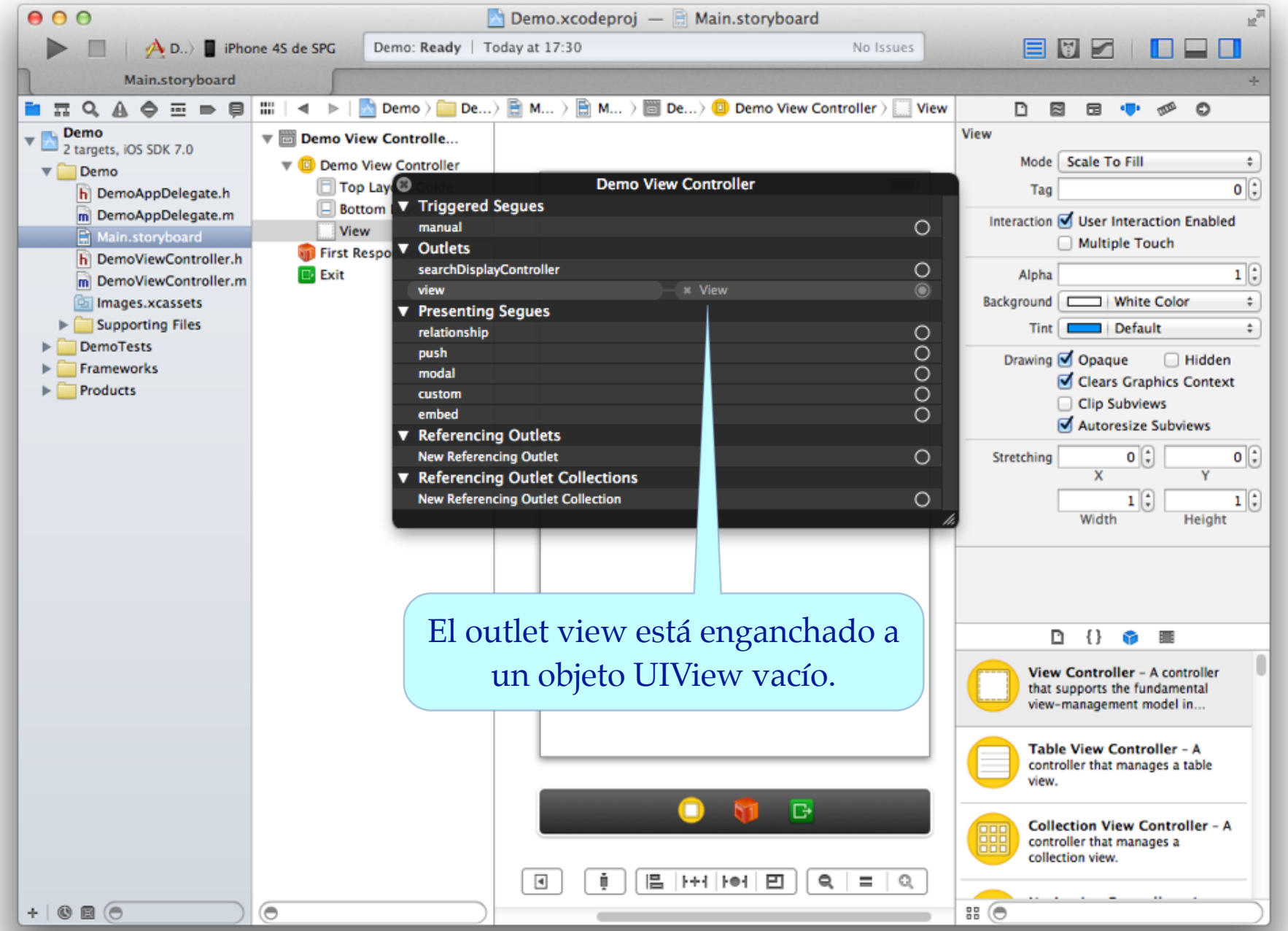












Inicialmente la pantalla no tiene nada.
Solo es una UIView vacía.

Propiedades y Métodos de UIViewController

view

- De la clase base UIViewController heredamos la propiedad **view**.

```
@property (nonatomic,strong) UIView *view;
```

  - Apunta a la raíz de nuestra jerarquía de views.

- Normalmente la jerarquía de views se carga desde un fichero storyboard (o XIB).

- Si no creamos las views con un fichero storyboard (o XIB), el valor de la propiedad **view** es **nil**, y debemos crear el GUI programáticamente.
  - El GUI se crea de forma perezosa:
    - Hasta que no consultamos por primera vez al valor de la propiedad **`self.view`** (método de acceso getter view) no se construye el GUI.
  - Si al acceder a la propiedad **`self.view`** su valor es **nil**, se llama automáticamente al método **`loadView`**.
    - Este método debe crear el GUI programáticamente y asignar un valor a la propiedad **`self.view`**.

- No retener subviews:
  - La propiedad **self.view** retiene a todos los objetos que forman parte de su jerarquía de subviews.
    - los apuntará directamente o a través de una subview intermedia
  - No es necesario que nuestros outlets también retengan a estas subviews.

```
@property (nonatomic,weak) IBOutlet UILabel *label;
```

  - Hay que evitar crear bucles de retenciones.

viewDidLoad

```
-(void) viewDidLoad;
```

- Este método se llama cuando ya se ha cargado en memoria la view (y sus subviews) y los outlets han sido enganchados.
  - tanto si la view se cargó desde un storyboard/XIB o se creó en loadView.
- Este método se usa típicamente para hacer inicializaciones que no pueden hacerse hasta que vista ya se ha cargado.
- Cuando se invoca este método, aun no se ha calculado la geometría de la vista.
  - En este método no pueden hacerse configuraciones que estén relacionadas con la geometría de las vistas.

- Ejemplo:
  - Crear un VC que muestre en una `UILabel` el texto guardado en una propiedad.

    - En el método de inicialización (`init`???) del VC no se puede asignar el texto de la `UILabel`.
      - Ya que la `UILabel` aun no existe. La vista no se ha cargado aun.

    - Sin embargo, el texto de la `UILabel` si puede ponerse en el método `viewDidLoad`.
      - Cuando se llama a este método, la vista y sus subvistas ya han sido cargadas.
        - Sobreescribiremos el método `viewDidLoad` para poner el valor de la propiedad como texto de la etiqueta.

```
- (void)viewDidLoad {

    [super viewDidLoad];

    self.unaLabelDeLaVista.text = self.msg;
}
```

Muy importante:
No olvidar llamar al padre.

El texto a poner en la etiqueta está guardado en una propiedad.

unaLabelDeLaVista que se carga desde un storyboard.
Al crear el objeto VC esta propiedad es nil.
Cuando se llama a **viewDidLoad**, la vista ya ha sido cargada, y **unaLabelDeLaVista** ya no es nil. Ahora es cuando se puede poner el texto de la etiqueta.

awakeFromNib

- Este método se hereda de **NSObject**.
  - Lo crea una categoría sobre NSObject.
- Se usa para hacer configuraciones **después de haber cargado** un fichero XIB o una escena de un Storyboard.
- Los VC lo implementan para realizar configuraciones que no pueden hacerse hasta que se han creado **todos** los objetos definidos en un fichero Interface Builder.
  - Se invoca en todos los objetos creados al cargar un fichero XIB o una escena de un Storyboard.
  - Se invoca después de que todos los objetos del fichero XIB o escena del Storyboard ya han sido creados, y todos los outlets y conexiones se han realizado.
- No olvidar llamar a la versión de este método tapada de la superclase.

Cambios en la Visibilidad del VC

- Avisos indicando que el objeto ViewController va a hacerse visible, que ya es visible, que va a ocultarse, o que ya se ha ocultado.

```
-(void) viewWillAppear:(BOOL)animated;
-(void) viewDidAppear:(BOOL)animated;
-(void) viewWillDisappear:(BOOL)animated;
-(void) viewDidDisappear:(BOOL)animated;
```

- Sobreescribir estos métodos si queremos hacer algo en estos instantes.
  - Por ejemplo, gestionar la persistencia de algún valor, refrescar el dato mostrado por alguna view, ...
- Estos métodos se llaman cada vez que cambia la visibilidad del VC.
  - Ocurrirá con frecuencia en las aplicaciones que tienen varias pantallas.
    - Cuando la aplicación termina no se llama a XXXDisappear.

Geometría de la Vistas

- Cuando se invoca a **viewDidLoad** aun no se ha calculado la geometría de las vistas.
  - El valor de las propiedades **bounds** y **frame** no se ha calculado.
  - Dentro de viewDidLoad no podemos hacer cambios que dependan de las geometrías.
- Cuando se invoca a **viewWillAppear** puede que la geometría de las vistas tampoco se hayan calculado aun..
- Cuando cambia el valor de la propiedad **bounds** o **frame** de una vista, ésta reajusta la posición de todas sus subviews.
  - Pero antes de reposicionar las subviews se llama al método:
    - `- (void) viewWillLayoutSubviews;`
  - Y después de reposicionar las subviews se llama al método:
    - `- (void) viewDidLayoutSubviews;`
  - Redefinir estos métodos cuando queramos hacer cambios relacionados con la geometría de las vistas.
    - Por ejemplo: rotación de la pantalla.

Orientación del Terminal

- La propiedad **`interfaceOrientation`** indica cual es la orientación actual del terminal.

```
UIInterfaceOrientationPortrait
UIInterfaceOrientationPortraitUpsideDown
UIInterfaceOrientationLandscapeLeft
UIInterfaceOrientationLandscapeRight
```

- Las orientaciones soportadas se indican en **Info.plist**, pero un VC puede modificar estos valores sobreescribiendo el método

```
-(NSUInteger)supportedInterfaceOrientations;
```

- Para indicar cual es la orientación preferida para presentar inicialmente el VC hay que sobreescribir el método:

```
-(UIInterfaceOrientation)preferredInterfaceOrientationForPresentation;
```

- Para indicar si el contenido de un VC debe rotar hay que sobreescribir el método:

```
-(BOOL)shouldAutorotate;
```

- Métodos que se llaman antes y después de una rotación:
  - `willRotateToInterfaceOrientation:duration:`
  - `willAnimateRotationToInterfaceOrientation:duration:`
  - `didRotateFromInterfaceOrientation:`

    - Estos métodos no suelen usarse.
      - Normalmente se usa **Autolayout** y `viewWillLayoutSubviews`.

Notas

ANTIGUO:

- viewDidUnload ha sido deprecado en iOS6.
  - También ha sido deprecado viewWillUnload.

- Este método se invocaba cuando había problemas de falta de memoria en la aplicación.
  - Se invocaba en los VC que no estaban visibles en ese momento para que liberarán toda la memoria que pudieran.
  - Si el VC se hacía visible otra vez en el futuro, se volvía a llamar a viewDidLoad que regeneraría los datos liberados.
  - En viewDidUnload debía liberarse todo aquello que pudiera reconstruirse más tarde en viewDidLoad.

- Ahora las liberaciones de memoria se hacen en:
  - didReceiveMemoryWarning.

didReceiveMemoryWarning

- Se llama cuando hay problemas de falta de memoria.
- Solo debe liberarse memoria de un VC cuando **no se esté mostrando** en la pantalla.

```
- (void) didReceiveMemoryWarning {

    [super didReceiveMemoryWarning];

    if ([self isViewLoaded] && self.view.window == nil) {

        // Liberar jerarquia de views:
        // self.view = nil;

        // Liberar memoria que pueda regenerarse en viewDidLoad:
        mi_cache = nil; // es una ivar de mi app
    }
}
```

Sentencia comentada. Ver explicación en la siguiente transparencia.

- Aclaraciones sobre el ejemplo anterior:

- Primero debe llamarse a la versión del método tapada en el padre.

```
[super didReceiveMemoryWarning];
```

- La sentencia **if** comprueba que el VC no sea visible en la pantalla en este momento.

  - Primero hay que comprobar que **self.view** esté cargada:

    ```
    [self isViewLoaded]
    ```

    - Esta comprobación es necesaria porque si **self.view** no está cargada, se cargaría automáticamente de nuevo al acceder a **self.view** al evaluar la expresión **self.view.window**.

  - Un VC no está visible en pantalla si la propiedad **window** de su propiedad **self.view** es nil.

    ```
    self.view.window == nil.
    ```

- Si la condición del **if** se cumple, **self.view** y sus subviews pueden descargarse asignando:

```
self.view = nil;
```

  - Apple recomienda no solucionar los problemas de falta de memoria destruyendo la jerarquía de views.
    - El ahorro de memoria suele ser mínimo y es una fuente de errores en las apps porque para el programador es más dificil desarrollar el código,
      - sobre todo cuando se usan VC que contienen a otros VC.
  - Siguiendo la recomendación de Apple, la sentencia **`self.view = nil;`** está comentada en el ejemplo.

- Dentro del **if** se liberan todos aquellos objetos que puedan regenerarse en un futuro en caso de necesidad.
  - Si este VC vuelve a hacerse visible, se llamará otra vez a **viewDidLoad**, donde se regenerarían los objetos liberados aquí.

dealloc

- Este método libera la memoria del objeto.
  - **No lo llamamos nunca**: lo llama el sistema cuando lo considere oportuno.
- Con **ARC**:
  - No suele ser necesario reescribir este método.
  - Sólo si hemos creado recursos que no están bajo el control de ARC.
- Con **MRC**:
  - El sistema lo llama cuando todos sus propietarios han enviado release y la cuenta de retenciones del objeto llegó a cero.
  - Hay que sobreescribir el método dealloc para:
    - enviar release a los objetos que esté reteniendo.
    - Ejecutar dealloc para liberar la memoria del propio objeto.

Solo para MRC

```
- (void)dealloc {

    [outlet1 release];
    [outlet2 release];

    [propiedad release];

    [super dealloc];
}
```

Libero mis propiedades.
Disminuyo el contador de retenciones.
Los que lleguen a cero liberarán su memoria.

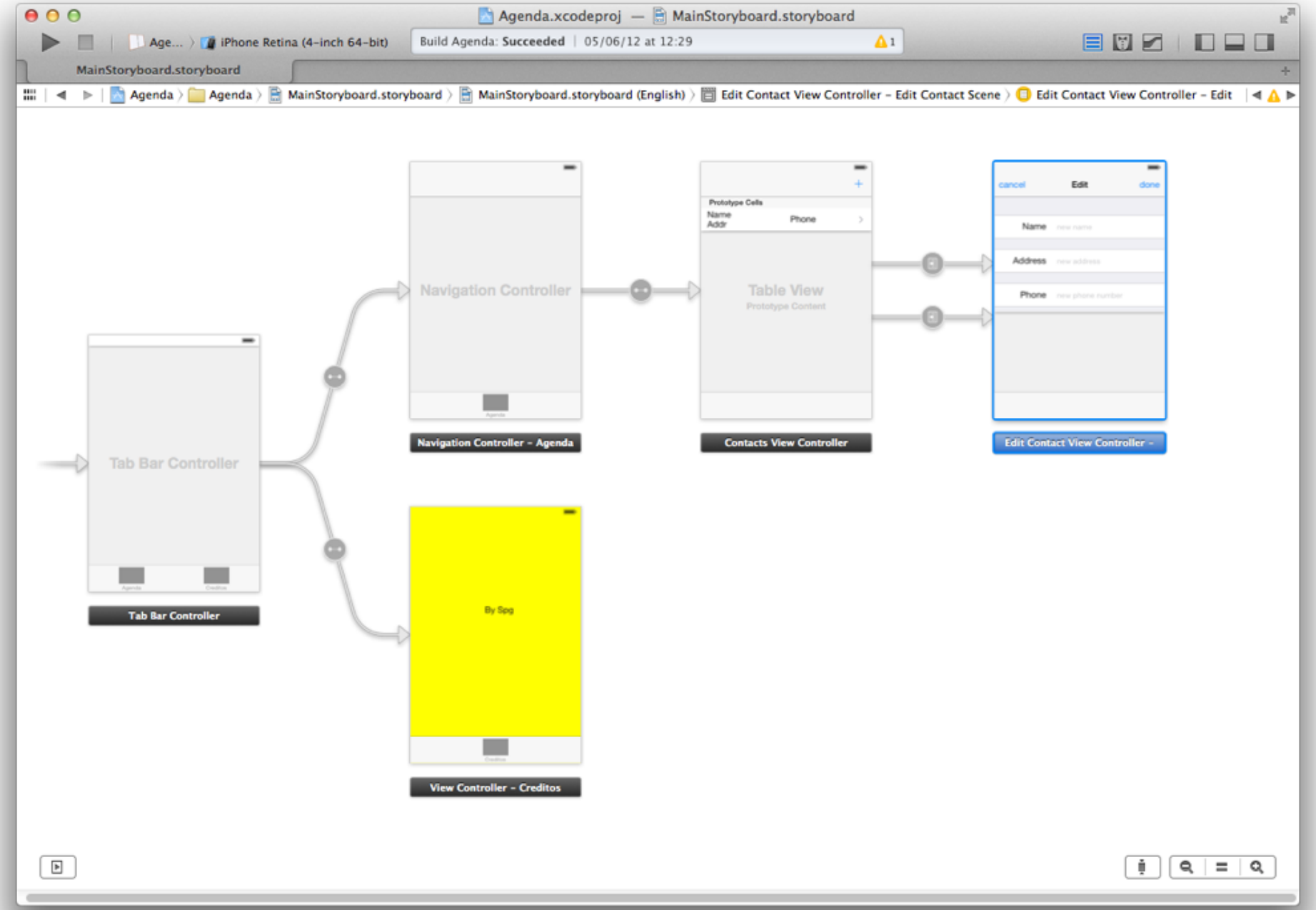
Libero mi memoria.

Es el único sitio donde se puede llamar a dealloc.

IMPORTANTE: Es la última sentencia.
No quiero hacer release de las propiedades de un objeto al que ya se le ha hecho dealloc.

Crear VC usando Storyboard

- Lo más normal es que las aplicaciones usen un storyboard para diseñar sus pantallas o escenas,
  - y las conecten y relacionen entre sí usando segues.
- Estas pantallas o escenas serán:
  - objetos View Controller y
  - controladores de navegación.
- Los objetos VC diseñados en el storyboard se instanciarán **automáticamente** cuando se necesiten.
  - Los segues existentes indican que VC deben instanciarse.
  - y sólo necesitamos configurarlos en **prepareForSegue:sender:**.
- No es muy normal crear los objetos VC programáticamente.



- Para crear **programáticamente** un VC definido en un fichero storyboard:

  - Primero hay que obtener el objeto storyboard.
    - Puede obtenerse:
      - desde un VC ya existente accediendo a su propiedad **storyboard**.
      - creándolo desde un fichero **.storyboard** con el siguiente método de la clase **UIStoryboard**:

```
+ (UIStoryboard*)storyboardWithName:(NSString *)name
                             bundle:(NSBundle *)bundleOrNil
```

  - Segundo, al objeto storyboard que hemos obtenido en el punto anterior, le pedimos que instancie un objeto VC usando el método:

```
- (id)instantiateViewControllerWithIdentifier:(NSString*)id
```

    - El parámetro indica cual todos los VC definidos en el storyboard es el que queremos instanciar.
      - Estos identificadores se crean con el inspector de atributos del Interface Builder.

    - También podemos instanciar el VC inicial del storyboard con este método:

```
- (id)instantiateInitialViewController
```

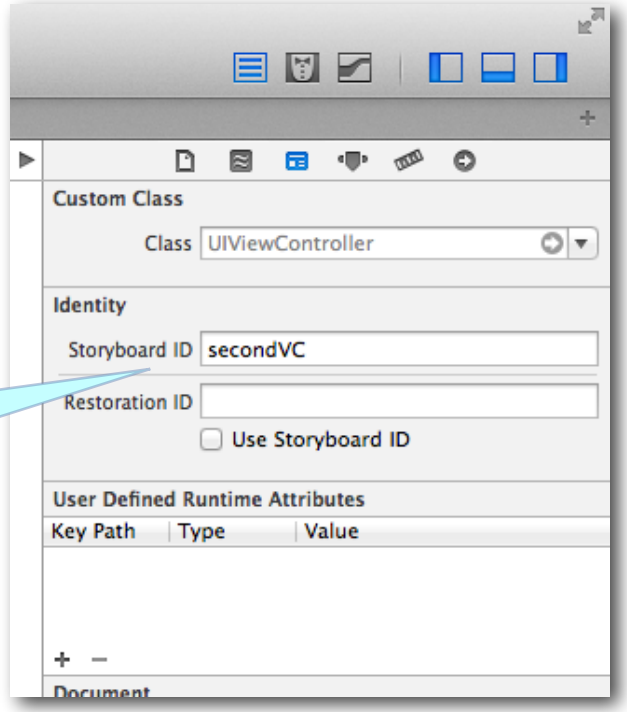
Instanciar programáticamente un VC desde un storyboard:

```
UIStoryboard * storyboard = self.storyboard;
NSString * id = @"secondVC";

UIViewController *vc =
     [storyboard instantiateViewControllerWithIdentifier:id];

[self presentModalViewController:vc animated:YES];
```

El VC actual se creó desde el storyboard apuntado por esta propiedad.

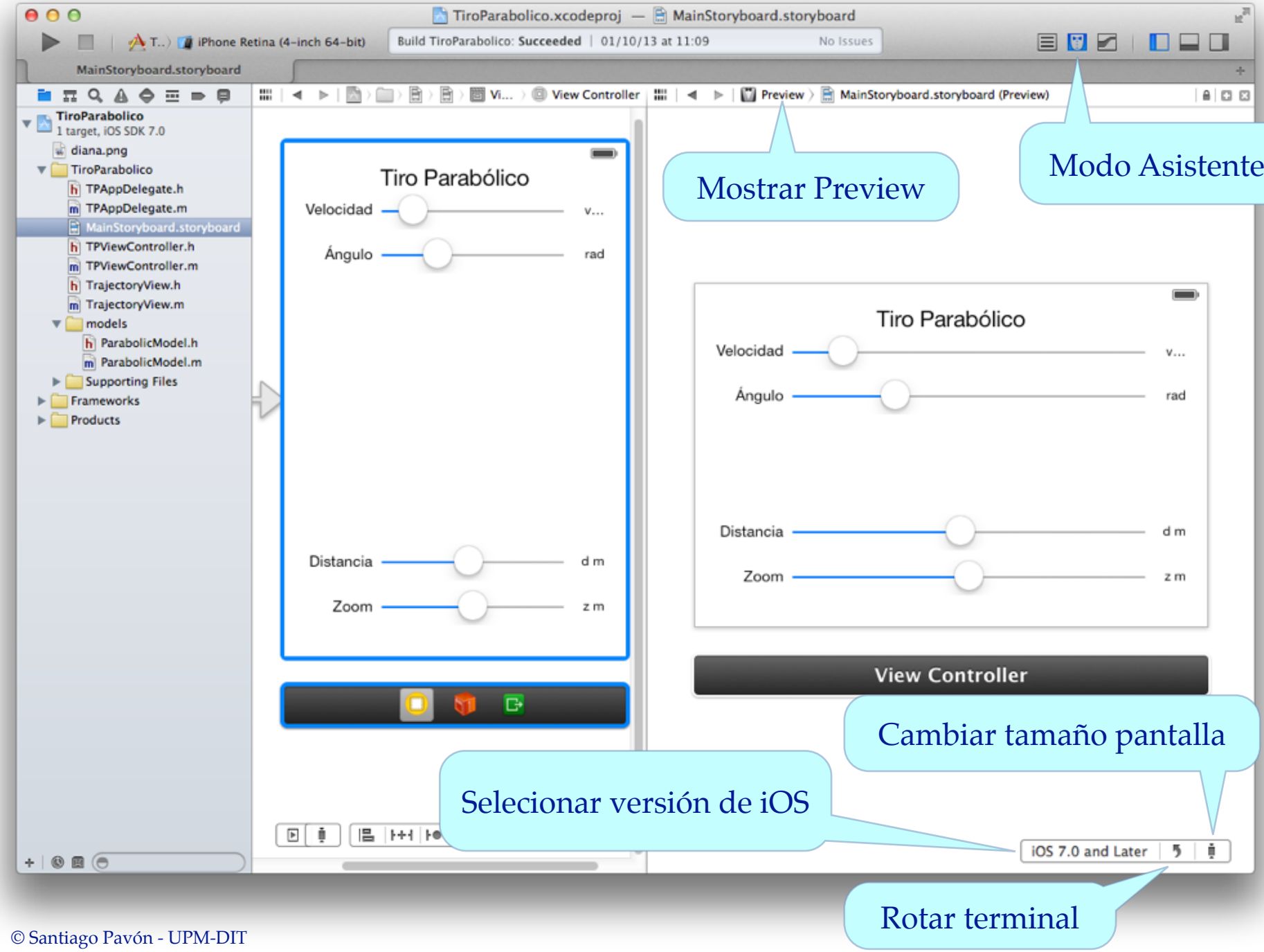
Desde el VC actual muestro el VC recién creado de forma modal

Identificador del VC



Previsualización de Pantallas

- Con el Interface Builder podemos previsualizar cómo quedan las pantallas (*los View Controles*) diseñados para distintas versiones de iOS, tamaños de pantalla y orientaciones del terminal.



Crear VC usando XIB

- Para crear programáticamente una instancia VC que cargue un fichero XIB:
  - primero pedimos memoria con alloc
  - y después inicializamos con el método:

```
-(id)initWithNibName:(NSString *)nibName
              bundle:(NSBundle *)aBundle;
```

  - Usa el XIB y el bundle especificado.

Instanciar programáticamente un VC usando un XIB:

```
@interface SubvistaModal : UIViewController
```

```
SubvistaModal* m = [[SubvistaModal alloc]
                        initWithNibName:@"SubvistaModal"
                                 bundle:nil];

m.modalTransitionStyle = UIModalTransitionStyleFlipHorizontal;
[self presentModalViewController:m animated:YES];
```

- El método **`initWithNibName:bundle:`** puede sobreescribirse.
  - Por ejemplo, cuando queremos inicializar algunos atributos del objeto desde el mismo momento en que se crea.

```
- (id)initWithNibName:(NSString *)nibNameOrNil
                bundle:(NSBundle *)nibBundleOrNil {

    if (self = [super initWithNibName:nibNameOrNil
                                bundle:nibBundleOrNil]) {
        // Custom initialization
        _code = 666;
        _name = @"Lucifer";
    }
    return self;
}
```

  - Recordad que todo se hace de forma perezosa:
    - la carga del XIB/Storyboard/**loadView** y la invocación a **viewDidLoad** se puede retrasar bastante.

Relaciones entre VC

Aplicaciones complejas

- Muchas aplicaciones están formadas por varias pantallas.
  - Cada pantalla será un VC que proporcionará una determinada funcionalidad al usuario.
  - Se mostrará una pantalla u otra según las acciones realizadas.
- ¿Cómo se crean aplicaciones con varias pantallas?

Controladores de ViewControllers

- Existen VC controladores que manejan otros VC:
  - Crean su vista usando las vistas de otros VC.

➡**`UINavigationController`**

- Maneja una pila de VC.

➡**`UITabBarController`**

- Selección de VC independientes usando pestañas.

➡**`UISplitViewController`**

- VC maestro que controla los detalles mostrados en otro VC.

- etc . . .

ViewControllers Modales

- Un ViewController puede mostrar de forma modal otro VC.
  - En iPhone los VC modales ocupan toda la pantalla
  - En iPad pueden mostrarse con diferentes estilos.

Relaciones entre Controladores

- Los objetos ViewController poseen propiedades para acceder a los VC con los que están relacionados

```
tabBarController
navigationController
parentViewController
presentingViewController
presentedViewController
splitViewController
searchDisplayController
```