Persistencia de datos en iOS: User Defaults y Core Data

Índice

1. Introducción

En esta sesión veremos como utilizar dos nuevos métodos de persistencia en iOS, ambos muy interesantes y usados muy frecuentemente:

- **User Defaults**: Muy útil para almacenar datos simples y de uso repetido dentro de nuestras aplicaciones iOS.
- **Core Data**: Método avanzado de almacenamiento de objetos enteros dentro de la memoria del dispositivo iOS. Muy útil para almacenar datos complejos, con relaciones, etc.

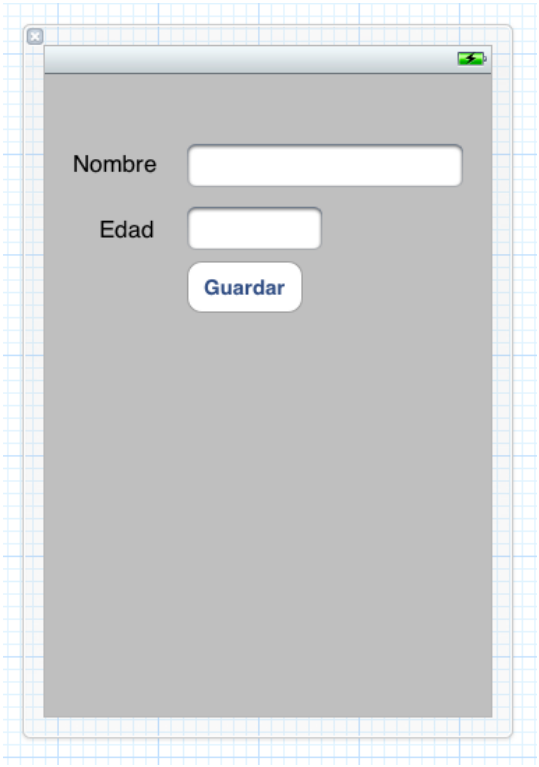
2. User Defaults

Este método es muy útil cuando queremos almacenar pequeñas cantidades de datos como puntuaciones, información de usuario, el estado de nuestra aplicación, configuraciones, etc.

Este método es el más rápido de usar ya que no requiere de una base de datos, ni ficheros, ni ninguna otra capa intermedia. Los datos se almacenan directamente en la memoria del dispositivo dentro de un objeto de la clase `NSUserDefaults`.

Para aprender a usar este método de almacenamiento de datos vamos a crear una aplicación muy simple que lea un nombre y una edad de pantalla y las guarde en un objeto `NSUserDefaults`. Para ello seguimos los siguientes pasos:

- 1) Comenzamos creando un nuevo proyecto llamado `"sesion04-ejemplo1"` dentro de xCode.
- 2) Abrimos la vista `sesion04_ejemplo1ViewController` y arrastramos dos *Text Field*, un botón y dos labels quedando la vista de la siguiente manera:



Vista de la aplicación

- 3) Abrimos el fichero `sesion04_ejemplo1ViewController.h` y definimos el Outlet del evento onclick del botón. El fichero quedará de la siguiente manera:

```
//imports iniciales

        @interface sesion04_ejemplo1ViewController : UIViewController {
            UITextField *tfNombre;
            UITextField *tfEdad;
        }

        @property(nonatomic, retain) IBOutlet UITextField *tfNombre;
        @property(nonatomic, retain) IBOutlet UITextField *tfEdad;

        -(IBAction) guardaDatos;

        @end
```

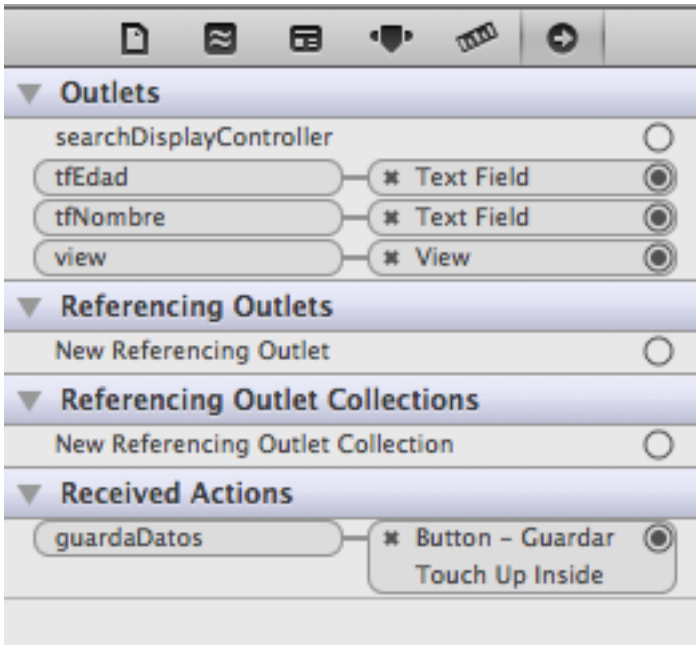
- 4) Ahora implementamos el código para el método del botón dentro de `sesion04_ejemplo1ViewController.m`:

```
        @synthesize tfEdad;
        @synthesize tfNombre;
```

```
        -(IBAction) guardaDatos {
            NSLog(@"Guardamos los datos...");
```

```
        // creamos el objeto NSUserDefaults
        NSUserDefaults *datos = [NSUserDefaults standardUserDefaults];

        // guardamos el nombre (NSString)
        [datos setObject:tfNombre.text forKey:@"nombre"];

        // guardamos la edad (Integer)
        [datos setInteger:[tfEdad.text integerValue] forKey:@"edad"];

        // almacenamos los datos en la memoria
        [datos synchronize];
    }
```

- 5) Tenemos que enlazar el botón con el evento que hemos programado en el paso anterior, para ello abrimos la vista `sesion04_ejemplo1ViewController.xib` y con la tecla *ctlr* apretada, arrastramos desde el botón hasta `File's Owner` (columna de la izquierda, parte superior) y seleccionamos el método que hemos creado (`guardaDatos`). De esta forma ya tenemos "conectado" el botón.
- 6) Ahora hacemos lo mismo con los *Text Field*. Tenemos que "conectarlos" al controlador, para ello seleccionamos el objeto `"File's Owner"` y abrimos la pestaña de conexiones (la que está más a la derecha en la columna de la derecha). Hacemos click en `tfEdad` y arrastramos hasta el *Text Field* de edad. Hacemos lo mismo para el *text field* de nombre. Ya tenemos todos los elementos de la vista conectados.



Conexión de Outlets

- 7) Arrancamos la aplicación y la probamos:

Aplicación

- 8) Ahora vamos a hacer que la aplicación cargue los datos que se han guardado al arrancar la vista, para ello simplemente tenemos que escribir el siguiente código en el método `(void)viewDidLoad` de la clase `sesion04_ejemplo1ViewController.m`:

```
- (void)viewDidLoad
{
    [super viewDidLoad];

    NSUserDefaults *datos = [NSUserDefaults standardUserDefaults];

    // obtenemos un NSString
    NSString *nombre = [datos stringForKey:@"nombre"];

    // obtenemos un NSInteger
    NSInteger edad = [datos integerForKey:@"edad"];

    if (nombre!=nil && edad!=0){
        NSLog(@"Nombre cargado: %@, edad: %d",nombre,edad);
    }
    else {
        NSLog(@"Sin datos!");
    }
}
```

- 9) Ya podemos arrancar de nuevo la aplicación y veremos que si escribimos cualquier

texto en los *Text Fields*, estos se almacenarán dentro del objeto `NSUserDefaults`. Al apagar la aplicación y volver a arrancarla, estos datos se cargarán (aparecerán en consola).

```
2011-09-02 18:18:38.451 sesion04-ejemplo1[4730:207] Nombre cargado: Javier, edad: 28
```

Consola

Nota

Los datos permanecerán en nuestro dispositivo hasta que se elimine la aplicación, en ese caso todos los datos guardados en `NSUserDefaults` se borrarán.

- 10) Probar eliminar la aplicación del dispositivo/simulador y volver arrancar. Comprobamos que los datos se han borrado:

```
2011-09-02 18:16:28.020 sesion04-ejemplo1[4701:207] Sin datos!
```

Consola

3. Core Data

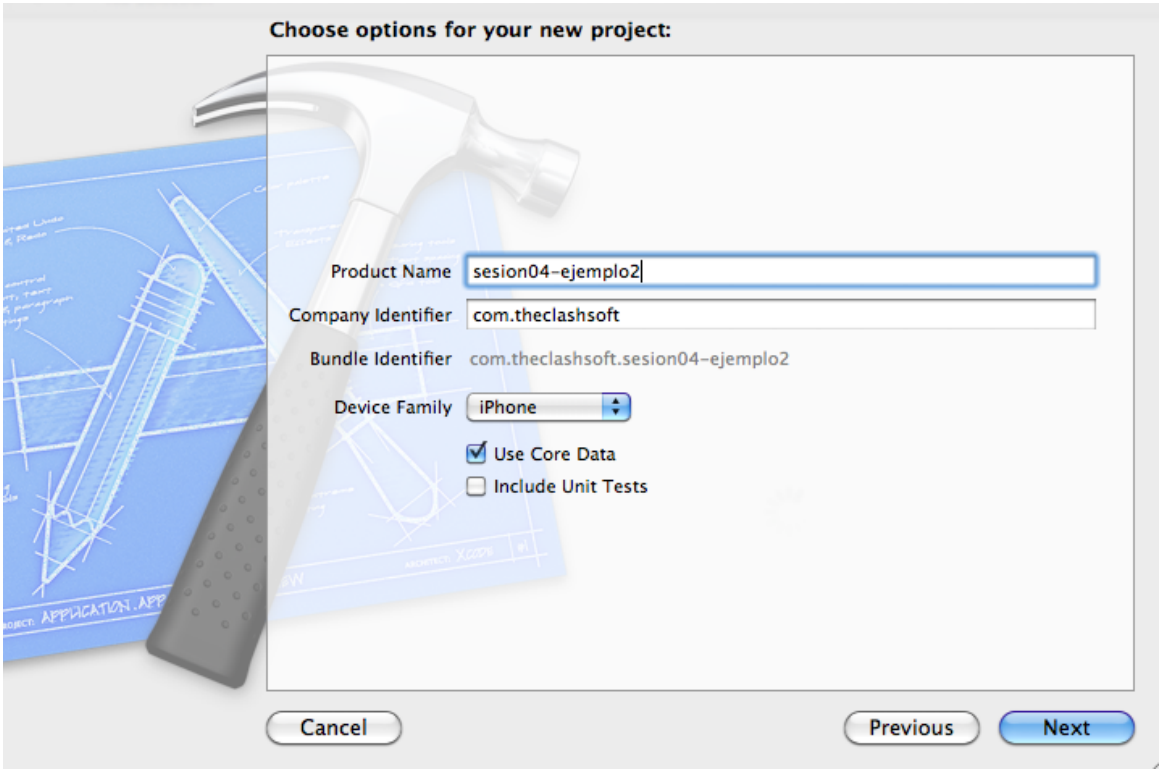
De todas las formas de persistencia de datos que existen en iOS, **Core Data** es la más apropiada para usar en caso de manejar datos no triviales, con relaciones algo más complejas entre ellos. El uso de *Core Data* dentro de nuestra aplicación optimiza al máximo el consumo de memoria, mejora el tiempo de respuesta y reduce la cantidad de código redundante a escribir.

Core Data utiliza de fondo el método de almacenamiento *SQLite* visto en la sesión 3 de este módulo, por tanto para explicar el funcionamiento de este método de persistencia vamos a utilizar el mismo modelo de datos que usamos en esa sesión: *Persona -> DetallePersona*.

El el siguiente ejemplo haremos la misma aplicación que hicimos con *SQLite*, pero usando este nuevo método de persistencia.

3.1. Creando el proyecto

Para empezar tenemos que crearnos un nuevo proyecto en xCode. Para ello abrimos xCode y seleccionamos crear una nueva aplicación basada en ventanas (`Window-based application`). Hacemos click en `"Next"`, escribimos como nombre del proyecto `"sesion04-ejemplo2"` y seleccionamos `"Use Core Data"`.



Creando un proyecto nuevo

Con esto ya tenemos la estructura básica para empezar a programar. Antes de nada hechamos un vistazo rápido a toda la estructura de directorios creada y vemos que dentro del directorio principal tenemos un archivo que se llama `"sesion04_ejemplo2.xcdatamodeld"`. Al hacer click sobre el archivo se nos abre un editor especial que en principio está vacio y que más adelante utilizaremos para diseñar nuestro modelo de datos.

Ahora hechamos un vistazo a la clase delegada `"sesion04_ejemplo2AppDelegate.m"`, vemos que hay unos cuantos métodos nuevos que servirán para configurar el *Core Data*. Vamos a explicar los más importantes:

- `- (NSManagedObjectModel *)managedObjectModel`. NSManagedObjectModel es la clase que contiene la definición de cada uno de los objetos o entidades que almacenamos en la base de datos. Normalmente este método no lo utilizaremos ya que nosotros vamos a utilizar el editor que hemos visto antes, con el que podremos crear nuevas entidades, crear sus atributos y relaciones.
- `- (NSPersistentStoreCoordinator *)persistentStoreCoordinator`. Aquí es donde configuramos los nombres y las ubicaciones de las bases de datos que se usarán para almacenar los objetos. Cuando un *"managed object"* necesite guardar algo pasará por este método.
- `- (NSManagedObjectContext *)managedObjectContext`. Esta clase NSManagedObjectContext será la más usada con diferencia de las tres, y por tanto, la

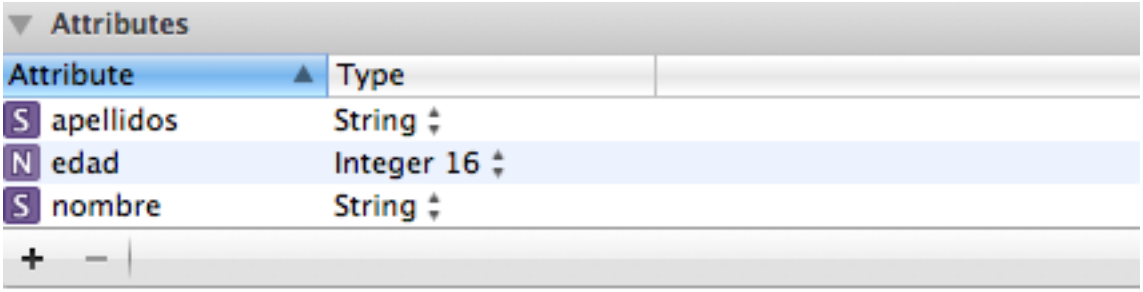
más importante. La utilizaremos básicamente para obtener objetos, insertarlos o borrarlos.

3.2. Definiendo el modelo de datos

En el ejemplo que hicimos en la sesión anterior de *SQLite* creamos dos tablas, una para los datos básicos de la persona y otra para el detalle. Ahora haremos lo mismo pero usando Core Data, la diferencia principal es que mientras que con *SQLite* obteniamos sólo los datos que necesitábamos de la base de datos, aquí obtenemos el objeto completo.

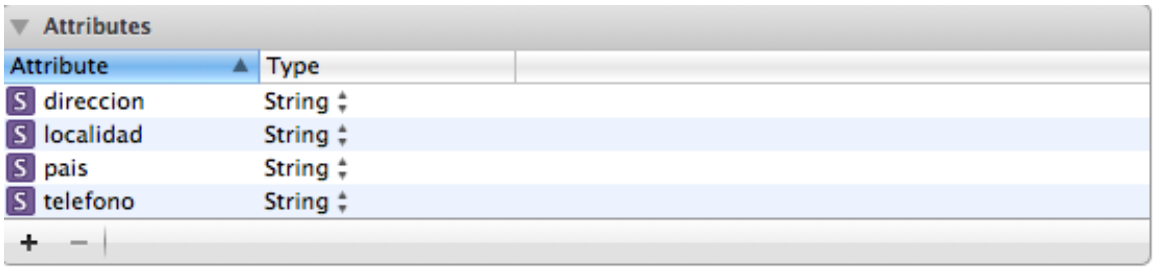
Para definir nuestro modelo de datos basado en dos objetos (Persona y DetallePersona) abrimos el editor visual haciendo click sobre el fichero `sesion04_ejemplo2.xcdatamodeld`. Empezaremos creando una nueva entidad, para ello hacemos click sobre el botón `"Add Entity (+)"` que se encuentra en la parte inferior de la pantalla. A la nueva entidad le pondremos como nombre `"Persona"`.

Dentro de la sección de `"Attributes"` hacemos click sobre `(+)` y creamos un nuevo atributo para nuestra entidad, le llamamos `"nombre"` y le asignamos como tipo `"String"`. Realizamos este mismo paso 2 veces más para añadir `"apellidos"` y `"edad"`, este útlimo de tipo `"Integer 16"`.



Atributos Persona

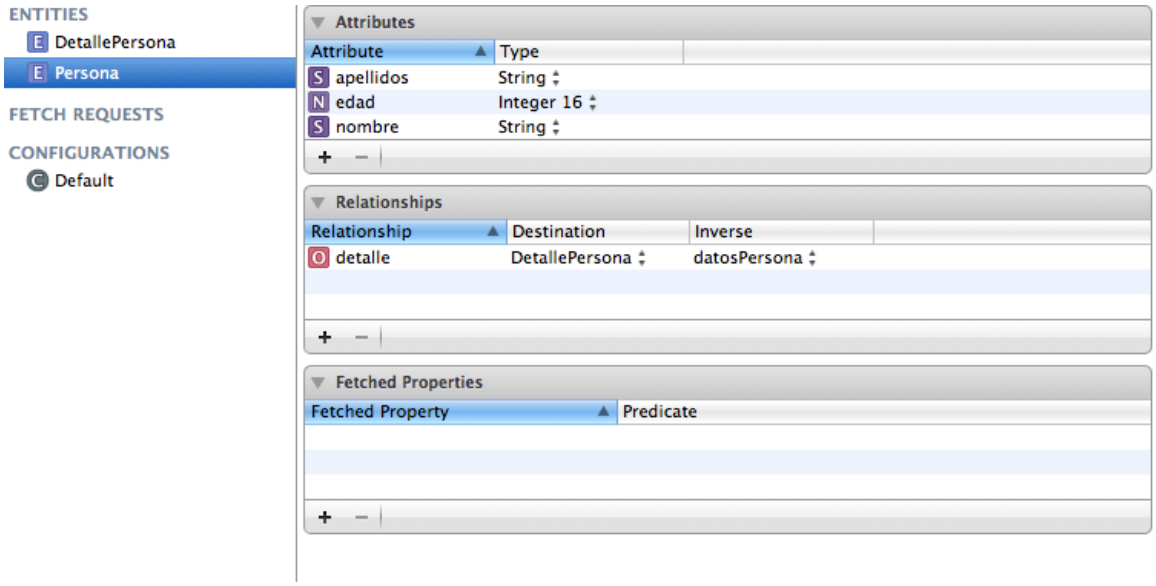
Ahora creamos la entidad de `"DetallePersona"` de la misma manera que la anterior pero creando los atributos: `"localidad"`, `"pais"`, `"direccion"`, `"cp"` y `"telefono"`. Todas de tipo `"String"`.



Atributos DetallePersona

Finalmente nos queda relacionar de algún modo las dos entidades, para ello añadimos una

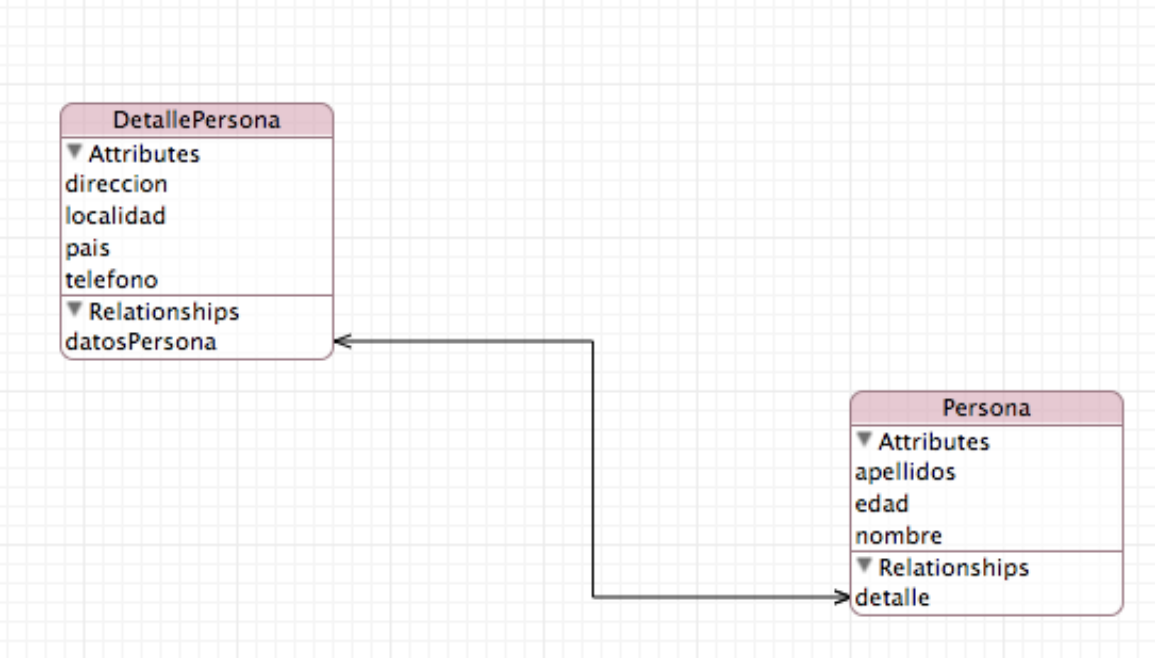
relación dentro de la entidad de `"Persona"` hacia `"DetallePersona"` simplemente haciendo click sobre el botón `(+)` de la sección de `"Relationships"`. A la relación le ponemos como nombre `"detalle"`, seleccionamos como destino la entidad `"DetallePersona"` y será `"No inverse"`. Este tipo de relación es uno a uno. Por dentro, *Core Data* realizará creará una clave ajena en la tabla de persona que apunte a DetallePersona, pero eso a nosotros no nos interesa.



Vista general modelo de datos (1)

Apple recomienda que siempre que hagamos una relación de una entidad a otra hagamos también la relación inversa, por lo tanto hacemos justo lo anterior pero al revés (de `"DetallePersona"` a `"Persona"`) con la diferencia que en la columna de `"Inverse"` seleccionamos `"datosPersona"`.

Ya tenemos las entidades y las relaciones creadas. Si hacemos click en el botón de `"Editor Style"`, en la parte de abajo a la derecha del editor, podemos ver de una forma más clara la estructura que hemos diseñado:



Vista general modelo de datos (2)

3.3. Probando el modelo

Una vez que tenemos el modelo creado pasamos a probarlo para comprobar que está todo correcto, para ello abrimos la clase delegada (`sesion04_ejemplo2AppDelegate.m`) y añadimos el siguiente fragmento de código arriba del método `applicationDidFinishLaunching`:

```
NSManagedObjectContext *context = [self managedObjectContext];
    NSManagedObject *persona = [NSEntityDescription
insertNewObjectForEntityForName:@"Persona"
                                      inManagedObjectContext:context];
    [persona setValue:@"Juan" forKey:@"nombre"];
    [persona setValue:@"Martinez" forKey:@"apellidos"];
    [persona setValue:[NSNumber numberWithInt:28] forKey:@"edad"];

    NSManagedObject *detallePersona = [NSEntityDescription
                                          insertNewObjectForEntityForName:
                                          @"DetallePersona"
                                          inManagedObjectContext:context];
    [detallePersona setValue:@"Calle Falsa, 123" forKey:@"direccion"];
    [detallePersona setValue:@"Alicante" forKey:@"localidad"];
    [detallePersona setValue:@"España" forKey:@"pais"];
    [detallePersona setValue:@"965656565" forKey:@"telefono"];

    [detallePersona setValue:persona forKey:@"datosPersona"];
    [persona setValue:detallePersona forKey:@"detalle"];

    NSError *error;
    if (![context save:&error]) {
        NSLog(@"Uhh... Algo ha fallado: %@", [error
```

```
localizedDescription]);
    }
```

Con este código hemos creado un objeto `"Persona"` enlazado con otro objeto `"DetallePersona"`. ¡Como puedes ver no hemos necesitado nada de código *SQL*!. Primero creamos un objeto del tipo `NSManagedObjectContext` que almacena el conexto de *Core Data*. Después creamos el objeto `"Persona"` en sí con `NSManagedObject`. A este objeto le pondremos los valores para cada una de las propiedades definidas en el modelo. Hacemos lo mismo con `"DetallePersona"`. Finalmente relacionamos el objeto `"Persona"` con el objeto `"DetallePersona"`.

Compilamos el proyecto, comprobamos que todo funciona bien y no salen errores.

Para comprobar que los datos se han almacenado correctamente en nuestro dispositivo los mostramos en la consola de la siguiente manera:

```
NSFetchRequest *fetchRequest = [[NSFetchRequest alloc] init];
    NSEntityDescription *entity = [NSEntityDescription
                                   entityForName:@"Persona"
                                   inManagedObjectContext:context];
    [fetchRequest setEntity:entity];
    NSArray *fetchedObjects = [context executeFetchRequest:fetchRequest
error:&error];
    for (NSManagedObject *info in fetchedObjects) {
        NSLog(@"Nombre: %@", [info valueForKey:@"nombre"]);
        NSLog(@"Apellidos: %@", [info valueForKey:@"apellidos"]);
        NSLog(@"Edad: %d", (NSInteger)[[info valueForKey:@"edad"]
intValue]);

        NSManagedObject *details = [info valueForKey:@"detalle"];
        NSLog(@"Direccion: %@", [details valueForKey:@"direccion"]);
        NSLog(@"Localidad: %@", [details valueForKey:@"localidad"]);
        NSLog(@"Pais: %@", [details valueForKey:@"pais"]);
        NSLog(@"Telefono: %@", [details valueForKey:@"telefono"]);

        NSLog(@"------------------");

    }
    [fetchRequest release];
```

El código de arriba lo escribiremos dentro del método `didFinishLaunchingWithOptions` de la clase delegada. Lo que hace este código es recorrer todos los objetos almacenados en memoria y mostrarlos por consola. Si arrancamos la aplicación veremos que se muestran todos los datos de los objetos almacenados. Cada vez que arranquemos la aplicación se mostrarán más objetos ya que también los creamos.

Para listar los objetos usamos la clase `NSFetchRequest`. Esta clase es equivalente a ejecutar una sentencia *"Select"* de SQL. Al objeto `NSFetchRequest` le asignamos la entidad que queremos listar *("SELECT * FROM ?? ")*, en nuestro caso será `"Persona"`. El listado de los objetos se obtiene mediante el método `executeFetchRequest` de la clase `NSManagedObjectContext` que hemos definido en el bloque anterior de código. Este método devolverá un `NSArray` con todos los objetos `NSManagedObject`.

```
2011-09-02 18:38:58.404 sesion04-ejemplo2[4893:207] Nombre: Juan
2011-09-02 18:38:58.405 sesion04-ejemplo2[4893:207] Apellidos: Martinez
2011-09-02 18:38:58.406 sesion04-ejemplo2[4893:207] Edad: 28
2011-09-02 18:38:58.406 sesion04-ejemplo2[4893:207] Direccion: Calle Falsa, 123
2011-09-02 18:38:58.407 sesion04-ejemplo2[4893:207] Localidad: Alicante
2011-09-02 18:38:58.408 sesion04-ejemplo2[4893:207] Pais: España
2011-09-02 18:38:58.408 sesion04-ejemplo2[4893:207] Telefono: 965656565
2011-09-02 18:38:58.409 sesion04-ejemplo2[4893:207] ------------------
```

Salida de consola

Nota

Si queremos ver qué sentencias SQL está ejecutando *Core Data* por debajo tenemos que activar el "flag" de debug, para ello seleccionamos la opción: `"Product" > "Edit Scheme"`. En la columna de la izquierda seleccionamos el esquema que estamos utilizando para Debug y en la parte de la derecha, en el cuadro de `"Arguments Passed on Launch"` añadimos 2 parámetros: `"-com.apple.CoreData.SQLDebug"` y "1".

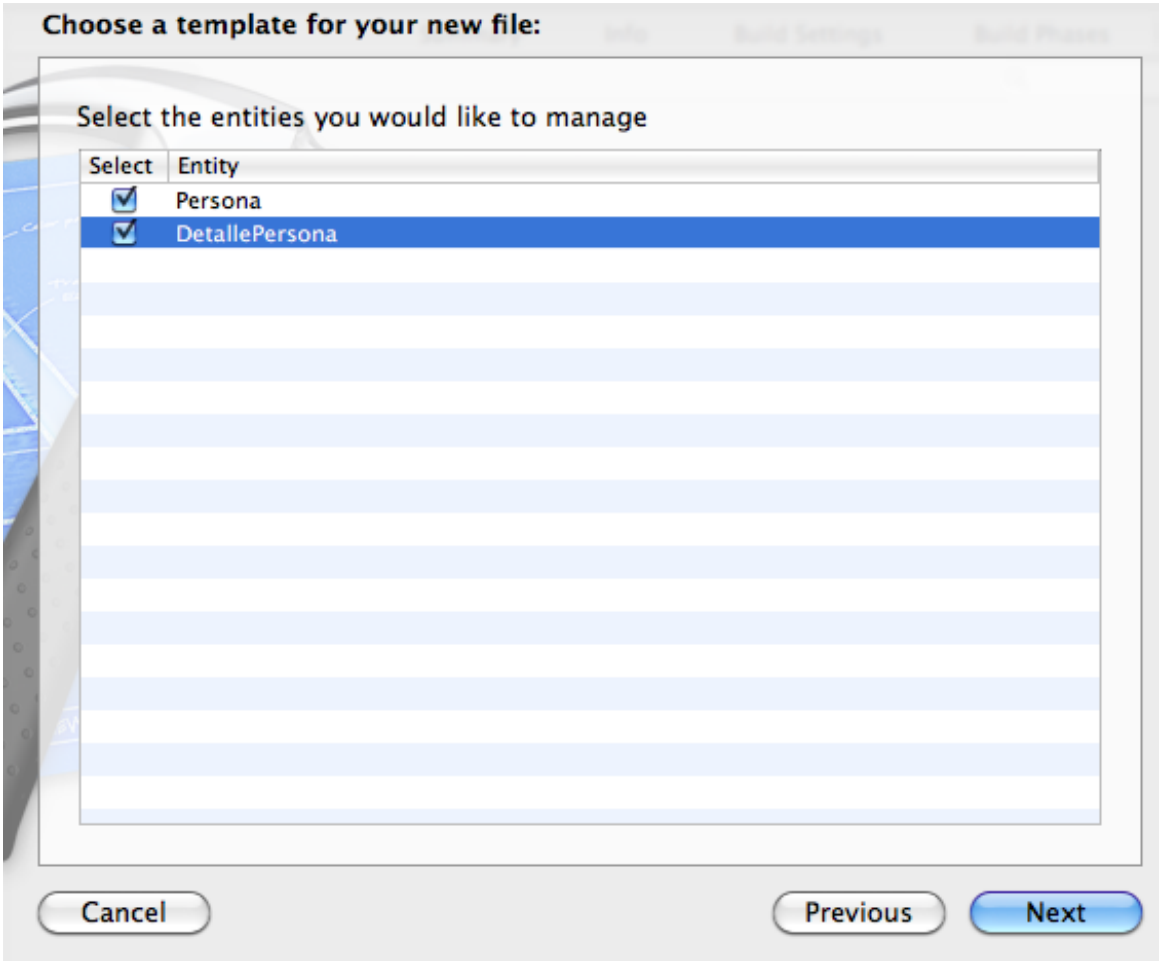
Ahora arrancamos de nuevo la aplicación y veremos que en la consola aparecerán las sentencias SQL utilizadas por *Core Data*:

```
(...)

INSERT INTO ZDETALLEPERSONA(Z_PK, Z_ENT, Z_OPT, ZDATOSPERSONA, ZTELEFONO,
ZLOCALIDAD, ZDIRECCION, ZPAIS)
VALUES(?, ?, ?, ?, ?, ?, ?, ?)
CoreData: sql: SELECT 0, t0.Z_PK, t0.Z_OPT, t0.ZTELEFONO, t0.ZLOCALIDAD,
t0.ZDIRECCION, t0.ZPAIS, t0.ZDATOSPERSONA
FROM ZDETALLEPERSONA t0 WHERE  t0.Z_PK = ?

(...)
```

3.4. Generando los modelos automáticamente

Generar los modelos a mano directamente desde el editor no es lo más recomendado ya que podemos cometer errores de escritura, errores de tipo, etc. La mejor manera de hacerlo es creando un archivo para cada entidad. Esto se puede hacer desde 0, pero *xCode* tiene un generador de clases que es muy fácil de usar y ahorremos bastante tiempo. ¡Vamos a usarlo!

Hacemos click sobre el raiz de nuestro proyecto y seleccionamos `"New File"`. En la columna de la izquierda seleccionamos `"iOS" > "Core Data"` y en el cuadro de la derecha `"NSManagedObject subclass"`. Click en `"next"`. En la siguiente pantalla seleccionamos nuestro modelo de datos `"sesion04_ejemplo2"` y click en `"next"`. Ahora nos aseguramos de seleccionar las dos entidades que hemos creado anteriormente: `"Persona"` y `"DetallePersona"`, click en `"next"`. Seleccionamos el raiz de nuestro proyecto para guardar los ficheros que se generen y aceptamos.



Ventana de selección de entidad

Ahora veremos que xsource ha creado automáticamente 4 ficheros nuevos dentro de nuestro proyecto: Persona.m/.h y DetallePersona.m/.h con todos los atributos especificados. Estas nuevas clases que heredan de `NSManagedObject` serán las que utilicemos a partir de ahora.

Cambiamos el código de la clase delegada por el siguiente:

```
NSManagedObjectContext *context = [self managedObjectContext];
    Persona *datosPersona = [NSEntityDescription
                             insertNewObjectForEntityForName:@"Persona"
                             inManagedObjectContext:context];
    datosPersona.nombre = @"Juan";
    datosPersona.apellidos = @"Martinez";
    datosPersona.edad = [NSNumber numberWithInt:28];

    DetallePersona *detallePersona = [NSEntityDescription
insertNewObjectForEntityForName:@"DetallePersona"
                                    inManagedObjectContext:context];
    detallePersona.direccion = @"Calle Falsa, 123";
```

```
    detallePersona.localidad = @"Alicante";
    detallePersona.pais = @"España";
    detallePersona.telefono = @"965656565";

    detallePersona.datosPersona = datosPersona;
    datosPersona.detalle = detallePersona;

    NSError *error;
    if (![context save:&error]) {
        NSLog(@"Uhh... Algo ha fallado: %@", [error
localizedDescription]);
    }



    //Listamos los datos
    NSFetchRequest *fetchRequest = [[NSFetchRequest alloc] init];
    NSEntityDescription *entity = [NSEntityDescription
                                   entityForName:@"Persona"
                                   inManagedObjectContext:context];
    [fetchRequest setEntity:entity];
    NSArray *fetchedObjects = [context executeFetchRequest:fetchRequest
error:&error];
    for (NSManagedObject *info in fetchedObjects) {
        NSLog(@"Nombre: %@", [info valueForKey:@"nombre"]);
        NSLog(@"Apellidos: %@", [info valueForKey:@"apellidos"]);
        NSLog(@"Edad: %d", (NSInteger)[[info valueForKey:@"edad"]
integerValue]);

        NSManagedObject *details = [info valueForKey:@"detalle"];
        NSLog(@"Direccion: %@", [details valueForKey:@"direccion"]);
        NSLog(@"Localidad: %@", [details valueForKey:@"localidad"]);
        NSLog(@"Pais: %@", [details valueForKey:@"pais"]);
        NSLog(@"Telefono: %@", [details valueForKey:@"telefono"]);

        NSLog(@"-----------------");
    }
    [fetchRequest release];
```

Ejecutamos el proyecto y comprobamos que en la consola aparece lo mismo que antes, pero ahora tenemos el código bastante más claro.

3.5. Creando la vista de tabla

Ahora vamos a mostrar los objetos que antes hemos listado por consola en una vista de tabla `UITableView`. Para ello creamos un `UITableViewController`:

Click en `"New file..." > "UIViewController subclass"`. En la siguiente pantalla nos aseguramos de seleccionar `"Subclass of UITableViewController"`, de esta forma nos aparecerán ya todos los métodos de la tabla creados por defecto. Guardamos el archivo con el nombre que queramos, por ejemplo: `"PersonasViewController"`.

Abrimos ahora `"PersonasViewController.h"` y añadimos 2 atributos a la clase:

- Un `NSArray` que se será el listado de personas (objetos de la clase Persona).
- Una referencia a `NSManagedObjectContext`.

El fichero `"PersonasViewController.h"` quedará de la siguiente manera:

```
//imports iniciales

@interface PersonasViewController : UITableViewController {
    NSArray *_listaPersonas;
    NSManagedObjectContext *_context;
}

@property (nonatomic, retain) NSArray *listaPersonas;
@property (nonatomic, retain) NSManagedObjectContext *context;

@end
```

Ahora en el .m importamos en el encabezado las clases `"Persona.h"` y `"DetallePersona.h"` y en el método `"viewDidLoad"` creamos el array listaPersonas con todos los objetos Persona de forma similar a cuando lo hicimos en el apartado anterior:

```
#import "PersonasViewController.h"

#import "Persona.h"
#import "DetallePersona.h"

@implementation PersonasViewController

@synthesize listaPersonas = _listaPersonas;
@synthesize context = _context;
```

En `viewDidLoad`:

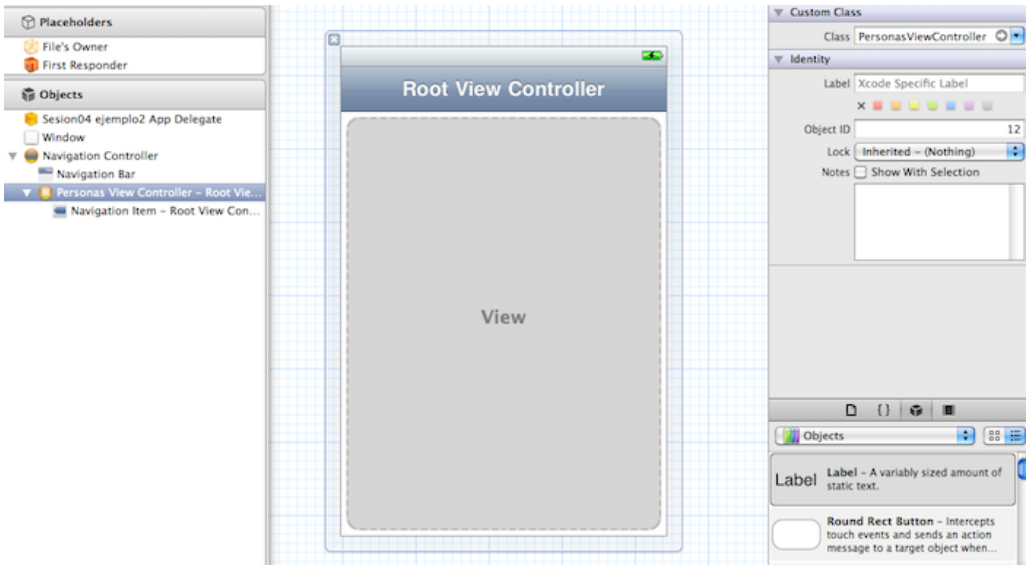
```
- (void)viewDidLoad
{
    [super viewDidLoad];

    NSFetchRequest *fetchRequest = [[NSFetchRequest alloc] init];
    NSEntityDescription *entity = [NSEntityDescription
                                   entityForName:@"Persona"
                                   inManagedObjectContext:_context];
    [fetchRequest setEntity:entity];
    NSError *error;
    self.listaPersonas = [_context executeFetchRequest:fetchRequest
error:&error];
    self.title = @"Listado Personas";
    [fetchRequest release];
}
```

Ahora completamos el resto de métodos heredados de `UITableViewController` de la misma forma que hicimos en sesiones anteriores:

```
- (NSInteger)numberOfSectionsInTableView:(UITableView *)tableView
{

    // Return the number of sections.
    return 1;
}

- (NSInteger)tableView:(UITableView *)tableView
numberOfRowsInSection:(NSInteger)section
{
```

```
    // Return the number of rows in the section.
    return [_listaPersonas count];
}

- (UITableViewCell *)tableView:(UITableView *)tableView
cellForRowAtIndexPath:(NSIndexPath *)indexPath
{
    static NSString *CellIdentifier = @"Cell";

    UITableViewCell *cell = [tableView
dequeueReusableCellWithIdentifier:CellIdentifier];
    if (cell == nil) {
      cell = [[[UITableViewCell alloc]
initWithStyle:UITableViewCellStyleSubtitle
      reuseIdentifier:CellIdentifier] autorelease];
    }

    // Configure the cell...
    Persona *p = (Persona *)[_listaPersonas objectAtIndex:indexPath.row];
    cell.textLabel.text = p.nombre;
    cell.detailTextLabel.text = [NSString stringWithFormat:@"%d años",
                                 [p.edad intValue]];

    return cell;
}
```

Ahora para poder mostrar la tabla dentro de nuestra aplicación debemos asignarla dentro de la clase delegada, para ello abrimos la vista `"MainWindow.xib"` y arrastramos una controladora de navegación `"Navigation Controller"` a la lista de objetos de la izquierda. La desplegamos y dentro de `"View Controller"` en la pestaña de la derecha de `"Identity Inspector"` escribimos el nombre de nuestra clase que tiene la vista de la tabla: `"PersonasViewController"`.



Pantalla del diseñador

Ahora abrimos el fichero `"sesion04_ejemplo2AppDelegate.h"` y creamos un

"UINavigationController" dentro de la declaración de variables dejando el fichero como sigue:

```
//imports iniciales

@interface sesion04_ejemplo2AppDelegate : NSObject <UIApplicationDelegate>
{
    UINavigationController *_navController;
}

@property (nonatomic, retain) IBOutlet UIWindow *window;
@property (nonatomic, retain) IBOutlet UINavigationController
*navController;

@property (nonatomic, retain, readonly) NSManagedObjectContext
*managedObjectContext;
@property (nonatomic, retain, readonly) NSManagedObjectModel
*managedObjectModel;
@property (nonatomic, retain, readonly) NSPersistentStoreCoordinator
*persistentStoreCoordinator;

- (void)saveContext;
- (NSURL *)applicationDocumentsDirectory;

@end
```

Dentro del .m importamos la clase "PersonasViewController.h" y completamos el @synthetize:

```
@synthesize navController = _navController;
```

Y por último, justo antes de la llamada al método "makeKeyAndVisible" escribimos el siguiente código para asignar el contexto de la clase delegada al "PersonasViewController":

```
PersonasViewController *root = (PersonasViewController *) [_navController
topViewController];
    root.context = [self managedObjectContext];
    [self.window addSubview:_navController.view];
```

Ahora compilamos y ejecutamos nuestro proyecto y, si todo ha ido bien, debería de salir la tabla con los datos de las personas:

Listado de personas

3.6. Migraciones de bases de datos con Core Data

Una vez que tenemos diseñada nuestra base de datos inicial, ¿qué pasaría si queremos modificarla para añadirle un campo más a una tabla? ¿Qué hará Core Data para realizar la migración de la versión anterior a la actual?

Siguiendo con el ejemplo anterior, si después de ejecutar la aplicación en nuestro dispositivo o simulador modificamos la estructura de la base de datos añadiendo un atributo nuevo en la entidad "DetallePersona", por ejemplo: "codigoPostal", con el tipo "Integer 16", veremos que al volver a arrancar la aplicación nos aparecerá un error en tiempo de ejecución similar al siguiente:

```
Unresolved error Error Domain=NSCocoaErrorDomain Code=134100 "The
operation
couldn’t be completed.
(Cocoa error 134100.)" UserInfo=0x6b6c680 {metadata=<CFBasicHash 0x6b68380
[0x1337b38]>{type = immutable dict, count = 7,entries =>
        2 : <CFString 0x6b70320 [0x1337b38]>{contents =
"NSStoreModelVersionIdentifiers"}
        =
        <CFArray 0x6b706e0 [0x1337b38]>{type = immutable, count = 1,
values = (
        0 : <CFString 0x1332cd8 [0x1337b38]>{contents = ""}
)}
        4 : <CFString 0x6b70350 [0x1337b38]>{contents =
"NSPersistenceFrameworkVersion"}
        =
```

```
        <CFNumber 0x6b6f2c0 [0x1337b38]>{value = +386, type =
kCFNumberSInt64Type}
        6 : <CFString 0x6b70380 [0x1337b38]>{contents =
"NSStoreModelVersionHashes"} =
        <CFBasicHash 0x6b707e0 [0x1337b38]>{type = immutable dict, count =
2,
entries =>
        1 : <CFString 0x6b70700 [0x1337b38]>{contents = "Persona"} =
<CFData 0x6b70740
        [0x1337b38]>{length = 32, capacity = 32, bytes =
        0x1c50e5a379ff0ddc3cda7b82a3934c5e ... 75649ac3c919beea}
        2 : <CFString 0x6b70720 [0x1337b38]>{contents = "DetallePersona"}
=
        <CFData 0x6b70790
        [0x1337b38]>{length = 32, capacity = 32, bytes =
        0xb7ba2eb498f9a1acdd6630d56f6cd12c ... e8963dd3e488ba95}
}

        7 : <CFString 0x10e3aa8 [0x1337b38]>{contents = "NSStoreUUID"} =
         <CFString 0x6b70510
        [0x1337b38]>{contents = "4F80A909-DA41-48ED-B24D-DBA73EC2BB9E"}
        8 : <CFString 0x10e3948 [0x1337b38]>{contents = "NSStoreType"} =
        <CFString 0x10e3958
        [0x1337b38]>{contents = "SQLite"}
        9 : <CFString 0x6b703b0 [0x1337b38]>{contents =
"_NSAutoVacuumLevel"} = <CFString
        0x6b70830 [0x1337b38]>{contents = "2"}
        10 : <FString 0x6b703d0 [0x1337b38]>{contents =
"NSStoreModelVersionHashesVersion"}
         =
        <CFNumber 0x6b61950 [0x1337b38]>{value = +3, type =
kCFNumberSInt32Type}
}
, reason=The model used to open the store is incompatible with the one
used to
create the store}, {
    metadata =     {
        NSPersistenceFrameworkVersion = 386;
        NSStoreModelVersionHashes =         {
            DetallePersona = <b7ba2eb4 98f9a1ac dd6630d5 6f6cd12c f8900077
46b16f00
            e8963dd3 e488ba95>;
            Persona = <1c50e5a3 79ff0ddc 3cda7b82 a3934c5e 5cb4ee4b
4ac98838 75649ac3
            c919beea>;
        };
        NSStoreModelVersionHashesVersion = 3;
        NSStoreModelVersionIdentifiers =         (
            ""
        );
        NSStoreType = SQLite;
        NSStoreUUID = "4F80A909-DA41-48ED-B24D-DBA73EC2BB9E";
        "_NSAutoVacuumLevel" = 2;
    };
    reason = "The model used to open the store is incompatible with the
one used
    to create the store";
}
```

Este error es muy común cuando estamos desarrollando aplicaciones con Core Data. Básicamente nos dice que el modelo de base de datos que está intentando cargar de memoria es distinto al que se indica en la aplicación, esto es porque no hemos indicado ningún método de migración. Realmente existen dos formas de solucionar este error:

- Por un lado, y **sólo cuando estemos desarrollando y no tengamos ninguna versión de nuestra aplicación lanzada en la App Store**, podremos eliminar la base de datos del dispositivo / simulador simplemente borrando la aplicación de el. También podemos ejecutar el siguiente código una sóla vez: `[[NSFileManager defaultManager] removeItemAtURL:storeURL error:nil]`
- En el caso de que estemos realizando una migración sobre una versión ya publicada en la App Store el método anterior no nos serviría, ya que obligaríamos al usuario a borrar su aplicación del dispositivo con lo que todos los datos que tuviera se perderían. Esto no es viable, por lo que necesitaremos usar otro método para realizar la migración y solucionar el error de una forma más "limpia", este método es el que vamos a comentar a continuación.

XCode soporta Versioning para Core Data, esto es que podemos crear versiones de la base de datos de una forma sencilla si seguimos una serie de pasos:

1) Añadimos las siguientes opciones en forma de `NSDictionary` a nuestro código del `persistentStoreCoordinator`

```
NSDictionary *options = [NSDictionary dictionaryWithObjectsAndKeys:
                            [NSNumber numberWithBool:YES],
                            NSMigratePersistentStoresAutomaticallyOption,
                            [NSNumber numberWithBool:YES],
                            NSInferMappingModelAutomaticallyOption,
                            nil];
```

Y modificamos el método `addPersistentStoreWithType` pasándole como parámetro el diccionario `options`:

```
if (![__persistentStoreCoordinator
addPersistentStoreWithType:NSSQLiteStoreType
configuration:nil URL:storeURL options:options error:&error])
    {
    ....
    }
```

Con el código anterior indicamos al `persistentStoreCoordinator` que la migración, en el caso de que se deba de realizar, sea de forma automática.

Atención

Deberemos de tener en cuenta que este tipo de migración que estamos estudiando en este punto sólo es válida para cambios simples en Core Data, lo que Apple llama como *lightweight migration*. Este tipo de migración admite una serie de cambios que se indican dentro de la documentación de Apple, para cambios más complejos deberemos realizar migraciones personalizadas y mucho más complejas. El documento en donde Apple explica el tipo de migraciones que existen para Core Data y su forma de implementarlas lo podemos consultar desde esta web.

2) Ahora deberemos de crear la versión del esquema de Core Data dentro de XCode, para

ello seleccionamos el fichero del esquema `sesion04_ejemplo2.xcdatamodeld` y seleccionamos en el menú principal: *Editor > Add Model Version*. A la nueva versión la llamaremos: `sesion04_ejemplo_2` (versión 2). Una vez que hayamos pulsado sobre *Finish* se nos habrá creado la nueva versión del esquema dentro de la principal, y estará marcada con un "tick" la versión anterior.

3) Una vez que tenemos creada la nueva versión, deberemos modificarla añadiendole el nuevo atributo. Seleccionamos la versión recién creada `sesion04_ejemplo_2.xcdatamodel` y añadimos a la entidad `DetallePersona` el atributo `codigoPostal`. **Atención:** Si este atributo lo hemos añadido anteriormente deberemos de borrarlo de la versión anterior y añadirlo en la segunda versión.

4) Ahora nos queda indicar a Core Data que queremos usar la versión 2 de la base de datos para nuestra aplicación. Para ello seleccionamos el esquema principal (el raiz) `sesion04_ejemplo2.xcdatamodeld` y en la ventana de la derecha, en la pestaña de *File Inspector* seleccionamos como Versión actual la versión 2, esto lo hacemos dentro del apartado de *Versioned Core Data Model*. Automáticamente veremos que el símbolo de *tick* cambia a la versión 2.

5) Ahora ya podemos volver a compilar y veremos que no nos aparece el error y funciona todo según lo esperado.