

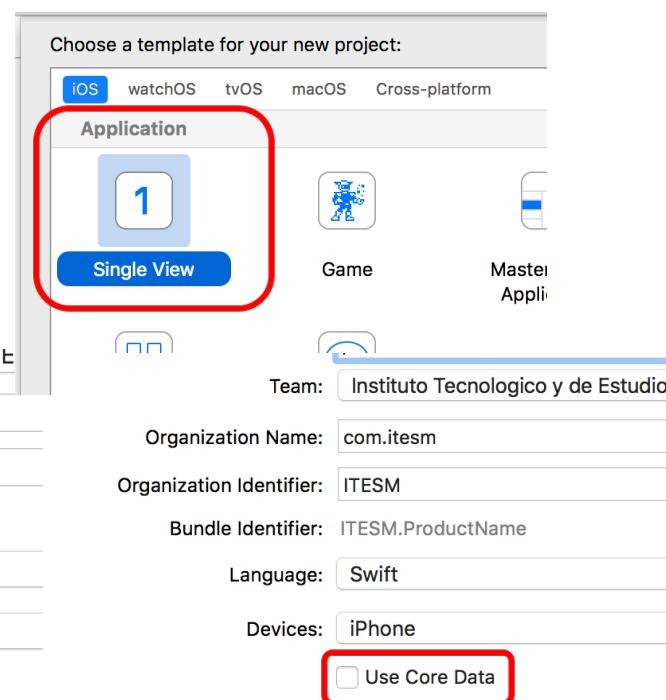
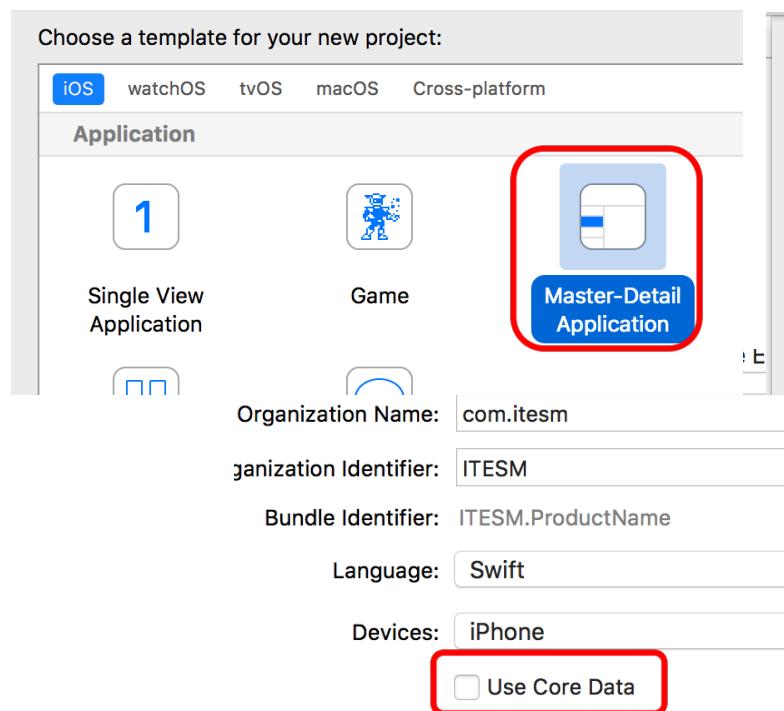


Persistencia con Core Data

Yolanda Martínez Treviño

Los templates

- + Xcode provee la opción de agregar **Core Data** a algunos de los templates que incluye, de manera que hace mucho del trabajo.



El modelo de datos

- + `nombreProyecto.xcdatamodeld` es el modelo de datos.
- + **Core Data** nos permite diseñar el modelo de datos de manera visual.
- + Si le das clic a este archivo aparece el editor de modelo de datos.

Editor del modelo de datos

The screenshot shows the Xcode Data Model Editor interface. On the left, there's a sidebar with sections for ENTITIES, FETCH REQUESTS, and CONFIGURATIONS. Under ENTITIES, two entities are listed: PhoneNumber and Record. The Record entity is selected, indicated by a blue background. A blue box labeled "clase" points to the Record entity in the sidebar. The main area is titled "PhoneBook.xcdatamodeld" and contains two sections: "Attributes" and "Relationships".

Attributes

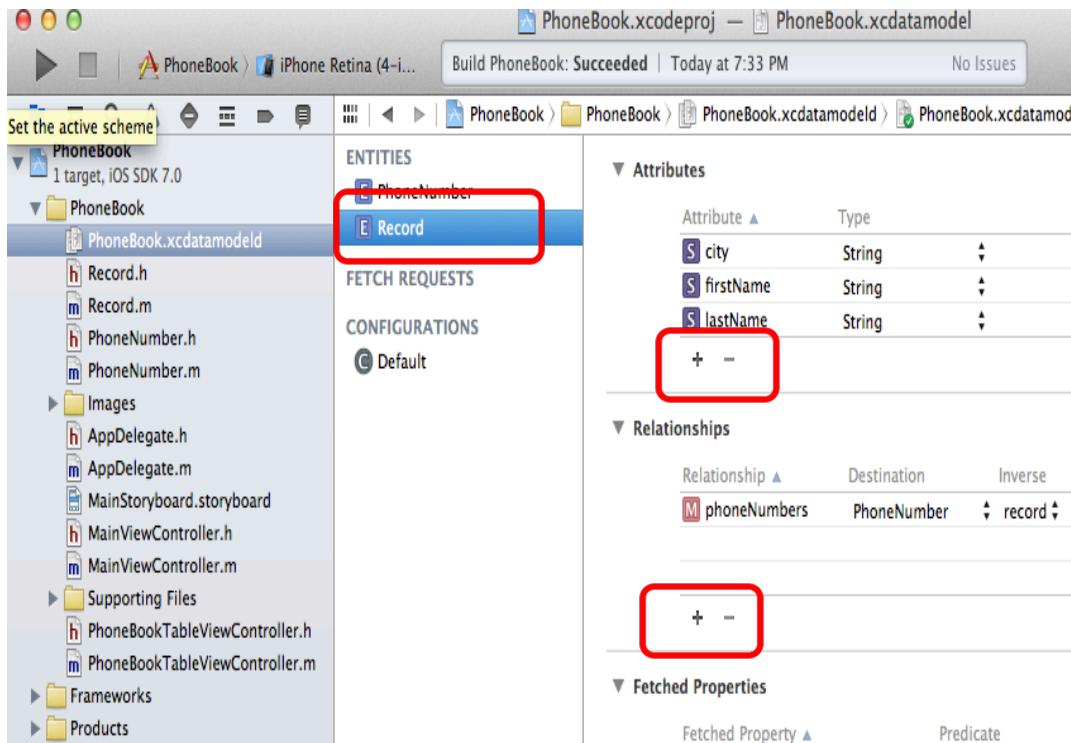
Attribute	Type
S city	String
S firstName	String
S lastName	String

Relationships

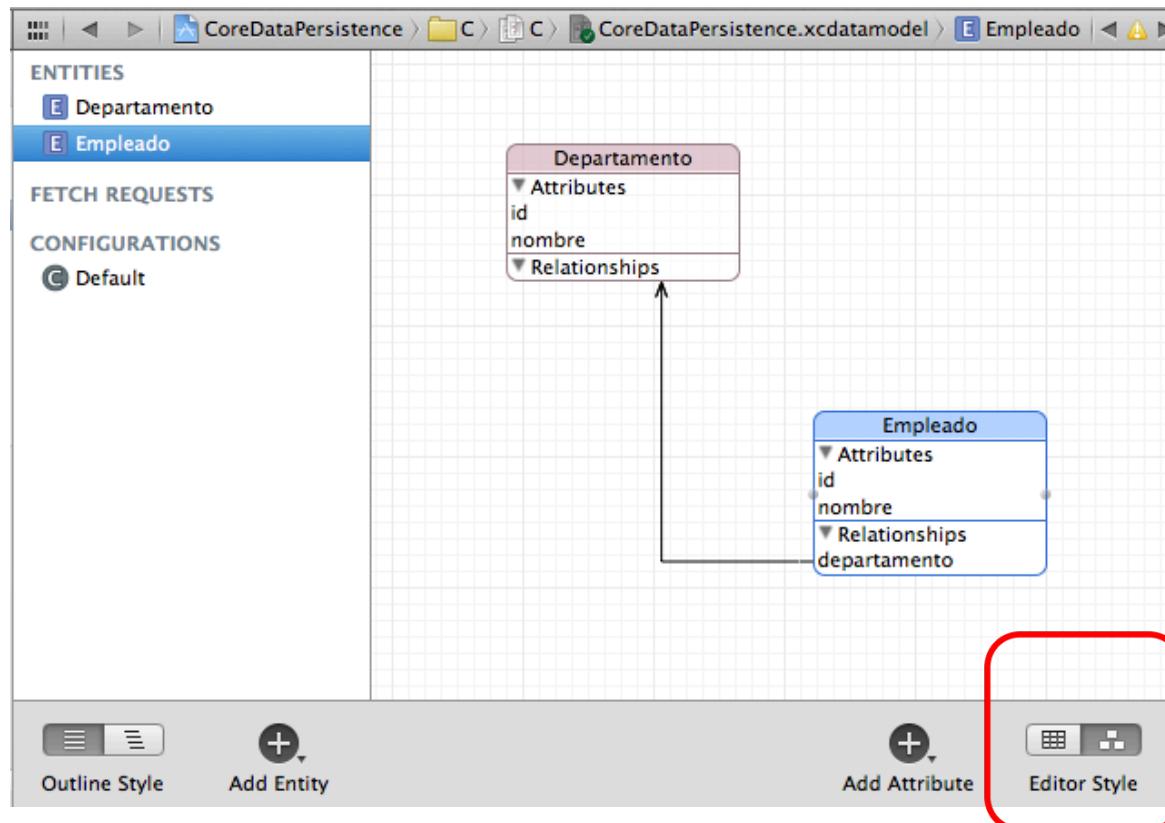
Relationship	Destination	Inverse
M phoneNumbers	PhoneNumber	record

A blue box labeled "Variable de instancia" points to the Attributes section, and another blue box labeled "Relación entre entidades" points to the Relationships section.

Usando el editor del modelo de datos



Eligiendo la vista de diagrama



La estructura del Core Data

+ Tomado de:

http://www.techotopia.com/index.php/IOS_8_Databases_in_Swift_using_Core_Data

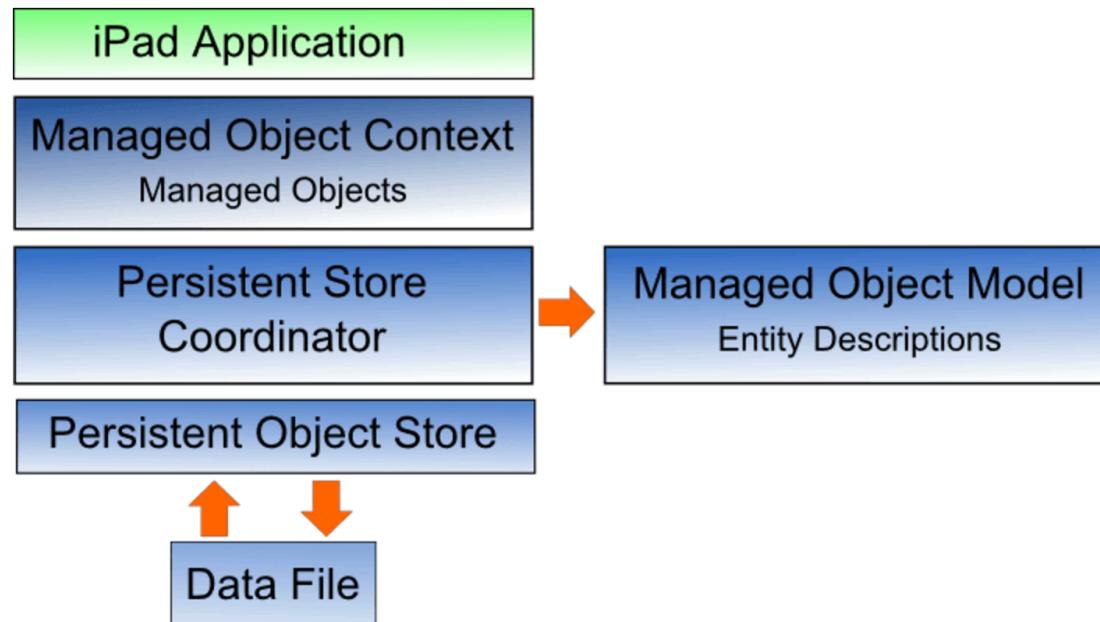


Figure 43-1

Managed objects

- + En el programa se crean los **managed objects** para colocar ahí los datos de un registro que será guardado en la base de datos.
- + El término **managed object** se refiere a las instancias concretas de la entidad creadas en tiempo de ejecución.
- + El **managed object** es como el renglón o registro de una tabla en la base de datos.
- + Los **managed objects** se encuentran dentro del Managed Object Context.

Managed object context

- + El **managed object context**, o **contexto** es el intermediario del programa con la memoria persistente.
- + Contiene los **managed objects** con los que el programa está trabajando.
- + Los cambios a los objetos se mantienen de manera temporal en el contexto, mientras esos cambios no sean guardados a la memoria permanente.

¿En dónde se guardan la información?

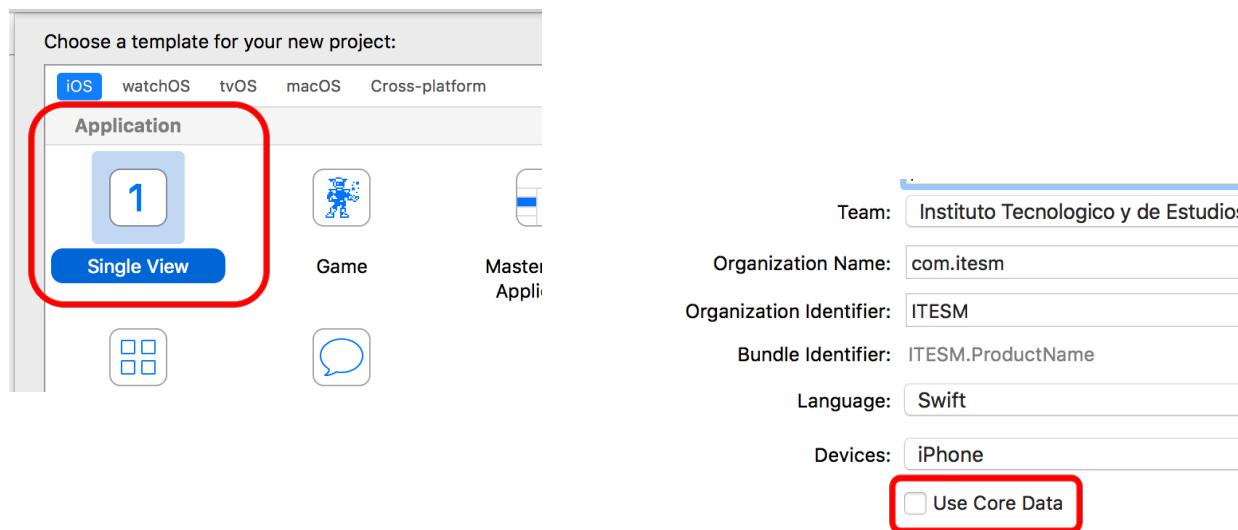
- + Por default, **Core Data** implementa el almacenamiento de datos como una base de datos de **SQLite** y lo guarda en el directorio **/Documents**.
- + Las clases del framework **Core Data** hacen todo el trabajo de guardar y cargar los datos, de manera que si usas **Core Data** no escribes instrucciones de **SQLite**.



Ejercicio

Crea un proyecto

+ De tipo Single View y que use Core Data



Observa el AppDelegate.swift

```
o
9 import UIKit
10 import CoreData
11
46
47 // MARK: - Core Data stack
48
49 lazy var persistentContainer: NSPersistentContainer = {
50     /*
51      The persistent container for the application. This implementation
52      creates and returns a container, having loaded the store for the
53      application to it. This property is optional since there are legitimate
54      error conditions that could cause the creation of the store to fail.
55     */
56     let container = NSPersistentContainer(name: "ejCoreDataA16")
57     container.loadPersistentStores(completionHandler: { (storeDescription, error) in
58         if let error = error as NSError? {
59             // Replace this implementation with code to handle the error appropriately
60             // fatalError() causes the application to generate a crash log and terminate
61             // should not use this function in a shipping application, although it may
62             // be useful during development.
```

Propiedad lazy calculada

NSPersistentContainer

API Reference

Class

NSPersistentContainer

A container that encapsulates the Core Data stack in your application.

Overview

NSPersistentContainer simplifies the creation and management of the Core Data stack by handling the creation of the [NSManagedObjectModel](#), [NSPersistentStoreCoordinator](#), and the [NSManagedObjectContext](#).

Propiedad calculada

- + En swift se pueden declarar propiedades calculadas.
- + No guardan un valor, sino que definen métodos get y set para calcular su valor cuando se manden llamar.

```
struct Rect {  
    var origin = Point()  
    var size = Size()  
    var center: Point {  
        get {  
            let centerX = origin.x +  
                (size.width / 2)  
            let centerY = origin.y +  
                (size.height / 2)  
            return Point(x: centerX, y:  
                centerY)  
        }  
        set(newCenter) {  
            origin.x = newCenter.x -  
                (size.width / 2)  
            origin.y = newCenter.y -  
                (size.height / 2)  
        }  
    }  
}
```

Propiedad lazy calculada

- + Es una propiedad cuyo valor inicial se calcula hasta la primera vez que se usa.
- + Son útiles cuando el valor inicial de la propiedad depende de valores que se conocerán hasta que se termine de inicializar el objeto.
- + Cuando el cálculo consume muchos recursos, entonces se define lazy para ejecutarse solamente si es necesario en el momento en el que se utilice por primera vez.

Crea un modelo de datos

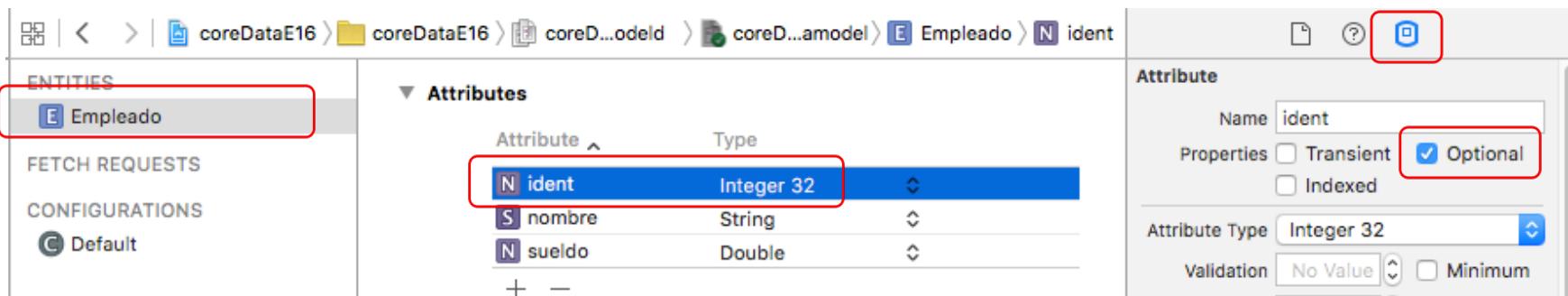
- + Agrega una entidad, llámala **Empleado**
- + Agrega atributos **ident**, **nombre** y **suelo** como se muestra enseguida:

The screenshot shows the 'Entities' section of the Xcode storyboard editor. On the left, there's a sidebar with 'ENTITIES', 'FETCH REQUESTS', and 'CONFIGURATIONS'. Under 'ENTITIES', 'Empleado' is selected, indicated by a grey background. To the right, under 'Attributes', three attributes are listed: 'ident' (Integer 32), 'nombre' (String), and 'suelo' (Float). Each attribute has a small disclosure triangle icon to its left.

Attribute	Type
N ident	Integer 32
S nombre	String
N sueldo	Float

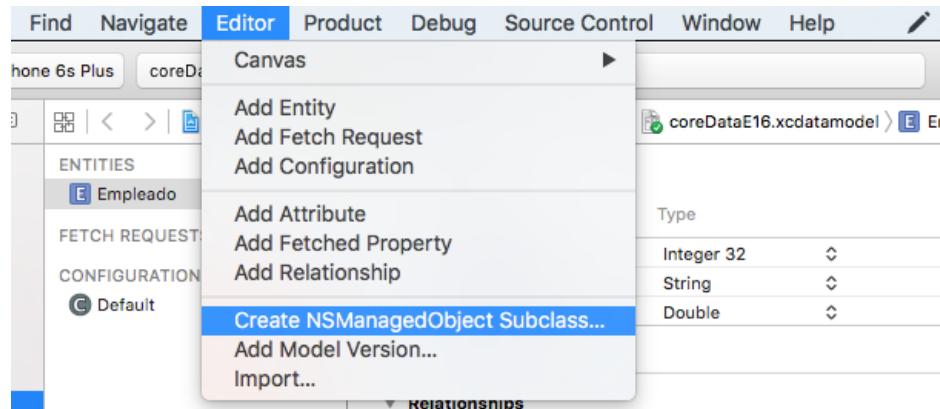
Tipos de datos

- + Core Data crea por default todas las variables de tipo Opcional. Si no se requiere de esa forma se puede modificar quitando el checkbox que lo indica, como se muestra enseguida:



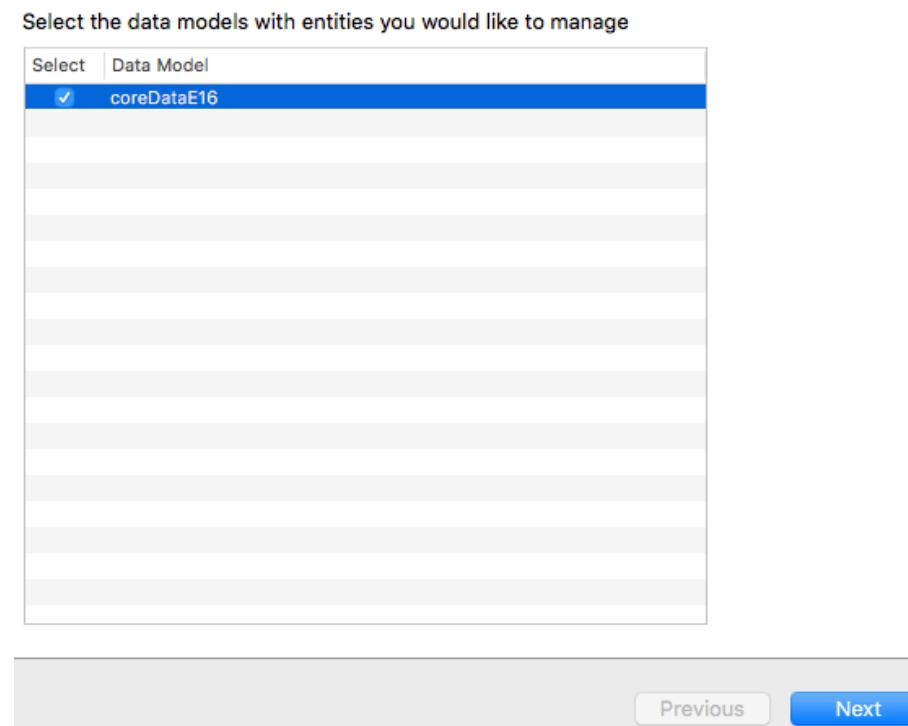
Crear la clase para manejar los objetos dentro de la aplicación

- + Core Data permite crear de manera automática una clase que contenga todos los datos de la entidad que definimos para poder usarla dentro de la aplicación.
- + Con el modelo de datos abierto da clic en **Editor/CreateNSManagedObject Subclass**



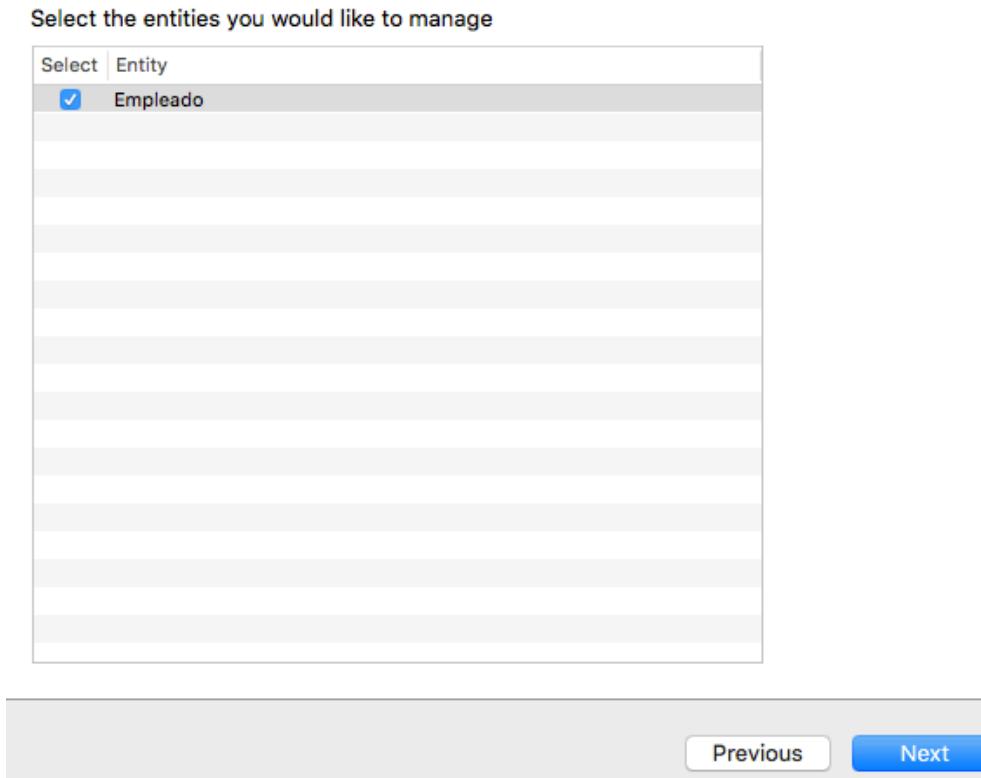
Primero selecciona la base de datos

- + Aparecerá una vista como la siguiente, selecciona la base de datos y da clic en Next

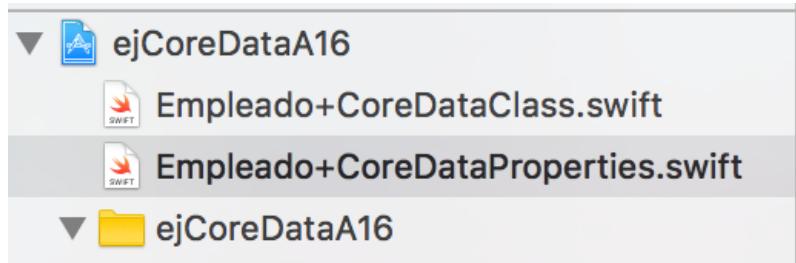


Después selecciona la Entidad

- + Enseguida aparecerá una vista como esta, selecciona la entidad:



Al dar crear aparecerán 2 archivos

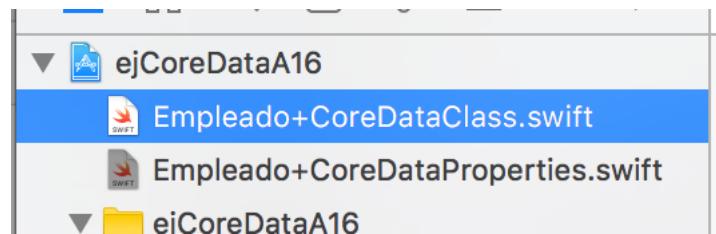


El atributo **@NSManaged** se usa para indicar que Core Data provee la implementación y el almacenamiento del atributo basado en la descripción del Entity.

```
extension Empleado {  
  
    @nonobjc public class func fetchRequest() -> NSFetchedResultsController<Empleado> {  
        return NSFetchedResultsController<Empleado>(entityName: "Empleado");  
    }  
  
    @NSManaged public var ident: Int32  
    @NSManaged public var nombre: String?  
    @NSManaged public var sueldo: Float  
}
```

Como no necesitamos que las propiedades sean opcionales, quitemos el ?

Al dar crear aparecerán 2 archivos



```
8  
9 import Foundation  
10 import CoreData  
11  
12  
13 public class Empleado: NSManagedObject {  
14  
15 }  
16
```

En el view controller crea una interfaz



Agrega los outlets y actions

- + Se requiere un outlet para cada text field y un action para cada botón. Agrega también el import de CoreData

```
12  
13 @IBOutlet weak var tfId: UITextField!  
14 @IBOutlet weak var tfNombre: UITextField!  
15 @IBOutlet weak var tfSueldo: UITextField!  
16 @IBOutlet weak var lbEstatus: UILabel!  
17  
18 @IBAction func oprimioGuardar(_ sender: UIButton) {  
19 }  
20  
21 @IBAction func oprimioBuscar(_ sender: AnyObject) {  
22 }
```

Agrega lo necesario para guardar

```
@IBAction func oprimioGuardar(_ sender: UIButton) {
    // Obtiene el contexto
    let appDelegate = UIApplication.shared.delegate as! AppDelegate
    let managedContext = appDelegate.persistentContainer.viewContext

    let id = Int(tfId.text!)
    let nom = tfNombre.text
    let suel = Double(tfSueldo.text!)
    if id == nil || nom == nil || suel == nil {
        lbEstatus.text = "No están completos los datos"
        return
    }

    // Crea el objeto en el contexto
    let entidad = NSEntityDescription.entity(forEntityName: "Empleado", in: managedContext)!
    let empleado = NSManagedObject(entity: entidad, insertInto: managedContext)

    empleado.setValue(id!, forKeyPath: "ident")
    empleado.setValue(nom!, forKeyPath: "nombre")
    empleado.setValue(suel!, forKey: "sueldo")

    // Se llama el método save del managedContext, lo que hace que se guarde en la bd
    if managedContext.hasChanges {
        do {
            try managedContext.save()
            tfId.text = ""
            tfNombre.text = ""
            tfSueldo.text = ""
            lbEstatus.text = "Empleado guardado "
        } catch let error as NSError {
            lbEstatus.text = "El empleado no fue guardado "
            print("Could not save. \(error), \(error.userInfo)")
        }
    }
}
```

Ahora agrega lo necesario para buscar

```
@IBAction func oprimioBuscar(_ sender: AnyObject) {
    //Obtiene el contexto
    let appDelegate = UIApplication.shared.delegate as! AppDelegate
    let managedContext = appDelegate.persistentContainer.viewContext

    let fetchRequest = NSFetchedResultsController<Empleado>(entityName: "Empleado")

    // agrega el predicado al request
    let predicate = NSPredicate(format: "(ident = %@)", tfId.text!)
    fetchRequest.predicate = predicate

    var resultados : [Empleado]!

    do {
        resultados = try managedContext.fetch(fetchRequest)

        if (resultados.count) > 0 {
            tfId.text = String(describing: resultados[0].ident)
            tfNombre.text = resultados[0].nombre
            tfSueldo.text = String(describing: resultados[0].sueldo)
            lbEstatus.text = ""
        }
        else {
            lbEstatus.text = "El empleado no existe"
        }
    } catch let error as NSError {
        print("Could not fetch. \(error), \(error.userInfo)")
        lbEstatus.text = "Error al leer la base de datos"
    }
}
```