HIT3329 / HIT8329
Creating Data Driven Mobile Applications

Lecture 5
Core Data

Presented by Paul Chapman

Adjunct Lecturer F-ICT
Director, Long Weekend LLC
Swinburne University of Technology

Last Lecture Reviewed

1. Protocols
2. Delegates & Datasources
3. Interface Builder
4. Application Object
5. Collections
6. File IO

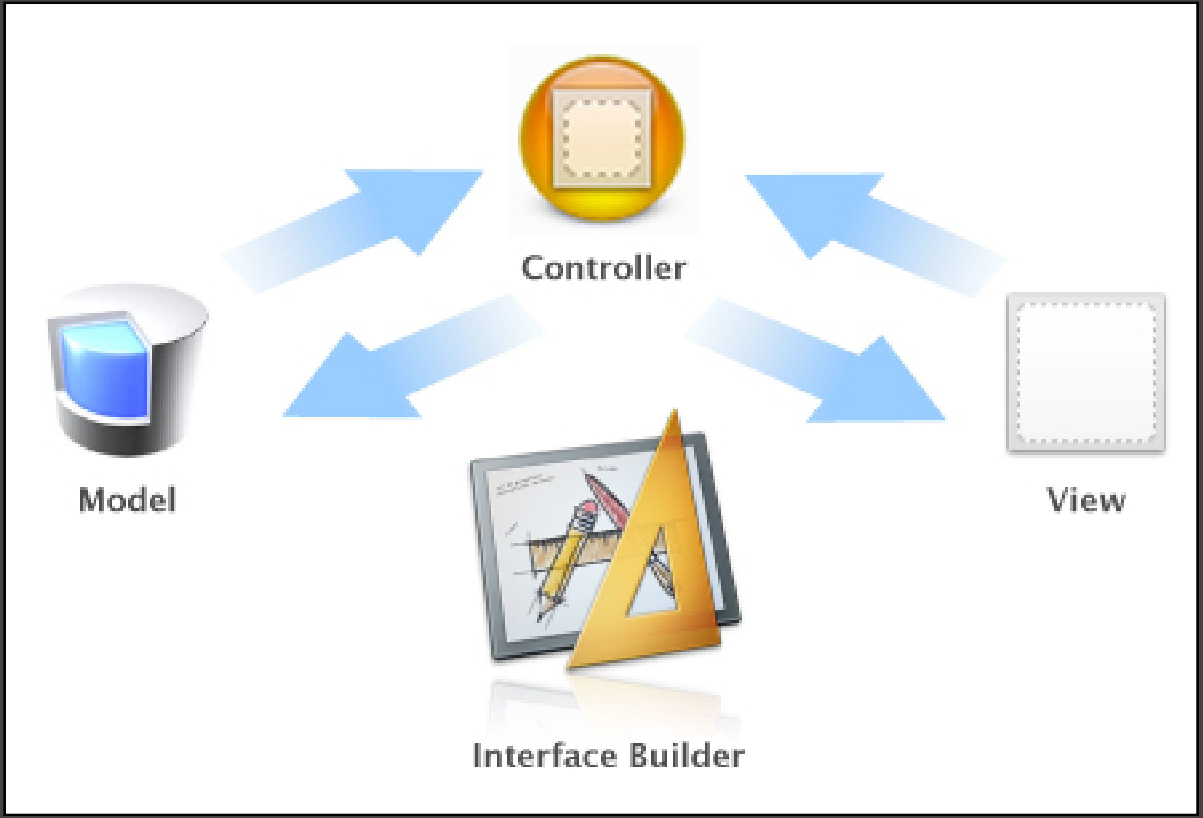
Questions?

What's On For Today?

1. Core Data Concepts
2. Creating a Managed Model
3. Retrieving Data
4. CRUD Operations
5. Notification Center

1.0 Core Data Concepts

In iOS the *Model* from Model-View-Controller (MVC) design pattern is often implemented with Core Data



1.1 What is Core Data?

- An "Object Graph Management Framework"
  - Not a database
  - Not an ORM
- APIs for storing/retrieving data objects
- Helps you make ***persistent data objects***

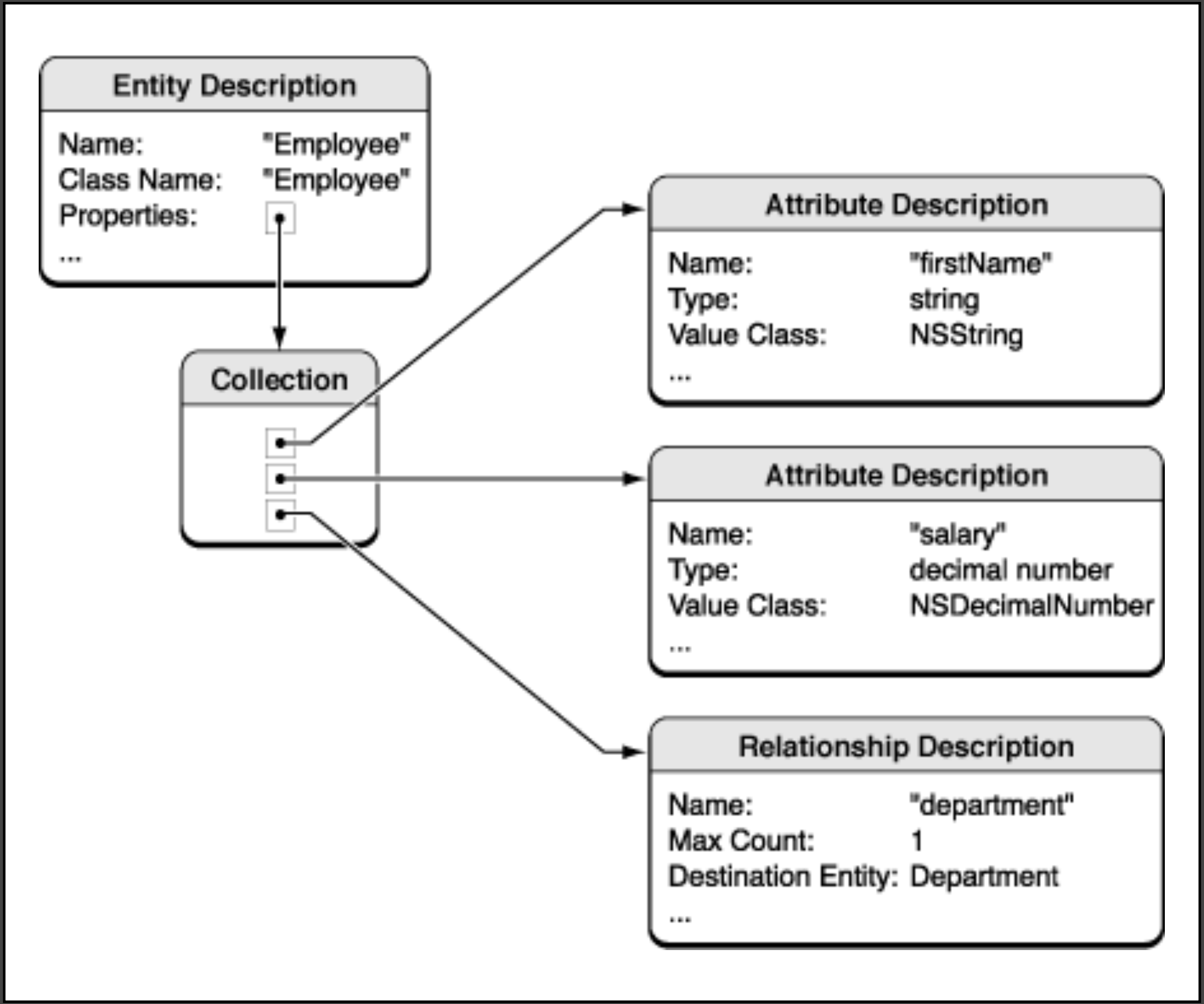
1.2 What Can Core Data Do For You? I

- Save, search, & load objects from disk (i.e. to the "persistent store")
- No need to worry about storage details -- could be any of:
  - XML
  - SQLite database
  - Flat text file
- Ideal for managing the '*Model*' in MVC

1.3 What Can Core Data Do For You? II

- Maintains object relationship integrity, enforces property constraints:
- "**Pizza** *always* has **crust**, **tomato sauce** and cheese. It □*may* have 1 or more **Toppings**"
- If we try to save a *Pizza* entity without cheese or cream sauce, Core Data would complain -- it would also not let us save the object!

1.4 Core Data Illustrated



Source: http://developer.apple.com/library/ios/#documentation/Cocoa/Conceptual/CoreData/Articles/cdBasics.html

1.5 Core Concepts

1. **Managed Objects**
   - Managed Object Model
   - Managed Object Context
2. **Fetch Requests**
   - Predicates
   - Sort Descriptors
3. **Persistent Stores**
   - Persistent Store Coordinator

1.6 Managed Objects

- In Object Oriented Programming, we worry about ***Objects***, not tables, records, or files
- Stored objects are known as ***Entities***
- Entities differ from a ***Class***, they do not contain code when stored
- Entities contain code when loaded in memory
- You can store, retrieve and search entities

1.7 Managed Object Model

- ***Entities*** may have relationships with each other (meaningful ones, of course)
- Relationships are defined in a ***Schema***
- The schema describes each entity's:
  - properties (attributes)
  - constraints on its properties
  - relationships

1.8 The Managed Object Context (MOC)

"You can think of a ***managed object context*** as an intelligent scratch pad."

"When you fetch objects ... you bring temporary copies onto the scratch pad where they form an object graph (or a collection of object graphs)."

"You can then modify those objects however you like [but] unless you actually save those changes ... the persistent store [will] remain unaltered."

Source: http://developer.apple.com/library/ios/#documentation/Cocoa/Conceptual/CoreData/Articles/cdBasics.html

1.9 Fetch Requests

- ***Fetch Requests*** retrieve stored objects
- They usually require 3 parts:

1. Entity to fetch	"get me Pizzas"
2. Predicate (filter)	"with no toppings"
3. Sort descriptor	"sorted by size"

NB: A Managed Object Context (MOC) is needed to fetch anything - it retrieves the managed objects for us

1.10 Predicates

- Use ***Predicates*** to filter fetch requests
- Built from formatted string
- Boolean syntax similar to SQL
- Compound predicates supported

```
// Get pizzas with marinara sauce
[NSPredicate predicateWithFormat:
        @"sauce = %@", kMarinaraSauce];
```

1.11 Sort Descriptors

- **Sort Descriptors** take:
  - An property name
  - A sort direction for that property
- Combine in array to use in ***fetch requests***

```
// Sort our pizzas by size, biggest to smallest
sortDescriptor =
[NSSortDescriptor sortDescriptorWithKey:@"pizzaSize"
                                        ascending:NO];
NSArray *sortDescriptors = [NSArray
                        arrayWithObject:sortDescriptor];
```

1.12 Example of a Fetch Request

```
NSManagedObjectContext *moc = /*assume this exists*/
NSFetchRequest *fetch = [[NSFetchRequest alloc] init];

NSEntityDescription *pizza =
   [NSEntityDescription entityForName:@"Pizza"
                 inManagedObjectContext:moc];

fetchRequest.entity = pizza;
fetchRequest.sortDescriptors = sortDescriptors;
fetchRequest.predicate = predicate;

NSError *error = NULL;
NSArray *results = [moc executeFetchRequest:fetch
                               error:&error];
[fetch release];
```

1.13 Persistent Stores

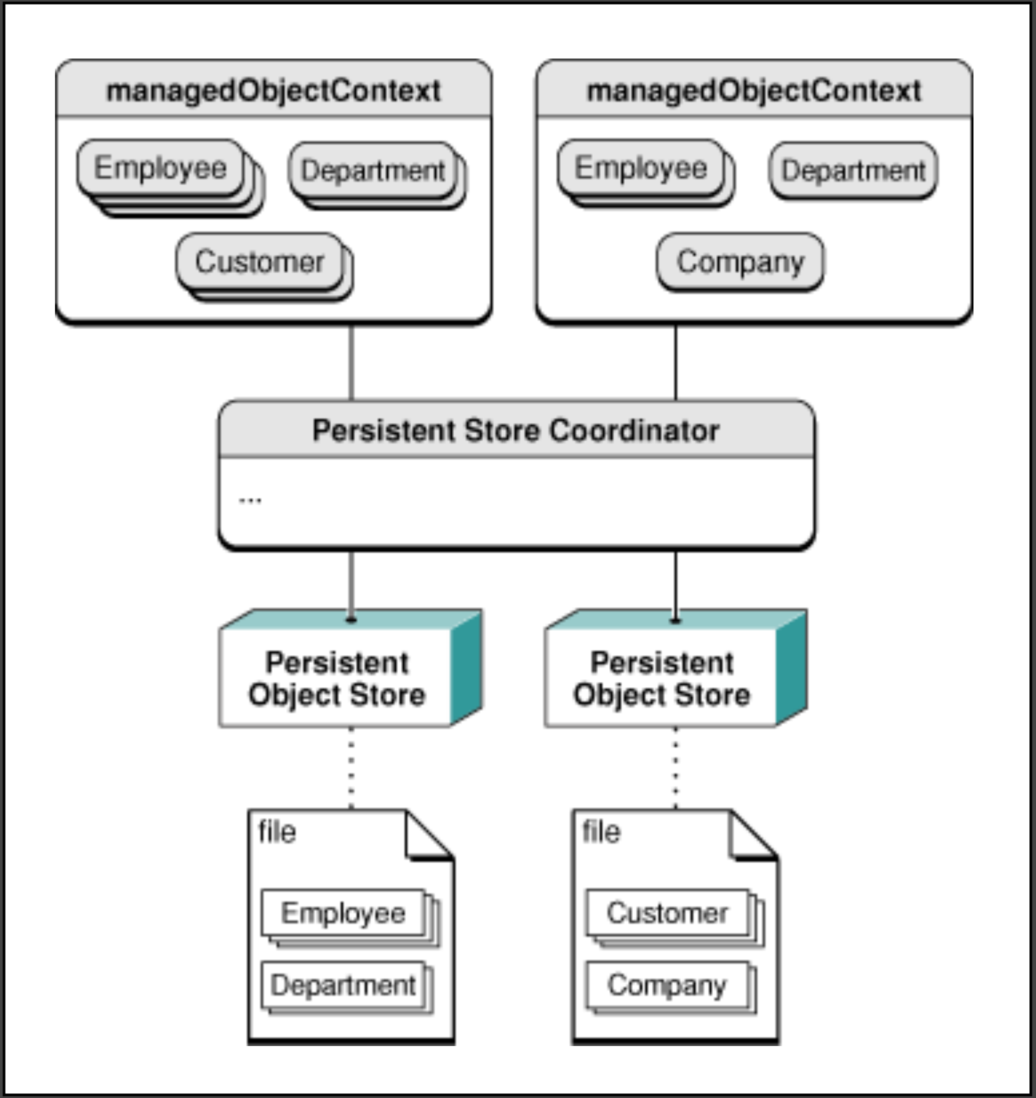
- Discrete sources of data, including flat files and databases
- Rarely used, except to define "where" it is
- Core Data abstracts away implementation details of each storage format providing a consistent interface

Benefits of Abstraction:
If your data store format changes, your app does not need to change.

1.14 Persistent Store Coordinator

- Groups together different ***Persistent Stores***
- Presents them as a single store
- The components coordinating your objects with external data sources are collectively known as the ***Persistence Stack***

1.15 Persistence Stack Illustrated



Source: http://developer.apple.com/library/ios/#documentation/Cocoa/Conceptual/CoreData/Articles/cdBasics.html

1.16 Core Concepts Reviewed

1. **Managed Objects**
   - Managed Object Model
   - Managed Object Context
2. **Fetch Requests**
   - Predicates
   - Sort Descriptors
3. **Persistent Stores**
   - Persistent Store Coordinator

1.17 When to use Core Data

- Core Data will fit most needs
- 3rd party add-ons like *Magical Record* make it easier and much less verbose
- However Core Data is not recommended for:
    1. Large object counts
    2. Bulk object updates
    3. Fulltext indexing
- For these cases, SQLite is often better

Further Reading:
Switching from Core Data - http://inessential.com/2010/02/26/on_switching_away_from_core_data
Magical Record - https://github.com/magicalpanda/MagicalRecord

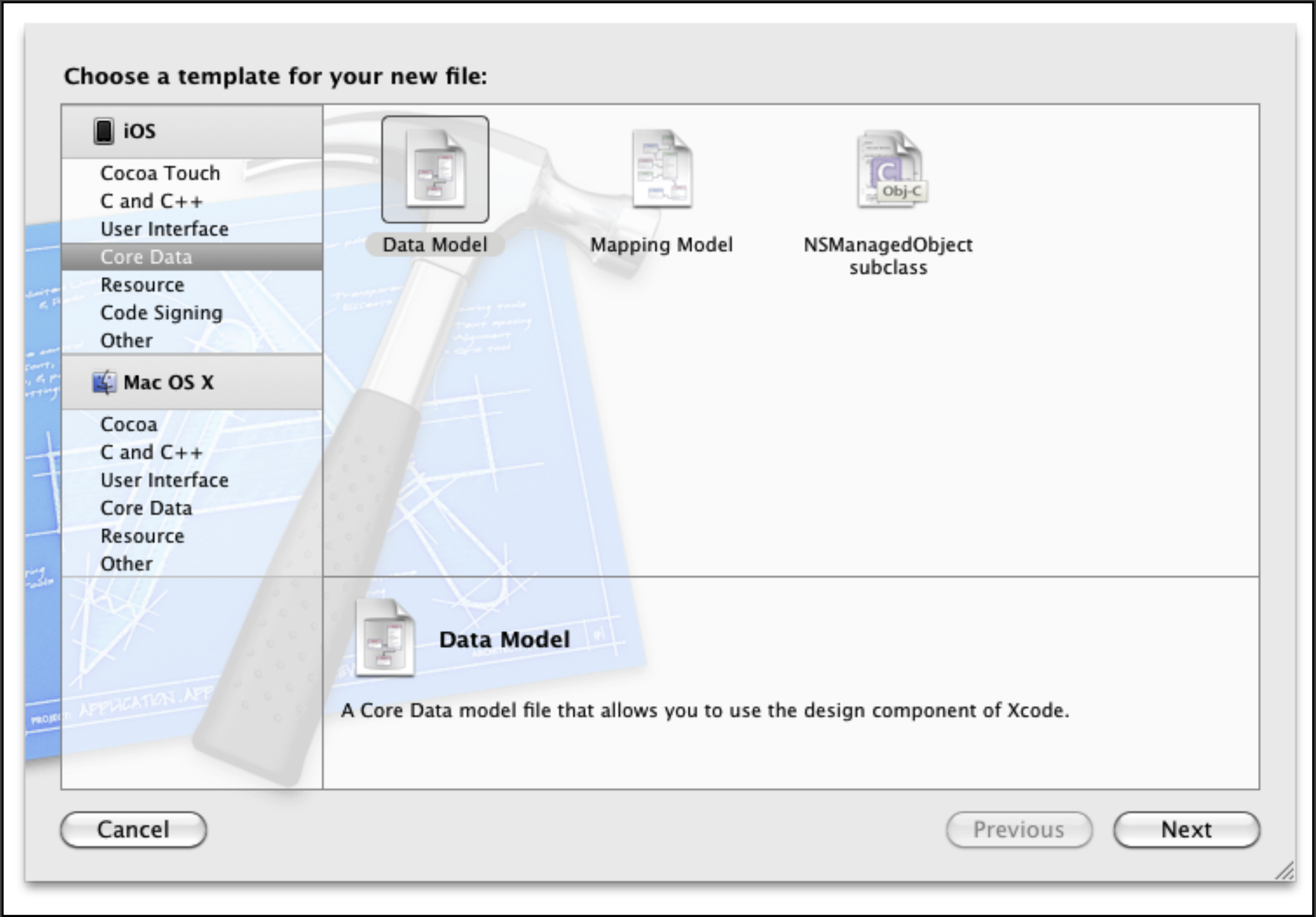
What's On For Today?

1. Core Data Concepts
2. Creating a Managed Model
3. Retrieving Data
4. CRUD Operations
5. Notification Center

2.0 Creating a Managed Model

Before using Core Data we need to setup a managed model:

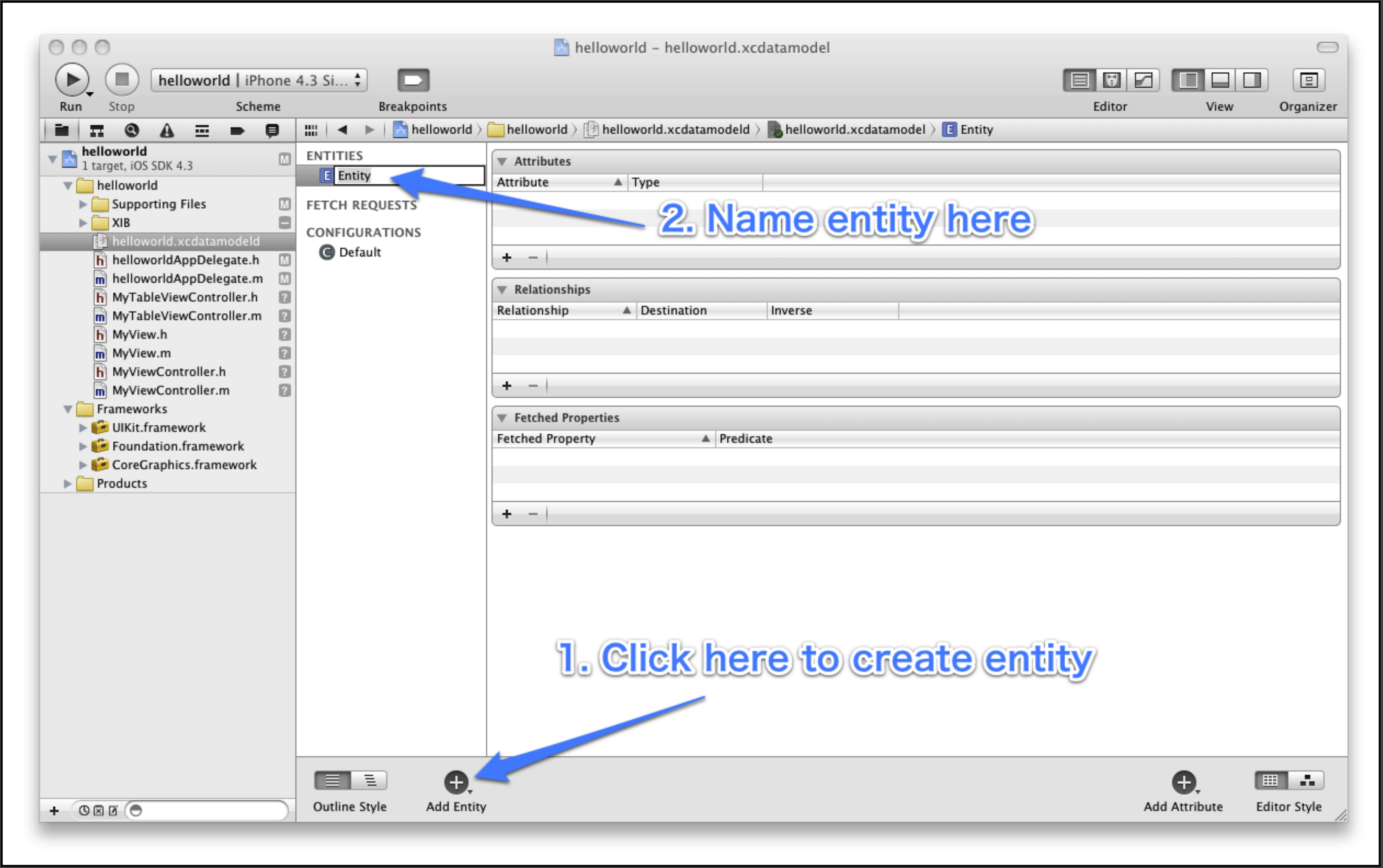
1. Create a Data Model
2. Add Entities
3. Add Attributes (Properties)
4. Define Relationships, if any
5. Create Managed Object Classes


Choose a template for your new file:

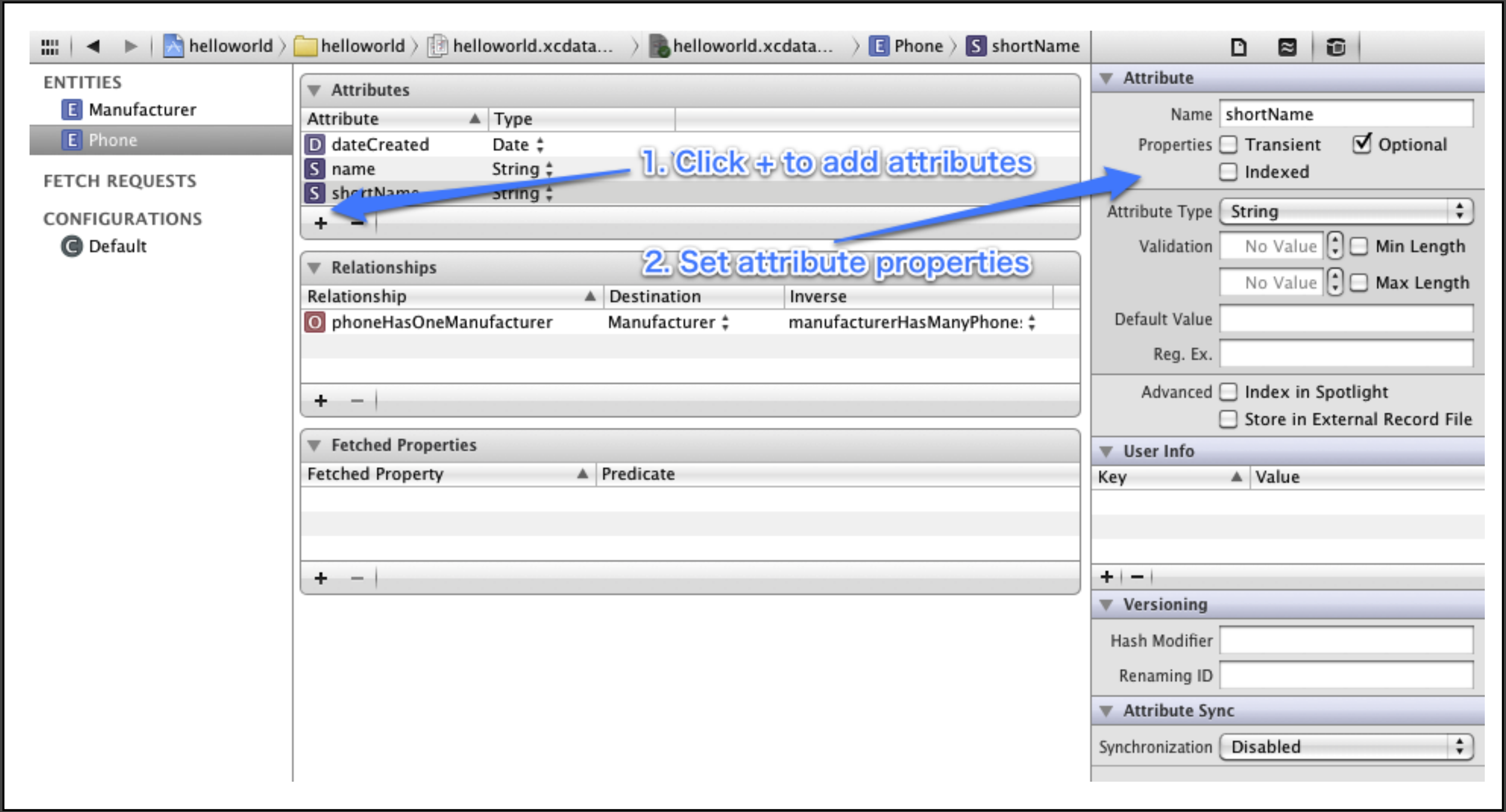
Data Model

A Core Data model file that allows you to use the design component of Xcode.

2.0.1 Create a Data Model



2.0.2 Add Entities



1. Click + to add attributes

2. Set attribute properties

ENTITIES
- Manufacturer
- Phone

FETCH REQUESTS

CONFIGURATIONS
- Default

Attributes

Attribute	Type
dateCreated	Date
name	String
shortName	String

Relationships

Relationship	Destination	Inverse
phoneHasOneManufacturer	Manufacturer	manufacturerHasManyPhone:

Fetched Properties

Fetched Property	Predicate

Attribute

Name: shortName
Properties: Transient [ ] Optional [✓] Indexed [ ]
Attribute Type: String
Validation: No Value Min Length [ ]
No Value Max Length [ ]
Default Value:
Reg. Ex.:
Advanced: Index in Spotlight [ ] Store in External Record File [ ]

User Info

Key	Value

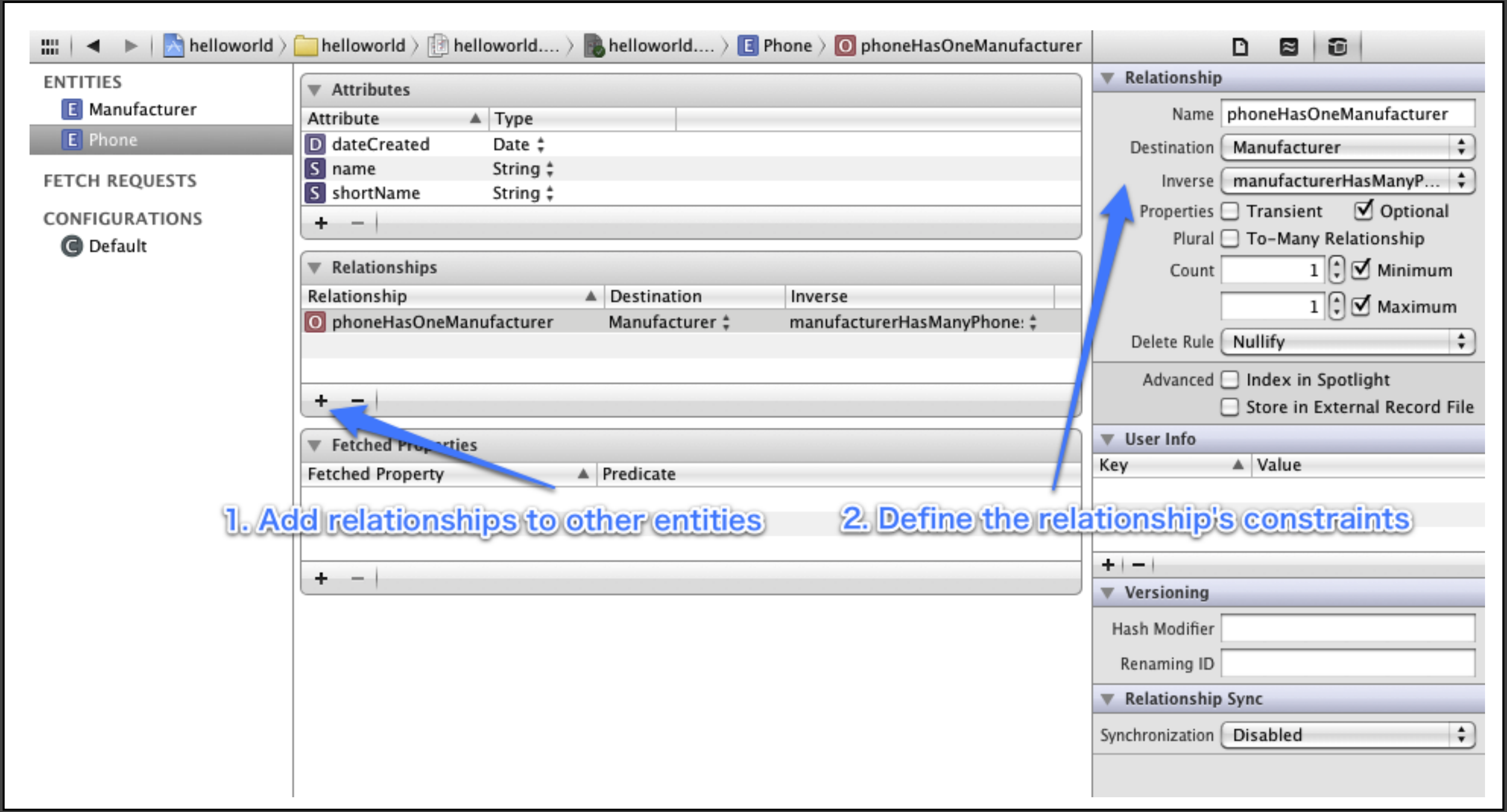
Versioning

Hash Modifier:
Renaming ID:

Attribute Sync

Synchronization: Disabled

2.0.3 Add Attributes



Attribute	Type
dateCreated	Date
name	String
shortName	String

Relationship	Destination	Inverse
phoneHasOneManufacturer	Manufacturer	manufacturerHasManyPhone:

Fetched Property	Predicate

Name	phoneHasOneManufacturer
Destination	Manufacturer
Inverse	manufacturerHasManyP...
Properties	☐ Transient ☑ Optional
Plural	☐ To-Many Relationship
Count	1 ☑ Minimum
	1 ☑ Maximum
Delete Rule	Nullify
Advanced	☐ Index in Spotlight
	☐ Store in External Record File
Hash Modifier	
Renaming ID	
Synchronization	Disabled

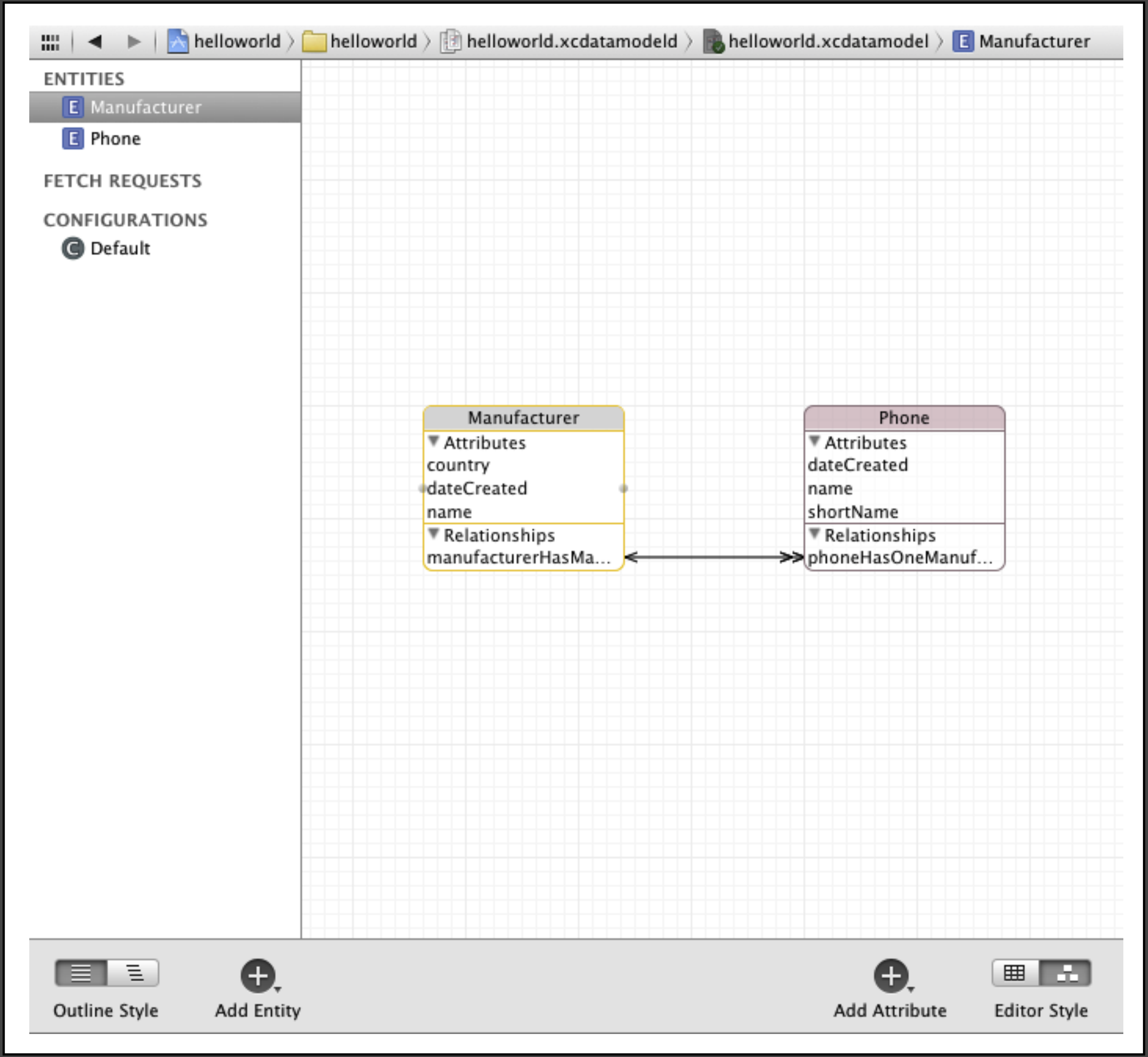
Key	Value

1. Add relationships to other entities

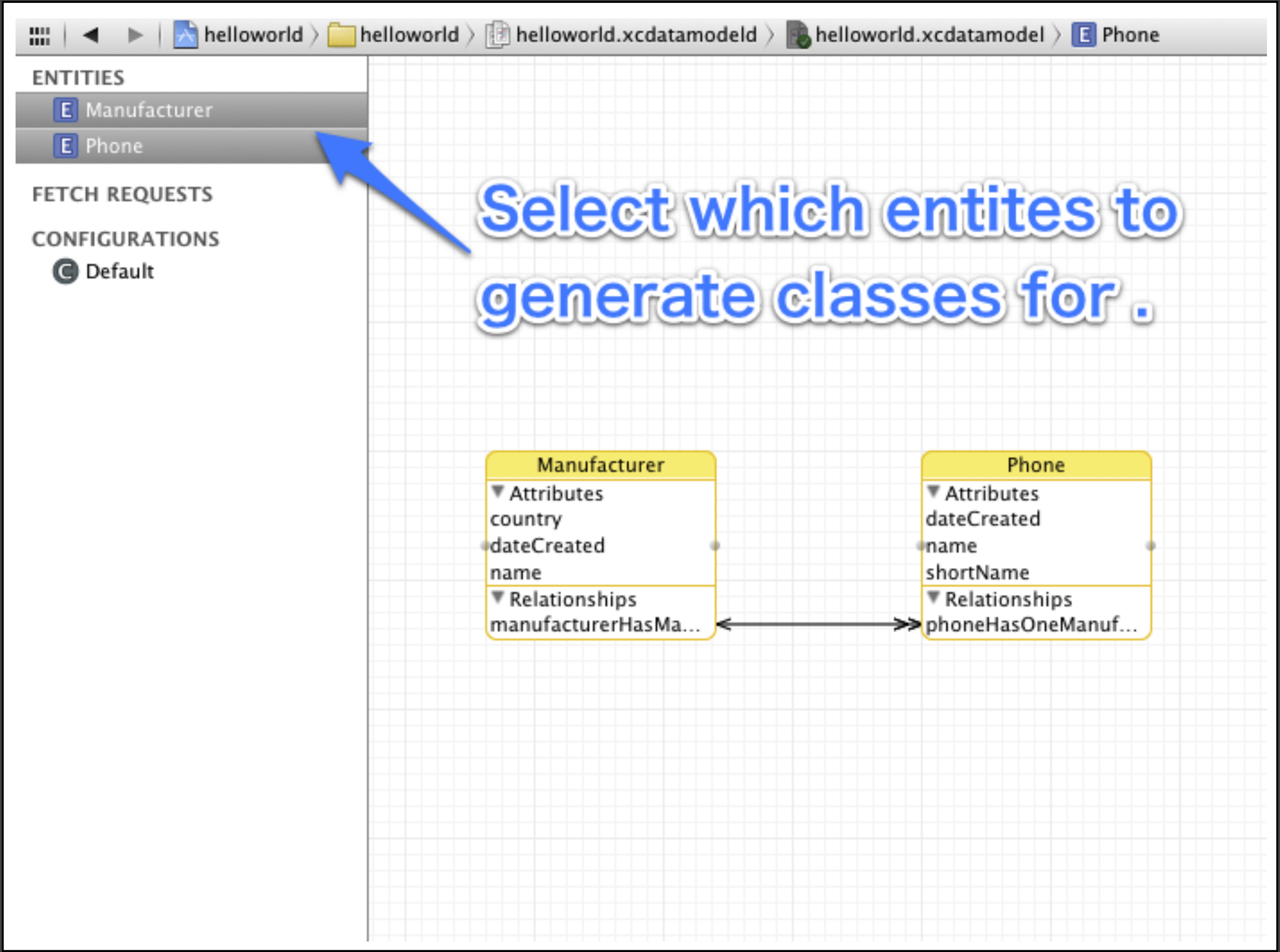
2. Define the relationship's constraints

2.0.4 Define Relationships

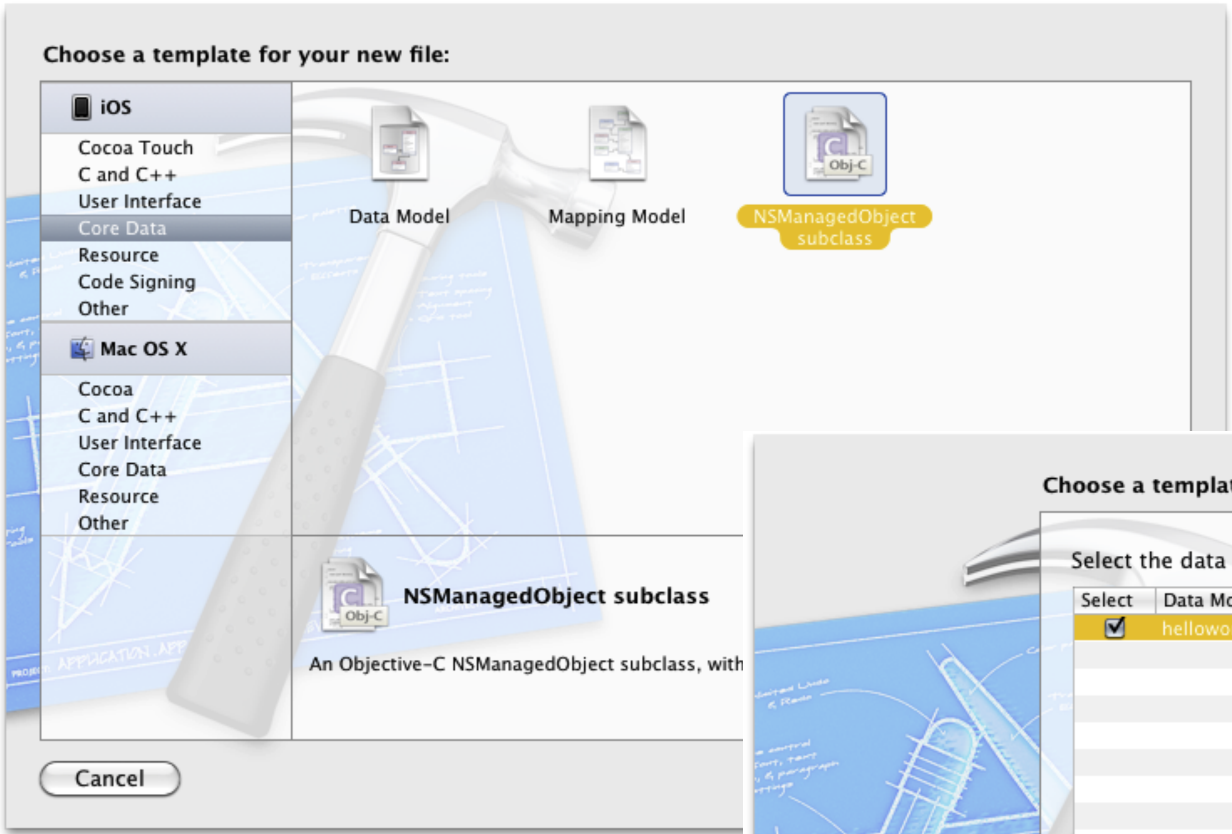
Swinburne University of Technology



2.0.5 Completed Schema

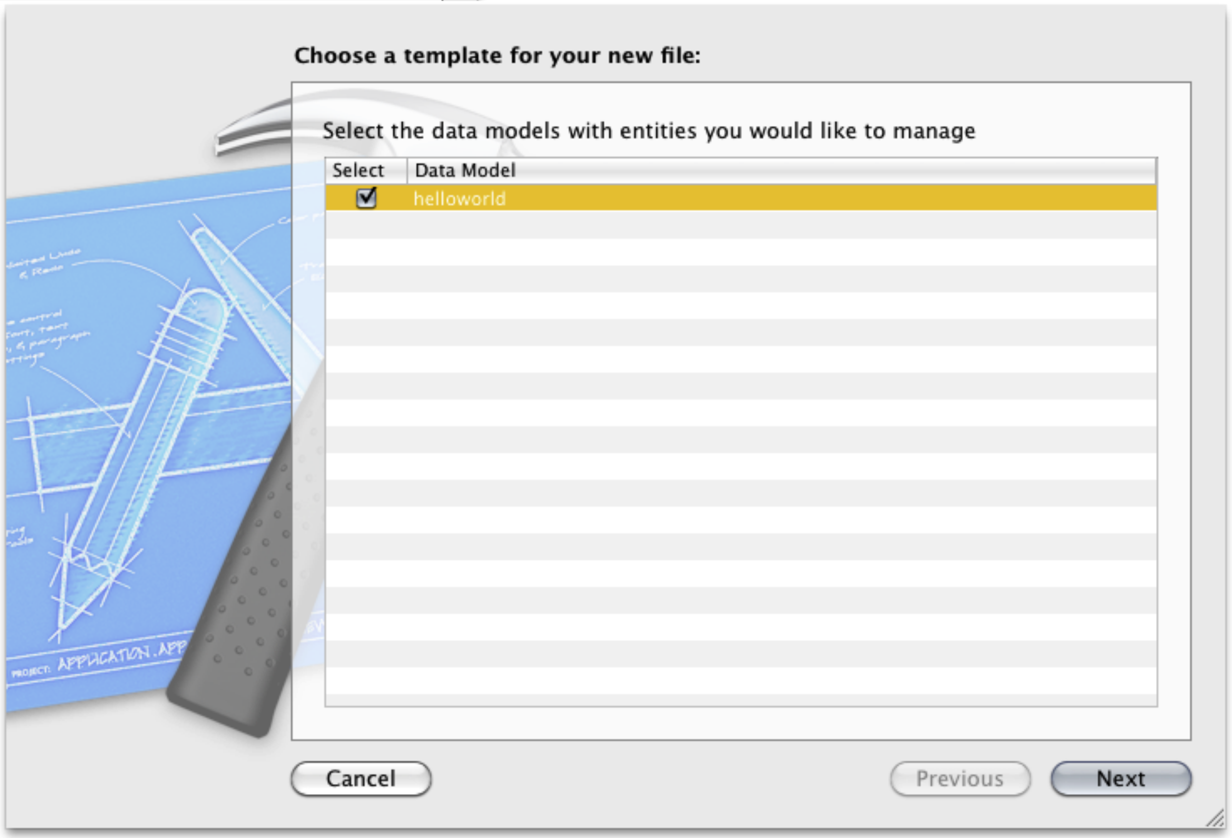


2.0.6 Creating Managed Objects - Select Entities



NSManagedObject subclass

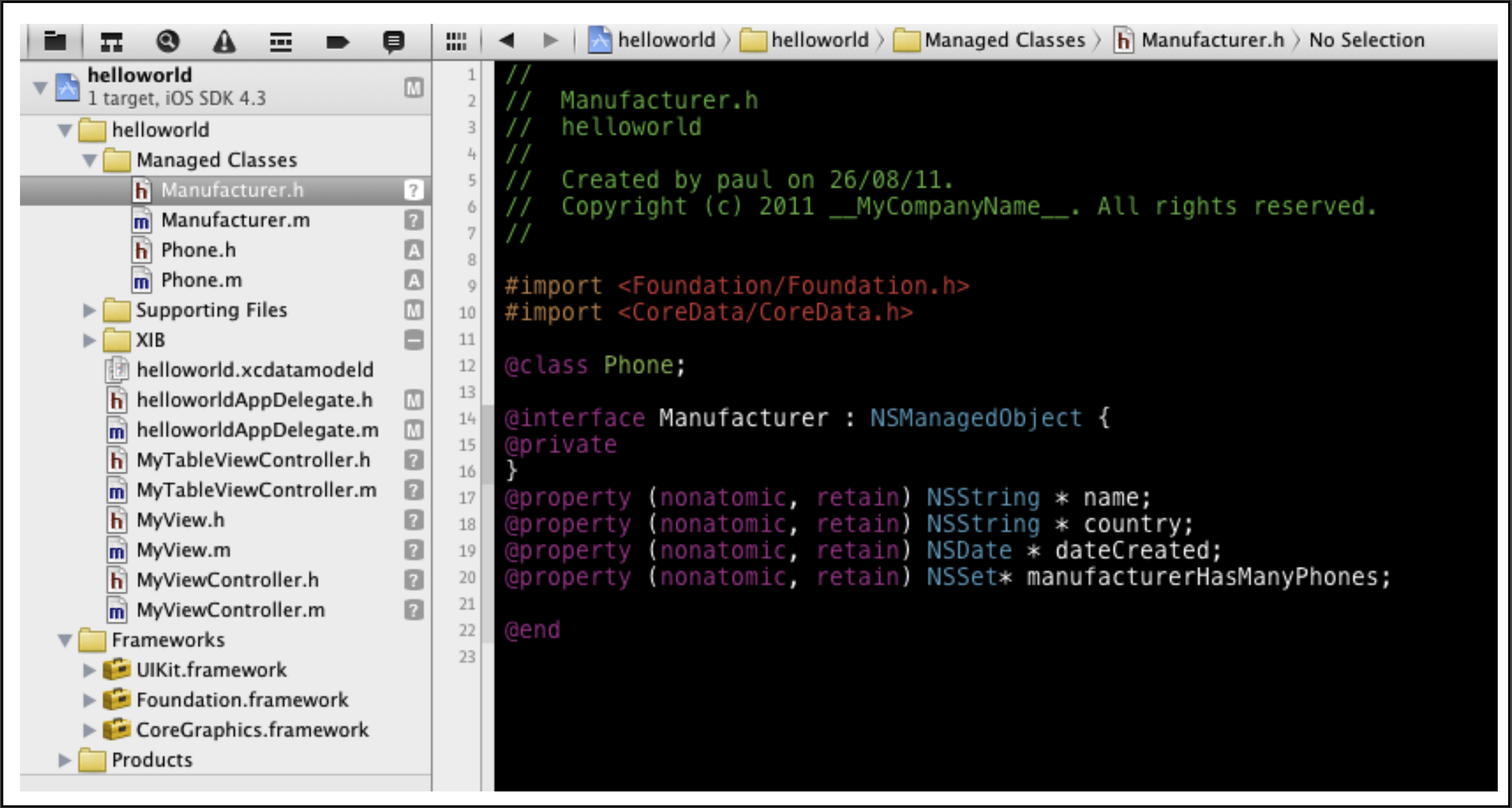
An Objective-C NSManagedObject subclass, with



Select the data models with entities you would like to manage

Select	Data Model
✓	helloworld

2.0.7 Creating Managed Objects - Generate Classes



```
//
//  Manufacturer.h
//  helloworld
//
//  Created by paul on 26/08/11.
//  Copyright (c) 2011 __MyCompanyName__. All rights reserved.
//

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

@class Phone;

@interface Manufacturer : NSManagedObject {
@private
}
@property (nonatomic, retain) NSString * name;
@property (nonatomic, retain) NSString * country;
@property (nonatomic, retain) NSDate * dateCreated;
@property (nonatomic, retain) NSSet* manufacturerHasManyPhones;

@end
```

2.0.8 Generated Class Files in Tree

What's On For Today?

1. Core Data Concepts
2. Creating a Managed Model
3. Retrieving Data
4. CRUD Operations
5. Notification Center

3.0 Retrieving Data

1. Create Managed Object Model **= DONE**
2. Setup App Delegate
3. Pass Core Data objects to *view controller*
4. Create a Fetch Request

Reminder:

CRUD = Create - Retrieve - Update - Delete

3.1 Setup App Delegate Header

Include Core Data headers:

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

Add 3 declared properties:

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

3.2 Setup App Delegate Implementation

Synthesize your declared properties:

```
@synthesize managedObjectContext, managedObjectModel,
  persistentStoreCoordinator;
```

*Add custom getters:**

```
- (NSManagedObjectContext *)managedObjectContext;

- (NSManagedObjectModel *)managedObjectModel;

- (NSPersistentStoreCoordinator *)persistentStoreCoordinator;
```

**These are templated for you by Xcode*

```
// Returns 'managed object context' for application
- (NSManagedObjectContext *)managedObjectContext {

  if (managedObjectContext != nil) {
      return managedObjectContext;
  }
  NSPersistentStoreCoordinator *coordinator =
                        [self persistentStoreCoordinator];
  if (coordinator != nil) {
    managedObjectContext =
                    [[NSManagedObjectContext alloc] init];
    [managedObjectContext
              setPersistentStoreCoordinator:coordinator];
  }
  return managedObjectContext;
}
```

3.2.1 ManagedObjectContext Get Method

```
// Returns 'managed object model' for application
- (NSManagedObjectModel *)managedObjectModel {

    if (managedObjectModel != nil) {
        return __managedObjectModel;
    }
    NSURL *modelURL = [[NSBundle mainBundle]
       URLForResource:@"coreDataHello" withExtension:@"momd"];

    managedObjectModel = [[NSManagedObjectModel alloc]
                              initWithContentsOfURL:modelURL];
    return managedObjectModel;
}
```

3.2.2 ManagedObjectModel Get Method

```
// Returns 'persistent store coordinator' for application
- (NSPersistentStoreCoordinator *)persistentStoreCoordinator {

  if (persistentStoreCoordinator != nil) {
      return persistentStoreCoordinator;
  }
  NSURL *storeURL = [[self applicationDocumentsDirectory]
      URLByAppendingPathComponent: @"yourCoreDataStore.sqlite"];

  NSError *error = nil;

  persistentStoreCoordinator = [[NSPersistentStoreCoordinator
   alloc] initWithManagedObjectModel:[self managedObjectModel]];

  // continued next slide ...
```

3.2.3 PersistentStoreCoordinator Get Method

```
  // ... continued
  if (![persistentStoreCoordinator
        addPersistentStoreWithType:NSSQLiteStoreType
        configuration:nil URL:storeURL options:nil error:&error])
  {
      NSLog(@"Error %@, %@", error, [error userInfo]);
      abort();
  }
  return persistentStoreCoordinator;
}
```

3.2.4 PersistentStoreCoordinator Get Method (cont)

```
// Pass MOC into the Navigation Controller's root VC
- (void)awakeFromNib {

  RootViewController *rootVC =(RootViewController *)
                  [self.navigationController topViewController];

  rootVC.managedObjectContext = self.managedObjectContext;
}
```

3.2.5 Pass MOC to View Controller

3.3 Meanwhile, In Your View Controller

We create Entity Description, Request and Sort Descriptor objects.

```
NSFetchRequest *request = [[NSFetchRequest alloc] init];

NSEntityDescription *entity = [NSEntityDescription entityForName:
  @"Phone" inManagedObjectContext:managedObjectContext];

// set request's entity
[request setEntity:entity];
```

3.4 Optional: Sort Descriptor & Predicate

We create Entity Description, Request and Sort Descriptor objects.

```
NSSortDescriptor *sortDesc = [[NSSortDescriptor alloc]
  initWithKey:@"name" ascending:NO];

// wrap in array - you can sort on multiple criteria
NSArray *sortDescArray = [NSArray arrayWithObject:sortDesc];
[request setSortDescriptors:sortDescArray]; // add to req obj
[sortDescriptor release];

NSPredicate *predicate = [NSPredicate predicateWithFormat:
 @"(name like %@) or (name like %@)", @"iPhone", @"Android"];

[request setPredicate:predicate]; // add to req obj
[predicate release];
```

3.5 Core Data, Fetch!

Run fetch request and store objects in an array:

```
NSArray *results = [managedObjectContext executeFetchRequest:
  request error:&error];
```

Or store a mutable copy instead:

```
NSMutableArray *mutableResults = [[managedObjectContext
  executeFetchRequest:request error:&error] mutableCopy];
```

What's On For Today?

1. Core Data Concepts
2. Creating a Managed Model
3. Retrieving Data
4. CRUD Operations
5. Notification Center

4.0 CRUD Operations

C = Create (Create Managed Object)
R = Retrieve (Fetch Request)
U = Update (Update Managed Object)
D = Delete (Remove from Store)

4.1 Creating a Managed Object

Create and save an NSManagedObject in a Managed Object Context object.

```
NSManagedObjectContext *context = /* assume this exists */;
NSManagedObject *newPhone = [NSEntityDescription
  insertNewObjectForEntityForName:@"Phone"
  inManagedObjectContext:context];

[newPhone setValue:@"iPhone 5" forKey:@"shortName"];
[newPhone setValue:@"Apple iPhone 5G" forKey:@"name"];
[newPhone setValue:[NSDate date] forKey:@"dateCreated"];

// Tell context to "save" unsaved changes
NSError *error = NULL;
if (![context save:&error]) {
    NSLog(@"Error on save: %@", [error localizedDescription]);
}
```

4.2 Managed Objects Inherit form NSManagedObject

A managed object's generated class inherits from NSManagedObject so instead of this:

```
NSManagedObject *newPhone = [NSEntityDescription
  insertNewObjectForEntityForName:@"Phone"
  inManagedObjectContext:context];
```

We can write this:

```
Phone *newPhone = (Phone*)[NSEntityDescription
  insertNewObjectForEntityForName:@"Phone"
  inManagedObjectContext:context];
```

4.3 Retrieving Managed Objects (Reviewed)

```
// get context and create a request object
NSManagedObjectContext *context = /* assume this exists */;
NSFetchRequest *request = [[NSFetchRequest alloc] init];

// set the entity on the request object
NSEntityDescription *entity= [NSEntityDescription entityForName:
  @"Phone" inManagedObjectContext:managedObjectContext];
[request setEntity:entity];

// Run our 'fetch request' on context object
NSArray *results = [context executeFetchRequest:request
  error:nil];
```

Note: We have note used a predicate (filter) or assigned a sort descriptor for simplicity. See slides 3.3 - 3.5 for a detailed fetch request example.

4.4 Updating a Managed Object

Continuing the previous example, let's update the first object returned by the fetch request.

```
// get first object in results from previous slide
Phone *phoneObject = [results objectAtIndex:0];

// change a value on it
[phoneObject setValue:@"iPhone 5.1" forKey:@"shortName"];

// tell context to "save" unsaved changes
NSError *error;
if (![context save:&error]) {
    NSLog(@"Error on save: %@", [error localizedDescription]);
}
```

4.5 Deleting Managed Objects

A delete object message needs to be sent to the Managed Object Context as follows.

```
NSManagedObjectContext *context = /* assume this exists */;

// remove phoneObject from context
[context deleteObject:phoneObject];

// tell context to "save" unsaved changes
NSError *error;
if (![context save:&error]) {
    NSLog(@"Error on save: %@", [error localizedDescription]);
}
```

What's On For Today?

1. Core Data Concepts
2. Creating a Managed Model
3. Retrieving Data
4. CRUD Operations
5. Notification Center

5.0 Notification Center

- A method for objects to broadcast notifications of state changes to other objects in memory
- An object 'posts' a notification
- Objects registered for that notification are alerted and allowed to respond

5.1 Posting A Notification

Call -postNotificationName: on the default NSNotificationCenter object

Specify i) notification name, ii) delegate object

```
// Posts a "doSomething" message
- (void)notify
{
  [[NSNotificationCenter defaultCenter]
     postNotificationName:@"doSomething" object:self];
}
```

5.2 Registering for Notification

Specify:

1. Object registering as an observer
2. Selector called when notification received
3. Name of the notification (a string)
4. Object being observed

```
// Meanwhile, in another object...
[[NSNotificationCenter defaultCenter] addObserver:self
   selector:@selector(myNotificationHandler:)
      name:@"doSomething" object:nil];
```

5.3 Cleaning Up

When a class is deallocated, its observers must be removed, otherwise they might be called on a nil object.

```
// Typically called in observer's -dealloc method
- (void) dealloc
{
  [[NSNotificationCenter defaultCenter]
                                        removeObserver:self];
}
```

5.4 When to Use Notifications

Tightly coupled objects create problems. Ideally objects have no knowledge of another object's implementation details.

Post Notifications are useful for:

- Decoupling objects
- Notifying more than one object (without knowing what they are)

Read More:
http://developer.apple.com/library/mac/#documentation/Cocoa/Reference/Foundation/Classes/NSNotificationCenter_Class/Reference/Reference.html

What We Covered Today

1. Core Data Concepts
2. Creating a Managed Model
3. Retrieving Data
4. CRUD Operations
5. Notification Center

End of Lecture 5

1. Lab
2. Assignments
3. Aiight