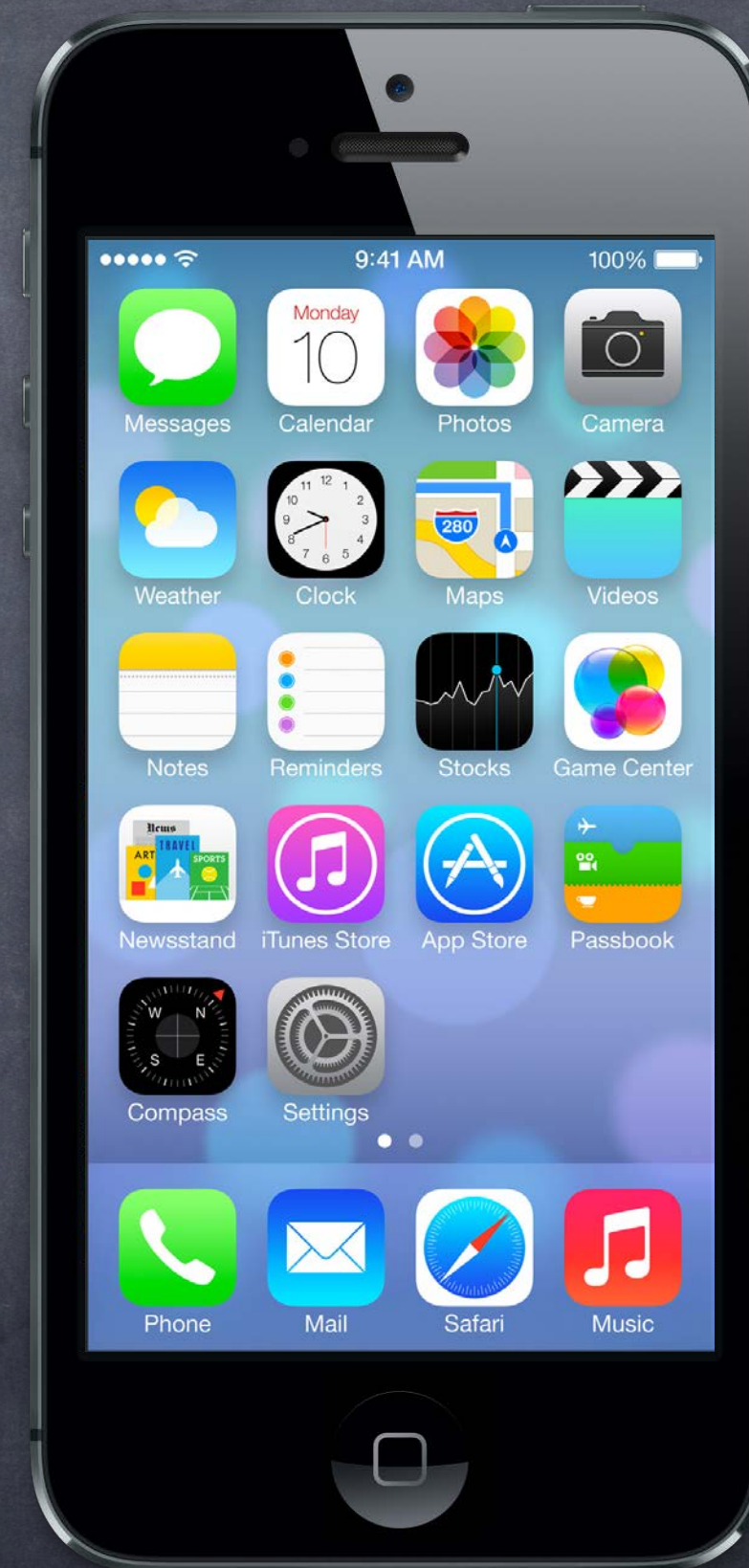


Stanford CS193p

Developing Applications for iOS
Fall 2013-14



Agenda

• Core Data

Storing your Model permanently in an object-oriented database.

• Homework

Assignment 5 due Wednesday.

Final homework (Assignment 6) will be assigned Wednesday, due the next Wednesday.

• Wednesday

Final Project Requirements

Core Data and UITableView

Core Data Demo

• Next Week

Multitasking

Advanced Segueing

Map Kit?

Core Data

• Database

Sometimes you need to store large amounts of data or query it in a sophisticated manner.
But we still want it to be object-oriented objects!

• Enter Core Data

Object-oriented database.

Very, very powerful framework in iOS (we will only be covering the absolute basics).

• It's a way of creating an object graph backed by a database

Usually backed by SQL (but also can do XML or just in memory).

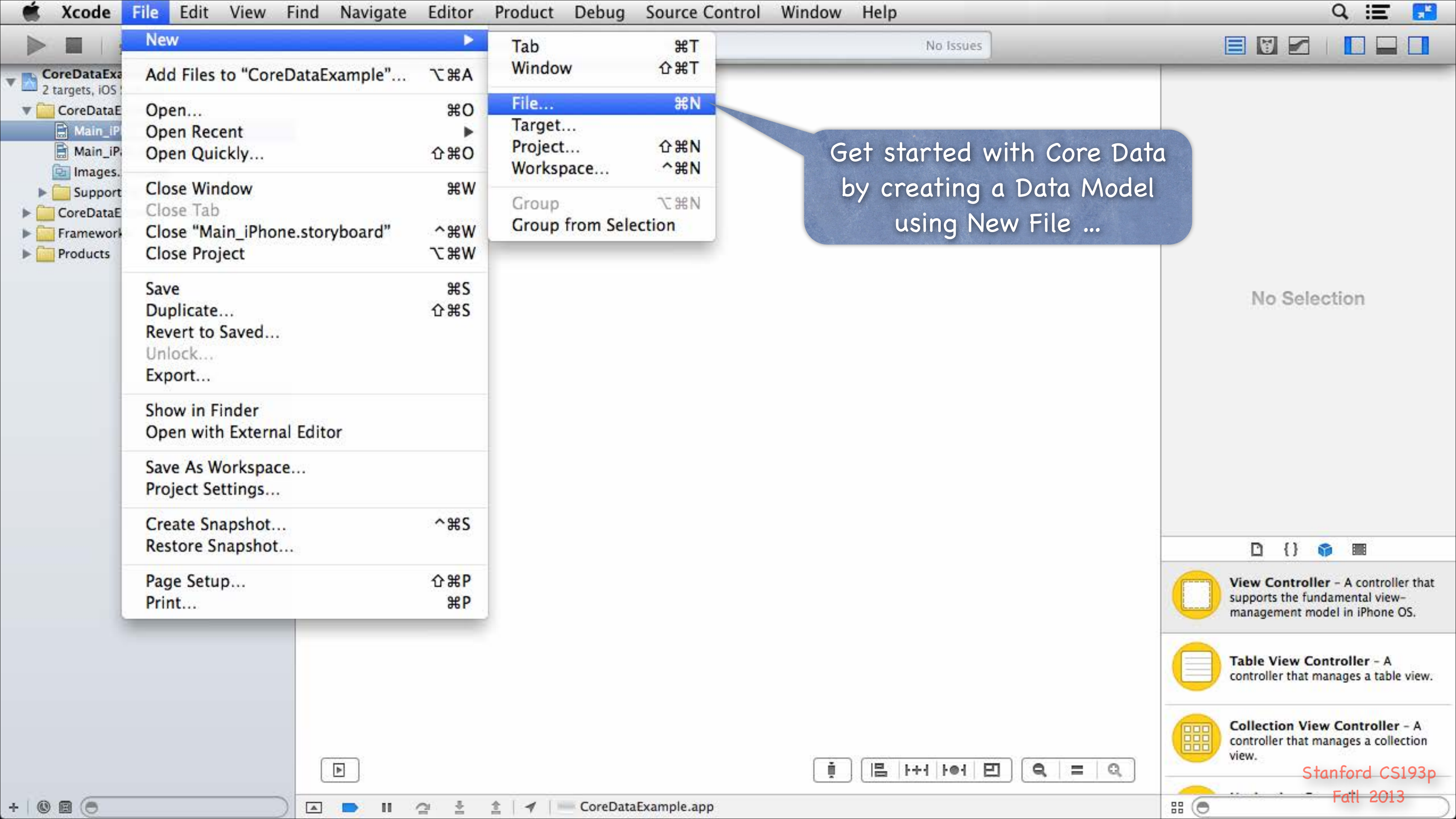
• How does it work?

Create a visual mapping (using Xcode tool) between database and objects.

Create and query for objects using object-oriented API.

Access the "columns in the database table" using @propertys on those objects.

Let's get started by creating that visual map ...



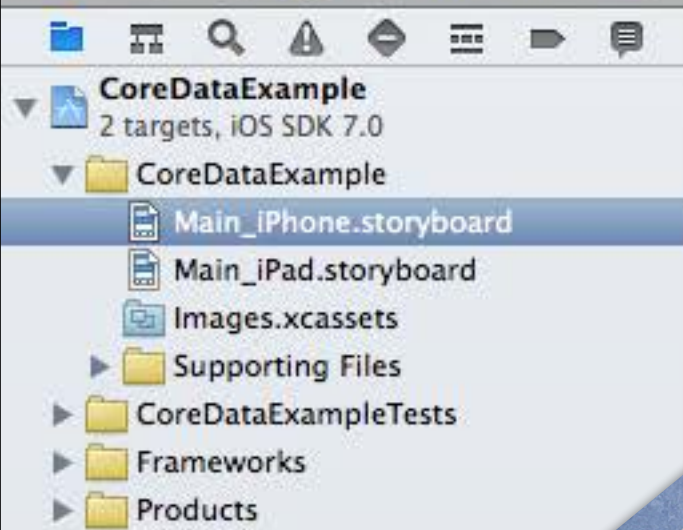
- New
- Add Files to "CoreDataExample"... ⌘⇧A
- Open... ⌘O
- Open Recent
- Open Quickly... ⌘⇧O
- Close Window ⌘W
- Close Tab
- Close "Main_iPhone.storyboard" ⌘⇧W
- Close Project ⌘⇧W
- Save ⌘S
- Duplicate... ⌘⇧S
- Revert to Saved...
- Unlock...
- Export...
- Show in Finder
- Open with External Editor
- Save As Workspace...
- Project Settings...
- Create Snapshot... ⌘⇧S
- Restore Snapshot...
- Page Setup... ⌘⇧P
- Print... ⌘P

- Tab Window ⌘T
- Window ⌘⇧T
- File... ⌘N**
- Target...
- Project... ⌘⇧N
- Workspace... ⌘⇧N
- Group ⌘⇧N
- Group from Selection

Get started with Core Data by creating a Data Model using New File ...





No Selection

- View Controller** - A controller that supports the fundamental view-management model in iPhone OS.
- Table View Controller** - A controller that manages a table view.
- Collection View Controller** - A controller that manages a collection view.



This section.

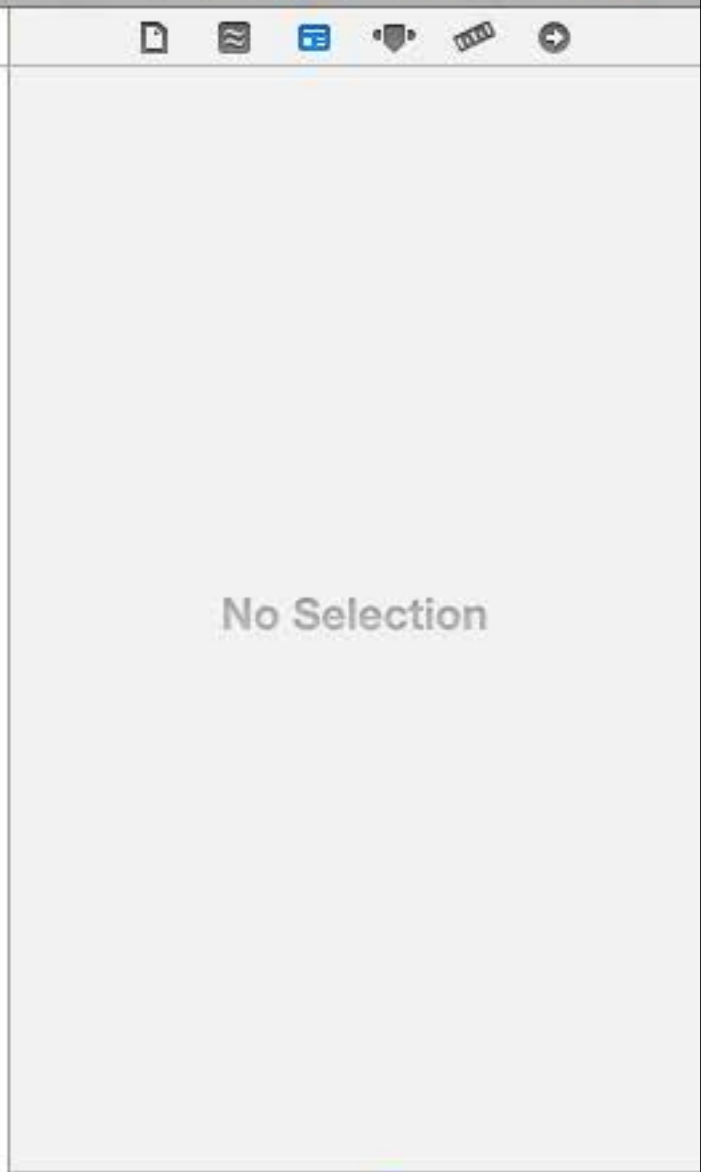
Choose a template for your new file:

iOS Cocoa Touch C and C++ User Interface Core Data Resource Other	 Data Model	 Mapping Model	 NSManagedObject subclass
OS X Cocoa C and C++ User Interface Core Data Resource Other	 Data Model A Core Data model file that allows you to use the design component of Xcode.		

Buttons: Cancel, Previous, Next

This template.

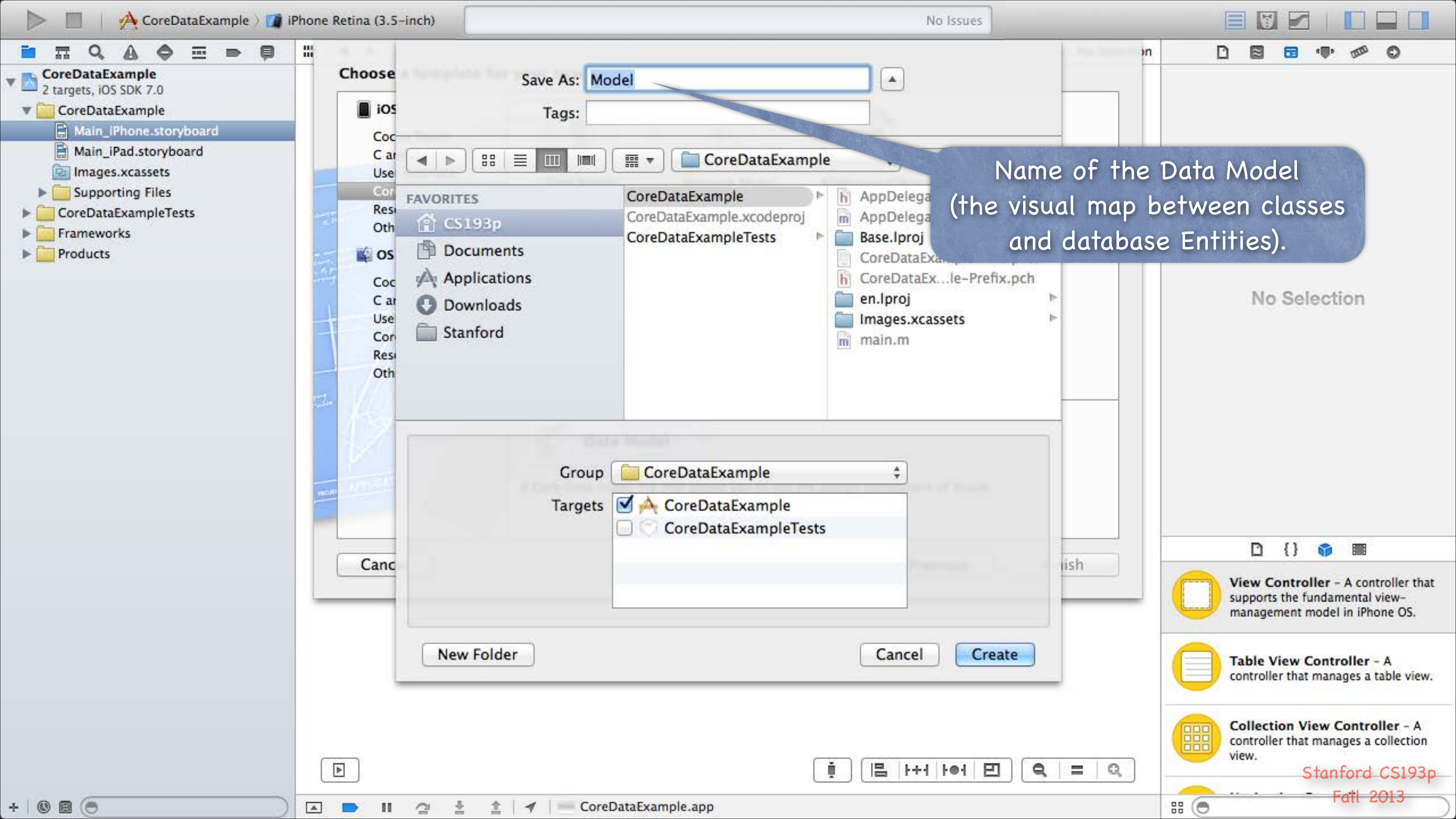
Don't accidentally pick this one.



View Controller - A controller that supports the fundamental view-management model in iPhone OS.

Table View Controller - A controller that manages a table view.

Collection View Controller - A controller that manages a collection view.



Save As: Model

Tags:

Name of the Data Model
(the visual map between classes
and database Entities).

Group CoreDataExample

- CoreDataExample
- CoreDataExampleTests

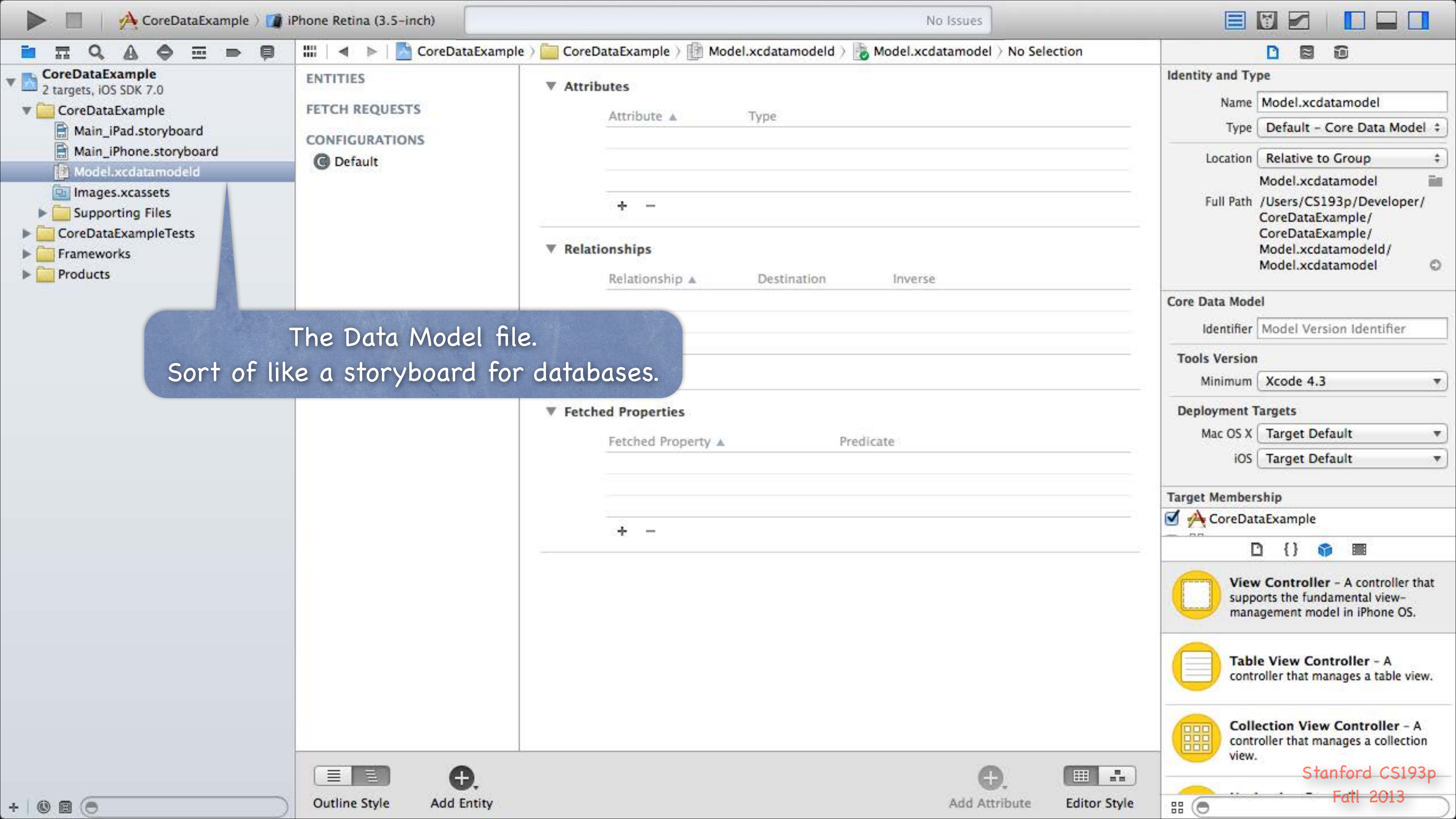
New Folder

Cancel

Create

No Selection

- View Controller** - A controller that supports the fundamental view-management model in iPhone OS.
- Table View Controller** - A controller that manages a table view.
- Collection View Controller** - A controller that manages a collection view.



The Data Model file.
Sort of like a storyboard for databases.

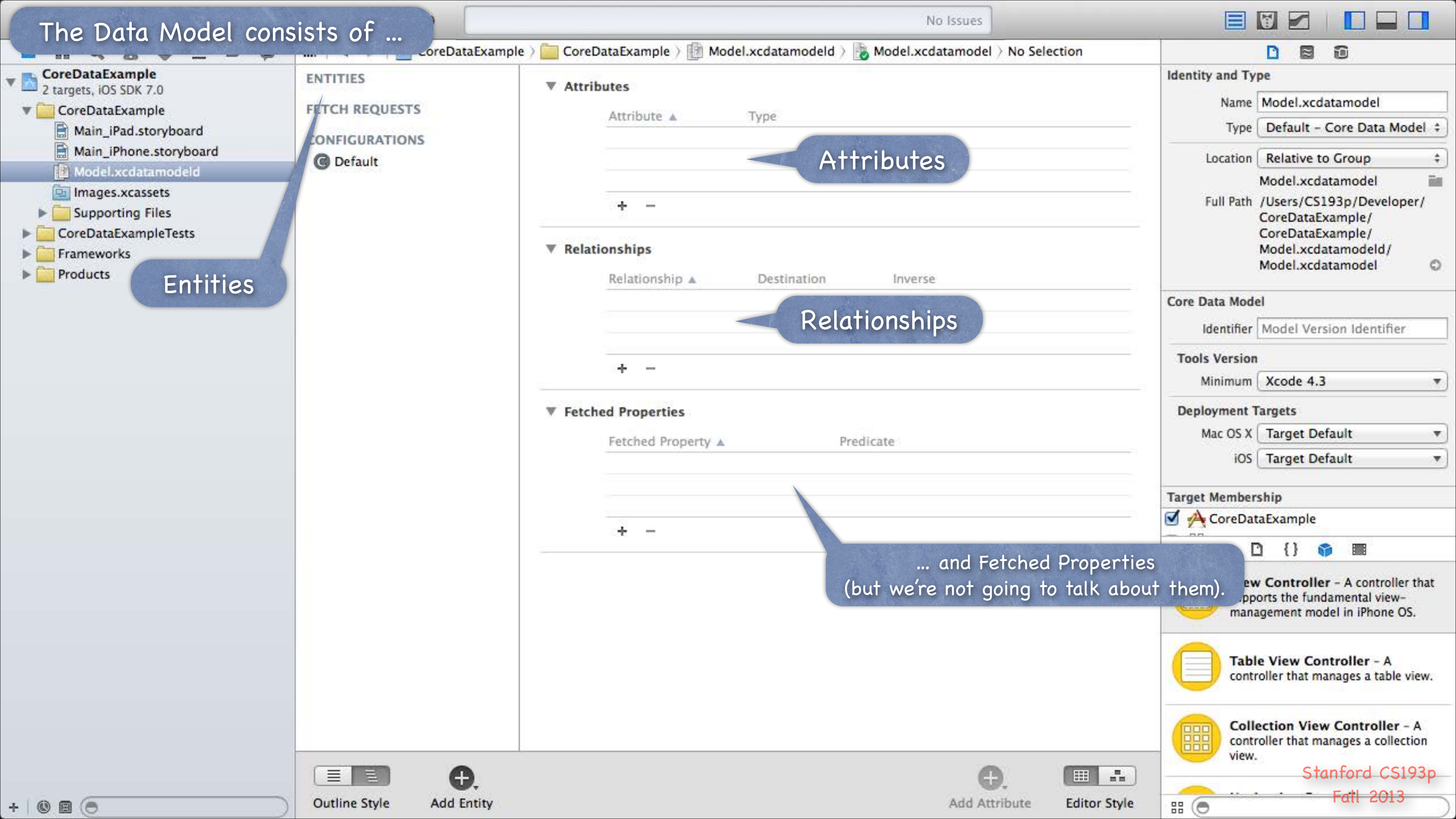
The Data Model consists of ...

Entities

Attributes

Relationships

... and Fetched Properties
(but we're not going to talk about them).



CoreDataExample
2 targets, iOS SDK 7.0

- CoreDataExample
 - Main_iPad.storyboard
 - Main_iPhone.storyboard
 - Model.xcdatamodeld
 - Images.xcassets
 - Supporting Files
 - CoreDataExampleTests
 - Frameworks
 - Products

CoreDataExample > CoreDataExample > Model.xcdatamodeld > Model.xcdatamodel > No Selection

ENTITIES

FETCH REQUESTS

CONFIGURATIONS

- Default

Attributes

Attribute ▲	Type
+	-

Relationships

Relationship ▲	Destination	Inverse
+	-	

Fetches Properties

Fetches Property ▲	Predicate
+	-

Identity and Type

Name: Model.xcdatamodel

Type: Default - Core Data Model

Location: Relative to Group

Model.xcdatamodel

Full Path: /Users/CS193p/Developer/CoreDataExample/CoreDataExample/Model.xcdatamodeld/Model.xcdatamodel

Core Data Model

Identifier: Model Version Identifier

Tools Version

Minimum: Xcode 4.3

Deployment Targets

Mac OS X: Target Default

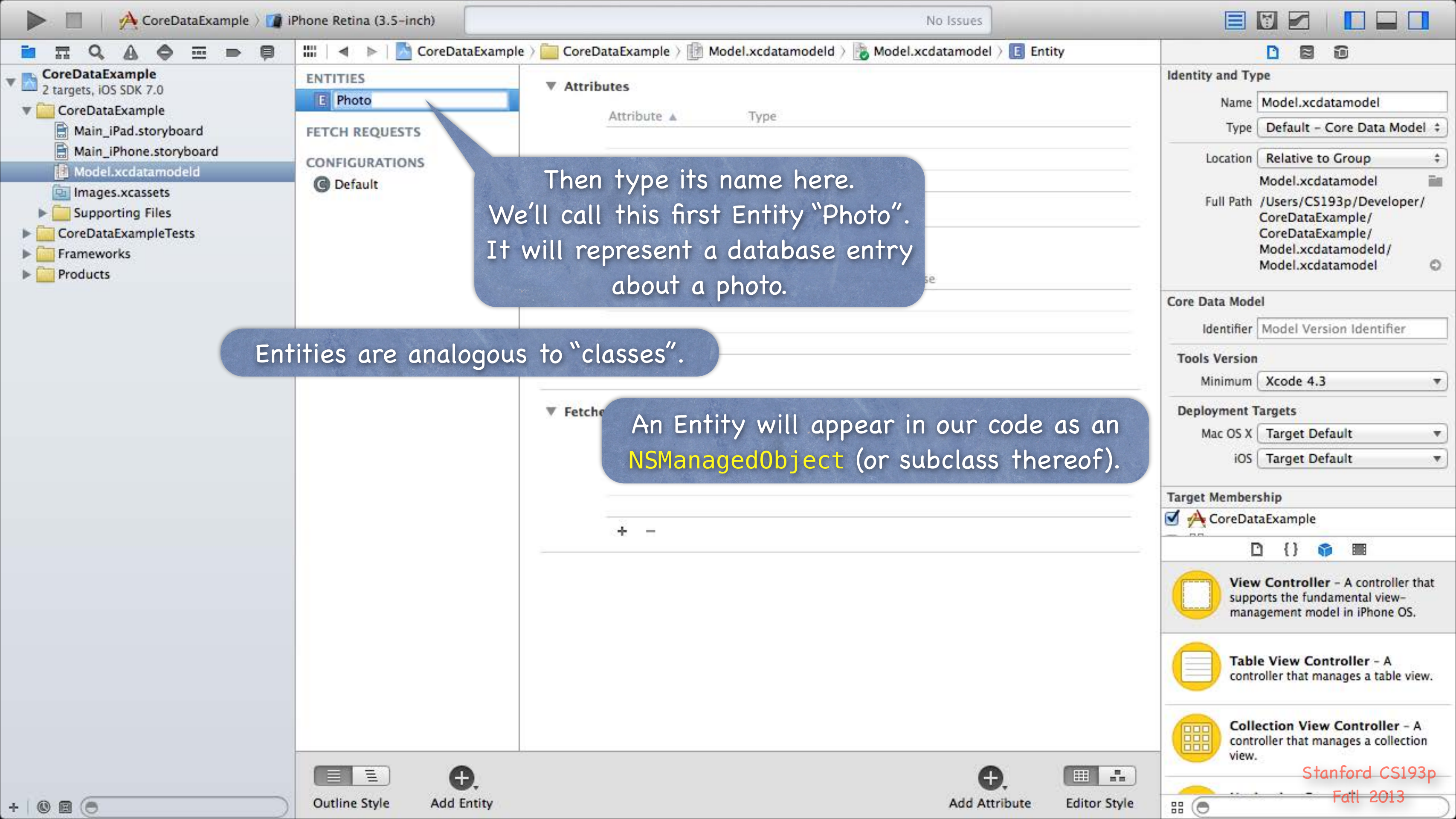
iOS: Target Default

Target Membership

CoreDataExample

Click here to add an Entity.

- Add Entity
- Add Fetch Request
- Add Configuration



Entities are analogous to "classes".

Then type its name here. We'll call this first Entity "Photo". It will represent a database entry about a photo.

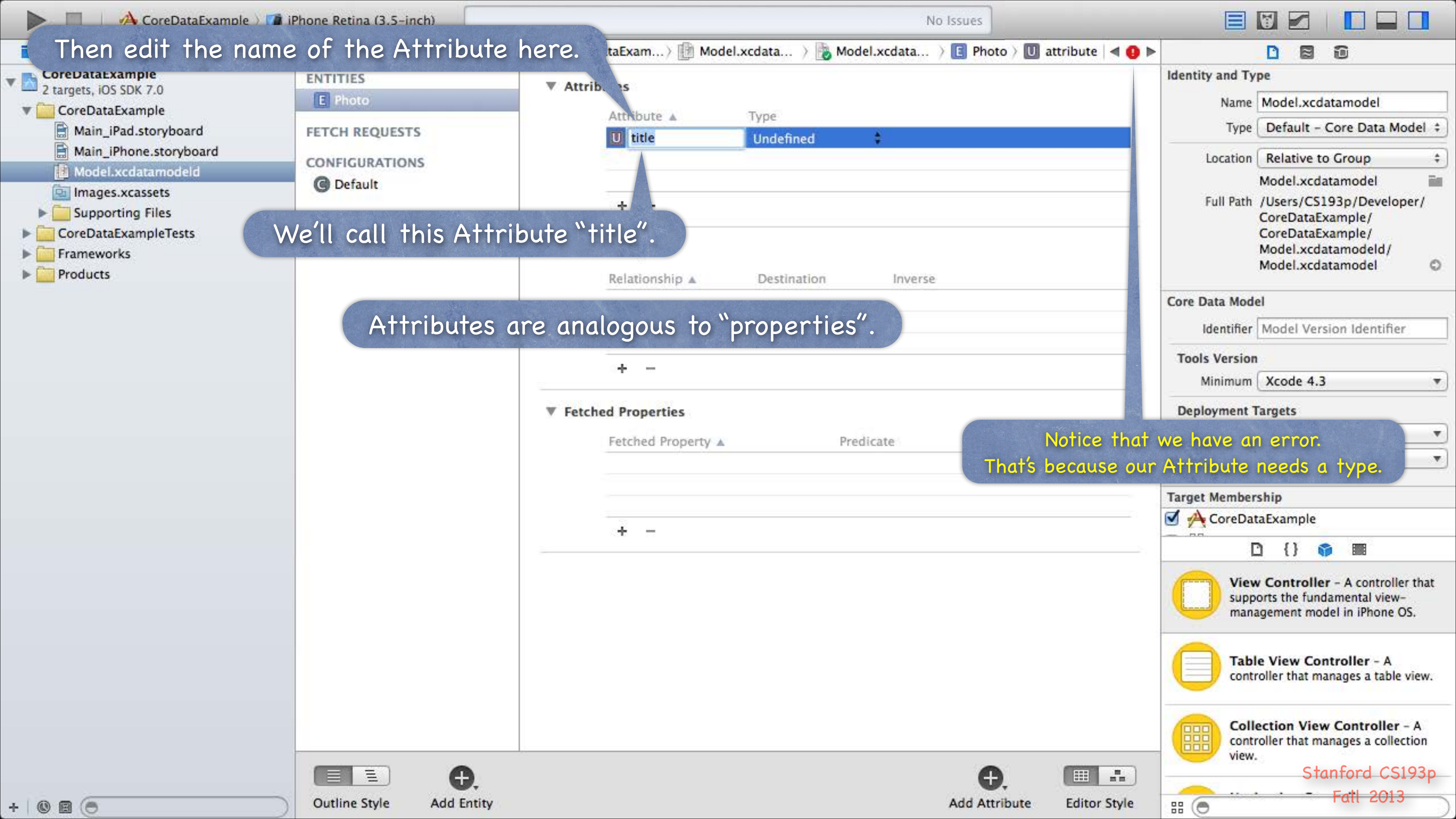
An Entity will appear in our code as an `NSManagedObject` (or subclass thereof).

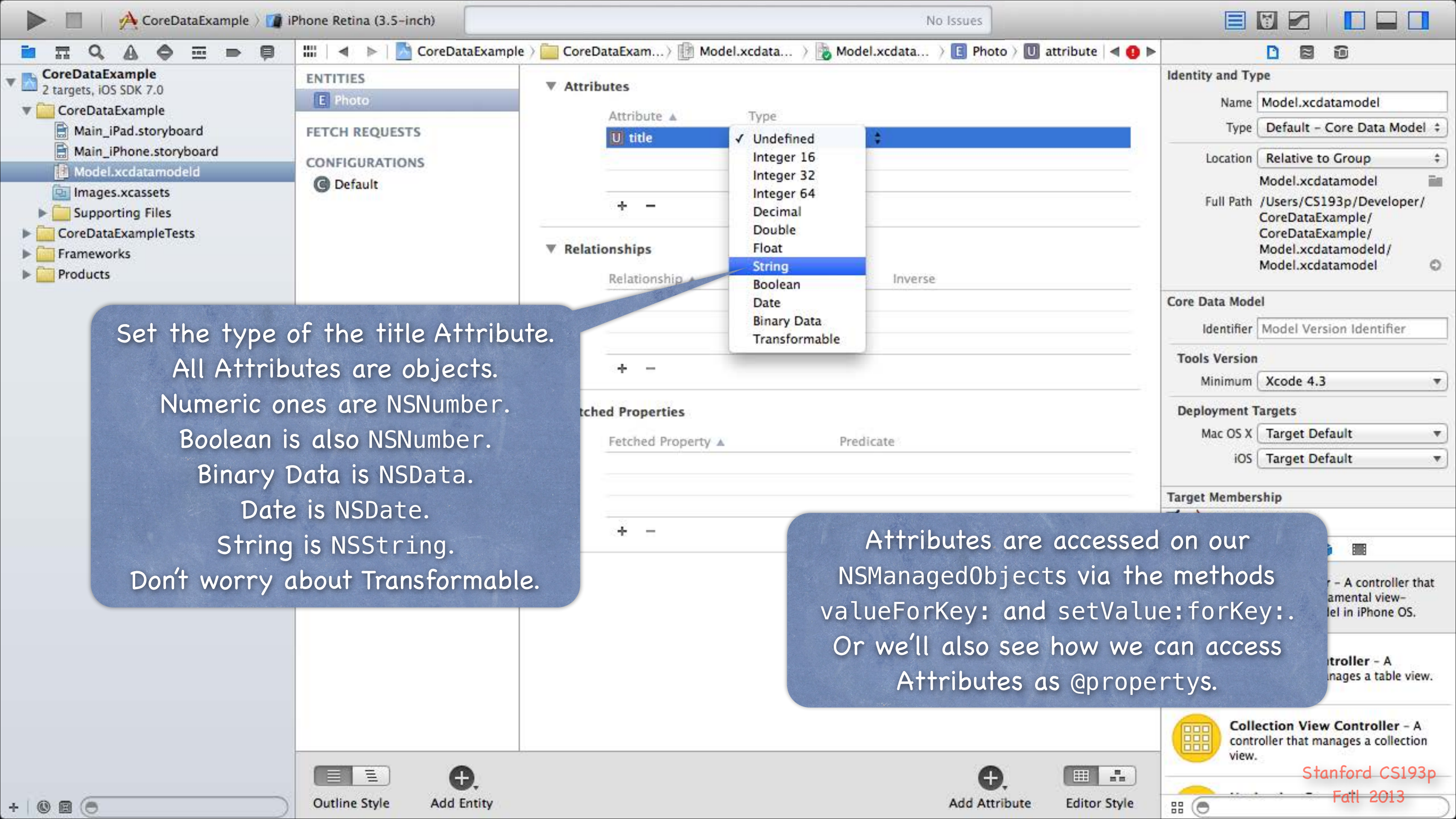
Then edit the name of the Attribute here.

We'll call this Attribute "title".

Attributes are analogous to "properties".

Notice that we have an error. That's because our Attribute needs a type.





Set the type of the title Attribute.
 All Attributes are objects.
 Numeric ones are NSNumber.
 Boolean is also NSNumber.
 Binary Data is NSData.
 Date is NSDate.
 String is NSString.
 Don't worry about Transformable.

Attributes are accessed on our
 NSManagedObjects via the methods
 valueForKey: and setValue:forKey:.
 Or we'll also see how we can access
 Attributes as @propertys.

CoreDataExample
2 targets, iOS SDK 7.0

- CoreDataExample
 - Main_iPad.storyboard
 - Main_iPhone.storyboard
 - Model.xcdatamodeld
 - Images.xcassets
 - Supporting Files
 - CoreDataExampleTests
 - Frameworks
 - Products

ENTITIES

- Photo

FETCH REQUESTS

CONFIGURATIONS

- Default

Attributes

Attribute	Type
S title	String

Relationships

Relationship	Destination	Inverse
--------------	-------------	---------

Fetches Properties

Fetches Property	Predicate
------------------	-----------

Identity and Type

Type: Default - Core Data Model

Location: Relative to Group
Model.xcdatamodeld

Full Path: /Users/CS193p/Developer/CoreDataExample/CoreDataExample/Model.xcdatamodeld/Model.xcdatamodel

Core Data Model

Identifier: Model Version Identifier

Tools Version

Minimum: Xcode 4.3

Deployment Targets

Mac OS X: Target Default

iOS: Target Default

Target Membership

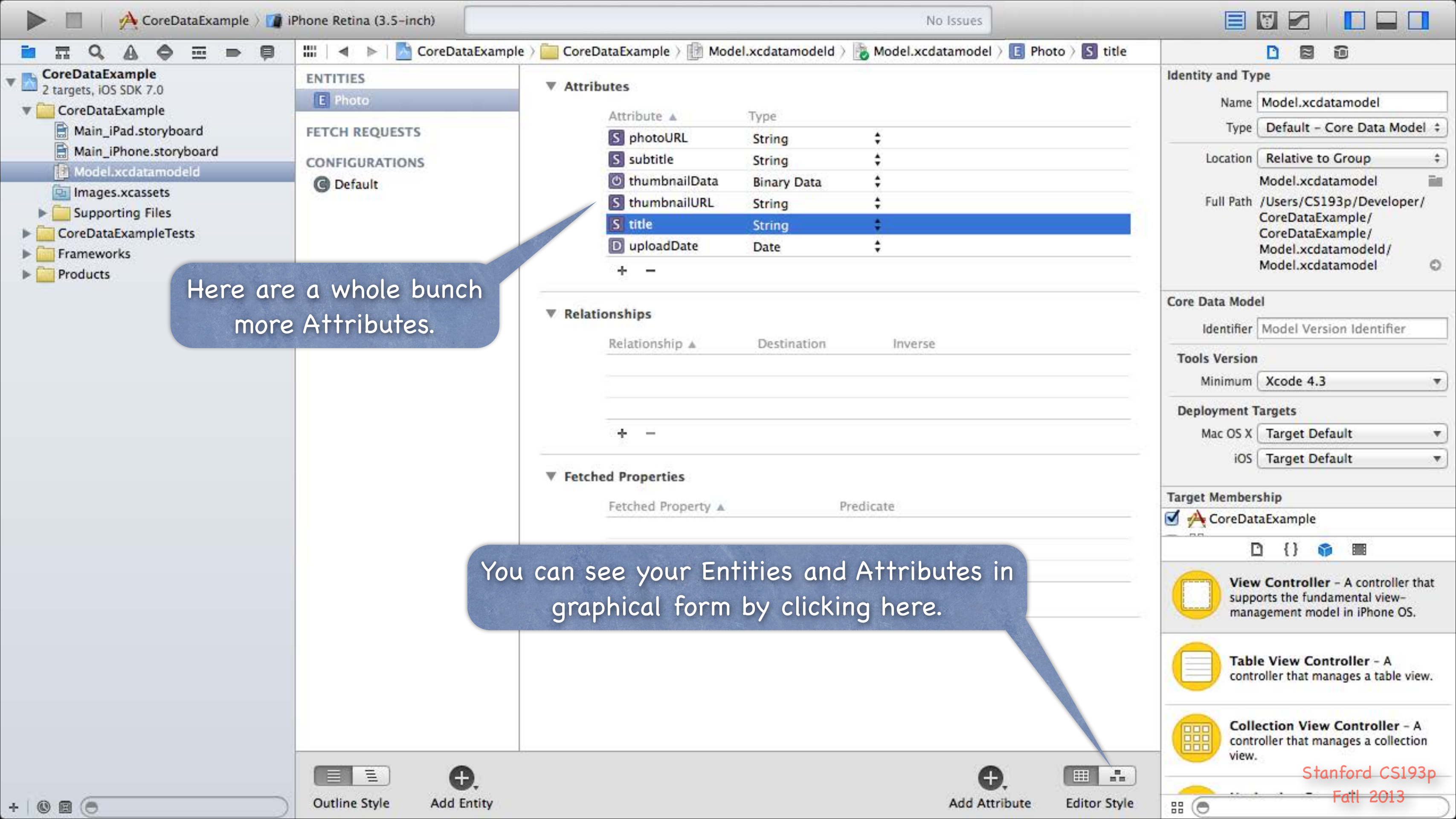
- CoreDataExample

View Controller - A controller that supports the fundamental view-management model in iPhone OS.

Table View Controller - A controller that manages a table view.

Collection View Controller - A controller that manages a collection view.

No more error!



Here are a whole bunch more Attributes.

You can see your Entities and Attributes in graphical form by clicking here.

CoreDataExample
2 targets, iOS SDK 7.0

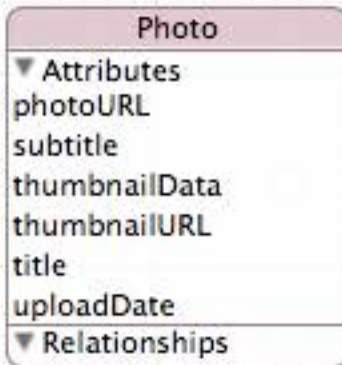
- CoreDataExample
 - Main_iPad.storyboard
 - Main_iPhone.storyboard
 - Model.xcdatamodeld
 - Images.xcassets
 - Supporting Files
 - CoreDataExampleTests
 - Frameworks
 - Products

ENTITIES
E Photo

FETCH REQUESTS

CONFIGURATIONS
C Default

This is the same thing we were just looking at, but in a graphical view.



Identity and Type

Name Model.xcdatamodel

Type Default - Core Data Model

Location Relative to Group
Model.xcdatamodel

Full Path /Users/CS193p/Developer/CoreDataExample/CoreDataExample/Model.xcdatamodeld/Model.xcdatamodel

Core Data Model

Identifier Model Version Identifier

Tools Version

Minimum Xcode 4.3

Deployment Targets

Mac OS X Target Default

iOS Target Default

Target Membership

- CoreDataExample

View Controller - A controller that supports the fundamental view-management model in iPhone OS.

Table View Controller - A controller that manages a table view.

Collection View Controller - A controller that manages a collection view.

CoreDataExample
2 targets, iOS SDK 7.0

- CoreDataExample
 - Main_iPad.storyboard
 - Main_iPhone.storyboard
 - Model.xcdatamodeld
 - Images.xcassets
 - Supporting Files
 - CoreDataExampleTests
 - Frameworks
 - Products

ENTITIES
E Photo

FETCH REQUESTS

CONFIGURATIONS
C Default

Photo

- Attributes
 - photoURL
 - subtitle
 - thumbnailData
 - thumbnailURL
 - title
 - uploadDate
- Relationships

Identity and Type

Name Model.xcdatamodel

Type Default - Core Data Model

Location Relative to Group
Model.xcdatamodel

Full Path /Users/CS193p/Developer/CoreDataExample/CoreDataExample/Model.xcdatamodeld/Model.xcdatamodel

Core Data Model

Identifier Model Version Identifier

Tools Version

Minimum Xcode 4.3

Deployment Targets

Mac OS X Target Default

iOS Target Default

Target Membership

CoreDataExample

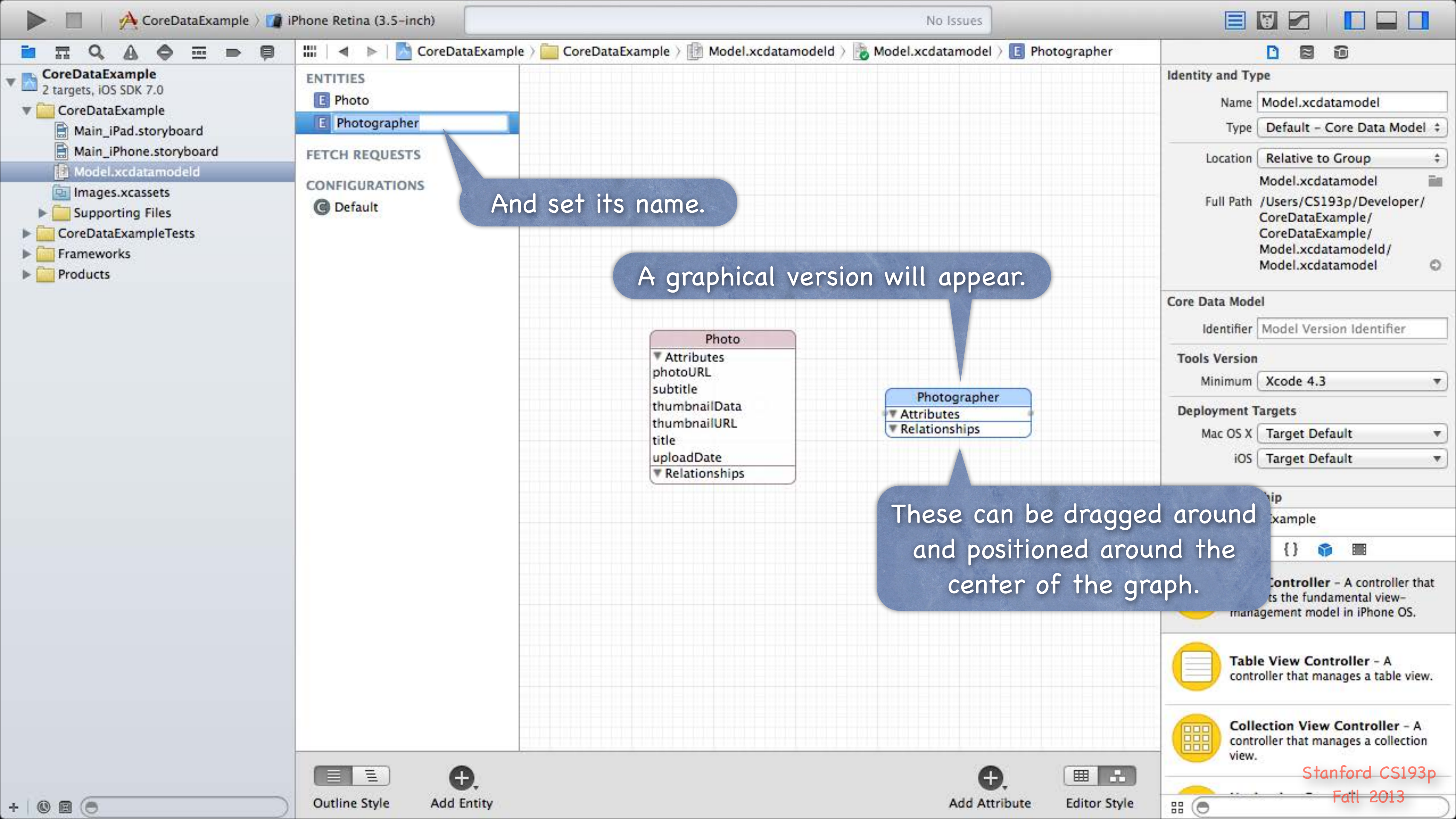
View Controller - A controller that supports the fundamental view-management model in iPhone OS.

Table View Controller - A controller that manages a table view.

Collection View Controller - A controller that manages a collection view.

Let's add another Entity.

- Add Entity
- Add Fetch Request
- Add Configuration



And set its name.

A graphical version will appear.

These can be dragged around and positioned around the center of the graph.

CoreDataExample
2 targets, iOS SDK 7.0

- CoreDataExample
 - Main_iPad.storyboard
 - Main_iPhone.storyboard
 - Model.xcdatamodeld
 - Images.xcassets
 - Supporting Files
 - CoreDataExampleTests
 - Frameworks
 - Products

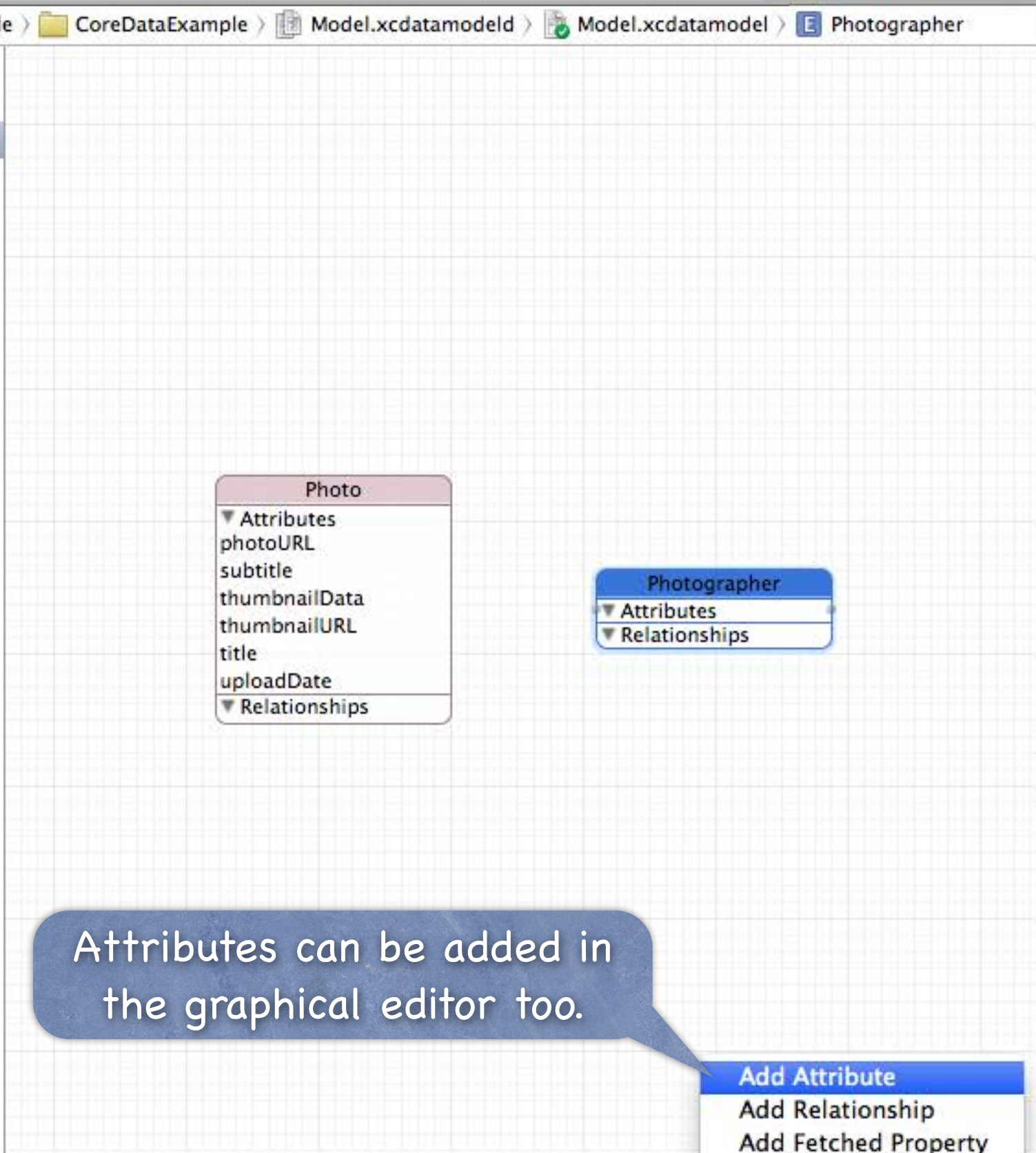
ENTITIES

- Photo
- Photographer

FETCH REQUESTS

CONFIGURATIONS

- Default



Identity and Type

Name: Model.xcdatamodel

Type: Default - Core Data Model

Location: Relative to Group

Model.xcdatamodel

Full Path: /Users/CS193p/Developer/CoreDataExample/CoreDataExample/Model.xcdatamodeld/Model.xcdatamodel

Core Data Model

Identifier: Model Version Identifier

Tools Version

Minimum: Xcode 4.3

Deployment Targets

Mac OS X: Target Default

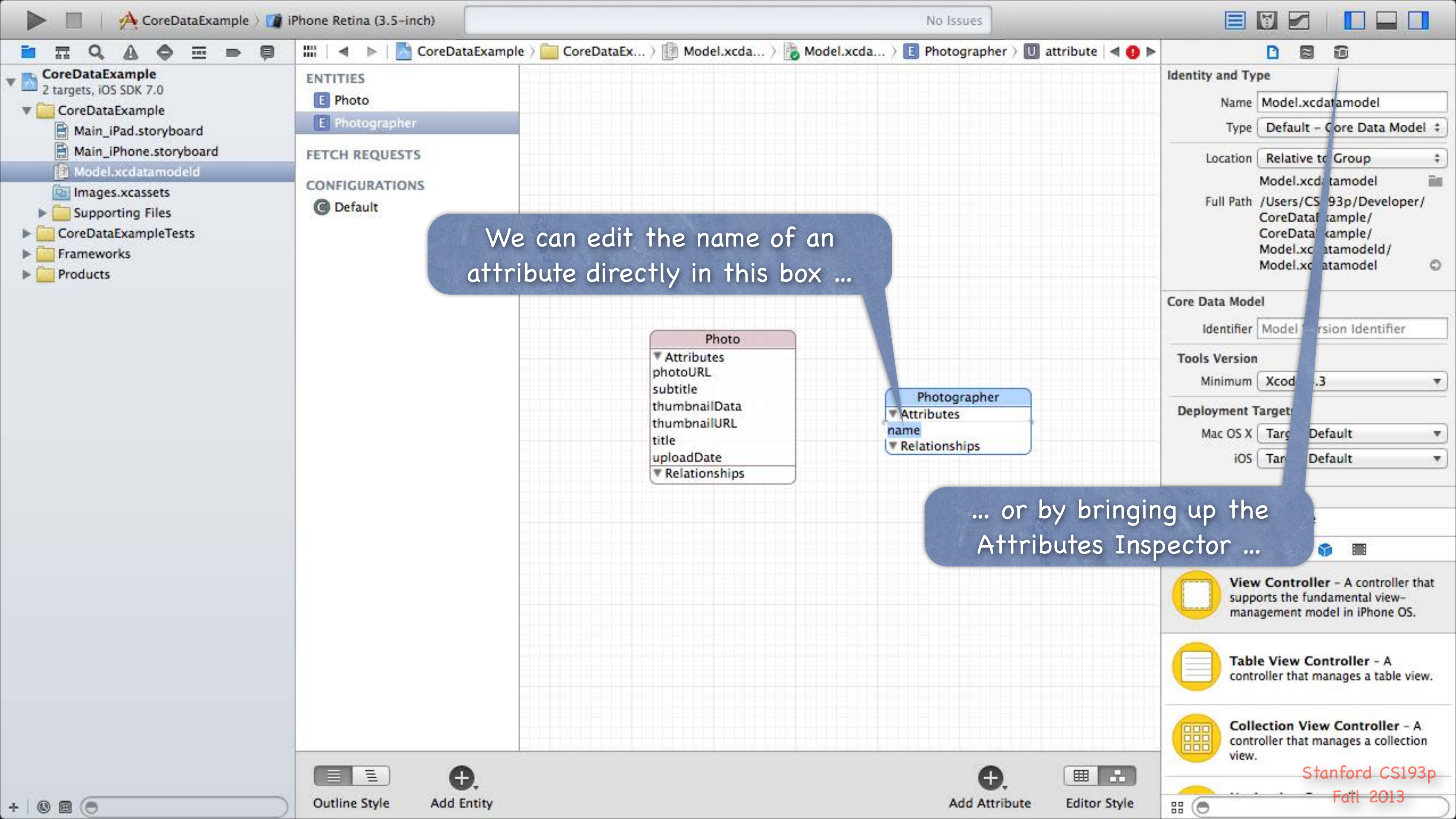
iOS: Target Default

Target Membership

CoreDataExample

Attributes can be added in the graphical editor too.

- Add Attribute
- Add Relationship
- Add Fetched Property



We can edit the name of an attribute directly in this box ...

... or by bringing up the Attributes Inspector ...

CoreDataExample
2 targets, iOS SDK 7.0

- CoreDataExample
 - Main_iPad.storyboard
 - Main_iPhone.storyboard
 - Model.xcdatamodeld
 - Images.xcassets
 - Supporting Files
 - CoreDataExampleTests
 - Frameworks
 - Products

ENTITIES

- Photo
- Photographer

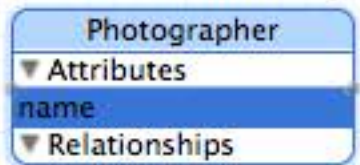
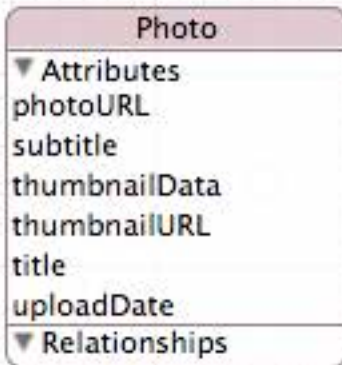
FETCH REQUESTS

CONFIGURATIONS

- Default

There are a number of advanced features you can set on an Attribute ...

... but we're just going to set its type.



Attribute

Name: name

Properties: Transient Optional Indexed

Attribute Type: Undefined Integer 16 Integer 32 Integer 64 Decimal Double Float **String** Boolean Date Binary Data Transformable

Versioning

Hash Modifier: Version Hash Modifier

Renaming ID: Renaming Identifier

View Controller - A controller that supports the fundamental view-management model in iPhone OS.

Table View Controller - A controller that manages a table view.

Collection View Controller - A controller that manages a collection view.

CoreDataExample
2 targets, iOS SDK 7.0

- CoreDataExample
 - Main_iPad.storyboard
 - Main_iPhone.storyboard
 - Model.xcdatamodeld
 - Images.xcassets
 - Supporting Files
 - CoreDataExampleTests
 - Frameworks
 - Products

ENTITIES

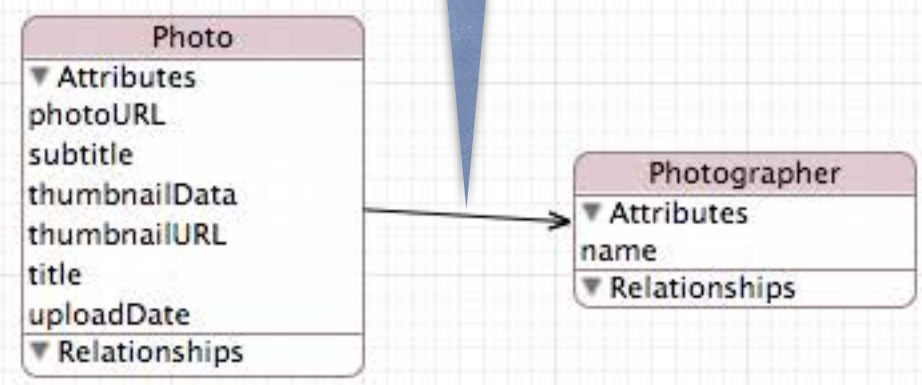
- Photo
- Photographer

FETCH REQUESTS

CONFIGURATIONS

- Default

Similar to outlets and actions, we can ctrl-drag to create Relationships between Entities.



Entity

Name: Photographer

Class: NSObject

Abstract Entity

Parent Entity: No Parent Entity

Indexes

+ -

User Info

Key	Value
-----	-------

+ -

Versioning

Hash Modifier: Version Hash Modifier

Renaming ID: Renaming Identifier

+ -

- View Controller** - A controller that supports the fundamental view-management model in iPhone OS.
- Table View Controller** - A controller that manages a table view.
- Collection View Controller** - A controller that manages a collection view.

CoreDataExample
2 targets, iOS SDK 7.0

- CoreDataExample
 - Main_iPad.storyboard
 - Main_iPhone.storyboard
 - Model.xcdatamodeld
 - Images.xcassets
 - Supporting Files
 - CoreDataExampleTests
 - Frameworks
 - Products

ENTITIES

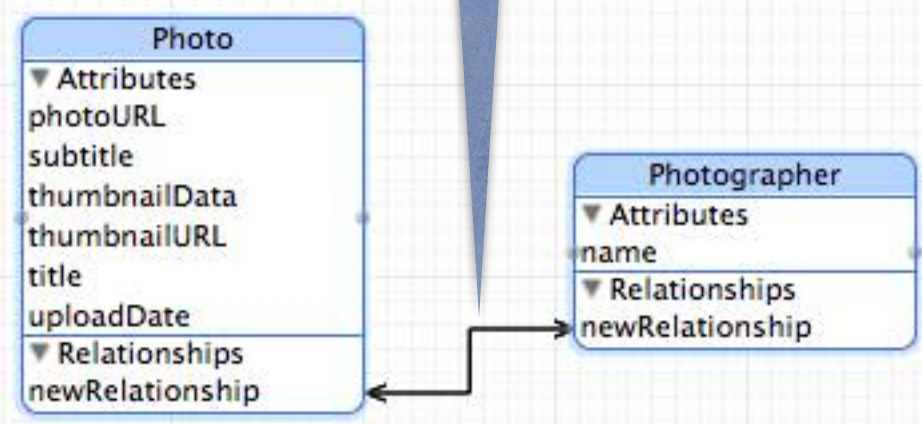
- Photo
- Photographer

FETCH REQUESTS

CONFIGURATIONS

- Default

A Relationship is analogous to a pointer to another object" (or NSSet of other objects).



Entity

Name: Multiple Values

Class: NSObject

Abstract Entity

Parent Entity: No Parent Entity

Indexes

+ -

User Info

Key	Value
-----	-------

+ -

Versioning

Hash Modifier: Version Hash Modifier

Renaming ID: Renaming Identifier

+ -

- View Controller** - A controller that supports the fundamental view-management model in iPhone OS.
- Table View Controller** - A controller that manages a table view.
- Collection View Controller** - A controller that manages a collection view.

CoreDataExample
2 targets, iOS SDK 7.0

- CoreDataExample
 - Main_iPad.storyboard
 - Main_iPhone.storyboard
 - Model.xcdatamodeld
 - Images.xcassets
 - Supporting Files
 - CoreDataExampleTests
 - Frameworks
 - Products

ENTITIES

- Photo
- Photographer

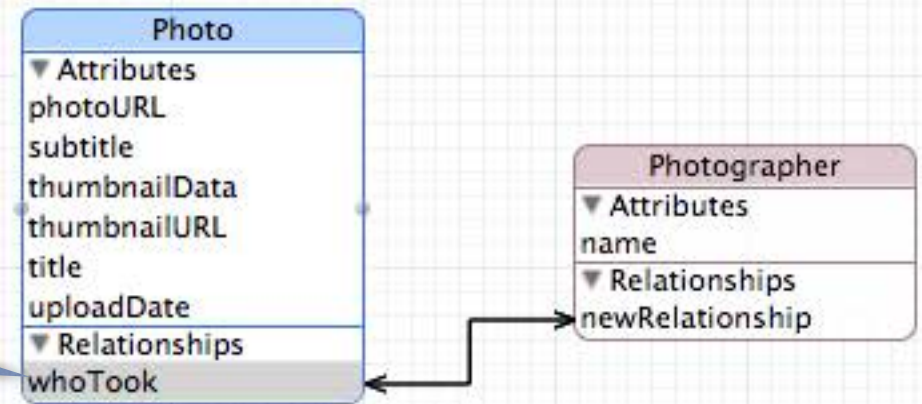
FETCH REQUESTS

CONFIGURATIONS

- Default

From a Photo's perspective, this Relationship to a Photographer is "who took" the Photo ...

... so we'll call the Relationship "whoTook" on the Photo side.



Relationship

Name: whoTook

Properties: Transient Optional

Destination: Photographer

Inverse: newRelationship

Delete Rule: Nullify

Type: To One

Advanced: Index in Spotlight Store in External Record File

User Info

Key	Value

Versioning

Hash Modifier: Version Hash Modifier

Renaming ID: Renaming Identifier

View Controller - A controller that supports the fundamental view-management model in iPhone OS.

Table View Controller - A controller that manages a table view.

Collection View Controller - A controller that manages a collection view.

CoreDataExample
2 targets, iOS SDK 7.0

- CoreDataExample
 - Main_iPad.storyboard
 - Main_iPhone.storyboard
 - Model.xcdatamodeld
 - Images.xcassets
 - Supporting Files
 - CoreDataExampleTests
 - Frameworks
 - Products

ENTITIES

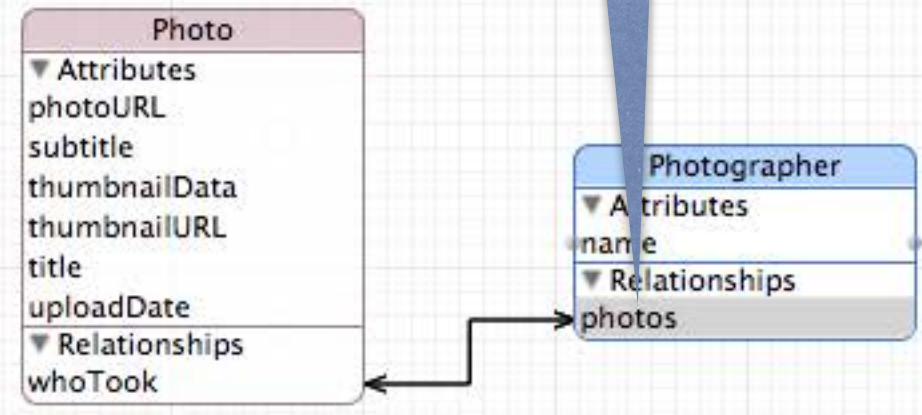
- Photo
- Photographer

FETCH REQUESTS

CONFIGURATIONS

- Default

A Photographer can take many Photos, so we'll call this Relationship "photos" on the Photographer side.



Relationship

Name: photos

Properties: Transient Optional

Destination: Photo

Inverse: whoTook

Delete Rule: Nullify

Type: To One

Advanced: Index in Spotlight Store in External Record File

User Info

Key	Value
-----	-------

Versioning

Hash Modifier: Version Hash Modifier

See how Xcode notes the inverse relationship between photos and whoTook.

- View Controller** - A controller that supports the fundamental view-management model in iPhone OS.
- Table View Controller** - A controller that manages a table view.
- Collection View Controller** - A controller that manages a collection view.

CoreDataExample
2 targets, iOS SDK 7.0

- CoreDataExample
 - Main_iPad.storyboard
 - Main_iPhone.storyboard
 - Model.xcdatamodeld
 - Photo+Flickr.h
 - Photo+Flickr.m
 - Photo.h
 - Photo.m
 - Photographer.h
 - Photographer.m
 - Images.xcassets
 - Supporting Files
 - CoreDataExampleTests
 - Frameworks
 - Products

ENTITIES

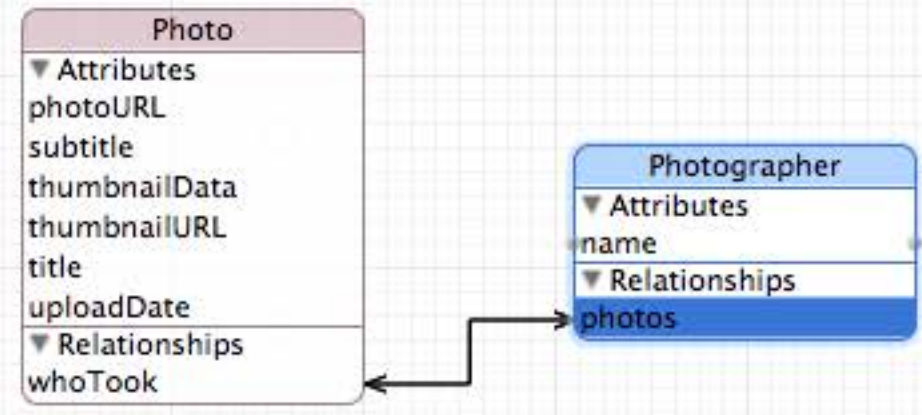
- Photo
- Photographer

FETCH REQUESTS

CONFIGURATIONS

- Default

We also need to note that there can be many Photos per Photographer.



Relationship

Name: photos

Properties: Transient Optional

Destination: Photo

Inverse: whoTook

Delete Rule: Delete Self Delete Other

Type: To One To Many

Advanced: Index in Spotlight Store in External Record File

User Info

Key	Value

Versioning

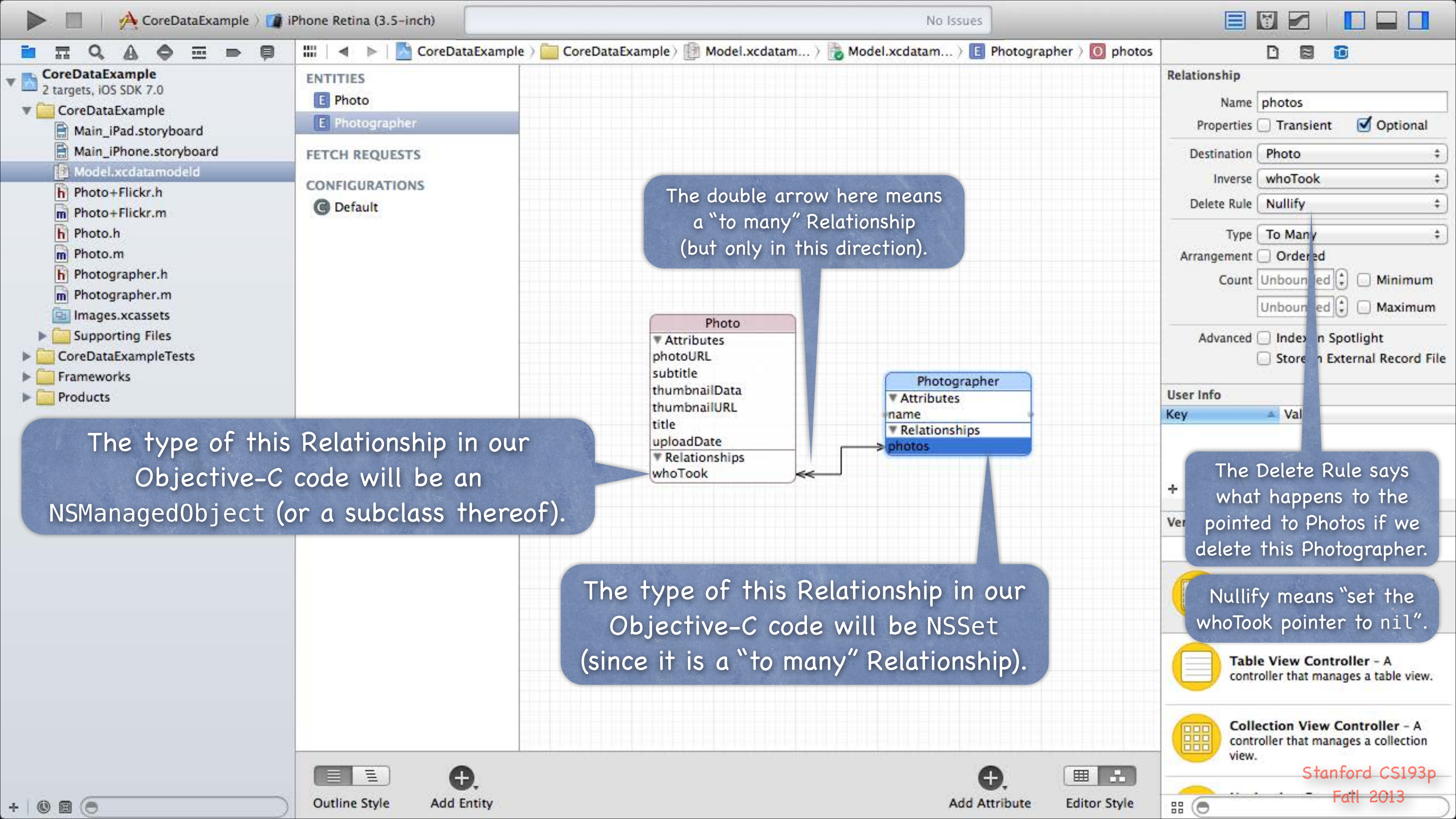
Hash Modifier: Version Hash Modifier

Renaming ID: Renaming Identifier

View Controller - A controller that supports the fundamental view-management model in iPhone OS.

Table View Controller - A controller that manages a table view.

Collection View Controller - A controller that manages a collection view.



The double arrow here means a "to many" Relationship (but only in this direction).

The type of this Relationship in our Objective-C code will be an NSObject (or a subclass thereof).

The type of this Relationship in our Objective-C code will be NSSet (since it is a "to many" Relationship).

The Delete Rule says what happens to the pointed to Photos if we delete this Photographer. Nullify means "set the whoTook pointer to nil".

Nullify means "set the whoTook pointer to nil".

Table View Controller - A controller that manages a table view.

Collection View Controller - A controller that manages a collection view.

Core Data

- There are lots of other things you can do

But we are going to focus on creating Entities, Attributes and Relationships.

- So how do you access all of this stuff in your code?

You need an `NSManagedObjectContext`.

It is the hub around which all Core Data activity turns.

- How do I get one?

There are two ways ...

1. Create a `UIManagedDocument` and ask for its `managedObjectContext` (a @property).
2. Click the "Use Core Data" button when you create a project (only works with certain templates) (then your AppDelegate will have a `managedObjectContext` @property).

If you study what the template (e.g. Master-Detail template) does, you'll get an idea how it works.

We're going to focus on doing the first one.

UIManagedDocument

UIManagedDocument

It inherits from UIDocument which provides a lot of mechanism for the management of storage. If you use UIManagedDocument, you'll be on the fast-track to iCloud support. Think of a UIManagedDocument as simply a container for your Core Data database.

Creating a UIManagedDocument

```
NSFileManager *fileManager = [NSFileManager defaultManager];
NSURL *documentsDirectory = [[fileManager URLsForDirectory:NSDocumentDirectory
                                                             inDomains:NSUserDomainMask] firstObject];
NSString *documentName = @"MyDocument";
NSURL *url = [documentsDirectory URLByAppendingPathComponent:documentName];
UIManagedDocument *document = [[UIManagedDocument alloc] initWithFileURL:url];
```

This creates the UIManagedDocument instance, but does not open nor create the underlying file.

UIManagedDocument

How to open or create a UIManagedDocument

First, check to see if the UIManagedDocument's underlying file exists on disk ...

```
BOOL fileExists = [[NSFileManager defaultManager] fileExistsAtPath:[url path]];
```

... if it does, open the document using ...

```
[document openWithCompletionHandler:^(BOOL success) { /* block to execute when open */ }];
```

... if it does not, create the document using ...

```
[document saveToURL:url // could (should?) use document.fileURL property here  
forSaveOperation:UIDocumentSaveForCreating  
completionHandler:^(BOOL success) { /* block to execute when create is done */ }];
```

What is that `completionHandler`?

Just a block of code to execute when the open/save completes.

That's needed because the open/save is asynchronous (i.e. happens on its own queue).

Do not ignore this fact!

UIManagedDocument

Example

```
self.document = [[UIManagedDocument alloc] initWithFileURL:(URL *)url];
if ([[NSFileManager defaultManager] fileExistsAtPath:[url path]]) {
    [document openWithCompletionHandler:^(BOOL success) {
        if (success) [self documentIsReady];
        if (!success) NSLog(@"couldn't open document at %@", url);
    }];
} else {
    [document saveToURL:url forSaveOperation:UIDocumentSaveForCreating
    completionHandler:^(BOOL success) {
        if (success) [self documentIsReady];
        if (!success) NSLog(@"couldn't create document at %@", url);
    }];
}
// can't do anything with the document yet (do it in documentIsReady).
```

UIManagedDocument

- Once document is open/created, you can start using it

But you might want to check the `documentState` when you do ...

```
- (void)documentIsReady
{
    if (self.document.documentState == UIDocumentStateNormal) {
        // start using document
    }
}
```

- Other documentStates

`UIDocumentStateClosed` (you haven't done the open or create yet)

`UIDocumentStateSavingError` (success will be NO in completion handler)

`UIDocumentStateEditingDisabled` (temporary situation, try again)

`UIDocumentStateInConflict` (e.g., because some other device changed it via iCloud)

We don't have time to address these (you can ignore in homework), but know that they exist.

UIManagedDocument

- Okay, document is ready to use, now what?

Now you can get a managedObjectContext from it and use it to do Core Data stuff!

```
- (void)documentIsReady
{
    if (self.document.documentState == UIDocumentStateNormal) {
        NSManagedObjectContext *context = self.document.managedObjectContext;
        // start doing Core Data stuff with context
    }
}
```

Okay, just a couple of more UIManagedDocument things before we start using that context ...

UIManagedDocument

Saving the document

UIManagedDocuments **AUTOSAVE** themselves!

However, if, for some reason you wanted to manually save (asynchronous!) ...

```
[document saveToURL:document.fileURL
    forSaveOperation:UIDocumentSaveForOverwriting
    completionHandler:^(BOOL success) { /* block to execute when save is done */ }];
```

Note that this is almost identical to creation (just **UIDocumentSaveForOverwriting** is different).

This is a UIKit class and so this method must be called on the main queue.

Closing the document

Will automatically close if there are no **strong** pointers left to it.

But you can explicitly close with (asynchronous!) ...

```
[self.document closeWithCompletionHandler:^(BOOL success) {
    if (!success) NSLog(@"failed to close document %@", self.document.localizedName);
}];
```

UIManagedDocument's `localizedName` method ...

```
@property (strong) NSString *localizedName; // suitable for UI (but only valid once saved)
```

UIManagedDocument

Multiple instances of UIManagedDocument on the same document

This is perfectly legal, but understand that they will not share an NSManagedObjectContext. Thus, changes in one will not automatically be reflected in the other.

You'll have to refetch in other UIManagedDocuments after you make a change in one.

Conflicting changes in two different UIManagedDocuments would have to be resolved by you! It's exceedingly rare to have two "writing" instances of UIManagedDocument on the same file. But a single writer and multiple readers? Less rare. But you need to know when to refetch.

You can watch (via "radio station") other documents' managedObjectContexts (then refetch). Or you can use a single UIManagedDocument instance (per actually document) throughout.

NSNotification

• How would you watch a document's managedObjectContext?

```
- (void)viewDidAppear:(BOOL)animated
{
    [super viewDidAppear:animated];
    [center addObserver:self
                 selector:@selector(contextChanged:)
                 name:NSManagedObjectContextDidSaveNotification
                 object:document.managedObjectContext]; // don't pass nil here!
}
- (void)viewWillDisappear:(BOOL)animated
{
    [center removeObserver:self
                     name:NSManagedObjectContextDidSaveNotification
                     object:document.managedObjectContext];
    [super viewWillDisappear:animated];
}
```

NSNotification

◉ NSManagedObjectContextDidSaveNotification

```
- (void)contextChanged:(NSNotification *)notification
{
    // The notification.userInfo object is an NSDictionary with the following keys:
    NSInsertedObjectsKey // an array of objects which were inserted
    NSUpdatedObjectsKey  // an array of objects whose attributes changed
    NSDeletedObjectsKey  // an array of objects which were deleted
}
```

◉ Merging changes

If you get notified that another NSManagedObjectContext has changed your database ...
... you can just refetch (if you haven't changed anything in your NSMOC, for example).
... or you can use the NSManagedObjectContext method:

```
- (void)mergeChangesFromContextDidSaveNotification:(NSNotification *)notification;
```

Core Data

• Okay, we have an `NSManagedObjectContext`, now what?

We grabbed it from an open `UIManagedDocument`'s `managedObjectContext` @property.

Now we use it to insert/delete objects in the database and query for objects in the database.

Core Data

◉ Inserting objects into the database

```
NSManagedObjectContext *context = aDocument.managedObjectContext;  
NSManagedObject *photo =  
    [NSEntityDescription insertNewObjectForEntityForName:@"Photo"  
        inManagedObjectContext:context];
```

Note that this `NSEntityDescription` class method returns an `NSManagedObject` instance. All objects in the database are represented by `NSManagedObjects` or subclasses thereof.

An instance of `NSManagedObject` is a manifestation of an Entity in our Core Data Model*. Attributes of a newly-inserted object will start out `nil` (unless you specify a default in Xcode).

* i.e., the Data Model that we just graphically built in Xcode!

Core Data

• How to access Attributes in an NSManagedObject instance

You can access them using the following two `NSKeyValueCoding` protocol methods ...

- `(id)valueForKey:(NSString *)key;`
- `(void)setValue:(id)value forKey:(NSString *)key;`

You can also use `valueForKeyPath:/setValue:forKeyPath:` and it will follow your Relationships!

• The `key` is an Attribute name in your data mapping

For example, `@“thumbnailURL”` or `@“title”`.

• The `value` is whatever is stored (or to be stored) in the database

It'll be `nil` if nothing has been stored yet (unless Attribute has a default value in Xcode).

Note that all values are objects (numbers and booleans are `NSNumber` objects).

Binary data values are `NSData` objects.

Date values are `NSDate` objects.

“To-many” mapped relationships are `NSSet` objects (or `NSOrderedSet` if ordered).

Non-“to-many” relationships are `NSManagedObjects`.

Core Data

- Changes (writes) only happen in memory, until you save

Remember, `UIManagedDocument` autosaves.

When the document is saved, the context is saved and your changes get written to the database.

`UIManagedDocumentDidSaveNotification` will be “broadcast” at that point.

Be careful during development where you press “Stop” in Xcode (sometimes autosave is pending).

Core Data

- But calling `valueForKey:/setValue:forKey:` is pretty ugly

There's no type-checking.

And you have a lot of literal strings in your code (e.g. @"thumbnailURL")

- What we really want is to set/get using `@property`s!

- No problem ... we just create a subclass of `NSManagedObject`

The subclass will have `@property`s for each attribute in the database.

We name our subclass the same name as the Entity it matches (not strictly required, but do it).

And, as you might imagine, we can get Xcode to generate both the header file `@property` entries, and the corresponding implementation code (which is not `@synthesize`, so watch out!).

CoreDataExample
2 targets, iOS SDK 7.0

- CoreDataExample
 - Main_iPad.storyboard
 - Main_iPhone.storyboard
 - Model.xcdatamodeld
 - Images.xcassets
 - Supporting Files
 - CoreDataExampleTests
 - Frameworks
 - Products

ENTITIES

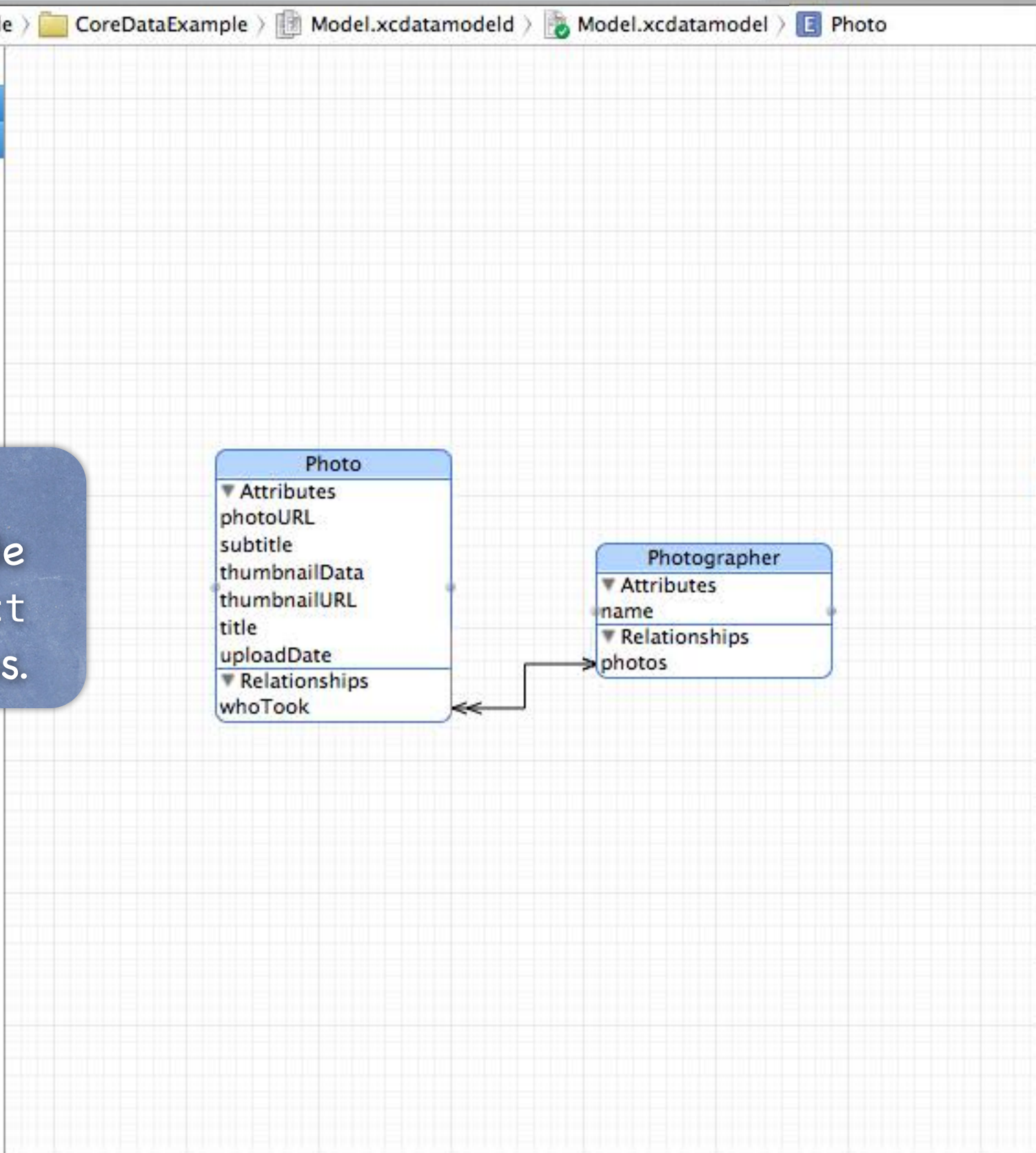
- E Photo
- E Photographer

FETCH REQUESTS

CONFIGURATIONS

- Default

Select both Entities.
We're going to have Xcode generate NSObject subclasses for them for us.



Entity

Name: Multiple Values

Class: NSObject

Abstract Entity

Parent Entity: No Parent Entity

Indexes

+ -

User Info

Key	Value
-----	-------

+ -

Versioning

Hash Modifier: Version Hash Modifier

Renaming ID: Renaming Identifier

- View Controller** - A controller that supports the fundamental view-management model in iPhone OS.
- Table View Controller** - A controller that manages a table view.
- Collection View Controller** - A controller that manages a collection view.

CoreDataExample > iPhone Retina (3.5-in

CoreDataExample
2 targets, iOS SDK 7.0

- CoreDataExample
 - Main_iPad.storyboard
 - Main_iPhone.storyboard
 - Model.xcdatamodeld
 - Images.xcassets
 - Supporting Files
 - CoreDataExampleTests
 - Frameworks
 - Products

ENTITIES

- Photo
- Photographer

FETCH REQUESTS

CONFIGURATION

- Default

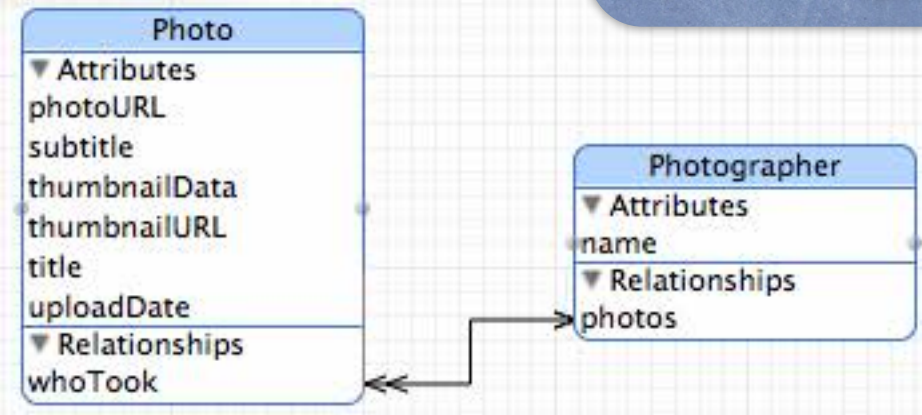
Canvas

- Add Entity
- Add Fetch Request
- Add Configuration

- Add Attribute
- Add Fetched Property
- Add Relationship

- Create NSManagedObject Subclass...
- Add Model Version...
- Import...

Ask Xcode to generate NSManagedObject subclasses for our Entities.



Entity

Name: Multiple Values

Class: NSManagedObject

Abstract Entity

Entity: No Parent Entity

Indexes

+ -

Key Value

+ -

Versioning

Hash Modifier: Version Hash Modifier

Renaming ID: Renaming Identifier

View Controller - A controller that supports the fundamental view-management model in iPhone OS.

Table View Controller - A controller that manages a table view.

Collection View Controller - A controller that manages a collection view.

CoreDataExample
2 targets, iOS SDK 7.0

- CoreDataExample
 - Main_iPad.storyboard
 - Main_iPhone.storyboard
 - Model.xcdatamodeld
 - Images.xcassets
 - Supporting Files
 - CoreDataExampleTests
 - Frameworks
 - Products

Select the data models with entities you would like to manage

Select	Data Model
<input checked="" type="checkbox"/>	Model

Cancel Previous Next

Which Data Models to generate subclasses for (we only have one Data Model).

Entity

Name: Multiple Values
Class: NSObject
 Abstract Entity
Parent Entity: No Parent Entity
Indexes: + -

User Info

Key	Value
-----	-------

+ -

Versioning

Hash Modifier: Version Hash Modifier
Renaming ID: Renaming Identifier

View Controller - A controller that supports the fundamental view-management model in iPhone OS.

Table View Controller - A controller that manages a table view.

Collection View Controller - A controller that manages a collection view.

CoreDataExample
2 targets, iOS SDK 7.0

- CoreDataExample
 - Main_iPad.storyboard
 - Main_iPhone.storyboard
 - Model.xcdatamodeld
 - Images.xcassets
 - Supporting Files
 - CoreDataExampleTests
 - Frameworks
 - Products

Select the entities you would like to manage

Select	Entity
<input checked="" type="checkbox"/>	Photo
<input checked="" type="checkbox"/>	Photographer

Cancel Previous Next

Which Entities to generate subclasses for (usually we choose all of them).

Entity

Name: Multiple Values
Class: NSObject
 Abstract Entity
Parent Entity: No Parent Entity
Indexes: + -

User Info

Key	Value
-----	-------

+ -

Versioning

Hash Modifier: Version Hash Modifier
Renaming ID: Renaming Identifier

View Controller - A controller that supports the fundamental view-management model in iPhone OS.

Table View Controller - A controller that manages a table view.

Collection View Controller - A controller that manages a collection view.

CoreDataExample
2 targets, iOS SDK 7.0

- CoreDataExample
 - Main_iPad.storyboard
 - Main_iPhone.storyboard
 - Model.xcdatamodeld
 - Images.xcassets
 - Supporting Files
 - CoreDataExampleTests
 - Frameworks
 - Products

CoreDataExample

FAVORITES

- All My Files
- CS193p
- Documents
- Applications
- Downloads
- Stanford

CoreDataExample

- CoreDataExample
- Example
- Matchismo

CoreDataExample

- CoreDataExample.xcodeproj
- CoreDataExampleTests

AppDelegate.h

AppDelegate.m

Base.lproj

CoreDataExample-Info.plist

CoreDataExample-Prefix.pch

en.lproj

Images.xcassets

main.m

Model.xcdatamodeld

Name: Multiple Values

Class: NSObject

Abstract Entity

Parent Entity: No Parent Entity

Indexes

+ -

Pick which group you want your new classes to be stored (default is often one directory level higher, so watch out).

This will make your @properties be scalars (e.g. int instead of NSNumber *) where possible. Be careful if one of your Attributes is an NSDate, you'll end up with an NSTimeInterval @property.

Options Use scalar properties for primitive data types

Group

Targets

- CoreDataExample
- CoreDataExampleTests

New Folder

Cancel Create

Modifier: Version Hash Modifier

Renaming ID: Renaming Identifier

View Controller - A controller that supports the fundamental view-management model in iPhone OS.

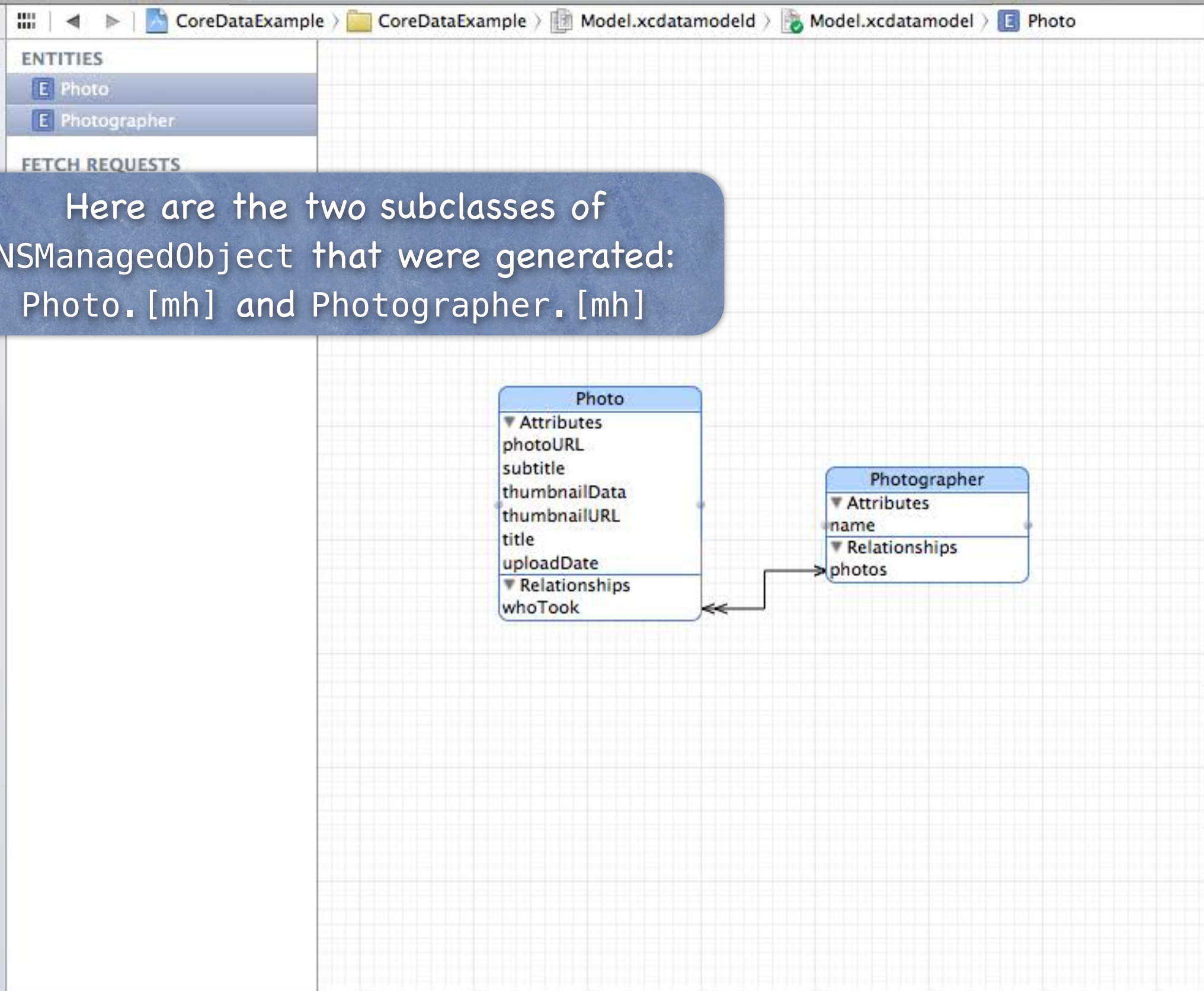
Table View Controller - A controller that manages a table view.

Collection View Controller - A controller that manages a collection view.

CoreDataExample
2 targets, iOS SDK 7.0

- CoreDataExample
 - Main_iPad.storyboard
 - Main_iPhone.storyboard
 - Model.xcdatamodeld
 - Photo.h
 - Photo.m
 - Photographer.h
 - Photographer.m
 - Images.xcassets
- Supporting Files
- CoreDataExampleTests
- Frameworks
- Products

Here are the two subclasses of NSManagedObject that were generated: Photo.[mh] and Photographer.[mh]



No Selection

- View Controller** - A controller that supports the fundamental view-management model in iPhone OS.
- Table View Controller** - A controller that manages a table view.
- Collection View Controller** - A controller that manages a collection view.

CoreDataExample
2 targets, iOS SDK 7.0

- CoreDataExample
 - Main_iPad.storyboard
 - Main_iPhone.storyboard
 - Model.xcdatamodeld
 - Photo.h
 - Photo.m
 - Photographer.h
 - Photographer.m
 - Images.xcassets
 - Supporting Files
 - CoreDataExampleTests
 - Frameworks
 - Products

```
//
// Photo.h
// CoreDataExample
//
// Created by CS193p Instructor.
// Copyright (c) 2013 Stanford University. All rights reserved.
//

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

@class Photographer;

@interface Photo : NSObject

@property (nonatomic, retain) NSString * title;
@property (nonatomic, retain) NSString * photoURL;
@property (nonatomic, retain) NSString * thumbnailURL;
@property (nonatomic, retain) NSString * subtitle;
@property (nonatomic, retain) NSDate * uploadDate;
@property (nonatomic, retain) NSData * thumbnailData;
@property (nonatomic, retain) Photographer *whoTook;

@end
```

@properties generated for all of our Attributes!
Now we can use dot notation to access these in code.

Depending on the order Xcode generated Photo and Photographer, it might not have gotten whoTook's type (Photographer *) right (it might say NSObject *). If that happens, just generate again.

Quick Help

No Quick Help

View Controller - A controller that supports the fundamental view-management model in iPhone OS.

Table View Controller - A controller that manages a table view.

Collection View Controller - A controller that manages a collection view.

Stanford CS193p
Fall 2013

CoreDataExample
2 targets, iOS SDK 7.0

- CoreDataExample
 - Main_iPad.storyboard
 - Main_iPhone.storyboard
 - Model.xcdatamodeld
 - Photo.h
 - Photo.m
 - Photographer.h**
 - Photographer.m
 - Images.xcassets
 - Supporting Files
 - CoreDataExampleTests
 - Frameworks
 - Products

```
//
// Photographer.h
// CoreDataExample
//
// Created by CS193p Instructor.
// Copyright (c) 2013 Stanford University. All rights reserved.
//

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

@class Photo;

@interface Photographer : NSObject

@property (nonatomic, retain) NSString * name;
@property (nonatomic, retain) NSSet * photos;
@end

@interface Photographer (CoreDataGeneratedAccessors)

- (void)addPhotosObject:(Photo *)value;
- (void)removePhotosObject:(Photo *)value;
- (void)addPhotos:(NSSet *)values;
- (void)removePhotos:(NSSet *)values;

@end
```

Inherits from NSObject.

Photographer also got some convenient methods for adding/removing photos that this Photographer has taken.

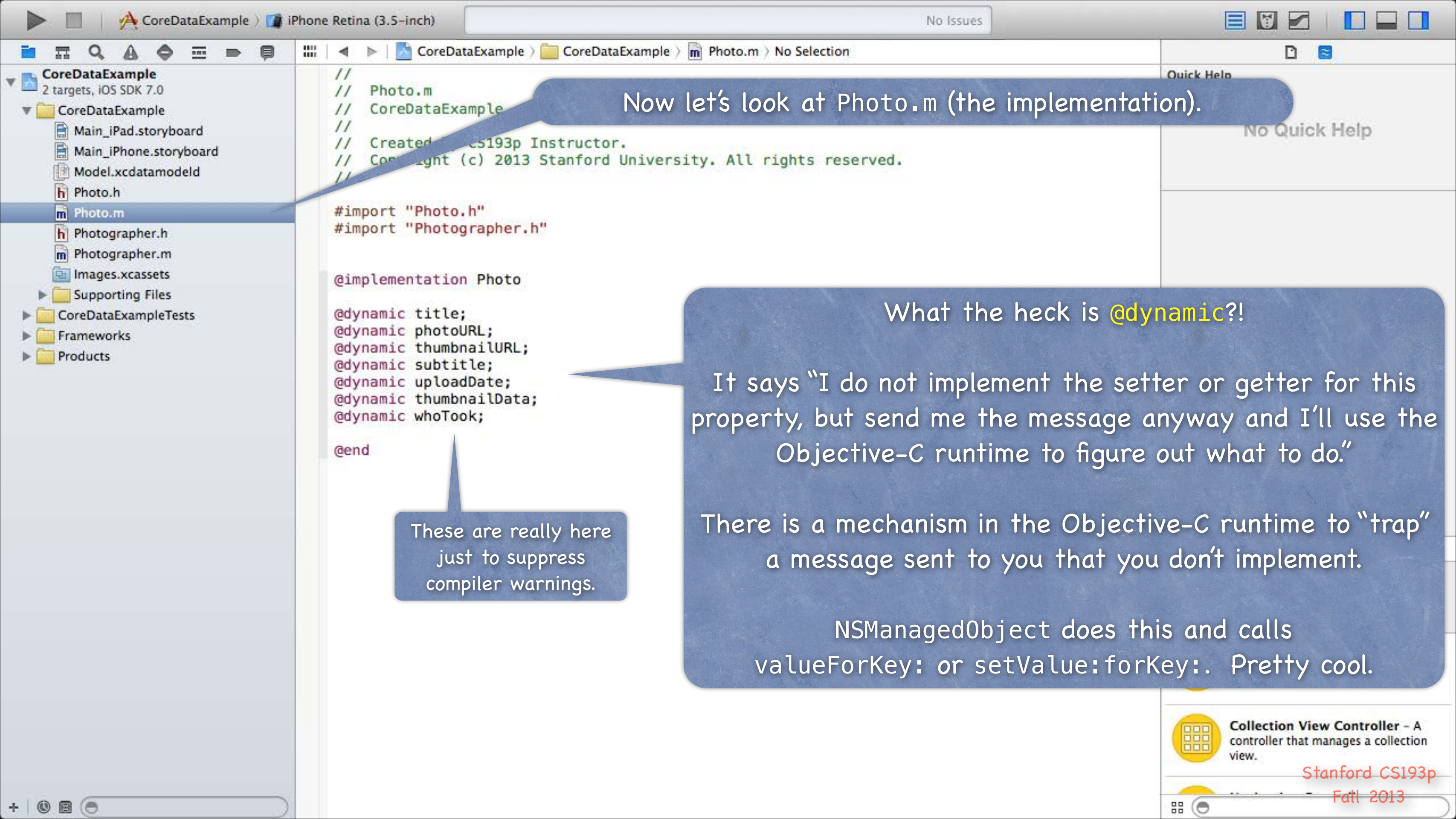
Quick Help

No Quick Help

View Controller - A controller that supports the fundamental view-management model in iPhone OS.

Table View Controller - A controller that manages a table view.

Collection View Controller - A controller that manages a collection view.



Now let's look at Photo.m (the implementation).


What the heck is @dynamic?!

It says "I do not implement the setter or getter for this property, but send me the message anyway and I'll use the Objective-C runtime to figure out what to do."

There is a mechanism in the Objective-C runtime to "trap" a message sent to you that you don't implement.

NSObject does this and calls valueForKey: or setValue:forKey:. Pretty cool.

These are really here just to suppress compiler warnings.

 **Collection View Controller** - A controller that manages a collection view.

Core Data

• So how do I access my Entities' Attributes with dot notation?

// let's create an instance of the Photo Entity in the database ...

```
NSManagedObjectContext *context = document.managedObjectContext;
```

```
Photo *photo = [NSEntityDescription insertNewObjectForEntityForName:@"Photo"  
                inManagedObjectContext:context];
```

// then set the attributes in our Photo using, say, an NSDictionary we got from Flickr ...

```
e.g. photo.title = [flickrData objectForKey:FLICKR_PHOTO_TITLE];
```

// the information will automatically be saved (i.e. autosaved) into our document by Core Data

// now here's some other things we could do too ...

```
NSString *myThumbnail = photo.thumbnailURL;
```

```
photo.lastViewedDate = [NSDate date];
```

```
photo.whoTook = ...; // a Photographer object we created or got by querying
```

```
photo.whoTook.name = @"CS193p Instructor"; // yes, multiple dots will follow relationships!
```

Core Data

• What if I want to add code to my `NSManagedObject` subclass?

For example, we might want to add a method or two (to the `@propertys` added by Xcode).

It would be especially nice to add class methods to create and set up an object in the database (e.g. set all the properties of a `Photo` or `Photographer` using an `NSDictionary` from Flickr).

Or maybe to derive new `@propertys` based on ones in the database (e.g. a `UIImage` based on a `URL` in the database).

But that could be a problem if we edited `Photo.m` or `Photographer.m` ...

Because you might want to modify your schema and re-generate those `.h` and `.m` files from Xcode!

To get around this, we need to use an Objective-C feature called “categories”.

So let's take a moment to learn about that ...

Categories

- Categories are an Objective-C syntax for adding to a class ...

Without subclassing it.

Without even having to have access to the code of the class (e.g. you don't need its .m).

- Examples

NSAttributedString's drawAtPoint: method.

- Added by UIKit (since it's a UI method) even though NSAttributedString is in Foundation.

NSIndexPath's row and section properties (used in UITableView-related code).

- Added by UIKit too, even though NSIndexPath is also in Foundation.

- Syntax

```
@interface Photo (AddOn)
```

```
- (UIImage *)image;
```

```
@property (readonly) BOOL isOld;
```

```
@end
```

Categories have their own .h and .m files (usually ClassName+PurposeOfExtension.[mh]).

Categories cannot have instance variables!

Categories

Implementation

```
@implementation Photo (AddOn)
- (UIImage *)image // image is not an attribute in the database, but photoURL is
{
    NSURL *imageURL = [NSURL URLWithString:self.photoURL];
    NSData *imageData = [NSData dataWithContentsOfURL:imageURL];
    return [UIImage imageData:imageData];
}
- (BOOL)isOld // whether this Photo was uploaded more than a day ago
{
    return [self.uploadDate TimeIntervalSinceNow] > -24*60*60;
}
@end
```

Other examples ... sometimes we add @propertys to an NSObject subclass via categories to make accessing BOOL attributes (which are NSNumber) more cleanly.

Or we add @propertys to convert NSDatas to whatever the bits represent.

Any class can have a category added to it, but don't overuse/abuse this mechanism.

Categories

- Most common category on an NSManagedObject subclass?

Creation ...

```
@implementation Photo (Create)
```

```
+ (Photo *)photoWithFlickrData:(NSDictionary *)flickrData  
    inManagedObjectContext:(NSManagedObjectContext *)context
```

```
{
```

```
    Photo *photo = ...; // see if a Photo for that Flickr data is already in the database
```

```
    if (!photo) {
```

```
        photo = [NSEntityDescription insertNewObjectForEntityForName:@"Photo"  
                inManagedObjectContext:context];
```

```
        // initialize the photo from the Flickr data
```

```
        // perhaps even create other database objects (like the Photographer)
```

```
    }
```

```
    return photo;
```

```
}
```

```
@end
```


How do we create a category?

- CoreDataExample
- Main_iPad.storyboard
- Main_iPhone.storyboard
- Model.xcdatamodeld
- Photo.h
- Photo.m
- Photographer.h
- Photographer.m
- Images.xcassets
- Supporting Files
- CoreDataExampleTests
- Frameworks
- Products

- Cocoa Touch
 - C and C++
 - User Interface
 - Core Data
 - Resource
 - Other
- OS X
 - Cocoa
 - C and C++
 - User Interface
 - Core Data
 - Resource
 - Other

Choose your new file:

- Objective-C class
- Objective-C category**
- Objective-C class extension
- Objective-C protocol
- Objective-C test case class
- Objective-C category

An Objective-C category, with implementation and header files.

Choose New File ...then pick "Objective-C category" from the Cocoa Touch section.

Entity

Name: Multiple Values

Class: Multiple Values

Abstract Entity

Parent Entity: No Parent Entity

Indexes

+ -

Value

Versioning

Hash Modifier: Version Hash Modifier

Renaming ID: Renaming Identifier

- View Controller** - A controller that supports the fundamental view-management model in iPhone OS.
- Table View Controller** - A controller that manages a table view.
- Collection View Controller** - A controller that manages a collection view.

CoreDataExample
2 targets, iOS SDK 7.0

- CoreDataExample
 - Main_iPad.storyboard
 - Main_iPhone.storyboard
 - Model.xcdatamodeld
 - Photo.h
 - Photo.m
 - Photographer.h
 - Photographer.m
 - Images.xcassets
 - Supporting Files
 - CoreDataExampleTests
 - Frameworks
 - Products

Choose options for your new file:

Category: Flickr

Category on: Photo

Cancel Previous Next

Enter the name of the category, as well as the name of the class the category's methods will be added to.

Entity

Name: Multiple Values

Class: Multiple Values

Abstract Entity

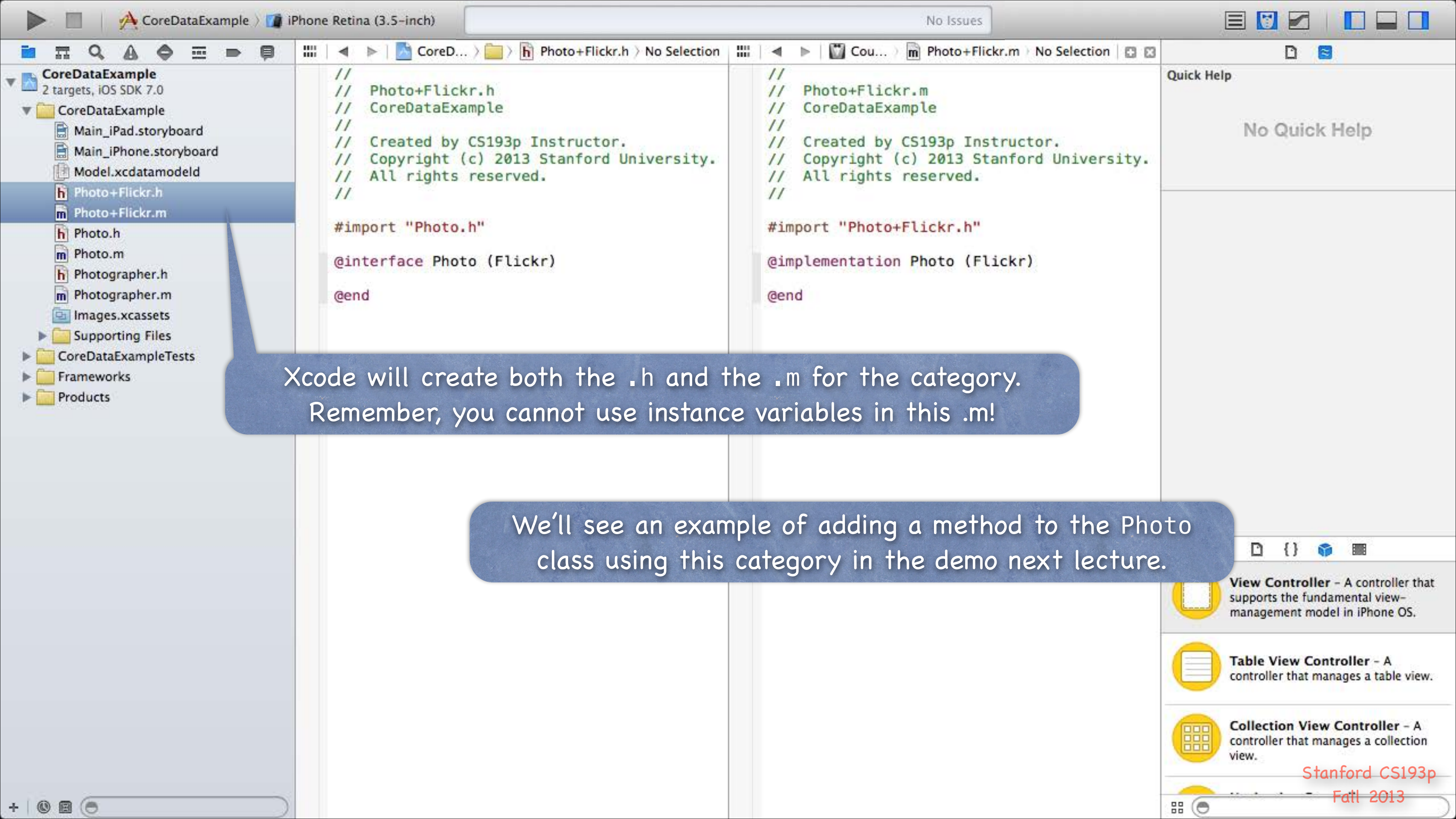
Parent Entity: No Parent Entity

Indexes: + -

User Info

Key	Value
Flash Modifier	Version Flash Modifier
Renaming ID	Renaming Identifier

- View Controller** - A controller that supports the fundamental view-management model in iPhone OS.
- Table View Controller** - A controller that manages a table view.
- Collection View Controller** - A controller that manages a collection view.



Xcode will create both the .h and the .m for the category. Remember, you cannot use instance variables in this .m!

We'll see an example of adding a method to the Photo class using this category in the demo next lecture.

Quick Help
No Quick Help

- View Controller** - A controller that supports the fundamental view-management model in iPhone OS.
- Table View Controller** - A controller that manages a table view.
- Collection View Controller** - A controller that manages a collection view.

Deletion

Deletion

Deleting objects from the database is easy (sometimes too easy!)

```
[aDocument.managedObjectContext deleteObject:photo];
```

Make sure that the rest of your objects in the database are in a sensible state after this.

Relationships will be updated for you (if you set Delete Rule for relationship attributes properly).

And don't keep any **strong** pointers to photo after you delete it!

prepareForDeletion

This is another method we sometimes put in a category of an NSManagedObject subclass ...

```
@implementation Photo (Deletion)
```

```
- (void)prepareForDeletion
```

```
{
```

```
    // we don't need to set our whoTook to nil or anything here (that will happen automatically)
```

```
    // but if Photographer had, for example, a "number of photos taken" attribute,
```

```
    // we might adjust it down by one here (e.g. self.whoTook.photoCount--).
```

```
}
```

```
@end
```

Querying

• So far you can ...

Create objects in the database with `insertNewObjectForEntityForName:inManagedObjectContext:`.
Get/set properties with `valueForKey:/setValue:forKey:` or `@properties` in a custom subclass.
Delete objects using the `NSManagedObjectContext deleteObject:` method.

• One very important thing left to know how to do: QUERY

Basically you need to be able to retrieve objects from the database, not just create new ones
You do this by executing an `NSFetchRequest` in your `NSManagedObjectContext`

• Four important things involved in creating an `NSFetchRequest`

1. `Entity` to fetch (required)
2. How many objects to fetch at a time and/or maximum to fetch (optional, default: all)
3. `NSSortDescriptors` to specify the order in which the array of fetched objects are returned
4. `NSPredicate` specifying which of those Entities to fetch (optional, default is all of them)

Querying

• Creating an `NSFetchRequest`

We'll consider each of these lines of code one by one ...

```
NSFetchRequest *request = [NSFetchRequest fetchRequestWithEntityName:@"Photo"];  
request.fetchBatchSize = 20;  
request.fetchLimit = 100;  
request.sortDescriptors = @[sortDescriptor];  
request.predicate = ...;
```

• Specifying the kind of Entity we want to fetch

A given fetch returns objects all of the same Entity.

You can't have a fetch that returns some Photos and some Photographers (it's one or the other).

• Setting fetch sizes/limits

If you created a fetch that would match 1000 objects, the request above faults 20 at a time.

And it would stop fetching after it had fetched 100 of the 1000.

Querying

• NSSortDescriptor

When we execute a fetch request, it's going to return an `NSArray` of `NSManagedObjects`. `NSArray`s are "ordered," so we should specify the order when we fetch.

We do that by giving the fetch request a list of "sort descriptors" that describe what to sort by.

```
NSSortDescriptor *sortDescriptor =  
    [NSSortDescriptor sortDescriptorWithKey:@"title"  
                    ascending:YES  
                    selector:@selector(localizedStandardCompare:)];
```

The `selector:` argument is just a method (conceptually) sent to each object to compare it to others. Some of these "methods" might be smart (i.e. they can happen on the database side).

`localizedStandardCompare:` is for ordering strings like the Finder on the Mac does (very common).

We give an array of these `NSSortDescriptors` to the `NSFetchRequest` because sometimes we want to sort first by one key (e.g. last name), then, within that sort, by another (e.g. first name).
Examples: `@[sortDescriptor]` or `@[lastNameSortDescriptor, firstNameSortDescriptor]`

Querying

• NSPredicate

This is the guts of how we specify exactly which objects we want from the database.

• Predicate formats

Creating one looks a lot like creating an NSString, but the contents have semantic meaning.

```
NSString *serverName = @"flickr-5";  
NSPredicate *predicate =  
    [NSPredicate predicateWithFormat:@"thumbnailURL contains %@", serverName];
```

• Examples

```
@“uniqueId = %@", [flickrInfo objectForKey:@"id"] // unique a photo in the database  
@“name contains[c] %@", (NSString *) // matches name case insensitively  
@“viewed > %@", (NSDate *) // viewed is a Date attribute in the data mapping  
@“whoTook.name = %@", (NSString *) // Photo search (by photographer’s name)  
@“any photos.title contains %@", (NSString *) // Photographer search (not a Photo search)
```

Many more options. Look at the class documentation for NSPredicate.

Querying

• NSCompoundPredicate

You can use AND and OR inside a predicate string, e.g. @"(name = %@) OR (title = %@)"

Or you can combine NSPredicate objects with special NSCompoundPredicates.

```
NSArray *array = @[predicate1, predicate2];
```

```
NSPredicate *predicate = [NSCompoundPredicate andPredicateWithSubpredicates:array];
```

This predicate is "predicate1 AND predicate2". Or available too, of course.

Advanced Querying

Key Value Coding

Can actually do predicates like `@“photos.@count > 5”` (Photographers with more than 5 photos).
`@count` is a function (there are others) executed in the database itself.

<https://developer.apple.com/library/ios/documentation/cocoa/conceptual/KeyValueCoding/Articles/CollectionOperators.html>.

By the way, all this stuff (and more) works on dictionaries, arrays and sets too ...

e.g. `[propertyListResults valueForKeyPath:@“photos.photo.@avg.latitude”]` on Flickr results
returns the average latitude of all of the photos in the results (yes, really)

e.g. `@“photos.photo.title.length”` would return an array of the lengths of the titles of the photos

NSEExpression

Advanced topic. Can do sophisticated data gathering from the database.

No time to cover it now, unfortunately.

If interested, for both NSEExpression and Key Value Coding queries, investigate ...

```
NSFetchRequest *request = [NSFetchRequest fetchRequestWithEntityName:@“...”];
```

```
[request setResultType:NSDictionaryResultType]; // fetch returns array of dicts instead of NSMO's
```

```
[request setPropertiesToFetch:@[@“name”, expression, etc.]];
```

Querying

● Putting it all together

Let's say we want to query for all Photographers ...

```
NSFetchRequest *request = [NSFetchRequest fetchRequestWithEntityName:@"Photographer"];
```

... who have taken a photo in the last 24 hours ...

```
NSDate *yesterday = [NSDate dateWithTimeIntervalSinceNow:-24*60*60];
```

```
request.predicate = [NSPredicate predicateWithFormat:@"any photos.uploadDate > %@", yesterday];
```

... sorted by the Photographer's name ...

```
request.sortDescriptors = @[[NSSortDescriptor sortDescriptorWithKey:@"name" ascending:YES]];
```

Querying

• Executing the fetch

```
NSManagedObjectContext *context = aDocument.managedObjectContext;  
NSError *error;  
NSArray *photographers = [context executeFetchRequest:request error:&error];
```

Returns `nil` if there is an error (check the `NSError` for details).

Returns an empty array (not `nil`) if there are no matches in the database.

Returns an `NSArray` of `NSManagedObjects` (or subclasses thereof) if there were any matches.

You can pass `NULL` for `error:` if you don't care why it fails.

That's it. Very simple really.

Query Results

• Faulting

The above fetch does not necessarily fetch any actual data.

It could be an array of “as yet unfaulted” objects, waiting for you to access their attributes.

Core Data is very smart about “faulting” the data in as it is actually accessed.

For example, if you did something like this ...

```
for (Photographer *photographer in photographers) {  
    NSLog(@"fetched photographer %@", photographer);  
}
```

You may or may not see the names of the photographers in the output

(you might just see “unfaulted object”, depending on whether it prefetched them)

But if you did this ...

```
for (Photographer *photographer in photographers) {  
    NSLog(@"fetched photographer named %@", photographer.name);  
}
```

... then you would definitely fault all the Photographers in from the database.

That's because in the second case, you actually access the `NSManagedObject`'s data.

Core Data Thread Safety

◉ NSManagedObjectContext is not thread safe

Luckily, Core Data access is usually very fast, so multithreading is only rarely needed. Usually we create NSManagedObjectContext using a queue-based concurrency model. This means that you can only touch a context and its NSMO's in the queue it was created on.

◉ Thread-Safe Access to an NSManagedObjectContext

```
[context performBlock:^( // or performBlockAndWait:  
    // do stuff with context in its safe queue (the queue it was created on)  
}];
```

Note that the Q might well be the main Q, so you're not necessarily getting "multithreaded."

◉ Parent Context (advanced)

Some contexts (including UIManagedDocument ones) have a parentContext (a @property on NSMOC). This parentContext will almost always be on a separate queue, but access the same database. This means you can performBlock: on it to access the database off the main queue (e.g.). But it is still a different context, so you'll have to refetch in the child context to see any changes.

Core Data

- There is so much more (that we don't have time to talk about!)
 - Optimistic locking (`deleteConflictsForObject:`)
 - Rolling back unsaved changes
 - Undo/Redo
 - Staleness (how long after a fetch until a refetch of an object is required?)

Coming Up

👁 Homework

Assignment 5 due Wednesday.

Final homework (Assignment 6) will be assigned Wednesday, due the next Wednesday.

👁 Wednesday

Final Project Requirements

Core Data and UITableView

Core Data Demo

👁 Next Week

Multitasking

Advanced Segueing

Map Kit?