Stanford CS193p

Developing Applications for iOS
Fall 2011

Today

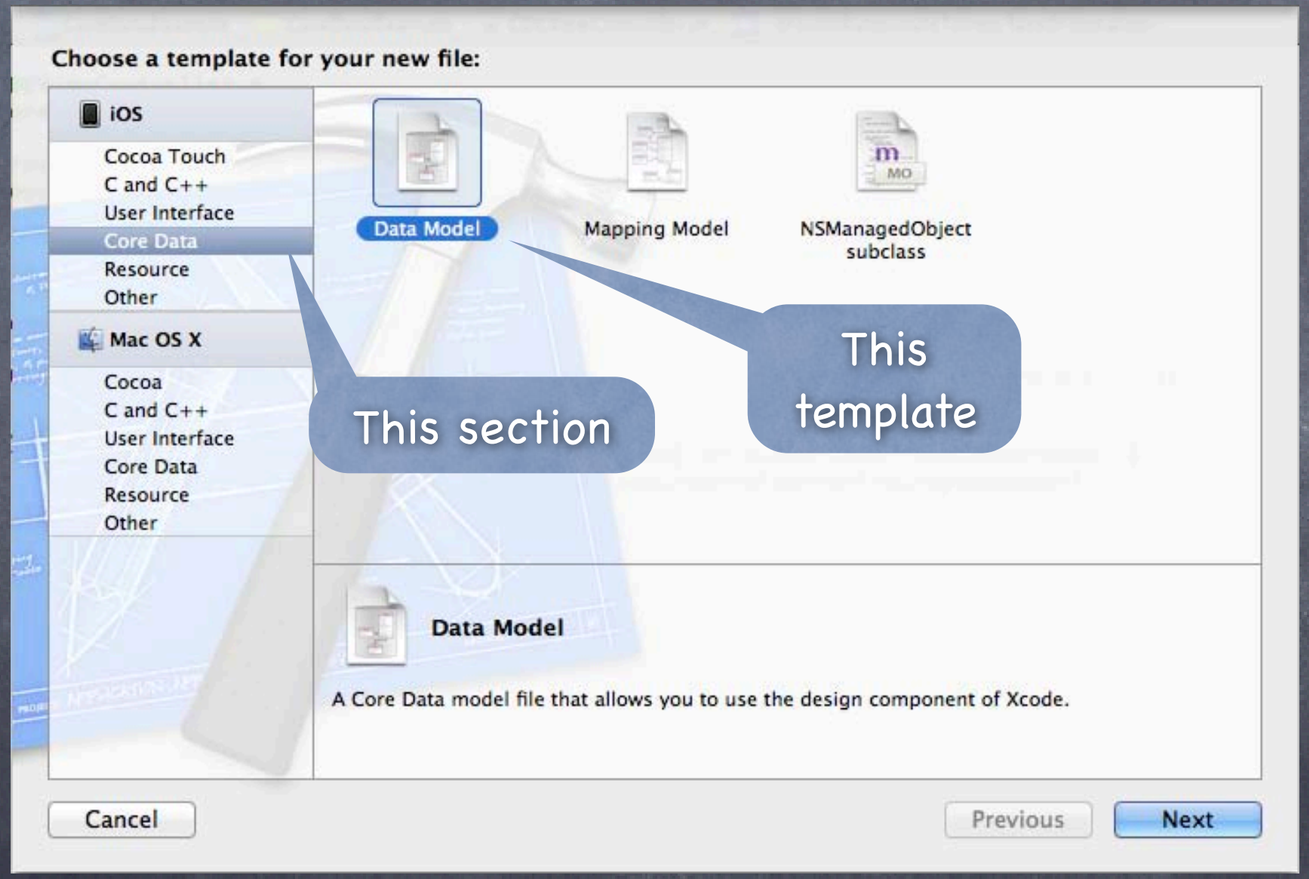
- Core Data and Documents
  - This is how you store something serious in iOS
  - Easy entreé into iCloud
- NSNotificationCenter
  - The little "radio station" we talked about in the very first lecture
- Objective-C Categories
  - A way to add methods to a class without subclassing

Core Data

- We're object-oriented programmers and we don't like C APIs!
  We want to store our data using object-oriented programming!
- Enter Core Data
  Object-oriented database.
- It's a way of creating an object graph backed by a database
  Usually SQL.
- 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 `@property`s on those objects.

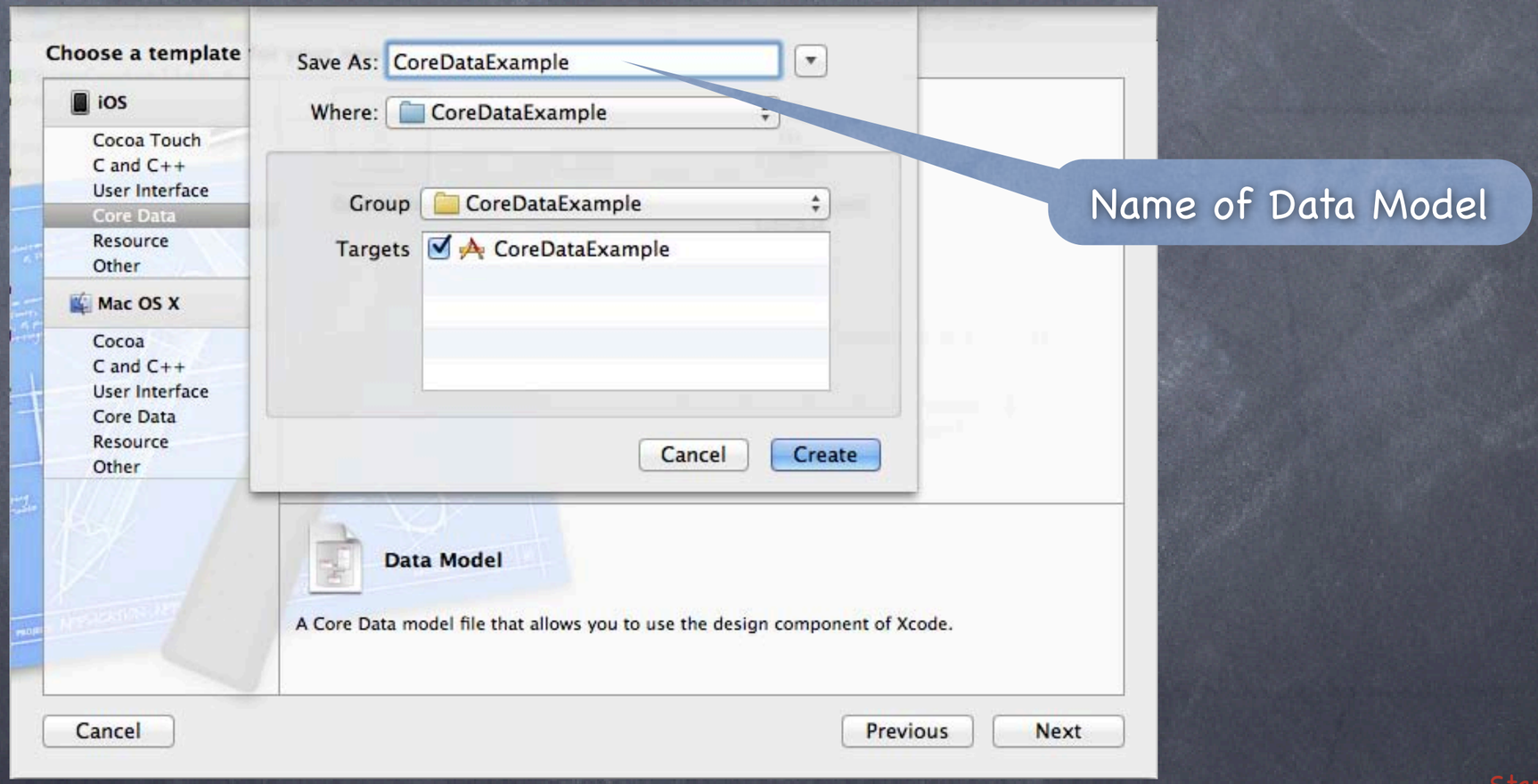
Core Data

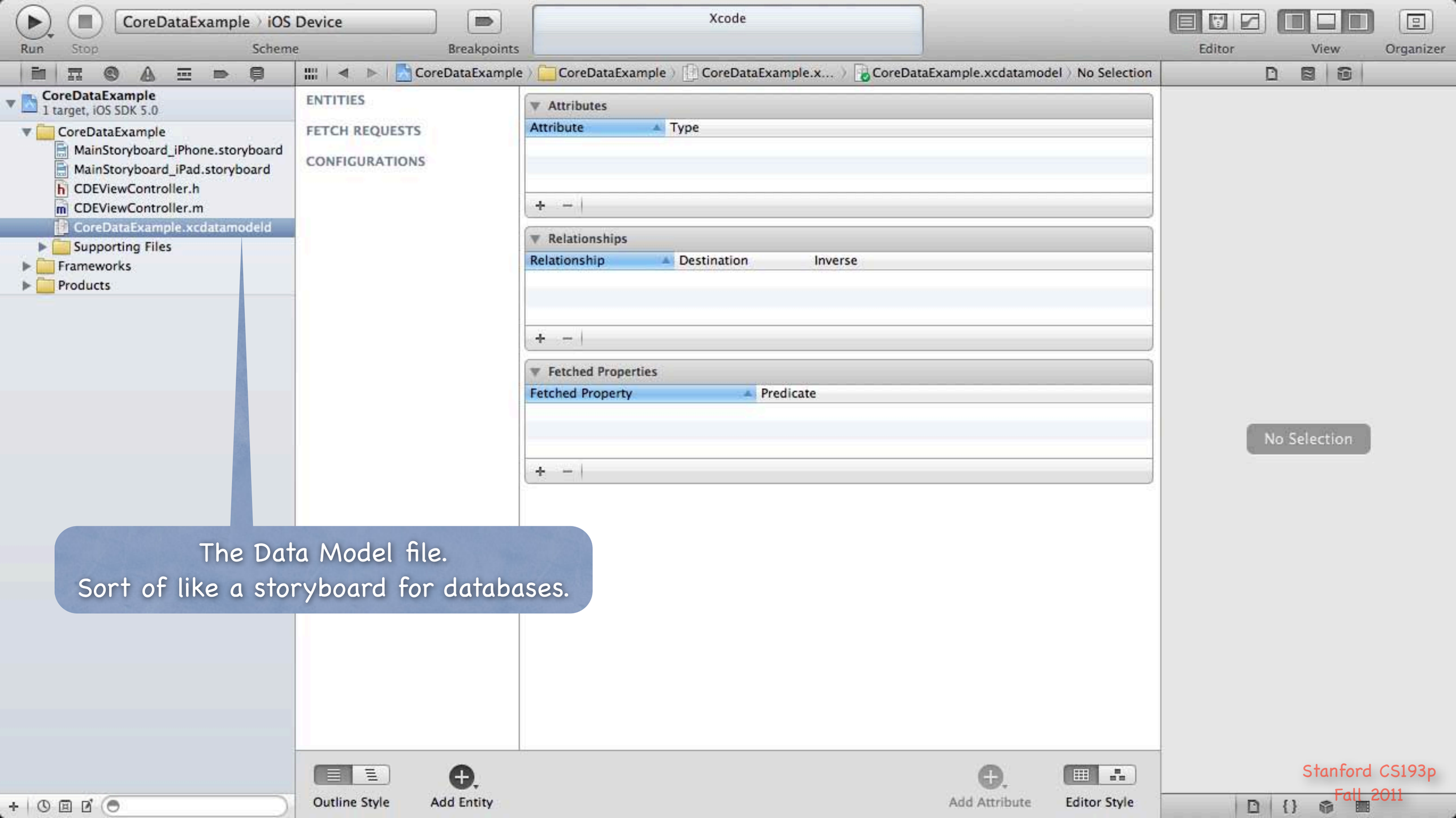
- Creating a visual map of your application's database objects
  New File … then Data Model under Core Data.

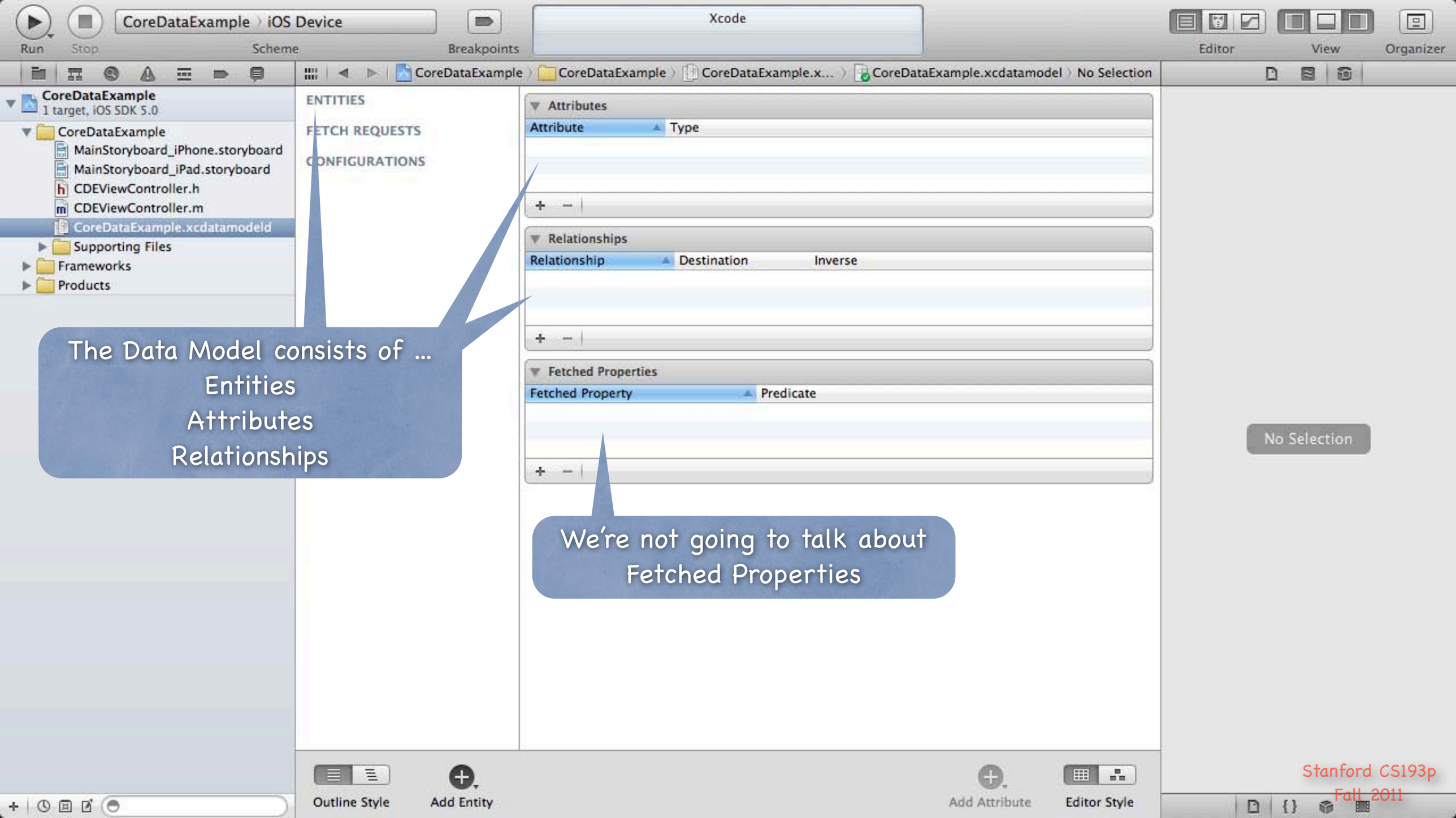


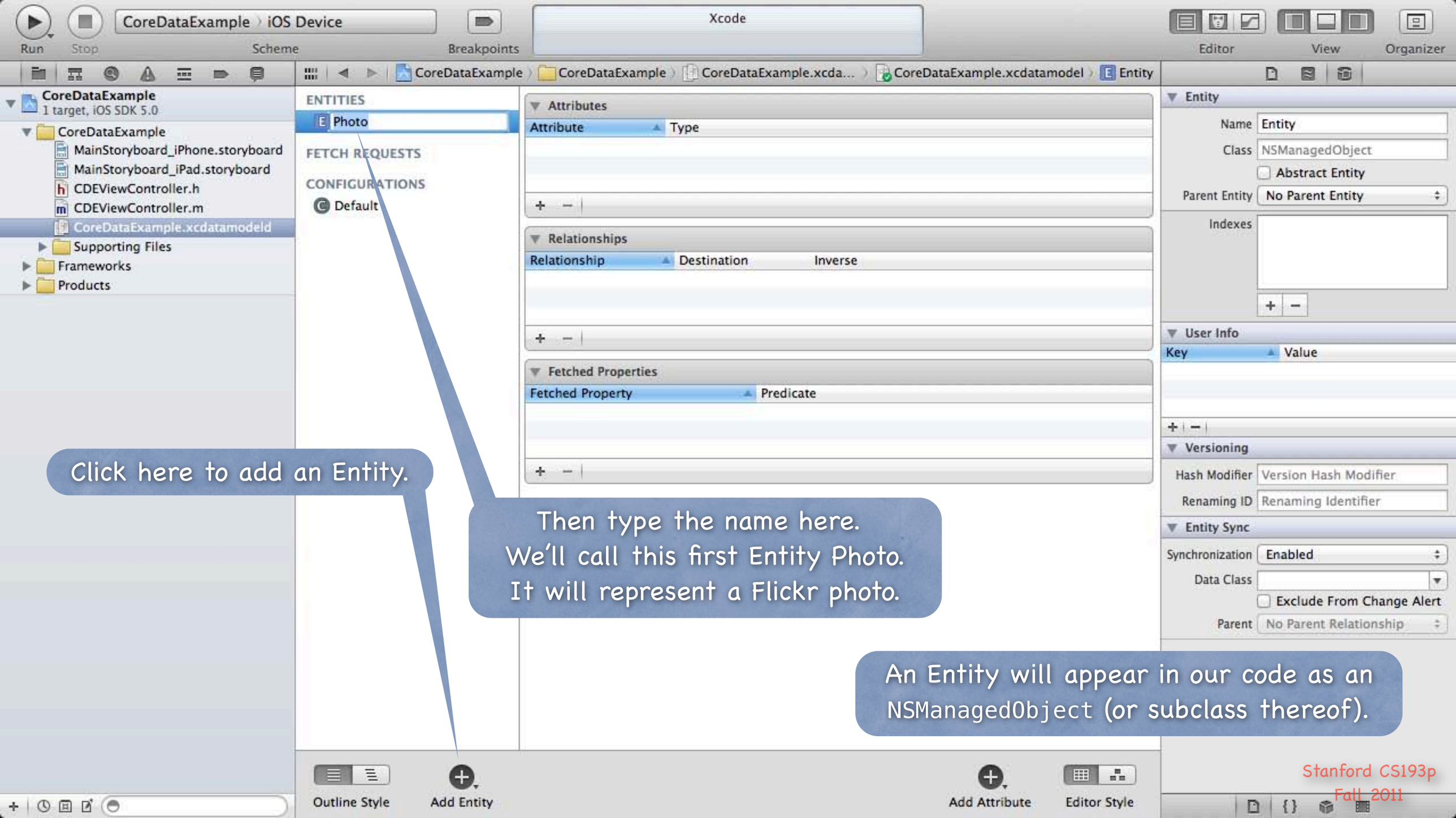
Core Data

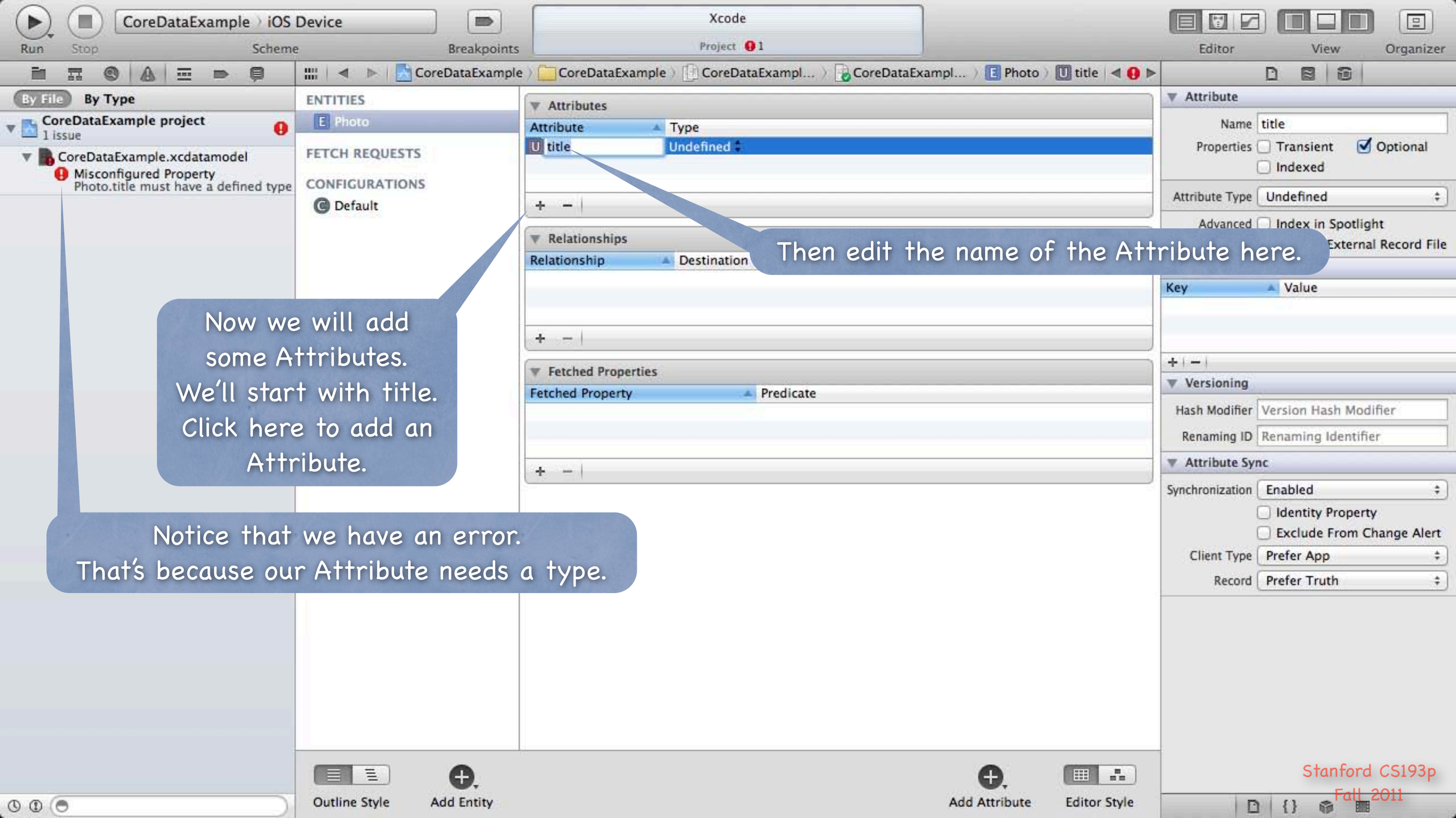
- Creating a visual map of your application's database objects
  Unless we have multiple databases, usually we name the Data Model our application name

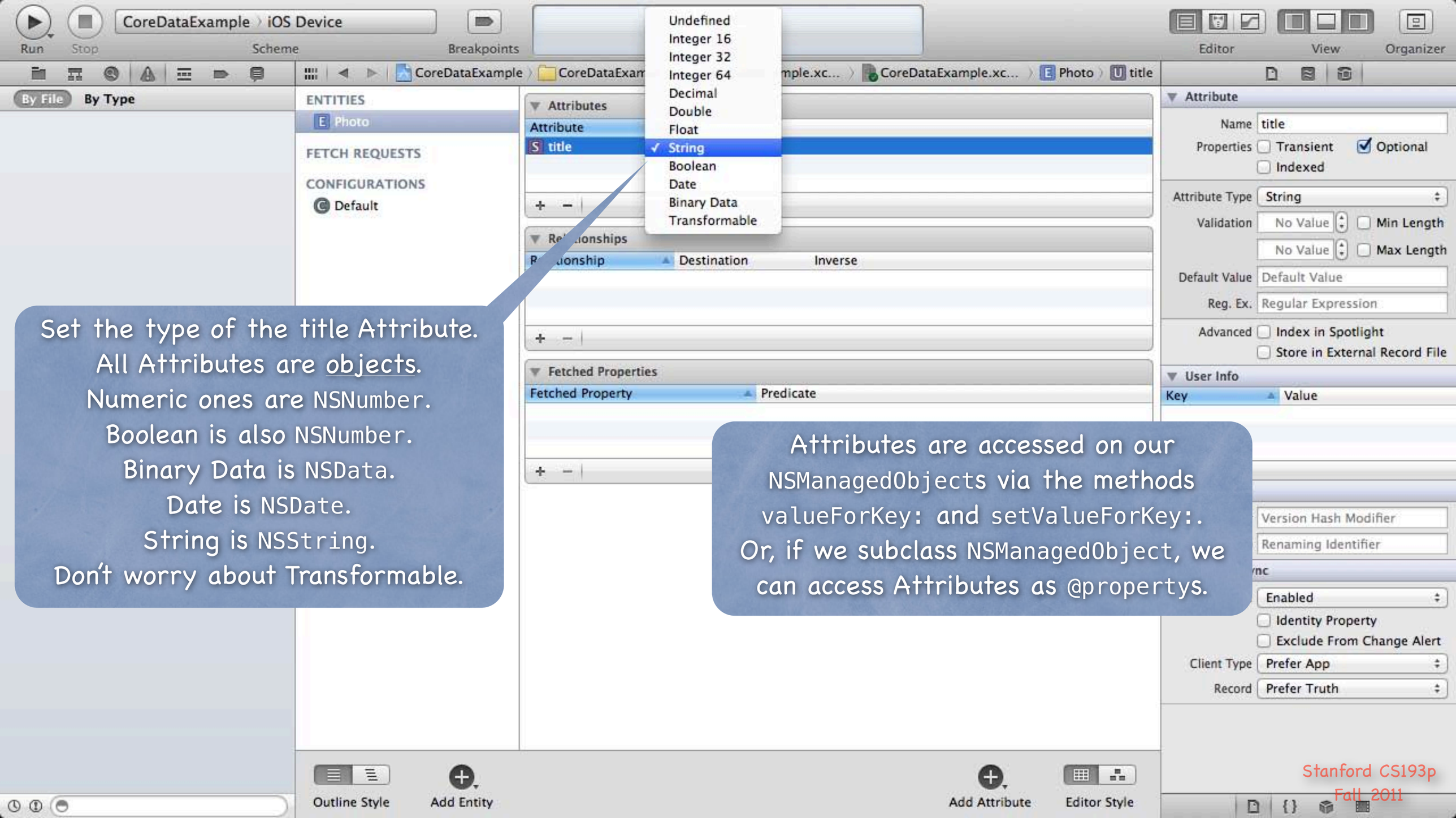


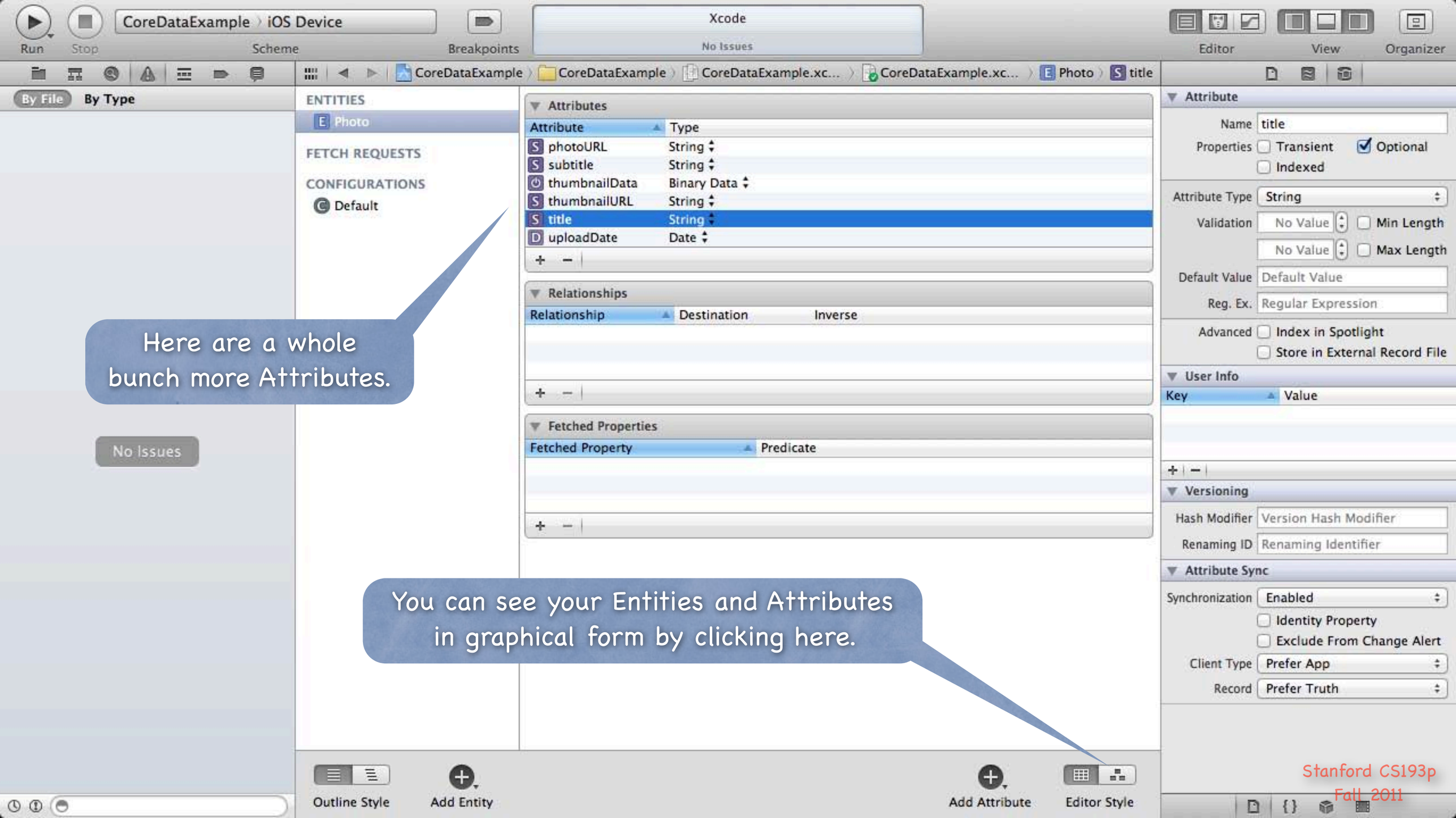

Run
Stop
CoreDataExample › iOS Device
Scheme
Breakpoints
Xcode
Editor
View
Organizer
CoreDataExample › CoreDataExample › CoreDataExample.x... › CoreDataExample.xcdatamodel › No Selection
CoreDataExample
1 target, iOS SDK 5.0
CoreDataExample
MainStoryboard_iPhone.storyboard
MainStoryboard_iPad.storyboard
CDEViewController.h
CDEViewController.m
CoreDataExample.xcdatamodeld
Supporting Files
Frameworks
Products
ENTITIES
FETCH REQUESTS
CONFIGURATIONS
Attributes
Attribute
Type
Relationships
Relationship
Destination
Inverse
Fetched Properties
Fetched Property
Predicate
No Selection
The Data Model file.
Sort of like a storyboard for databases.
Outline Style
Add Entity
Add Attribute
Editor Style
Stanford CS193p
Fall 2011

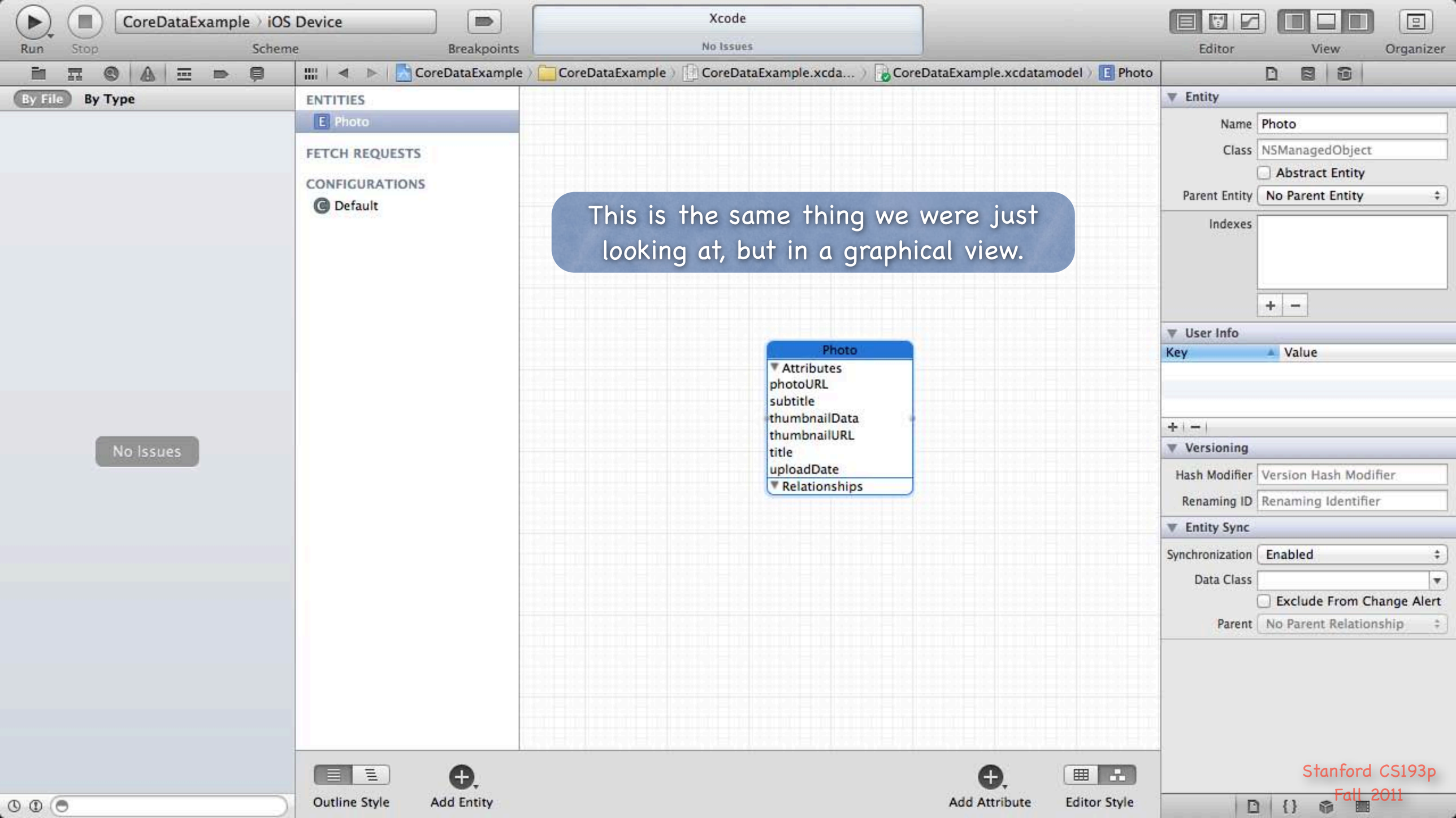

Run
Stop
CoreDataExample › iOS Device
Scheme
Breakpoints
Xcode
Editor
View
Organizer
CoreDataExample › CoreDataExample › CoreDataExample.x... › CoreDataExample.xcdatamodel › No Selection
CoreDataExample
1 target, iOS SDK 5.0
CoreDataExample
MainStoryboard_iPhone.storyboard
MainStoryboard_iPad.storyboard
CDEViewController.h
CDEViewController.m
CoreDataExample.xcdatamodeld
Supporting Files
Frameworks
Products
ENTITIES
FETCH REQUESTS
CONFIGURATIONS
Attributes
Attribute
Type
Relationships
Relationship
Destination
Inverse
Fetched Properties
Fetched Property
Predicate
The Data Model consists of ...
Entities
Attributes
Relationships
We're not going to talk about
Fetched Properties
No Selection
Outline Style
Add Entity
Add Attribute
Editor Style
Stanford CS193p
Fall 2011

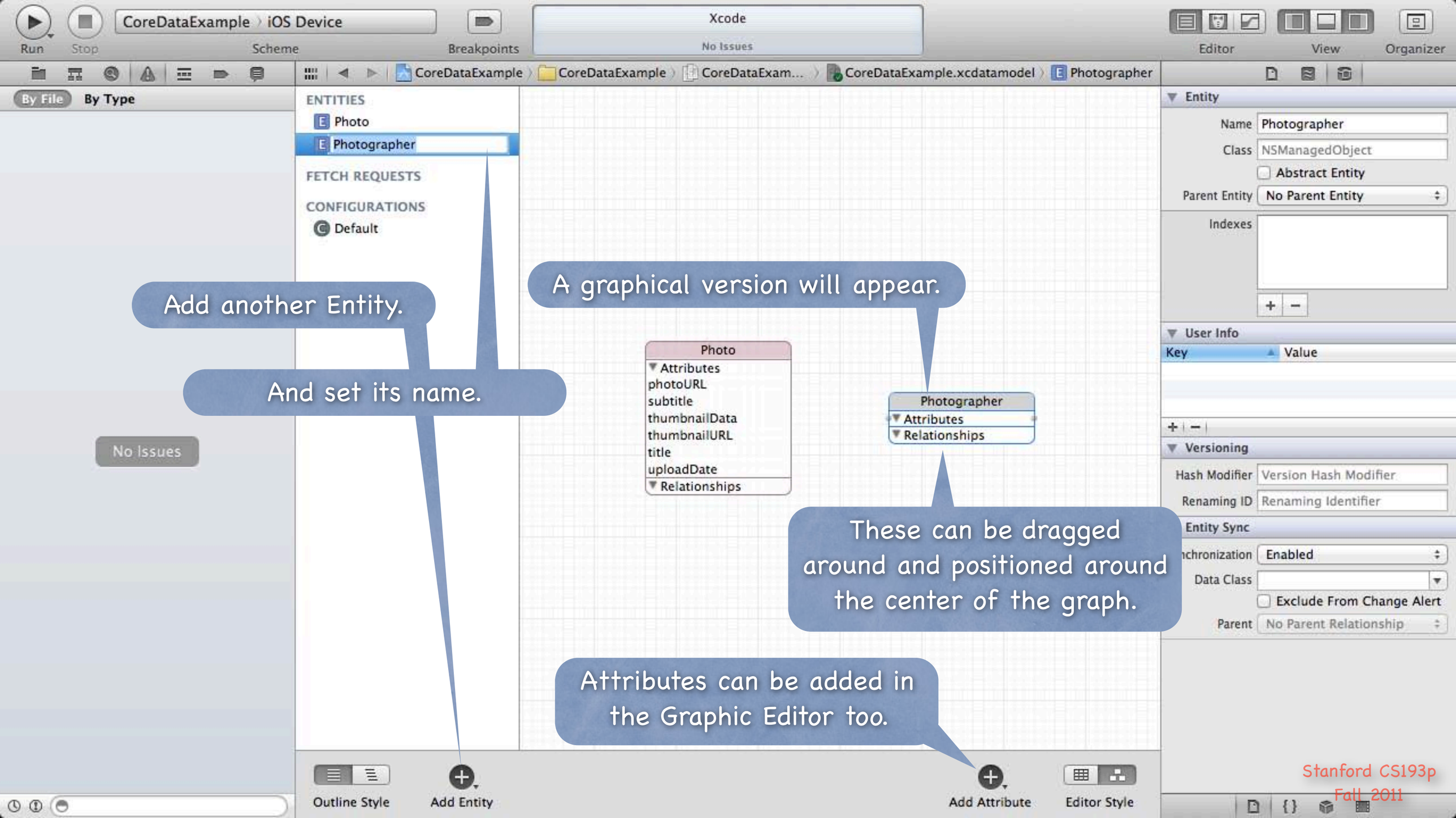

Run
Stop
CoreDataExample › iOS Device
Scheme
Breakpoints
Xcode
Editor
View
Organizer
CoreDataExample
1 target, iOS SDK 5.0
CoreDataExample
MainStoryboard_iPhone.storyboard
MainStoryboard_iPad.storyboard
CDEViewController.h
CDEViewController.m
CoreDataExample.xcdatamodeld
Supporting Files
Frameworks
Products
CoreDataExample › CoreDataExample › CoreDataExample.xcda... › CoreDataExample.xcdatamodel › Entity
ENTITIES
Photo
FETCH REQUESTS
CONFIGURATIONS
Default
Attributes
Attribute
Type
Relationships
Relationship
Destination
Inverse
Fetched Properties
Fetched Property
Predicate
Entity
Name
Entity
Class
NSManagedObject
Abstract Entity
Parent Entity
No Parent Entity
Indexes
User Info
Key
Value
Versioning
Hash Modifier
Version Hash Modifier
Renaming ID
Renaming Identifier
Entity Sync
Synchronization
Enabled
Data Class
Exclude From Change Alert
Parent
No Parent Relationship
Click here to add an Entity.
Then type the name here.
We'll call this first Entity Photo.
It will represent a Flickr photo.
An Entity will appear in our code as an NSManagedObject (or subclass thereof).
Outline Style
Add Entity
Add Attribute
Editor Style
Stanford CS193p
Fall 2011

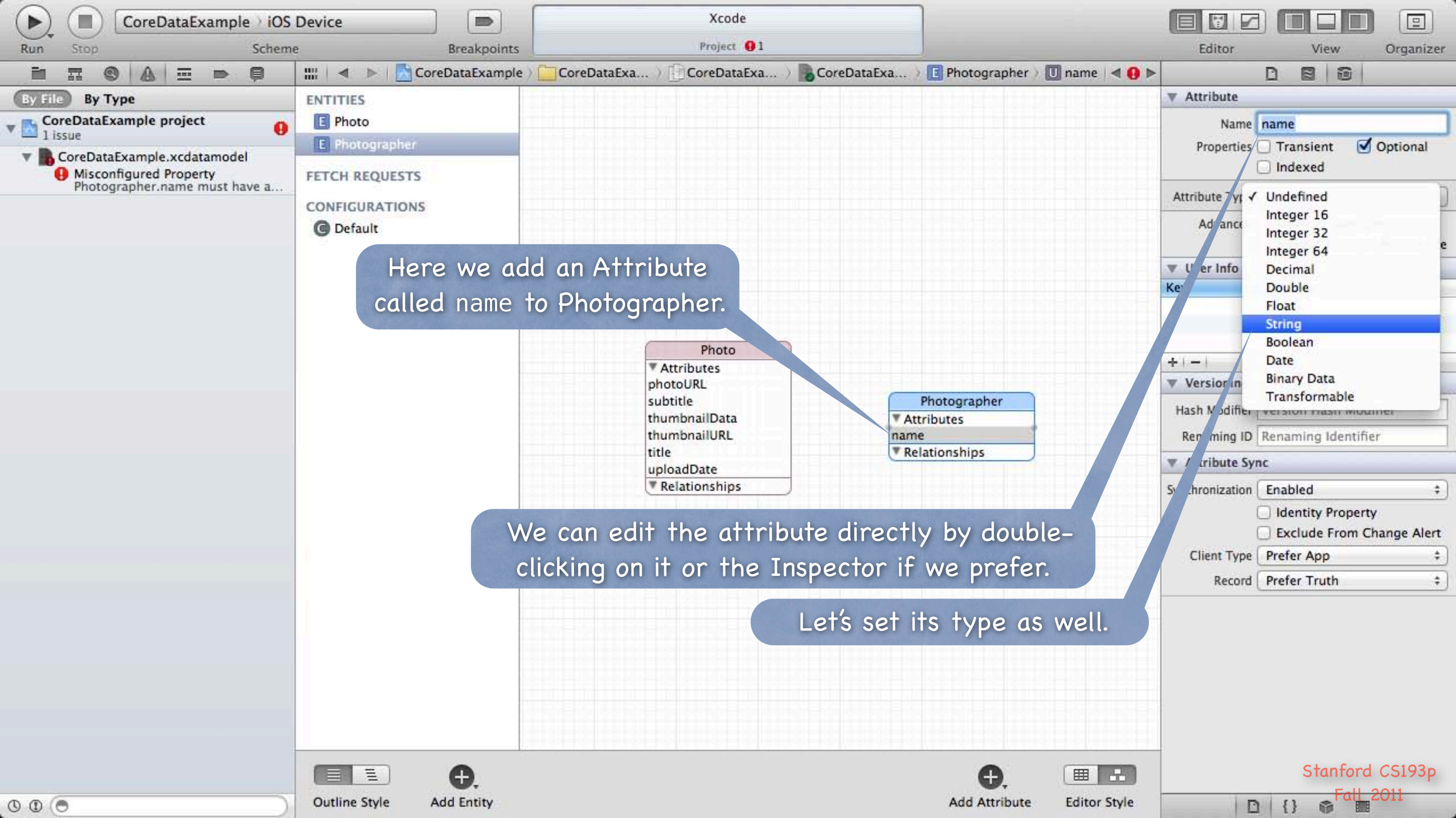

Run
Stop
CoreDataExample › iOS Device
Scheme
Breakpoints
Xcode
Project 1
Editor
View
Organizer
By File
By Type
CoreDataExample project
1 issue
CoreDataExample.xcdatamodel
Misconfigured Property
Photo.title must have a defined type
ENTITIES
Photo
FETCH REQUESTS
CONFIGURATIONS
Default
CoreDataExample › CoreDataExample › CoreDataExampl... › CoreDataExampl... › Photo › title
Attributes
Attribute
Type
title
Undefined
Relationships
Relationship
Destination
Fetched Properties
Fetched Property
Predicate
Attribute
Name
title
Properties
Transient
Optional
Indexed
Attribute Type
Undefined
Advanced
Index in Spotlight
External Record File
Key
Value
Versioning
Hash Modifier
Version Hash Modifier
Renaming ID
Renaming Identifier
Attribute Sync
Synchronization
Enabled
Identity Property
Exclude From Change Alert
Client Type
Prefer App
Record
Prefer Truth
Outline Style
Add Entity
Add Attribute
Editor Style
Then edit the name of the Attribute here.
Now we will add some Attributes. We'll start with title. Click here to add an Attribute.
Notice that we have an error. That's because our Attribute needs a type.
Stanford CS193p
Fall 2011

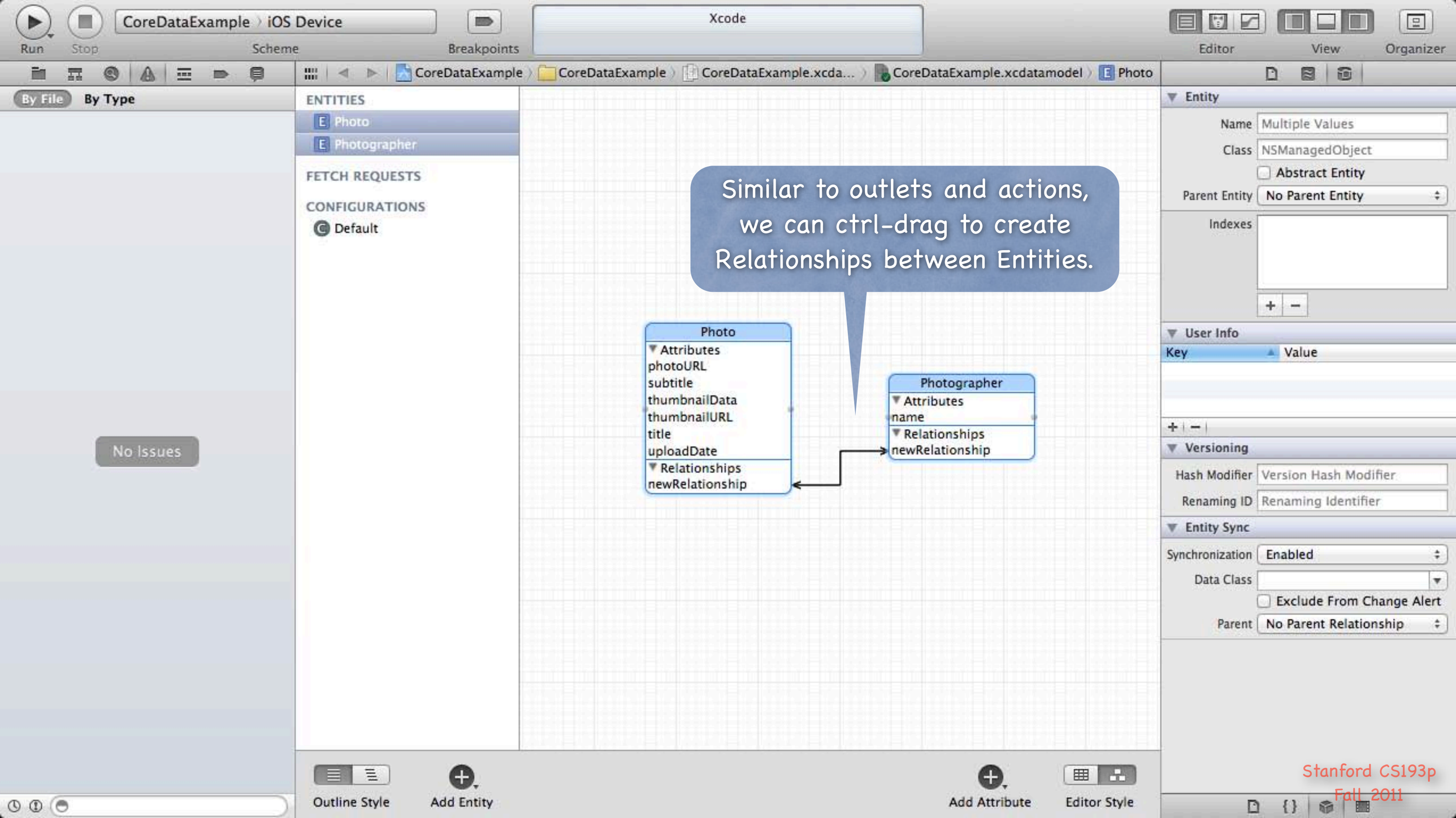

Undefined
Integer 16
Integer 32
Integer 64
Decimal
Double
Float
String
Boolean
Date
Binary Data
Transformable
Set the type of the title Attribute.
All Attributes are objects.
Numeric ones are NSNumber.
Boolean is also NSNumber.
Binary Data is NSData.
Date is NSDate.
String is NSString.
Don't worry about Transformable.
Attributes are accessed on our NSManagedObjects via the methods valueForKey: and setValueForKey:.
Or, if we subclass NSManagedObject, we can access Attributes as @propertys.
Stanford CS193p
Fall 2011

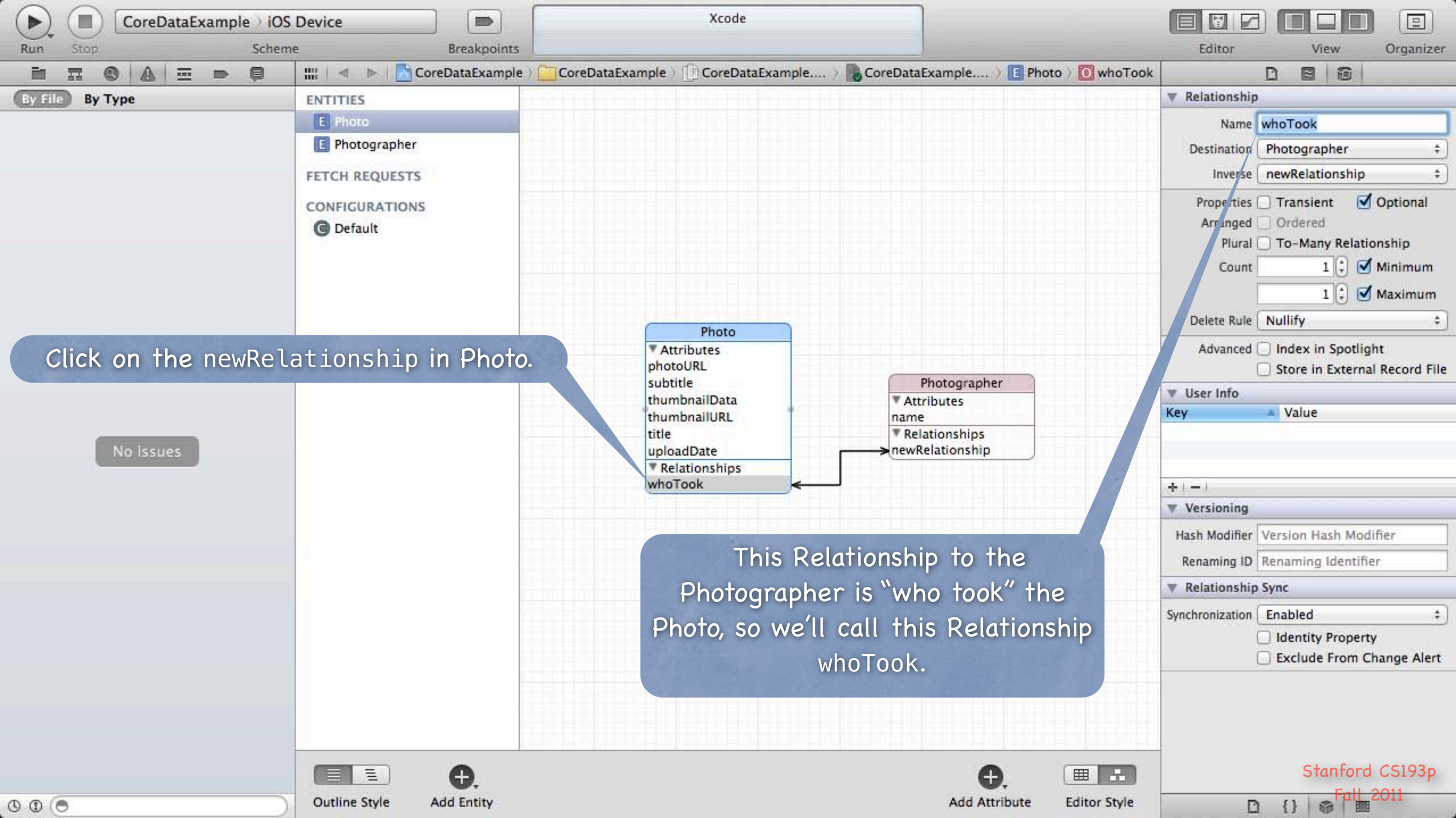

Run
Stop
CoreDataExample › iOS Device
Scheme
Breakpoints
Xcode
No Issues
Editor
View
Organizer
By File
By Type
ENTITIES
Photo
FETCH REQUESTS
CONFIGURATIONS
Default
CoreDataExample › CoreDataExample › CoreDataExample.xc... › CoreDataExample.xc... › Photo › title
Attributes
Attribute
Type
photoURL String
subtitle String
thumbnailData Binary Data
thumbnailURL String
title String
uploadDate Date
Relationships
Relationship
Destination
Inverse
Fetched Properties
Fetched Property
Predicate
Here are a whole bunch more Attributes.
No Issues
You can see your Entities and Attributes in graphical form by clicking here.
Outline Style
Add Entity
Add Attribute
Editor Style
Attribute
Name title
Properties Transient Optional Indexed
Attribute Type String
Validation No Value Min Length
No Value Max Length
Default Value Default Value
Reg. Ex. Regular Expression
Advanced Index in Spotlight
Store in External Record File
User Info
Key
Value
Versioning
Hash Modifier Version Hash Modifier
Renaming ID Renaming Identifier
Attribute Sync
Synchronization Enabled
Identity Property
Exclude From Change Alert
Client Type Prefer App
Record Prefer Truth
Stanford CS193p
Fall 2011

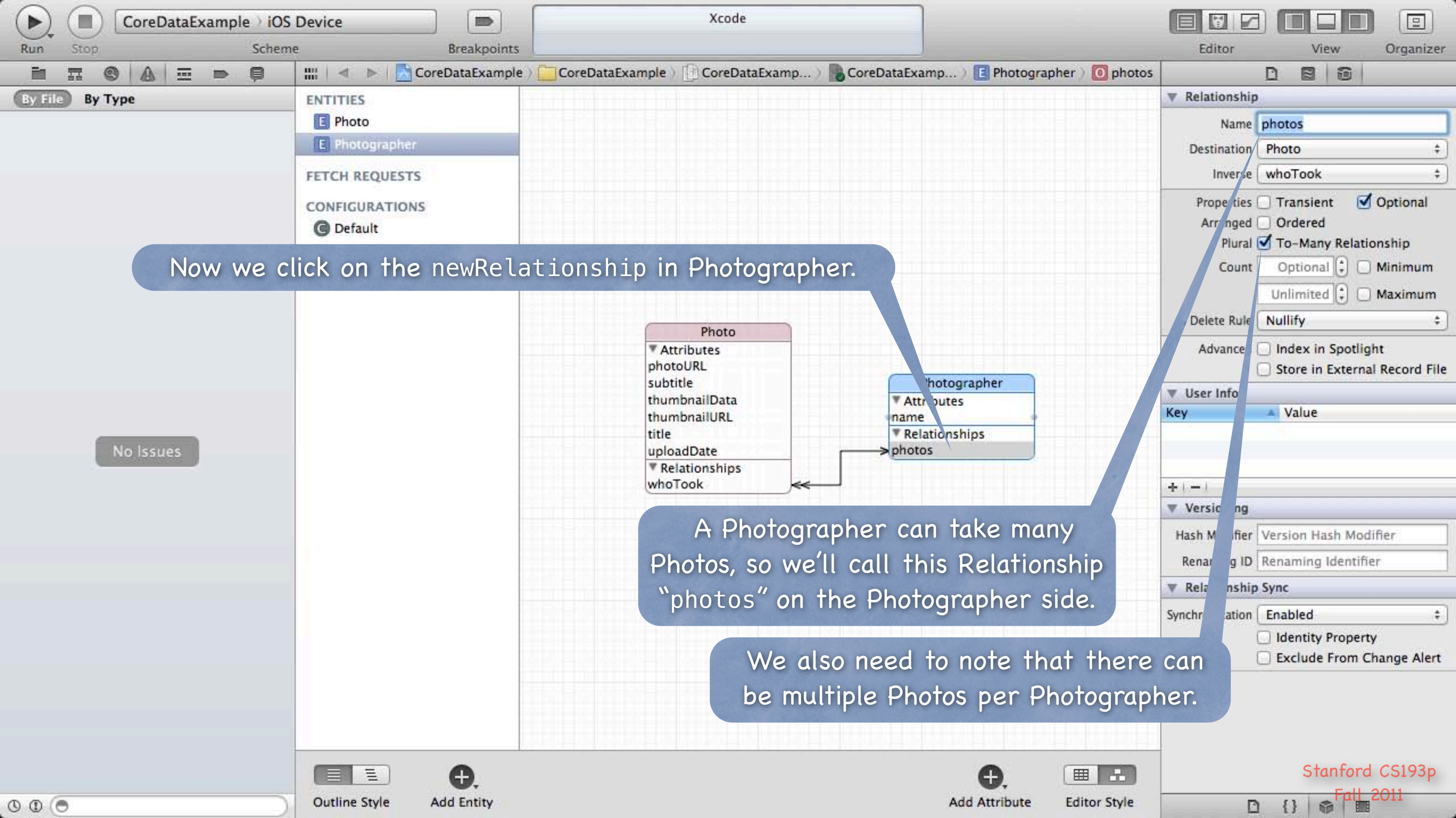

Run
Stop
CoreDataExample › iOS Device
Scheme
Breakpoints
Xcode
No Issues
Editor
View
Organizer
By File
By Type
No Issues
CoreDataExample › CoreDataExample › CoreDataExample.xcda... › CoreDataExample.xcdatamodel › Photo
ENTITIES
Photo
FETCH REQUESTS
CONFIGURATIONS
Default
This is the same thing we were just looking at, but in a graphical view.
Photo
Attributes
photoURL
subtitle
thumbnailData
thumbnailURL
title
uploadDate
Relationships
Entity
Name Photo
Class NSManagedObject
Abstract Entity
Parent Entity No Parent Entity
Indexes
User Info
Key Value
Versioning
Hash Modifier Version Hash Modifier
Renaming ID Renaming Identifier
Entity Sync
Synchronization Enabled
Data Class
Exclude From Change Alert
Parent No Parent Relationship
Outline Style
Add Entity
Add Attribute
Editor Style
Stanford CS193p
Fall 2011

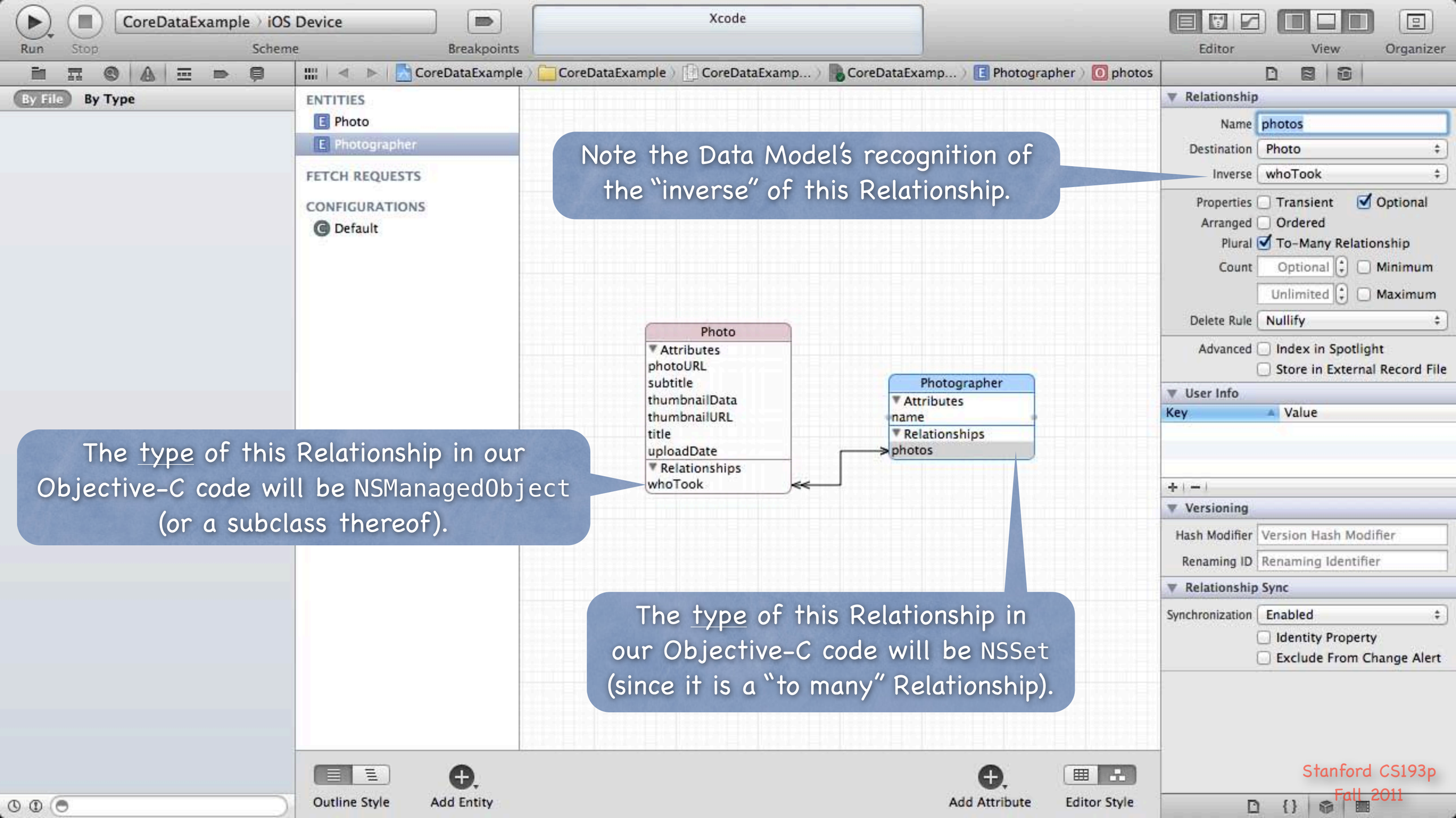

Run
Stop
CoreDataExample › iOS Device
Scheme
Breakpoints
Xcode
No Issues
Editor
View
Organizer
By File
By Type
No Issues
CoreDataExample › CoreDataExample › CoreDataExam... › CoreDataExample.xcdatamodel › Photographer
ENTITIES
Photo
Photographer
FETCH REQUESTS
CONFIGURATIONS
Default
Add another Entity.
And set its name.
A graphical version will appear.
Photo
Attributes
photoURL
subtitle
thumbnailData
thumbnailURL
title
uploadDate
Relationships
Photographer
Attributes
Relationships
These can be dragged around and positioned around the center of the graph.
Attributes can be added in the Graphic Editor too.
Entity
Name Photographer
Class NSManagedObject
Abstract Entity
Parent Entity No Parent Entity
Indexes
User Info
Key
Value
Versioning
Hash Modifier Version Hash Modifier
Renaming ID Renaming Identifier
Entity Sync
Enabled
Data Class
Exclude From Change Alert
Parent No Parent Relationship
Outline Style
Add Entity
Add Attribute
Editor Style
Stanford CS193p
Fall 2011


Run
Stop
CoreDataExample › iOS Device
Scheme
Breakpoints
Xcode
Project 1
Editor
View
Organizer
By File
By Type
CoreDataExample project
1 issue
CoreDataExample.xcdatamodel
Misconfigured Property
Photographer.name must have a...
ENTITIES
Photo
Photographer
FETCH REQUESTS
CONFIGURATIONS
Default
Here we add an Attribute called name to Photographer.
Photo
Attributes
photoURL
subtitle
thumbnailData
thumbnailURL
title
uploadDate
Relationships
Photographer
Attributes
name
Relationships
We can edit the attribute directly by double-clicking on it or the Inspector if we prefer.
Let's set its type as well.
Attribute
Name
name
Properties
Transient
Optional
Indexed
Undefined
Integer 16
Integer 32
Integer 64
Decimal
Double
Float
String
Boolean
Date
Binary Data
Transformable
Renaming ID
Renaming Identifier
Synchronization
Enabled
Identity Property
Exclude From Change Alert
Client Type
Prefer App
Record
Prefer Truth
Outline Style
Add Entity
Add Attribute
Editor Style
Stanford CS193p
Fall 2011


Run
Stop
CoreDataExample › iOS Device
Scheme
Breakpoints
Xcode
Editor
View
Organizer
By File
By Type
No Issues
CoreDataExample › CoreDataExample › CoreDataExample.xcda... › CoreDataExample.xcdatamodel › Photo
ENTITIES
Photo
Photographer
FETCH REQUESTS
CONFIGURATIONS
Default
Similar to outlets and actions, we can ctrl-drag to create Relationships between Entities.
Photo
Attributes
photoURL
subtitle
thumbnailData
thumbnailURL
title
uploadDate
Relationships
newRelationship
Photographer
Attributes
name
Relationships
newRelationship
Entity
Name
Multiple Values
Class
NSManagedObject
Abstract Entity
Parent Entity
No Parent Entity
Indexes
User Info
Key
Value
Versioning
Hash Modifier
Version Hash Modifier
Renaming ID
Renaming Identifier
Entity Sync
Synchronization
Enabled
Data Class
Exclude From Change Alert
Parent
No Parent Relationship
Outline Style
Add Entity
Add Attribute
Editor Style
Stanford CS193p
Fall 2011


Run
Stop
CoreDataExample › iOS Device
Scheme
Breakpoints
Xcode
Editor
View
Organizer
By File
By Type
No Issues
CoreDataExample › CoreDataExample › CoreDataExample.... › CoreDataExample.... › Photo › whoTook
ENTITIES
Photo
Photographer
FETCH REQUESTS
CONFIGURATIONS
Default
Click on the newRelationship in Photo.
Photo
Attributes
photoURL
subtitle
thumbnailData
thumbnailURL
title
uploadDate
Relationships
whoTook
Photographer
Attributes
name
Relationships
newRelationship
This Relationship to the Photographer is "who took" the Photo, so we'll call this Relationship whoTook.
Relationship
Name whoTook
Destination Photographer
Inverse newRelationship
Properties Transient Optional
Arranged Ordered
Plural To-Many Relationship
Count 1 Minimum
1 Maximum
Delete Rule Nullify
Advanced Index in Spotlight
Store in External Record File
User Info
Key Value
Versioning
Hash Modifier Version Hash Modifier
Renaming ID Renaming Identifier
Relationship Sync
Synchronization Enabled
Identity Property
Exclude From Change Alert
Outline Style
Add Entity
Add Attribute
Editor Style
Stanford CS193p
Fall 2011


Run
Stop
CoreDataExample › iOS Device
Scheme
Breakpoints
Xcode
Editor
View
Organizer
By File
By Type
No Issues
CoreDataExample › CoreDataExample › CoreDataExamp... › CoreDataExamp... › Photographer › photos
ENTITIES
Photo
Photographer
FETCH REQUESTS
CONFIGURATIONS
Default
Now we click on the newRelationship in Photographer.
Photo
Attributes
photoURL
subtitle
thumbnailData
thumbnailURL
title
uploadDate
Relationships
whoTook
Photographer
Attributes
name
Relationships
photos
A Photographer can take many Photos, so we'll call this Relationship "photos" on the Photographer side.
We also need to note that there can be multiple Photos per Photographer.
Relationship
Name photos
Destination Photo
Inverse whoTook
Properties Transient Optional
Arranged Ordered
Plural To-Many Relationship
Count Optional Minimum
Unlimited Maximum
Delete Rule Nullify
Advanced Index in Spotlight
Store in External Record File
User Info
Key
Value
Version Hash Modifier
Renaming Identifier
Synchronization Enabled
Identity Property
Exclude From Change Alert
Outline Style
Add Entity
Add Attribute
Editor Style
Stanford CS193p
Fall 2011


Run
Stop
CoreDataExample › iOS Device
Scheme
Breakpoints
Xcode
Editor
View
Organizer
By File
By Type
ENTITIES
Photo
Photographer
FETCH REQUESTS
CONFIGURATIONS
Default
CoreDataExample › CoreDataExample › CoreDataExamp... › CoreDataExamp... › Photographer › photos
Note the Data Model's recognition of the "inverse" of this Relationship.
Photo
Attributes
photoURL
subtitle
thumbnailData
thumbnailURL
title
uploadDate
Relationships
whoTook
Photographer
Attributes
name
Relationships
photos
The type of this Relationship in our Objective-C code will be NSManagedObject (or a subclass thereof).
The type of this Relationship in our Objective-C code will be NSSet (since it is a "to many" Relationship).
Relationship
Name photos
Destination Photo
Inverse whoTook
Properties Transient Optional
Arranged Ordered
Plural To-Many Relationship
Count Optional Minimum
Unlimited Maximum
Delete Rule Nullify
Advanced Index in Spotlight
Store in External Record File
User Info
Key Value
Versioning
Hash Modifier Version Hash Modifier
Renaming ID Renaming Identifier
Relationship Sync
Synchronization Enabled
Identity Property
Exclude From Change Alert
Outline Style
Add Entity
Add Attribute
Editor Style
Stanford CS193p
Fall 2011

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.
     (then your `AppDelegate` will have a `managedObjectContext` `@property`).

  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

```
UIManagedDocument *document = [[UIManagedDocument] initWithFileURL:(URL *)url];
```

UIManagedDocument

- But you must open/create the document to use it

  Check to see if it exists: `[[NSFileManager defaultManager] fileExistsAtPath:[url path]]`

  If it does, open the document …

  ```
  - (void)openWithCompletionHandler:(void (^)(BOOL success))completionHandler;
  ```

  If it does not, create it using …

  ```
  - (void)saveToURL:(NSURL *)url
   forSaveOperation:(UIDocumentSaveOperation)operation
   competionHandler:(void (^)(BOOL success))completionHandler;
  ```

- What is that `completionHander`?

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

  That's needed because the open/save is asynchronous!

  Do not ignore this fact!

UIManagedDocument

- Example

```
self.document = [[UIManagedDocument] 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 its `documentState` when you do …

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

- Other `documentState`s

  `UIDocumentStateClosed` (not opened or file does not exist yet)
  `UIDocumentStateSavingError` (success will be NO)
  `UIDocumentStateEditingDisabled` (temporarily unless failed to revert to saved)
  `UIDocumentStateInConflict` (e.g., because some other device changed it via iCloud)

- The `documentState` is often "observed"

  So it's about time we talked about using `NSNotification`s to observe other objects …

NSNotification

- NSNotificationCenter

  Get the default notification center via `[NSNotificationCenter defaultCenter]`

  Then send it the following message if you want to observe another object:

```
- (void)addObserver:(id)observer      // you (the object to get notified)
          selector:(SEL)methodToSendIfSomethingHappens
              name:(NSString *)name // what you're observing (a constant somewhere)
            object:(id)sender;      // whose changes you're interested in (nil is anyone's)
```

- You will then be notified when the named event happens

```
- (void)methodToSendIfSomethingHappens:(NSNotification *)notification
{
    notification.name     // the name passed above
    notification.object   // the object sending you the notification
    notification.userInfo // notification-specific information about what happened
}
```

NSNotification

- Example

```
NSNotificationCenter *center = [NSNotificationCenter defaultCenter];
```

Watching for changes in a document's state …

```
[center addObserver:self
           selector:@selector(documentChanged:)
               name:UIDocumentStateChangedNotification
             object:self.document];
```

Don't forget to remove yourself when you're done watching.

```
[center removeObserver:self];
```

or

```
[center removeObserver:self name:UIDocumentStateChangedNotification object:self.document];
```

Failure to remove yourself can sometimes result in crashers.

This is because the `NSNotificationCenter` keeps an "unsafe unretained" pointer to you.

NSNotification

- Another Example

  Watching for changes in a CoreData database (made via a given NSManagedObjectContext) …

```
- (void)viewDidAppear:(BOOL)animated
{
    [super viewDidAppear:animated];
    [center addObserver:self
               selector:@selector(contextChanged:)
                   name:NSManagedObjectContextObjectsDidChangeNotification
                 object:self.document.managedObjectContext];
}
- (void)viewWillDisappear:(BOOL)animated
{
    [center removeObserver:self
                      name:NSManagedObjectContextObjectsDidChangeNotification
                    object:self.document.managedObjectContext];
    [super viewWillDisappear:animated];
}
```

  There's also an NSManagedObjectContextDidSaveNotification.

NSNotification

- NSManagedObjectContextObjectsDidChangeNotification

  or 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
}
```

- Other things to observe

  Look in the documentation for various classes in iOS.

  They will document any notifications they will send out.

  You can post your own notifications too (see NSNotificationCenter documentation).

  Don't abuse this mechanism!

  Don't use it to essentially get "global variables" in your application.

UIManagedDocument

- Okay, back to UIManagedDocument …
- Saving a document (like creating or opening) is also asynchronous

  Documents are auto-saved, but you can explicitly save as well.

  You use the same method as when creating, but with a different "save operation."

```
[self.document saveToURL:self.document.fileURL
        forSaveOperation:UIDocumentSaveForOverwriting
       completionHandler:^(BOOL success) {
    if (!success) NSLog(@"failed to save document %@", self.document.localizedName);
}];
// the document is not saved at this point in the code (only once the block above executes)
```

  Note the two UIManagedDocument properties used above:

```
@property (nonatomic, strong) NSURL *fileURL; // specified originally in initWithFileURL:
@property (readonly) NSString *localizedName; // only valid once associated with a file
```

UIManagedDocument

- Closing a document is also asynchronous

  The document will be closed if there are no `strong` pointers left to the `UIManagedDocument`.

  But you can close it explicitly as well.

```
[self.document closeWithCompletionHandler:^(BOOL success) {
    if (!success) NSLog(@“failed to close document %@”, self.document.localizedName);
}];
// the document is not closed at this point in the code (only once the block above executes)
```

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? Not so rare. Just need to know when to refetch.

  For your homework, we recommend not doing this
  (i.e. we recommend only having one `UIManagedDocument` instance per actual document).
  This will require you to have a bit of global API, but we'll forgive it this time :).
  Hint #1 on the homework assignment will suggest an API to do this.

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.
- Inserting objects into the database

```
NSManagedObject *photo =
    [NSEntityDescription insertNewObjectForEntityForName:@“Photo”
                                  inManagedObjectContext:(NSManagedObjectContext *)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
  (the model that we just graphically built in Xcode)
  All the Attributes of a newly-inserted object will be nil
  (unless you specify a default value in Xcode)

Core Data

- How to access Attributes in an `NSManagedObject` instance

  You can access them using the following two `NSKeyValueObserving` 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"`

- 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).

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

  Non-"to-many" relationships are `NSManagedObject`s.

  Binary data values are `NSData` objects.

  Date values are `NSDate` objects.

Core Data

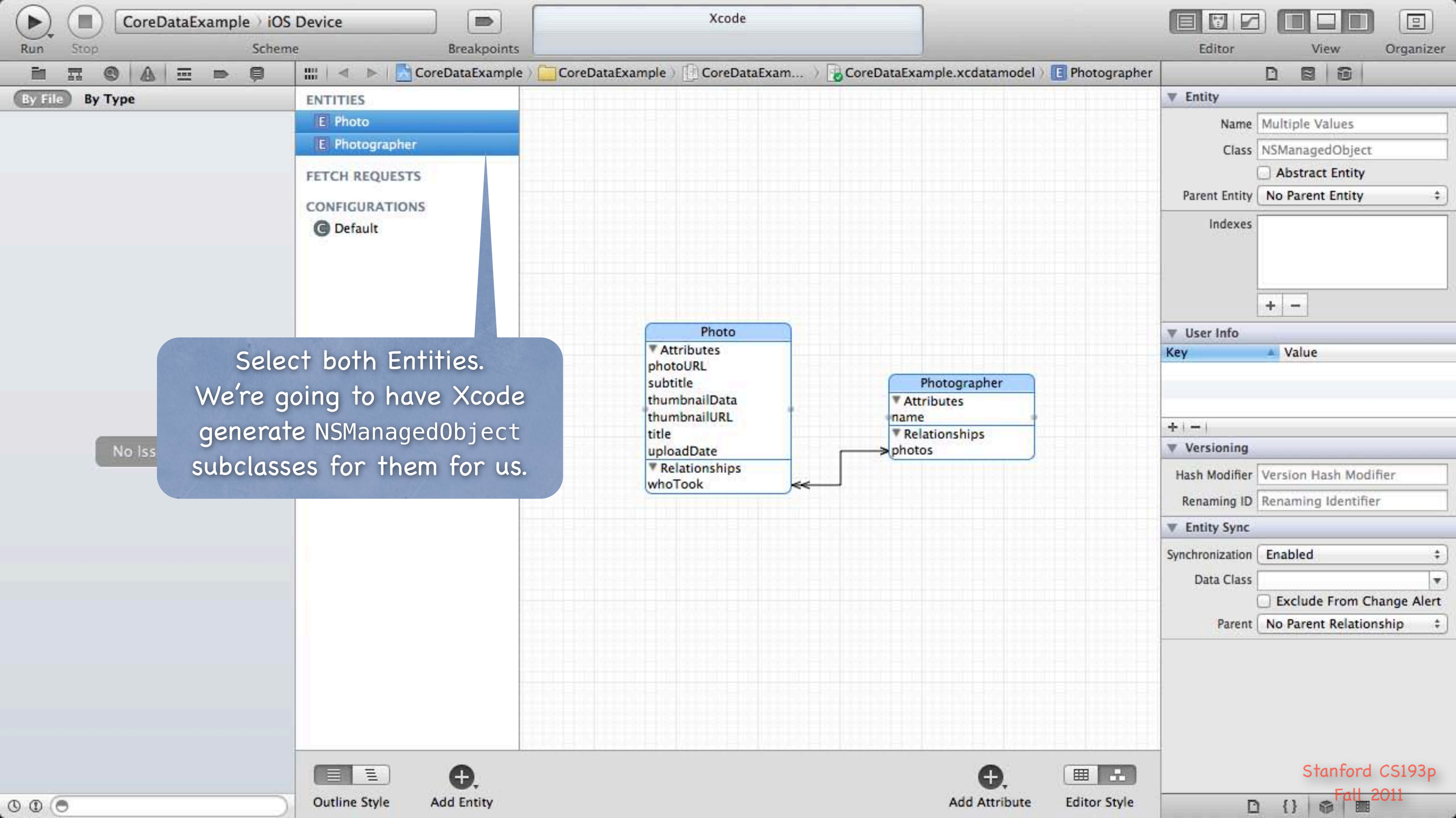
- Changes (writes) only happen in memory, until you save

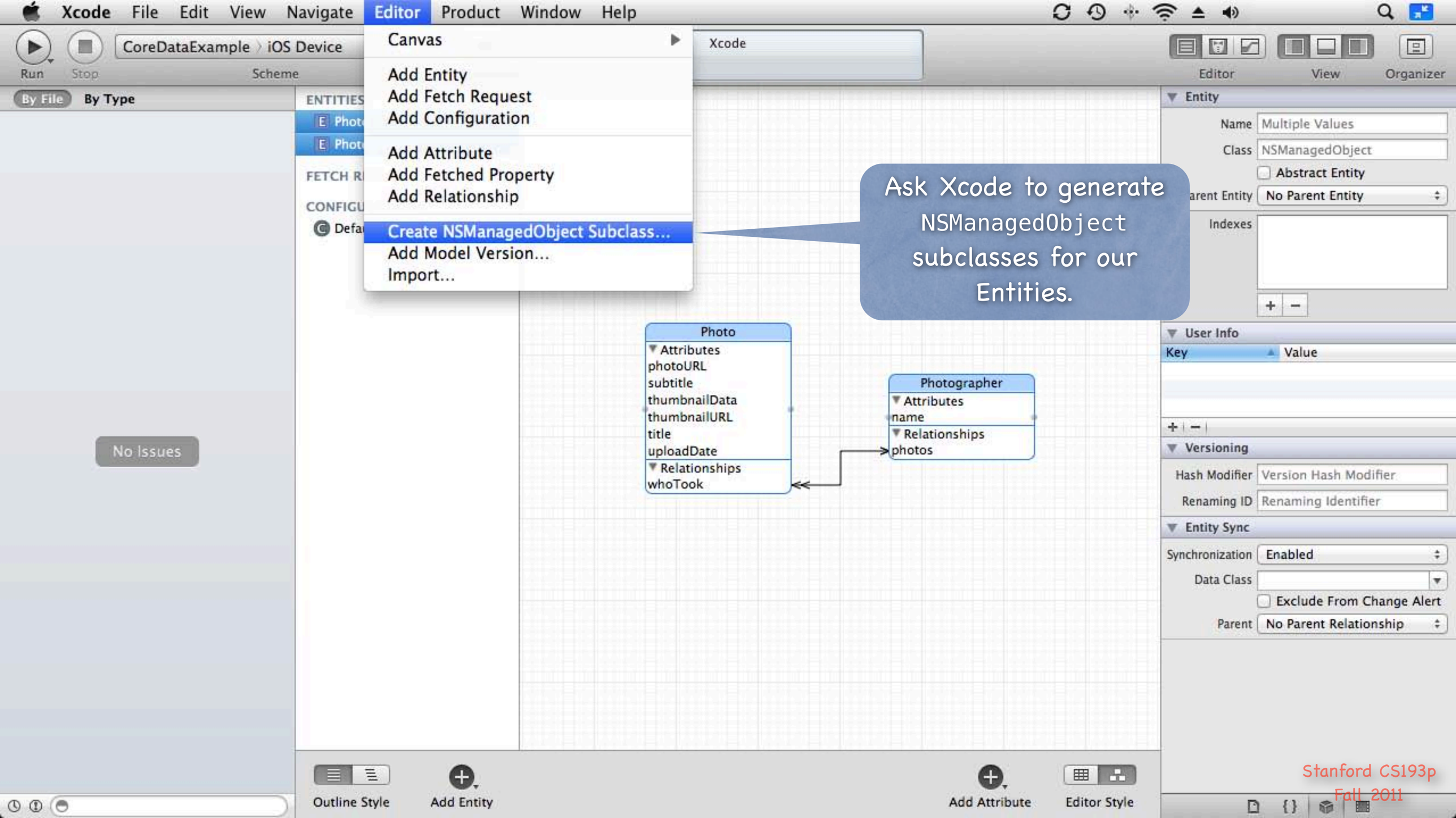
  Yes, `UIManagedDocument` auto-saves.

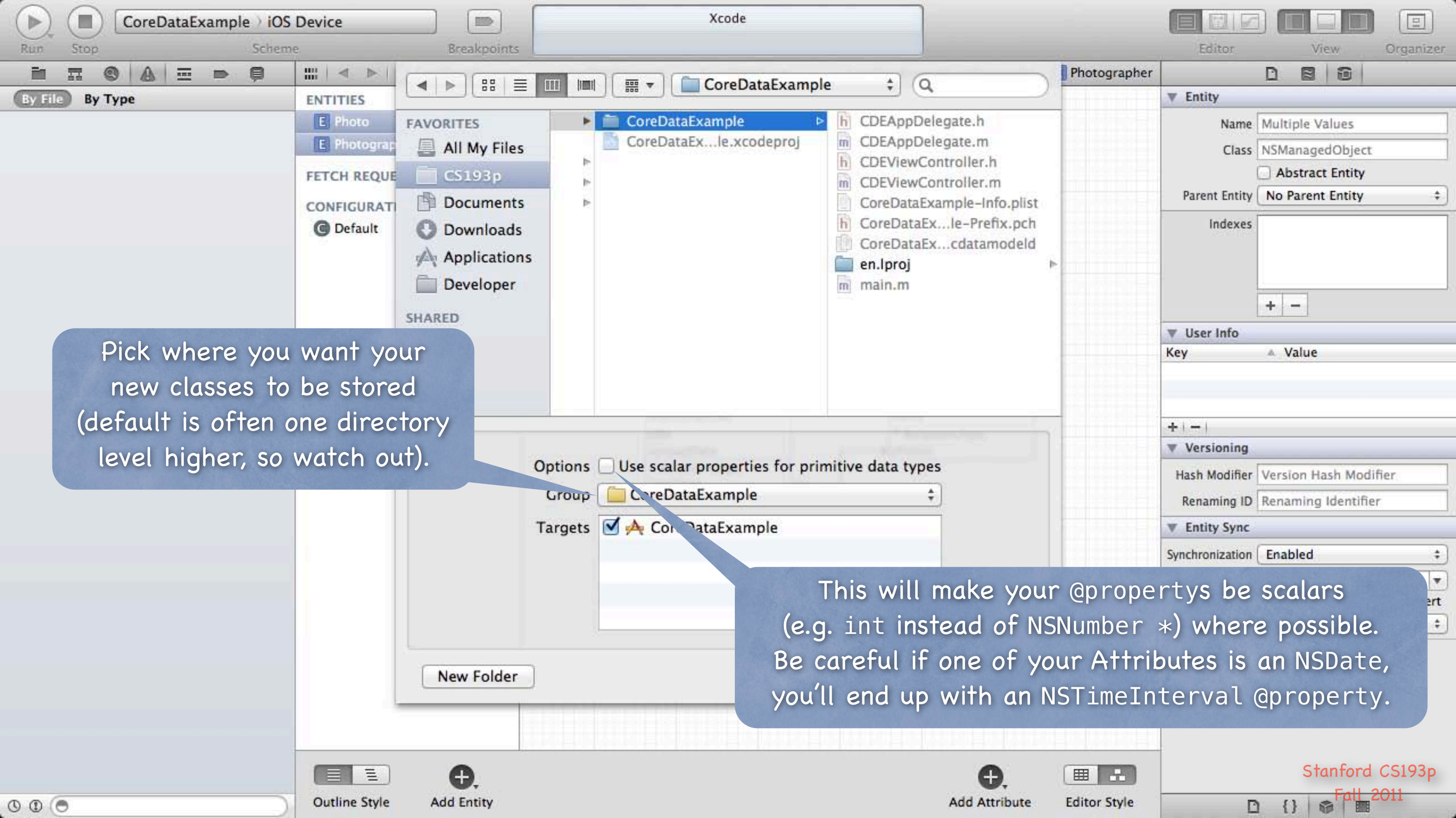
  But explicitly saving when a batch of changes is made is good practice.

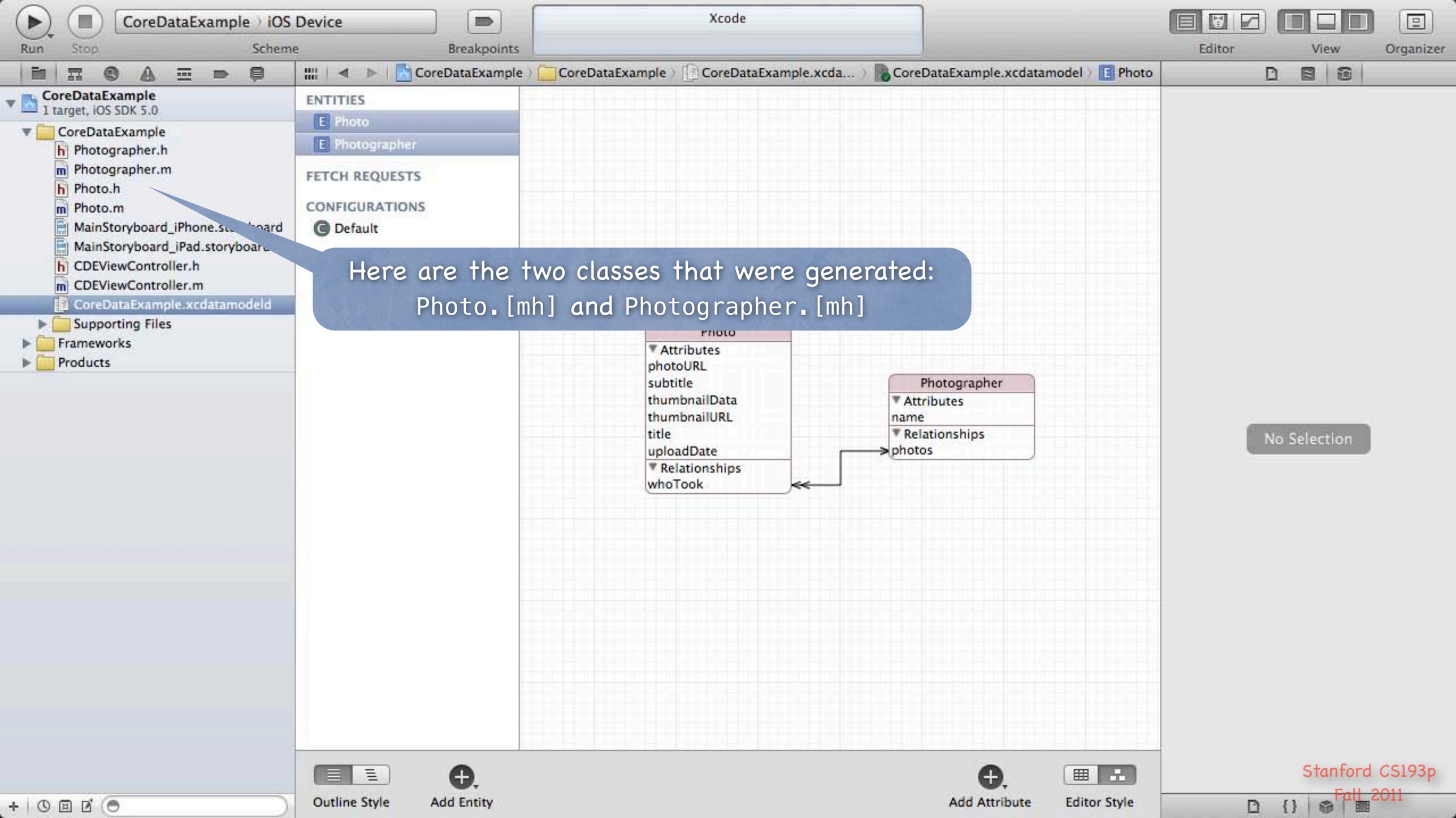
Core Data

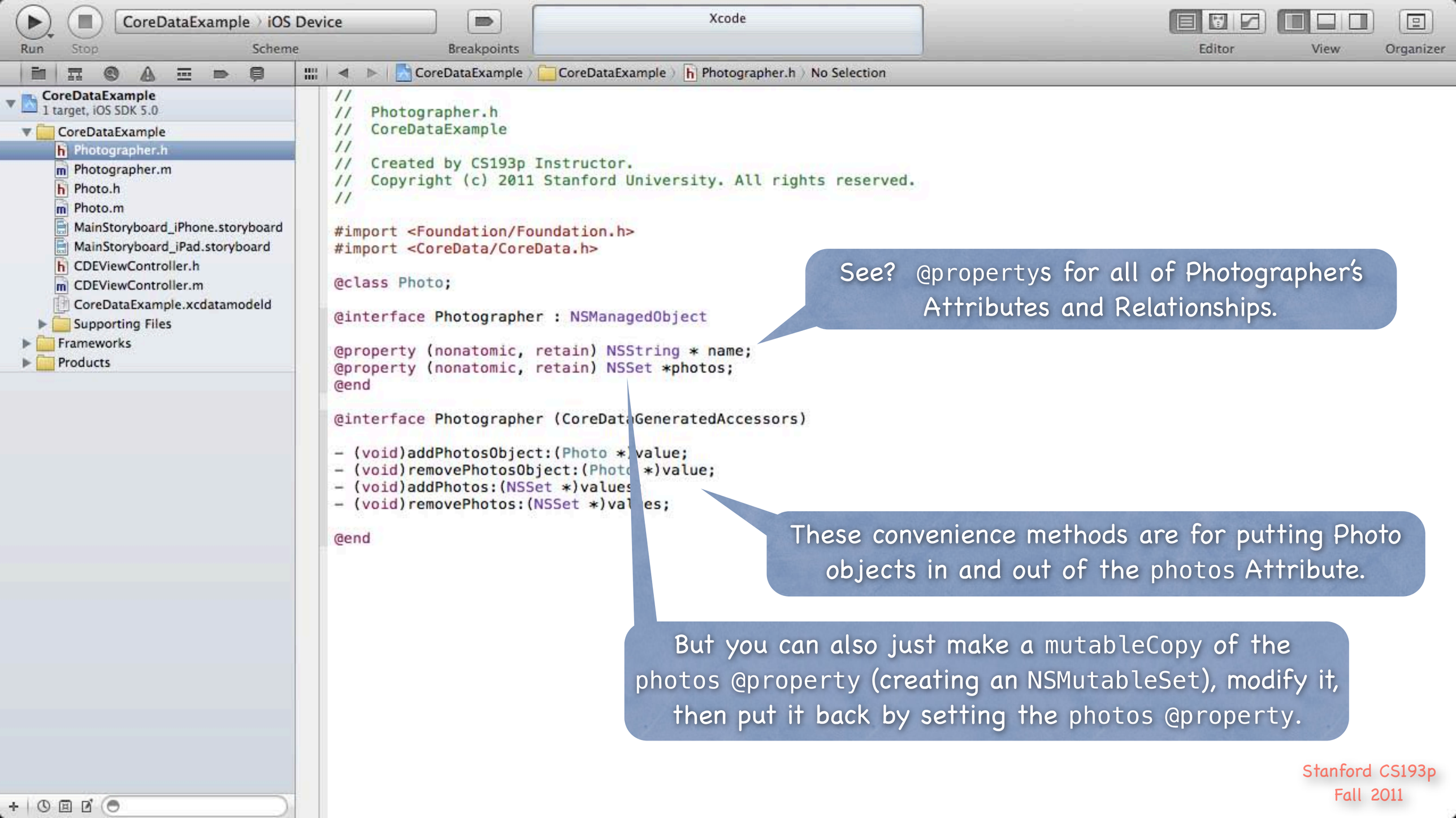
- But calling `valueForKey:`/`setValueForKey:` is pretty messy
  There's no type-checking.
  And you have a lot of literal strings in your code (e.g. `@"thumbnailURL"`)
- What we really want is to set/get using `@property`s!
- No problem … we just create a subclass of `NSManagedObject`
  The subclass 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!).

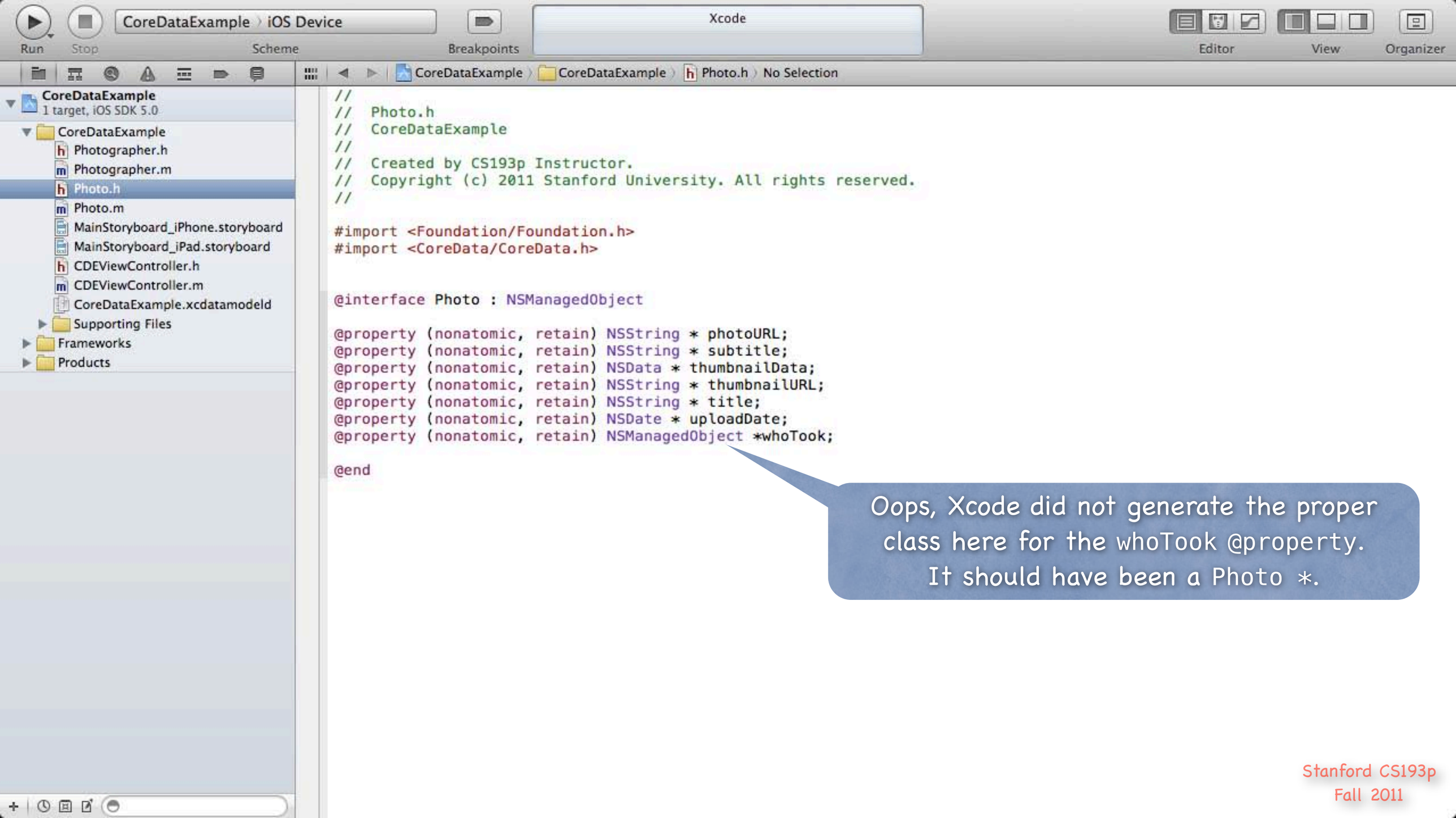

Run
Stop
CoreDataExample › iOS Device
Scheme
Breakpoints
Xcode
Editor
View
Organizer
By File
By Type
ENTITIES
Photo
Photographer
FETCH REQUESTS
CONFIGURATIONS
Default
Select both Entities.
We're going to have Xcode
generate NSManagedObject
subclasses for them for us.
Photo
Attributes
photoURL
subtitle
thumbnailData
thumbnailURL
title
uploadDate
Relationships
whoTook
Photographer
Attributes
name
Relationships
photos
Entity
Name
Multiple Values
Class
NSManagedObject
Abstract Entity
Parent Entity
No Parent Entity
Indexes
User Info
Key
Value
Versioning
Hash Modifier
Version Hash Modifier
Renaming ID
Renaming Identifier
Entity Sync
Synchronization
Enabled
Data Class
Exclude From Change Alert
Parent
No Parent Relationship
Outline Style
Add Entity
Add Attribute
Editor Style
Stanford CS193p
Fall 2011

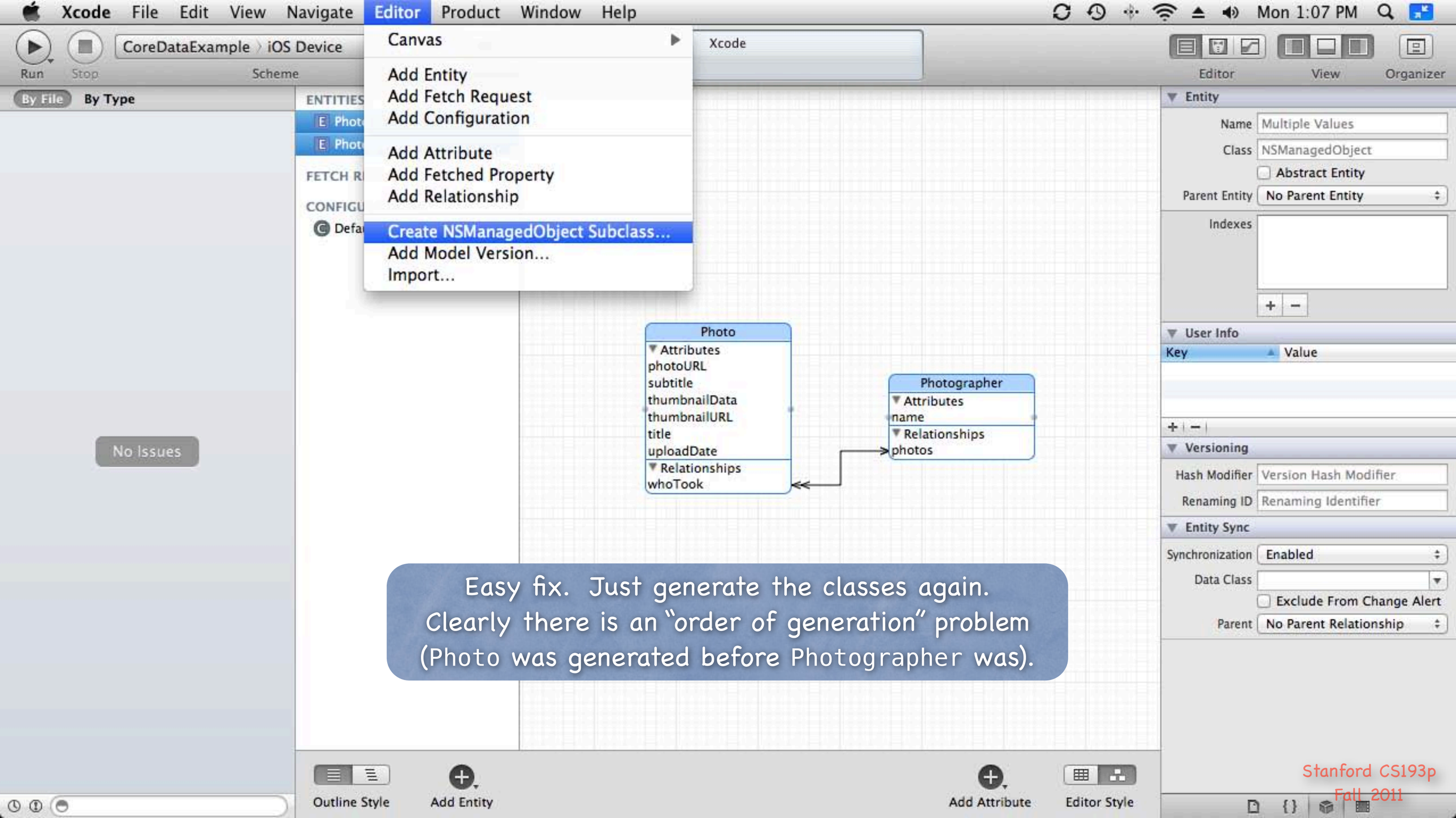

Xcode File Edit View Navigate Editor Product Window Help
Canvas
Add Entity
Add Fetch Request
Add Configuration
Add Attribute
Add Fetched Property
Add Relationship
Create NSManagedObject Subclass...
Add Model Version...
Import...
Run Stop Scheme
CoreDataExample 〉 iOS Device
Xcode
Editor View Organizer
By File By Type
No Issues
ENTITIES
FETCH R
CONFIGU
Ask Xcode to generate NSManagedObject subclasses for our Entities.
Photo
Attributes
photoURL
subtitle
thumbnailData
thumbnailURL
title
uploadDate
Relationships
whoTook
Photographer
Attributes
name
Relationships
photos
Entity
Name Multiple Values
Class NSManagedObject
Abstract Entity
Parent Entity No Parent Entity
Indexes
User Info
Key Value
Versioning
Hash Modifier Version Hash Modifier
Renaming ID Renaming Identifier
Entity Sync
Synchronization Enabled
Data Class
Exclude From Change Alert
Parent No Parent Relationship
Outline Style
Add Entity
Add Attribute
Editor Style
Stanford CS193p
Fall 2011

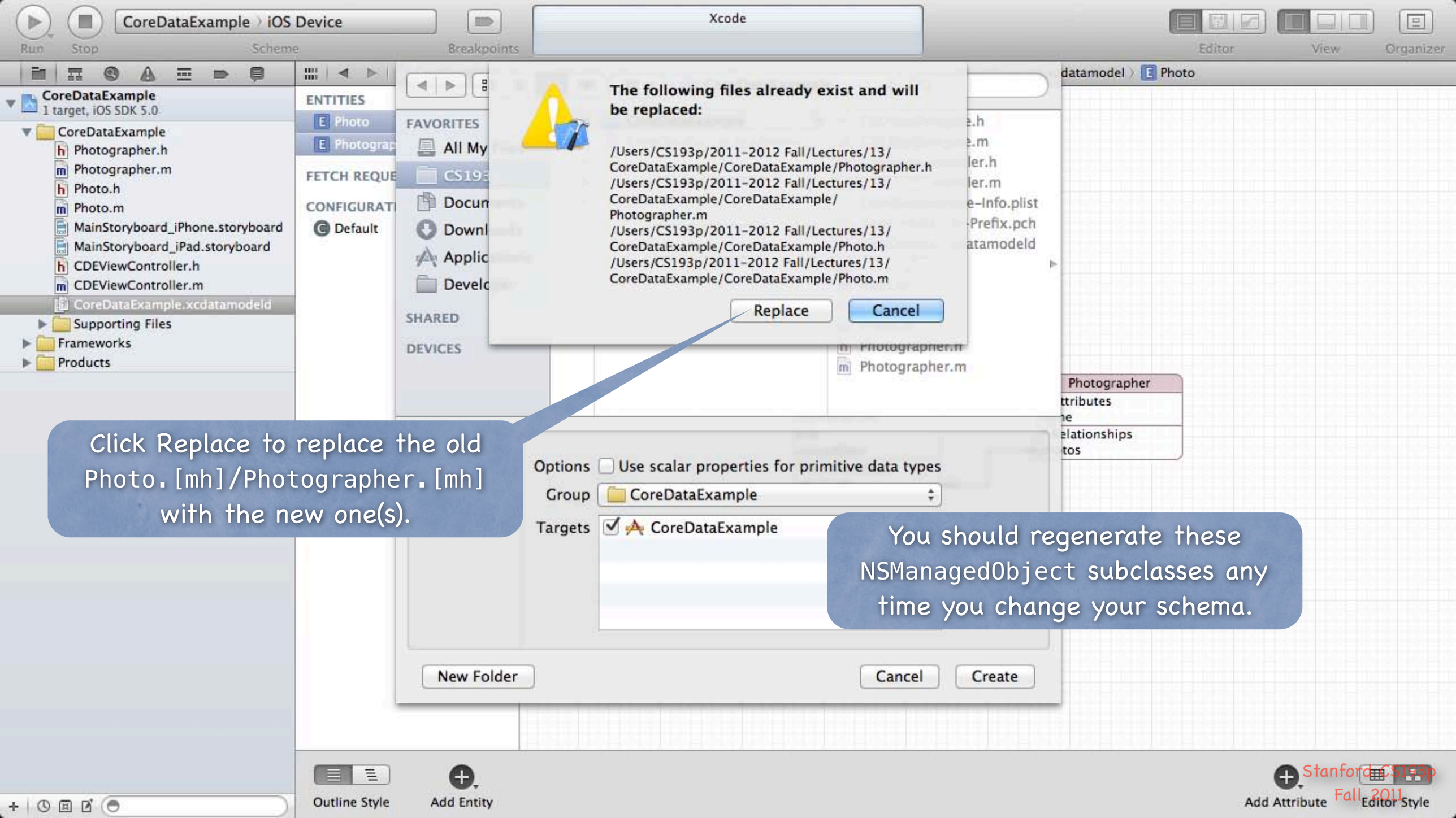

Run
Stop
CoreDataExample › iOS Device
Scheme
Breakpoints
Xcode
Editor
View
Organizer
By File
By Type
ENTITIES
Photo
FETCH REQUE
CONFIGURAT
Default
CoreDataExample
FAVORITES
All My Files
CS193p
Documents
Downloads
Applications
Developer
SHARED
CoreDataExample
CoreDataEx...le.xcodeproj
CDEAppDelegate.h
CDEAppDelegate.m
CDEViewController.h
CDEViewController.m
CoreDataExample-Info.plist
CoreDataEx...le-Prefix.pch
CoreDataEx...cdatamodeld
en.lproj
main.m
Photographer
Entity
Name
Multiple Values
Class
NSManagedObject
Abstract Entity
Parent Entity
No Parent Entity
Indexes
User Info
Key
Value
Versioning
Hash Modifier
Version Hash Modifier
Renaming ID
Renaming Identifier
Entity Sync
Synchronization
Enabled
Options
Use scalar properties for primitive data types
Group
Targets
New Folder
Pick where you want your new classes to be stored (default is often one directory level higher, so watch out).
This will make your @propertys 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.
Outline Style
Add Entity
Add Attribute
Editor Style
Stanford CS193p
Fall 2011

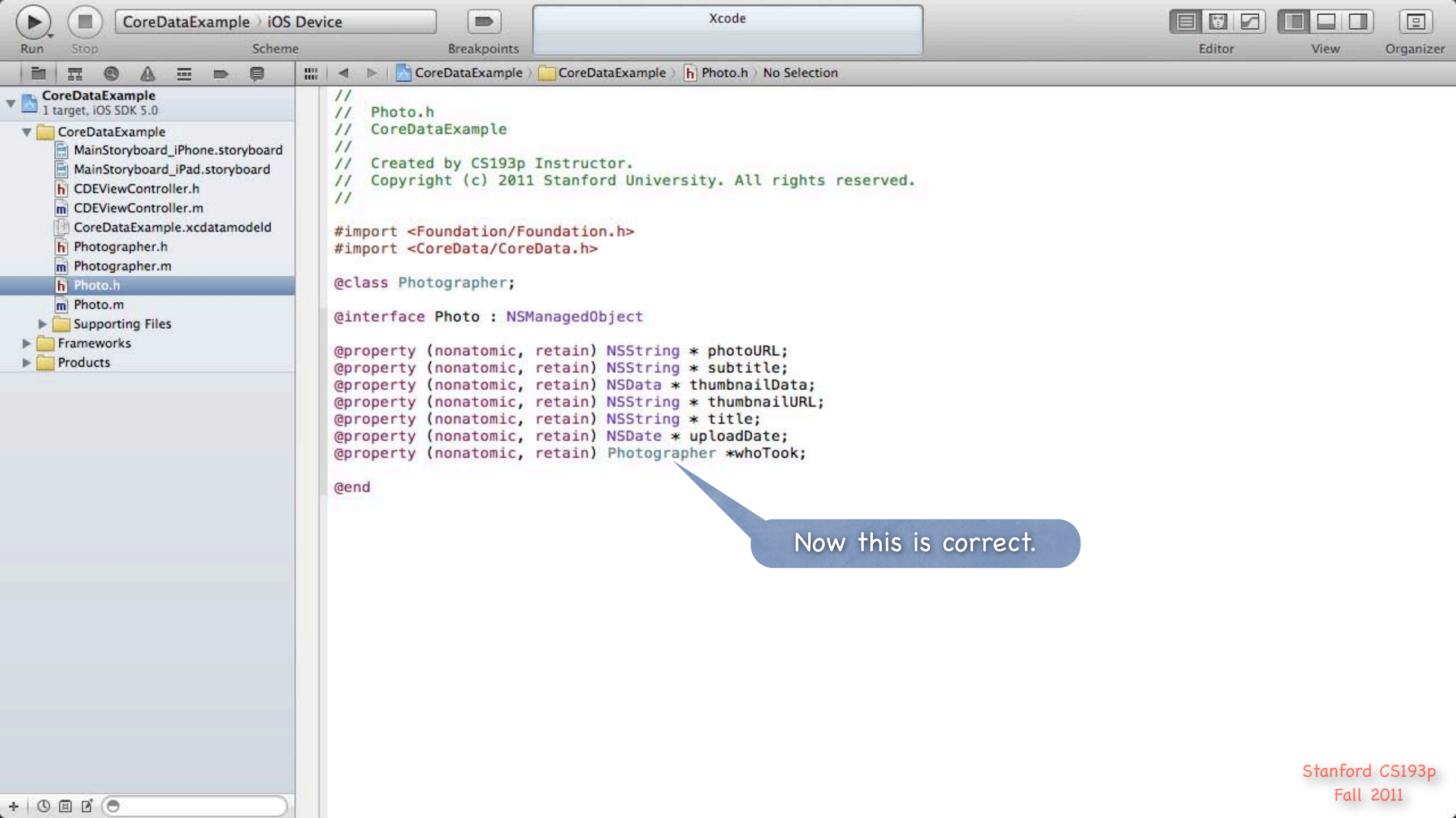

Run
Stop
CoreDataExample › iOS Device
Scheme
Breakpoints
Xcode
Editor
View
Organizer
CoreDataExample › CoreDataExample › CoreDataExample.xcda... › CoreDataExample.xcdatamodel › Photo
CoreDataExample
1 target, iOS SDK 5.0
CoreDataExample
Photographer.h
Photographer.m
Photo.h
Photo.m
MainStoryboard_iPhone.storyboard
MainStoryboard_iPad.storyboard
CDEViewController.h
CDEViewController.m
CoreDataExample.xcdatamodeld
Supporting Files
Frameworks
Products
ENTITIES
Photo
Photographer
FETCH REQUESTS
CONFIGURATIONS
Default
Here are the two classes that were generated:
Photo.[mh] and Photographer.[mh]
Photo
Attributes
photoURL
subtitle
thumbnailData
thumbnailURL
title
uploadDate
Relationships
whoTook
Photographer
Attributes
name
Relationships
photos
No Selection
Outline Style
Add Entity
Add Attribute
Editor Style
Stanford CS193p
Fall 2011

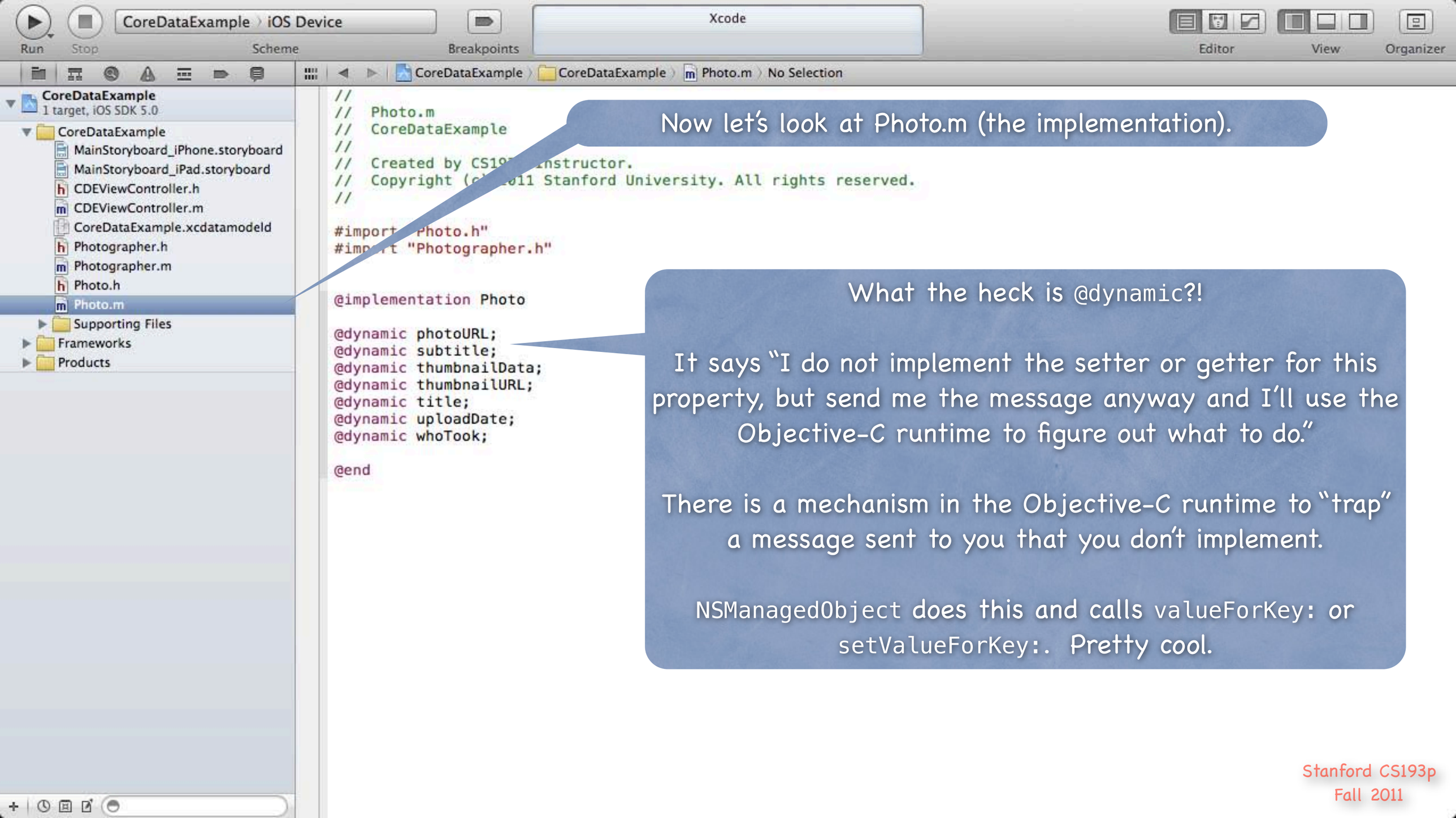

Run
Stop
CoreDataExample › iOS Device
Scheme
Breakpoints
Xcode
Editor
View
Organizer
CoreDataExample › CoreDataExample › Photographer.h › No Selection
CoreDataExample
1 target, iOS SDK 5.0
CoreDataExample
Photographer.h
Photographer.m
Photo.h
Photo.m
MainStoryboard_iPhone.storyboard
MainStoryboard_iPad.storyboard
CDEViewController.h
CDEViewController.m
CoreDataExample.xcdatamodeld
Supporting Files
Frameworks
Products
//
// Photographer.h
// CoreDataExample
//
// Created by CS193p Instructor.
// Copyright (c) 2011 Stanford University. All rights reserved.
//
#import <Foundation/Foundation.h>
#import <CoreData/CoreData.h>
@class Photo;
@interface Photographer : NSManagedObject
@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
See? @propertys for all of Photographer's Attributes and Relationships.
These convenience methods are for putting Photo objects in and out of the photos Attribute.
But you can also just make a mutableCopy of the photos @property (creating an NSMutableSet), modify it, then put it back by setting the photos @property.
Stanford CS193p
Fall 2011


Run
Stop
CoreDataExample › iOS Device
Scheme
Breakpoints
Xcode
Editor
View
Organizer
CoreDataExample › CoreDataExample › Photo.h › No Selection
CoreDataExample
1 target, iOS SDK 5.0
CoreDataExample
Photographer.h
Photographer.m
Photo.h
Photo.m
MainStoryboard_iPhone.storyboard
MainStoryboard_iPad.storyboard
CDEViewController.h
CDEViewController.m
CoreDataExample.xcdatamodeld
Supporting Files
Frameworks
Products
//
// Photo.h
// CoreDataExample
//
// Created by CS193p Instructor.
// Copyright (c) 2011 Stanford University. All rights reserved.
//
#import <Foundation/Foundation.h>
#import <CoreData/CoreData.h>
@interface Photo : NSManagedObject
@property (nonatomic, retain) NSString * photoURL;
@property (nonatomic, retain) NSString * subtitle;
@property (nonatomic, retain) NSData * thumbnailData;
@property (nonatomic, retain) NSString * thumbnailURL;
@property (nonatomic, retain) NSString * title;
@property (nonatomic, retain) NSDate * uploadDate;
@property (nonatomic, retain) NSManagedObject *whoTook;
@end
Oops, Xcode did not generate the proper class here for the whoTook @property. It should have been a Photo *.


Xcode File Edit View Navigate Editor Product Window Help
Mon 1:07 PM
Canvas
Add Entity
Add Fetch Request
Add Configuration
Add Attribute
Add Fetched Property
Add Relationship
Create NSManagedObject Subclass...
Add Model Version...
Import...
CoreDataExample iOS Device
Run Stop Scheme
Editor View Organizer
By File By Type
No Issues
ENTITIES
FETCH R
CONFIGU
Photo
Attributes
photoURL
subtitle
thumbnailData
thumbnailURL
title
uploadDate
Relationships
whoTook
Photographer
Attributes
name
Relationships
photos
Entity
Name Multiple Values
Class NSManagedObject
Abstract Entity
Parent Entity No Parent Entity
Indexes
User Info
Key Value
Versioning
Hash Modifier Version Hash Modifier
Renaming ID Renaming Identifier
Entity Sync
Synchronization Enabled
Data Class
Exclude From Change Alert
Parent No Parent Relationship
Easy fix. Just generate the classes again.
Clearly there is an "order of generation" problem
(Photo was generated before Photographer was).
Outline Style
Add Entity
Add Attribute
Editor Style
Stanford CS193p
Fall 2011


Run
Stop
CoreDataExample › iOS Device
Scheme
Breakpoints
Xcode
Editor
View
Organizer
CoreDataExample
1 target, iOS SDK 5.0
CoreDataExample
Photographer.h
Photographer.m
Photo.h
Photo.m
MainStoryboard_iPhone.storyboard
MainStoryboard_iPad.storyboard
CDEViewController.h
CDEViewController.m
CoreDataExample.xcdatamodeld
Supporting Files
Frameworks
Products
ENTITIES
Photo
FETCH REQUE
CONFIGURAT
Default
FAVORITES
SHARED
DEVICES
The following files already exist and will be replaced:
/Users/CS193p/2011-2012 Fall/Lectures/13/CoreDataExample/CoreDataExample/Photographer.h
/Users/CS193p/2011-2012 Fall/Lectures/13/CoreDataExample/CoreDataExample/Photographer.m
/Users/CS193p/2011-2012 Fall/Lectures/13/CoreDataExample/CoreDataExample/Photo.h
/Users/CS193p/2011-2012 Fall/Lectures/13/CoreDataExample/CoreDataExample/Photo.m
Replace
Cancel
datamodel › Photo
Photographer.m
Photographer
Options
Use scalar properties for primitive data types
Group
CoreDataExample
Targets
CoreDataExample
New Folder
Cancel
Create
Click Replace to replace the old Photo.[mh]/Photographer.[mh] with the new one(s).
You should regenerate these NSManagedObject subclasses any time you change your schema.
Outline Style
Add Entity
Add Attribute
Editor Style
Stanford CS193p Fall 2011


Run
Stop
CoreDataExample › iOS Device
Scheme
Breakpoints
Xcode
Editor
View
Organizer
CoreDataExample › CoreDataExample › Photo.h › No Selection
CoreDataExample
1 target, iOS SDK 5.0
CoreDataExample
MainStoryboard_iPhone.storyboard
MainStoryboard_iPad.storyboard
CDEViewController.h
CDEViewController.m
CoreDataExample.xcdatamodeld
Photographer.h
Photographer.m
Photo.h
Photo.m
Supporting Files
Frameworks
Products
//
// Photo.h
// CoreDataExample
//
// Created by CS193p Instructor.
// Copyright (c) 2011 Stanford University. All rights reserved.
//
#import <Foundation/Foundation.h>
#import <CoreData/CoreData.h>
@class Photographer;
@interface Photo : NSManagedObject
@property (nonatomic, retain) NSString * photoURL;
@property (nonatomic, retain) NSString * subtitle;
@property (nonatomic, retain) NSData * thumbnailData;
@property (nonatomic, retain) NSString * thumbnailURL;
@property (nonatomic, retain) NSString * title;
@property (nonatomic, retain) NSDate * uploadDate;
@property (nonatomic, retain) Photographer *whoTook;
@end
Now this is correct.


Run
Stop
CoreDataExample › iOS Device
Scheme
Breakpoints
Xcode
Editor
View
Organizer
CoreDataExample › CoreDataExample › Photo.m › No Selection
CoreDataExample
1 target, iOS SDK 5.0
CoreDataExample
MainStoryboard_iPhone.storyboard
MainStoryboard_iPad.storyboard
CDEViewController.h
CDEViewController.m
CoreDataExample.xcdatamodeld
Photographer.h
Photographer.m
Photo.h
Photo.m
Supporting Files
Frameworks
Products
//
// Photo.m
// CoreDataExample
//
// Created by CS193 Instructor.
// Copyright (c) 2011 Stanford University. All rights reserved.
//
#import "Photo.h"
#import "Photographer.h"
@implementation Photo
@dynamic photoURL;
@dynamic subtitle;
@dynamic thumbnailData;
@dynamic thumbnailURL;
@dynamic title;
@dynamic uploadDate;
@dynamic whoTook;
@end
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.
NSManagedObject does this and calls valueForKey: or setValueForKey:. Pretty cool.

Core Data

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

```
Photo *photo = [NSEntityDescription insertNewObjectForEntityForName:@“Photo” inManagedObj…];
NSString *myThumbnail = photo.thumbnailURL;
photo.thumbnailData = [FlickrFetcher urlForPhoto:photoDictionary format:FlickrPhotoFormat…];
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?

  Hmm, that's a problem.

  Because you might want to modify your schema and re-generate the subclasses!

  And it'd be really cool to be able to add code (very object-oriented).

  Especially code to create an object and set it up properly (and also tear one down, it turns out).

  Or maybe to derive new `@property`s based on ones in the database

  (e.g. a `UIImage` based on a URL in the database).

  Time for an aside about an Objective-C feature called "categories" ...

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. its `.m`).
- Examples
  `NSString`'s `drawAtPoint:withFont:` method.
  This method is added by UIKit (since it's a UI method) even though `NSString` is in Foundation.
  `NSIndexPath`'s `row` and `section` properties (used in `UITableView`-related code)
  are 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, so no `@synthesize` allowed in its implementation.

Categories

- Implementation

```
@implementation Photo (AddOn)
- (UIImage *)image  // image is not an attribute in the database, but photoURL is
{
    NSData *imageData = [NSData dataWithContentsOfURL:self.photoURL];
    return [UIImage imageWithData: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 `@property`s to an `NSManagedObject` subclass via categories to make accessing `BOOL` attributes (which are `NSNumber`s) cleaner.

Or we add `@property`s to convert `NSData`s to whatever the bits represent.

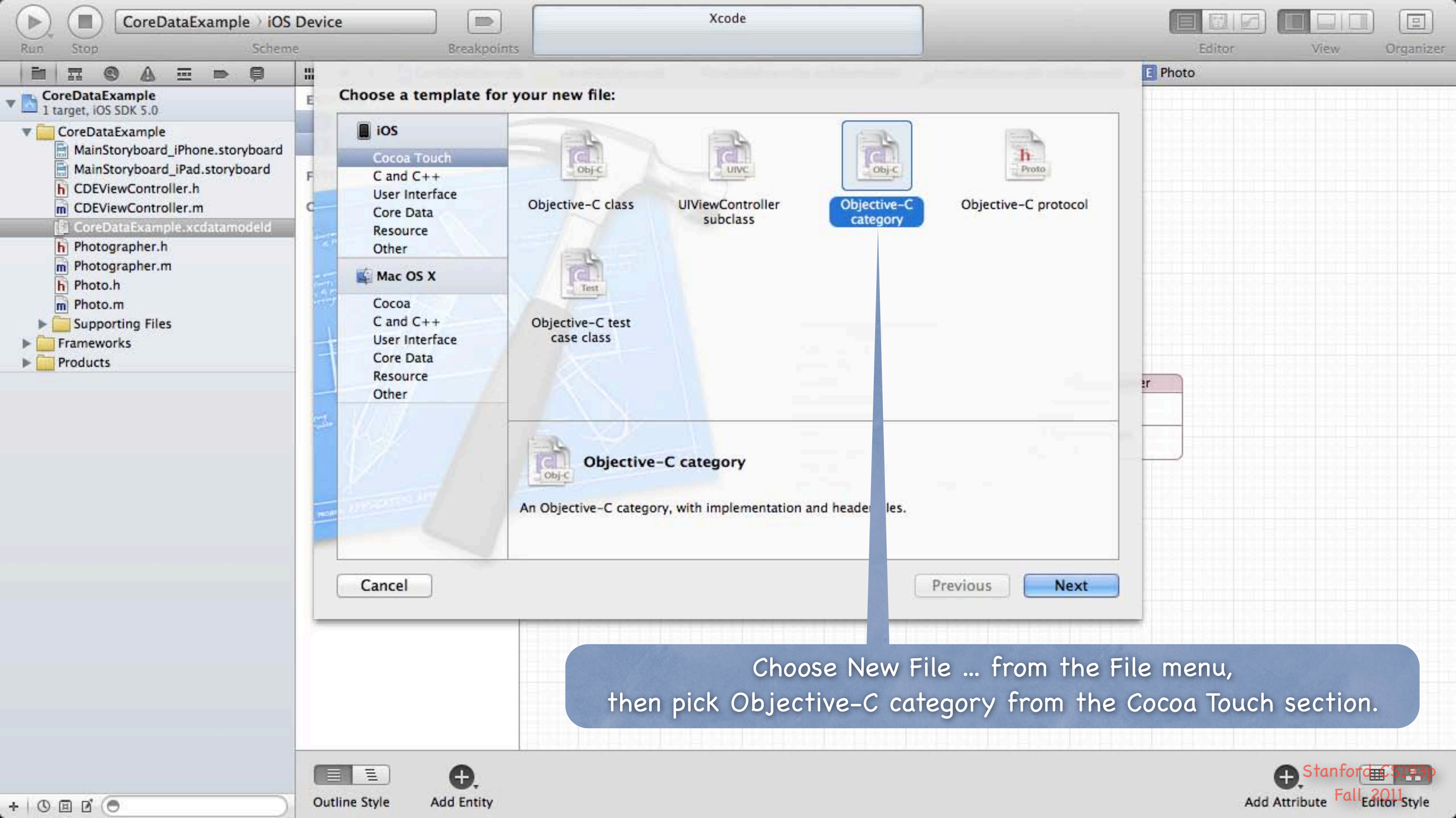
Any class can have a category added to it, but don't overuse/abuse this mechanism.

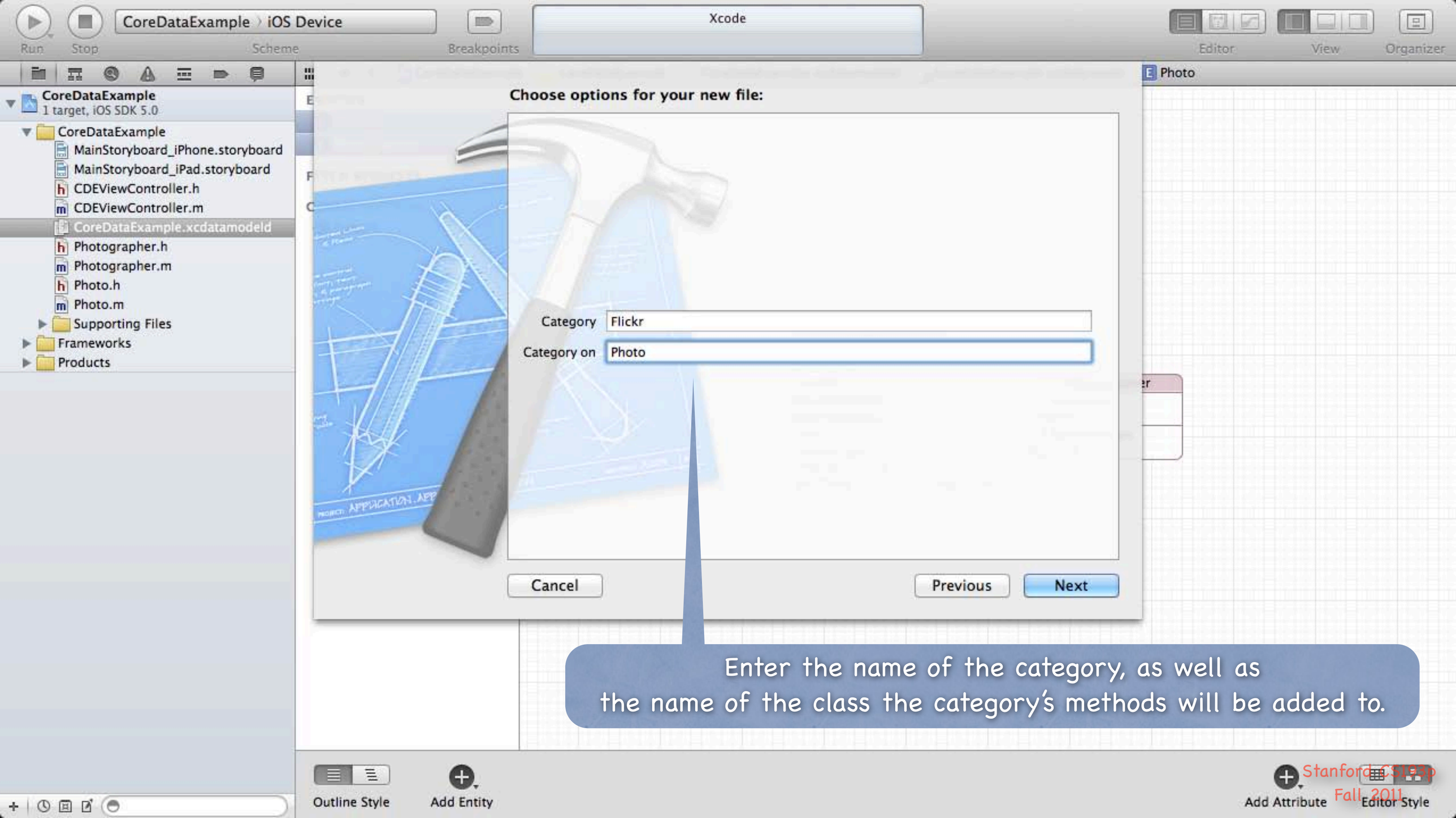
Categories

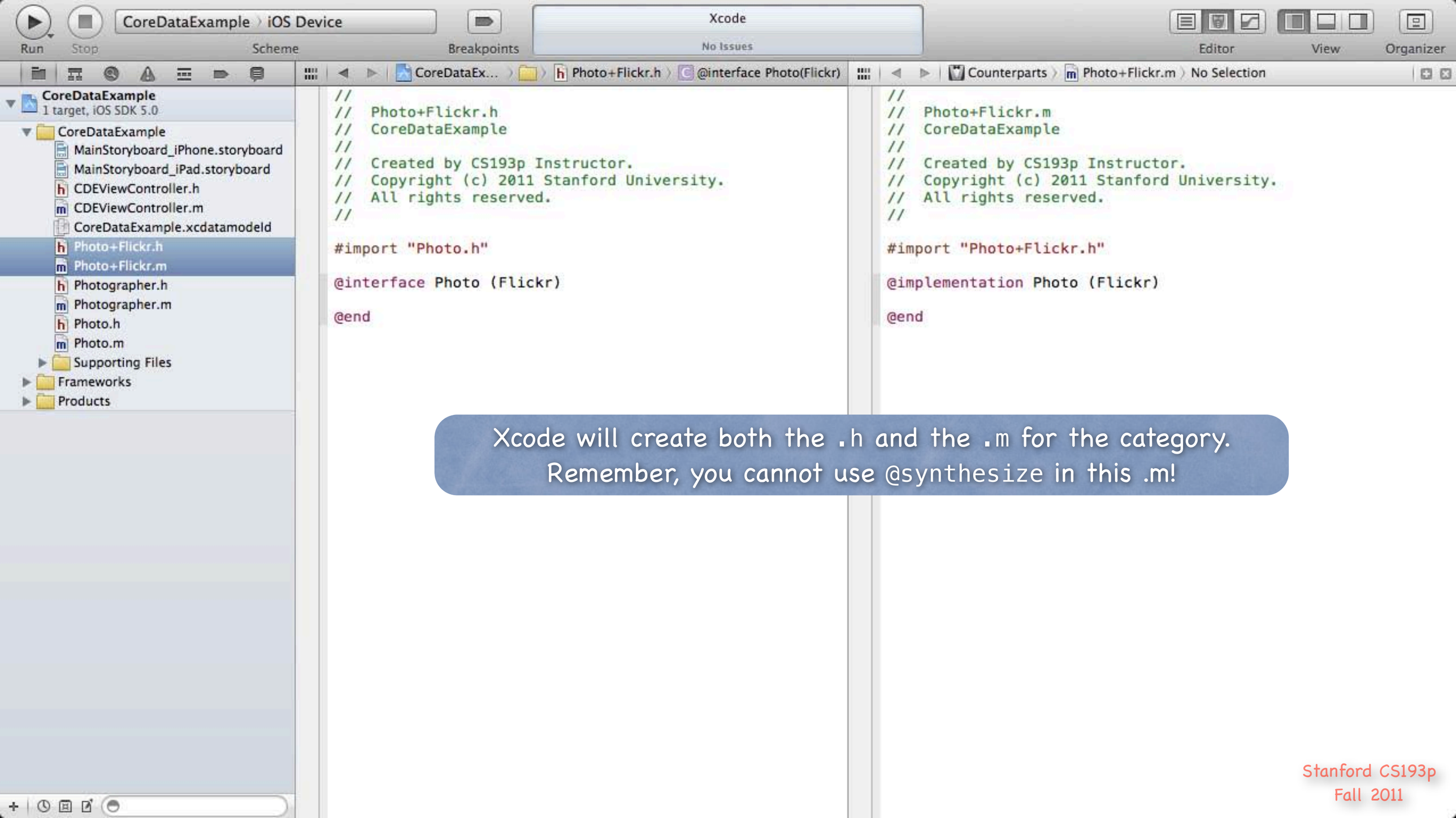
- Most common category on an NSManagedObject subclass?

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
```


Run
Stop
CoreDataExample › iOS Device
Scheme
Breakpoints
Xcode
Editor
View
Organizer
CoreDataExample
1 target, iOS SDK 5.0
CoreDataExample
MainStoryboard_iPhone.storyboard
MainStoryboard_iPad.storyboard
CDEViewController.h
CDEViewController.m
CoreDataExample.xcdatamodeld
Photographer.h
Photographer.m
Photo.h
Photo.m
Supporting Files
Frameworks
Products
Photo
Choose a template for your new file:
iOS
Cocoa Touch
C and C++
User Interface
Core Data
Resource
Other
Mac OS X
Cocoa
C and C++
User Interface
Core Data
Resource
Other
Objective-C class
UIViewController subclass
Objective-C category
Objective-C protocol
Objective-C test case class
Objective-C category
An Objective-C category, with implementation and header files.
Cancel
Previous
Next
Choose New File ... from the File menu,
then pick Objective-C category from the Cocoa Touch section.
Outline Style
Add Entity
Add Attribute
Editor Style
Stanford CS193p
Fall 2011


Run
Stop
CoreDataExample › iOS Device
Scheme
Breakpoints
Xcode
Editor
View
Organizer
CoreDataExample
1 target, iOS SDK 5.0
CoreDataExample
MainStoryboard_iPhone.storyboard
MainStoryboard_iPad.storyboard
CDEViewController.h
CDEViewController.m
CoreDataExample.xcdatamodeld
Photographer.h
Photographer.m
Photo.h
Photo.m
Supporting Files
Frameworks
Products
Photo
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.
Outline Style
Add Entity
Add Attribute
Editor Style
Stanford CS193p
Fall 2011



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

Deletion

- Deletion

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

  ```
  [self.document.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`

  Here 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
```

Core Data

- So far you can …

  Create objects in the database with `insertNewObjectForEntityForName:inManagedObjectContext:`.

  Get/set properties with `valueForKey:`/`setValueForKey:` or `@property`s 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. `NSPredicate` specifying which of those Entities to fetch (optional, default is all of them)
  3. `NSSortDescriptor`s to specify the order in which fetched objects are returned
  4. How many objects to fetch at a time and/or maximum to fetch (optional, all)

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 = [NSArray arrayWithObject: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 (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.

NSArrays are "ordered," so we have to 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:@"thumbnailURL"
                                  ascending:YES
                                   selector:@selector(localizedCaseInsensitiveCompare:)];
```

There's another version with no selector: argument (default is the method compare:).

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).

We give a list of these to the NSFetchRequest because sometimes we want to sort first by one key (e.g. last name), then, within that sort, sort by another (e.g. first name).

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 = [NSArray arrayWithObjects:predicate1, predicate2, nil];
NSPredicate *predicate = [NSCompoundPredicate andPredicateWithSubpredicates:array];
```

  This predicate is "predicate1 AND predicate2".

  "Or" predicate also available, of course.

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 …

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

- Executing the fetch

```
NSManagedObjectContext *moc = self.document.managedObjectContext;
NSError *error;
NSArray *photographers = [moc 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 array of `NSManagedObject`s (or subclasses thereof) if there were any matches.
  You can pass `NULL` for `error:` if you don't care why it fails.

Querying

- 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.

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

- Thursday

  More Core Data

- Friday Section

  Mike Ghaffary

  Director of Business Development at Yelp!

  Also co-founder of BarMax, the most expensive iPhone/iPad app on the AppStore

  Topic: Building Apps that People Want

  - Understanding Market Opportunity
  - Building a Prototype
  - Financing a Company or Team
  - Getting User Feedback
  - Distribution through the AppStore