Developing Applications for iOS

Lecture 10:
Core Data and Categories

Radu Ionescu
raducu.ionescu@gmail.com
Faculty of Mathematics and Computer Science
University of Bucharest

Content

- Core Data and Documents

  This is how you store something serious in iOS.

  Easy entry point into iCloud.
- `NSNotificationCenter`

  The little “radio station” we talked about in the first lecture.
- Objective-C Categories

  A way to add methods to a class without subclassing.

Core Data

- We are object-oriented programmers and we don't really like C APIs. We want to store our data using object-oriented programming.

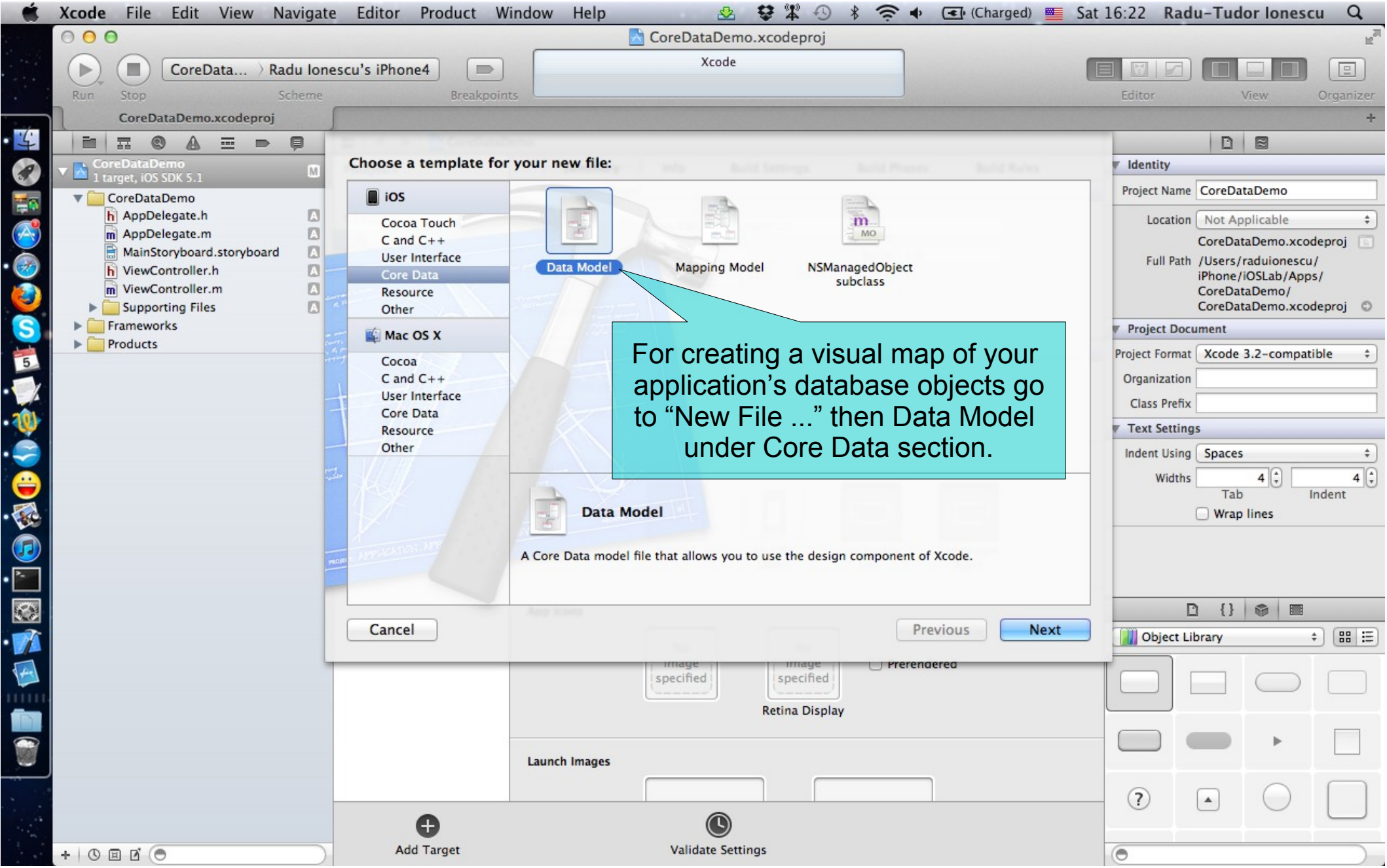
Welcome to Core Data

- This is an object-oriented database.
- It's a way of creating an object graph backed by a database (usually SQL).

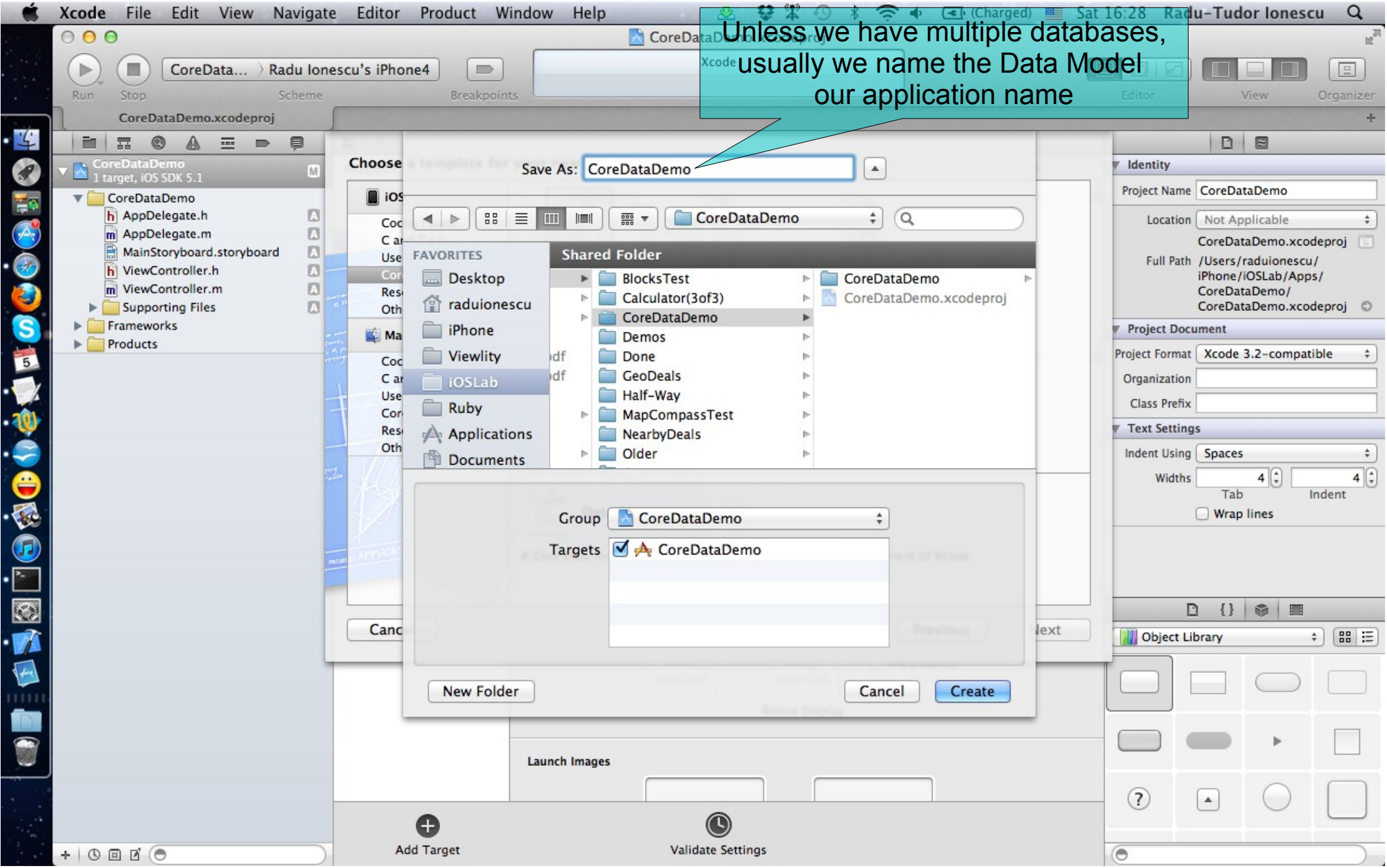
How does it work?

- 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.

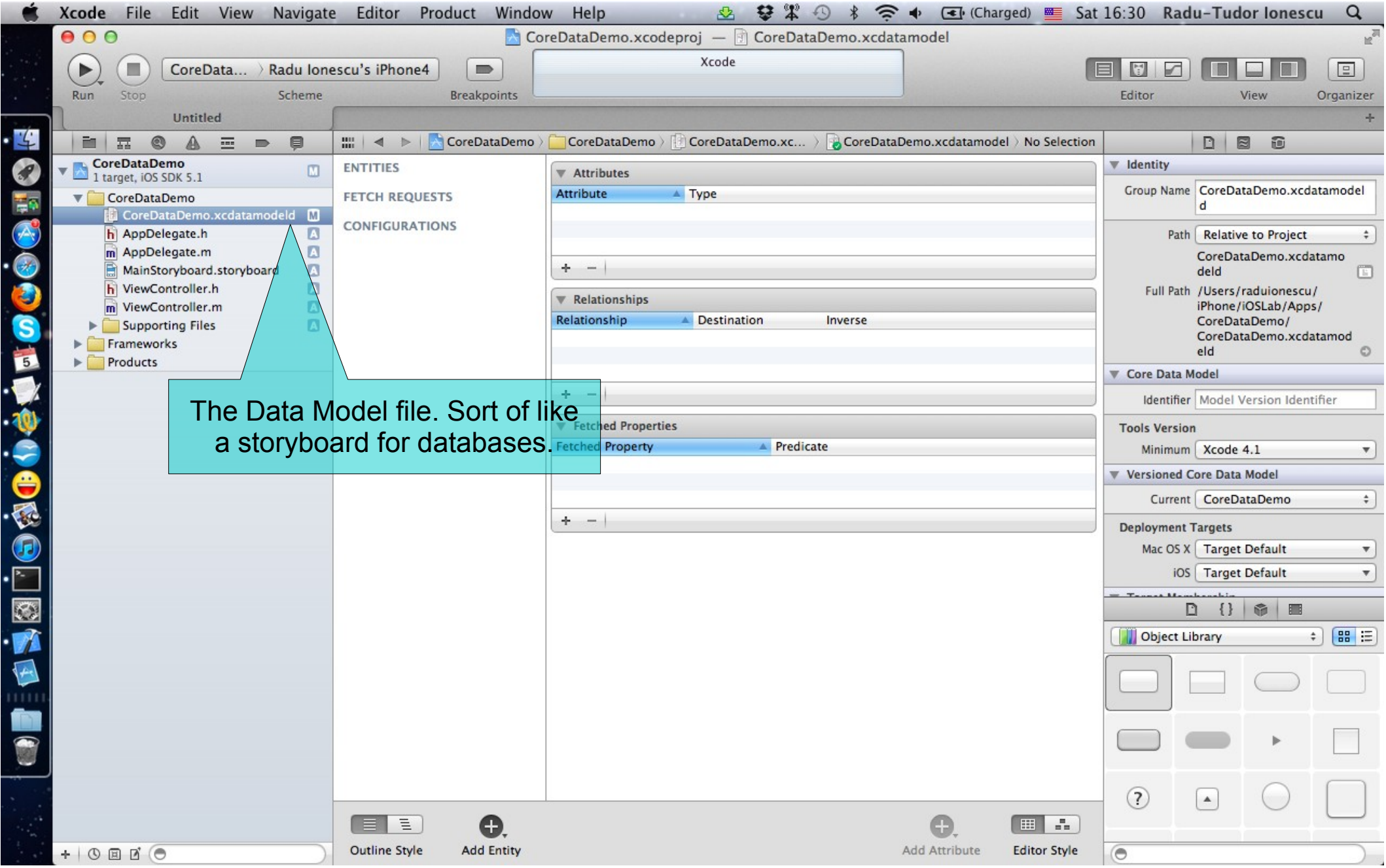
Core Data


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
Data Model
Mapping Model
NSManagedObject subclass
Data Model
A Core Data model file that allows you to use the design component of Xcode.
Cancel
Previous
Next
For creating a visual map of your application's database objects go to "New File ..." then Data Model under Core Data section.
Retina Display
Launch Images
Add Target
Validate Settings

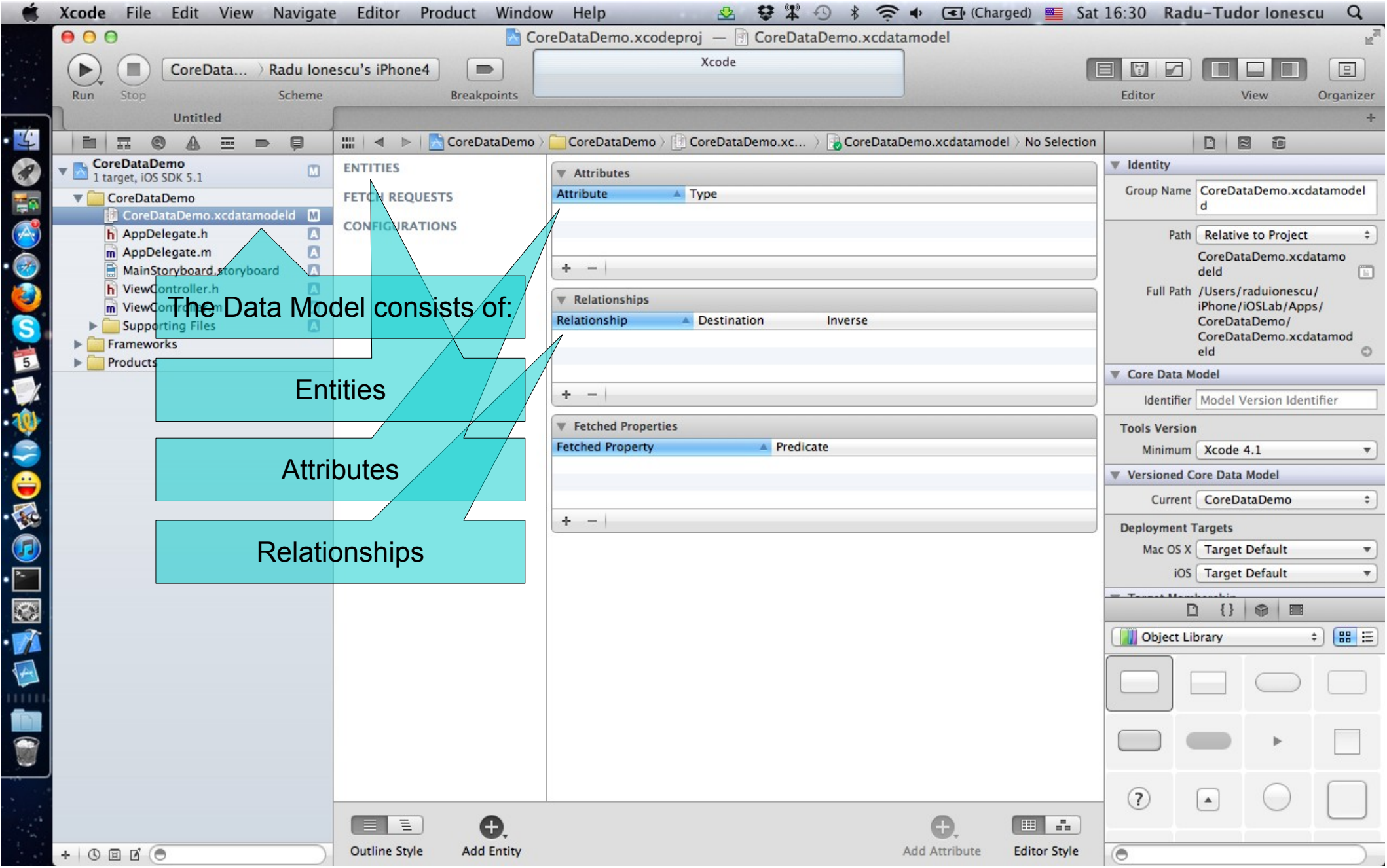
Core Data


Unless we have multiple databases, usually we name the Data Model our application name
Save As: CoreDataDemo
CoreDataDemo
FAVORITES
Desktop
raduionescu
iPhone
Viewlity
iOSLab
Ruby
Applications
Documents
Shared Folder
BlocksTest
Calculator(3of3)
CoreDataDemo
Demos
Done
GeoDeals
Half-Way
MapCompassTest
NearbyDeals
Older
CoreDataDemo
CoreDataDemo.xcodeproj
Group CoreDataDemo
Targets CoreDataDemo
New Folder
Cancel
Create
Identity
Project Name CoreDataDemo
Location Not Applicable
CoreDataDemo.xcodeproj
Full Path /Users/raduionescu/iPhone/iOSLab/Apps/CoreDataDemo/CoreDataDemo.xcodeproj
Project Document
Project Format Xcode 3.2-compatible
Organization
Class Prefix
Text Settings
Indent Using Spaces
Widths 4 4
Tab Indent
Wrap lines
Object Library
Launch Images
Add Target
Validate Settings

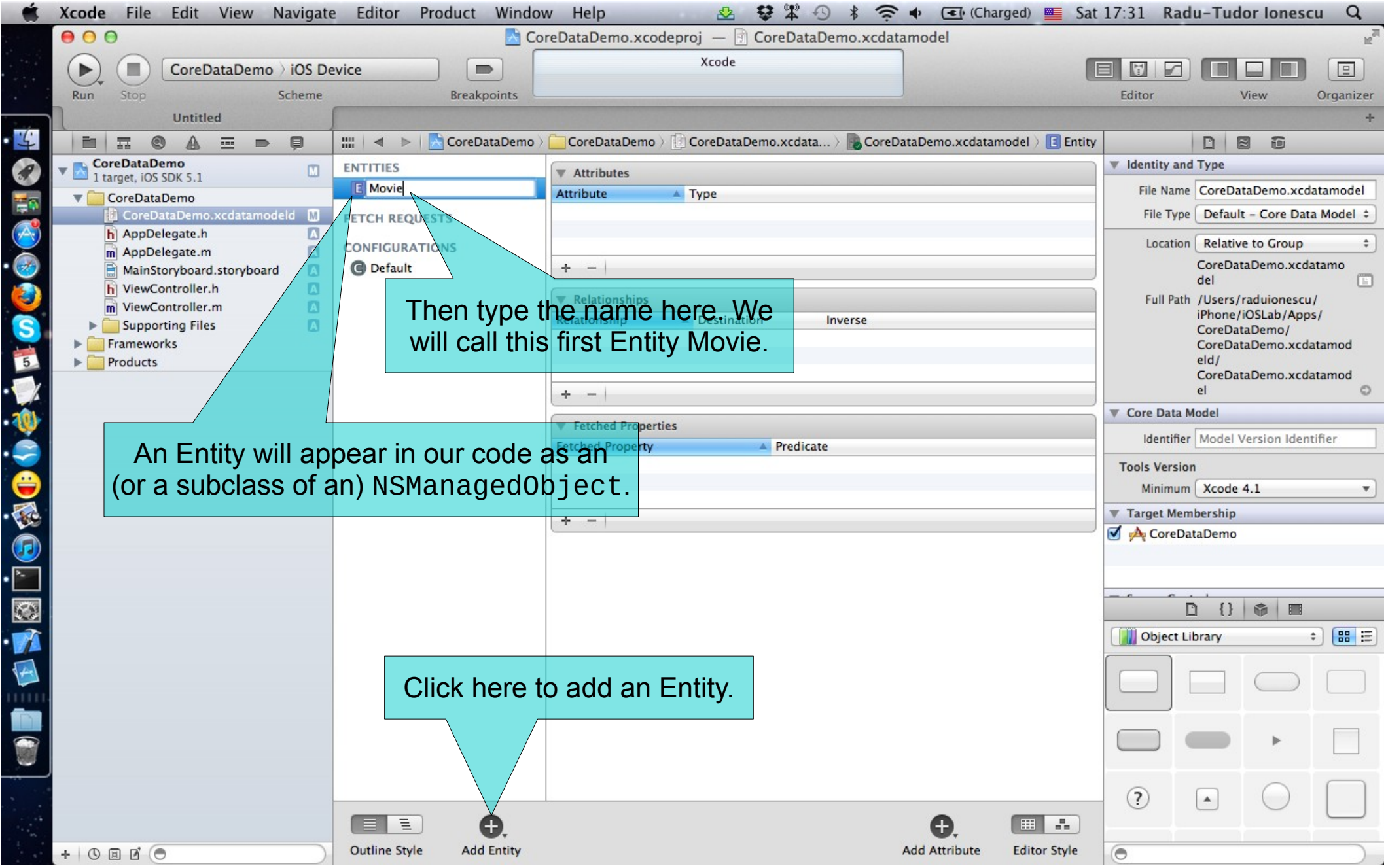
Core Data



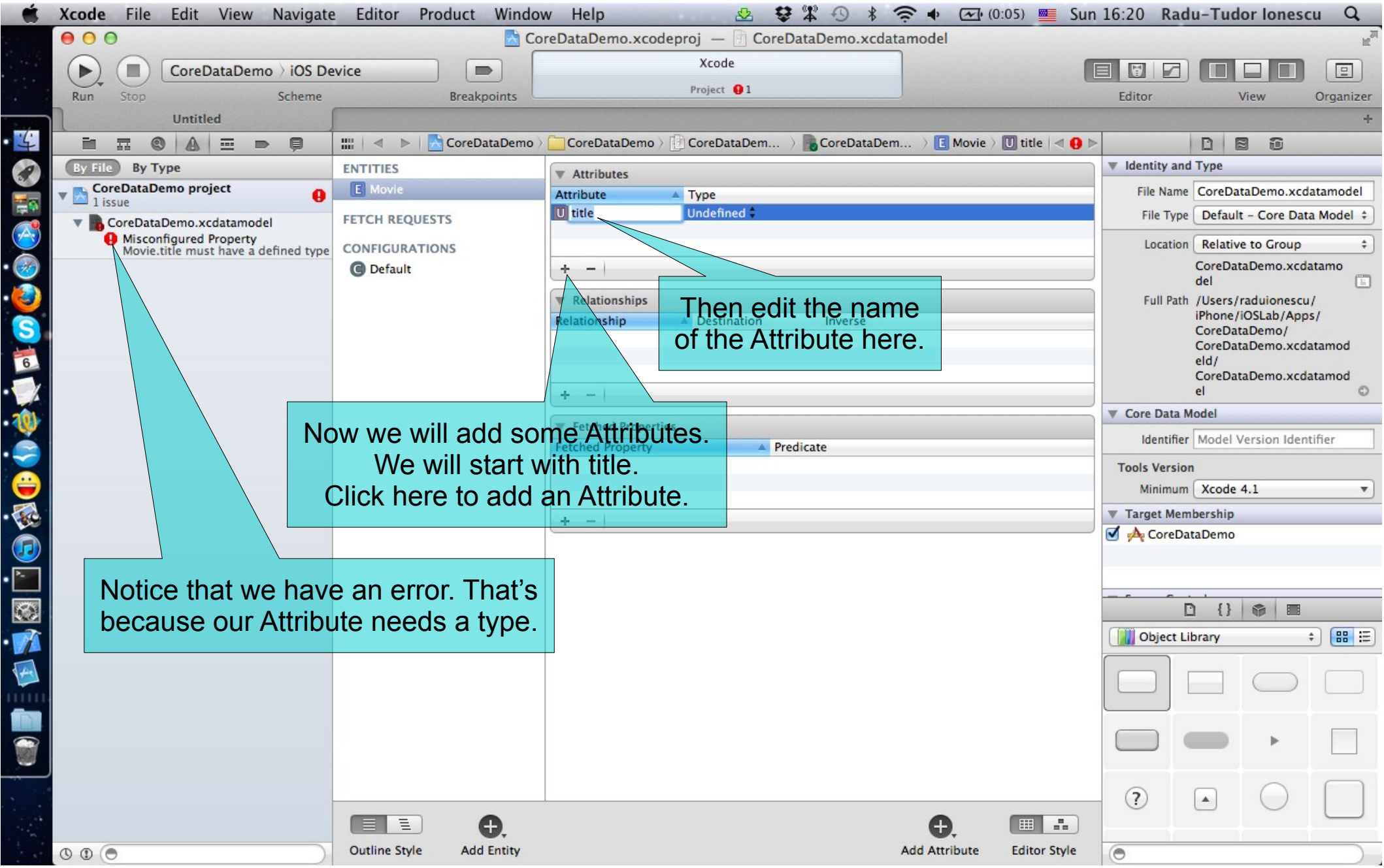
Core Data


The Data Model consists of:
Entities
Attributes
Relationships

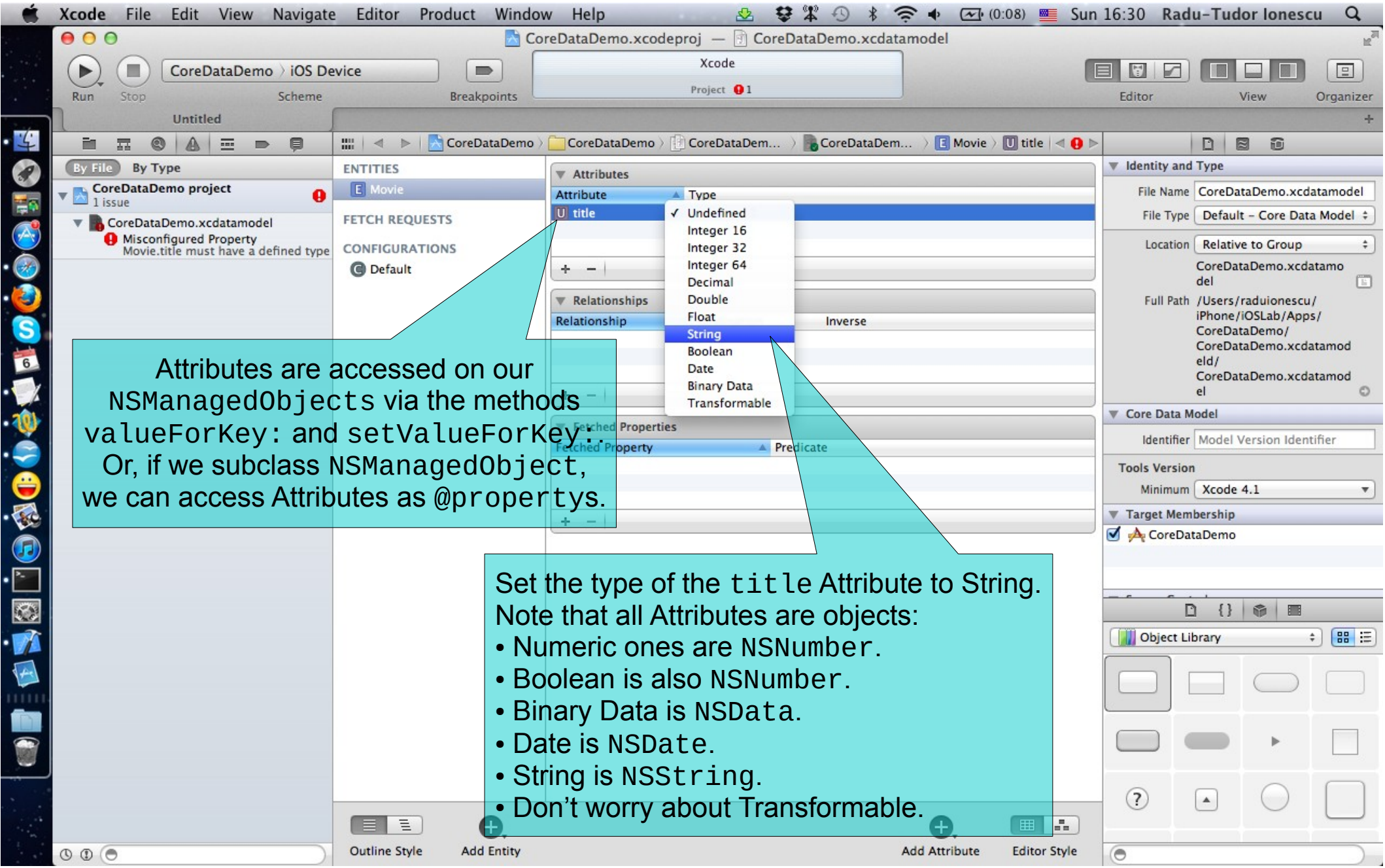
Core Data


Then type the name here. We will call this first Entity Movie.
An Entity will appear in our code as an (or a subclass of an) NSManagedObject.
Click here to add an Entity.
Outline Style
Add Entity
Add Attribute
Editor Style

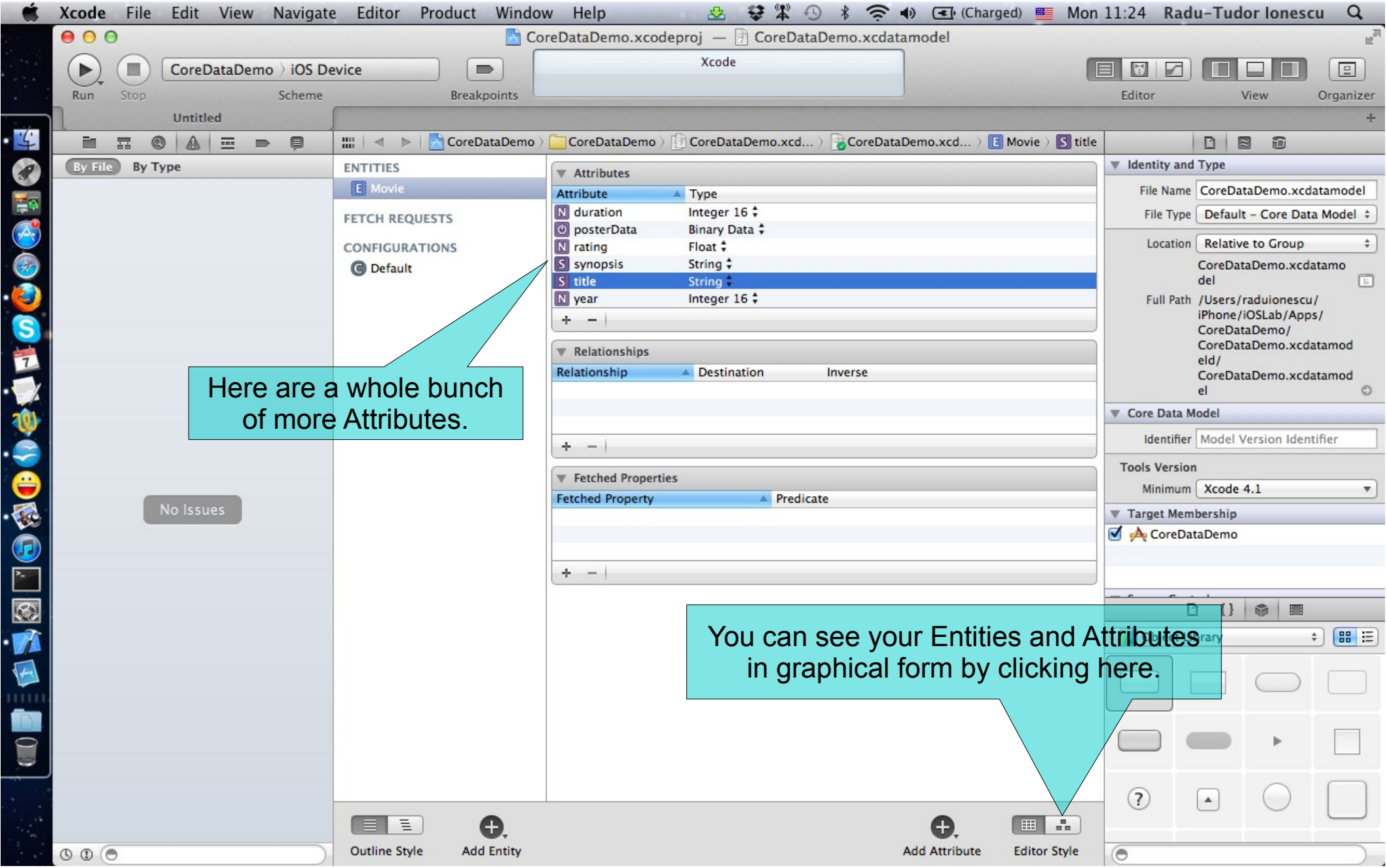
Core Data


Xcode File Edit View Navigate Editor Product Window Help
CoreDataDemo.xcodeproj — CoreDataDemo.xcdatamodel
Run Stop Scheme Breakpoints Editor View Organizer
Untitled
By File By Type
CoreDataDemo project
1 issue
CoreDataDemo.xcdatamodel
Misconfigured Property
Movie.title must have a defined type
ENTITIES
Movie
FETCH REQUESTS
CONFIGURATIONS
Default
Attributes
Attribute Type
title Undefined
Relationships
Relationship Destination Inverse
Fetched Property Predicate
Identity and Type
File Name CoreDataDemo.xcdatamodel
File Type Default – Core Data Model
Location Relative to Group
Full Path /Users/raduionescu/iPhone/iOSLab/Apps/CoreDataDemo/CoreDataDemo.xcdatamodeld/CoreDataDemo.xcdatamodel
Core Data Model
Identifier Model Version Identifier
Tools Version
Minimum Xcode 4.1
Target Membership
CoreDataDemo
Object Library
Outline Style
Add Entity
Add Attribute
Editor Style
Then edit the name of the Attribute here.
Now we will add some Attributes.
We will start with title.
Click here to add an Attribute.
Notice that we have an error. That's because our Attribute needs a type.

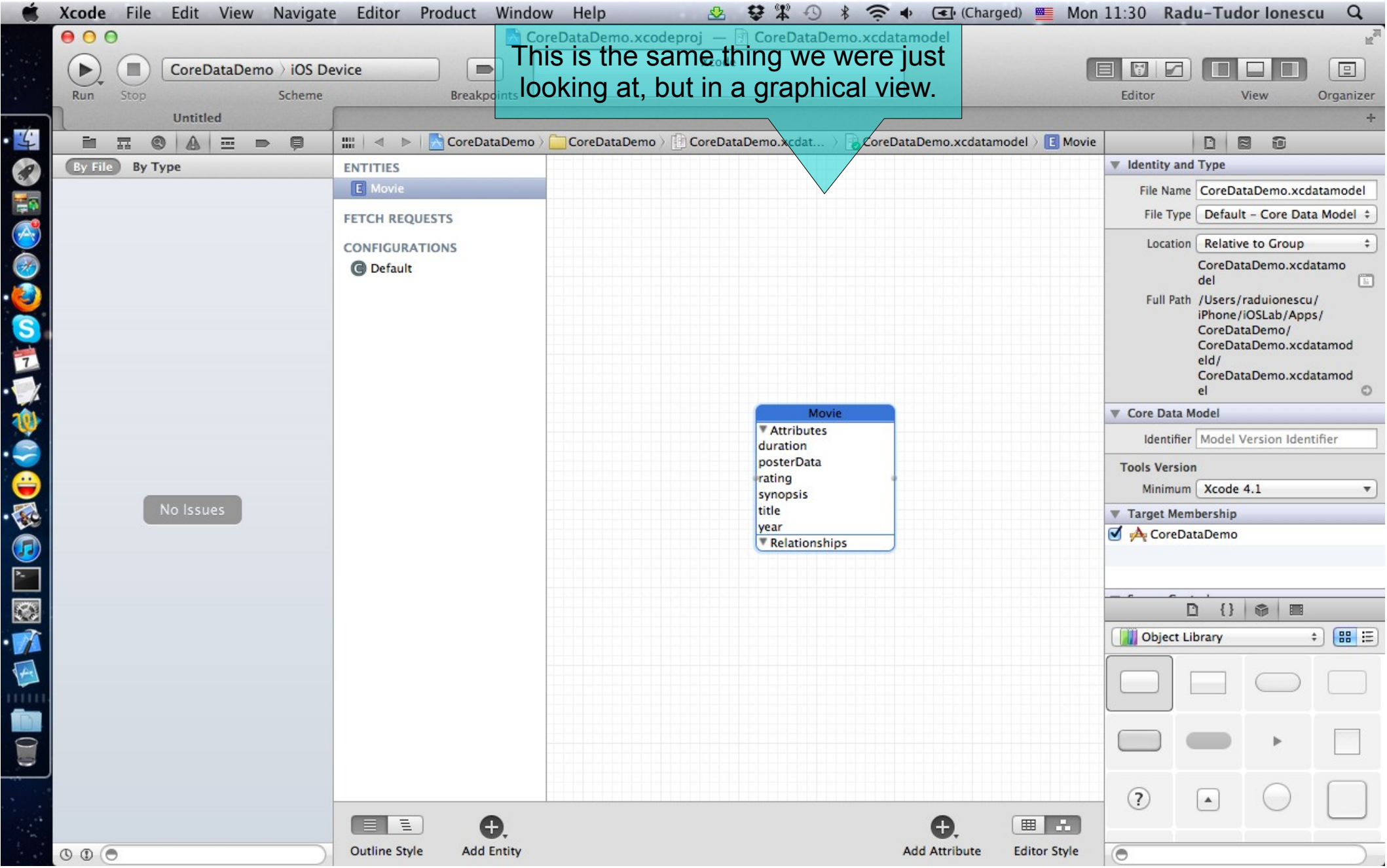
Core Data


Attributes are accessed on our NSManagedObjects via the methods valueForKey: and setValueForKey:.
Or, if we subclass NSManagedObject, we can access Attributes as @propertys.
Set the type of the title Attribute to String.
Note that 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.
Undefined
Integer 16
Integer 32
Integer 64
Decimal
Double
Float
String
Boolean
Date
Binary Data
Transformable
Outline Style
Add Entity
Add Attribute
Editor Style

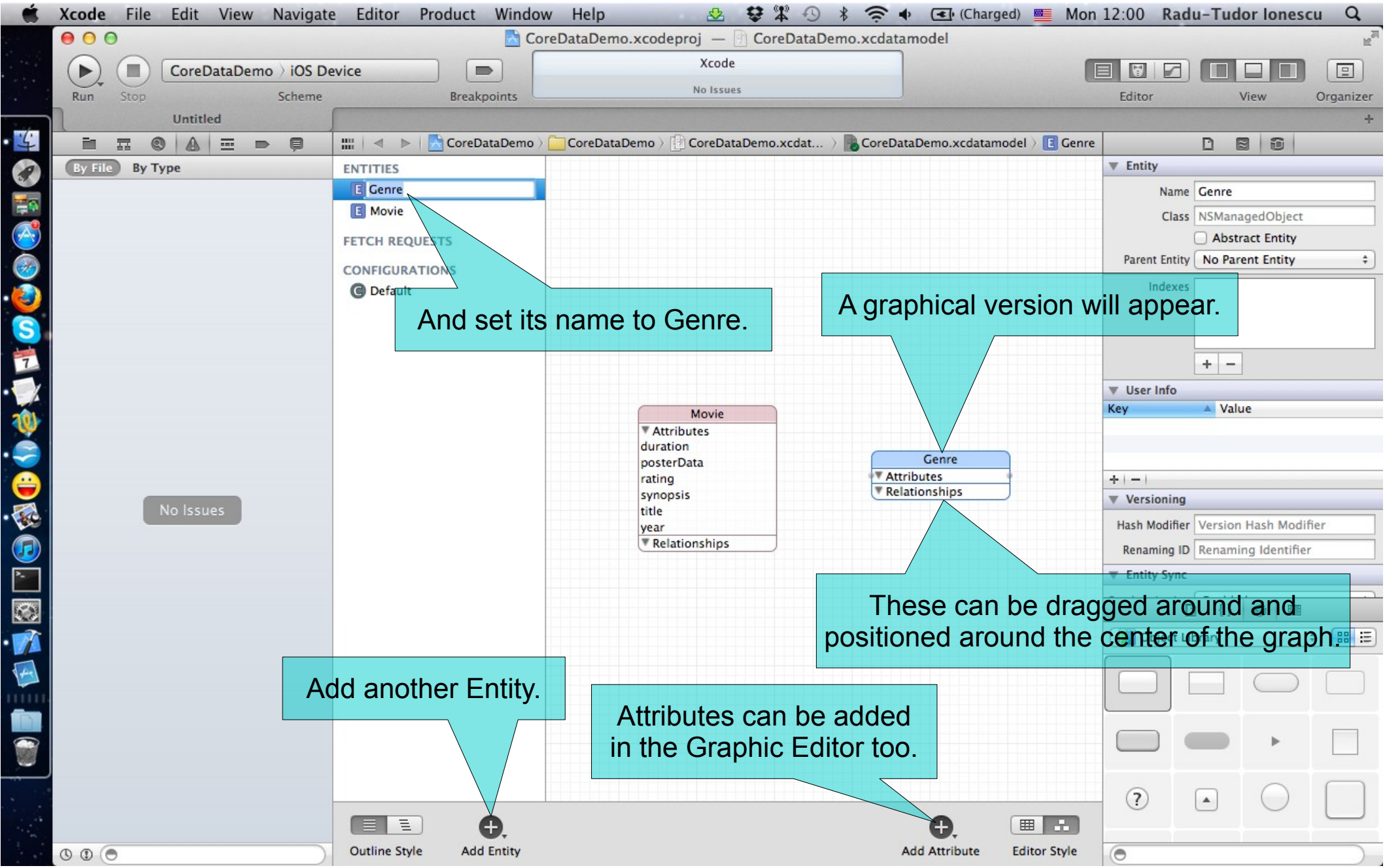
Core Data


Here are a whole bunch of more Attributes.
You can see your Entities and Attributes in graphical form by clicking here.
CoreDataDemo.xcodeproj — CoreDataDemo.xcdatamodel
Run
Stop
CoreDataDemo 〉 iOS Device
Scheme
Breakpoints
Editor
View
Organizer
Untitled
By File
By Type
No Issues
ENTITIES
Movie
FETCH REQUESTS
CONFIGURATIONS
Default
Attributes
Attribute
Type
duration
Integer 16
posterData
Binary Data
rating
Float
synopsis
String
title
String
year
Integer 16
Relationships
Relationship
Destination
Inverse
Fetched Properties
Fetched Property
Predicate
Identity and Type
File Name
CoreDataDemo.xcdatamodel
File Type
Default – Core Data Model
Location
Relative to Group
Full Path
/Users/raduionescu/iPhone/iOSLab/Apps/CoreDataDemo/CoreDataDemo.xcdatamodeld/CoreDataDemo.xcdatamodel
Core Data Model
Identifier
Model Version Identifier
Tools Version
Minimum
Xcode 4.1
Target Membership
CoreDataDemo
Outline Style
Add Entity
Add Attribute
Editor Style

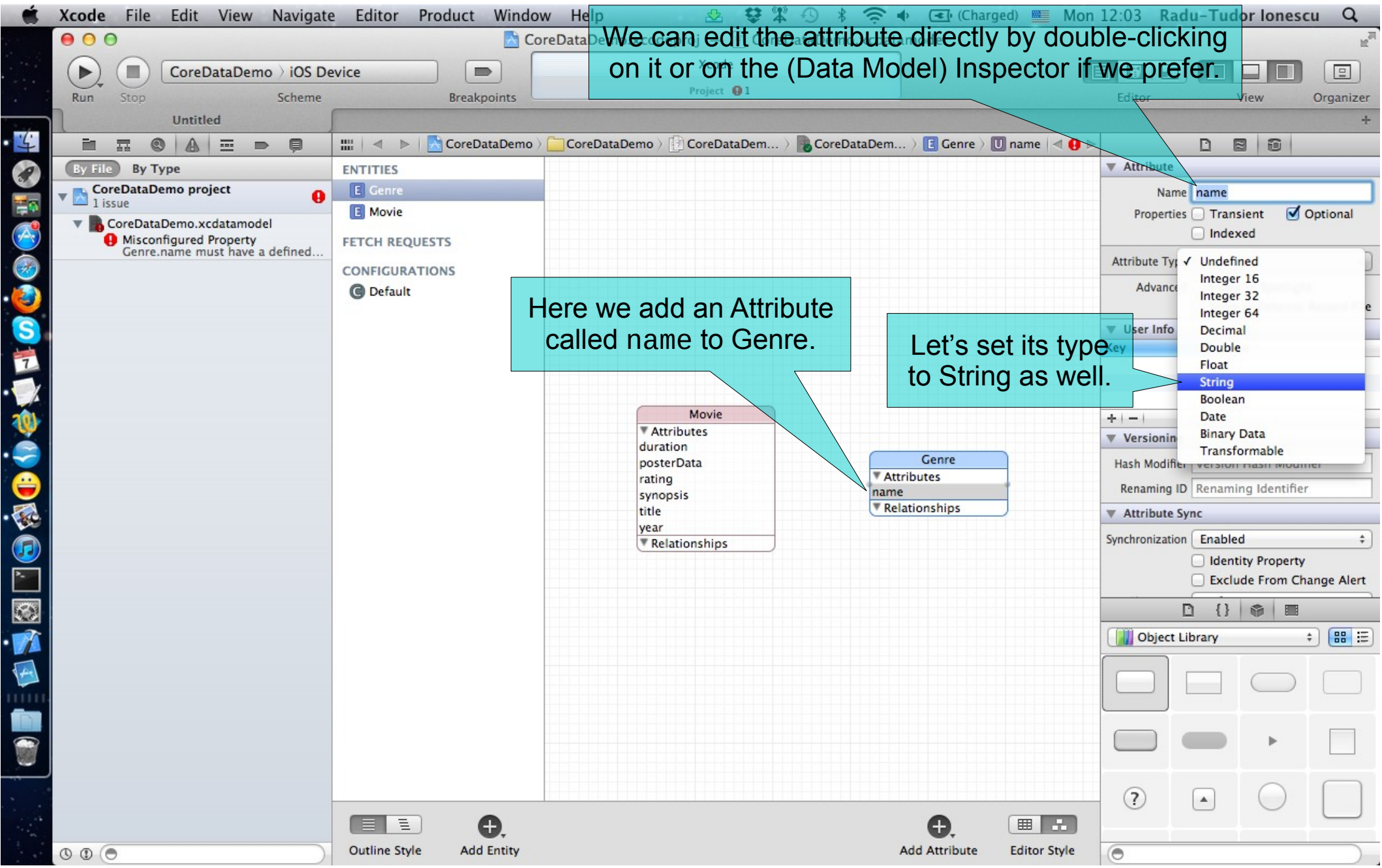
Core Data


This is the same thing we were just looking at, but in a graphical view.
Xcode File Edit View Navigate Editor Product Window Help
Mon 11:30 Radu-Tudor Ionescu
CoreDataDemo.xcodeproj — CoreDataDemo.xcdatamodel
Run Stop Scheme Breakpoints Editor View Organizer
Untitled
By File By Type
No Issues
ENTITIES
Movie
FETCH REQUESTS
CONFIGURATIONS
Default
Movie
Attributes
duration
posterData
rating
synopsis
title
year
Relationships
Identity and Type
File Name CoreDataDemo.xcdatamodel
File Type Default – Core Data Model
Location Relative to Group
Full Path /Users/raduionescu/iPhone/iOSLab/Apps/CoreDataDemo/CoreDataDemo.xcdatamodeld/CoreDataDemo.xcdatamodel
Core Data Model
Identifier Model Version Identifier
Tools Version
Minimum Xcode 4.1
Target Membership
CoreDataDemo
Object Library
Outline Style
Add Entity
Add Attribute
Editor Style

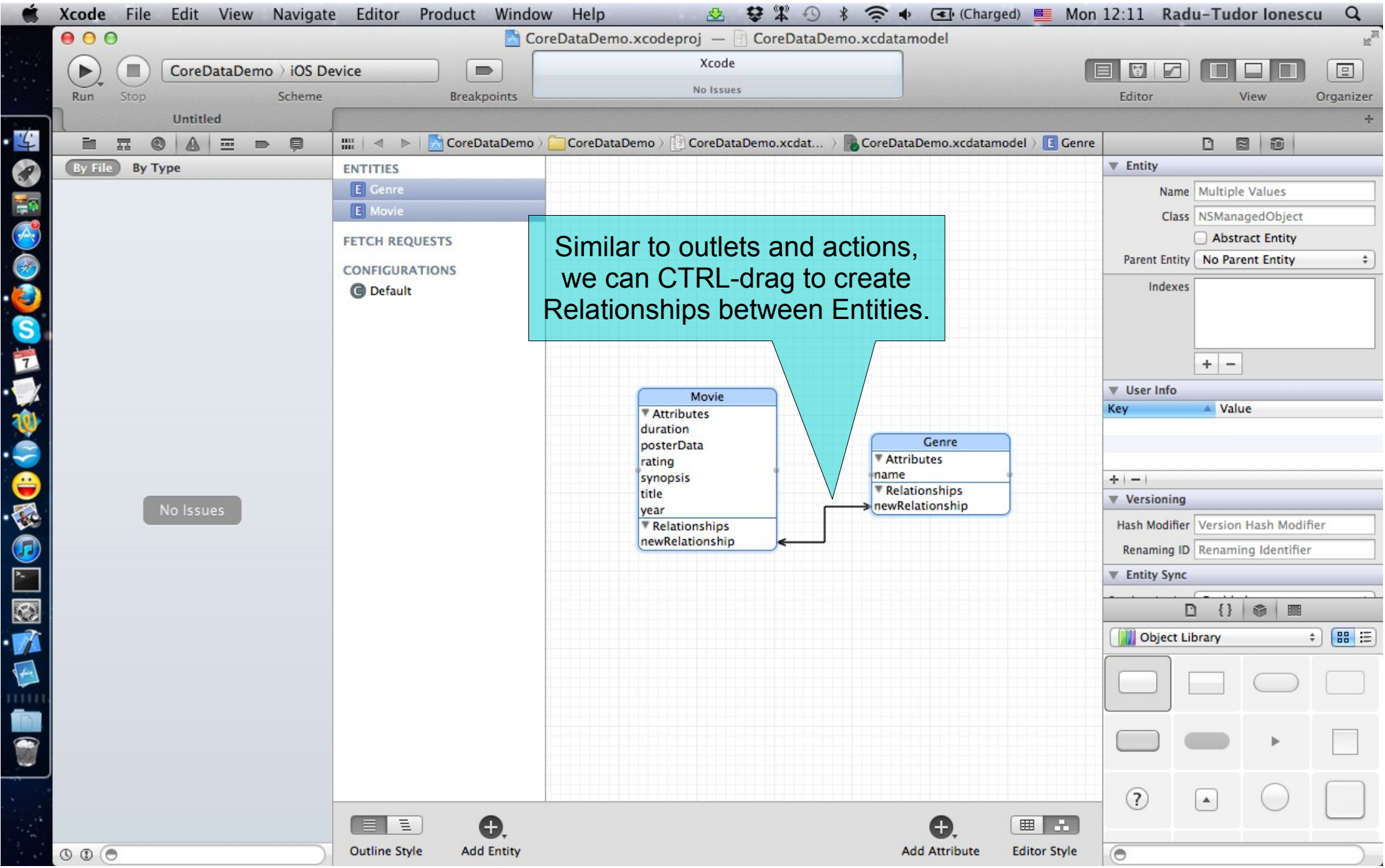
Core Data


Xcode File Edit View Navigate Editor Product Window Help
CoreDataDemo.xcodeproj — CoreDataDemo.xcdatamodel
Run Stop Scheme Breakpoints Editor View Organizer
ENTITIES
Genre
Movie
FETCH REQUESTS
CONFIGURATIONS
Default
And set its name to Genre.
A graphical version will appear.
Movie
Attributes
duration
posterData
rating
synopsis
title
year
Relationships
Genre
Attributes
Relationships
These can be dragged around and positioned around the center of the graph.
Add another Entity.
Attributes can be added in the Graphic Editor too.
Outline Style
Add Entity
Add Attribute
Editor Style
Entity
Name Genre
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
No Issues

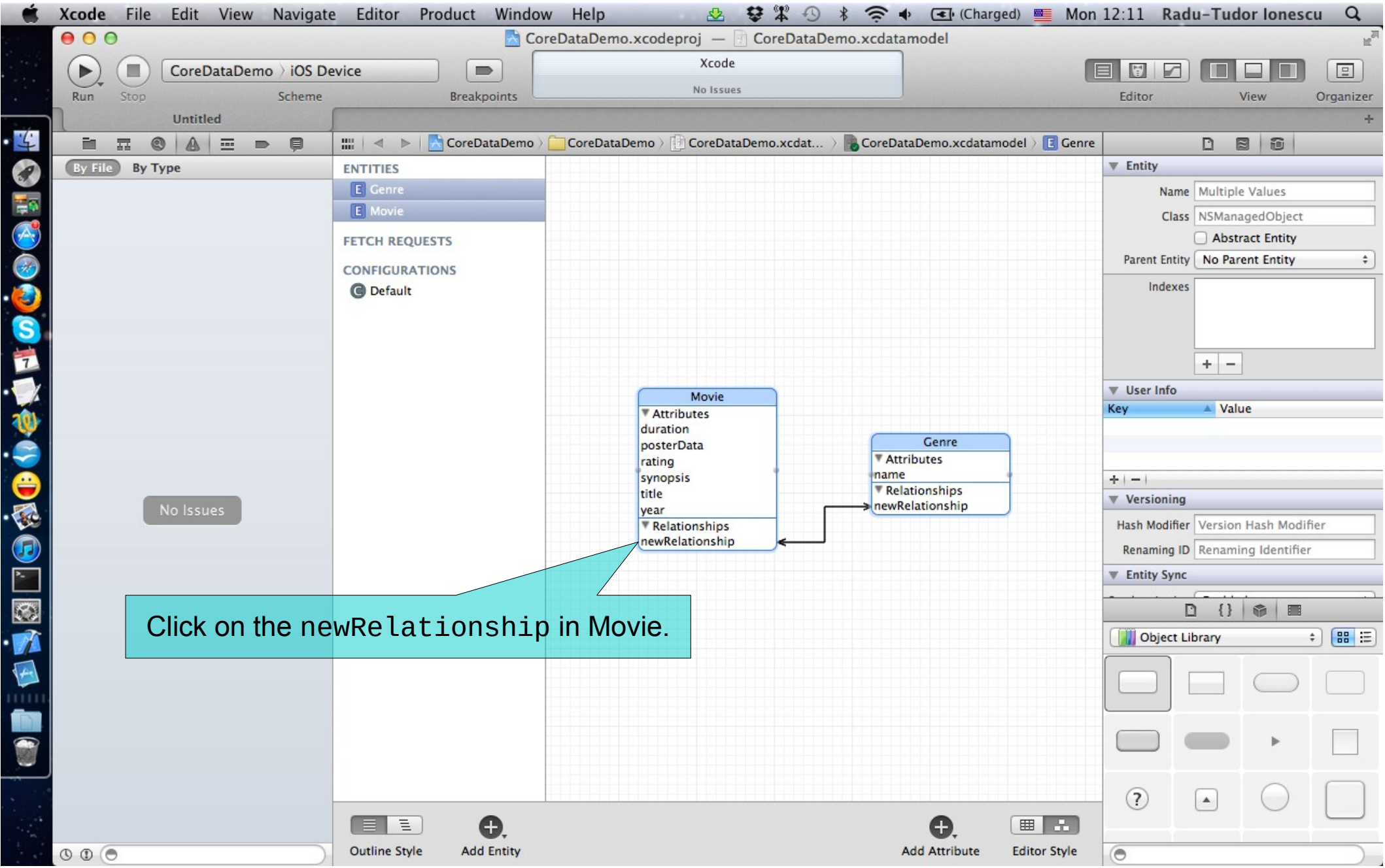
Core Data


We can edit the attribute directly by double-clicking on it or on the (Data Model) Inspector if we prefer.
Here we add an Attribute called name to Genre.
Let's set its type to String as well.
Xcode File Edit View Navigate Editor Product Window Help
CoreDataDemo iOS Device
Run Stop Scheme Breakpoints
Untitled
By File By Type
CoreDataDemo project
1 issue
CoreDataDemo.xcdatamodel
Misconfigured Property
Genre.name must have a defined...
ENTITIES
Genre
Movie
FETCH REQUESTS
CONFIGURATIONS
Default
Movie
Attributes
duration
posterData
rating
synopsis
title
year
Relationships
Genre
Attributes
name
Relationships
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
Attribute Sync
Synchronization Enabled
Identity Property
Exclude From Change Alert
Object Library
Outline Style
Add Entity
Add Attribute
Editor Style

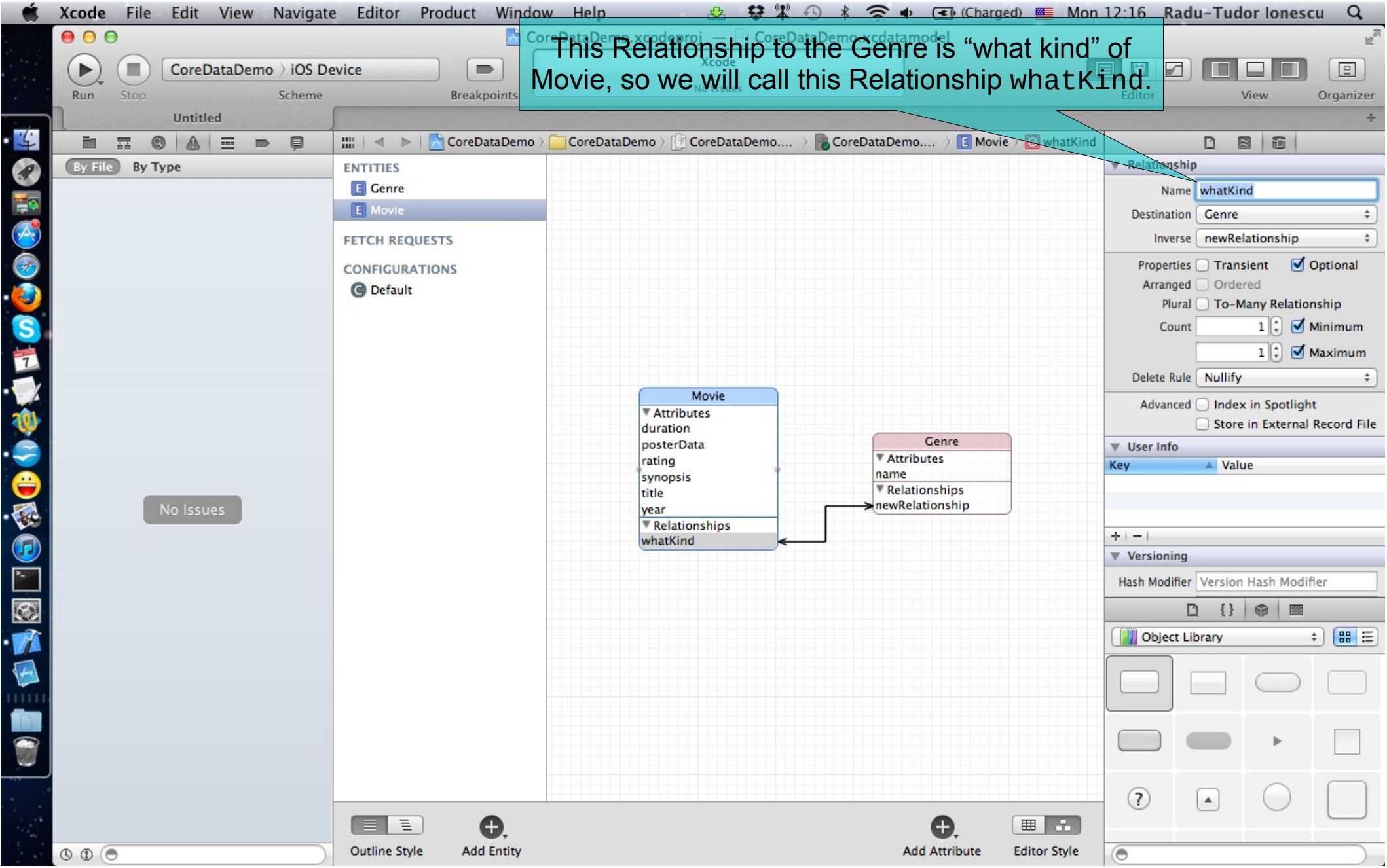
Core Data


Xcode File Edit View Navigate Editor Product Window Help (Charged) Mon 12:11 Radu-Tudor Ionescu
CoreDataDemo.xcodeproj — CoreDataDemo.xcdatamodel
Run Stop CoreDataDemo iOS Device Scheme Breakpoints
Xcode
No Issues
Editor View Organizer
Untitled
By File By Type
No Issues
CoreDataDemo CoreDataDemo CoreDataDemo.xcdat... CoreDataDemo.xcdatamodel Genre
ENTITIES
Genre
Movie
FETCH REQUESTS
CONFIGURATIONS
Default
Similar to outlets and actions, we can CTRL-drag to create Relationships between Entities.
Movie
Attributes
duration
posterData
rating
synopsis
title
year
Relationships
newRelationship
Genre
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
Object Library
Outline Style Add Entity Add Attribute Editor Style

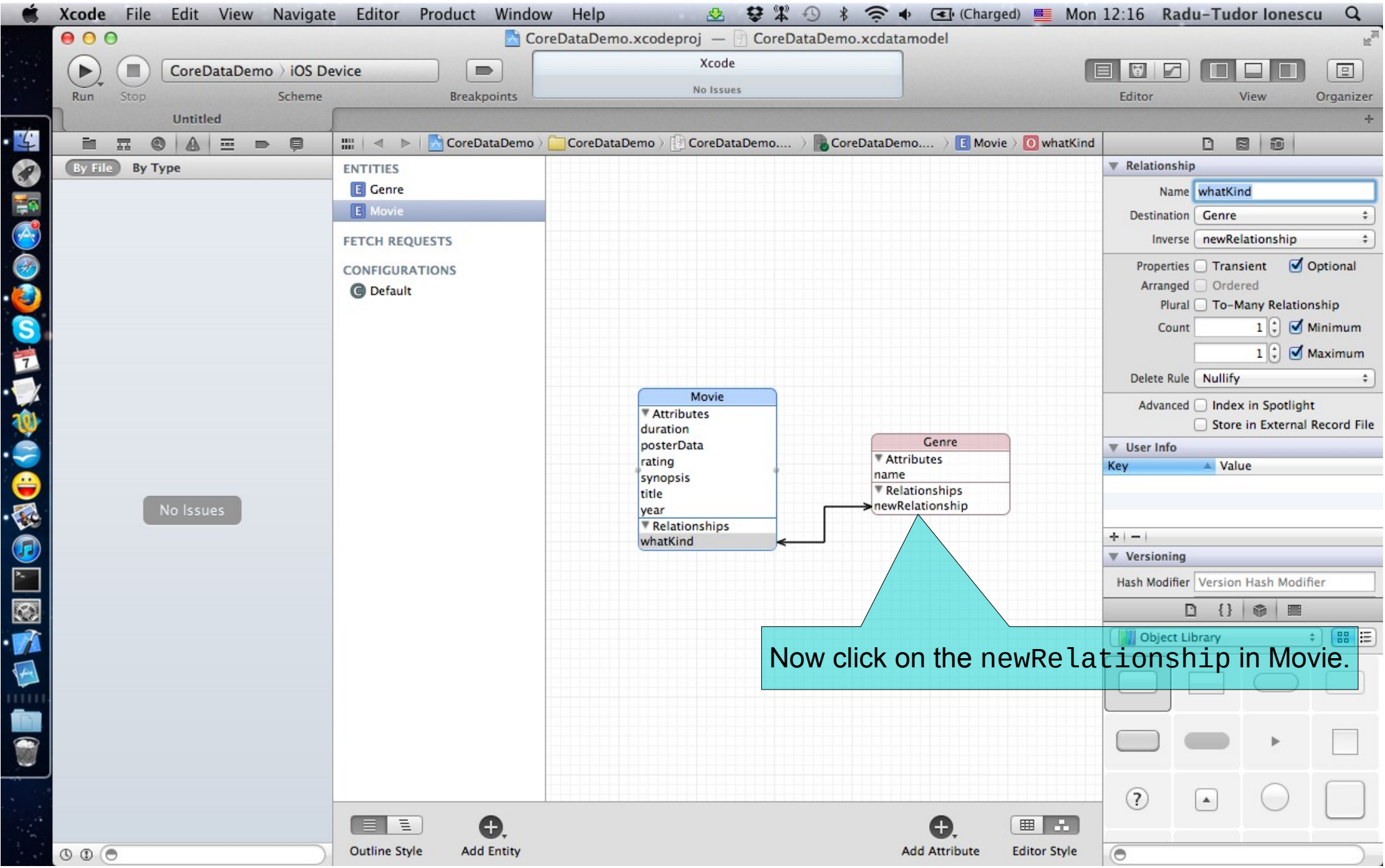
Core Data


Xcode File Edit View Navigate Editor Product Window Help
CoreDataDemo.xcodeproj — CoreDataDemo.xcdatamodel
Run Stop CoreDataDemo iOS Device Scheme Breakpoints
Xcode
No Issues
Editor View Organizer
Untitled
By File By Type
No Issues
ENTITIES
Genre
Movie
FETCH REQUESTS
CONFIGURATIONS
Default
CoreDataDemo CoreDataDemo CoreDataDemo.xcdat... CoreDataDemo.xcdatamodel Genre
Movie
Attributes
duration
posterData
rating
synopsis
title
year
Relationships
newRelationship
Genre
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
Object Library
Outline Style
Add Entity
Add Attribute
Editor Style
Click on the newRelationship in Movie.

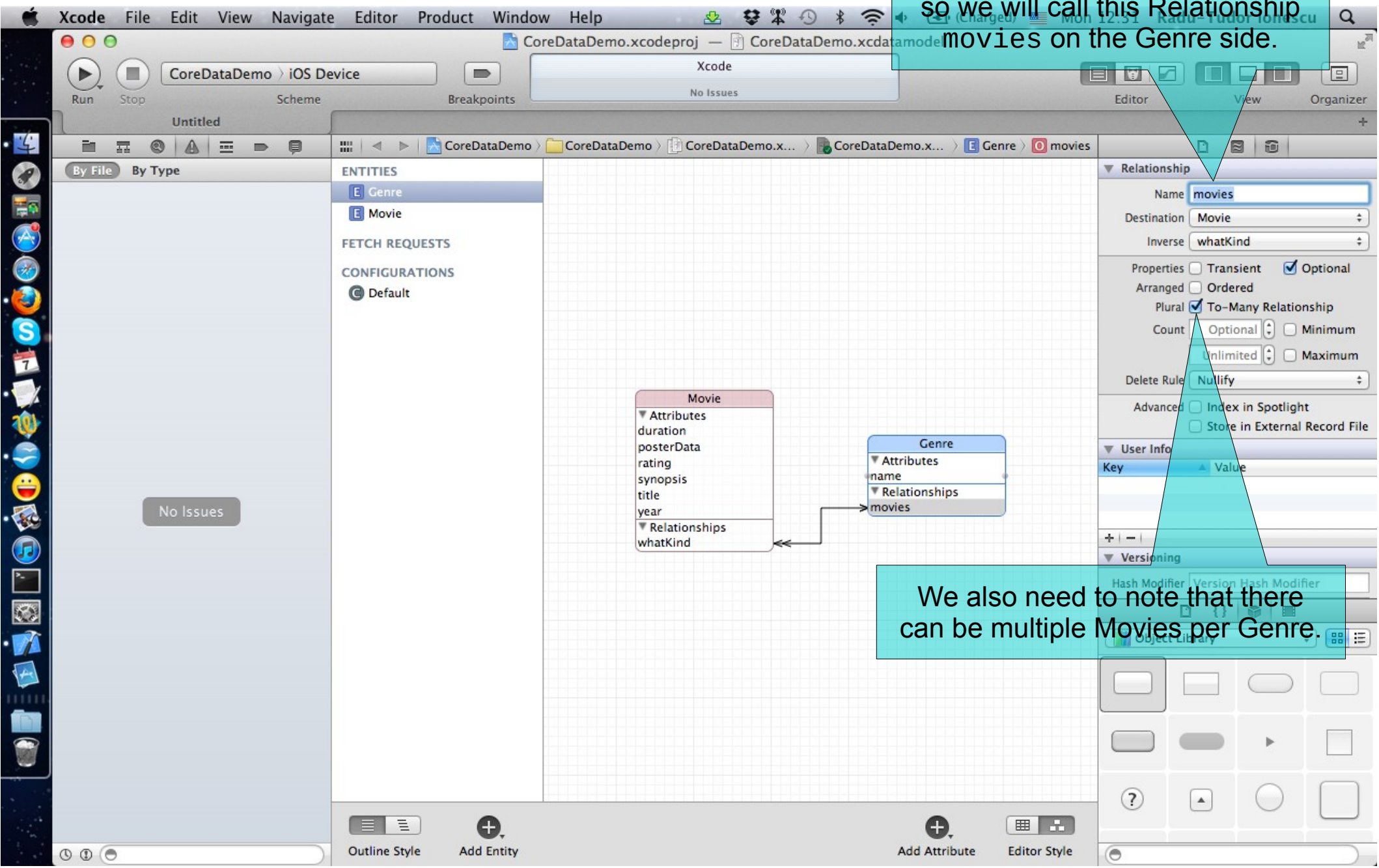
Core Data


This Relationship to the Genre is “what kind” of Movie, so we will call this Relationship whatKind.
Xcode File Edit View Navigate Editor Product Window Help
Mon 12:16 Radu-Tudor Ionescu
CoreDataDemo › iOS Device
Run Stop Scheme Breakpoints
Untitled
By File By Type
No Issues
ENTITIES
Genre
Movie
FETCH REQUESTS
CONFIGURATIONS
Default
Movie
Attributes
duration
posterData
rating
synopsis
title
year
Relationships
whatKind
Genre
Attributes
name
Relationships
newRelationship
Relationship
Name whatKind
Destination Genre
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
Object Library
Outline Style
Add Entity
Add Attribute
Editor Style

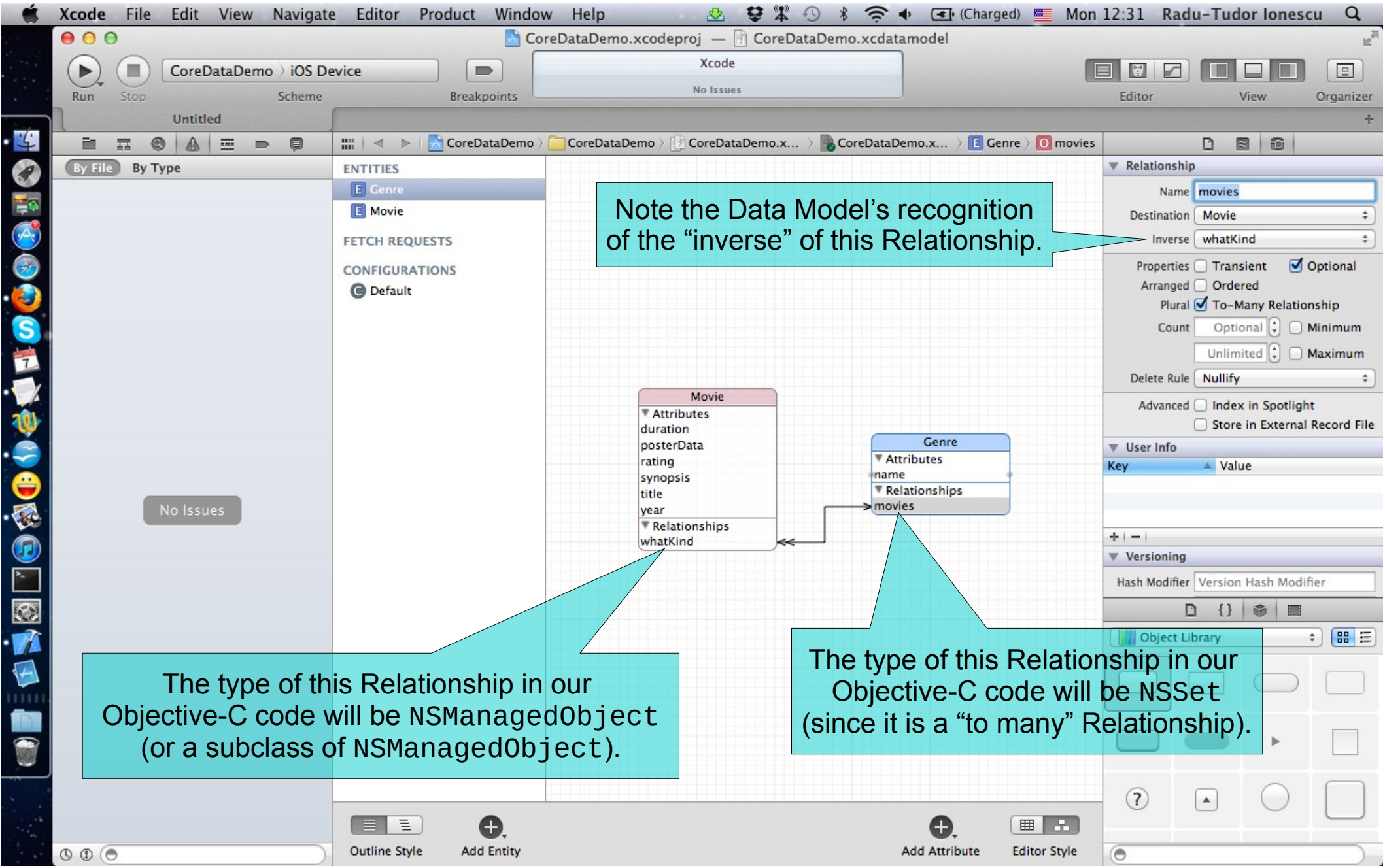
Core Data


Xcode File Edit View Navigate Editor Product Window Help
CoreDataDemo.xcodeproj — CoreDataDemo.xcdatamodel
Run Stop CoreDataDemo › iOS Device Scheme Breakpoints
Xcode
No Issues
Editor View Organizer
Untitled
By File By Type
No Issues
ENTITIES
Genre
Movie
FETCH REQUESTS
CONFIGURATIONS
Default
Movie
Attributes
duration
posterData
rating
synopsis
title
year
Relationships
whatKind
Genre
Attributes
name
Relationships
newRelationship
Relationship
Name whatKind
Destination Genre
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
Object Library
Now click on the newRelationship in Movie.
Outline Style
Add Entity
Add Attribute
Editor Style

Core Data


A Genre can have many Movies, so we will call this Relationship movies on the Genre side.
Xcode File Edit View Navigate Editor Product Window Help
CoreDataDemo.xcodeproj — CoreDataDemo.xcdatamodel
CoreDataDemo › iOS Device
Run Stop Scheme Breakpoints
Xcode
No Issues
Editor View Organizer
Untitled
By File By Type
No Issues
ENTITIES
Genre
Movie
FETCH REQUESTS
CONFIGURATIONS
Default
CoreDataDemo › CoreDataDemo › CoreDataDemo.x... › CoreDataDemo.x... › Genre › movies
Movie
Attributes
duration
posterData
rating
synopsis
title
year
Relationships
whatKind
Genre
Attributes
name
Relationships
movies
Relationship
Name movies
Destination Movie
Inverse whatKind
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
We also need to note that there can be multiple Movies per Genre.
Outline Style
Add Entity
Add Attribute
Editor Style

Core Data


Xcode File Edit View Navigate Editor Product Window Help (Charged) Mon 12:31 Radu-Tudor Ionescu
CoreDataDemo.xcodeproj — CoreDataDemo.xcdatamodel
Run Stop CoreDataDemo › iOS Device Scheme Breakpoints Xcode No Issues Editor View Organizer
Untitled
By File By Type
No Issues
ENTITIES
Genre
Movie
FETCH REQUESTS
CONFIGURATIONS
Default
CoreDataDemo › CoreDataDemo › CoreDataDemo.x... › CoreDataDemo.x... › Genre › movies
Note the Data Model's recognition of the "inverse" of this Relationship.
Movie
Attributes
duration
posterData
rating
synopsis
title
year
Relationships
whatKind
Genre
Attributes
name
Relationships
movies
Relationship
Name movies
Destination Movie
Inverse whatKind
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
Object Library
The type of this Relationship in our Objective-C code will be NSManagedObject (or a subclass of NSManagedObject).
The type of this Relationship in our Objective-C code will be NSSet (since it is a "to many" Relationship).
Outline Style Add Entity Add Attribute Editor Style

Core Data

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 you 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 an Empty Application Project. Then your AppDelegate will have a `managedObjectContext` `@property`.
- We are 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 alloc] initWithFileURL: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 using:

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

- If it does not, create it using:

```
- (void)saveToURL:(NSURL *)url
 forSaveOperation:(UIDocumentSaveOperation)operation
 completionHandler:(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 alloc]
                  initWithFileURL:(NSURL *)url];
if ([[NSFileManager defaultManager]
     fileExistsAtPath:[url path]])
{
   [document openWithCompletionHandler:^(BOOL success) {

      if (success) [self documentIsReady];
      else NSLog(@“Couldn’t open document at %@”, url);
   }];
}
else
{
   [sourceDocument saveToURL:url
             forSaveOperation:UIDocumentSaveForCreating
            completionHandler:^(BOOL success) {

      if (success) [self openDocument];
      else 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
    }
}
```

UIManagedDocument

Other documentStates

- 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 NSNotifications 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
          selector:(SEL)methodToSendIfSomethingHappens
              name:(NSString *)name
            object:(id)sender;
```

The meaning of the arguments

- **`observer`** is the object to get notified;
- **`name`** is what you are observing (a constant somewhere);
- **`sender`** is the object whose changes you're interested in (`nil` is anyone's).

NSNotification

NSNotificationCenter

- You will then be notified when the named event happens:

```
- (void)methodToSendIfSomethingHappens:
                                (NSNotification *)notification
{
 NSString* name = notification.name
 // the name passed above

 id obj = notification.object
 // the object sending you the notification

 NSDictionary *info = 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];
```

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

- Failure to remove yourself can sometimes result in crashes.
- 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

Receiving the NSManagedObjectContext notifications

- NSManagedObjectContextObjectsDidChangeNotification or NSManagedObjectContextDidSaveNotification:

```
- (void)contextChanged:(NSNotification *)notification
{
    NSDictionary *info = notification.userInfo;
}
```

The info NSDictionary contains the following keys

- NSInsertedObjectsKey gives an array of objects which were inserted.
- NSUpdatedObjectsKey gives an array of objects whose attributes changed.
- NSDeletedObjectsKey gives an array of objects which were deleted.

NSNotification

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. We did this in the NearbyDeals app that we created in our Labs:

```
[[NSNotificationCenter defaultCenter]
 postNotificationName:@"locationUpdateNotification"
               object:self];

[[NSNotificationCenter defaultCenter]
 addObserver:self
    selector:@selector(showMapRegionForNotification:)
        name:@"locationUpdateNotification"
      object:[DealsModel sharedModel]];
```

- See `NSNotificationCenter` documentation for more information.
- Don't abuse this mechanism!
- Don't use it to essentially get "global variables" in your application.

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.doc saveToURL:self.doc.fileURL
   forSaveOperation:UIDocumentSaveForOverwriting
  completionHandler:^(BOOL success) {

  if (!success)
    NSLog(@“Save failed for %@”, self.doc.localizedName);
}];

/* The document is not saved at this point in the
 * code (only once the block above executes). */
```

- Note the two `UIManagedDocument` properties used:

```
@property (nonatomic, strong) NSURL *fileURL;
// specified originally in initWithFileURL:

@property (readonly) NSString *localizedName;
```

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.doc closeWithCompletionHandler:^(BOOL success) {

  if (!success)
   NSLog(@“Close failed for %@”, self.doc.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.

Core Data

Inserting objects into the database

- We grabbed an `NSManagedObjectContext` 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:

```
NSManagedObject *movie = [NSEntityDescription
    insertNewObjectForEntityForName:@"Movie"
             inManagedObjectContext:managedObjectContext];
```

- Note that this `NSEntityDescription` class method returns an `NSManagedObject` instance.
- All objects in the database are represented by `NSManagedObjects` or by subclasses of `NSManagedObjects`.
- 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 Data Model Inspector).

Core Data

How to access Attributes in an `NSManagedObject` instance

- You can access the Attributes 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!

Core Data

How to access Attributes in an `NSManagedObject` instance

- The **key** is an Attribute name in your data mapping.

  For example, `@"posterData"`.

- The **value** is whatever is stored (or to be stored) in the database.

  It will 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

- Yes, `UIManagedDocument` auto-saves.
- But explicitly saving when a batch of changes is made is good practice.

Core Data

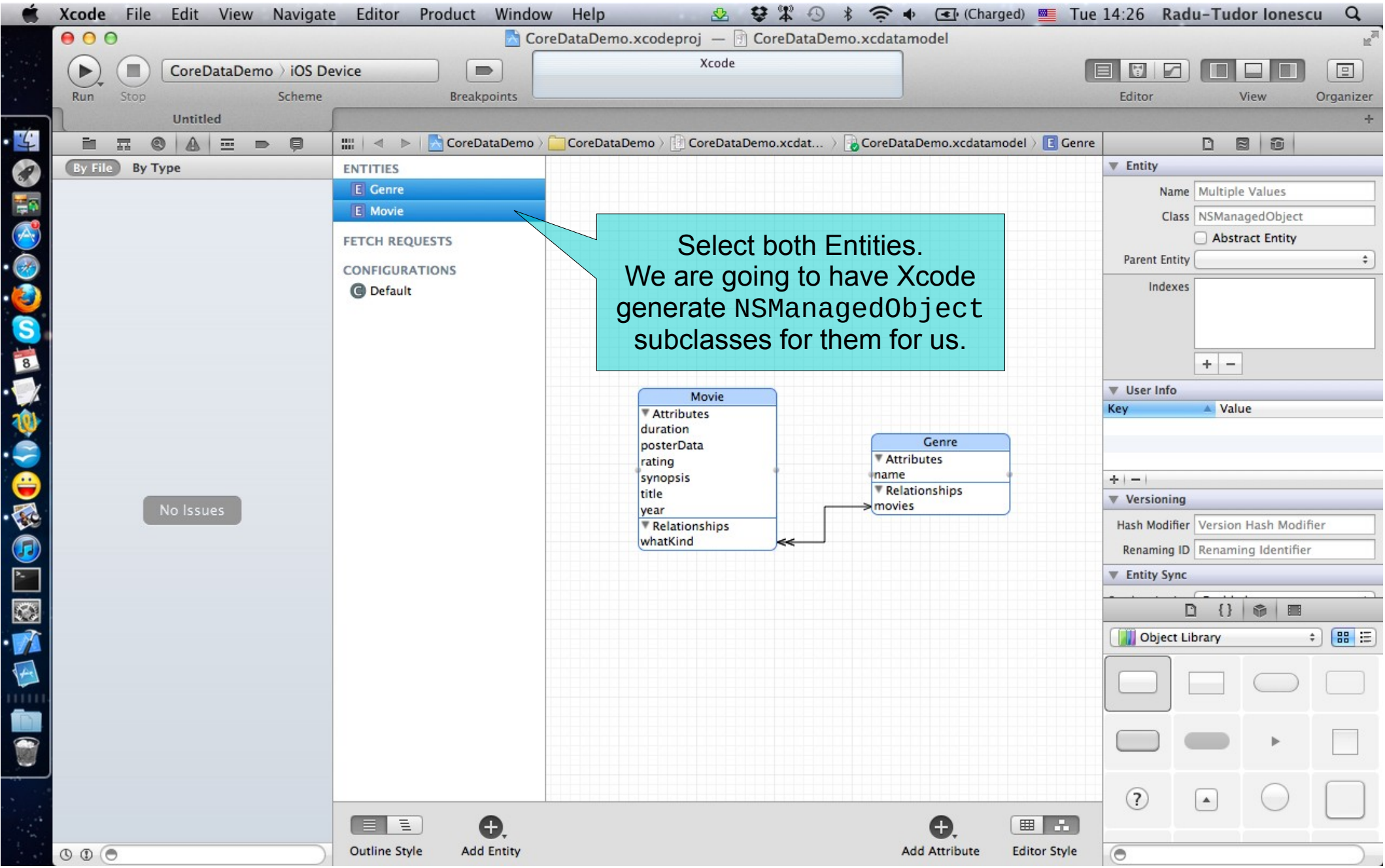
Calling `valueForKey:` and `setValue:forKey:` is pretty messy

- There's no type-checking.
- And you have a lot of literal strings in your code (e.g. `@"posterData"`).

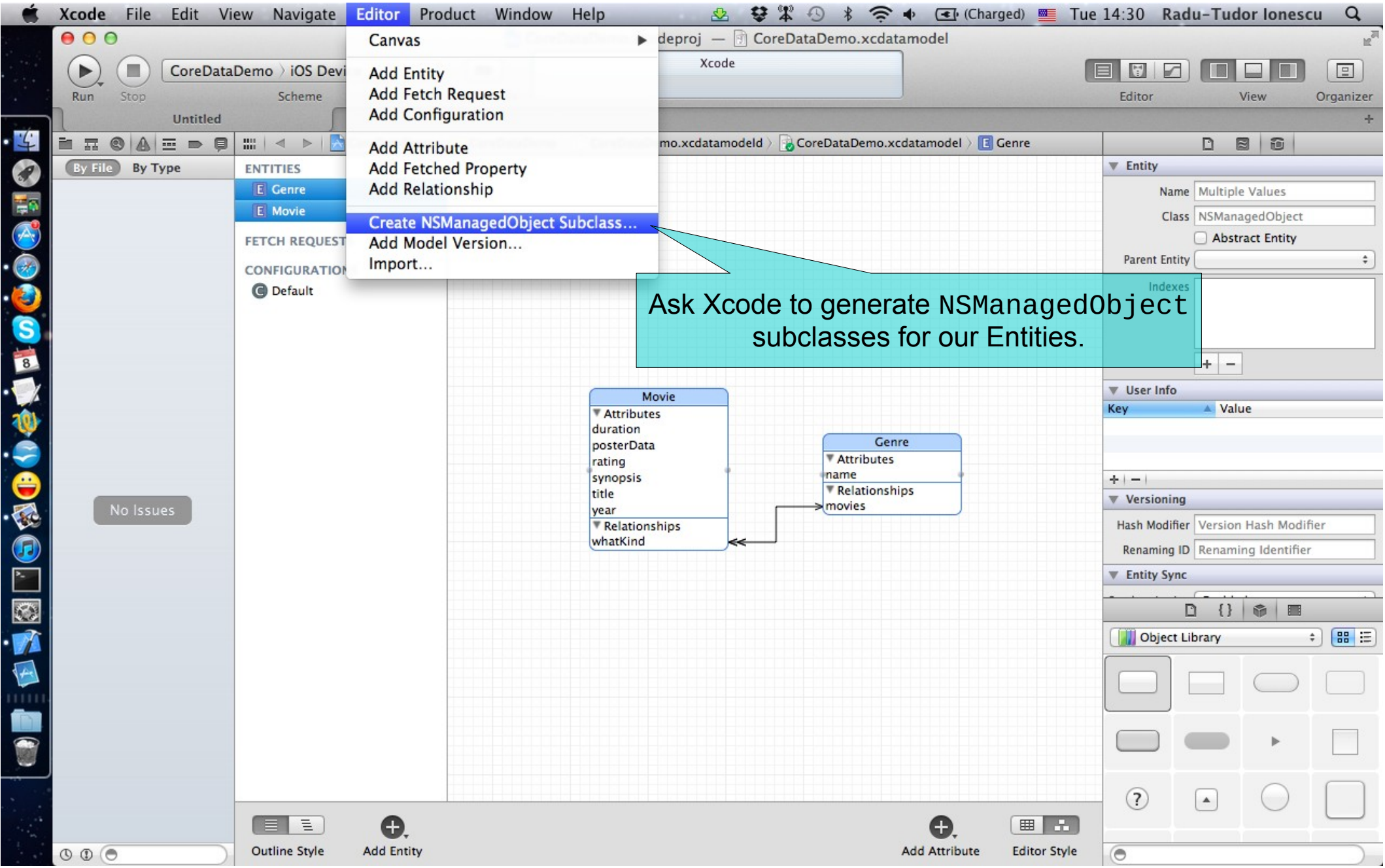
What we really want is to set/get using `@propertys`

- The solution is to create a subclass of `NSManagedObject`.
- The subclass will have `@propertys` for each attribute in the database.
- We name our subclass the same name as the Entity it matches (not strictly required, but it is recommended to do so).
- 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 **be careful** with this).

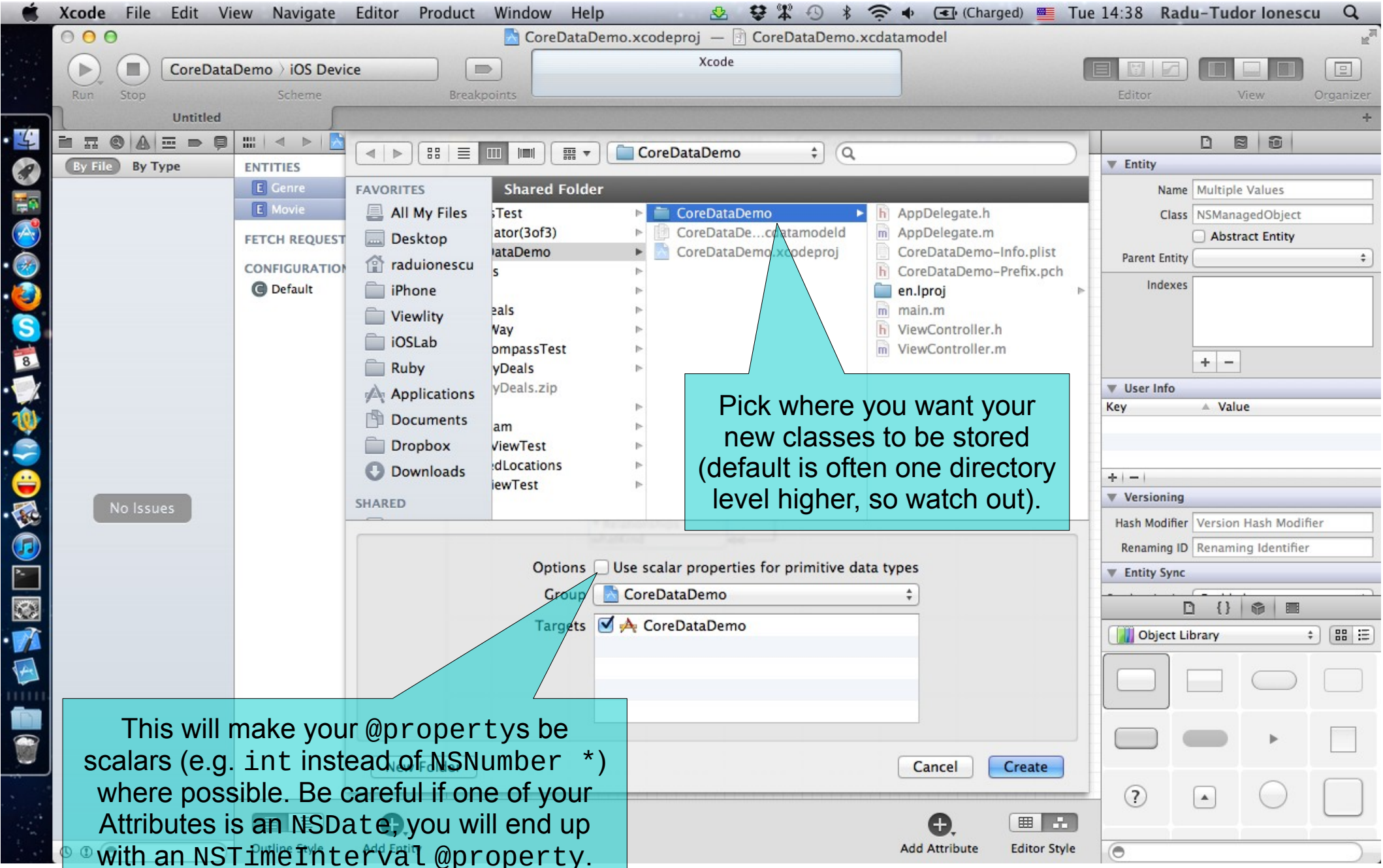
Core Data


Xcode File Edit View Navigate Editor Product Window Help
CoreDataDemo.xcodeproj — CoreDataDemo.xcdatamodel
Run Stop CoreDataDemo › iOS Device Scheme Breakpoints Xcode Editor View Organizer
Untitled
By File By Type
No Issues
ENTITIES
Genre
Movie
FETCH REQUESTS
CONFIGURATIONS
Default
Select both Entities.
We are going to have Xcode generate NSManagedObject subclasses for them for us.
Movie
Attributes
duration
posterData
rating
synopsis
title
year
Relationships
whatKind
Genre
Attributes
name
Relationships
movies
Entity
Name Multiple Values
Class NSManagedObject
Abstract Entity
Parent Entity
Indexes
User Info
Key Value
Versioning
Hash Modifier Version Hash Modifier
Renaming ID Renaming Identifier
Entity Sync
Object Library
Outline Style Add Entity Add Attribute Editor Style

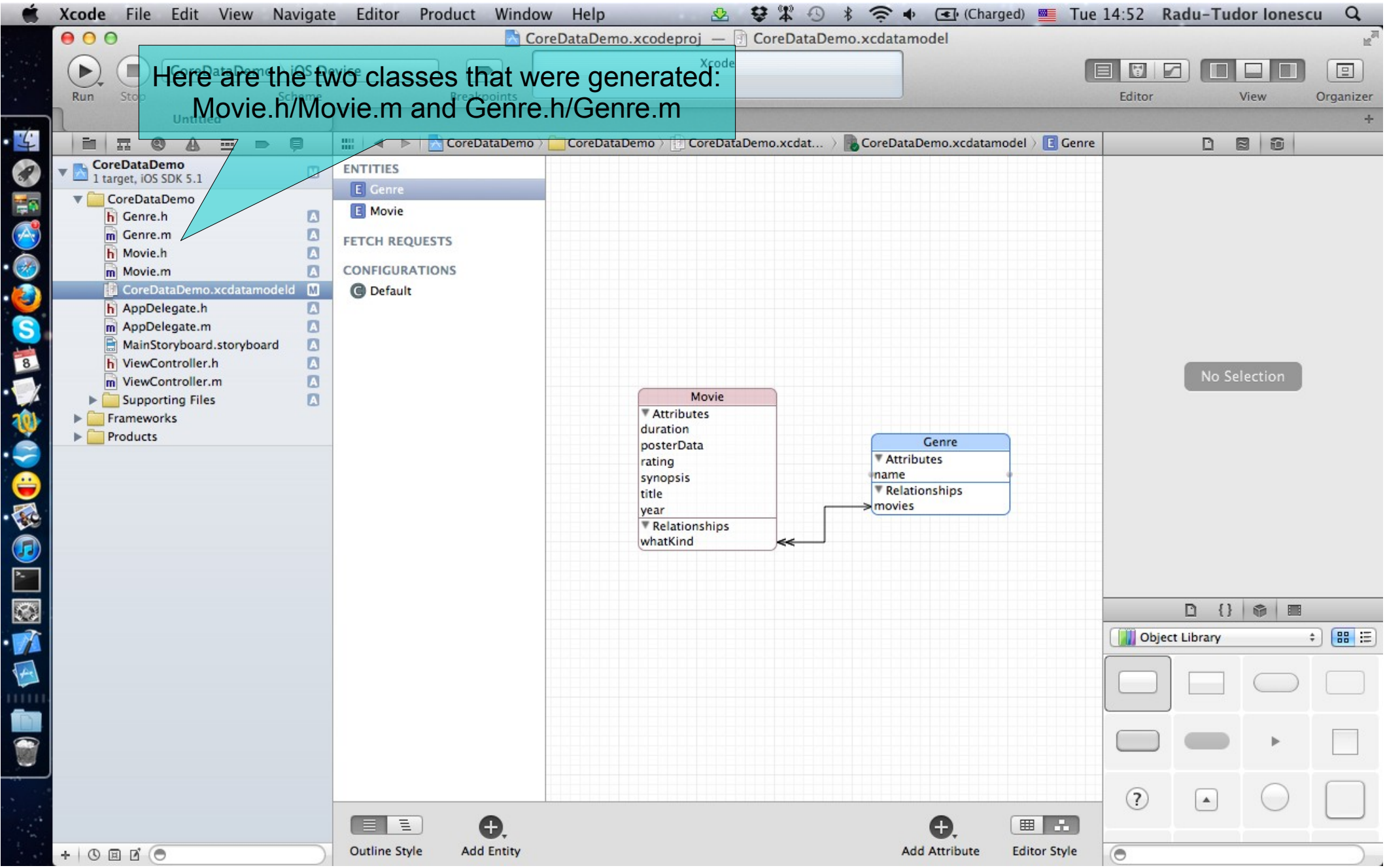
Core Data


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...
Ask Xcode to generate NSManagedObject subclasses for our Entities.
Movie
Attributes
duration
posterData
rating
synopsis
title
year
Relationships
whatKind
Genre
Attributes
name
Relationships
movies
Outline Style
Add Entity
Add Attribute
Editor Style

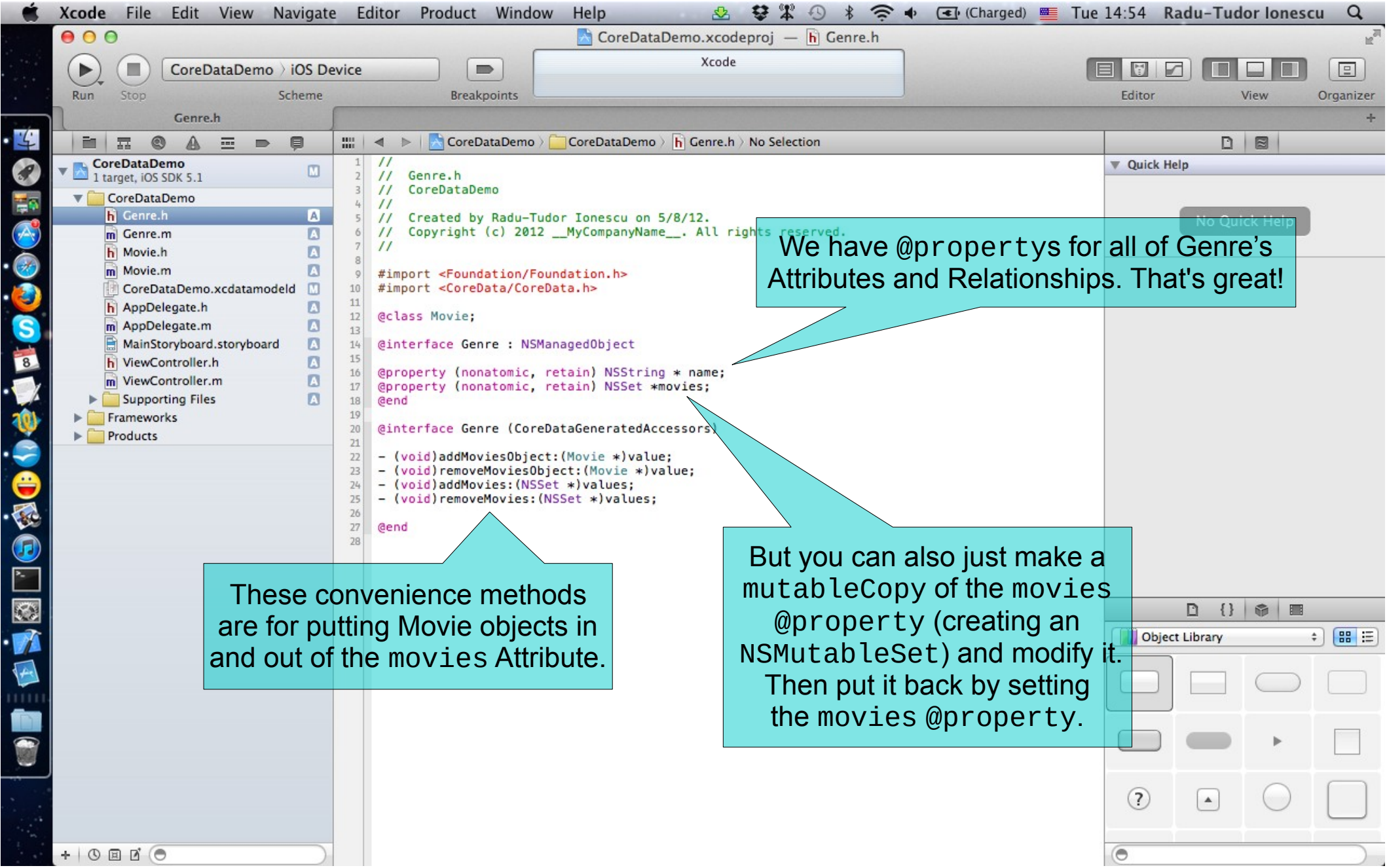
Core Data


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 will end up with an NSTimeInterval @property.
Options
Use scalar properties for primitive data types
Group
CoreDataDemo
Targets
CoreDataDemo
Cancel
Create
Shared Folder
FAVORITES
All My Files
Desktop
raduionescu
iPhone
Viewlity
iOSLab
Ruby
Applications
Documents
Dropbox
Downloads
SHARED
AppDelegate.h
AppDelegate.m
CoreDataDemo-Info.plist
CoreDataDemo-Prefix.pch
en.lproj
main.m
ViewController.h
ViewController.m
Entity
Name
Multiple Values
Class
NSManagedObject
Abstract Entity
Parent Entity
Indexes
User Info
Key
Value
Versioning
Hash Modifier
Version Hash Modifier
Renaming ID
Renaming Identifier
Entity Sync
Object Library
Add Attribute
Editor Style
No Issues

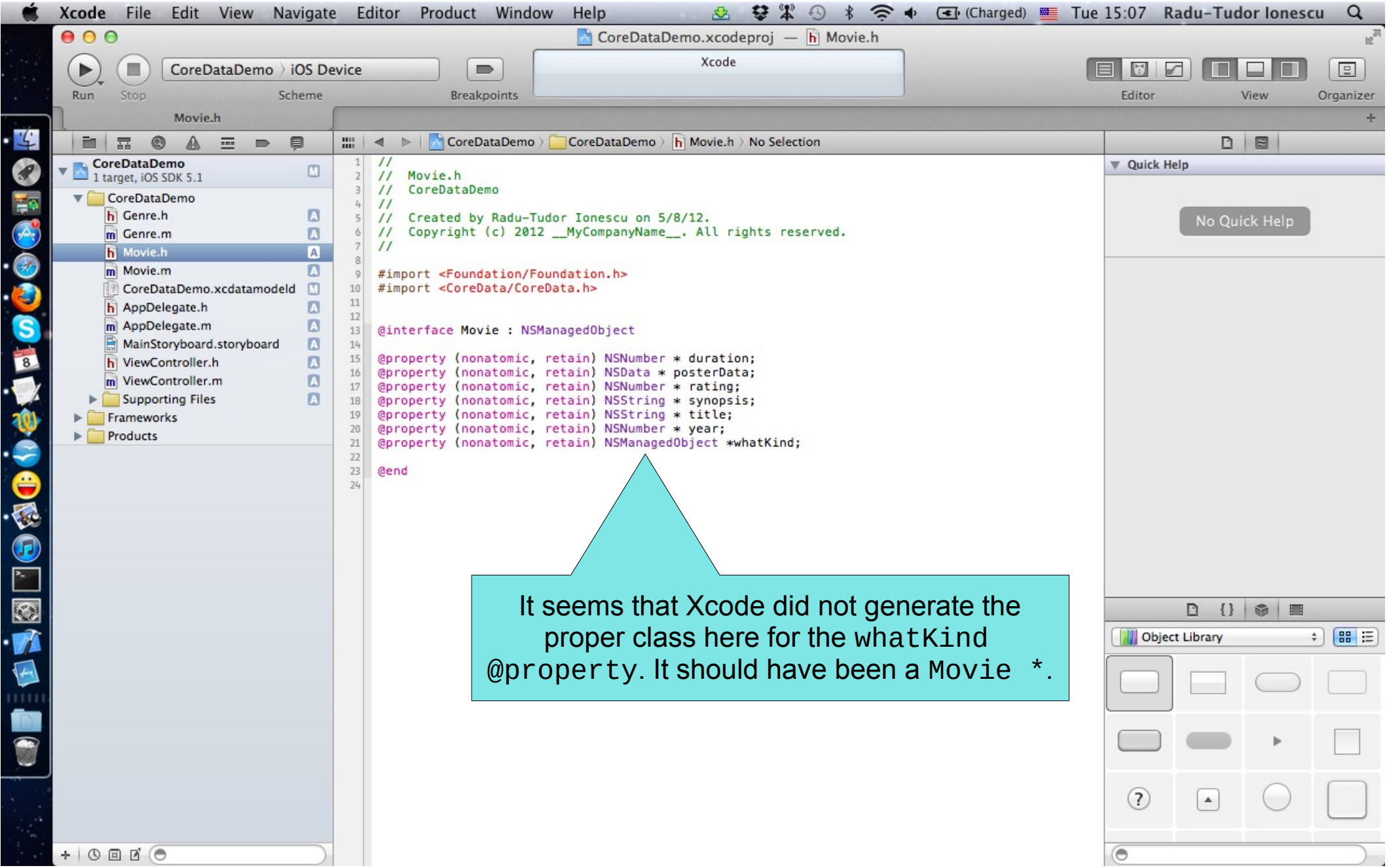
Core Data


Here are the two classes that were generated:
Movie.h/Movie.m and Genre.h/Genre.m
Xcode File Edit View Navigate Editor Product Window Help
CoreDataDemo.xcodeproj — CoreDataDemo.xcdatamodel
Run
Editor
View
Organizer
CoreDataDemo
1 target, iOS SDK 5.1
CoreDataDemo
Genre.h
Genre.m
Movie.h
Movie.m
CoreDataDemo.xcdatamodeld
AppDelegate.h
AppDelegate.m
MainStoryboard.storyboard
ViewController.h
ViewController.m
Supporting Files
Frameworks
Products
ENTITIES
Genre
Movie
FETCH REQUESTS
CONFIGURATIONS
Default
Movie
Attributes
duration
posterData
rating
synopsis
title
year
Relationships
whatKind
Genre
Attributes
name
Relationships
movies
No Selection
Object Library
Outline Style
Add Entity
Add Attribute
Editor Style

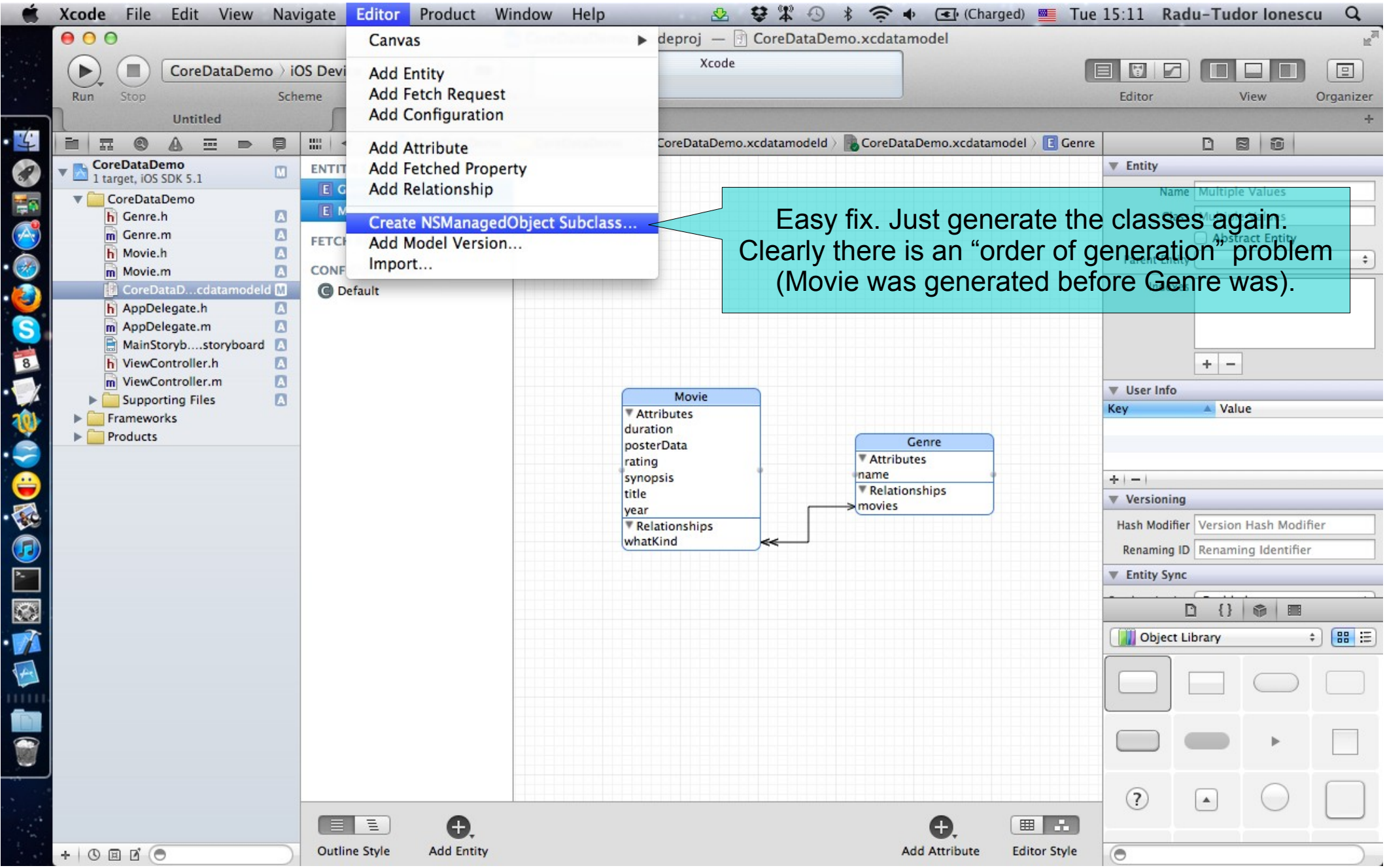
Core Data


Xcode File Edit View Navigate Editor Product Window Help (Charged) Tue 14:54 Radu-Tudor Ionescu
CoreDataDemo.xcodeproj — Genre.h
Run Stop CoreDataDemo › iOS Device Scheme Breakpoints Xcode Editor View Organizer
Genre.h
CoreDataDemo › CoreDataDemo › Genre.h › No Selection
CoreDataDemo
1 target, iOS SDK 5.1
CoreDataDemo
Genre.h
Genre.m
Movie.h
Movie.m
CoreDataDemo.xcdatamodeld
AppDelegate.h
AppDelegate.m
MainStoryboard.storyboard
ViewController.h
ViewController.m
Supporting Files
Frameworks
Products
1 //
2 // Genre.h
3 // CoreDataDemo
4 //
5 // Created by Radu-Tudor Ionescu on 5/8/12.
6 // Copyright (c) 2012 __MyCompanyName__. All rights reserved.
7 //
8
9 #import <Foundation/Foundation.h>
10 #import <CoreData/CoreData.h>
11
12 @class Movie;
13
14 @interface Genre : NSManagedObject
15
16 @property (nonatomic, retain) NSString * name;
17 @property (nonatomic, retain) NSSet *movies;
18 @end
19
20 @interface Genre (CoreDataGeneratedAccessors)
21
22 - (void)addMoviesObject:(Movie *)value;
23 - (void)removeMoviesObject:(Movie *)value;
24 - (void)addMovies:(NSSet *)values;
25 - (void)removeMovies:(NSSet *)values;
26
27 @end
28
Quick Help
No Quick Help
Object Library
We have @propertys for all of Genre's Attributes and Relationships. That's great!
These convenience methods are for putting Movie objects in and out of the movies Attribute.
But you can also just make a mutableCopy of the movies @property (creating an NSMutableSet) and modify it. Then put it back by setting the movies @property.

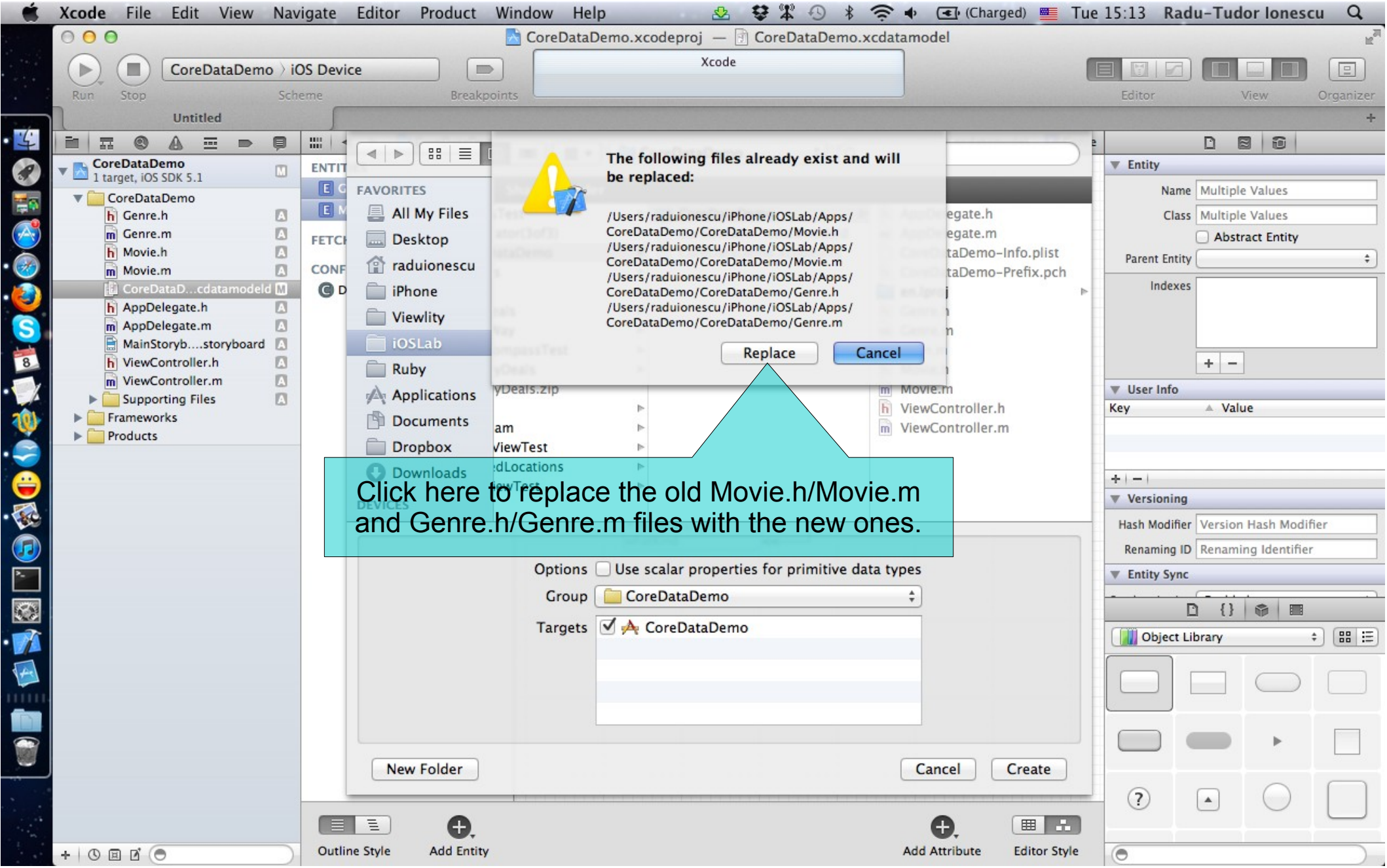
Core Data


Xcode File Edit View Navigate Editor Product Window Help (Charged) Tue 15:07 Radu-Tudor Ionescu
CoreDataDemo.xcodeproj — Movie.h
Run Stop CoreDataDemo › iOS Device Scheme Breakpoints Xcode Editor View Organizer
Movie.h
CoreDataDemo › CoreDataDemo › Movie.h › No Selection
CoreDataDemo
1 target, iOS SDK 5.1
CoreDataDemo
Genre.h
Genre.m
Movie.h
Movie.m
CoreDataDemo.xcdatamodeld
AppDelegate.h
AppDelegate.m
MainStoryboard.storyboard
ViewController.h
ViewController.m
Supporting Files
Frameworks
Products
1 //
2 // Movie.h
3 // CoreDataDemo
4 //
5 // Created by Radu-Tudor Ionescu on 5/8/12.
6 // Copyright (c) 2012 __MyCompanyName__. All rights reserved.
7 //
8
9 #import <Foundation/Foundation.h>
10 #import <CoreData/CoreData.h>
11
12
13 @interface Movie : NSManagedObject
14
15 @property (nonatomic, retain) NSNumber * duration;
16 @property (nonatomic, retain) NSData * posterData;
17 @property (nonatomic, retain) NSNumber * rating;
18 @property (nonatomic, retain) NSString * synopsis;
19 @property (nonatomic, retain) NSString * title;
20 @property (nonatomic, retain) NSNumber * year;
21 @property (nonatomic, retain) NSManagedObject *whatKind;
22
23 @end
24
Quick Help
No Quick Help
Object Library
It seems that Xcode did not generate the proper class here for the whatKind @property. It should have been a Movie *.

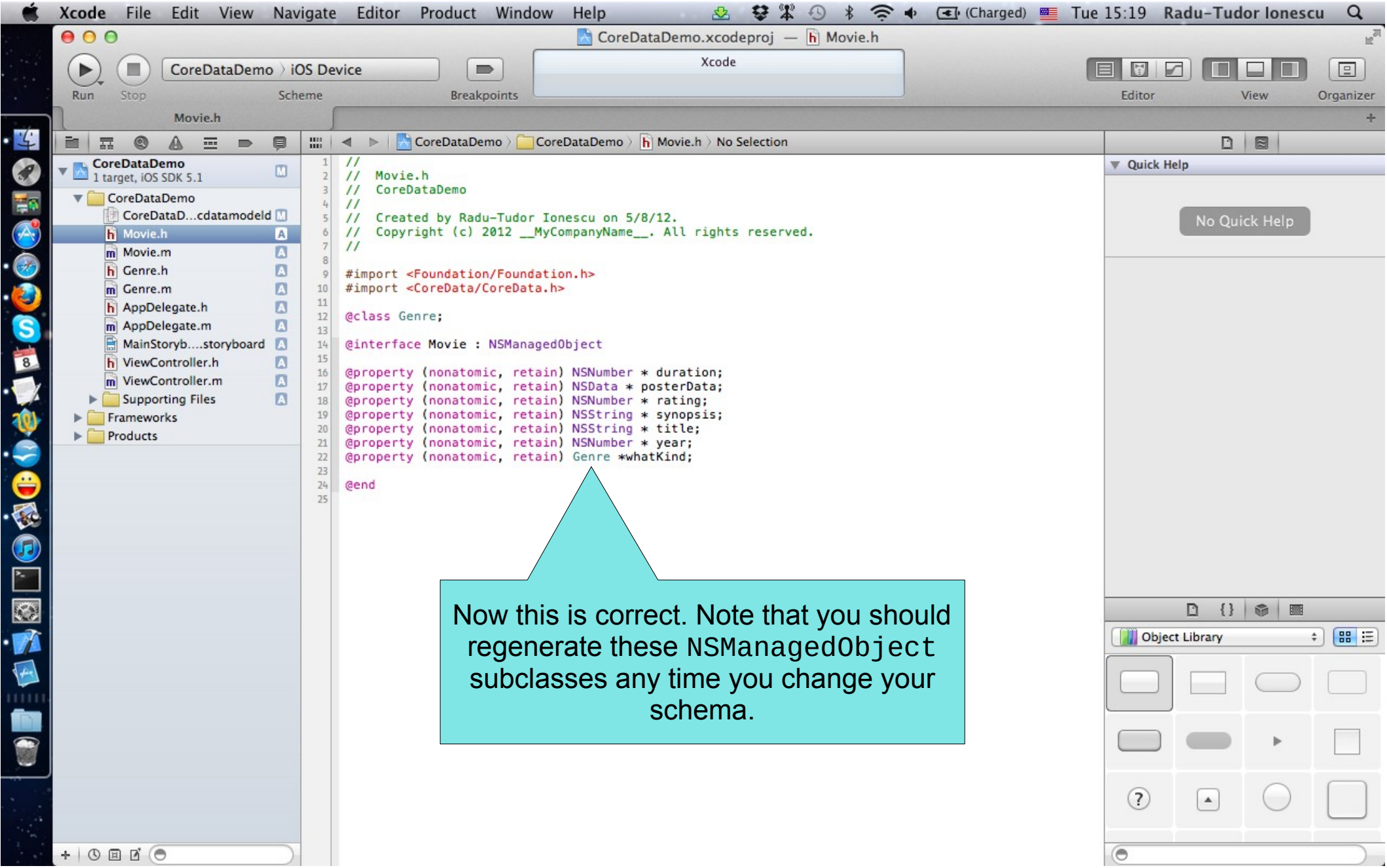
Core Data


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...
Easy fix. Just generate the classes again.
Clearly there is an “order of generation” problem
(Movie was generated before Genre was).
Movie
Attributes
duration
posterData
rating
synopsis
title
year
Relationships
whatKind
Genre
Attributes
name
Relationships
movies
Outline Style
Add Entity
Add Attribute
Editor Style

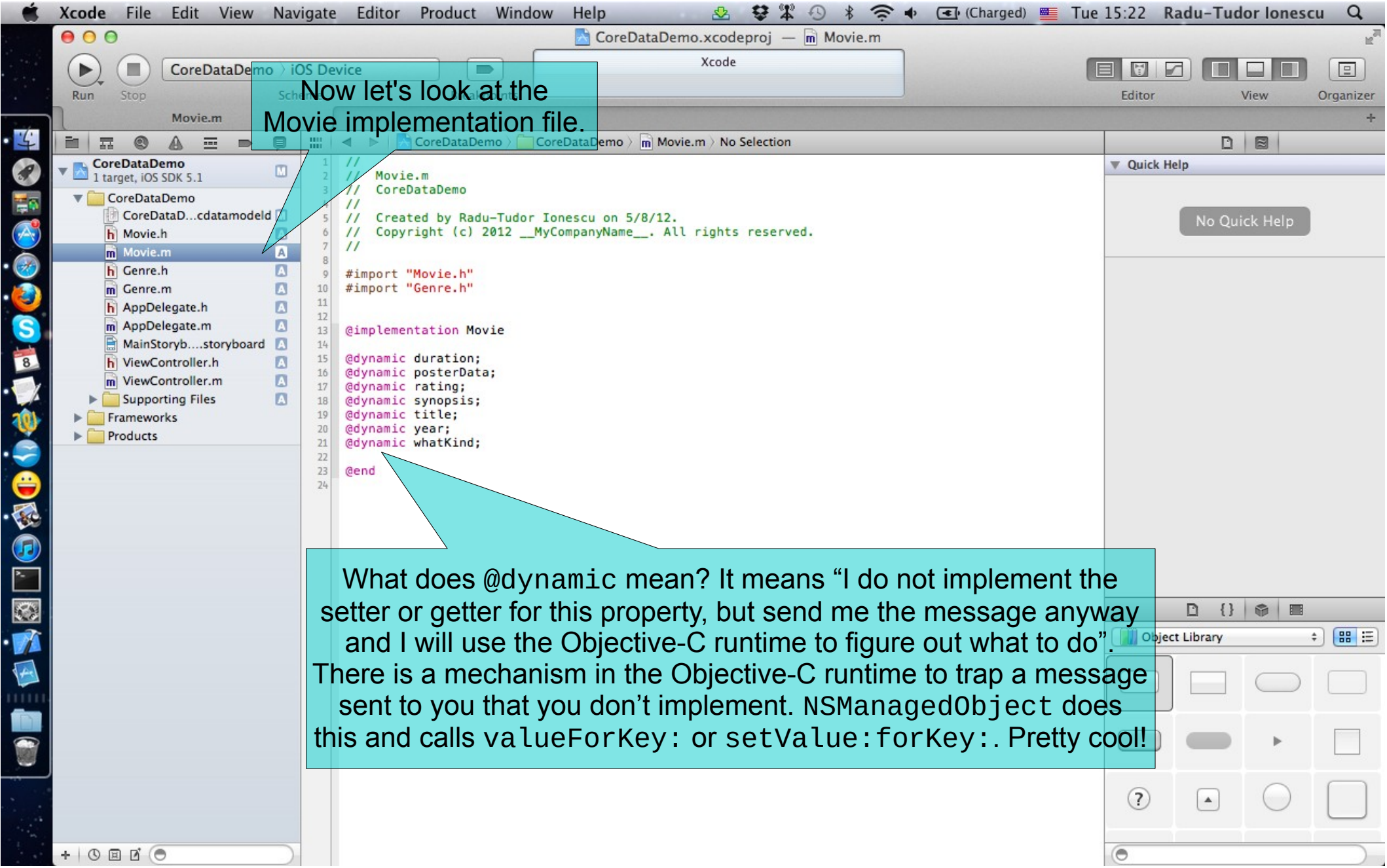
Core Data


The following files already exist and will be replaced:
/Users/raduionescu/iPhone/iOSLab/Apps/CoreDataDemo/CoreDataDemo/Movie.h
/Users/raduionescu/iPhone/iOSLab/Apps/CoreDataDemo/CoreDataDemo/Movie.m
/Users/raduionescu/iPhone/iOSLab/Apps/CoreDataDemo/CoreDataDemo/Genre.h
/Users/raduionescu/iPhone/iOSLab/Apps/CoreDataDemo/CoreDataDemo/Genre.m
Replace
Cancel
Click here to replace the old Movie.h/Movie.m and Genre.h/Genre.m files with the new ones.

Core Data


Xcode File Edit View Navigate Editor Product Window Help (Charged) Tue 15:19 Radu-Tudor Ionescu
CoreDataDemo.xcodeproj — Movie.h
Run Stop CoreDataDemo › iOS Device Scheme Breakpoints Xcode Editor View Organizer
Movie.h
CoreDataDemo › CoreDataDemo › Movie.h › No Selection
CoreDataDemo
1 target, iOS SDK 5.1
CoreDataDemo
CoreDataD...cdatamodeld
Movie.h
Movie.m
Genre.h
Genre.m
AppDelegate.h
AppDelegate.m
MainStoryb....storyboard
ViewController.h
ViewController.m
Supporting Files
Frameworks
Products
1 //
2 // Movie.h
3 // CoreDataDemo
4 //
5 // Created by Radu-Tudor Ionescu on 5/8/12.
6 // Copyright (c) 2012 __MyCompanyName__. All rights reserved.
7 //
8
9 #import <Foundation/Foundation.h>
10 #import <CoreData/CoreData.h>
11
12 @class Genre;
13
14 @interface Movie : NSManagedObject
15
16 @property (nonatomic, retain) NSNumber * duration;
17 @property (nonatomic, retain) NSData * posterData;
18 @property (nonatomic, retain) NSNumber * rating;
19 @property (nonatomic, retain) NSString * synopsis;
20 @property (nonatomic, retain) NSString * title;
21 @property (nonatomic, retain) NSNumber * year;
22 @property (nonatomic, retain) Genre *whatKind;
23
24 @end
25
Quick Help
No Quick Help
Object Library
Now this is correct. Note that you should regenerate these NSManagedObject subclasses any time you change your schema.

Core Data


Now let's look at the Movie implementation file.
What does @dynamic mean? It means “I do not implement the setter or getter for this property, but send me the message anyway and I will 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 setValue:forKey:. Pretty cool!
CoreDataDemo.xcodeproj — Movie.m
CoreDataDemo
Movie.h
Movie.m
Genre.h
Genre.m
AppDelegate.h
AppDelegate.m
ViewController.h
ViewController.m
Supporting Files
Frameworks
Products
// Movie.m
// CoreDataDemo
// Created by Radu-Tudor Ionescu on 5/8/12.
// Copyright (c) 2012 __MyCompanyName__. All rights reserved.
#import "Movie.h"
#import "Genre.h"
@implementation Movie
@dynamic duration;
@dynamic posterData;
@dynamic rating;
@dynamic synopsis;
@dynamic title;
@dynamic year;
@dynamic whatKind;
@end
Quick Help
No Quick Help
Object Library

Core Data

So how do I access my Attributes with dot notation?

- Here are some examples:

```
Movie *movie = [NSEntityDescription
            insertNewObjectForEntityForName:@”Movie”
                     inManagedObjectContext:context];

NSData *posterData = movie.posterData;
UIImage *posterImage = [UIImage
                         imageWithData:posterData];

movie.whatKind = ...;
// a Genre object we created or got by querying

movie.whatKind.name = @”Comedy”;
// multiple dots will follow relationships
```

Core Data

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

- That's a problem because you might want to modify your schema and re-generate the subclasses!
- But it would be really cool to be able to add code (very object-oriented).
- Especially code to create an object and set it up properly.
- Or maybe to derive new `@property`s based on ones in the database (for example, a `UIImage` based on a URL in the database).
- Time to introduce an Objective-C feature called **Categories**.

Categories

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

- Without subclassing it.
- Without even having to have access to the code of the class (for example, its .m file).

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.

Categories

Syntax

- Example: Adding the `AddOn` category to `Movie`.

```
@interface Movie (AddOn)
- (UIImage *)posterImage;
@property (readonly) BOOL isRecommended;
@end
```

- Categories have their own .h and .m files. They are usually named like this: ClassName+PurposeOfExtension.[mh].
- Categories cannot have instance variables, so **no** `@synthesize` allowed in its implementation.

Categories

Implementation

```
@implementation Movie (AddOn)
- (UIImage*)posterImage // is not in the database
{
  return [UIImage imageWithData:self.posterData];
}
- (BOOL)isRecommended // based on rating and year
{
  NSDateFormatter *df = [[NSDateFormatter alloc] init];
  df.dateFormat = @"yyyy";
  NSString *year = [df stringFromDate:[NSDate date]];
  return [self.rating floatValue] > 7.5
      && [self.year intValue] >= [year intValue] - 1;
}
@end
```

- Sometimes we add `@propertys` to an `NSManagedObject` subclass via categories to make accessing `BOOL` attributes (which are `NSNumbers`) cleaner. Or we add `@propertys` to convert `NSData` objects to whatever the bits represent.

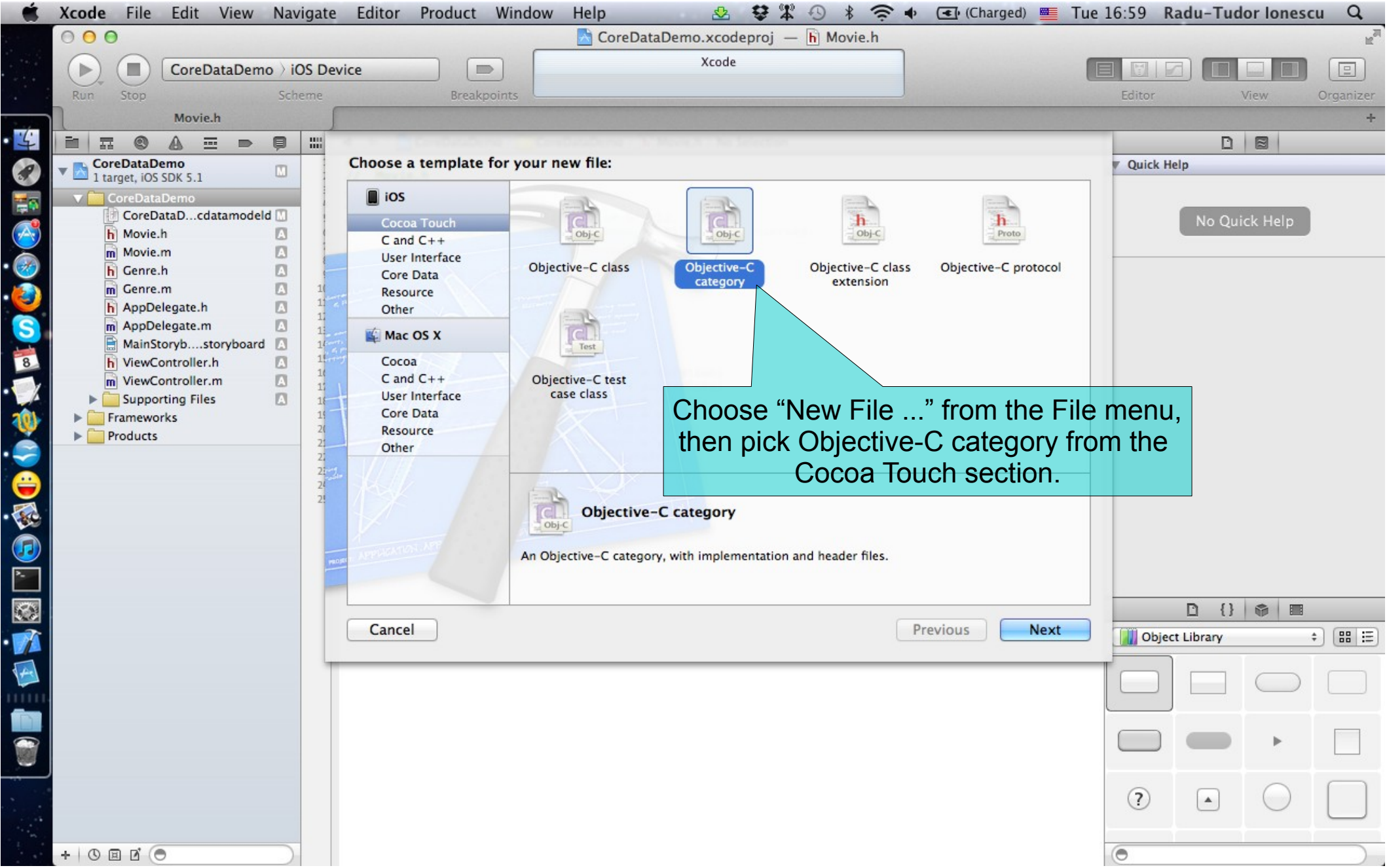
Categories

Most common category on an `NSManagedObject` subclass? Creation

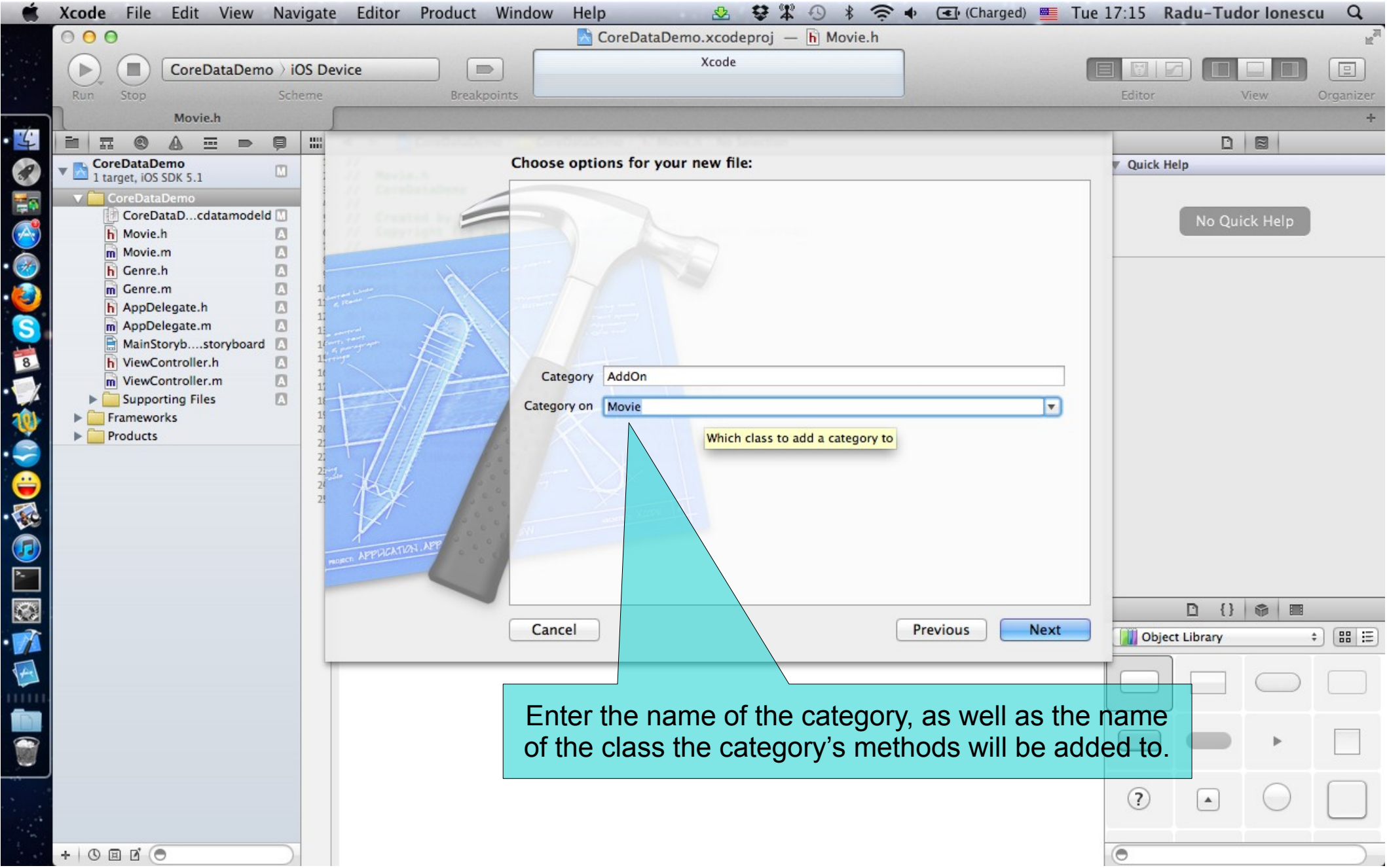
```
@implementation Movie (Create)
+ (Movie *)movieWithData:(NSDictionary *)movieData
  inManagedObjectContext:(NSManagedObjectContext *)context
{
    Movie *movie = ...;
    /* See if a Movie for that data is already in the
     * database. We don't know how to query yet. */
    if (!movie)
    {
        movie = [NSEntityDescription
                  insertNewObjectForEntityForName:@"Movie"
                           inManagedObjectContext:context];
      /* Initialize the movie from the movieData.
       * Perhaps even create other database objects. */
    }
    return movie;
}
@end
```

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

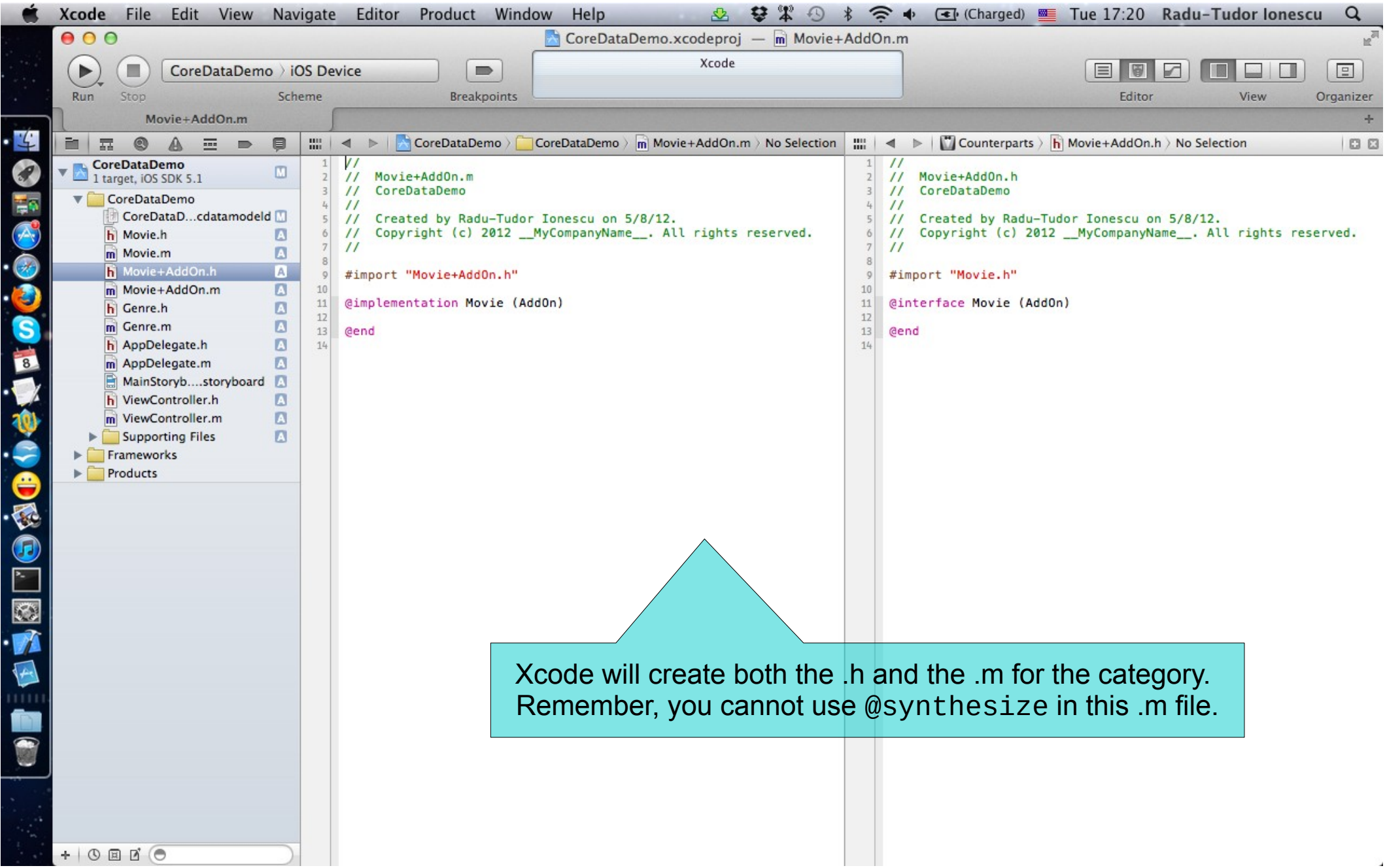
Core Data


Xcode File Edit View Navigate Editor Product Window Help (Charged) Tue 16:59 Radu-Tudor Ionescu
CoreDataDemo.xcodeproj — Movie.h
Run Stop Scheme Breakpoints Xcode Editor View Organizer
CoreDataDemo 〉 iOS Device
Movie.h
CoreDataDemo
1 target, iOS SDK 5.1
CoreDataDemo
CoreDataD...cdatamodeld
Movie.h
Movie.m
Genre.h
Genre.m
AppDelegate.h
AppDelegate.m
MainStoryb....storyboard
ViewController.h
ViewController.m
Supporting Files
Frameworks
Products
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
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.
Cancel
Previous
Next
Choose “New File ...” from the File menu, then pick Objective-C category from the Cocoa Touch section.
Quick Help
No Quick Help
Object Library

Core Data

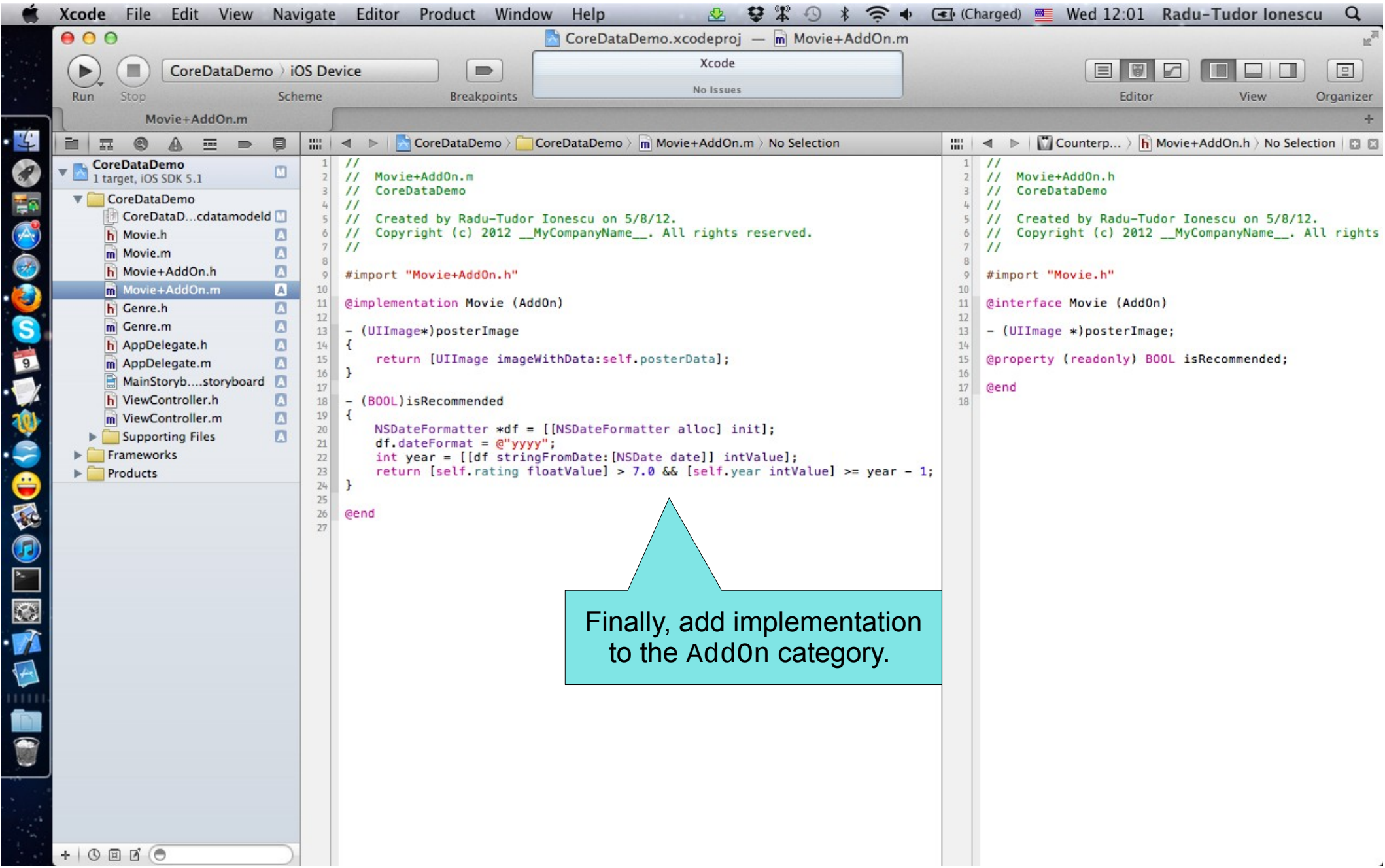

Xcode File Edit View Navigate Editor Product Window Help
Tue 17:15 Radu-Tudor Ionescu
CoreDataDemo.xcodeproj — Movie.h
CoreDataDemo 〉 iOS Device
Run Stop Scheme Breakpoints
Xcode
Editor View Organizer
Movie.h
CoreDataDemo
1 target, iOS SDK 5.1
CoreDataDemo
CoreDataD...cdatamodeld
Movie.h
Movie.m
Genre.h
Genre.m
AppDelegate.h
AppDelegate.m
MainStoryb....storyboard
ViewController.h
ViewController.m
Supporting Files
Frameworks
Products
Choose options for your new file:
Category AddOn
Category on Movie
Which class to add a category to
Cancel
Previous
Next
Quick Help
No Quick Help
Object Library
Enter the name of the category, as well as the name of the class the category's methods will be added to.

Core Data



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

Core Data


Xcode File Edit View Navigate Editor Product Window Help (Charged) Wed 12:01 Radu-Tudor Ionescu
CoreDataDemo.xcodeproj — Movie+AddOn.m
Run Stop CoreDataDemo iOS Device Scheme Breakpoints Xcode No Issues Editor View Organizer
Movie+AddOn.m
CoreDataDemo
1 target, iOS SDK 5.1
CoreDataDemo
CoreDataD...cdatamodeld
Movie.h
Movie.m
Movie+AddOn.h
Movie+AddOn.m
Genre.h
Genre.m
AppDelegate.h
AppDelegate.m
MainStoryb....storyboard
ViewController.h
ViewController.m
Supporting Files
Frameworks
Products
CoreDataDemo › CoreDataDemo › Movie+AddOn.m › No Selection
//
// Movie+AddOn.m
// CoreDataDemo
//
// Created by Radu-Tudor Ionescu on 5/8/12.
// Copyright (c) 2012 __MyCompanyName__. All rights reserved.
//
#import "Movie+AddOn.h"
@implementation Movie (AddOn)
- (UIImage*)posterImage
{
return [UIImage imageWithData:self.posterData];
}
- (BOOL)isRecommended
{
NSDateFormatter *df = [[NSDateFormatter alloc] init];
df.dateFormat = @"yyyy";
int year = [[df stringFromDate:[NSDate date]] intValue];
return [self.rating floatValue] > 7.0 && [self.year intValue] >= year - 1;
}
@end
Counterp... › Movie+AddOn.h › No Selection
//
// Movie+AddOn.h
// CoreDataDemo
//
// Created by Radu-Tudor Ionescu on 5/8/12.
// Copyright (c) 2012 __MyCompanyName__. All rights
//
#import "Movie.h"
@interface Movie (AddOn)
- (UIImage *)posterImage;
@property (readonly) BOOL isRecommended;
@end
Finally, add implementation to the AddOn category.

Deletion

- Deleting objects from the database is easy:

```
[self.doc.managedObjectContext deleteObject:movie];
```

- 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 `movie` after you delete it!
- Here is another method we sometimes put in a category of an `NSManagedObject` subclass:

```
@implementation Movie (Deletion)
- (void)prepareForDeletion
{
    self.whatKind.movieCount--;
}
@end
```

We don't need to set our `whatKind` to `nil` or anything here (that will happen automatically). But if Genre had a "number of movies" attribute, we might adjust it down by one here.

Core Data

What do you know so far?

- **Create** objects in the database with:

```
insertNewObjectForEntityForName:inManagedObjectContext:
```

- **Get** or **set** properties with `valueForKey:` or `setValue:forKey:`. Or using `@property`s in a custom subclass.
- **Delete** objects in the database using the `deleteObject:` method of the `NSManagedObjectContext`.

Core Data

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. `NSSortDescriptors` 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, default is all of them).

Querying

Creating an NSFetchRequest

- We will consider each of these lines of code one by one:

```
NSFetchRequest *request = [NSFetchRequest
                    fetchRequestWithEntityName:@”Movie”];
request.fetchBatchSize = 20;
request.fetchLimit = 100;
request.predicate = ...;
request.sortDescriptors = [NSArray
                        arrayWithObject:sortDescriptor];
```

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 Movies and some Genres (one or the other).

Setting fetch sizes/limits

- If you created a fetch that would match 500 objects, the request above faults 20 at a time. And it would stop fetching after it had fetched 100 of the 500.

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:@"title"
                                     ascending:YES
                              selector:@selector(caseInsensitiveCompare:)];
```

- There's another version with no `selector:` argument (default is the method `compare:`). The `selector:` argument is just a method sent to each object to compare it to others.
- Some of these "methods" might happen on the database side.
- We give a list of these to the `NSFetchRequest` because sometimes we want to sort first by one key, then sort by another (e.g. `lastName`, `firstName`).

NSPredicate

NSPredicate

- You use predicates to represent logical conditions.
- This is the basis 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.
- Example:

```
NSString *series = @“Harry Potter”;

NSPredicate *predicate = [NSPredicate
  predicateWithFormat:@“title contains %@”, series];
```

NSPredicate

Other examples

- Unique movie in the database:

```
@“uniqueId = %@”, [movieData objectForKey:@”id”]
```

- Matches title case insensitively:

```
@“title contains[c] %@”, (NSString *)
```

- If we had the Date of the release of a Movie in the data mapping:

```
@“releaseDate > %@”, [NSDate date]
```

- Movie search by Genre:

```
@“whatKind.name = %@”, (NSString *)
```

- Genre search (not a Movie search here):

```
@“any movies.title contains %@”, (NSString *)
```

- Many more options. Look at the `NSPredicate` class documentation.

NSPredicate

Combined predicates

- You can use AND and OR inside a predicate string:

```
@"(year = %@) OR (title = %@)" // same with ||
@"(year = %@) && (title = %@)" // same with AND
```

- Or you can use the alternative to 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 Genres:

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

- That have movies with a rating greater than 8:

```
request.predicate = [NSPredicate
  predicateWithFormat:@“any movies.rating > %@”, 8];
```

- Sorted by the Genre's name:

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

Querying

Executing the fetch

- Use the `executeFetchRequest:` method:

```
NSManagedObjectContext *managedObjectContext =
                                    self.doc.managedObjectContext;
NSError *error;
NSArray *genres =
  [managedObjectContext 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 `NSManagedObjects` (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 (Genre *genre in genres)
{
    NSLog(@“fetched genre %@”, genre);
}
```

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

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

Faulting

Faulting

- But if you did this:

```
for (Genre *genre in genres)
{
    NSLog(@“fetched genre named %@”, genre.name);
}
```

Then you would definitely fault all the Genres in from the database.

There is so much more (that we don’t have time to talk about)

- Optimistic locking (`deleteConflictsForObject:`).
- Rolling back unsaved changes.
- Undo and redo changes.

What should you study next?

- Modal View Controllers
- Core Motion (gyro, accelerometer, magnetometer)

  Measuring the device's movement.
- `UITextField`, `UITextView`, `UIActionSheet`
- `UIView` Animation
- `UIImagePickerController`

  Getting images from the camera or photo library.
- `NSTimer`

  Perform scheduled tasks on the main thread.
- iPad and Universal Applications

  There are specific Navigation and View Controllers.
- Open GL ES