Stanford CS193p

Developing Applications for iOS
Spring 2016

Today

- Core Data
  - Object-Oriented Database

Core Data

- Database

  Sometimes you need to store large amounts of data or query it in a sophisticated manner.

  But we still want it to be object-oriented!

- Enter Core Data

  Object-oriented database.

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

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

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

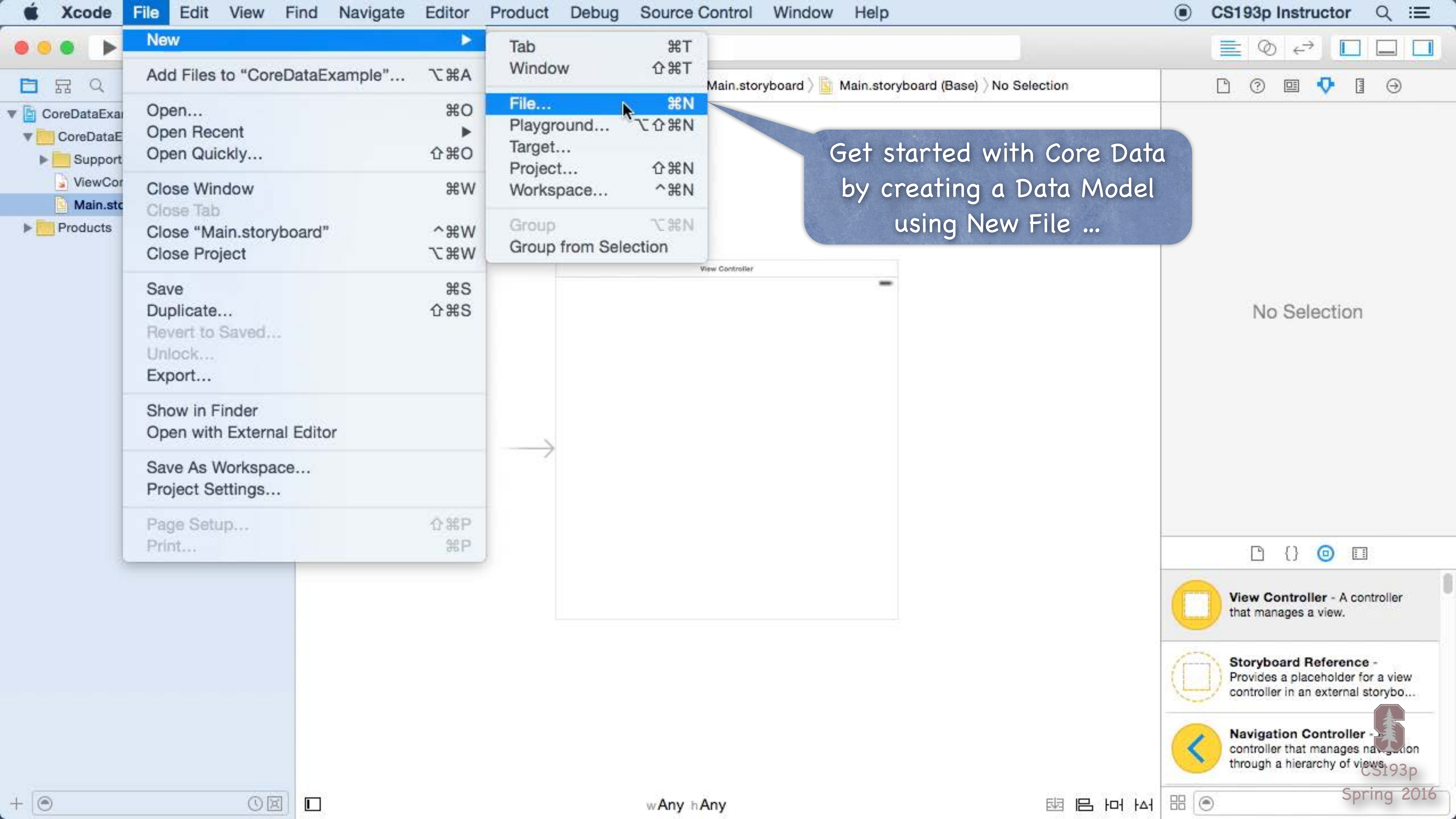
- How does it work?

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

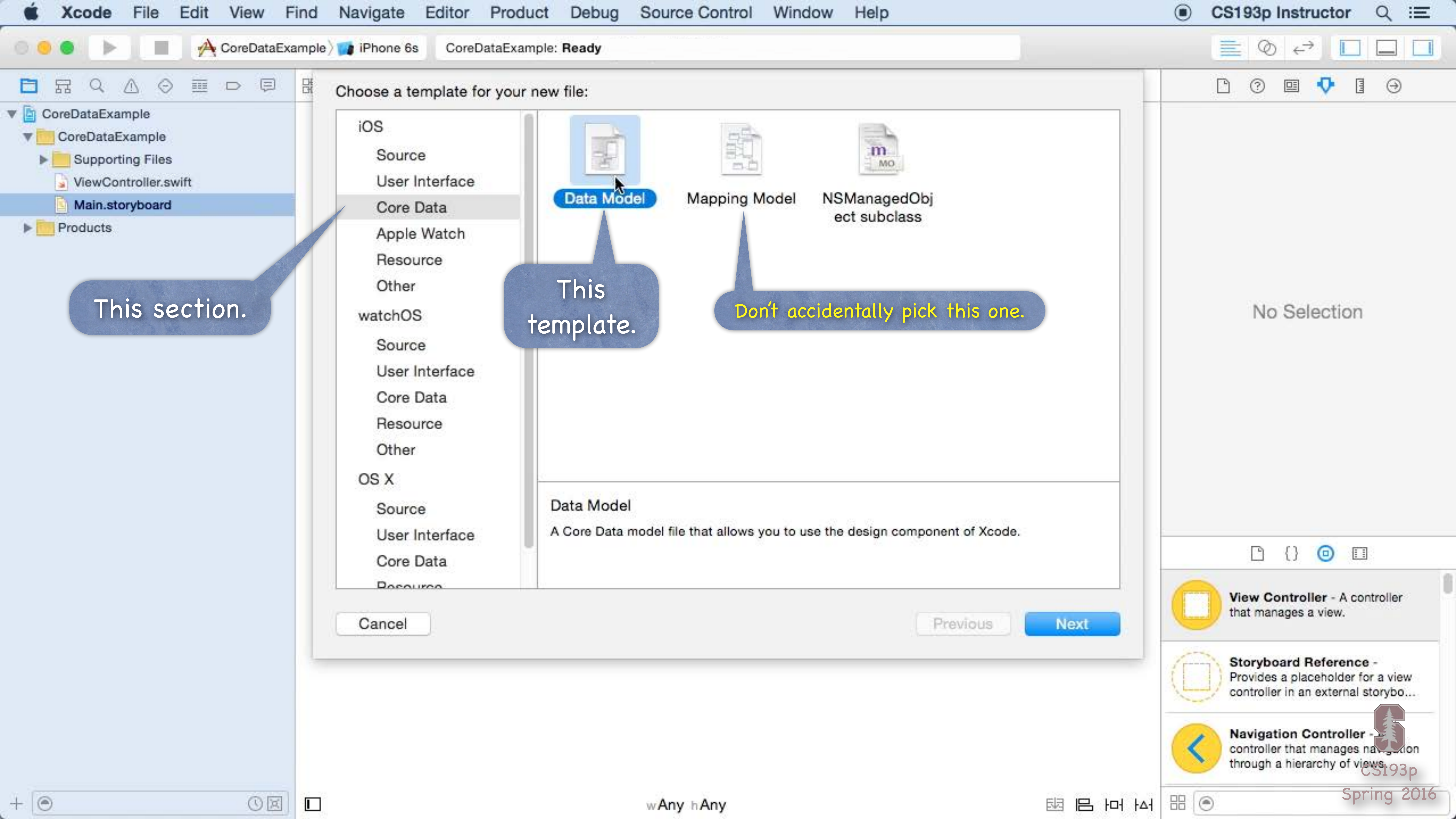
  Create and query for objects using object-oriented API.

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

  Let's get started by creating that visual map …



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



Choose a template for your new file:

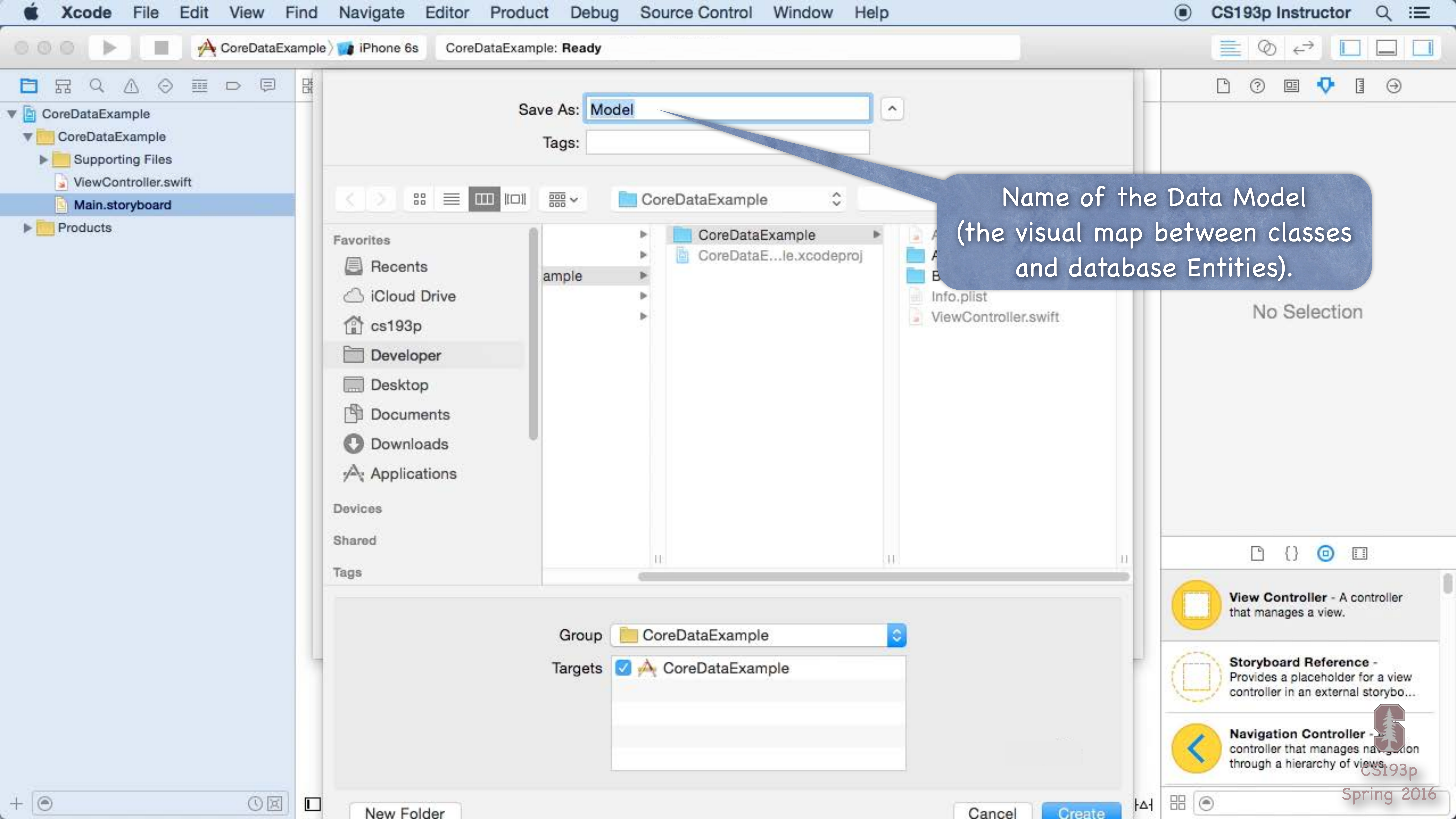
Data Model

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

View Controller - A controller that manages a view.

Storyboard Reference - Provides a placeholder for a view controller in an external storybo...

Navigation Controller - A controller that manages navigation through a hierarchy of views.



Save As: Model

Tags:

Group: CoreDataExample

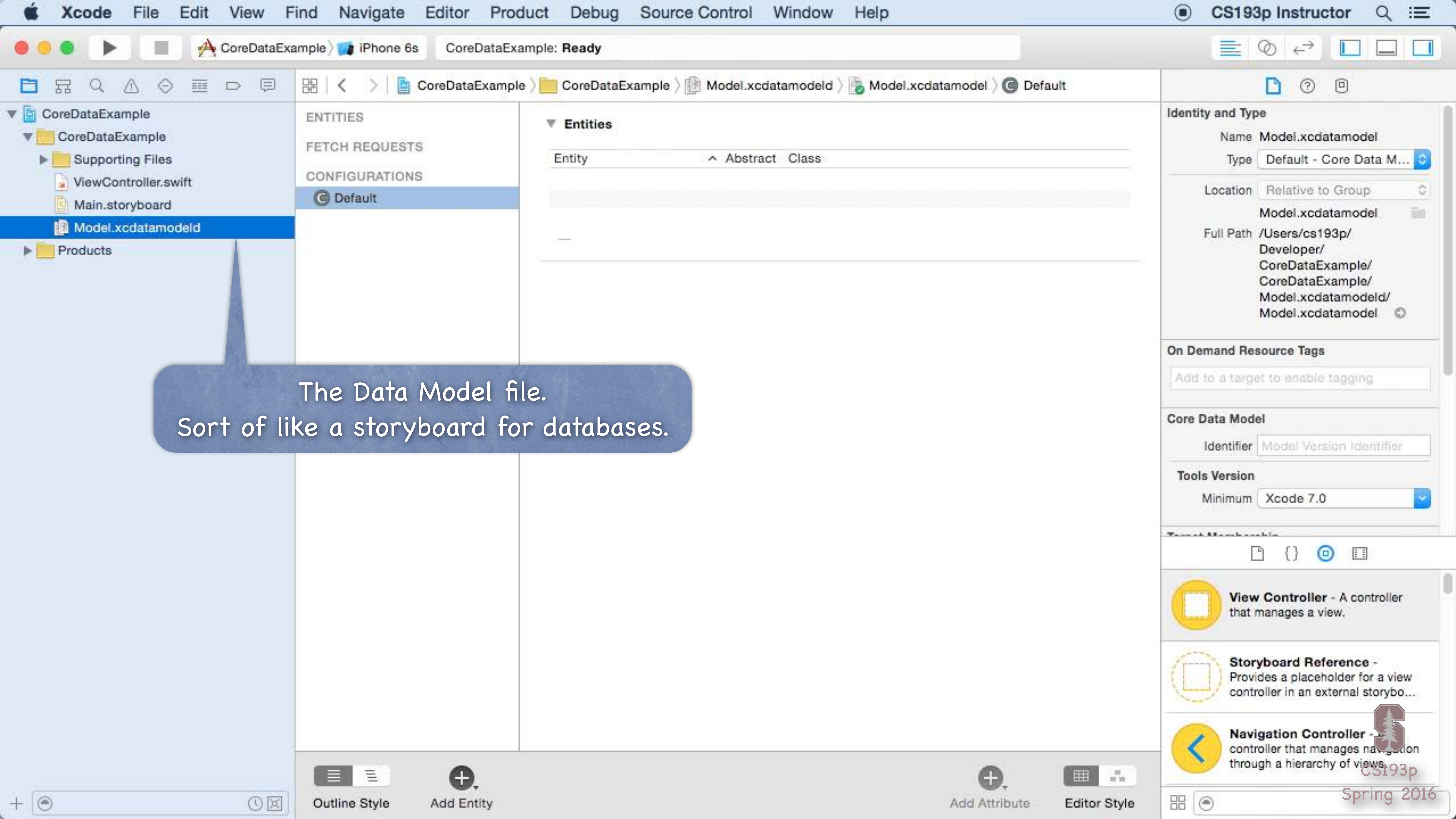
Targets: ✓ CoreDataExample

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

View Controller - A controller that manages a view.

Storyboard Reference - Provides a placeholder for a view controller in an external storybo...

Navigation Controller - A controller that manages navigation through a hierarchy of views.



Name Model.xcdatamodel
Type Default - Core Data M...
Location Relative to Group
Model.xcdatamodel
Full Path /Users/cs193p/Developer/CoreDataExample/CoreDataExample/Model.xcdatamodeld/Model.xcdatamodel

Add to a target to enable tagging

Identifier Model Version Identifier

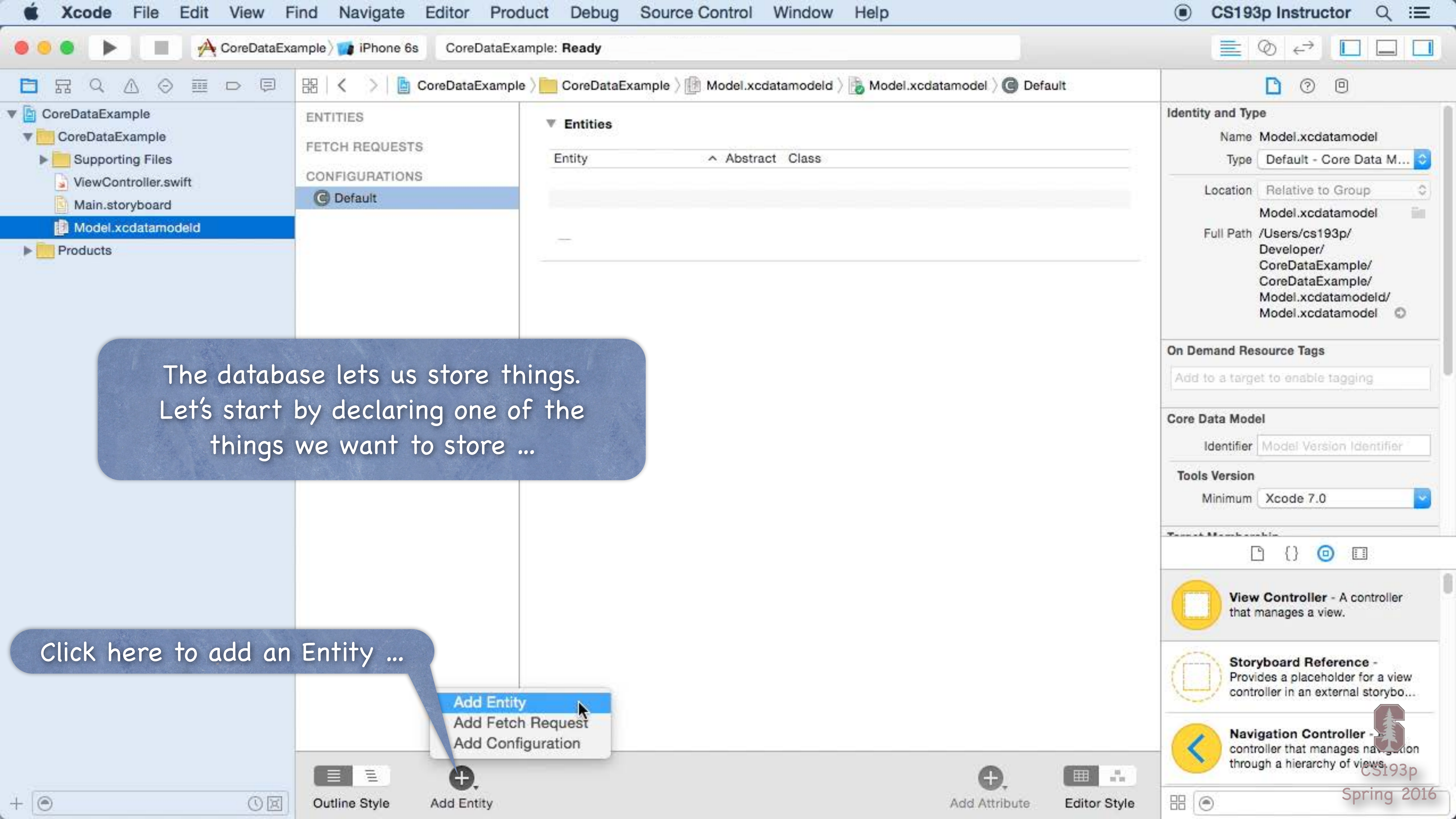
Minimum Xcode 7.0

View Controller - A controller that manages a view.

Storyboard Reference - Provides a placeholder for a view controller in an external storybo...

Navigation Controller - A controller that manages navigation through a hierarchy of views.

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



The database lets us store things.
Let's start by declaring one of the things we want to store ...

Click here to add an Entity ...

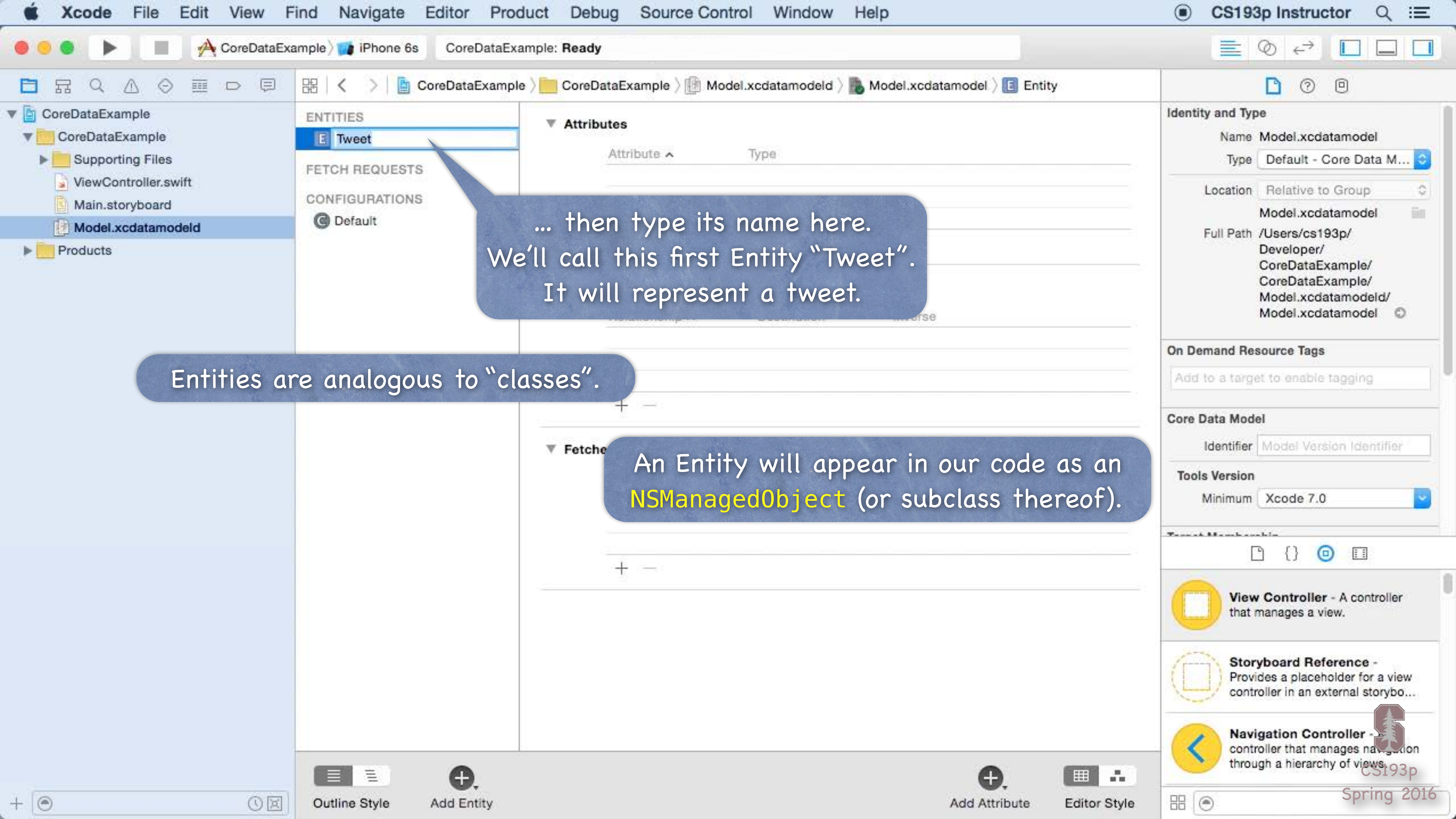
Identity and Type
Name Model.xcdatamodel
Type Default - Core Data M...
Location Relative to Group
Model.xcdatamodel
Full Path /Users/cs193p/Developer/CoreDataExample/CoreDataExample/Model.xcdatamodeld/Model.xcdatamodel

On Demand Resource Tags
Add to a target to enable tagging

Core Data Model
Identifier Model Version Identifier

Tools Version
Minimum Xcode 7.0

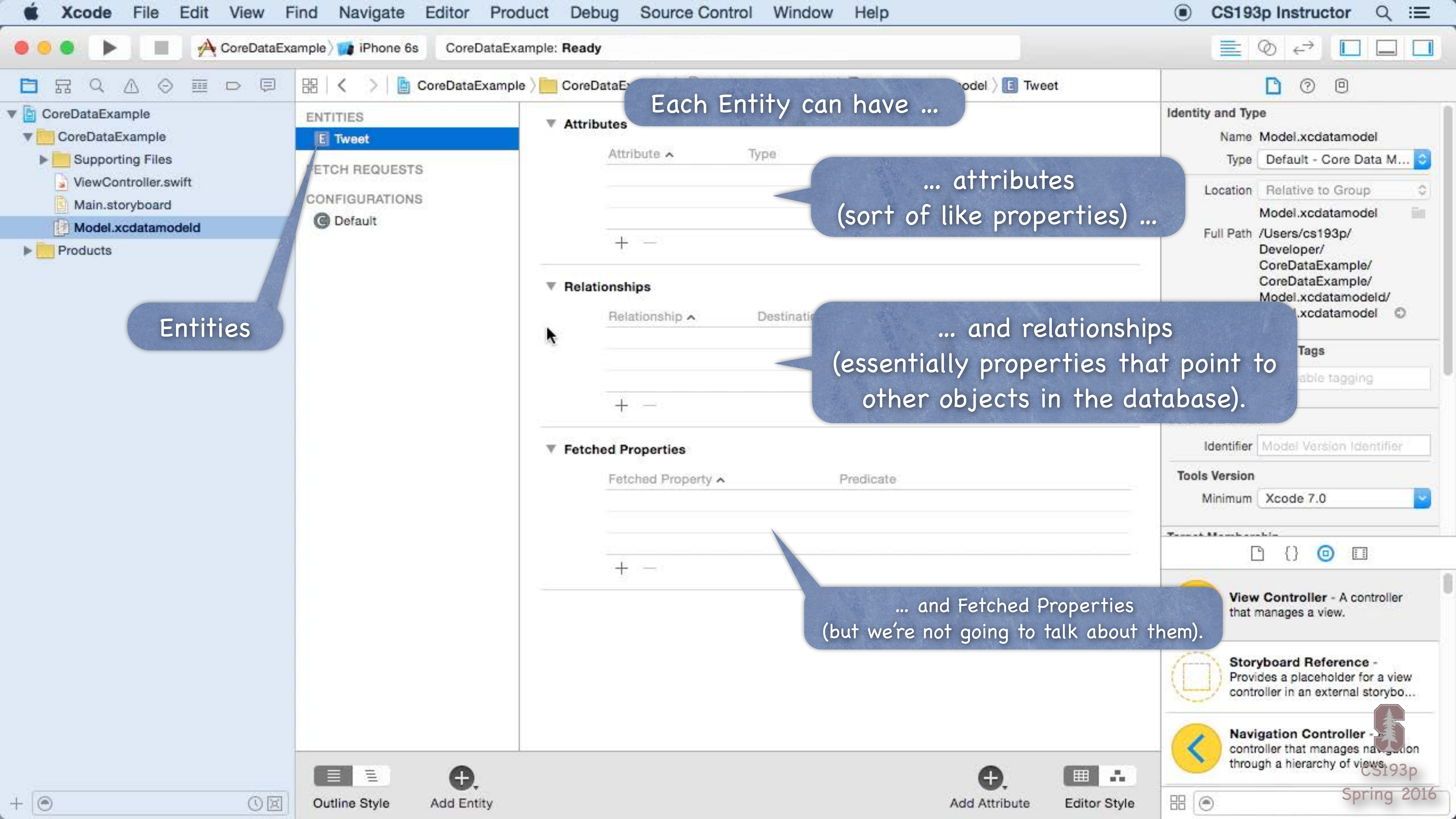
View Controller - A controller that manages a view.
Storyboard Reference - Provides a placeholder for a view controller in an external storybo...
Navigation Controller - A controller that manages navigation through a hierarchy of views.


Xcode File Edit View Find Navigate Editor Product Debug Source Control Window Help
CS193p Instructor
CoreDataExample iPhone 6s
CoreDataExample: Ready
CoreDataExample
CoreDataExample
Supporting Files
ViewController.swift
Main.storyboard
Model.xcdatamodeld
Products
ENTITIES
Tweet
FETCH REQUESTS
CONFIGURATIONS
Default
CoreDataExample CoreDataExample Model.xcdatamodeld Model.xcdatamodel Entity
Attributes
Attribute
Type
Fetche
Identity and Type
Name Model.xcdatamodel
Type Default - Core Data M...
Location Relative to Group
Model.xcdatamodel
Full Path /Users/cs193p/Developer/CoreDataExample/CoreDataExample/Model.xcdatamodeld/Model.xcdatamodel
On Demand Resource Tags
Add to a target to enable tagging
Core Data Model
Identifier Model Version Identifier
Tools Version
Minimum Xcode 7.0
View Controller - A controller that manages a view.
Storyboard Reference - Provides a placeholder for a view controller in an external storybo...
Navigation Controller - controller that manages navigation through a hierarchy of views.
Outline Style
Add Entity
Add Attribute
Editor Style

… then type its name here.
We'll call this first Entity "Tweet".
It will represent a tweet.

Entities are analogous to "classes".

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


Xcode File Edit View Find Navigate Editor Product Debug Source Control Window Help
CS193p Instructor
CoreDataExample iPhone 6s
CoreDataExample: Ready
CoreDataExample
CoreDataExample
Supporting Files
ViewController.swift
Main.storyboard
Model.xcdatamodeld
Products
ENTITIES
Tweet
FETCH REQUESTS
CONFIGURATIONS
Default
Attributes
Attribute
Type
Relationships
Relationship
Destination
Fetched Properties
Fetched Property
Predicate
Outline Style
Add Entity
Add Attribute
Editor Style
Identity and Type
Name Model.xcdatamodel
Type Default - Core Data M...
Location Relative to Group
Model.xcdatamodel
Full Path /Users/cs193p/Developer/CoreDataExample/CoreDataExample/Model.xcdatamodeld/Model.xcdatamodel
Tags
Identifier Model Version Identifier
Tools Version
Minimum Xcode 7.0
View Controller - A controller that manages a view.
Storyboard Reference - Provides a placeholder for a view controller in an external storyboard...
Navigation Controller - A controller that manages navigation through a hierarchy of views.

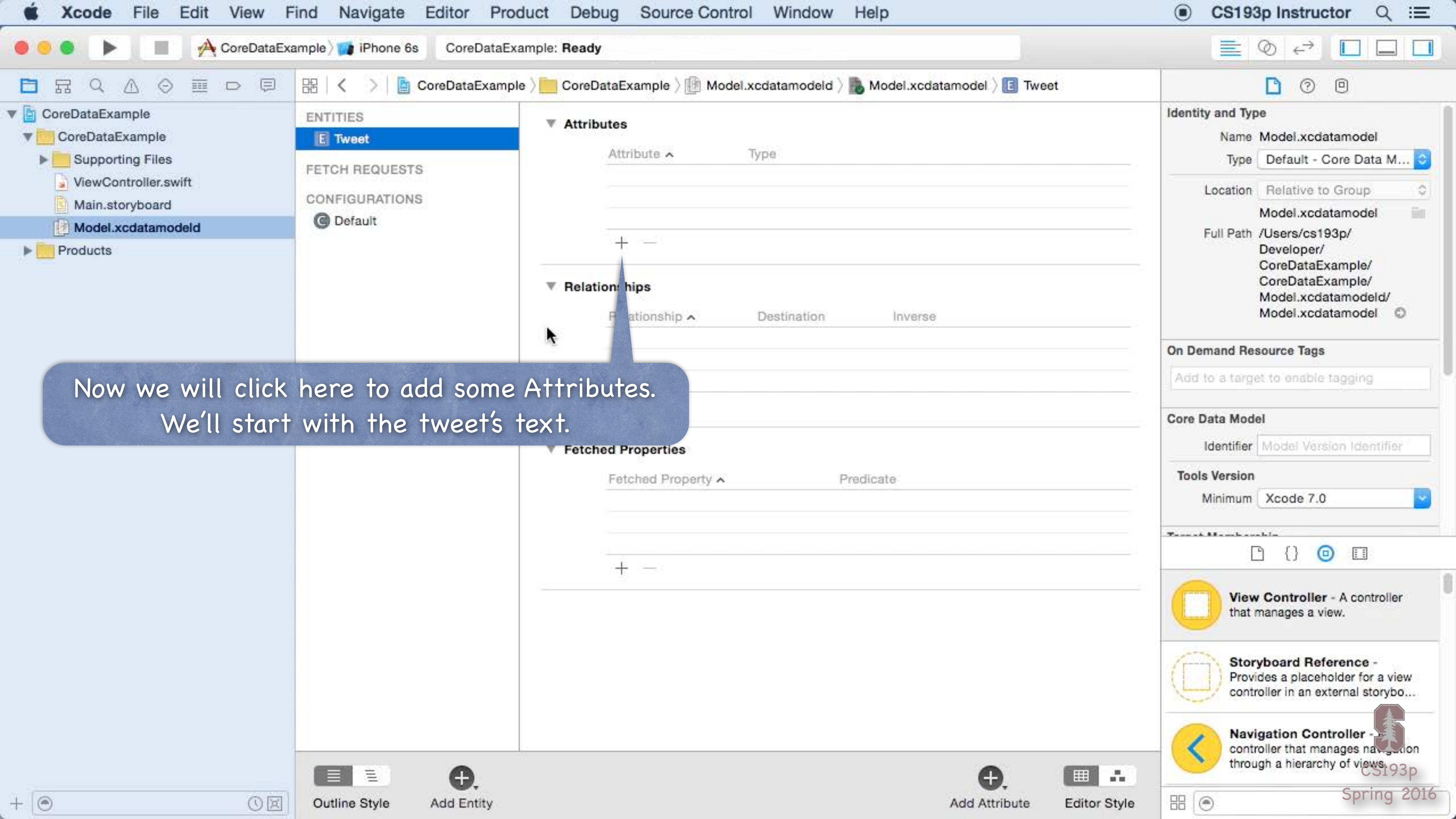
Each Entity can have …

… attributes (sort of like properties) …

… and relationships (essentially properties that point to other objects in the database).

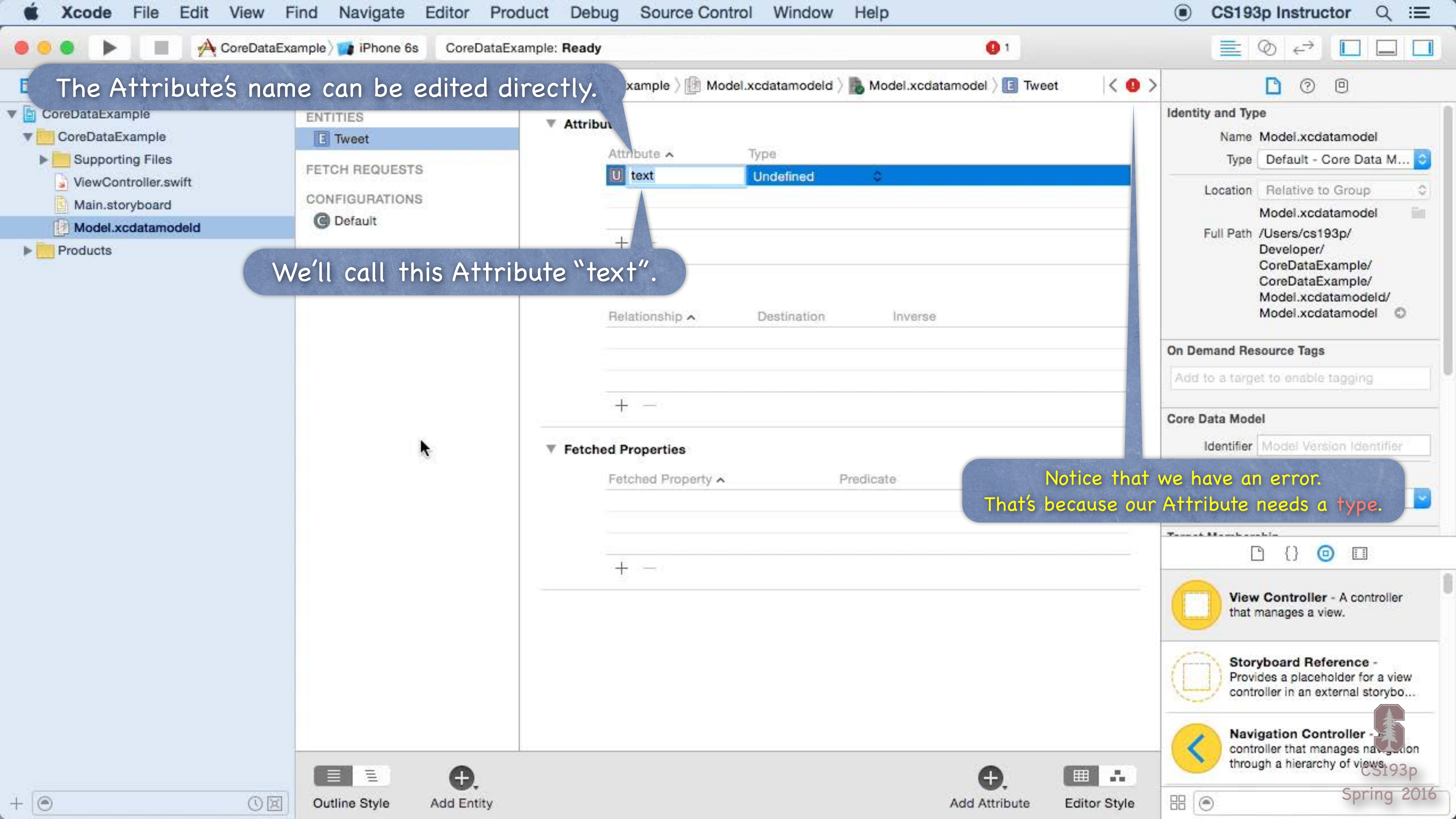
… and Fetched Properties (but we're not going to talk about them).

Entities


Xcode File Edit View Find Navigate Editor Product Debug Source Control Window Help
CS193p Instructor
CoreDataExample iPhone 6s
CoreDataExample: Ready
CoreDataExample > CoreDataExample > Model.xcdatamodeld > Model.xcdatamodel > Tweet
CoreDataExample
CoreDataExample
Supporting Files
ViewController.swift
Main.storyboard
Model.xcdatamodeld
Products
ENTITIES
Tweet
FETCH REQUESTS
CONFIGURATIONS
Default
Attributes
Attribute
Type
Relationships
Relationship
Destination
Inverse
Fetched Properties
Fetched Property
Predicate
Outline Style
Add Entity
Add Attribute
Editor Style
Identity and Type
On Demand Resource Tags
Add to a target to enable tagging
Core Data Model
Identifier
Model Version Identifier
Tools Version
View Controller - A controller that manages a view.
Storyboard Reference - Provides a placeholder for a view controller in an external storybo...
Navigation Controller - A controller that manages navigation through a hierarchy of views.

Name: Model.xcdatamodel
Type: Default - Core Data M...
Location: Relative to Group
Model.xcdatamodel
Full Path: /Users/cs193p/Developer/CoreDataExample/CoreDataExample/Model.xcdatamodeld/Model.xcdatamodel
Minimum: Xcode 7.0

Now we will click here to add some Attributes.
We'll start with the tweet's text.

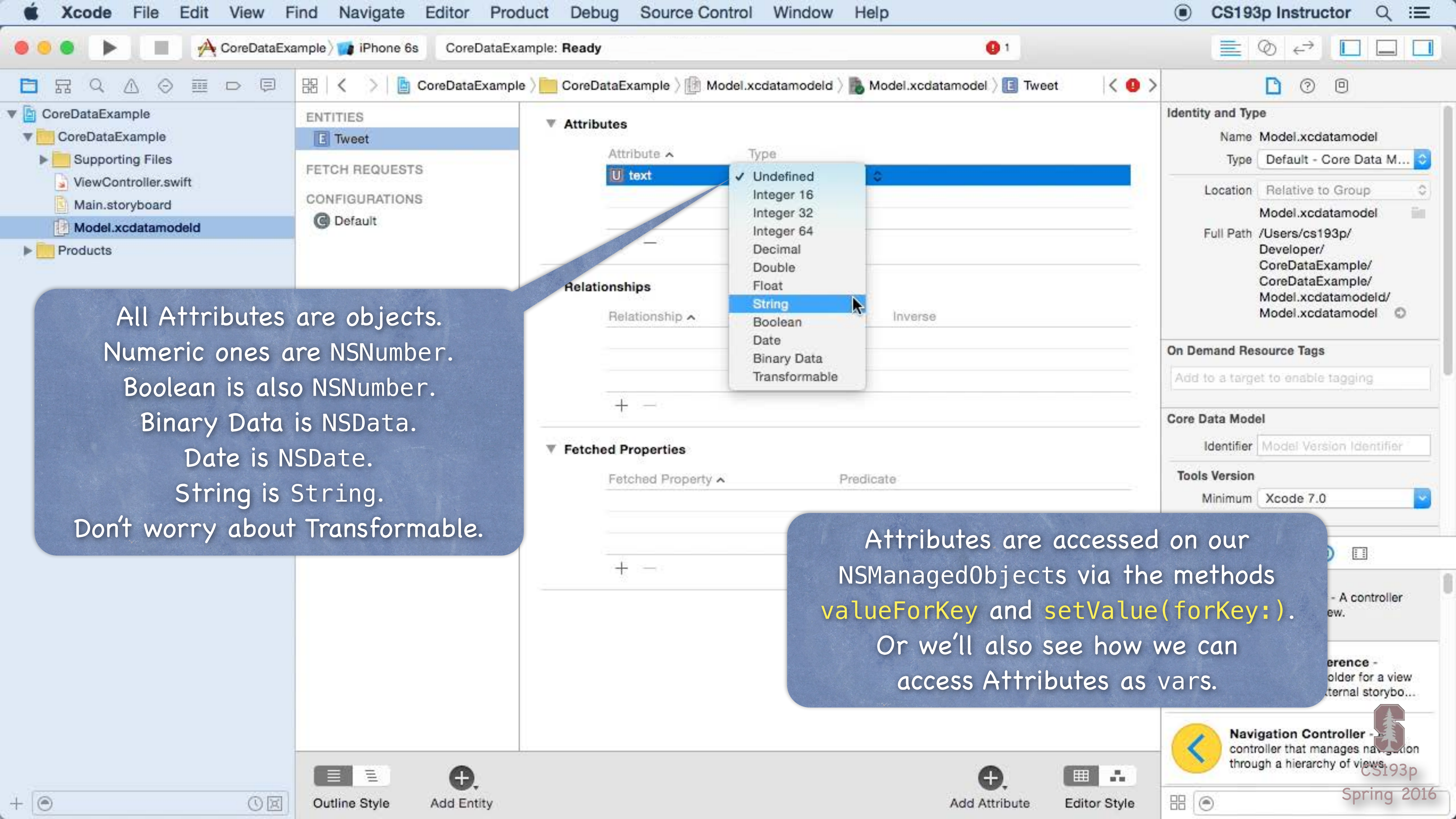


The Attribute's name can be edited directly.

We'll call this Attribute "text".

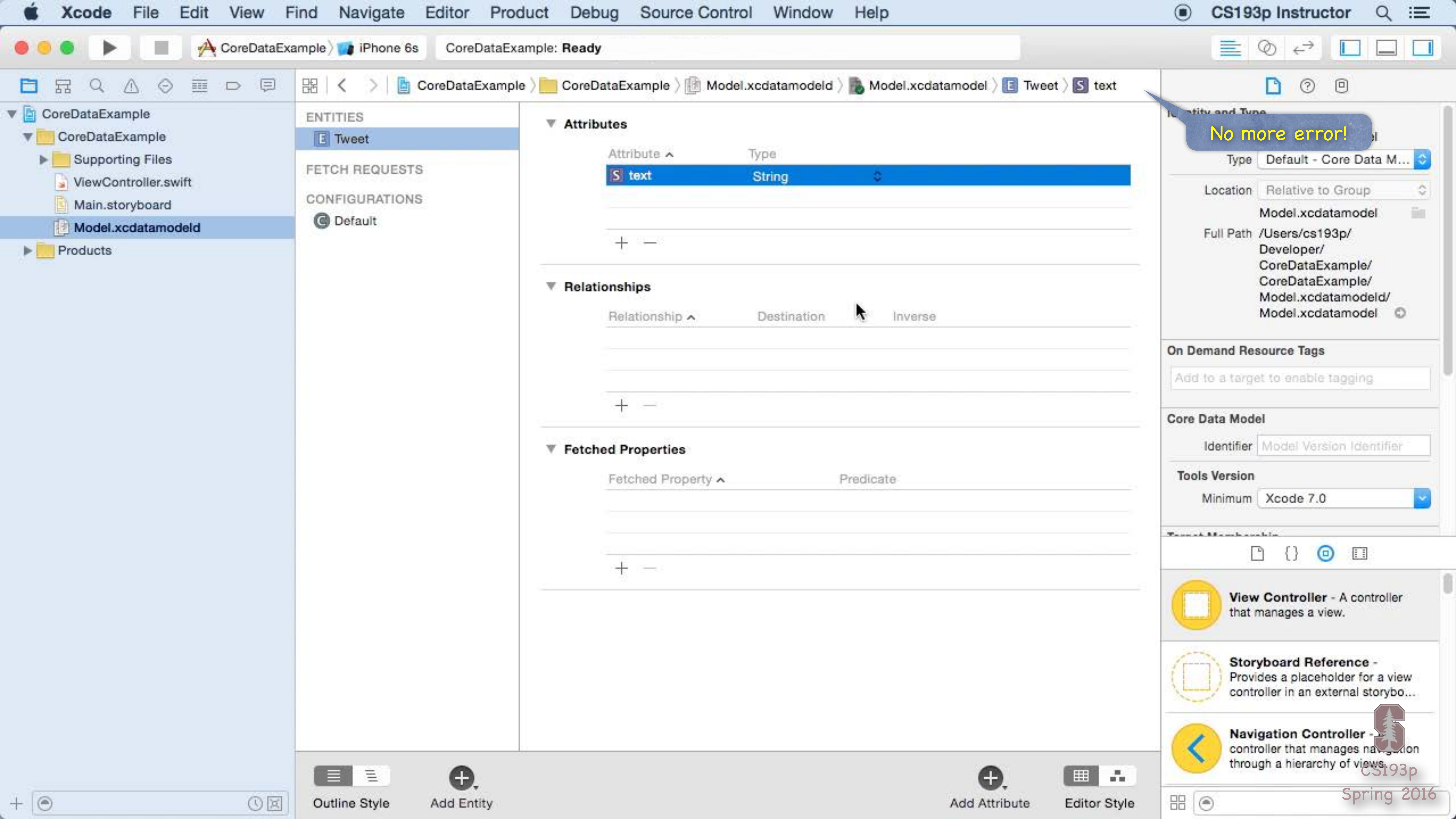
Notice that we have an error.
That's because our Attribute needs a type.

Xcode File Edit View Find Navigate Editor Product Debug Source Control Window Help
CS193p Instructor
CoreDataExample iPhone 6s
CoreDataExample: Ready
CoreDataExample
Supporting Files
ViewController.swift
Main.storyboard
Model.xcdatamodeld
Products
ENTITIES
Tweet
FETCH REQUESTS
CONFIGURATIONS
Default
Attribute
Type
text
Undefined
Relationship
Destination
Inverse
Fetched Properties
Fetched Property
Predicate
Outline Style
Add Entity
Add Attribute
Editor Style
Identity and Type
Name Model.xcdatamodel
Type Default - Core Data M...
Location Relative to Group
Full Path /Users/cs193p/Developer/CoreDataExample/CoreDataExample/Model.xcdatamodeld/Model.xcdatamodel
On Demand Resource Tags
Add to a target to enable tagging
Core Data Model
Identifier Model Version Identifier
View Controller - A controller that manages a view.
Storyboard Reference - Provides a placeholder for a view controller in an external storybo...
Navigation Controller - controller that manages navigation through a hierarchy of views.
CS193p Spring 2016



All Attributes are objects.
Numeric ones are NSNumber.
Boolean is also NSNumber.
Binary Data is NSData.
Date is NSDate.
String is String.
Don't worry about Transformable.

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


Xcode File Edit View Find Navigate Editor Product Debug Source Control Window Help
CS193p Instructor
CoreDataExample › iPhone 6s
CoreDataExample: Ready
CoreDataExample › CoreDataExample › Model.xcdatamodeld › Model.xcdatamodel › Tweet › text
CoreDataExample
CoreDataExample
Supporting Files
ViewController.swift
Main.storyboard
Model.xcdatamodeld
Products
ENTITIES
Tweet
FETCH REQUESTS
CONFIGURATIONS
Default
Relationships
Relationship
Destination
Inverse
Fetched Properties
Fetched Property
Predicate
Outline Style
Add Entity
Add Attribute
Editor Style
On Demand Resource Tags
Add to a target to enable tagging
View Controller - A controller that manages a view.
Storyboard Reference - Provides a placeholder for a view controller in an external storybo...
Navigation Controller - controller that manages navigation through a hierarchy of views.
CS193p
Spring 2016

No more error!

Attributes

Attribute	Type
text	String

Type: Default - Core Data M...

Location: Relative to Group

Model.xcdatamodel

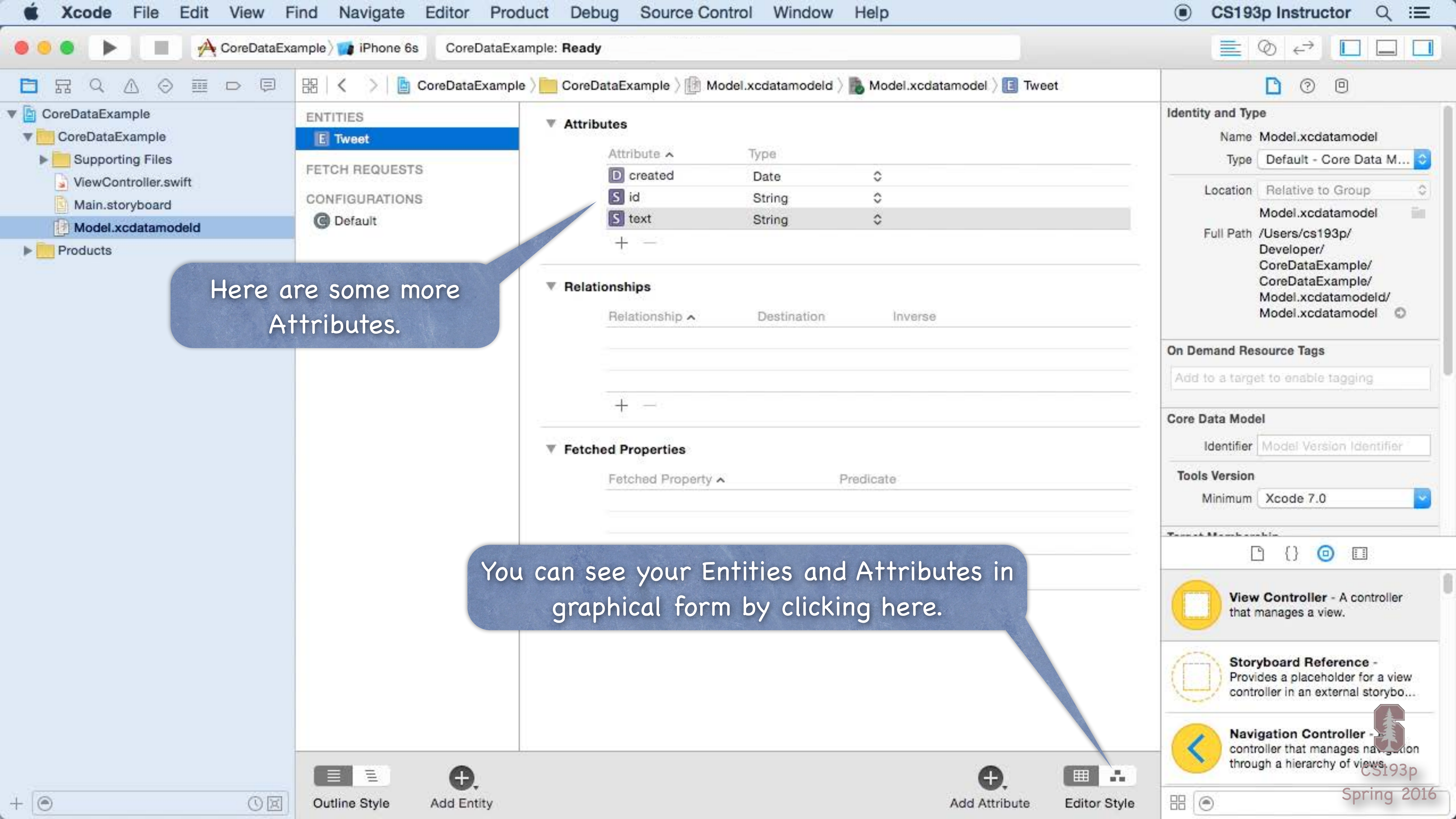
Full Path: /Users/cs193p/Developer/CoreDataExample/CoreDataExample/Model.xcdatamodeld/Model.xcdatamodel

Core Data Model

Identifier: Model Version Identifier

Tools Version

Minimum: Xcode 7.0


Xcode File Edit View Find Navigate Editor Product Debug Source Control Window Help
CS193p Instructor
CoreDataExample › iPhone 6s
CoreDataExample: Ready
CoreDataExample › CoreDataExample › Model.xcdatamodeld › Model.xcdatamodel › Tweet
CoreDataExample
CoreDataExample
Supporting Files
ViewController.swift
Main.storyboard
Model.xcdatamodeld
Products
ENTITIES
Tweet
FETCH REQUESTS
CONFIGURATIONS
Default
Outline Style
Add Entity
Add Attribute
Editor Style
View Controller - A controller that manages a view.
Storyboard Reference - Provides a placeholder for a view controller in an external storybo...
Navigation Controller - A controller that manages navigation through a hierarchy of views.

Attributes

Attribute	Type
created	Date
id	String
text	String

Relationships

Relationship	Destination	Inverse

Fetched Properties

Fetched Property	Predicate

Identity and Type

Name Model.xcdatamodel

Type Default - Core Data M...

Location Relative to Group

Model.xcdatamodel

Full Path /Users/cs193p/Developer/CoreDataExample/CoreDataExample/Model.xcdatamodeld/Model.xcdatamodel

On Demand Resource Tags

Add to a target to enable tagging

Core Data Model

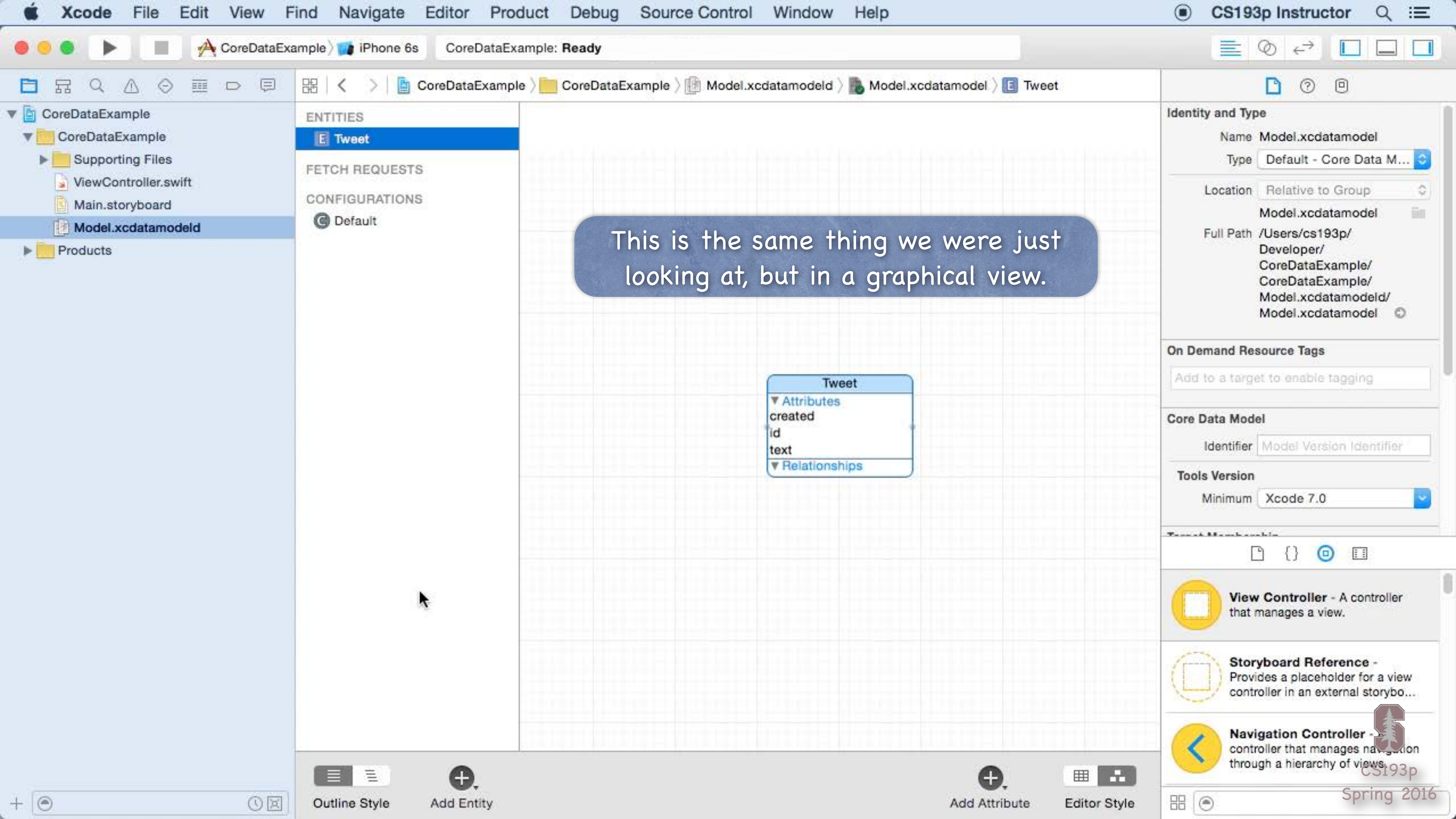
Identifier Model Version Identifier

Tools Version

Minimum Xcode 7.0

Here are some more Attributes.

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

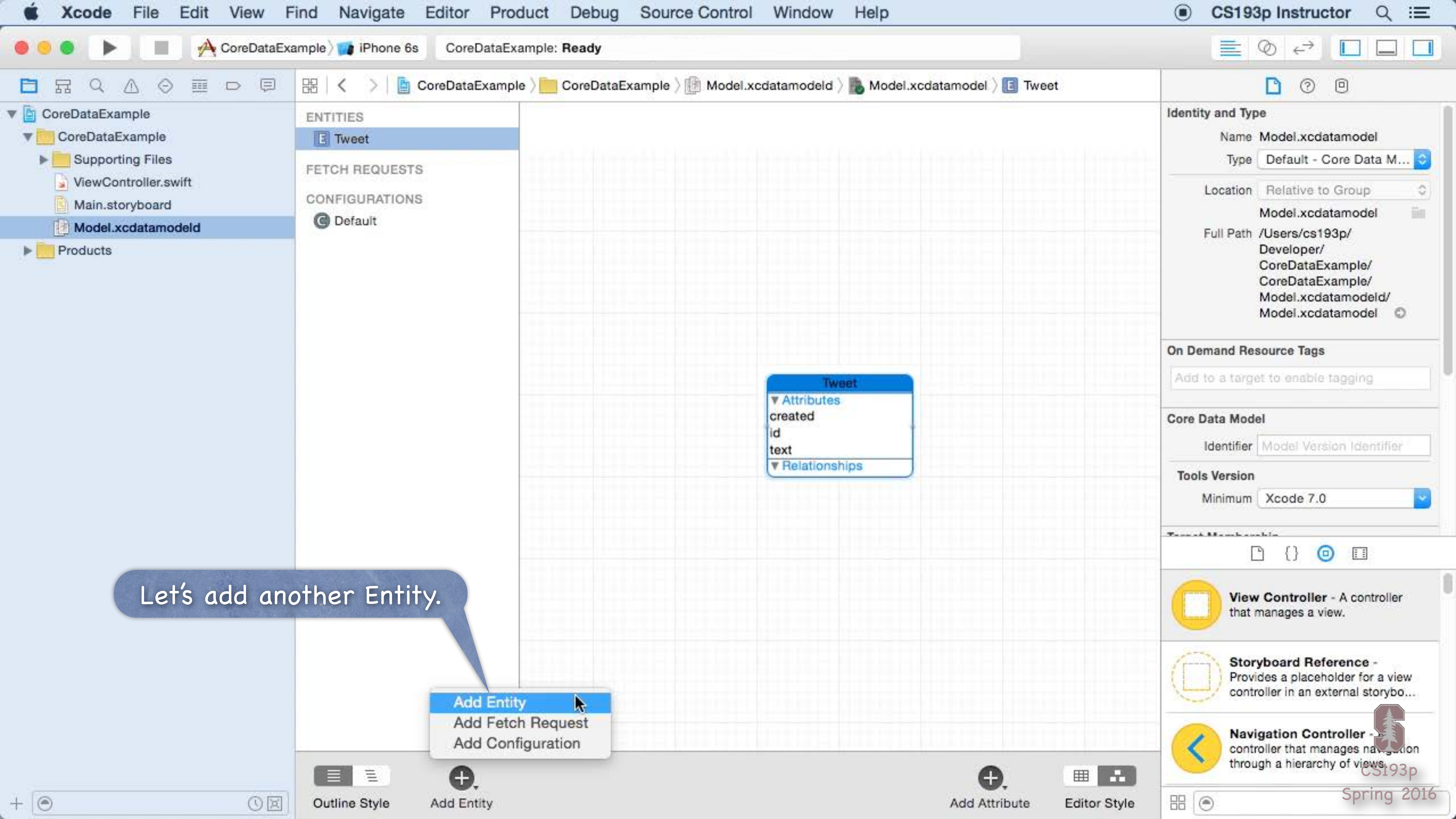

Xcode File Edit View Find Navigate Editor Product Debug Source Control Window Help
CS193p Instructor
CoreDataExample › iPhone 6s
CoreDataExample: Ready
CoreDataExample › CoreDataExample › Model.xcdatamodeld › Model.xcdatamodel › Tweet
CoreDataExample
CoreDataExample
Supporting Files
ViewController.swift
Main.storyboard
Model.xcdatamodeld
Products
ENTITIES
Tweet
FETCH REQUESTS
CONFIGURATIONS
Default
Tweet
Attributes
created
id
text
Relationships
Outline Style
Add Entity
Add Attribute
Editor Style
View Controller - A controller that manages a view.
Storyboard Reference - Provides a placeholder for a view controller in an external storybo...
Navigation Controller - A controller that manages navigation through a hierarchy of views.
CS193p
Spring 2016

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

Identity and Type
Name Model.xcdatamodel
Type Default - Core Data M...
Location Relative to Group
Model.xcdatamodel
Full Path /Users/cs193p/Developer/CoreDataExample/CoreDataExample/Model.xcdatamodeld/Model.xcdatamodel

On Demand Resource Tags
Add to a target to enable tagging

Core Data Model
Identifier Model Version Identifier
Tools Version
Minimum Xcode 7.0



Identity and Type

Name Model.xcdatamodel

Type Default - Core Data M...

Location Relative to Group

Model.xcdatamodel

Full Path /Users/cs193p/Developer/CoreDataExample/CoreDataExample/Model.xcdatamodeld/Model.xcdatamodel

On Demand Resource Tags

Add to a target to enable tagging

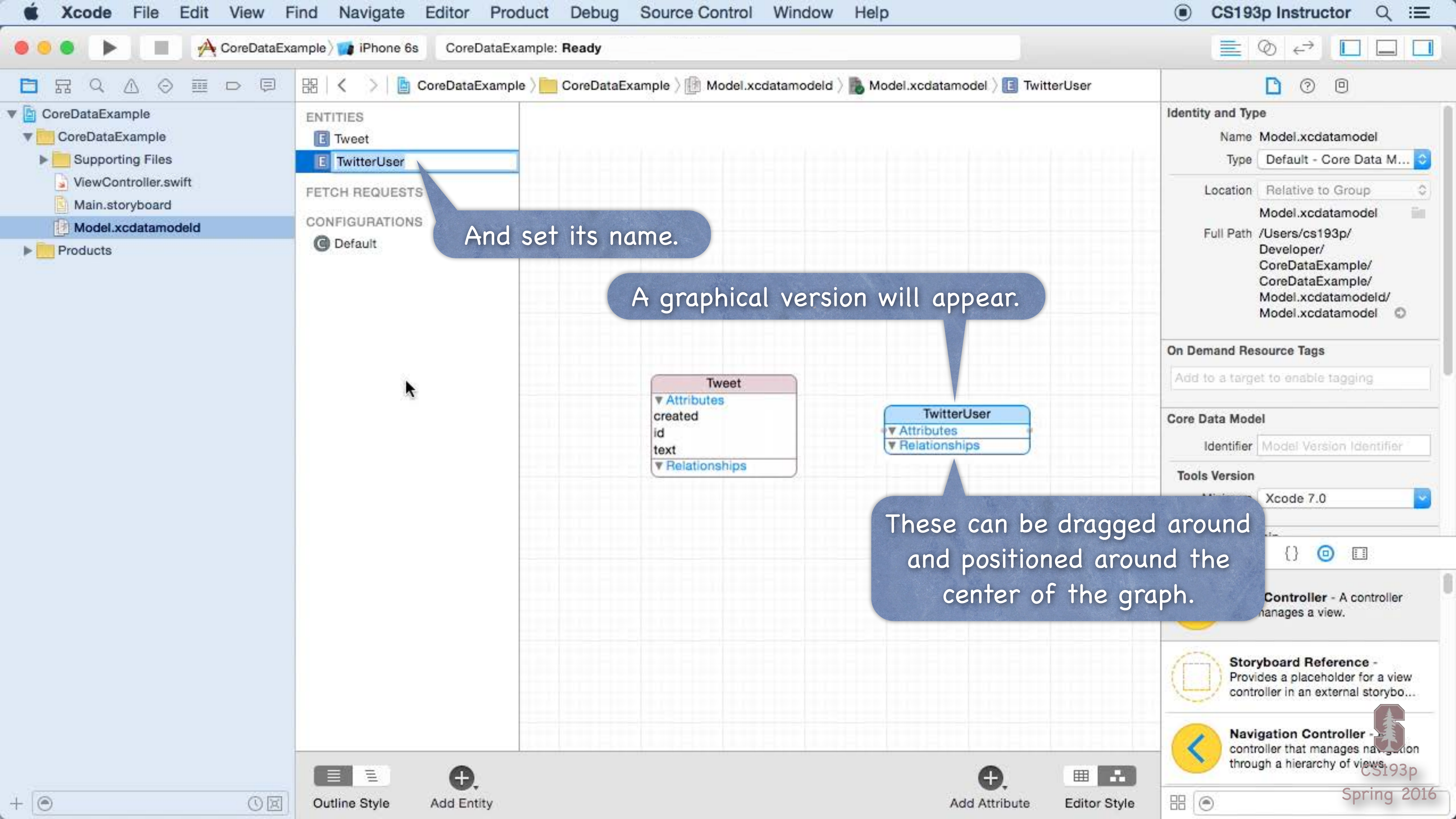
Core Data Model

Identifier Model Version Identifier

Tools Version

Minimum Xcode 7.0

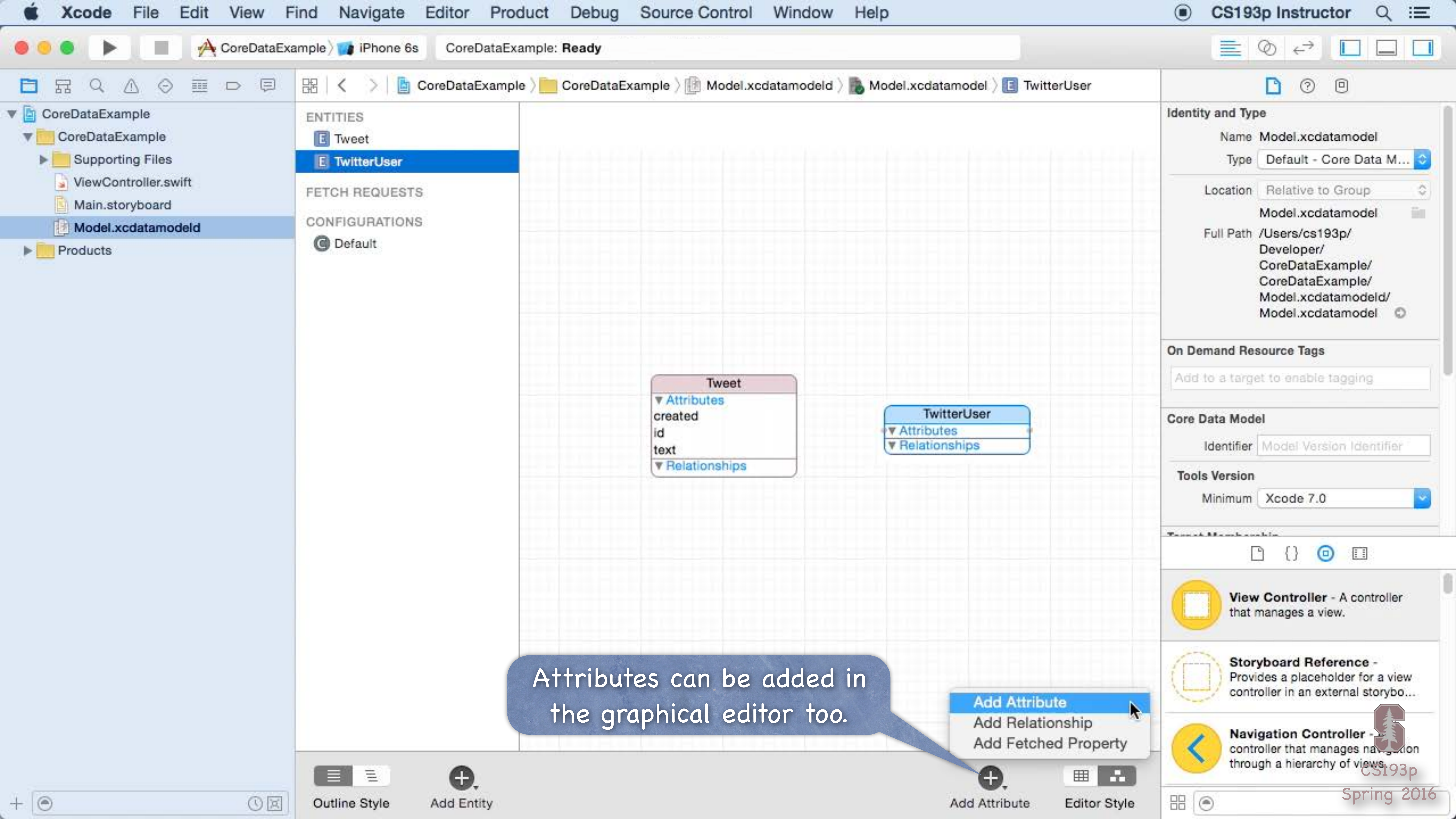
Let's add another Entity.


Xcode File Edit View Find Navigate Editor Product Debug Source Control Window Help
CS193p Instructor
CoreDataExample › iPhone 6s
CoreDataExample: Ready
CoreDataExample › CoreDataExample › Model.xcdatamodeld › Model.xcdatamodel › TwitterUser
CoreDataExample
CoreDataExample
Supporting Files
ViewController.swift
Main.storyboard
Model.xcdatamodeld
Products
ENTITIES
Tweet
TwitterUser
FETCH REQUESTS
CONFIGURATIONS
Default
Tweet
Attributes
created
id
text
Relationships
TwitterUser
Attributes
Relationships
Outline Style
Add Entity
Add Attribute
Editor Style
Identity and Type
Name Model.xcdatamodel
Type Default - Core Data M...
Location Relative to Group
Model.xcdatamodel
Full Path /Users/cs193p/Developer/CoreDataExample/CoreDataExample/Model.xcdatamodeld/Model.xcdatamodel
On Demand Resource Tags
Add to a target to enable tagging
Core Data Model
Identifier Model Version Identifier
Tools Version
Xcode 7.0
Controller - A controller manages a view.
Storyboard Reference - Provides a placeholder for a view controller in an external storybo...
Navigation Controller - controller that manages navigation through a hierarchy of views.

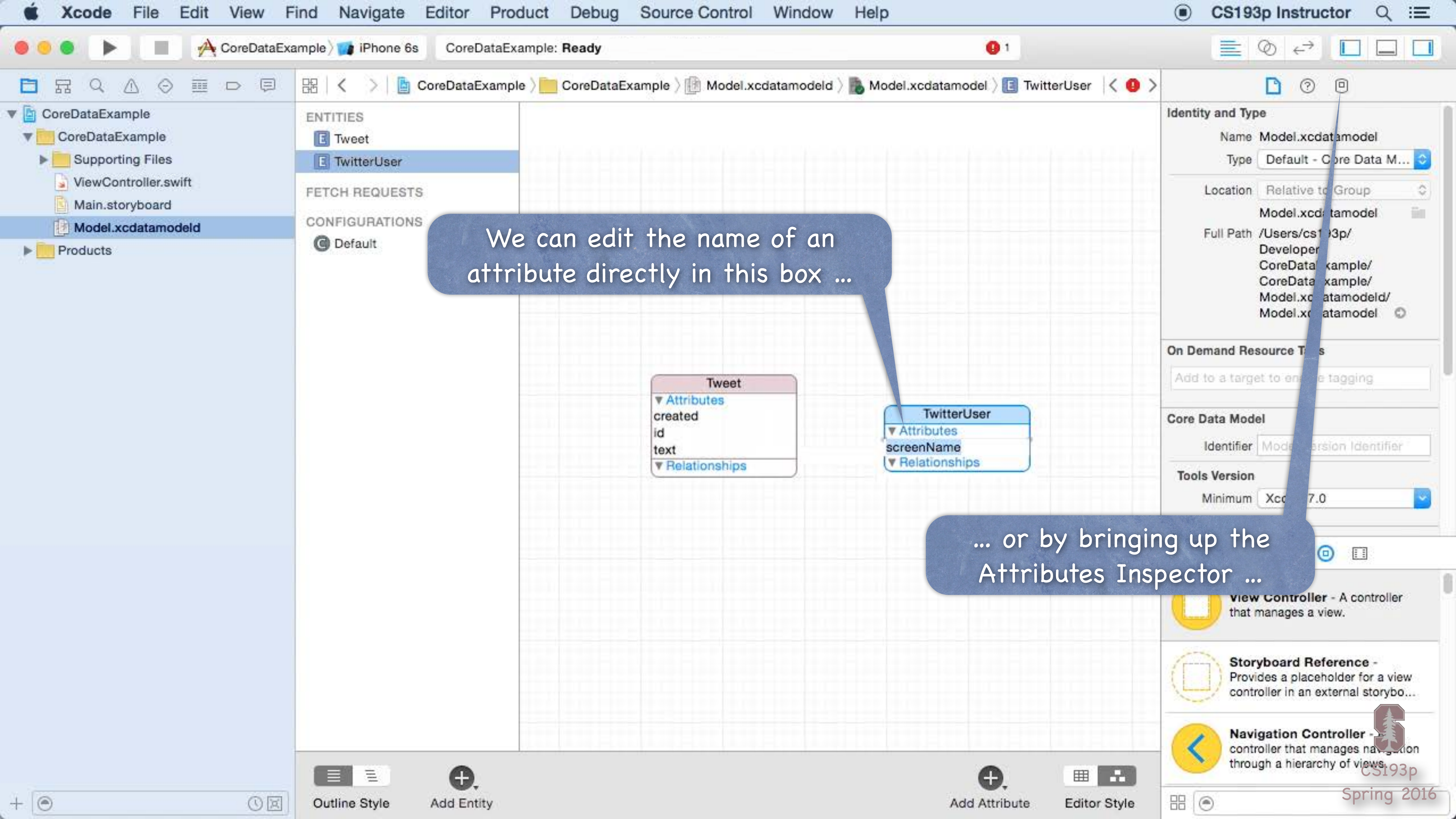
And set its name.

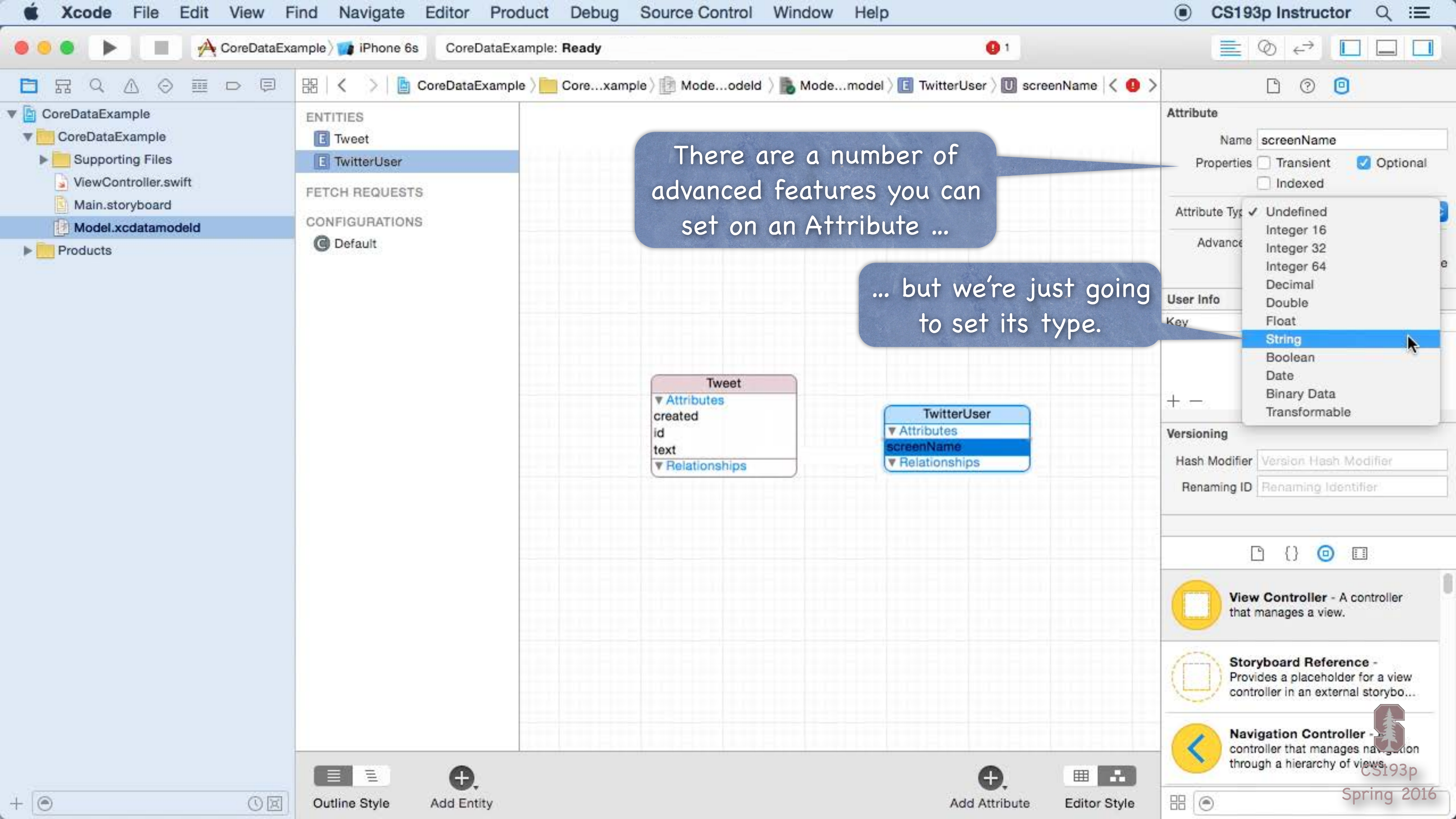
A graphical version will appear.

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



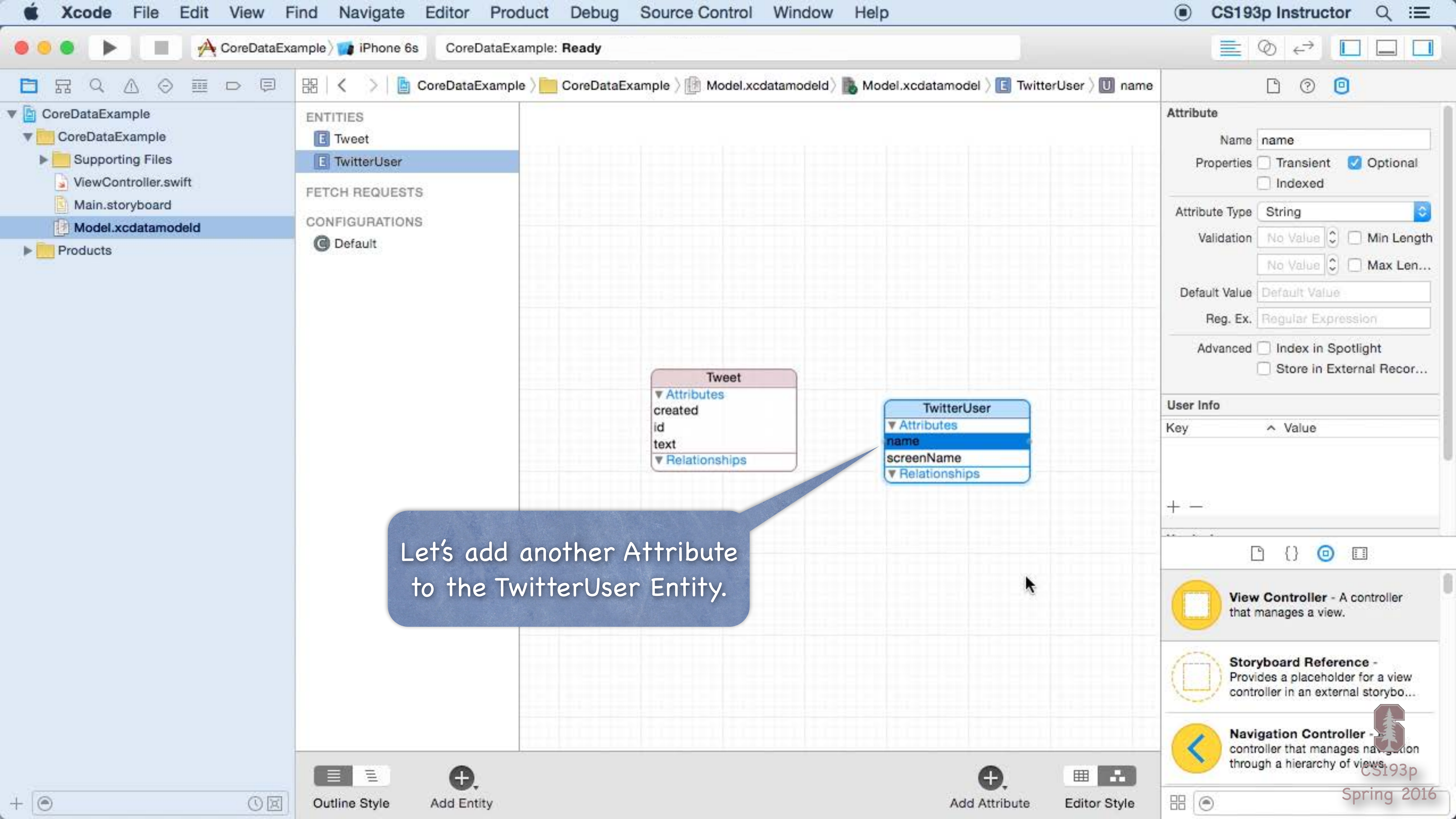
Attributes can be added in the graphical editor too.


Xcode File Edit View Find Navigate Editor Product Debug Source Control Window Help
CS193p Instructor
CoreDataExample › iPhone 6s
CoreDataExample: Ready
CoreDataExample › CoreDataExample › Model.xcdatamodeld › Model.xcdatamodel › TwitterUser
CoreDataExample
CoreDataExample
Supporting Files
ViewController.swift
Main.storyboard
Model.xcdatamodeld
Products
ENTITIES
Tweet
TwitterUser
FETCH REQUESTS
CONFIGURATIONS
Default
Tweet
Attributes
created
id
text
Relationships
TwitterUser
Attributes
screenName
Relationships
Identity and Type
Name Model.xcdatamodel
Type Default - Core Data M...
Location Relative to Group
Model.xcdatamodel
Full Path /Users/cs193p/Developer/CoreDataExample/CoreDataExample/Model.xcdatamodeld/Model.xcdatamodel
On Demand Resource Tags
Add to a target to enable tagging
Core Data Model
Identifier Model Version Identifier
Tools Version
Minimum Xcode 7.0
View Controller - A controller that manages a view.
Storyboard Reference - Provides a placeholder for a view controller in an external storybo...
Navigation Controller - A controller that manages navigation through a hierarchy of views.
Outline Style
Add Entity
Add Attribute
Editor Style
We can edit the name of an attribute directly in this box ...
... or by bringing up the Attributes Inspector ...



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

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



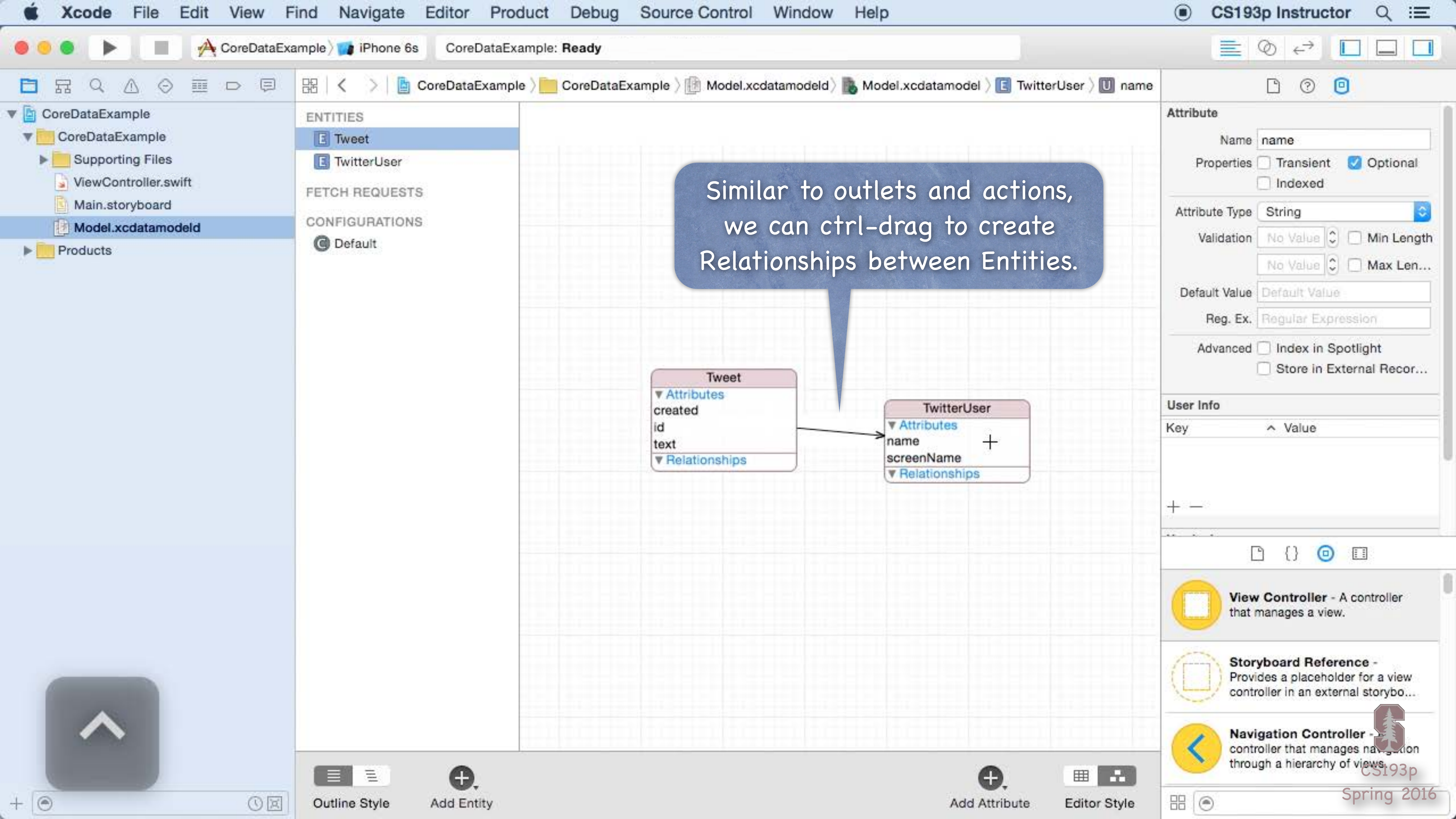
Let's add another Attribute to the TwitterUser Entity.

Attribute

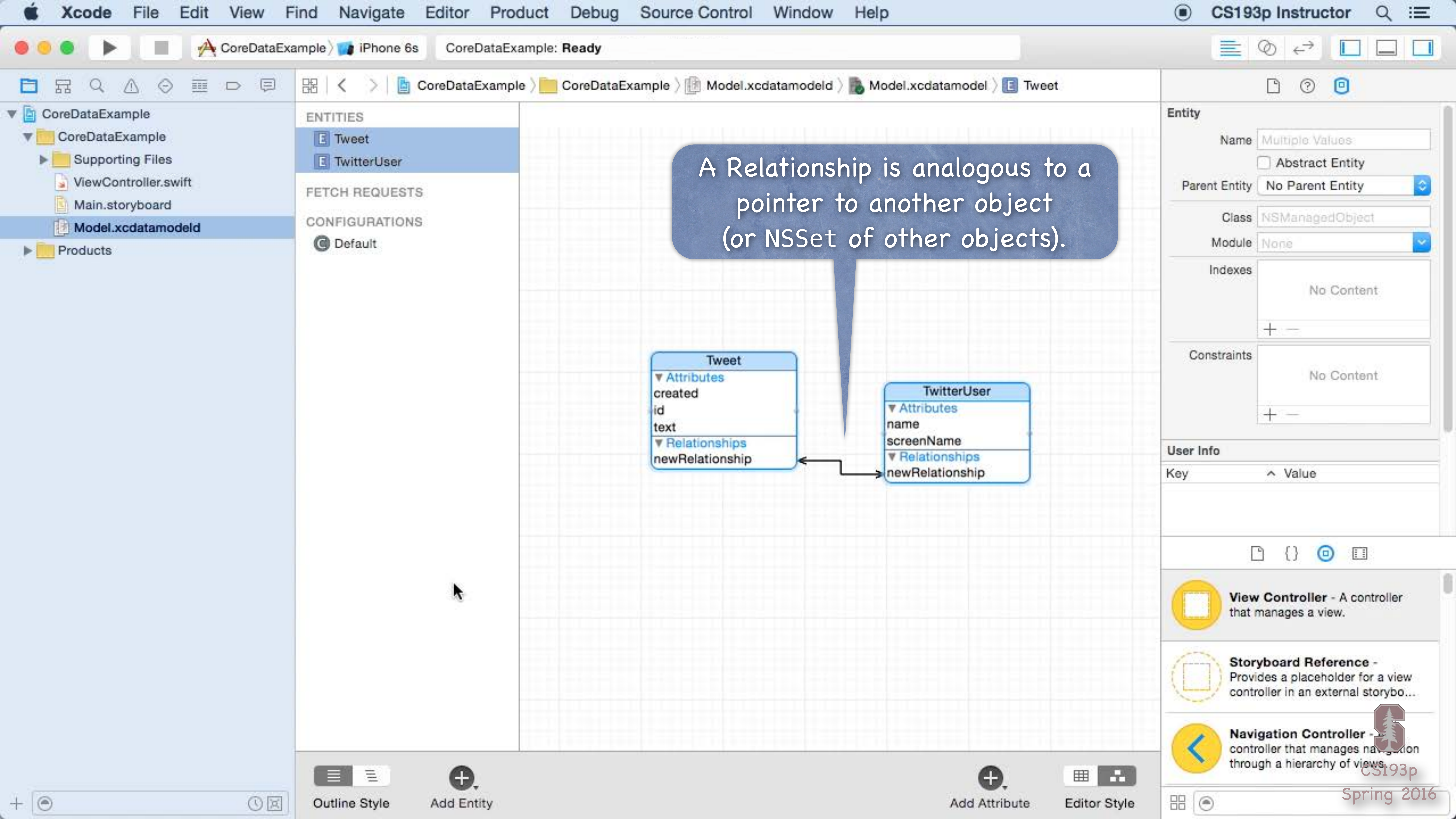
Name: name
Properties: [ ] Transient [✓ Optional] [ ] Indexed
Attribute Type: String
Validation: No Value [ ] Min Length
No Value [ ] Max Len...
Default Value: Default Value
Reg. Ex.: Regular Expression
Advanced: [ ] Index in Spotlight [ ] Store in External Recor...

User Info

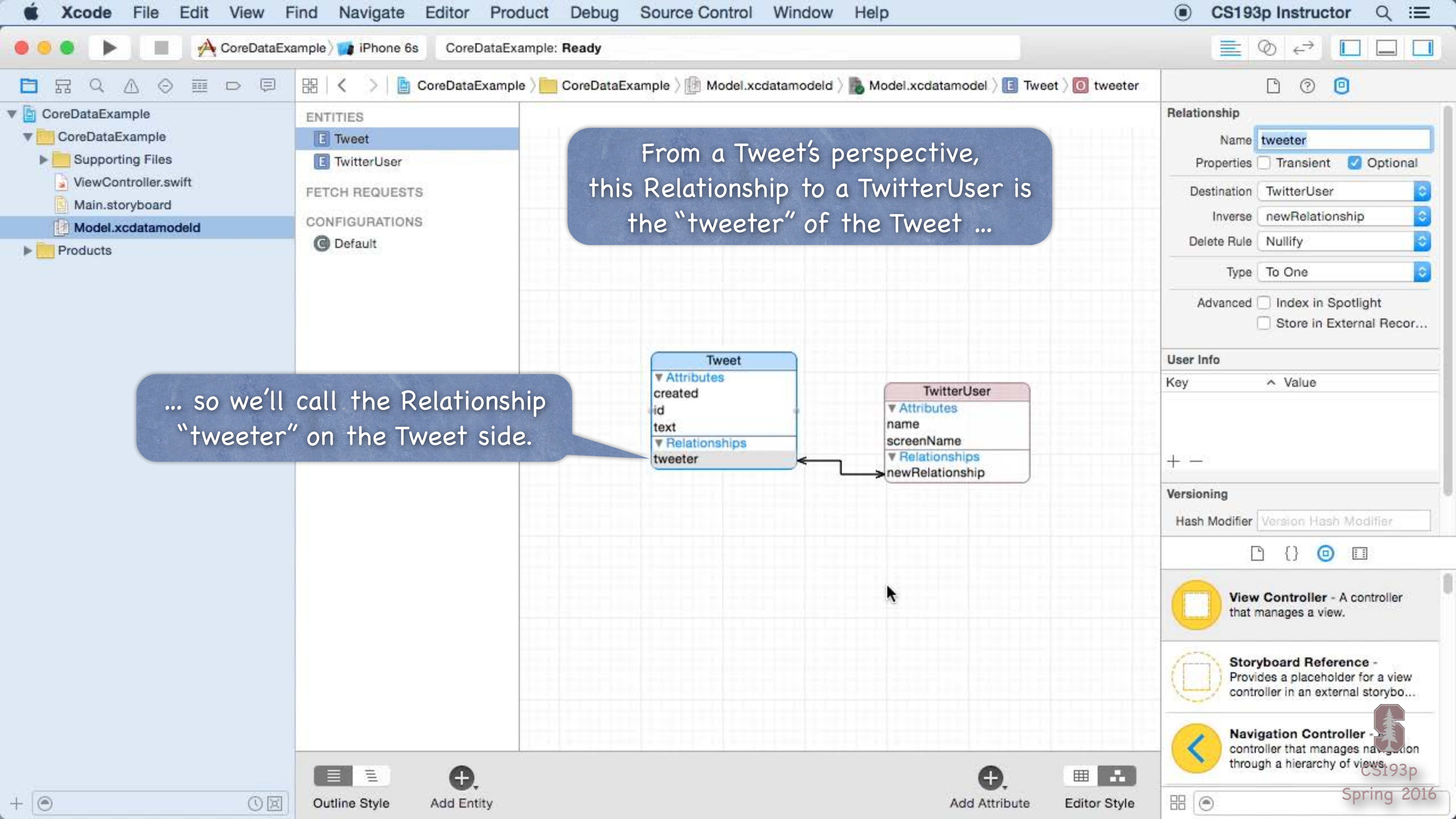
Key	Value



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



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


Xcode File Edit View Find Navigate Editor Product Debug Source Control Window Help
CS193p Instructor
CoreDataExample: Ready
CoreDataExample > CoreDataExample > Model.xcdatamodeld > Model.xcdatamodel > Tweet > tweeter
CoreDataExample
Supporting Files
ViewController.swift
Main.storyboard
Model.xcdatamodeld
Products
ENTITIES
Tweet
TwitterUser
FETCH REQUESTS
CONFIGURATIONS
Default
Tweet
Attributes
created
id
text
Relationships
tweeter
TwitterUser
Attributes
name
screenName
Relationships
newRelationship
Outline Style
Add Entity
Add Attribute
Editor Style
View Controller - A controller that manages a view.
Storyboard Reference - Provides a placeholder for a view controller in an external storybo...
Navigation Controller - controller that manages navigation through a hierarchy of views.

From a Tweet's perspective, this Relationship to a TwitterUser is the "tweeter" of the Tweet ...

... so we'll call the Relationship "tweeter" on the Tweet side.

Relationship

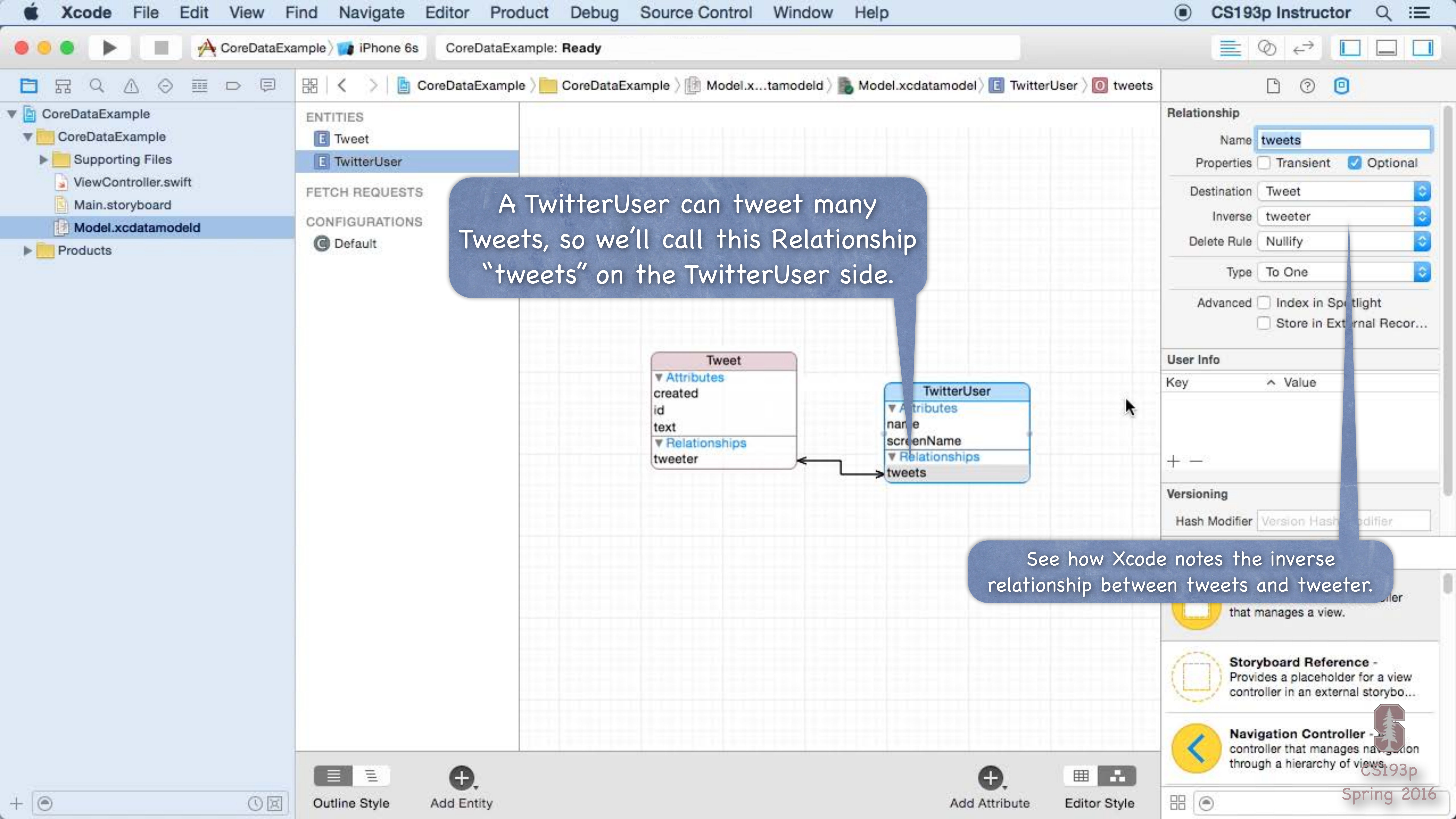
Name	tweeter
Properties	☐ Transient ☑ Optional
Destination	TwitterUser
Inverse	newRelationship
Delete Rule	Nullify
Type	To One
Advanced	☐ Index in Spotlight
	☐ Store in External Recor...

User Info

Key	Value

Versioning

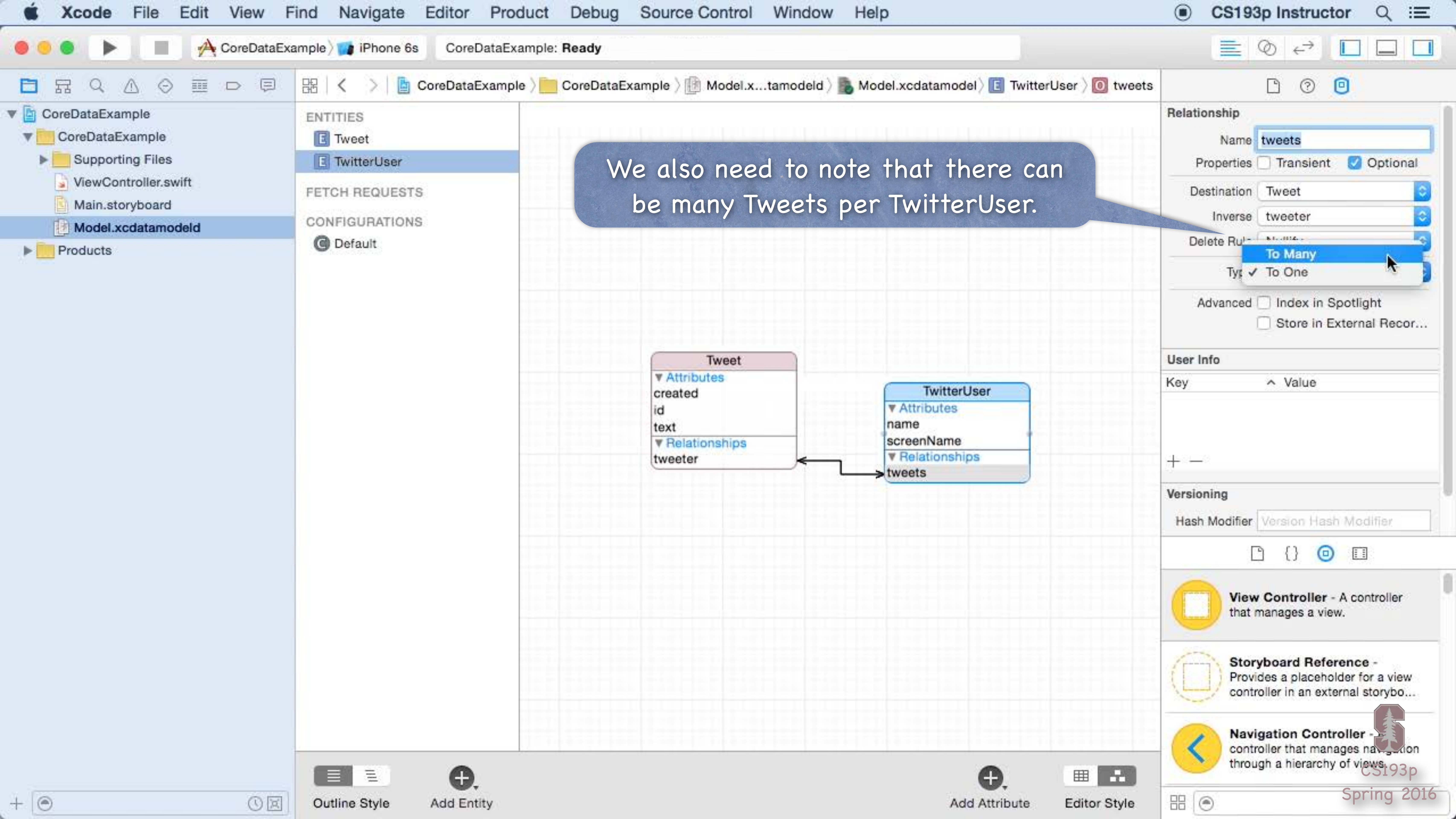
Hash Modifier: Version Hash Modifier



A TwitterUser can tweet many Tweets, so we'll call this Relationship "tweets" on the TwitterUser side.

See how Xcode notes the inverse relationship between tweets and tweeter.

Relationship
Name: tweets
Properties: Transient [✓ Optional]
Destination: Tweet
Inverse: tweeter
Delete Rule: Nullify
Type: To One


Xcode File Edit View Find Navigate Editor Product Debug Source Control Window Help
CS193p Instructor
CoreDataExample › iPhone 6s
CoreDataExample: Ready
CoreDataExample › CoreDataExample › Model.x...tamodeld › Model.xcdatamodel › TwitterUser › tweets
CoreDataExample
CoreDataExample
Supporting Files
ViewController.swift
Main.storyboard
Model.xcdatamodeld
Products
ENTITIES
Tweet
TwitterUser
FETCH REQUESTS
CONFIGURATIONS
Default
Tweet
Attributes
created
id
text
Relationships
tweeter
TwitterUser
Attributes
name
screenName
Relationships
tweets
Outline Style
Add Entity
Add Attribute
Editor Style
View Controller - A controller that manages a view.
Storyboard Reference - Provides a placeholder for a view controller in an external storybo...
Navigation Controller - controller that manages navigation through a hierarchy of views.

We also need to note that there can be many Tweets per TwitterUser.

Relationship

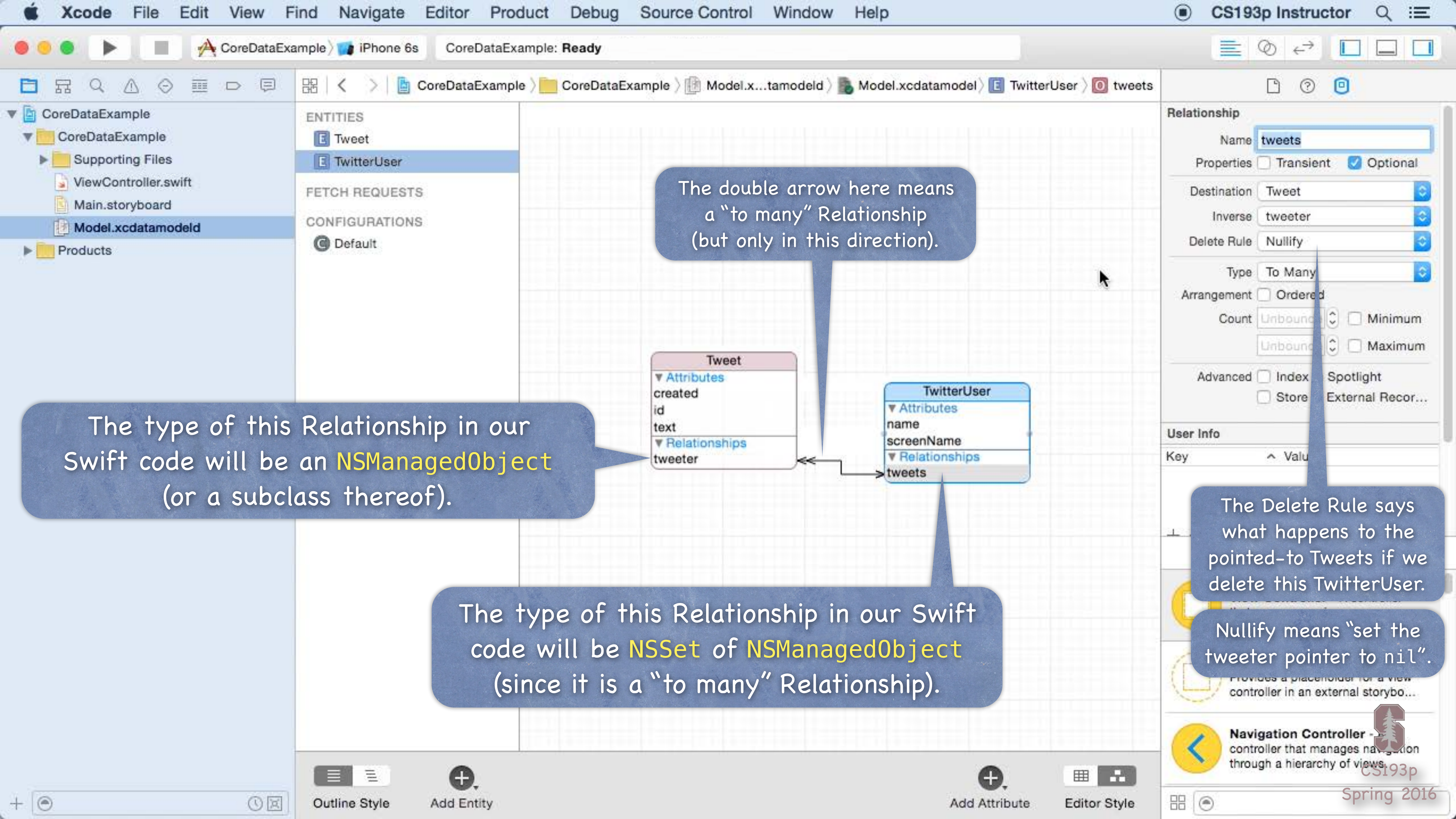
Name	tweets
Properties	☐ Transient ☑ Optional
Destination	Tweet
Inverse	tweeter
Delete Rule	Nullify
Type	To Many / ✓ To One
Advanced	☐ Index in Spotlight ☐ Store in External Recor...

User Info

Key	Value

Versioning

Hash Modifier: Version Hash Modifier



Xcode File Edit View Find Navigate Editor Product Debug Source Control Window Help

CoreDataExample > iPhone 6s

CoreDataExample: **Ready**

CoreDataExample > CoreDataExample > Model.x...tamodeld > Model.xcdatamodel > TwitterUser > tweets

Relationship

Name: tweets

Properties: Transient [✓ Optional]

Destination: Tweet

Inverse: tweeter

Delete Rule: Nullify

Type: To Many

Arrangement: Ordered

Count: Unbounded Minimum

Unbounded Maximum

Advanced: Index Spotlight

Store External Recor...

User Info

Key Value

Navigation Controller - controller that manages navigation through a hierarchy of views.

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

The type of this Relationship in our Swift code will be an NSManagedObject (or a subclass thereof).

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

The Delete Rule says what happens to the pointed-to Tweets if we delete this TwitterUser.

Nullify means "set the tweeter pointer to nil".

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. Click the "Use Core Data" button when you create a project
  - 2. Create a `UIManagedDocument` and ask for its `managedObjectContext` (a var).

Core Data

- Sharing a global NSManagedObjectContext in your AppDelegate

  Clicking the "Use Core Data" button when you create a project adds code to your AppDelegate.
  The most important thing it adds is a managedObjectContext var.

  You can access your AppDelegate's managedObjectContext var like this …

  ```
  (UIApplication.sharedApplication().delegate as! AppDelegate).managedObjectContext
  ```

  If you have an existing project, create a new project and copy the AppDelegate code over.
  You have to copy not just the managedObjectContext var, but all the methods it depends on.
  It's pretty obvious which those are.

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

  First, you need to create a URL to the file the document will be stored in.
  This requires knowing a little bit of how to use the file system which we have not yet covered!
  But the code goes like this …

```
let fm = NSFileManager.defaultManager()
if let docsDir = fm.URLsForDirectory(.DocumentDirectory, inDomains: .UserDomainMask).first {
    let url = docsDir.URLByAppendingPathComponent("MyDocumentName")
    let document = UIManagedDocument(fileURL: url)
}
```

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

UIManagedDocument

- How to open or create a UIManagedDocument

  Before you use a UIManagedDocument, you have to check to see if it's open or not.

  If it is already open (in the .Normal state), you are good to go using the managedObjectContext

  ```
  if document.documentState == .Normal { /* use managedObjectContext */ }
  ```

  If it's .Closed …

  ```
  if document.documentState == .Closed { /* need to open/create document */ }
  ```

  … you need to open (or create) it.

  To do that, check to see if the UIManagedDocument's underlying file exists on disk …

  ```
  if let path = fileURL.path,
     let fileExists = NSFileManager.defaultManager().fileExistsAtPath(path) { … }
  ```

  … if it does exist, open the document using …

  ```
  document.openWithCompletionHandler { (success: Bool) in /* use managedObjectContext */ }
  ```

  … if it does not exist, create the document using …

  ```
  document.saveToURL(document.fileURL, forSaveOperation: .ForCreating) { success in ... }
  ```

UIManagedDocument

- This is all asynchronous!

  Opening or creating the document might take a little time.

  And we do not want to block the main thread.

  However, your block does get executed back on the main thread eventually.

- Other documentStates

  `.SavingError` (success will be NO in completion handler)

  `.EditingDisabled` (temporary situation, try again)

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

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

UIManagedDocument

- Saving the document

  UIManagedDocuments AUTOSAVE themselves!

  However, if, for some reason you wanted to manually save (asynchronously, of course) …

  ```
  document.saveToURL(document.fileURL, forSaveOperation:.ForOverwriting) { success in ... }
  ```

  Note that this is almost identical to creation (just `.ForOverwriting` is different).

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

- Closing the document

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

  But you can explicitly close with this asynchronous method …

  ```
  document.closeWithCompletionHandler { success in  ... }
  ```

Core Data

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

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

  Or we got it from our `AppDelegate` with code we got from creating a new Core Data project.

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

Core Data

- Inserting objects into the database

```
let moc = aDocument.managedObjectContext // or from AppDelegate
let tweet: NSManagedObject =
    NSEntityDescription.insertNewObjectForEntityForName("Tweet", inManagedObjectContext: moc)
```

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

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

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

Core Data

- How to access Attributes in an NSManagedObject instance

  You can access them using the following two NSKeyValueCoding protocol methods …

  ```
  func valueForKey(String) -> AnyObject?
  func setValue(AnyObject?, forKey: String)
  ```

  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, "`created`" or "`text`".

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

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

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

  Binary data values are NSData objects.

  Date values are NSDate objects.

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

  Non-"to-many" relationships are other NSManagedObjects, of course.

Core Data

- Changes (writes) only happen in memory, until you save
  - Remember, `UIManagedDocument` autosaves.
  - When the document is saved, the context is saved & your changes get written to the database.
  - Be careful during development where you press "Stop" in Xcode (sometimes autosave is pending).
- You must explicitly `save()` if not using `UIManagedDocument`

```
let context = (UIApplication.sharedApplication as! AppDelegate).managedObjectContext
// do things with the context
context.save()
```

... ah, but it's not quite that easy!

The `save()` method in `UIManagedObjectContext` can `throw` an error!

How do we deal with thrown errors?!

Thrown Errors

- In Swift, methods can throw errors

  You will always know these methods because they'll have the keyword throws on the end.

  ```
  func save() throws
  ```

  You must put calls to functions like this in a do { } block and use the word try to call them.

  ```
  do {
       try context.save()
  } catch let error {
      // error will be something that implements the ErrorType protocol, e.g., NSError
      // usually these are enums that have associated values to get error details
      throw error // this would re-throw the error (only ok if the method we are in throws)
  }
  ```

  If you are certain a call will not throw, you can force try with try! …

  ```
  try! context.save() // will crash your program if save() actually throws an error
  ```

Core Data

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

  There's no type-checking.

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

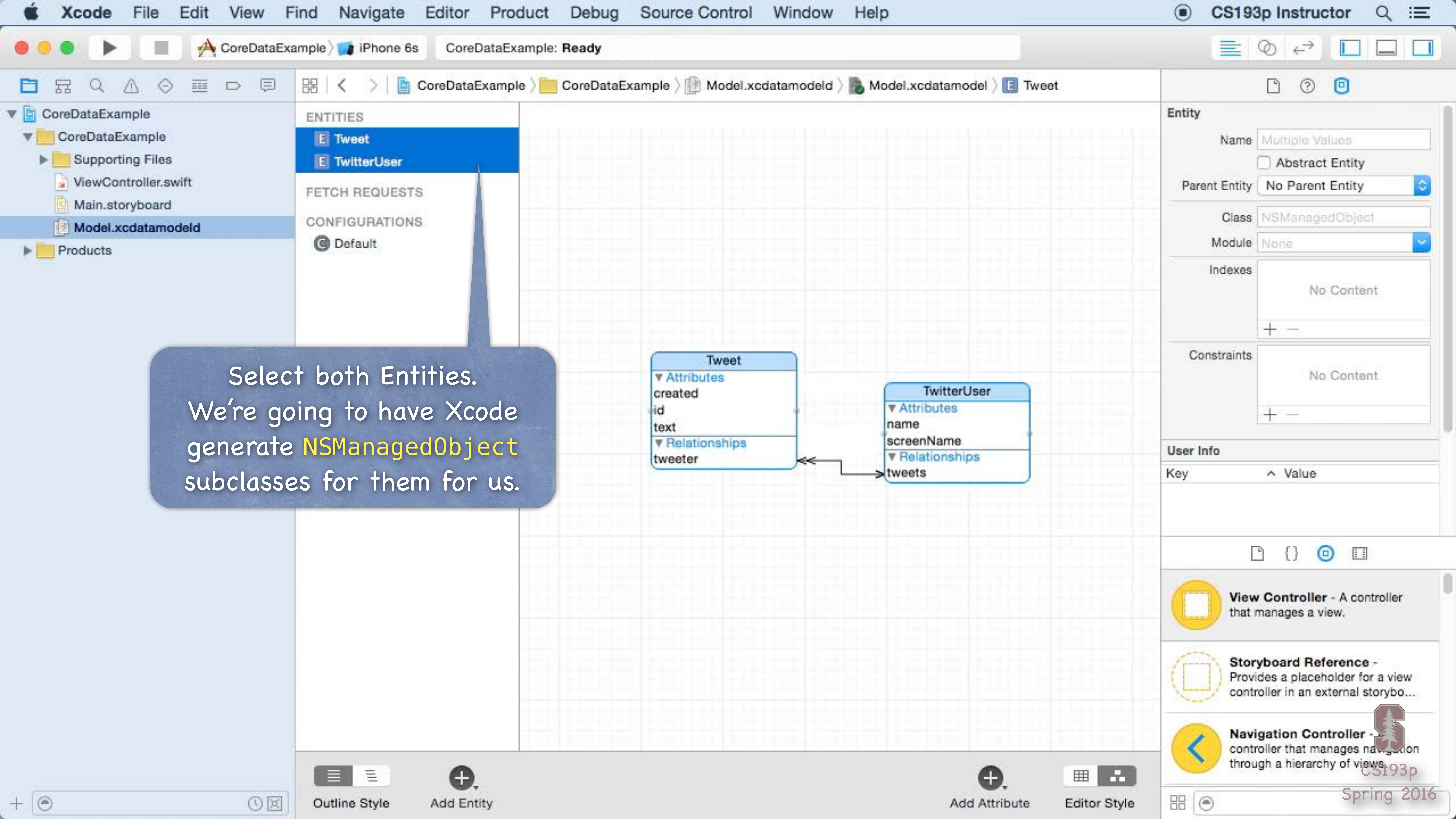
- What we really want is to set/get using `var`s!

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

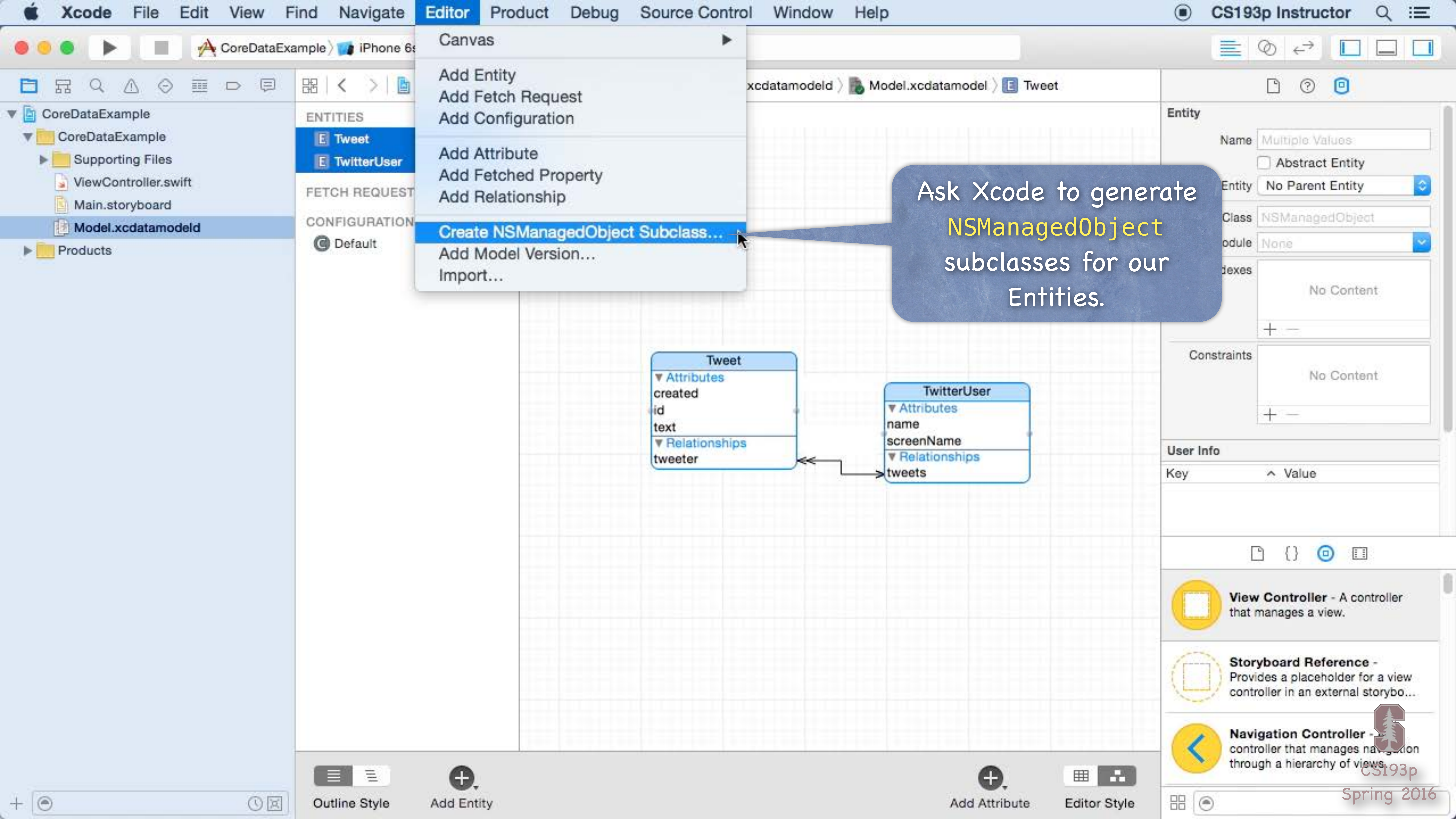
  The subclass will have `var`s for each attribute in the database.

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

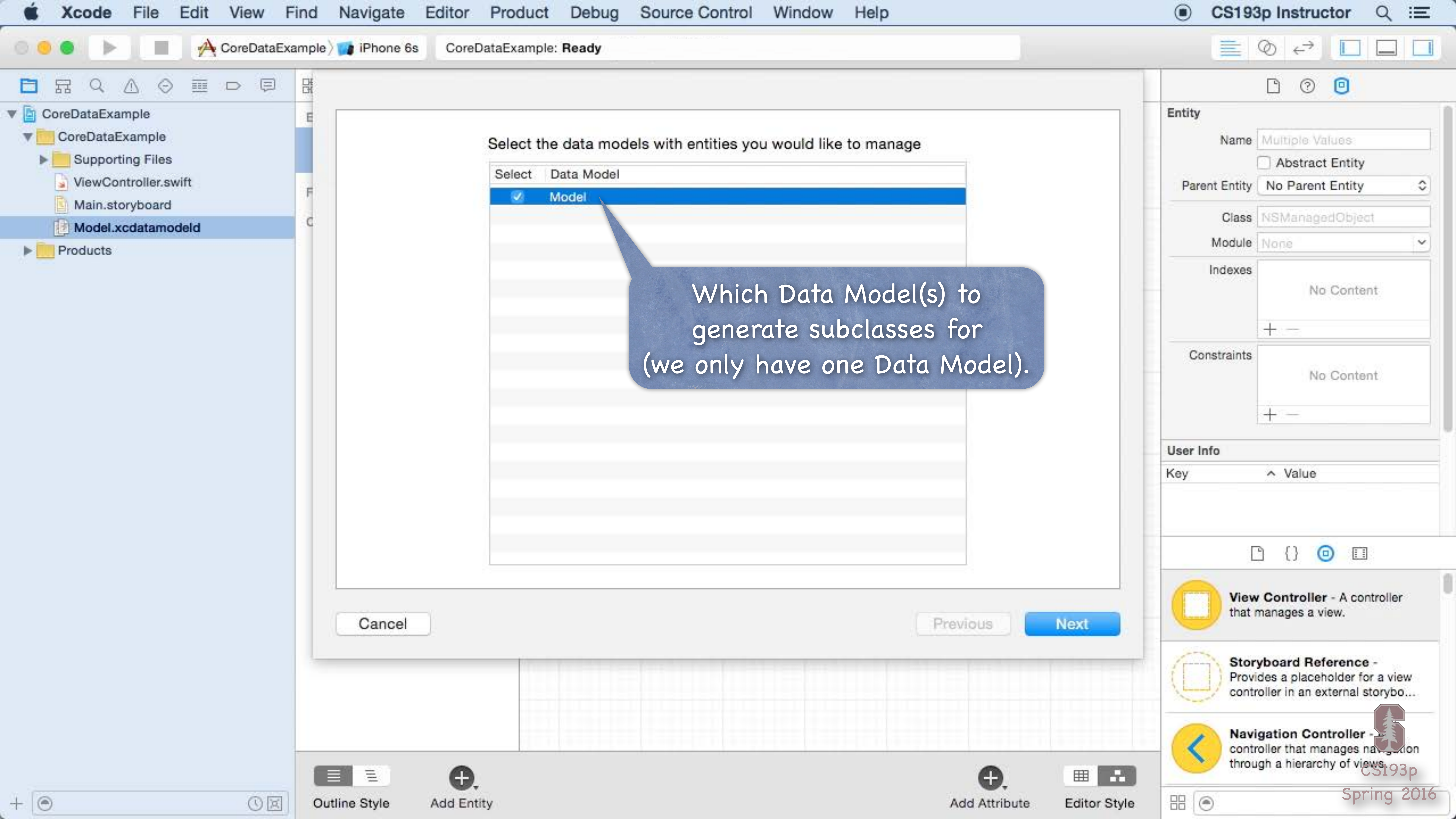
  And, as you might imagine, we can get Xcode to generate such a subclass for us!



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



Ask Xcode to generate NSManagedObject subclasses for our Entities.

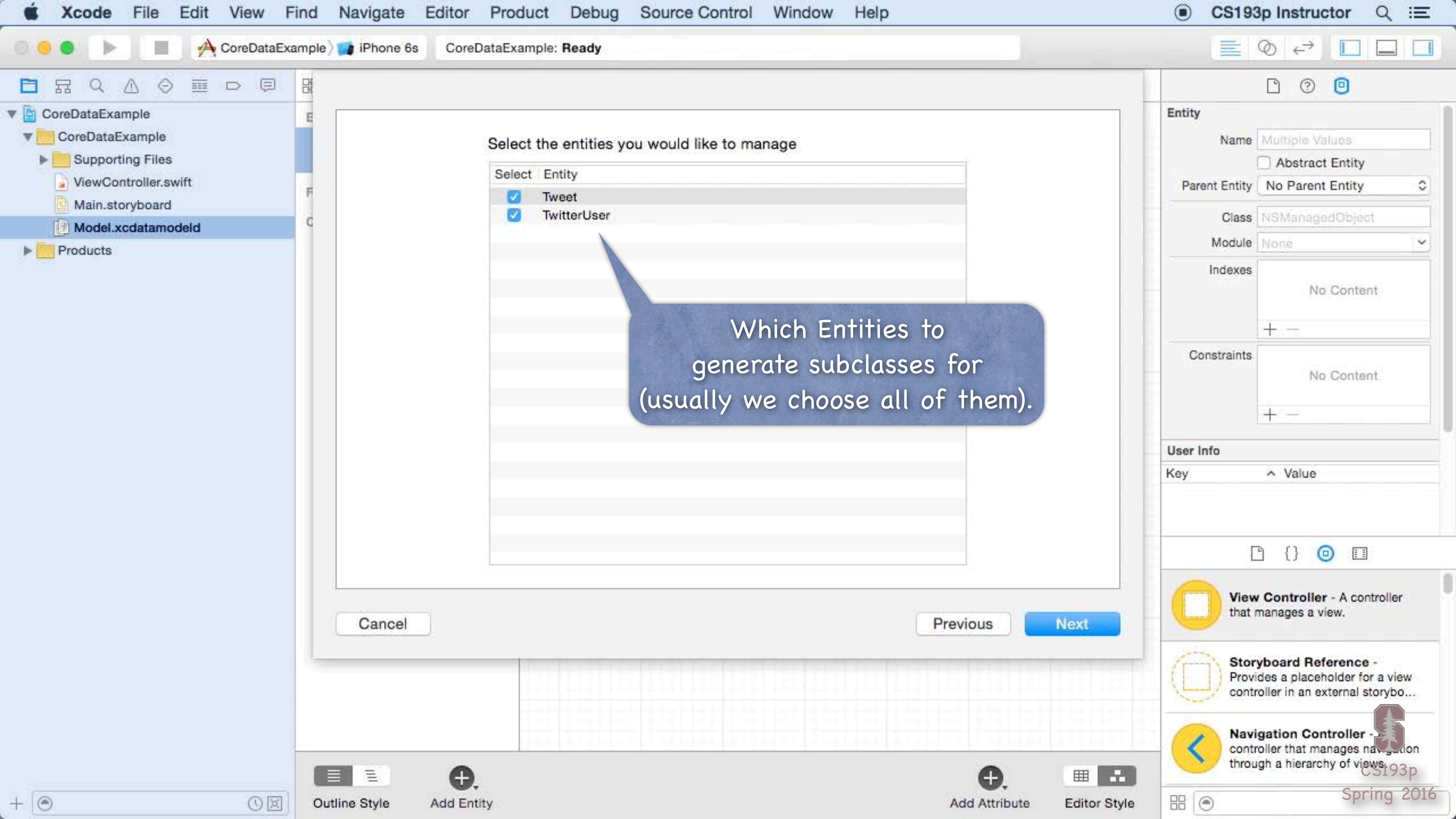


Select the data models with entities you would like to manage

Select	Data Model
✓	Model

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

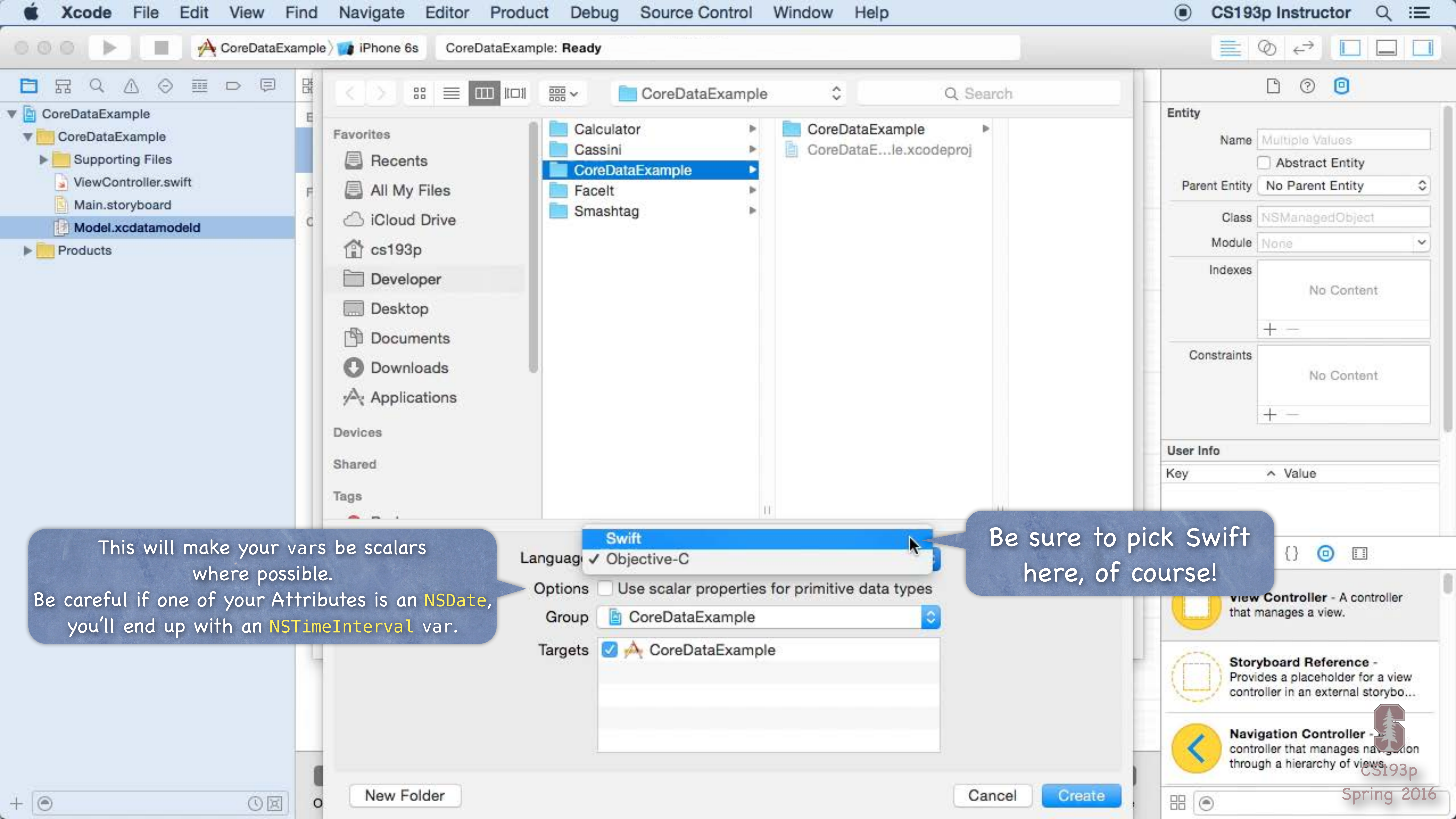
Xcode File Edit View Find Navigate Editor Product Debug Source Control Window Help
CS193p Instructor
CoreDataExample 〉 iPhone 6s
CoreDataExample: Ready
CoreDataExample
CoreDataExample
Supporting Files
ViewController.swift
Main.storyboard
Model.xcdatamodeld
Products
Cancel
Previous
Next
Entity
Name Multiple Values
Abstract Entity
Parent Entity No Parent Entity
Class NSManagedObject
Module None
Indexes No Content
Constraints No Content
User Info
Key Value
View Controller - A controller that manages a view.
Storyboard Reference - Provides a placeholder for a view controller in an external storybo...
Navigation Controller - A controller that manages navigation through a hierarchy of views.
Outline Style
Add Entity
Add Attribute
Editor Style


Xcode File Edit View Find Navigate Editor Product Debug Source Control Window Help
CS193p Instructor
CoreDataExample 〉 iPhone 6s
CoreDataExample: Ready
CoreDataExample
CoreDataExample
Supporting Files
ViewController.swift
Main.storyboard
Model.xcdatamodeld
Products
Entity
Name Multiple Values
Abstract Entity
Parent Entity No Parent Entity
Class NSManagedObject
Module None
Indexes No Content
Constraints No Content
User Info
Key Value
View Controller - A controller that manages a view.
Storyboard Reference - Provides a placeholder for a view controller in an external storybo...
Navigation Controller - A controller that manages navigation through a hierarchy of views.
Outline Style
Add Entity
Add Attribute
Editor Style

Select the entities you would like to manage

Select	Entity
✓	Tweet
✓	TwitterUser

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



Language [✓ Swift] Objective-C

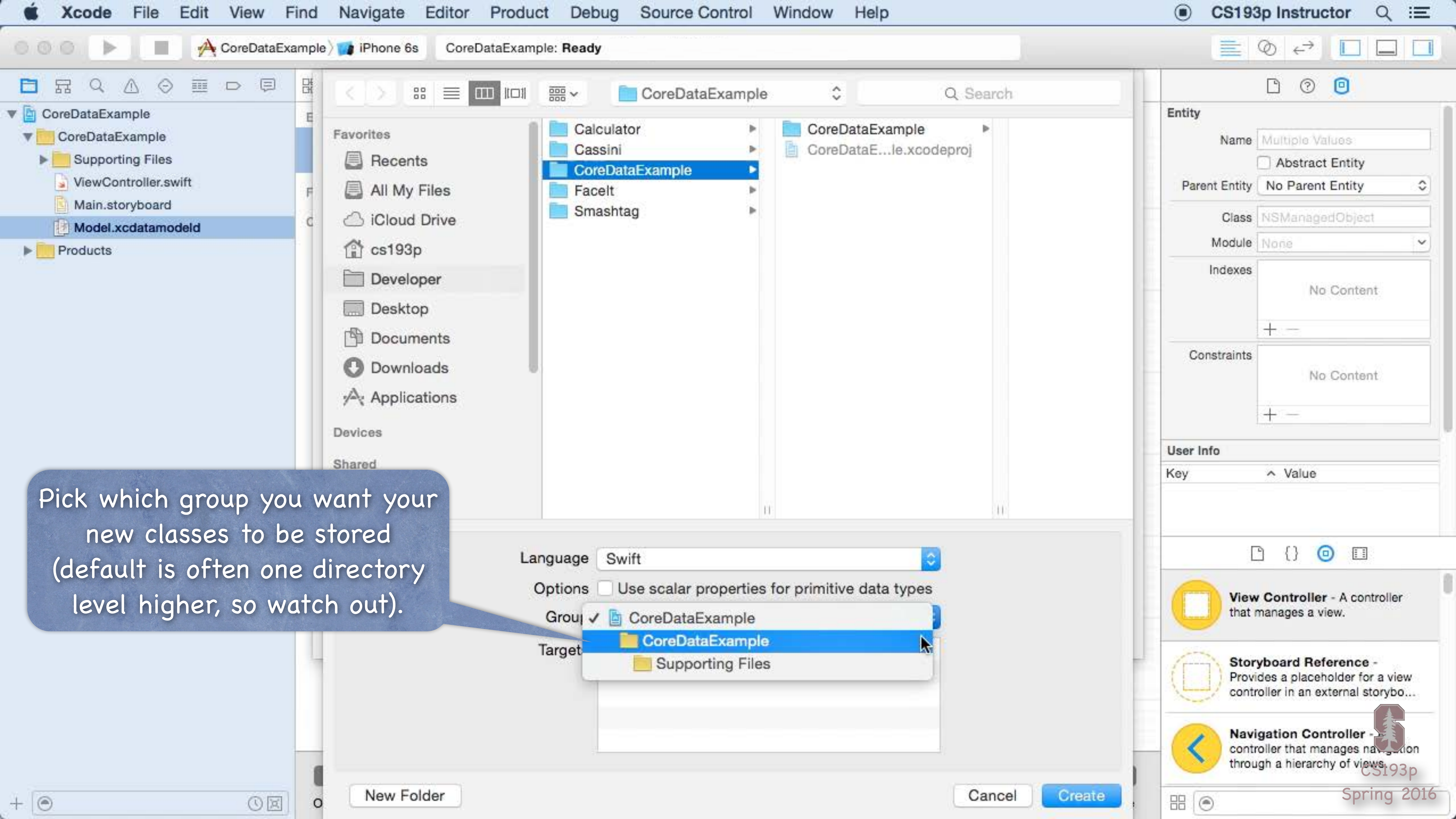
Options ☐ Use scalar properties for primitive data types

Group: CoreDataExample

Targets: ☑ CoreDataExample

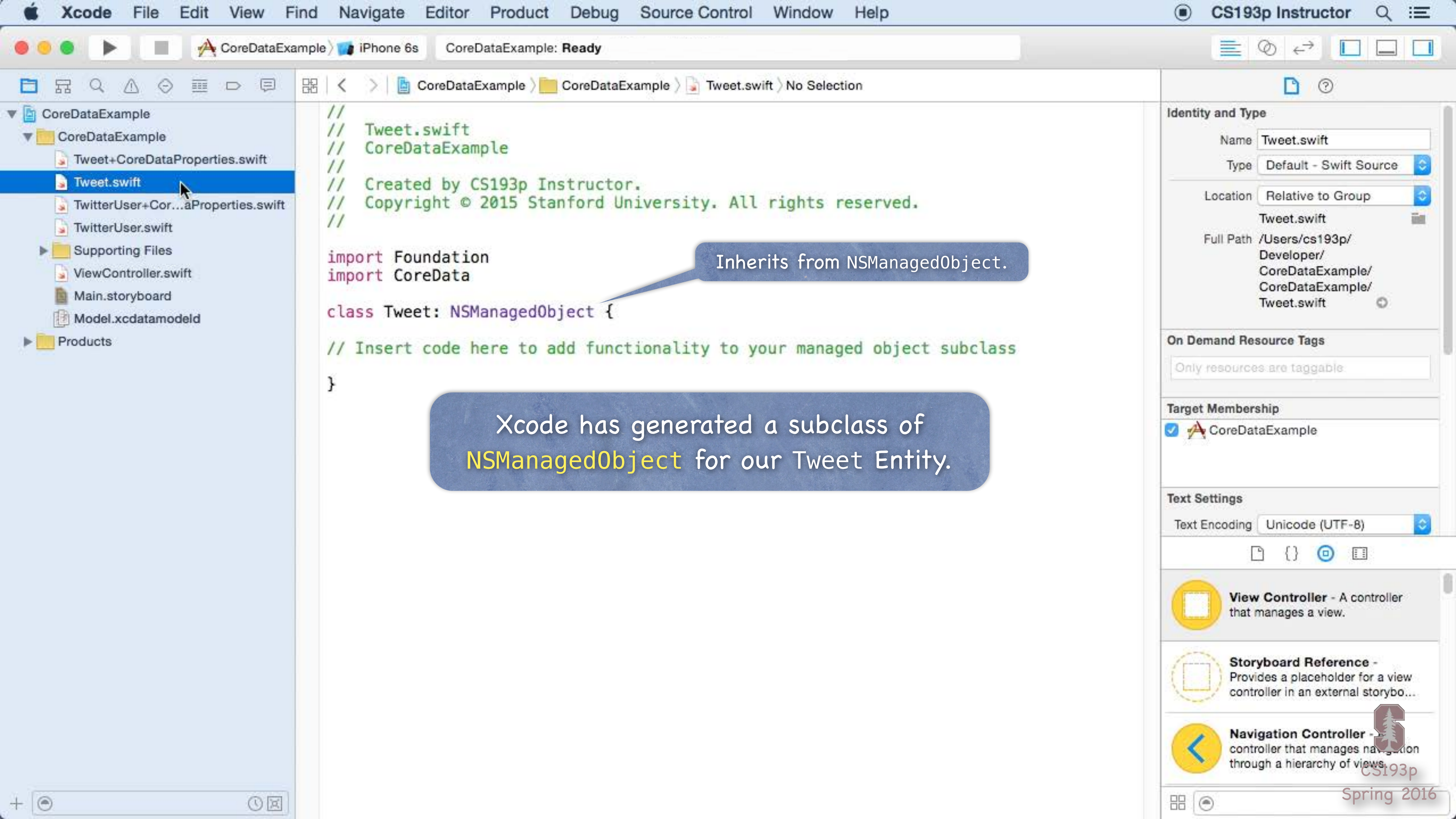
This will make your vars be scalars where possible.
Be careful if one of your Attributes is an NSDate, you'll end up with an NSTimeInterval var.

Be sure to pick Swift here, of course!



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

Language: Swift
Options: Use scalar properties for primitive data types
Group: ✓ CoreDataExample
CoreDataExample
Supporting Files

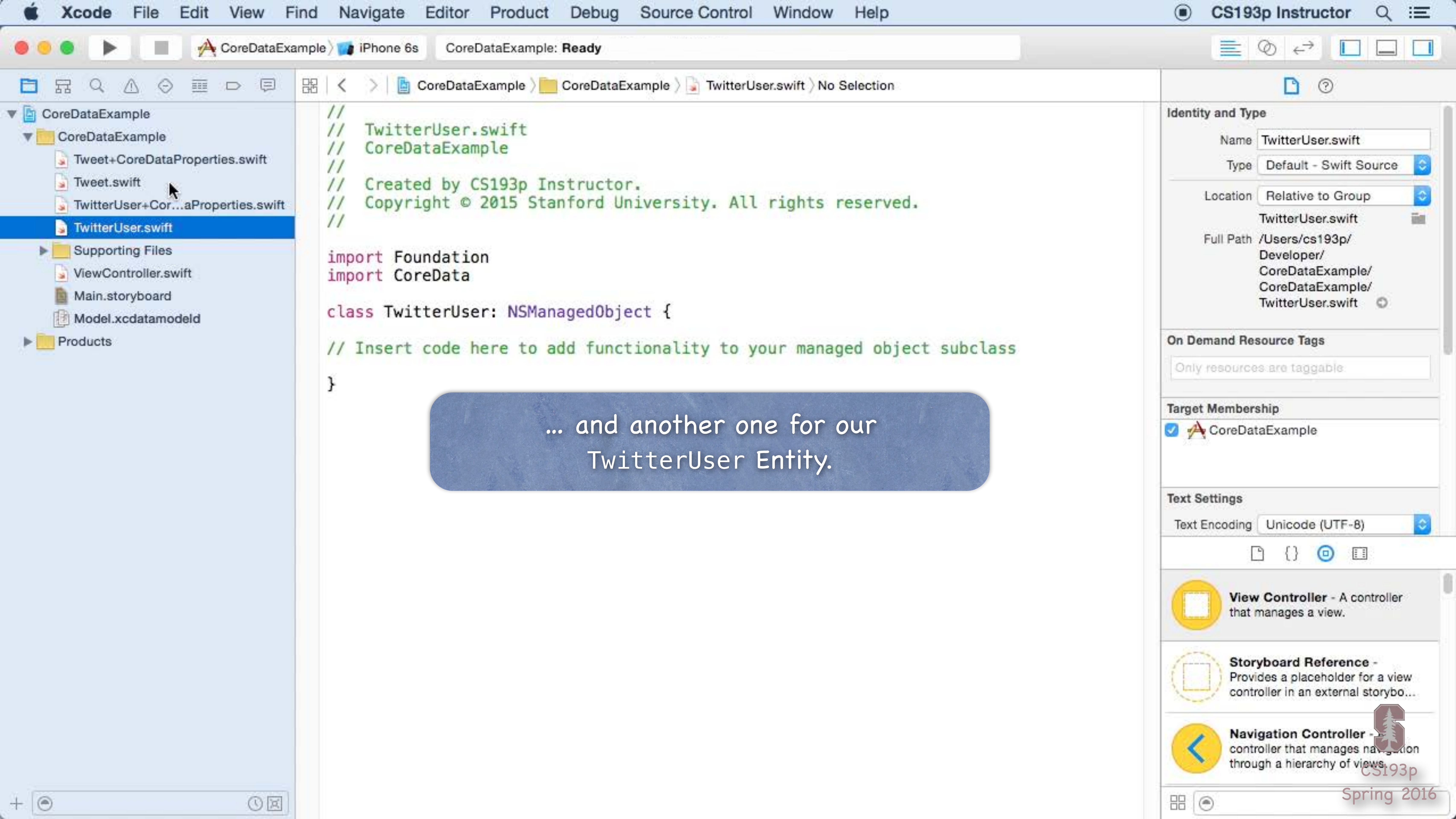


```
//
//  Tweet.swift
//  CoreDataExample
//
//  Created by CS193p Instructor.
//  Copyright © 2015 Stanford University. All rights reserved.
//

import Foundation
import CoreData

class Tweet: NSManagedObject {

// Insert code here to add functionality to your managed object subclass

}
```

Inherits from NSManagedObject.

Xcode has generated a subclass of NSManagedObject for our Tweet Entity.

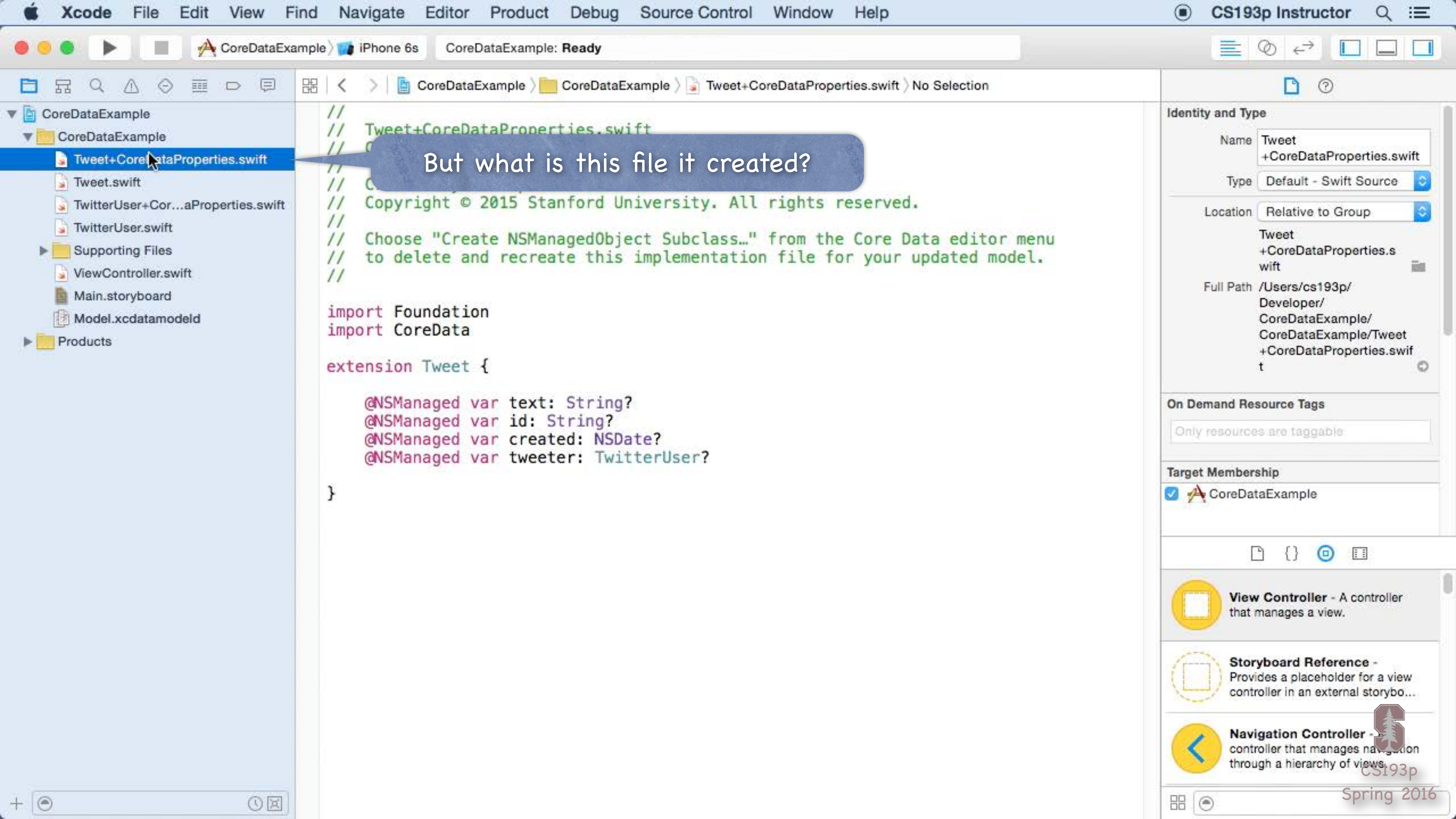
Xcode File Edit View Find Navigate Editor Product Debug Source Control Window Help
CS193p Instructor
CoreDataExample > iPhone 6s
CoreDataExample: Ready
CoreDataExample > CoreDataExample > Tweet.swift > No Selection
CoreDataExample
CoreDataExample
Tweet+CoreDataProperties.swift
Tweet.swift
TwitterUser+Cor...aProperties.swift
TwitterUser.swift
Supporting Files
ViewController.swift
Main.storyboard
Model.xcdatamodeld
Products
Identity and Type
Name Tweet.swift
Type Default - Swift Source
Location Relative to Group
Tweet.swift
Full Path /Users/cs193p/Developer/CoreDataExample/CoreDataExample/Tweet.swift
On Demand Resource Tags
Only resources are taggable
Target Membership
CoreDataExample
Text Settings
Text Encoding Unicode (UTF-8)
View Controller - A controller that manages a view.
Storyboard Reference - Provides a placeholder for a view controller in an external storybo...
Navigation Controller - A controller that manages navigation through a hierarchy of views.


Xcode File Edit View Find Navigate Editor Product Debug Source Control Window Help
CS193p Instructor
CoreDataExample › iPhone 6s
CoreDataExample: Ready
CoreDataExample › CoreDataExample › TwitterUser.swift › No Selection
CoreDataExample
CoreDataExample
Tweet+CoreDataProperties.swift
Tweet.swift
TwitterUser+Cor...aProperties.swift
TwitterUser.swift
Supporting Files
ViewController.swift
Main.storyboard
Model.xcdatamodeld
Products
Identity and Type
On Demand Resource Tags
Only resources are taggable
Target Membership
CoreDataExample
Text Settings
View Controller - A controller that manages a view.
Storyboard Reference - Provides a placeholder for a view controller in an external storybo...
Navigation Controller - controller that manages navigation through a hierarchy of views.
CS193p
Spring 2016

```swift
//
//  TwitterUser.swift
//  CoreDataExample
//
//  Created by CS193p Instructor.
//  Copyright © 2015 Stanford University. All rights reserved.
//

import Foundation
import CoreData

class TwitterUser: NSManagedObject {

// Insert code here to add functionality to your managed object subclass

}
```

Name: TwitterUser.swift
Type: Default - Swift Source
Location: Relative to Group
TwitterUser.swift
Full Path: /Users/cs193p/Developer/CoreDataExample/CoreDataExample/TwitterUser.swift
Text Encoding: Unicode (UTF-8)

… and another one for our TwitterUser Entity.


Xcode File Edit View Find Navigate Editor Product Debug Source Control Window Help
CS193p Instructor
CoreDataExample iPhone 6s
CoreDataExample: Ready
CoreDataExample
CoreDataExample
Tweet+CoreDataProperties.swift
Tweet.swift
TwitterUser+Cor...aProperties.swift
TwitterUser.swift
Supporting Files
ViewController.swift
Main.storyboard
Model.xcdatamodeld
Products
CoreDataExample 〉 CoreDataExample 〉 Tweet+CoreDataProperties.swift 〉 No Selection

But what is this file it created?

```
//
//  Tweet+CoreDataProperties.swift
//  C
//
//  C
//  Copyright © 2015 Stanford University. All rights reserved.
//
//  Choose "Create NSManagedObject Subclass…" from the Core Data editor menu
//  to delete and recreate this implementation file for your updated model.
//

import Foundation
import CoreData

extension Tweet {

    @NSManaged var text: String?
    @NSManaged var id: String?
    @NSManaged var created: NSDate?
    @NSManaged var tweeter: TwitterUser?

}
```

Identity and Type

Name: Tweet+CoreDataProperties.swift

Type: Default - Swift Source

Location: Relative to Group

Tweet+CoreDataProperties.swift

Full Path: /Users/cs193p/Developer/CoreDataExample/CoreDataExample/Tweet+CoreDataProperties.swift

On Demand Resource Tags

Only resources are taggable

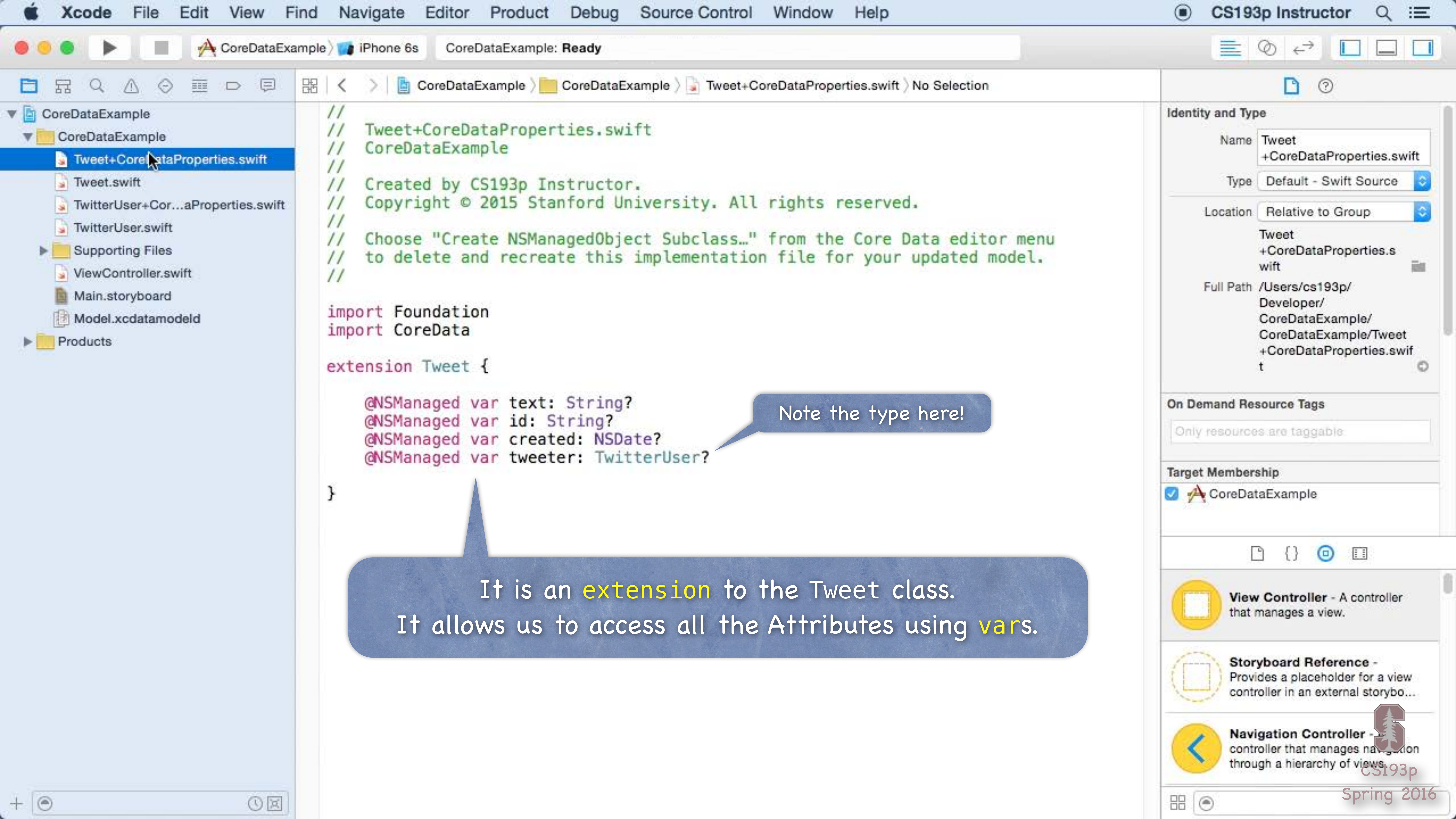
Target Membership

☑ CoreDataExample

View Controller - A controller that manages a view.

Storyboard Reference - Provides a placeholder for a view controller in an external storybo...

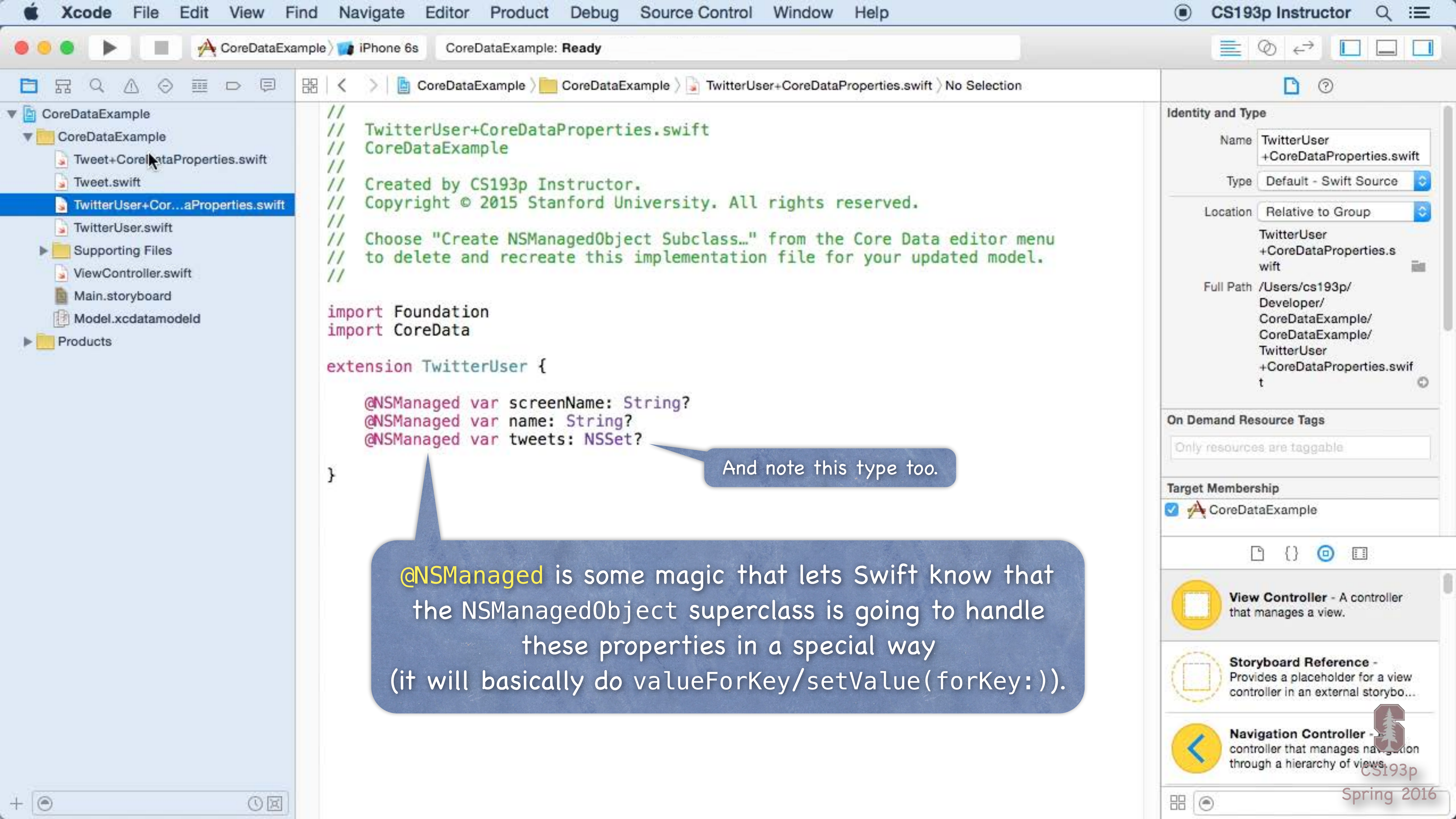
Navigation Controller - controller that manages navigation through a hierarchy of views.


Xcode File Edit View Find Navigate Editor Product Debug Source Control Window Help
CS193p Instructor
CoreDataExample › iPhone 6s
CoreDataExample: Ready
CoreDataExample › CoreDataExample › Tweet+CoreDataProperties.swift › No Selection
CoreDataExample
CoreDataExample
Tweet+CoreDataProperties.swift
Tweet.swift
TwitterUser+Cor...aProperties.swift
TwitterUser.swift
Supporting Files
ViewController.swift
Main.storyboard
Model.xcdatamodeld
Products
Identity and Type
Name Tweet+CoreDataProperties.swift
Type Default - Swift Source
Location Relative to Group
Tweet+CoreDataProperties.swift
Full Path /Users/cs193p/Developer/CoreDataExample/CoreDataExample/Tweet+CoreDataProperties.swift
On Demand Resource Tags
Only resources are taggable
Target Membership
CoreDataExample
View Controller - A controller that manages a view.
Storyboard Reference - Provides a placeholder for a view controller in an external storybo...
Navigation Controller - A controller that manages navigation through a hierarchy of views.
CS193p
Spring 2016

```swift
//
//  Tweet+CoreDataProperties.swift
//  CoreDataExample
//
//  Created by CS193p Instructor.
//  Copyright © 2015 Stanford University. All rights reserved.
//
//  Choose "Create NSManagedObject Subclass…" from the Core Data editor menu
//  to delete and recreate this implementation file for your updated model.
//

import Foundation
import CoreData

extension Tweet {

    @NSManaged var text: String?
    @NSManaged var id: String?
    @NSManaged var created: NSDate?
    @NSManaged var tweeter: TwitterUser?

}
```

Note the type here!

It is an extension to the Tweet class.
It allows us to access all the Attributes using vars.



Xcode File Edit View Find Navigate Editor Product Debug Source Control Window Help
CS193p Instructor
CoreDataExample › iPhone 6s
CoreDataExample: Ready
CoreDataExample › CoreDataExample › TwitterUser+CoreDataProperties.swift › No Selection
CoreDataExample
CoreDataExample
Tweet+CoreDataProperties.swift
Tweet.swift
TwitterUser+Cor...aProperties.swift
TwitterUser.swift
Supporting Files
ViewController.swift
Main.storyboard
Model.xcdatamodeld
Products
Identity and Type
Name TwitterUser+CoreDataProperties.swift
Type Default - Swift Source
Location Relative to Group
TwitterUser+CoreDataProperties.swift
Full Path /Users/cs193p/Developer/CoreDataExample/CoreDataExample/TwitterUser+CoreDataProperties.swift
On Demand Resource Tags
Only resources are taggable
Target Membership
CoreDataExample
View Controller - A controller that manages a view.
Storyboard Reference - Provides a placeholder for a view controller in an external storybo...
Navigation Controller - A controller that manages navigation through a hierarchy of views.

```swift
//
//  TwitterUser+CoreDataProperties.swift
//  CoreDataExample
//
//  Created by CS193p Instructor.
//  Copyright © 2015 Stanford University. All rights reserved.
//
//  Choose "Create NSManagedObject Subclass…" from the Core Data editor menu
//  to delete and recreate this implementation file for your updated model.
//

import Foundation
import CoreData

extension TwitterUser {

    @NSManaged var screenName: String?
    @NSManaged var name: String?
    @NSManaged var tweets: NSSet?

}
```

And note this type too.

@NSManaged is some magic that lets Swift know that the NSManagedObject superclass is going to handle these properties in a special way (it will basically do valueForKey/setValue(forKey:)).

Core Data

- So how do I access my Entities' Attributes with these subclasses?

```
// let's create an instance of the Tweet Entity in the database ...
let context = document.managedObjectContext // or from AppDelegate
if let tweet = NSEntityDescription.insertNewObjectForEntityForName("Tweet",
                        inManagedObjectContext:context) as? Tweet {
    tweet.text = "140 characters of pure joy"
    tweet.created = NSDate()
    tweet.tweeter = ... // a TwitterUser object we created or queried to get
    tweet.tweeter.name = "Joe Schmo" // yes, of course you can chain as usual
}
```

This is nicer than `setValue("140 characters of pure joy", forKey: "text")`

And Swift can type-check the key.

Deletion

- Deletion

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

  ```
  managedObjectContext.deleteObject(tweet)
  ```

  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 tweet after you delete it!

- prepareForDeletion

  This is a method we can implement in our NSManagedObject subclass …

  ```
  func prepareForDeletion()
  {
      // we don't need to set our tweeter to nil or anything here (that will happen automatically)
      // but if TwitterUser had, for example, a "number of tweets tweeted" attribute,
      //     we might adjust it down by one here (e.g. tweeter.tweetCount -= 1).
  }
  ```

Querying

- So far you can ...
  - Create objects in the database: `insertNewObjectForEntityForName(inManagedObjectContext:)`.
  - Get/set properties with `valueForKey/setValue(forKey:)` or vars in a custom subclass.
  - Delete objects using the `NSManagedObjectContext deleteObject` method.
- One very important thing left to know how to do: QUERY
  - Basically you need to be able to retrieve objects from the database, not just create new ones.
  - You do this by executing an `NSFetchRequest` in your `NSManagedObjectContext`.
- Four important things involved in creating an `NSFetchRequest`
  1. Entity to fetch (required)
  2. How many objects to fetch at a time and/or maximum to fetch (optional, default: all)
  3. `NSSortDescriptor`s to specify the order in which the array of fetched objects are returned
  4. `NSPredicate` specifying which of those Entities to fetch (optional, default is all of them)

Querying

- Creating an NSFetchRequest

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

```
let request = NSFetchRequest(entityName: "Tweet")
request.fetchBatchSize = 20
request.fetchLimit = 100
request.sortDescriptors = [sortDescriptor]
request.predicate = ...
```

- Specifying the kind of Entity we want to fetch

  A given fetch returns objects all of the same kind of Entity.
  You can't have a fetch that returns some Tweets and some TwitterUsers (it's one or the other).

- Setting fetch sizes/limits

  If you created a fetch that would match 1000 objects, the request above faults 20 at a time.
  And it would stop fetching after it had fetched 100 of the 1000.

Querying

- NSSortDescriptor

  When we execute a fetch request, it's going to return an Array of NSManagedObjects.
  Arrays are "ordered," of course, so we should specify that order when we fetch.

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

```
let sortDescriptor = NSSortDescriptor(
    key: "screenName",
    ascending: true,
    selector: #selector(NSString.localizedStandardCompare(_:))
)
```

  The selector: argument is just a method (conceptually) sent to each object to compare it to others.
  Some of these "methods" might be smart (i.e. they can happen on the database side).
  localizedStandardCompare is for ordering strings like the Finder on the Mac does (very common).

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

Querying

- NSPredicate

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

- Predicate formats

  You create them with a format string with strong semantic meaning (see NSPredicate doc).

  Note that we use %@ (more like printf) rather than \(expression) to specify variable data.

```
let searchString = “foo”
let predicate = NSPredicate(format: “text contains[c] %@“, searchString)
let joe: TwitterUser = ... // a TwitterUser we inserted or queried from the database
let predicate = NSPredicate(format: “tweeter = %@ && created > %@”, joe, aDate)
let predicate = NSPredicate(format: “tweeter.screenName = %@“, “CS193p”)
```

  The above would all be predicates for searches in the Tweet table only.

  Here’s a predicate for an interesting search for TwitterUsers instead …

```
let predicate = NSPredicate(format: “tweets.text contains %@“, searchString)
```

  This would be used to find TwitterUsers (not Tweets) who have tweets that contain the string.

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.

  ```
  let array = [predicate1, predicate2]
  let predicate = NSCompoundPredicate(andPredicateWithSubpredicates: array)
  ```

  This predicate is “predicate1 AND predicate2”. OR available too, of course.

Advanced Querying

- Key Value Coding

  Can actually do predicates like `“tweets.@count > 5”` (TwitterUsers with more than 5 tweets).

  `@count` is a function (there are others) executed in the database itself.

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

  By the way, all this stuff (and more) works on `Dictionarys`, `Arrays` and `Sets` too …

  e.g. `propertyList.valueForKeyPath(“tweets.tweet.@avg.latitude”)`

  returns the average latitude of the location of all the tweets (yes, really)

  e.g. `“tweets.tweet.text.length"` would return an `Array` of the lengths of the text of the tweets

- `NSExpression`

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

  No time to cover it now, unfortunately.

  If interested, for both `NSExpression` and Key Value Coding queries, investigate …

```
let request = NSFetchRequest(“…”)
request.resultType = .DictionaryResultType // fetch returns Array of Dicts instead of NSMO’s
request.propertiesToFetch = [“name”, expression, etc.]
```

Querying

- Putting it all together

  Let's say we want to query for all TwitterUsers …

```
let request = NSFetchRequest(entityName: "TwitterUser")
```

  … who have created a tweet in the last 24 hours …

```
let yesterday = NSDate(timeIntervalSinceNow:-24*60*60)
request.predicate = NSPredicate(format: "any tweets.created > %@", yesterday)
```

  … sorted by the TwitterUser's name …

```
request.sortDescriptors = [NSSortDescriptor(key: "name", ascending: true)]
```

Querying

- Executing the fetch

```
context = aDocument.managedObjectContext // or AppDelegate var
let users = try? context.executeFetchRequest(request)
```

Notice we are doing a different kind of `try?` here.
The ? means "`try` this and if it `throws` an error, just give me `nil` back."
We could, of course, use a normal `try` inside a `do { }` and `catch` errors if we were interested.

Otherwise this fetch request executing method …
Returns an empty `Array` (not `nil`) if it succeeds and there are no matches in the database.
Returns an `Array` of `NSManagedObjects` (or subclasses thereof) if there were any matches.

That's it. Very simple really.

Query Results

- Faulting

The above fetch does not necessarily fetch any actual data.
It could be an array of "as yet unfaulted" objects, waiting for you to access their attributes.
Core Data is very smart about "faulting" the data in as it is actually accessed.
For example, if you did something like this …

```
for user in twitterUsers) {
    print("fetched user \(user)")
}
```

You may or may not see the names of the users in the output
(you might just see "unfaulted object", depending on whether it prefetched them)
But if you did this …

```
for user in twitterUsers) {
    print("fetched user named \(user.name)")
}
```

… then you would definitely fault all these `TwitterUsers` in from the database.
That's because in the second case, you actually access the `NSManagedObject`'s data.

Core Data Thread Safety

- NSManagedObjectContext is not thread safe

  Luckily, Core Data access is usually very fast, so multithreading is only rarely needed.
  NSManagedObjectContexts are created using a queue-based concurrency model.
  This means that you can only touch a context and its NSMO's in the queue it was created on.
  When we say "queue" here, we mean "serial queue" not the QoS-based concurrent queues.
  The most common queue to use is the main queue (UIManagedDocument or AppDelegate).
  You can create your own NSManagedObjectContexts on other serial queues, but that's advanced.

- Thread-Safe Access to an NSManagedObjectContext

```
context.performBlock {  // or performBlockAndWait until it finishes
    // do stuff with context (this will happen in its safe queue (the queue it was created on))
}
```

  Note that the Q might well be the main Q, so you're not necessarily getting "multithreaded."
  It's generally a good idea to wrap all your calls to an NSManagedObjectContext using this.
  It won't cost anything if it's not in a multithreaded situation.

Core Data Thread Safety

- Parent Context (advanced)

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

Core Data

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

Core Data and UITableView

- NSFetchedResultsController

  Hooks an NSFetchRequest up to a UITableViewController.
  Usually you'll have an NSFetchedResultsController var in your UITableViewController.
  It will be hooked up to an NSFetchRequest that returns the data you want to show.
  Then use the NSFRC to answer all of your UITableViewDataSource protocol's questions!

- For example ...

```
var fetchedResultsController = ...
func numberOfSectionsInTableView(sender: UITableView) -> Int {
    return fetchedResultsController?.sections?.count ?? 1
}
func tableView(sender: UITableView, numberOfRowsInSection section: Int) -> Int {
    if let sections = fetchedResultsController?.sections where sections.count > 0 {
        return sections[section].numberOfObjects
    } else {
        return 0
    }
}
```

NSFetchedResultsController

- Very important method … `objectAtIndexPath`

  `NSFetchedResultsController` method …

```
func objectAtIndexPath(indexPath: NSIndexPath) -> NSManagedObject
```

  Here's how you would use it in, for example, `tableView(cellForRowAtIndexPath:)` …

```
func tableView(tv: UITableView, cellForRowAtIndexPath: NSIndexPath) -> UITableViewCell
{
    let cell = tv.dequeue…
    if let obj = fetchedResultsController.objectAtIndexPath(indexPath) as? Tweet {
        // load up the cell based on the properties of the obj
    }
    return cell
}
```

NSFetchedResultsController

- How do you create an NSFetchedResultsController?

  Just need the NSFetchRequest to drive it (and a NSManagedObjectContext to fetch from).
  Let's say we want to show all tweets posted by someone with the name theName in our table:

```
let request = NSFetchRequest(entityName: “Tweet”)
request.sortDescriptors = [NSSortDescriptor(key: “created” ...)]
request.predicate = NSPredicate(format: “tweeter.name = %@”, theName)
let frc = NSFetchedResultsController(
              fetchRequest: request,
      managedObjectContext: context,
        sectionNameKeyPath: keyThatSaysWhichAttributeIsTheSectionName,
                 cacheName: “MyTwitterQueryCache”)   // careful!
```

  Be sure that any cacheName you use is always associated with exactly the <u>same request</u>.
  It's okay to specify nil for the cacheName (no cacheing of fetch results in that case).

  It is critical that the sortDescriptor <u>matches up</u> with the keyThatSaysWhichAttribute…
  The results must sort such that all objects in the first section come first, second second, etc.

NSFetchedResultsController

- NSFRC also "watches" changes in Core Data & auto-updates table

  Uses a key-value observing mechanism.

  When it notices a change, it sends message like this to its delegate …

```
func controller(NSFetchedResultsController,
   didChangeObject: AnyObject
       atIndexPath: NSIndexPath?
     forChangeType: NSFetchedResultsChangeType
      newIndexPath: NSIndexPath?)
{
   // here you are supposed call appropriate UITableView methods to update rows
   // but don't worry, we're going to make it easy on you …
}
```

CoreDataTableViewController

- NSFetchedResultsController's doc shows how to do all this
  In fact, you're supposed to copy/paste the code from the doc into your table view subclass.
  But that's all a bit of a pain and it's not in Swift, so …
- Enter CoreDataTableViewController!
  We've put the code from NSFetchedResultsController into a subclass of UITVC for you!
- How does CoreDataTableViewController work?
  It's just a UITableViewController that adds an NSFetchedResultsController as a var.
  Whenever you set it, it will immediately start using it to fill the contents of its UITableView.
- Easy to use
  Download it along with the Core Data demo next week.
  Just subclass it and override the methods that load up cells and/or react to rows being selected
  (you'll use the NSFetchedResultsController method objectAtIndexPath mentioned earlier).
  Then set the fetchedResultsController var and watch it go!