Stanford CS193p

Developing Applications for iOS
Winter 2017

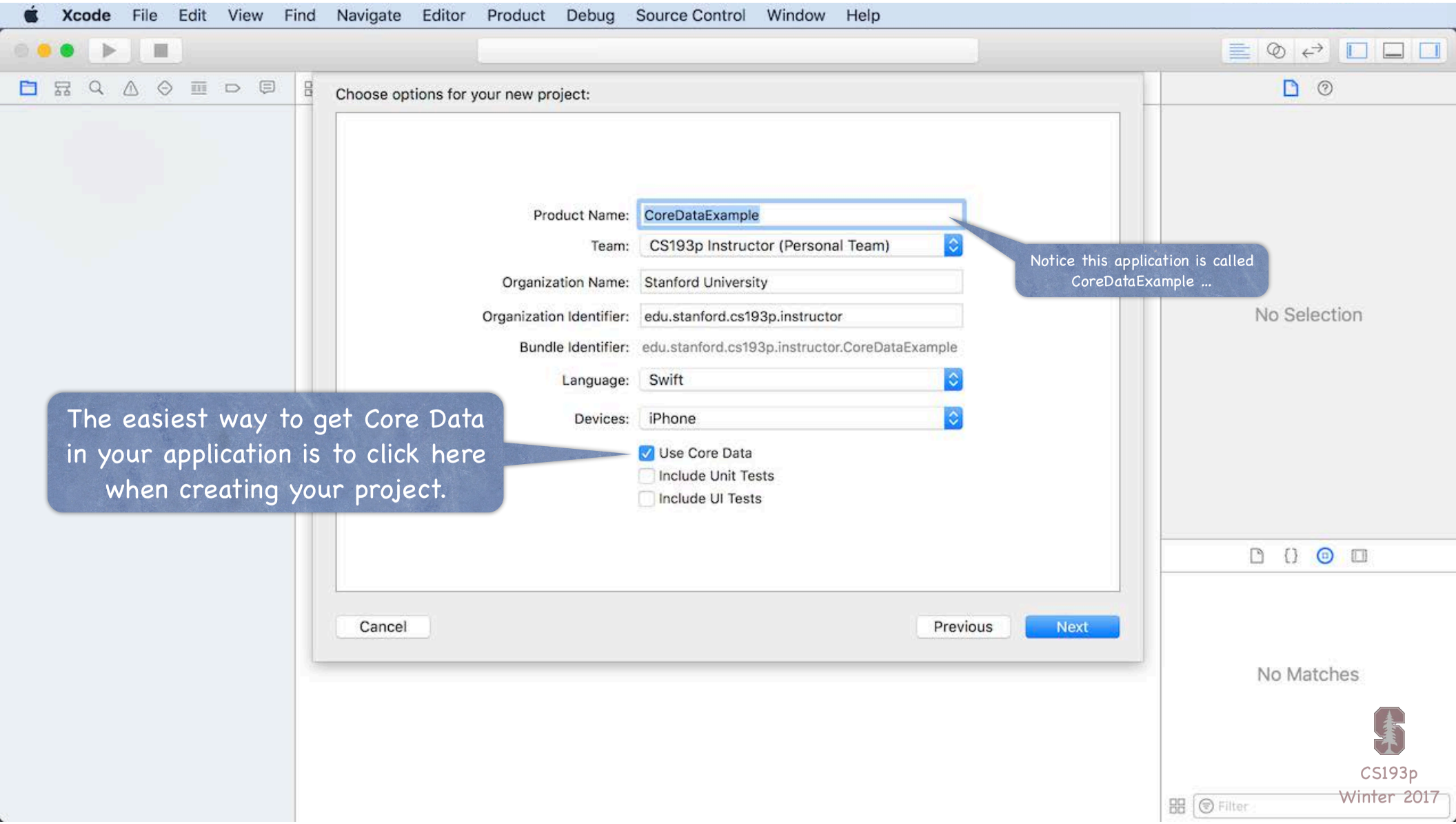
CS193p
Winter 2017

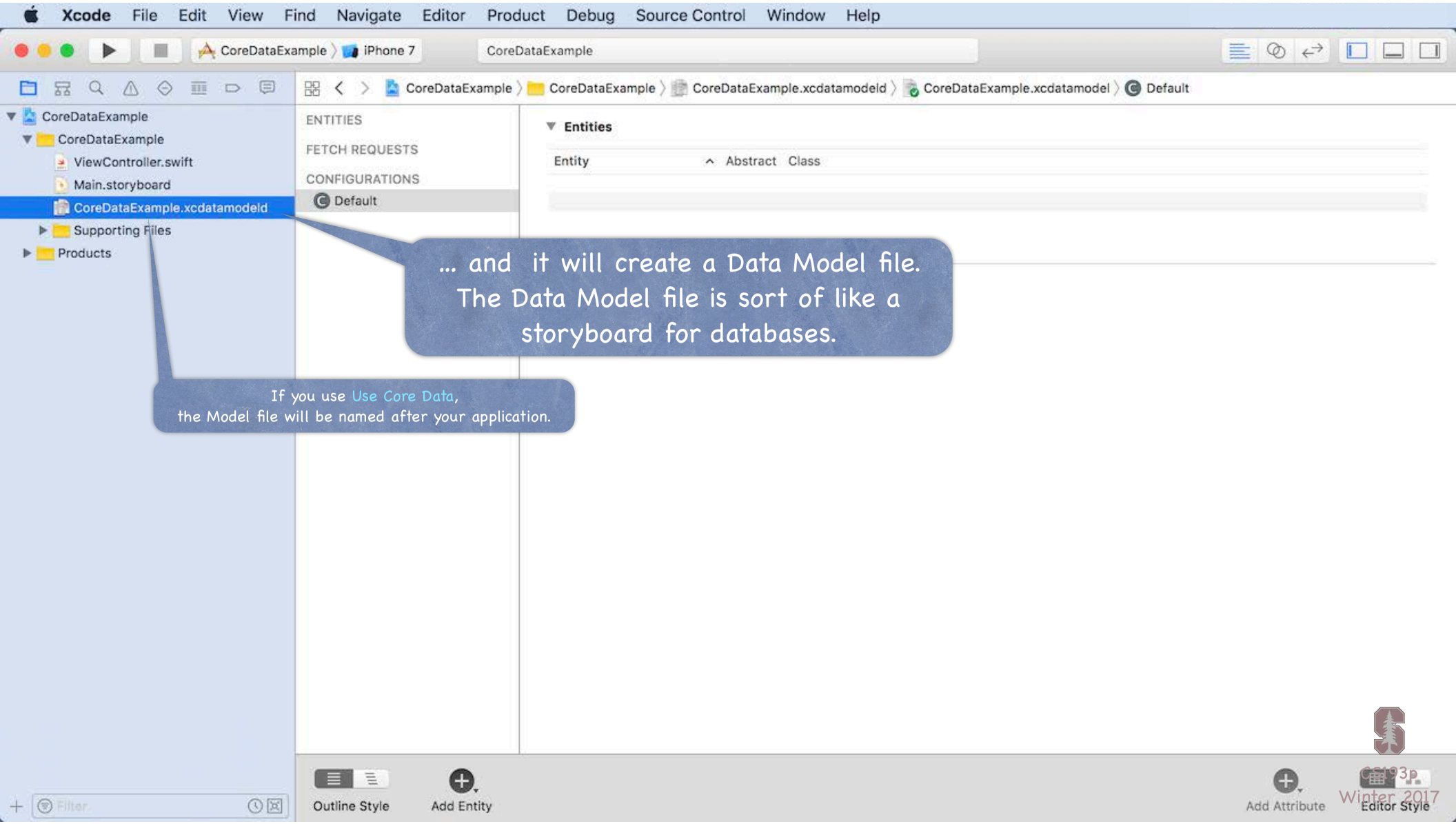
Today

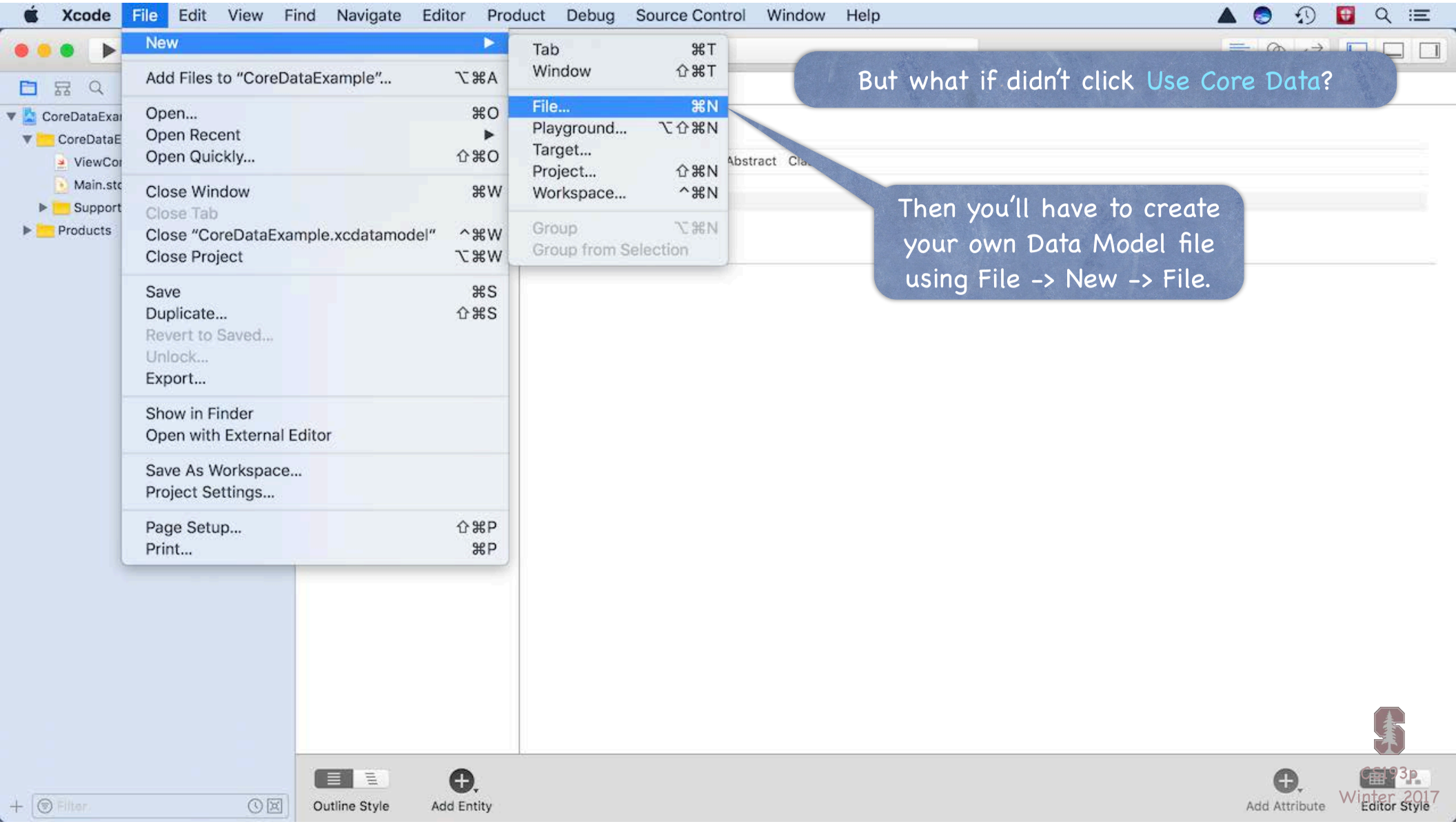
- Core Data
  Object-Oriented Database

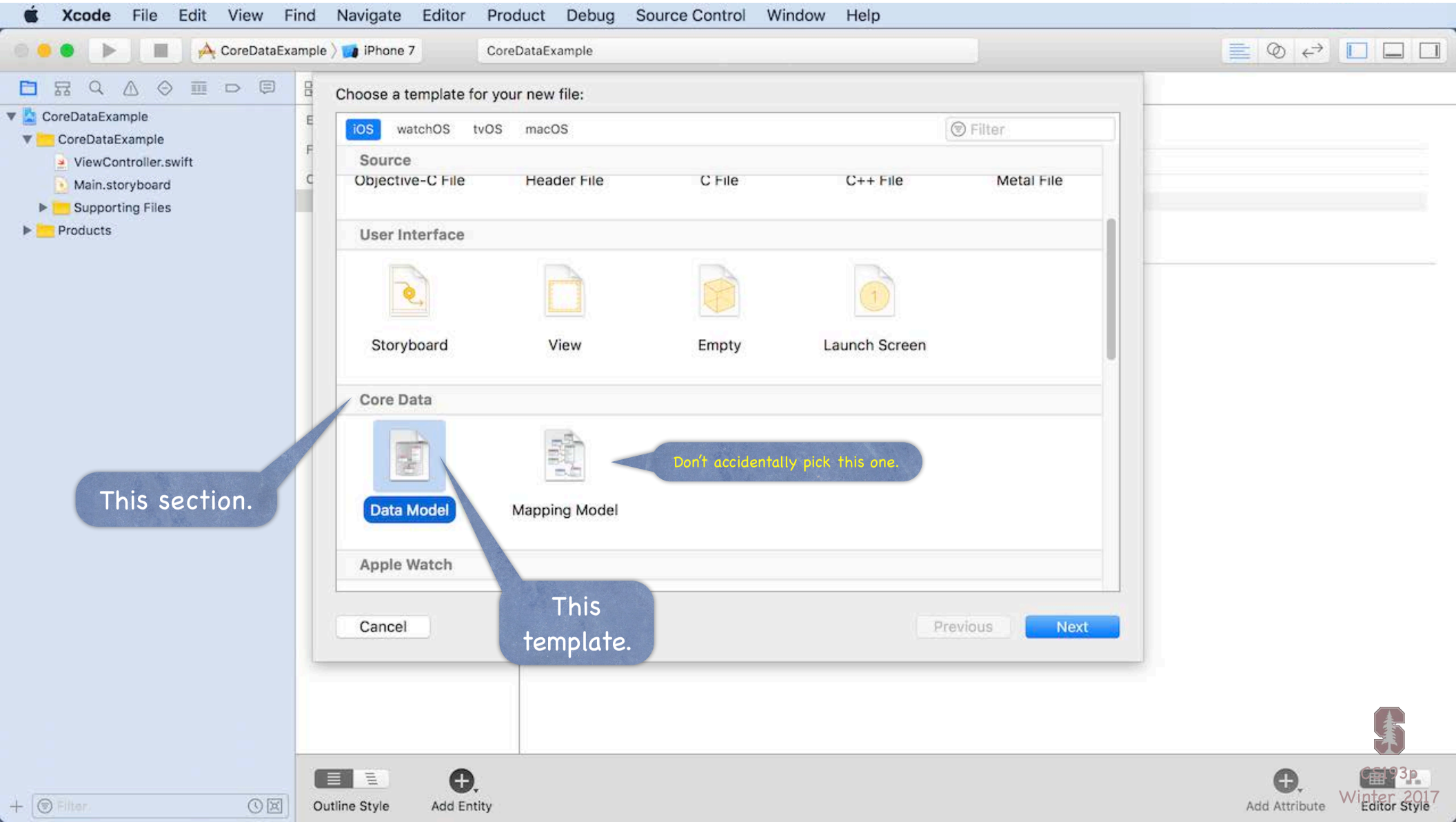
Core Data

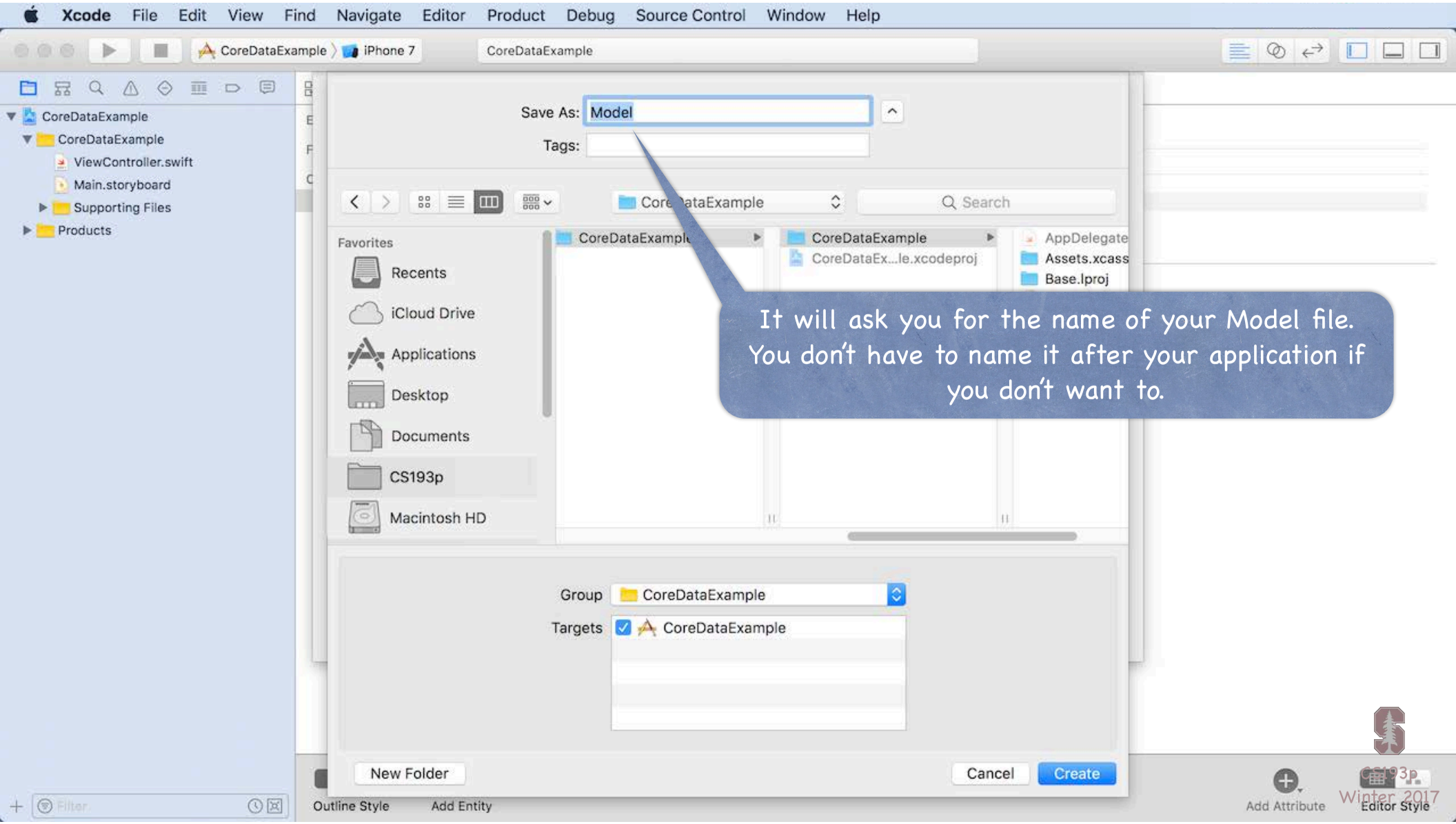
- Database
  - Sometimes you need to store large amounts of data or query it in a sophisticated manner.
  - But we still want it to be object-oriented!
- Enter Core Data
  - Object-oriented database.
  - Very, very powerful framework in iOS (we will only be covering the absolute basics).
- It's a way 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 ...

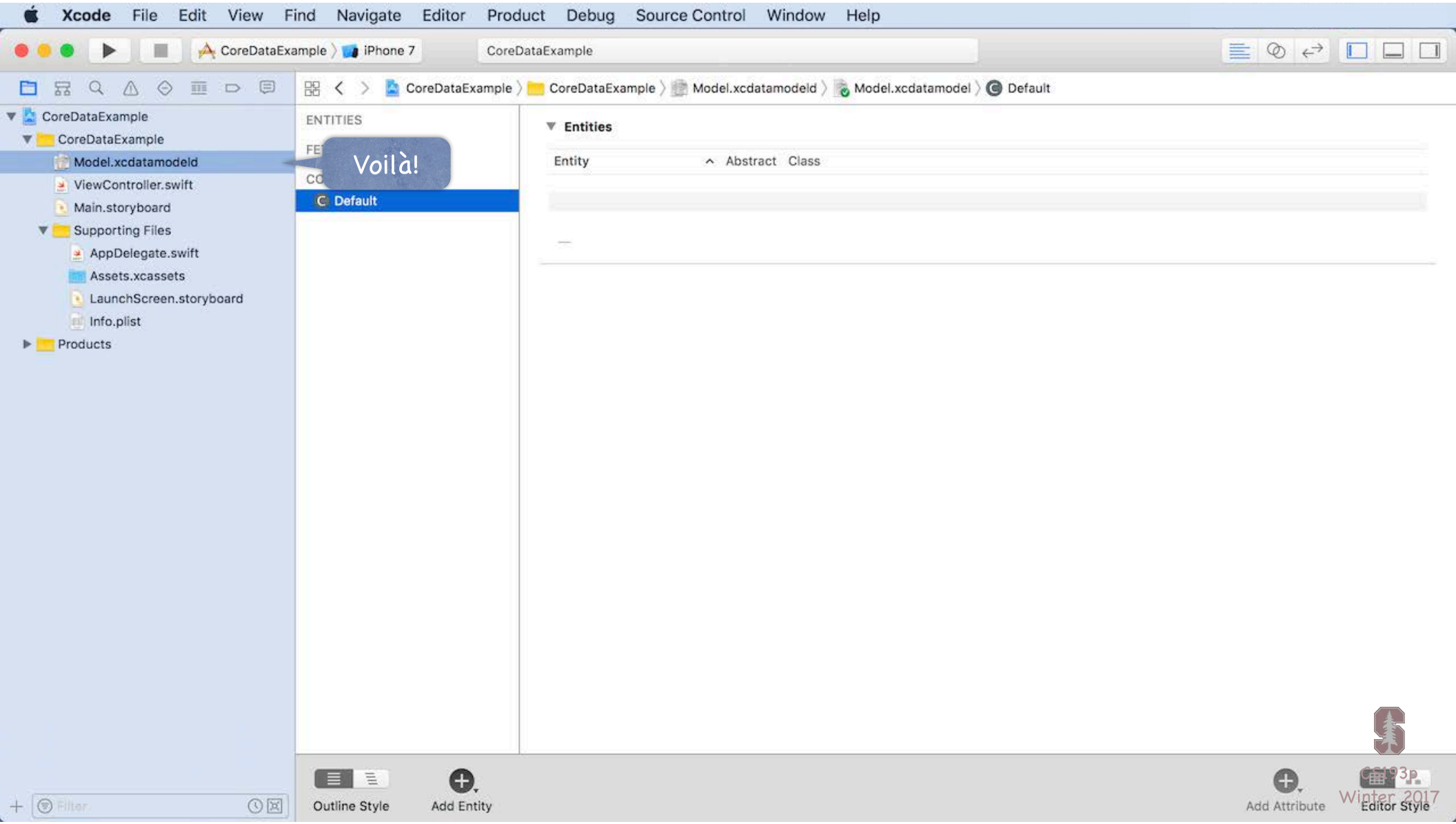

Xcode File Edit View Find Navigate Editor Product Debug Source Control Window Help
Choose options for your new project:
Product Name: CoreDataExample
Team: CS193p Instructor (Personal Team)
Organization Name: Stanford University
Organization Identifier: edu.stanford.cs193p.instructor
Bundle Identifier: edu.stanford.cs193p.instructor.CoreDataExample
Language: Swift
Devices: iPhone
Use Core Data
Include Unit Tests
Include UI Tests
Cancel
Previous
Next
Notice this application is called CoreDataExample ...
The easiest way to get Core Data in your application is to click here when creating your project.
No Selection
No Matches
CS193p
Winter 2017

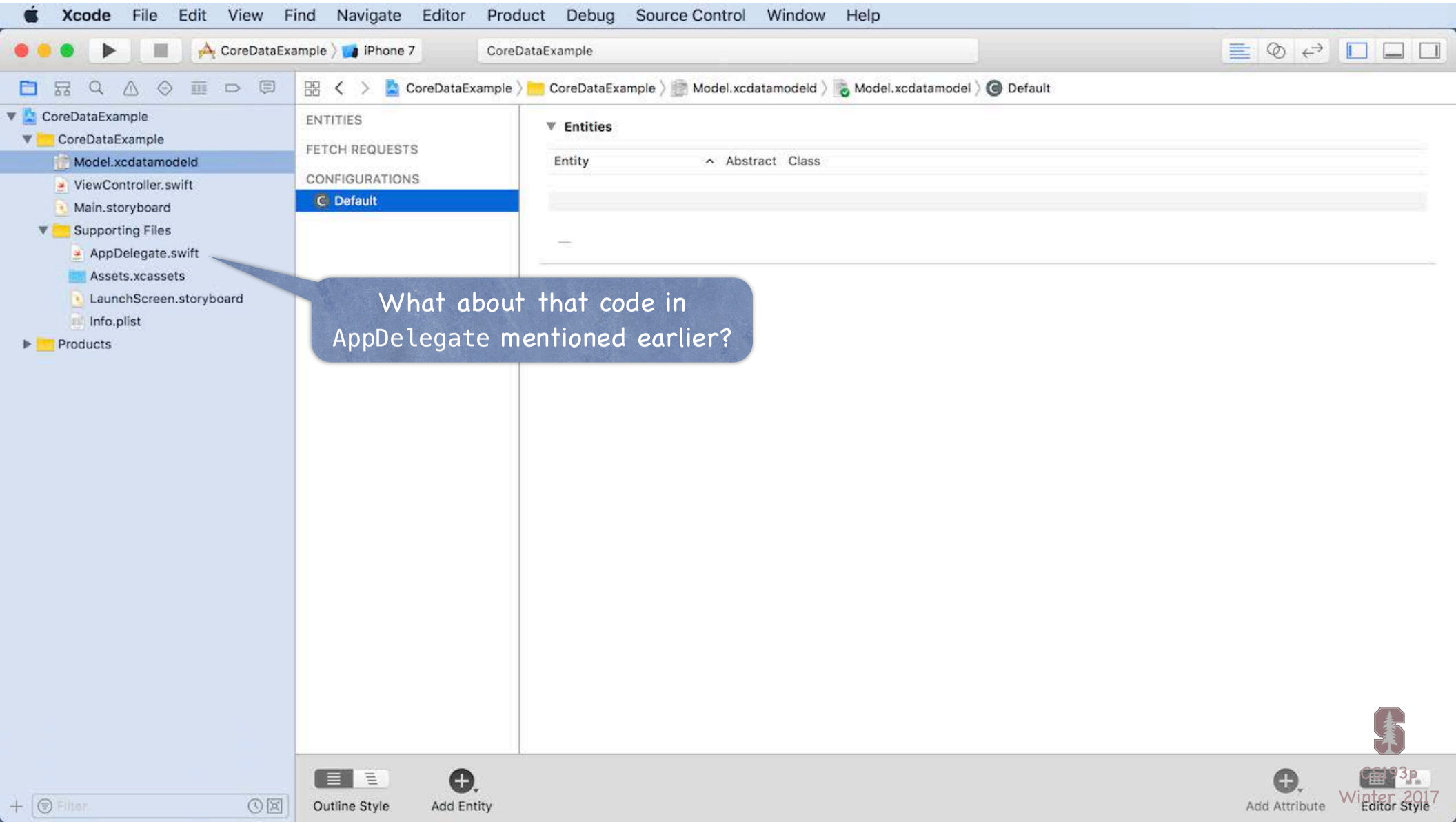

Xcode File Edit View Find Navigate Editor Product Debug Source Control Window Help
CoreDataExample 〉 iPhone 7
CoreDataExample
CoreDataExample 〉 CoreDataExample 〉 CoreDataExample.xcdatamodeld 〉 CoreDataExample.xcdatamodel 〉 Default
CoreDataExample
CoreDataExample
ViewController.swift
Main.storyboard
CoreDataExample.xcdatamodeld
Supporting Files
Products
ENTITIES
FETCH REQUESTS
CONFIGURATIONS
Default
Entities
Entity
Abstract Class
... and it will create a Data Model file. The Data Model file is sort of like a storyboard for databases.
If you use Use Core Data, the Model file will be named after your application.
Outline Style
Add Entity
Add Attribute
Editor Style
CS193p
Winter 2017

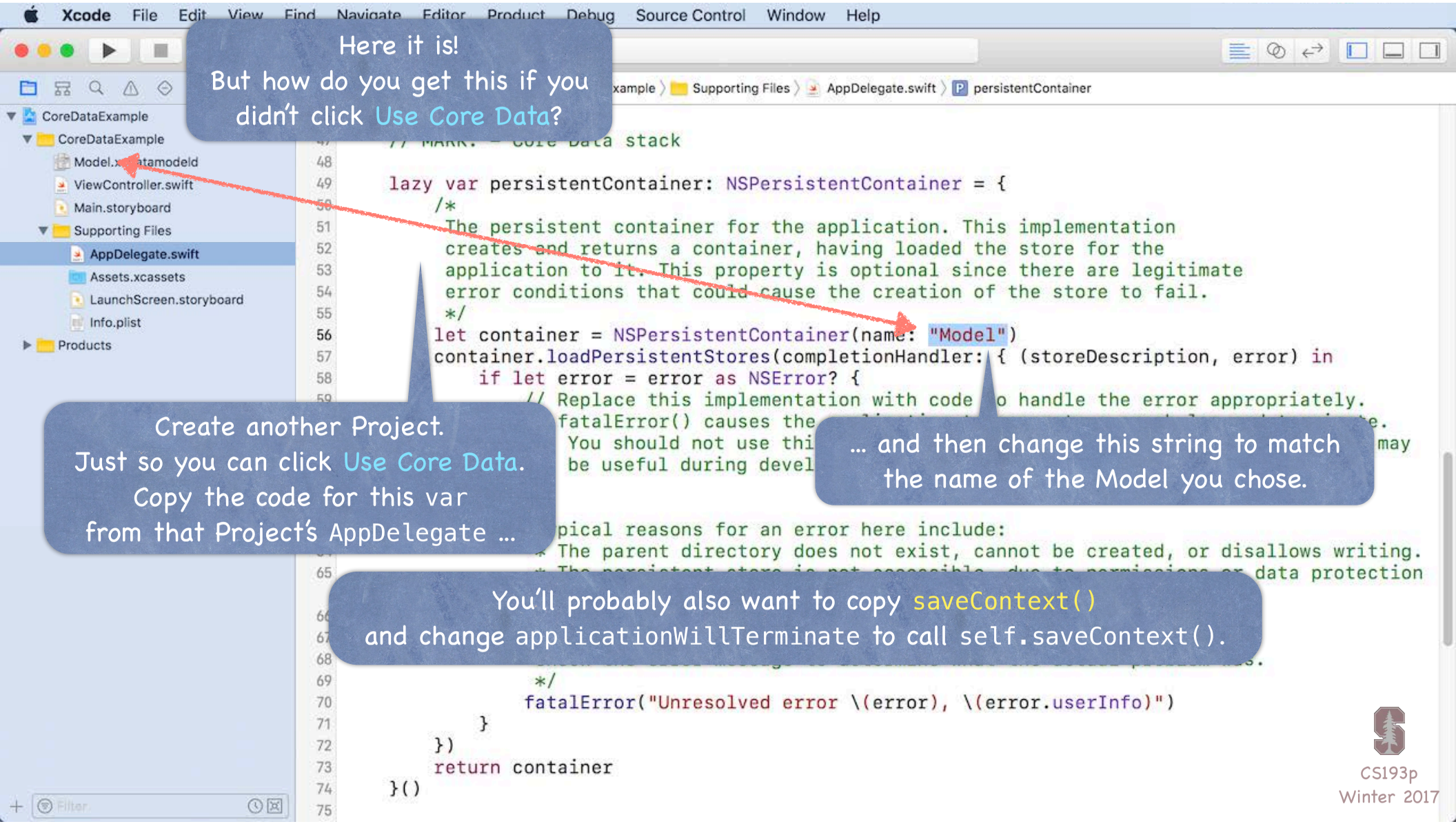

Xcode File Edit View Find Navigate Editor Product Debug Source Control Window Help
New
Add Files to "CoreDataExample"... ⌥⌘A
Open... ⌘O
Open Recent
Open Quickly... ⇧⌘O
Close Window ⌘W
Close Tab
Close "CoreDataExample.xcdatamodel" ^⌘W
Close Project ⌥⌘W
Save ⌘S
Duplicate... ⇧⌘S
Revert to Saved...
Unlock...
Export...
Show in Finder
Open with External Editor
Save As Workspace...
Project Settings...
Page Setup... ⇧⌘P
Print... ⌘P
Tab ⌘T
Window ⇧⌘T
File... ⌘N
Playground... ⌥⇧⌘N
Target...
Project... ⇧⌘N
Workspace... ^⌘N
Group ⌥⌘N
Group from Selection
But what if didn't click Use Core Data?
Then you'll have to create your own Data Model file using File -> New -> File.
Outline Style
Add Entity
Add Attribute
Editor Style
CS193p
Winter 2017

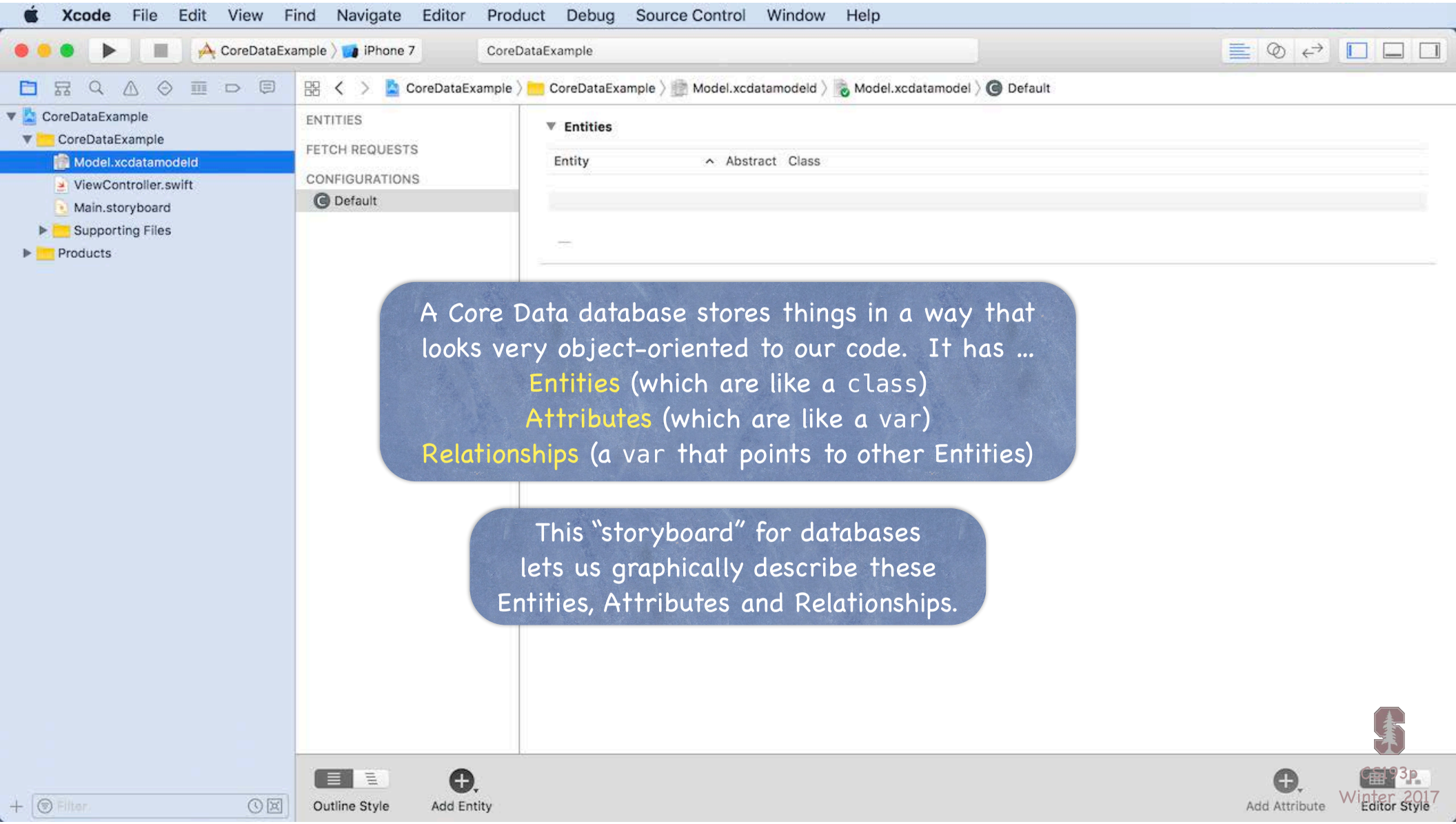

Xcode File Edit View Find Navigate Editor Product Debug Source Control Window Help
CoreDataExample iPhone 7
CoreDataExample
CoreDataExample
CoreDataExample
ViewController.swift
Main.storyboard
Supporting Files
Products
Choose a template for your new file:
iOS watchOS tvOS macOS
Filter
Source
Objective-C File
Header File
C File
C++ File
Metal File
User Interface
Storyboard
View
Empty
Launch Screen
Core Data
Data Model
Mapping Model
Don't accidentally pick this one.
Apple Watch
Cancel
Previous
Next
This section.
This template.
Outline Style
Add Entity
Add Attribute
Editor Style
CS193p
Winter 2017

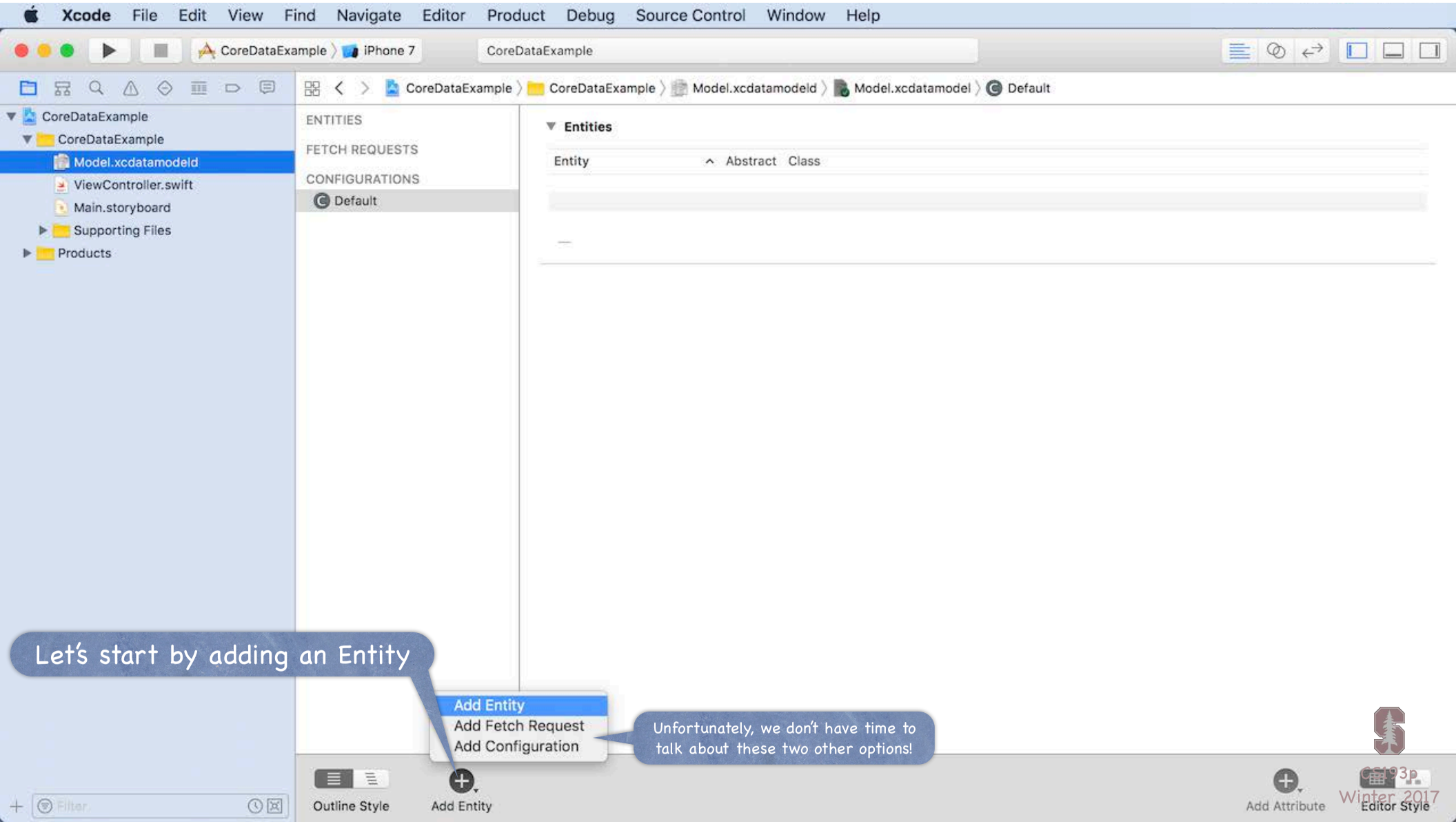

Xcode File Edit View Find Navigate Editor Product Debug Source Control Window Help
CoreDataExample iPhone 7
CoreDataExample
CoreDataExample
CoreDataExample
ViewController.swift
Main.storyboard
Supporting Files
Products
Save As: Model
Tags:
Search
Favorites
Recents
iCloud Drive
Applications
Desktop
Documents
CS193p
Macintosh HD
CoreDataExample
CoreDataEx...le.xcodeproj
AppDelegate
Assets.xcass
Base.lproj
It will ask you for the name of your Model file. You don't have to name it after your application if you don't want to.
Group CoreDataExample
Targets CoreDataExample
New Folder
Cancel
Create
Outline Style
Add Entity
Add Attribute
Editor Style
CS193p
Winter 2017

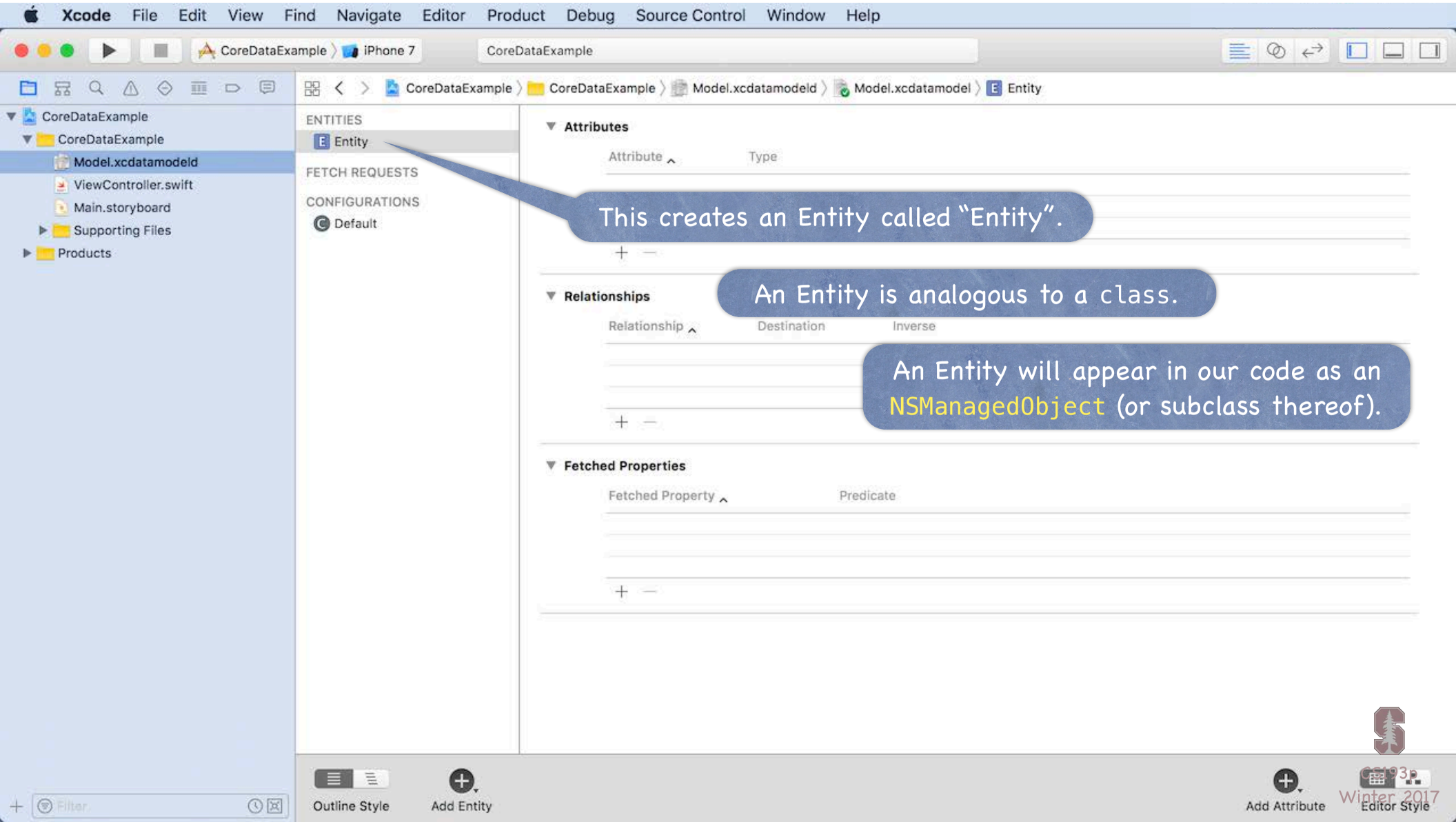

Xcode File Edit View Find Navigate Editor Product Debug Source Control Window Help
CoreDataExample 〉 iPhone 7
CoreDataExample
CoreDataExample 〉 CoreDataExample 〉 Model.xcdatamodeld 〉 Model.xcdatamodel 〉 Default
CoreDataExample
CoreDataExample
Model.xcdatamodeld
ViewController.swift
Main.storyboard
Supporting Files
AppDelegate.swift
Assets.xcassets
LaunchScreen.storyboard
Info.plist
Products
Voilà!
ENTITIES
Default
Entities
Entity
Abstract
Class
Outline Style
Add Entity
Add Attribute
Editor Style
CS193p
Winter 2017

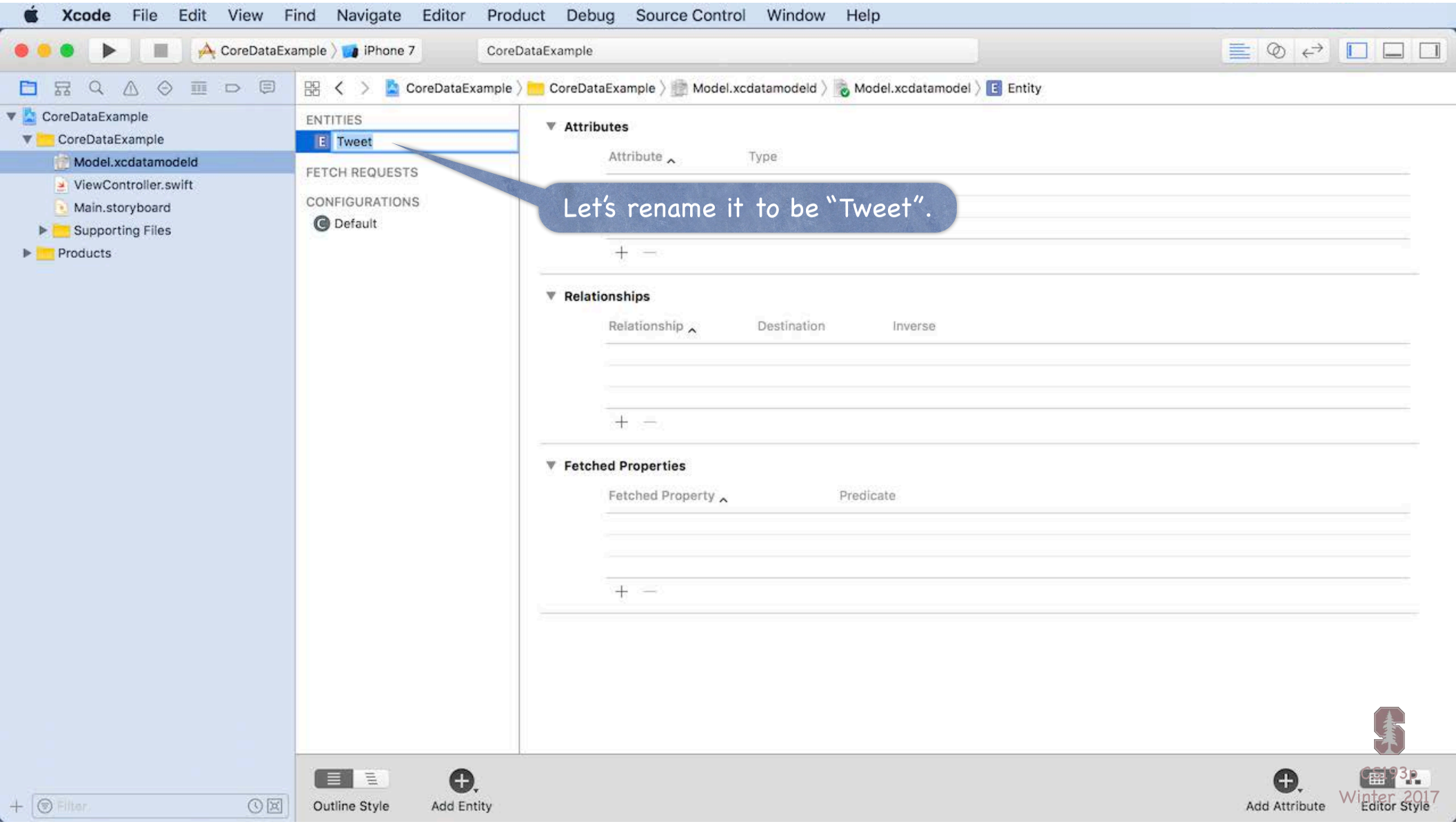

Xcode File Edit View Find Navigate Editor Product Debug Source Control Window Help
CoreDataExample 〉 iPhone 7
CoreDataExample
CoreDataExample 〉 CoreDataExample 〉 Model.xcdatamodeld 〉 Model.xcdatamodel 〉 Default
CoreDataExample
CoreDataExample
Model.xcdatamodeld
ViewController.swift
Main.storyboard
Supporting Files
AppDelegate.swift
Assets.xcassets
LaunchScreen.storyboard
Info.plist
Products
ENTITIES
FETCH REQUESTS
CONFIGURATIONS
Default
Entities
Entity
Abstract
Class
What about that code in AppDelegate mentioned earlier?
Outline Style
Add Entity
Add Attribute
Editor Style

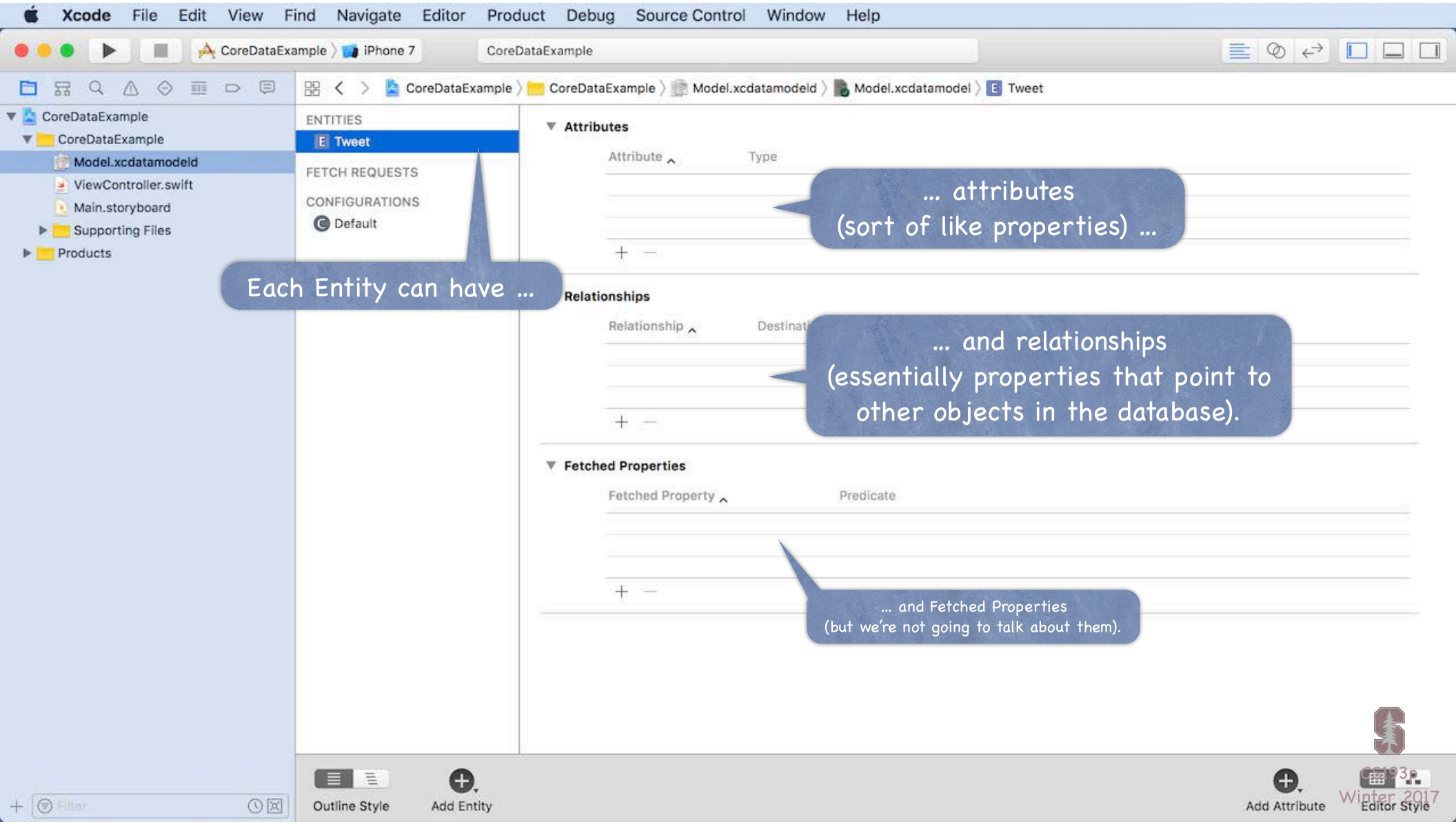

Xcode File Edit View Find Navigate Editor Product Debug Source Control Window Help
Here it is!
But how do you get this if you didn't click Use Core Data?
CoreDataExample
CoreDataExample
Model.xcdatamodeld
ViewController.swift
Main.storyboard
Supporting Files
AppDelegate.swift
Assets.xcassets
LaunchScreen.storyboard
Info.plist
Products
Supporting Files 〉 AppDelegate.swift 〉 persistentContainer
// MARK: - Core Data stack
lazy var persistentContainer: NSPersistentContainer = {
/*
The persistent container for the application. This implementation
creates and returns a container, having loaded the store for the
application to it. This property is optional since there are legitimate
error conditions that could cause the creation of the store to fail.
*/
let container = NSPersistentContainer(name: "Model")
container.loadPersistentStores(completionHandler: { (storeDescription, error) in
if let error = error as NSError? {
// Replace this implementation with code to handle the error appropriately.
fatalError() causes the
You should not use thi
be useful during devel
pical reasons for an error here include:
The parent directory does not exist, cannot be created, or disallows writing.
data protection
*/
fatalError("Unresolved error \(error), \(error.userInfo)")
}
})
return container
}()
Create another Project.
Just so you can click Use Core Data.
Copy the code for this var
from that Project's AppDelegate ...
... and then change this string to match
the name of the Model you chose.
You'll probably also want to copy saveContext()
and change applicationWillTerminate to call self.saveContext().
CS193p
Winter 2017

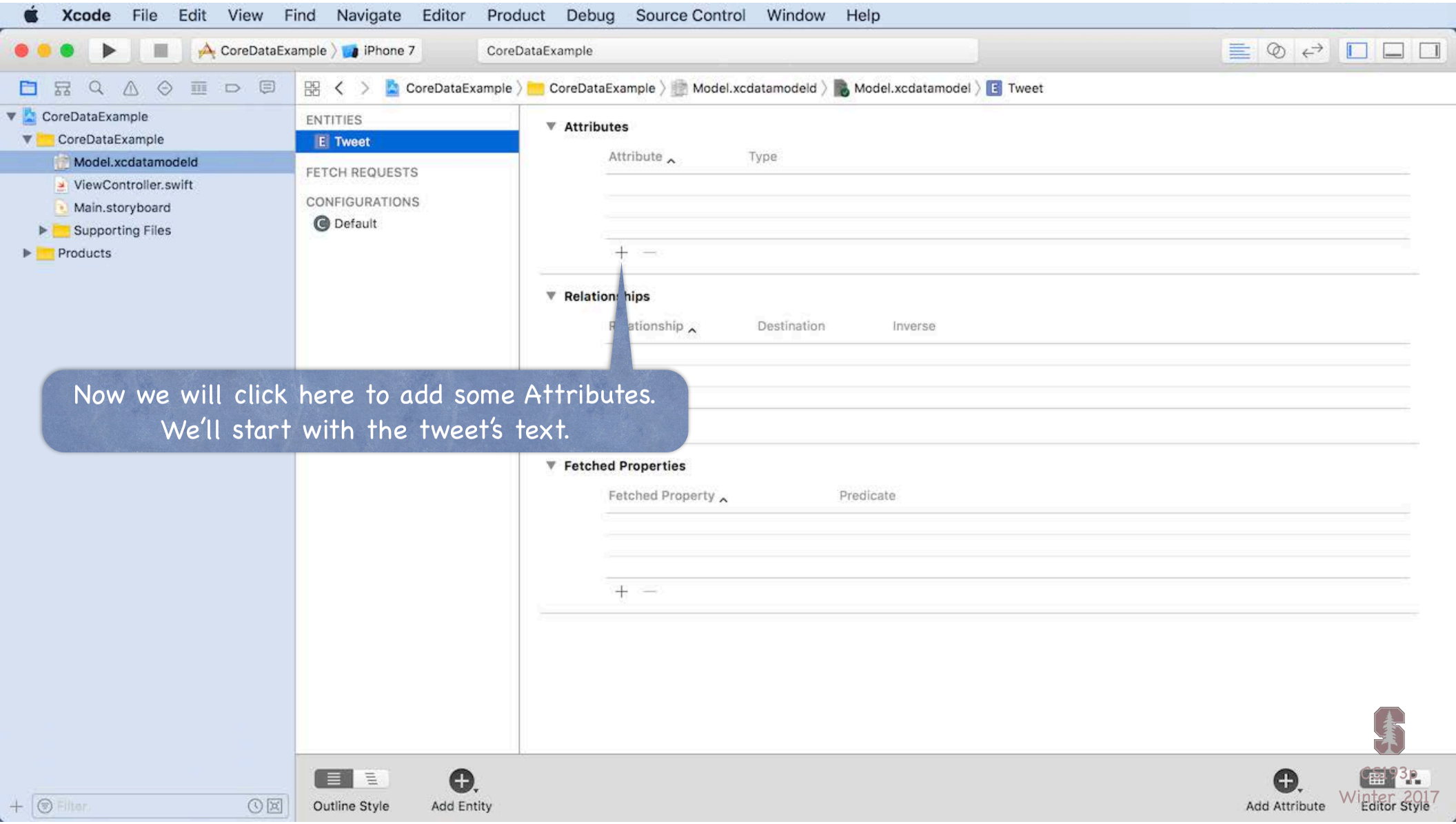

Xcode File Edit View Find Navigate Editor Product Debug Source Control Window Help
CoreDataExample 〉 iPhone 7
CoreDataExample
CoreDataExample 〉 CoreDataExample 〉 Model.xcdatamodeld 〉 Model.xcdatamodel 〉 Default
CoreDataExample
CoreDataExample
Model.xcdatamodeld
ViewController.swift
Main.storyboard
Supporting Files
Products
ENTITIES
FETCH REQUESTS
CONFIGURATIONS
Default
Entities
Entity
Abstract
Class
A Core Data database stores things in a way that looks very object-oriented to our code. It has ...
Entities (which are like a class)
Attributes (which are like a var)
Relationships (a var that points to other Entities)
This "storyboard" for databases lets us graphically describe these Entities, Attributes and Relationships.
Outline Style
Add Entity
Add Attribute
Editor Style
CS193p
Winter 2017

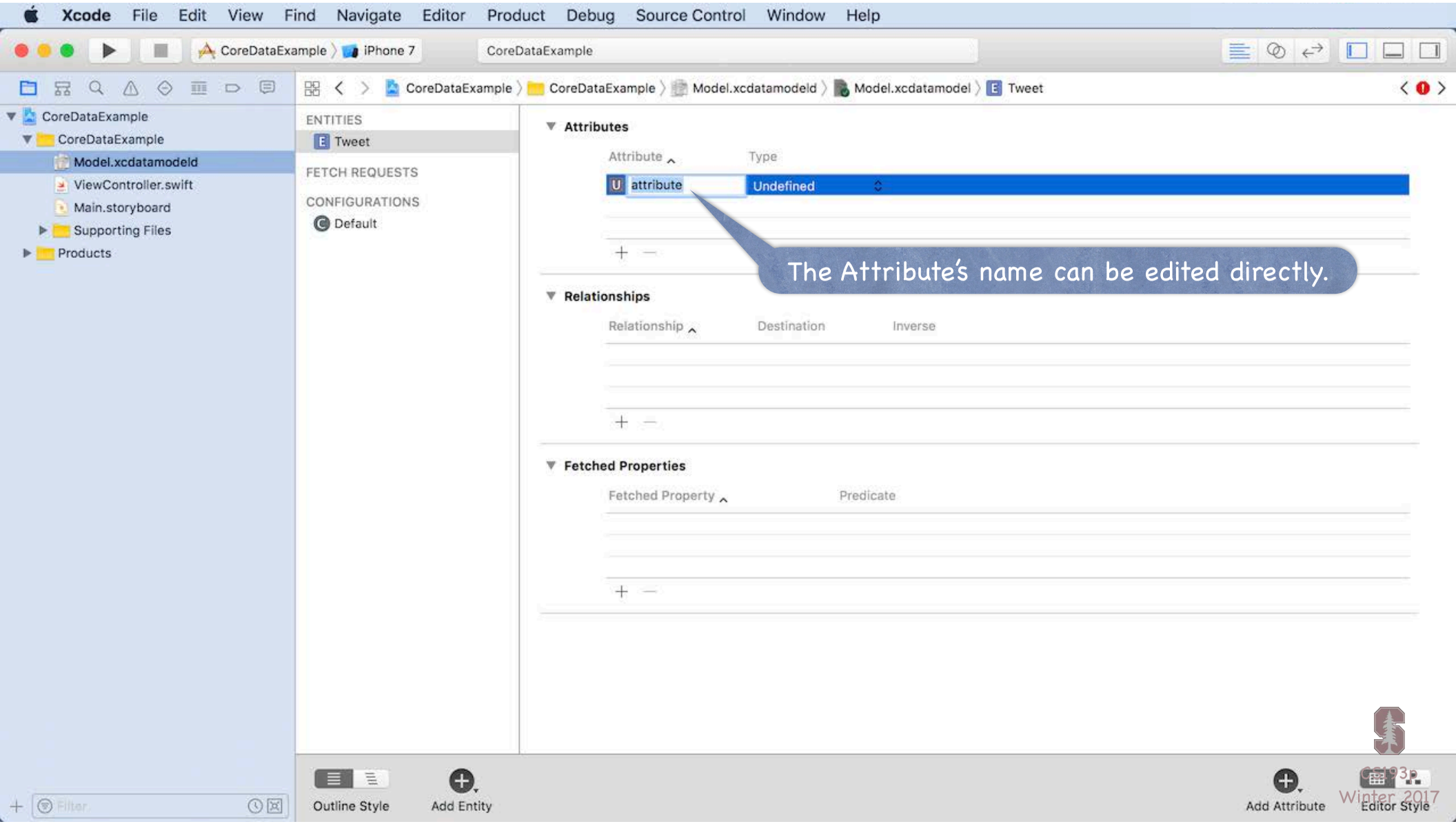

Xcode File Edit View Find Navigate Editor Product Debug Source Control Window Help
CoreDataExample 〉 iPhone 7
CoreDataExample
CoreDataExample 〉 CoreDataExample 〉 Model.xcdatamodeld 〉 Model.xcdatamodel 〉 Default
CoreDataExample
CoreDataExample
Model.xcdatamodeld
ViewController.swift
Main.storyboard
Supporting Files
Products
ENTITIES
FETCH REQUESTS
CONFIGURATIONS
Default
Entities
Entity
Abstract
Class
Let's start by adding an Entity
Add Entity
Add Fetch Request
Add Configuration
Unfortunately, we don't have time to talk about these two other options!
Outline Style
Add Entity
Add Attribute
Editor Style
CS193p
Winter 2017

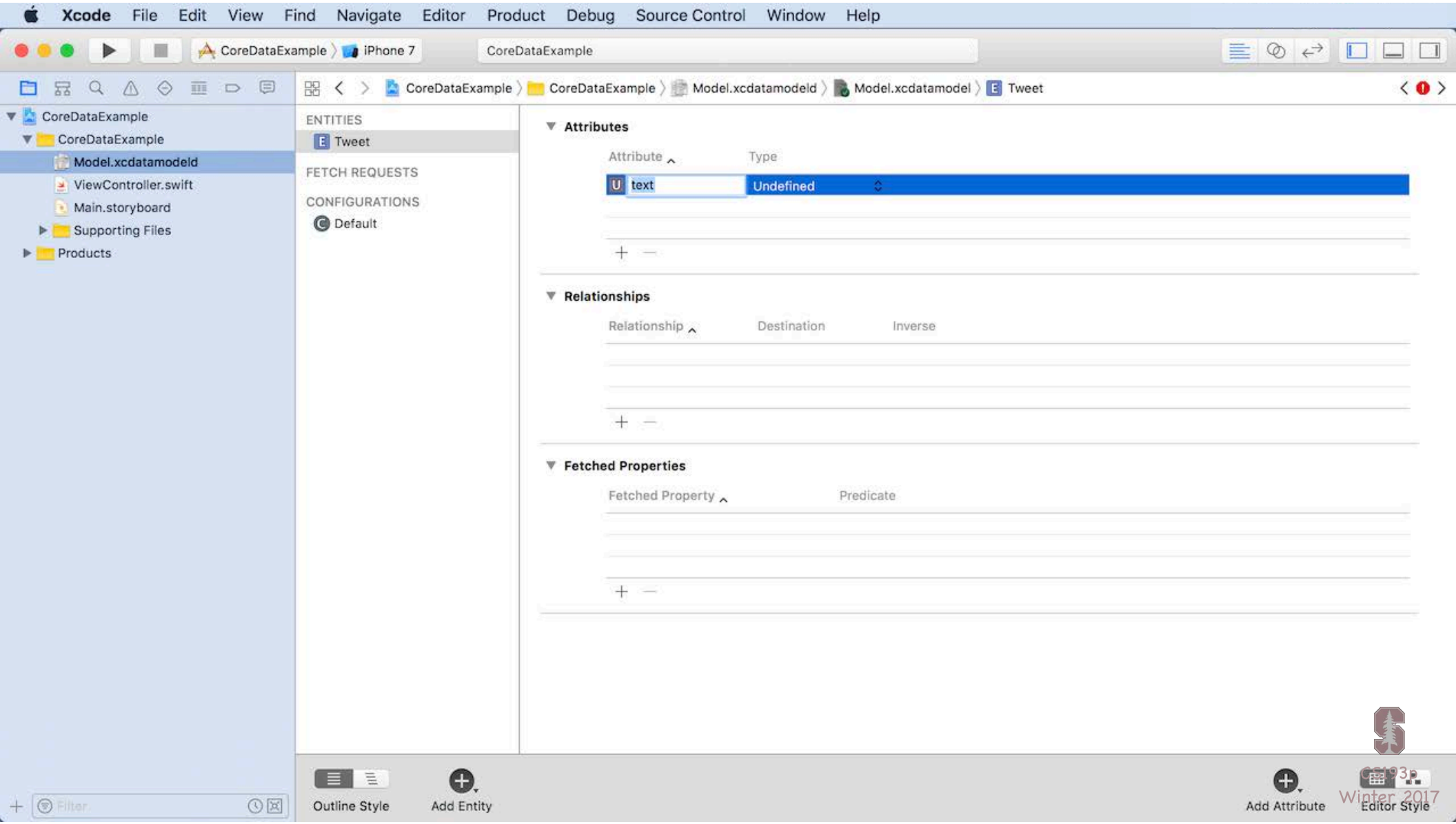


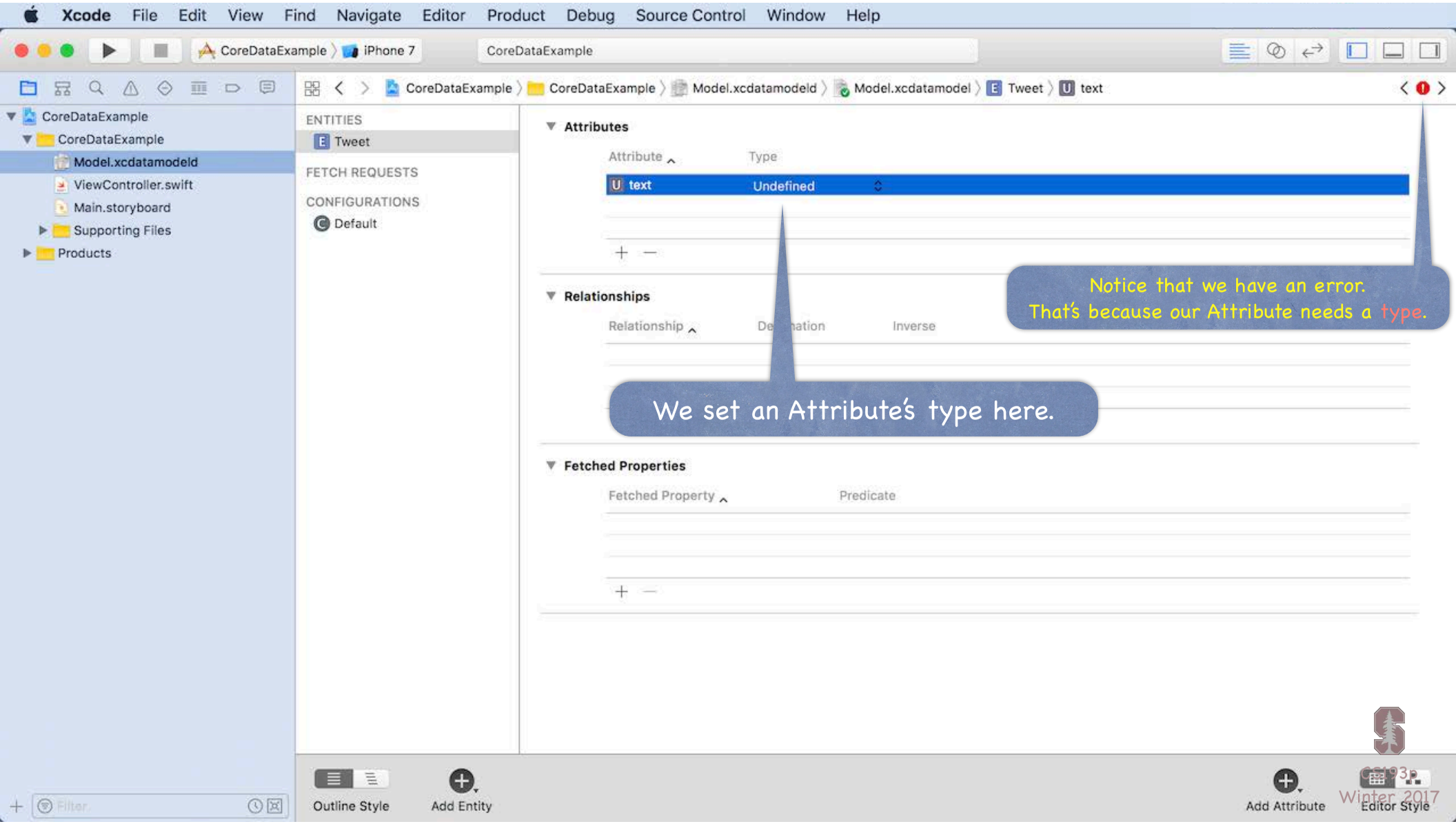

Xcode File Edit View Find Navigate Editor Product Debug Source Control Window Help
CoreDataExample 〉 iPhone 7
CoreDataExample
CoreDataExample 〉 CoreDataExample 〉 Model.xcdatamodeld 〉 Model.xcdatamodel 〉 Entity
CoreDataExample
CoreDataExample
Model.xcdatamodeld
ViewController.swift
Main.storyboard
Supporting Files
Products
ENTITIES
Tweet
FETCH REQUESTS
CONFIGURATIONS
Default
Let's rename it to be "Tweet".
Attributes
Attribute
Type
Relationships
Relationship
Destination
Inverse
Fetched Properties
Fetched Property
Predicate
Outline Style
Add Entity
Add Attribute
Editor Style
CS193p
Winter 2017

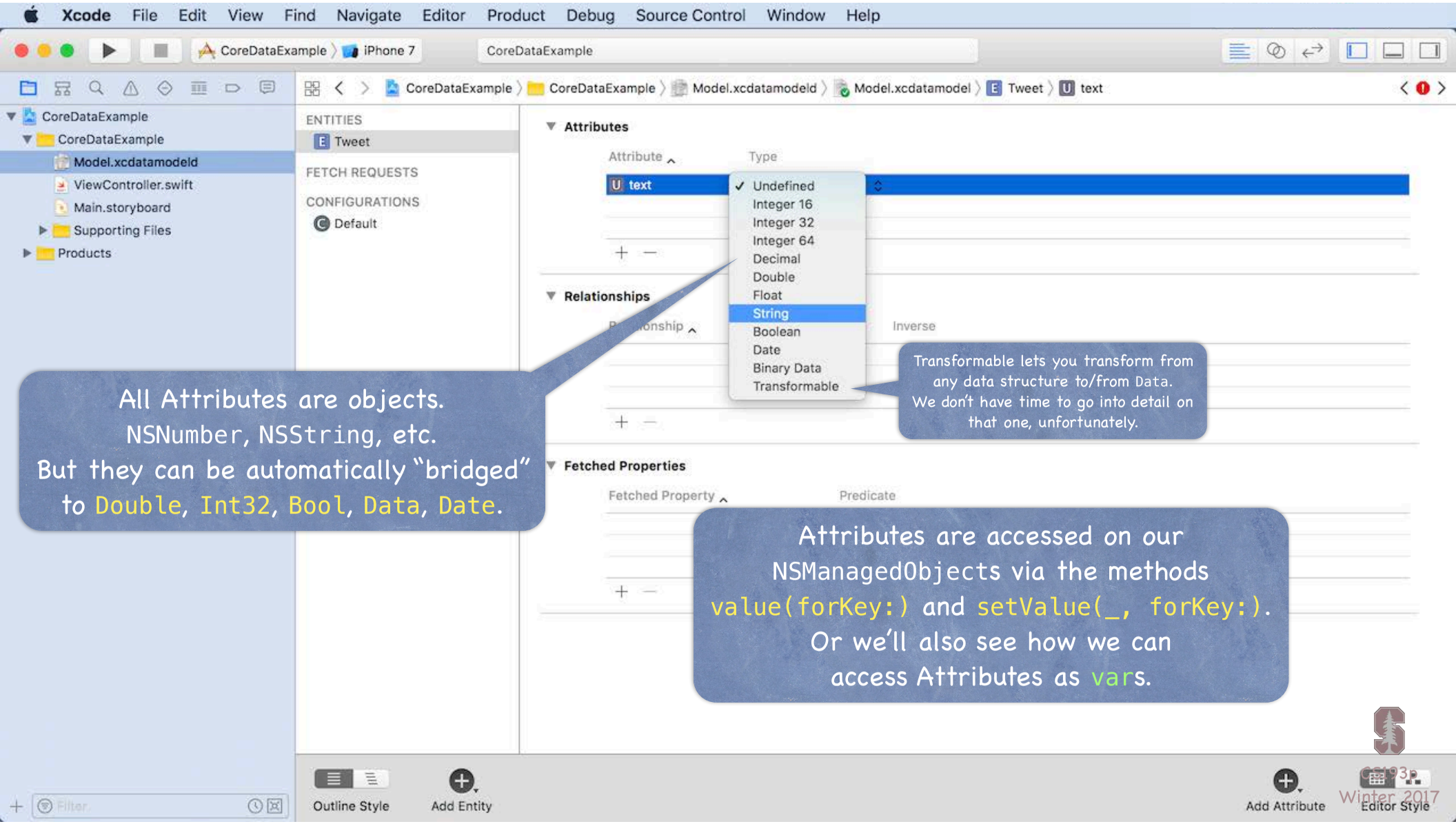

Xcode File Edit View Find Navigate Editor Product Debug Source Control Window Help
CoreDataExample iPhone 7
CoreDataExample
CoreDataExample CoreDataExample Model.xcdatamodeld Model.xcdatamodel Tweet
CoreDataExample
CoreDataExample
Model.xcdatamodeld
ViewController.swift
Main.storyboard
Supporting Files
Products
ENTITIES
Tweet
FETCH REQUESTS
CONFIGURATIONS
Default
Attributes
Attribute
Type
Relationships
Relationship
Fetched Properties
Fetched Property
Predicate
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).
Outline Style
Add Entity
Add Attribute
Editor Style

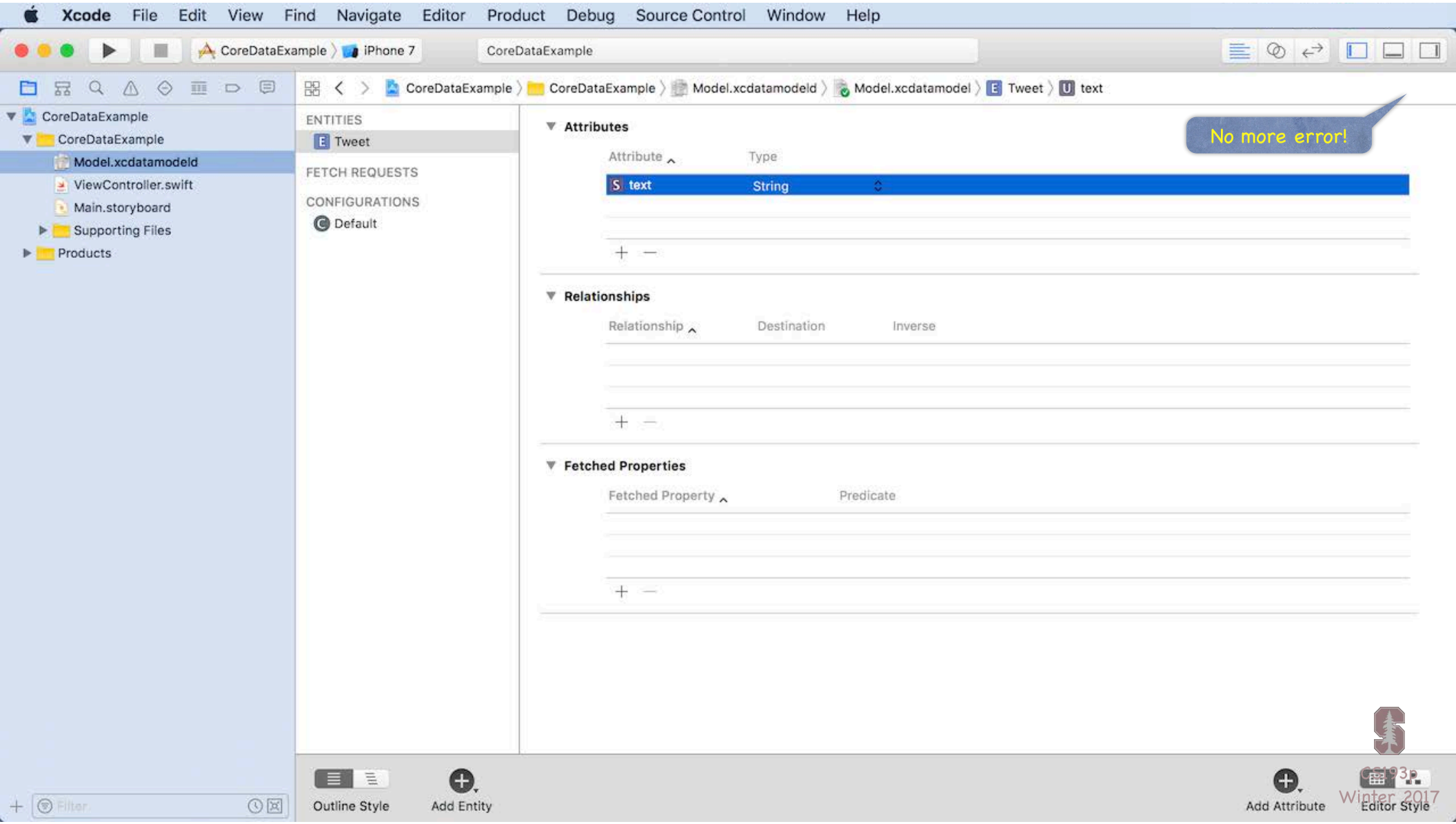

Xcode File Edit View Find Navigate Editor Product Debug Source Control Window Help
CoreDataExample 〉 iPhone 7
CoreDataExample
CoreDataExample 〉 CoreDataExample 〉 Model.xcdatamodeld 〉 Model.xcdatamodel 〉 Tweet
CoreDataExample
CoreDataExample
Model.xcdatamodeld
ViewController.swift
Main.storyboard
Supporting Files
Products
ENTITIES
Tweet
FETCH REQUESTS
CONFIGURATIONS
Default
Attributes
Attribute
Type
Relationships
Destination
Inverse
Fetched Properties
Fetched Property
Predicate
Now we will click here to add some Attributes.
We'll start with the tweet's text.
Outline Style
Add Entity
Add Attribute
Editor Style

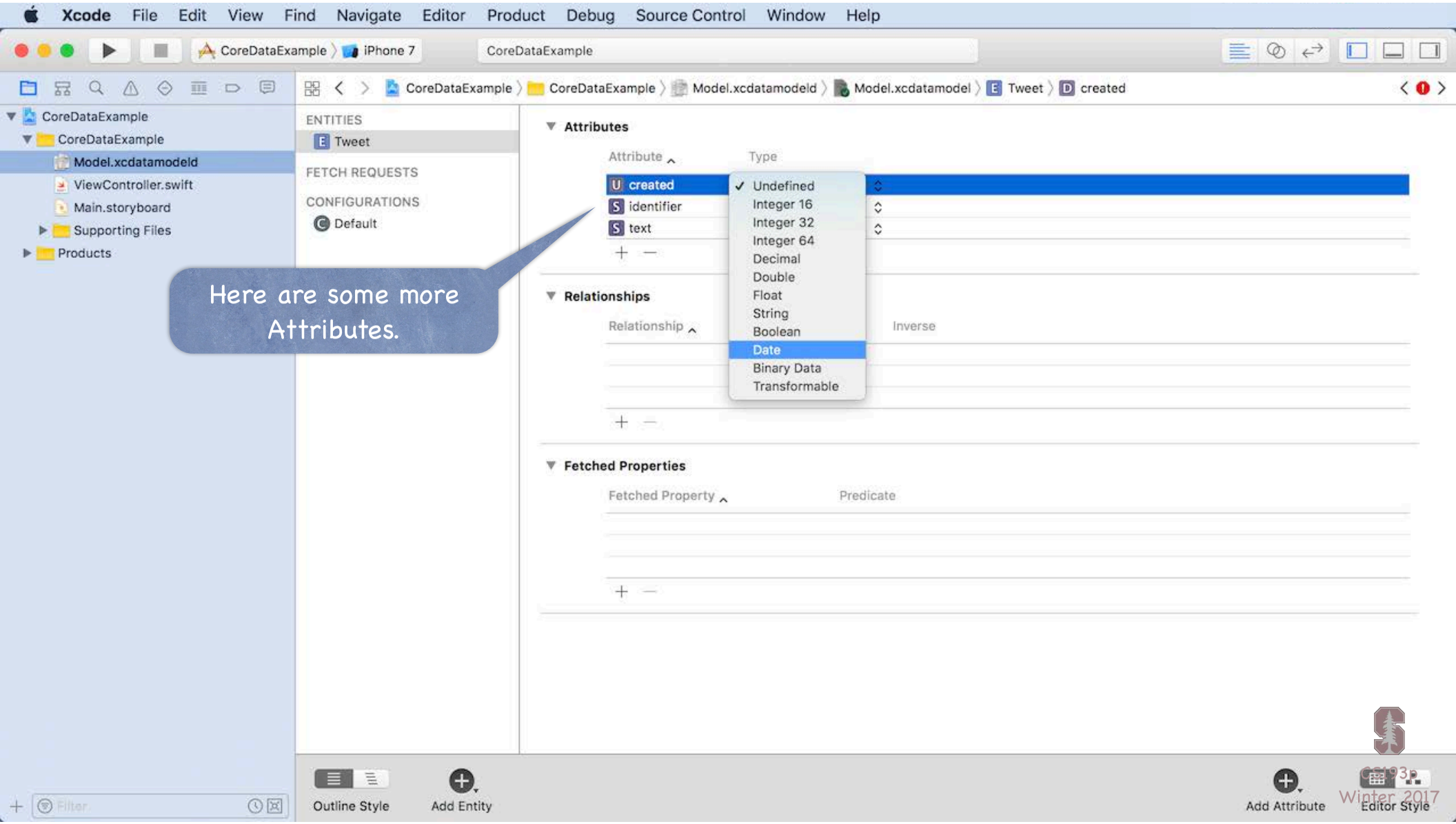


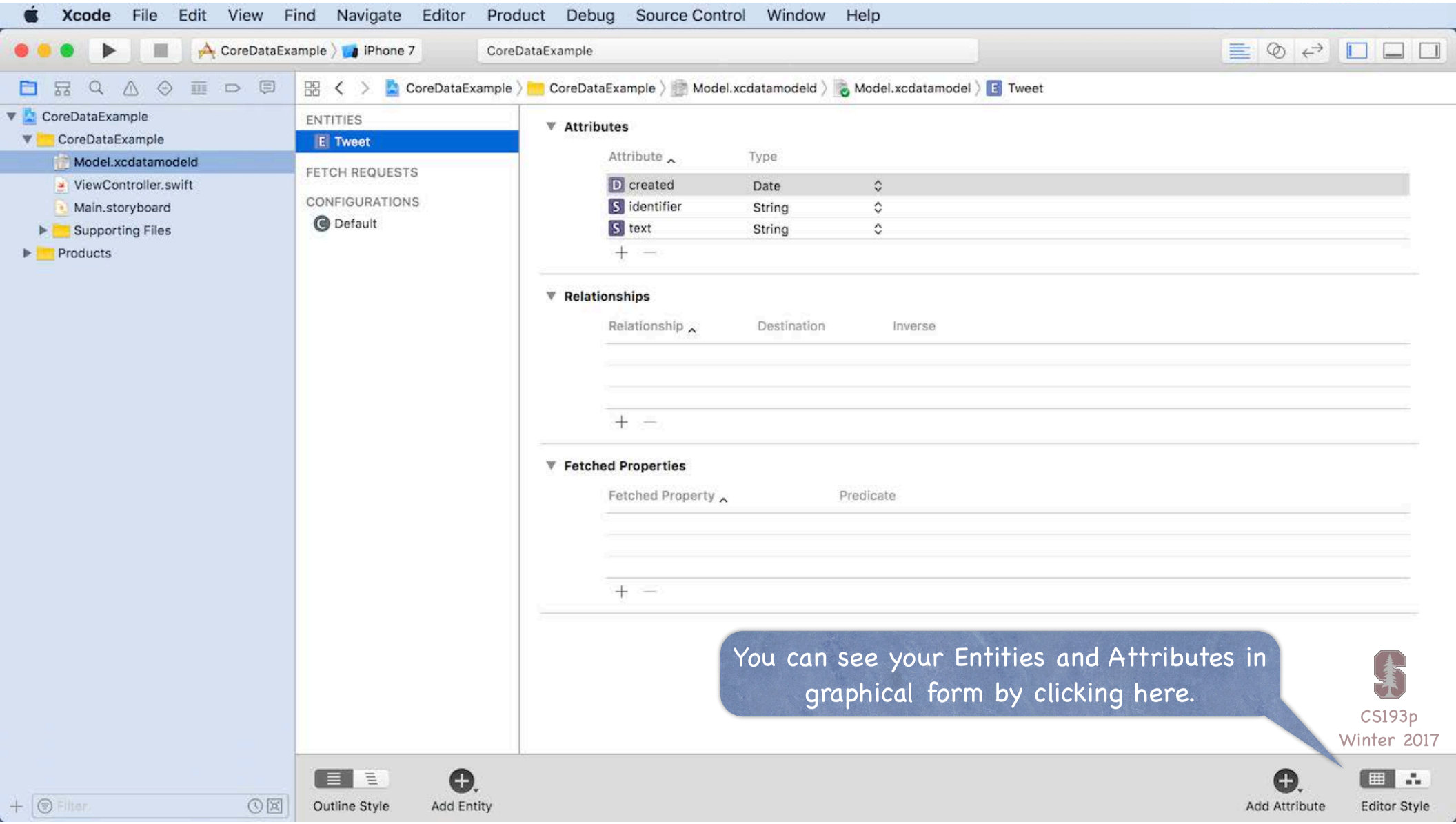

Xcode File Edit View Find Navigate Editor Product Debug Source Control Window Help
CoreDataExample 〉 iPhone 7
CoreDataExample
CoreDataExample 〉 CoreDataExample 〉 Model.xcdatamodeld 〉 Model.xcdatamodel 〉 Tweet
CoreDataExample
CoreDataExample
Model.xcdatamodeld
ViewController.swift
Main.storyboard
Supporting Files
Products
ENTITIES
Tweet
FETCH REQUESTS
CONFIGURATIONS
Default
Attributes
Attribute
Type
text
Undefined
Relationships
Relationship
Destination
Inverse
Fetched Properties
Fetched Property
Predicate
Outline Style
Add Entity
Add Attribute
Editor Style
CS193p
Winter 2017


Xcode File Edit View Find Navigate Editor Product Debug Source Control Window Help
CoreDataExample 〉 iPhone 7
CoreDataExample
CoreDataExample 〉 CoreDataExample 〉 Model.xcdatamodeld 〉 Model.xcdatamodel 〉 Tweet 〉 text
CoreDataExample
CoreDataExample
Model.xcdatamodeld
ViewController.swift
Main.storyboard
Supporting Files
Products
ENTITIES
Tweet
FETCH REQUESTS
CONFIGURATIONS
Default
Attributes
Attribute
Type
text
Undefined
Relationships
Relationship
Inverse
Fetched Properties
Fetched Property
Predicate
Notice that we have an error.
That's because our Attribute needs a type.
We set an Attribute's type here.
Outline Style
Add Entity
Add Attribute
Editor Style
CS193p
Winter 2017

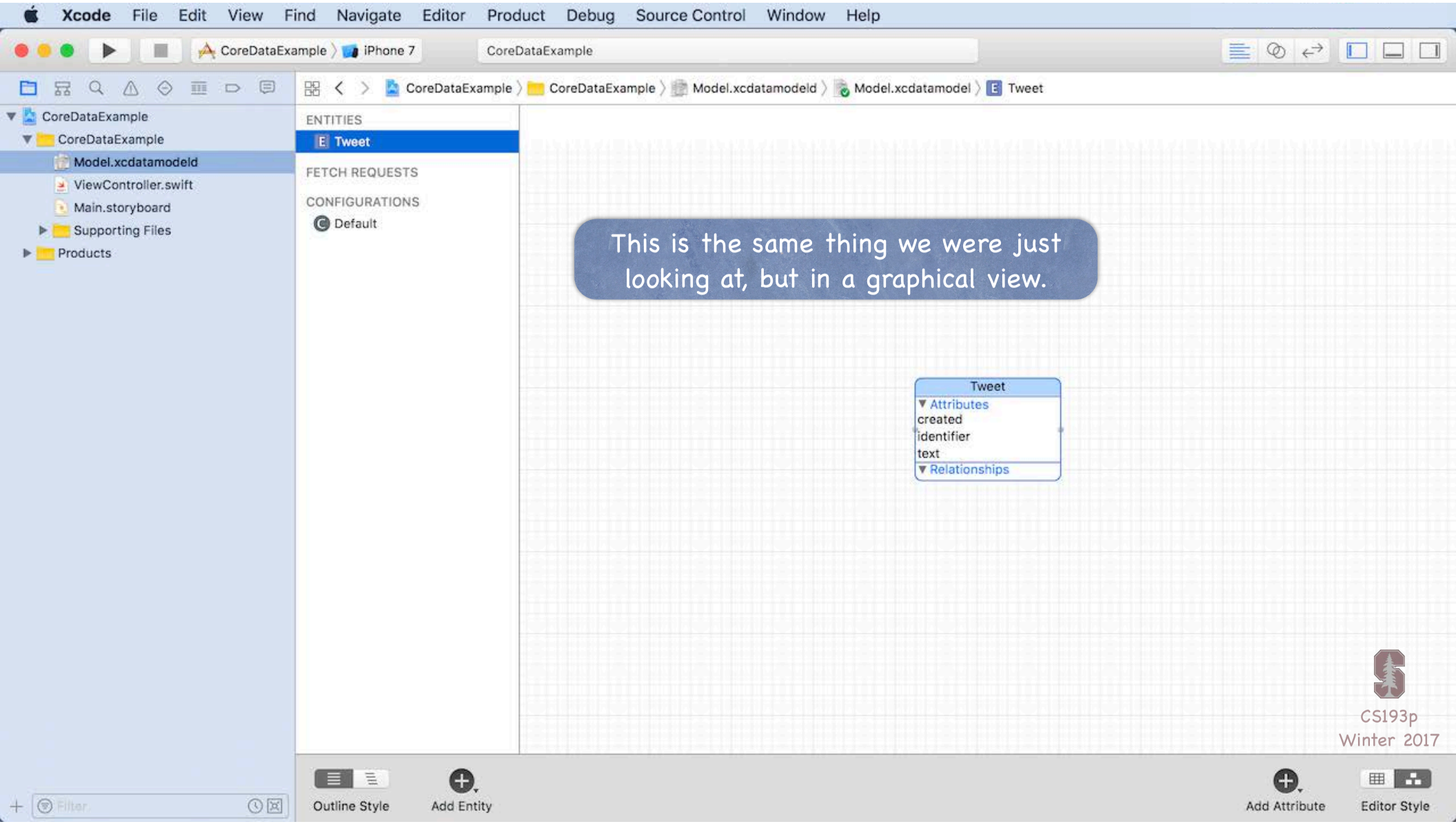

Xcode File Edit View Find Navigate Editor Product Debug Source Control Window Help
CoreDataExample
iPhone 7
CoreDataExample
CoreDataExample
CoreDataExample
Model.xcdatamodeld
Model.xcdatamodel
Tweet
text
CoreDataExample
CoreDataExample
Model.xcdatamodeld
ViewController.swift
Main.storyboard
Supporting Files
Products
ENTITIES
Tweet
FETCH REQUESTS
CONFIGURATIONS
Default
Attributes
Attribute
Type
text
Undefined
Integer 16
Integer 32
Integer 64
Decimal
Double
Float
String
Boolean
Date
Binary Data
Transformable
Relationships
Inverse
Fetched Properties
Fetched Property
Predicate
All Attributes are objects.
NSNumber, NSString, etc.
But they can be automatically "bridged"
to Double, Int32, Bool, Data, Date.
Transformable lets you transform from any data structure to/from Data. We don't have time to go into detail on that one, unfortunately.
Attributes are accessed on our NSManagedObjects via the methods value(forKey:) and setValue(_, forKey:). Or we'll also see how we can access Attributes as vars.
Outline Style
Add Entity
Add Attribute
Editor Style
CS193p
Winter 2017

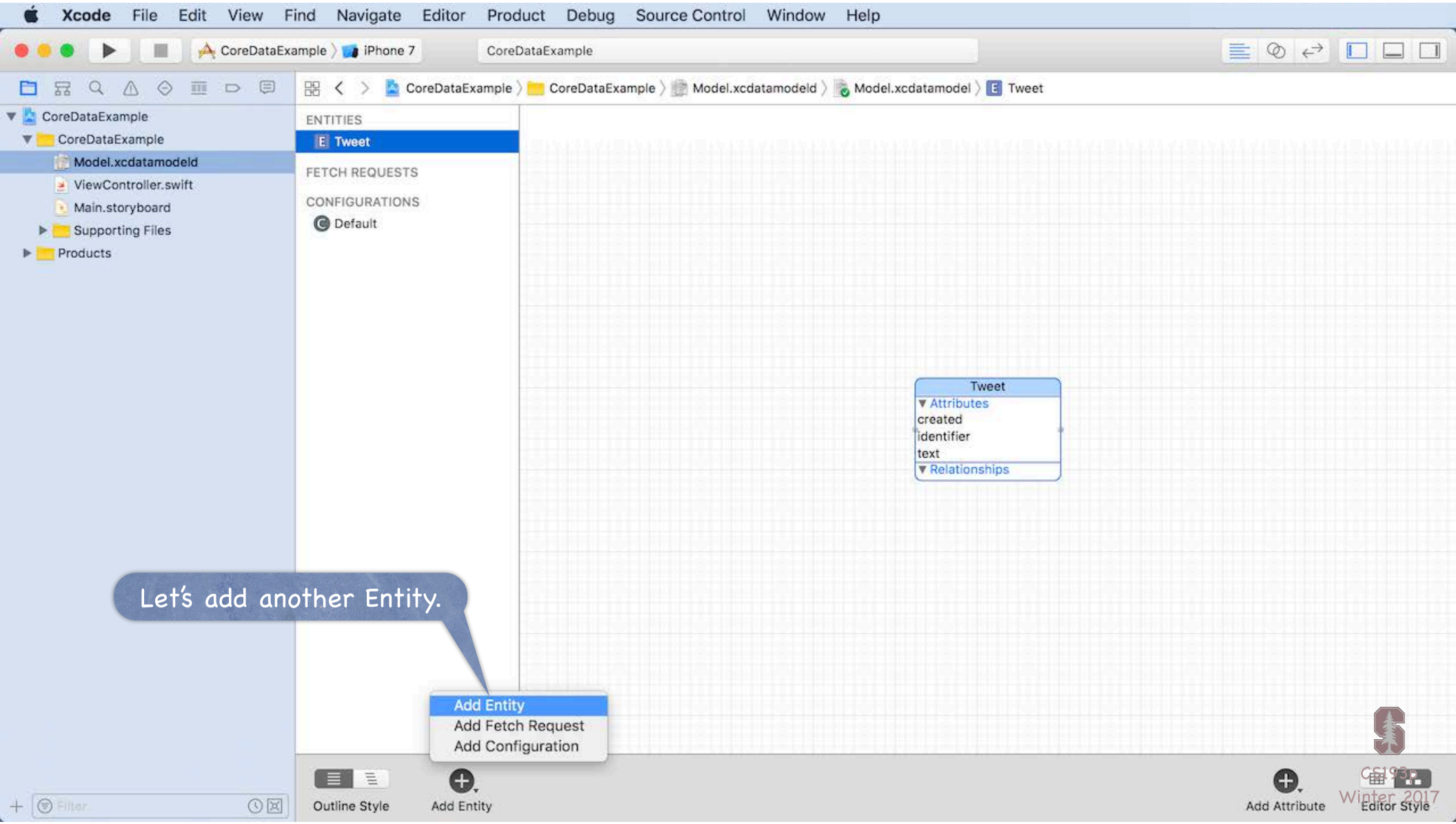

Xcode File Edit View Find Navigate Editor Product Debug Source Control Window Help
CoreDataExample 〉 iPhone 7
CoreDataExample
CoreDataExample 〉 CoreDataExample 〉 Model.xcdatamodeld 〉 Model.xcdatamodel 〉 Tweet 〉 text
CoreDataExample
CoreDataExample
Model.xcdatamodeld
ViewController.swift
Main.storyboard
Supporting Files
Products
ENTITIES
Tweet
FETCH REQUESTS
CONFIGURATIONS
Default
Attributes
Attribute
Type
text
String
No more error!
Relationships
Relationship
Destination
Inverse
Fetched Properties
Fetched Property
Predicate
Outline Style
Add Entity
Add Attribute
Editor Style
CS193p
Winter 2017

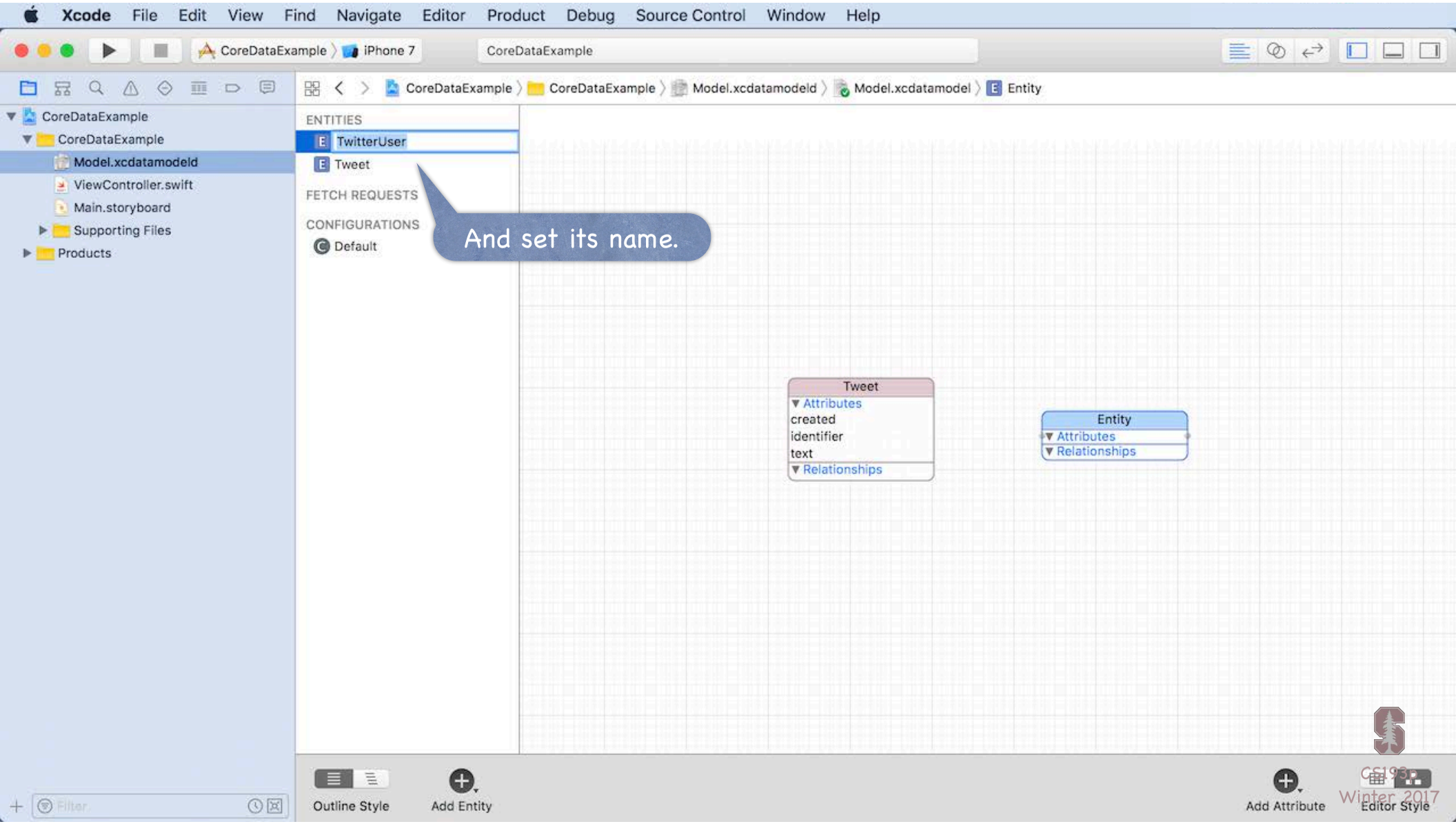

Xcode File Edit View Find Navigate Editor Product Debug Source Control Window Help
CoreDataExample 〉 iPhone 7
CoreDataExample
CoreDataExample 〉 CoreDataExample 〉 Model.xcdatamodeld 〉 Model.xcdatamodel 〉 Tweet 〉 created
CoreDataExample
CoreDataExample
Model.xcdatamodeld
ViewController.swift
Main.storyboard
Supporting Files
Products
ENTITIES
Tweet
FETCH REQUESTS
CONFIGURATIONS
Default
Here are some more Attributes.
Attributes
Attribute
Type
created
identifier
text
Undefined
Integer 16
Integer 32
Integer 64
Decimal
Double
Float
String
Boolean
Date
Binary Data
Transformable
Relationships
Relationship
Inverse
Fetched Properties
Fetched Property
Predicate
Outline Style
Add Entity
Add Attribute
Editor Style
CS193p
Winter 2017

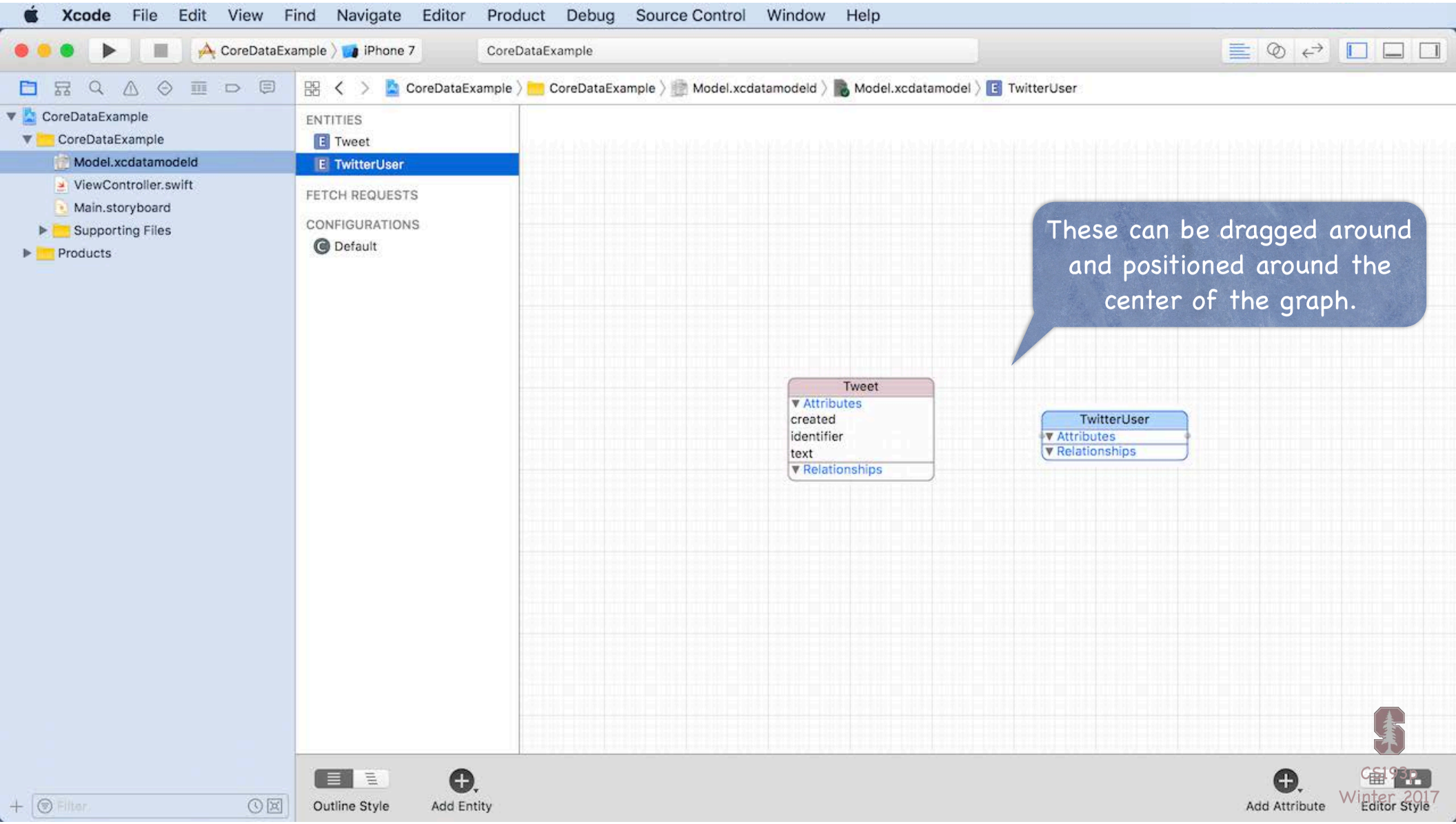


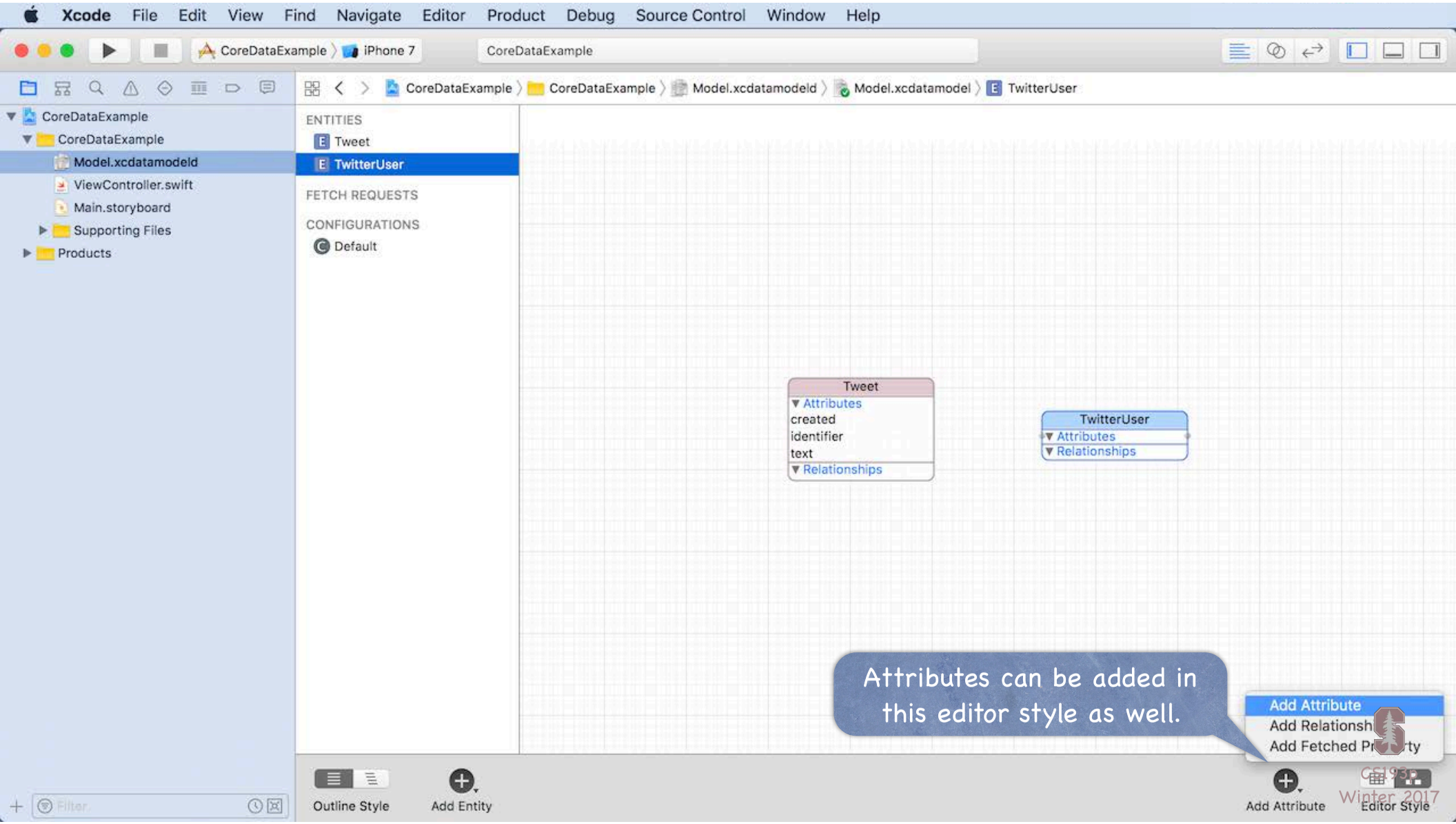
CS193p
Winter 2017

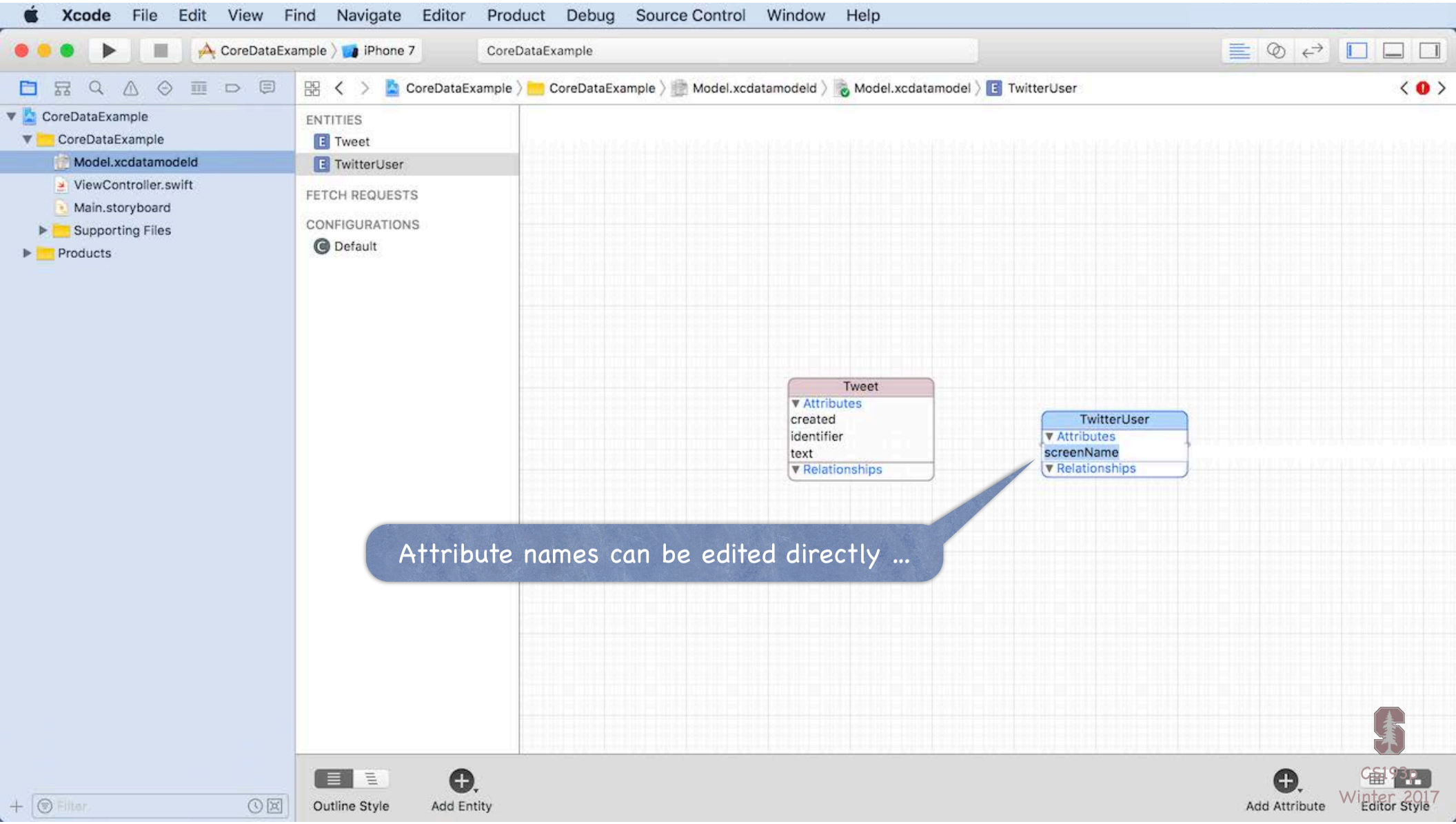

Xcode File Edit View Find Navigate Editor Product Debug Source Control Window Help
CoreDataExample 〉 iPhone 7
CoreDataExample
CoreDataExample 〉 CoreDataExample 〉 Model.xcdatamodeld 〉 Model.xcdatamodel 〉 Tweet
CoreDataExample
CoreDataExample
Model.xcdatamodeld
ViewController.swift
Main.storyboard
Supporting Files
Products
ENTITIES
Tweet
FETCH REQUESTS
CONFIGURATIONS
Default
This is the same thing we were just looking at, but in a graphical view.
Tweet
Attributes
created
identifier
text
Relationships
CS193p
Winter 2017
Outline Style
Add Entity
Add Attribute
Editor Style

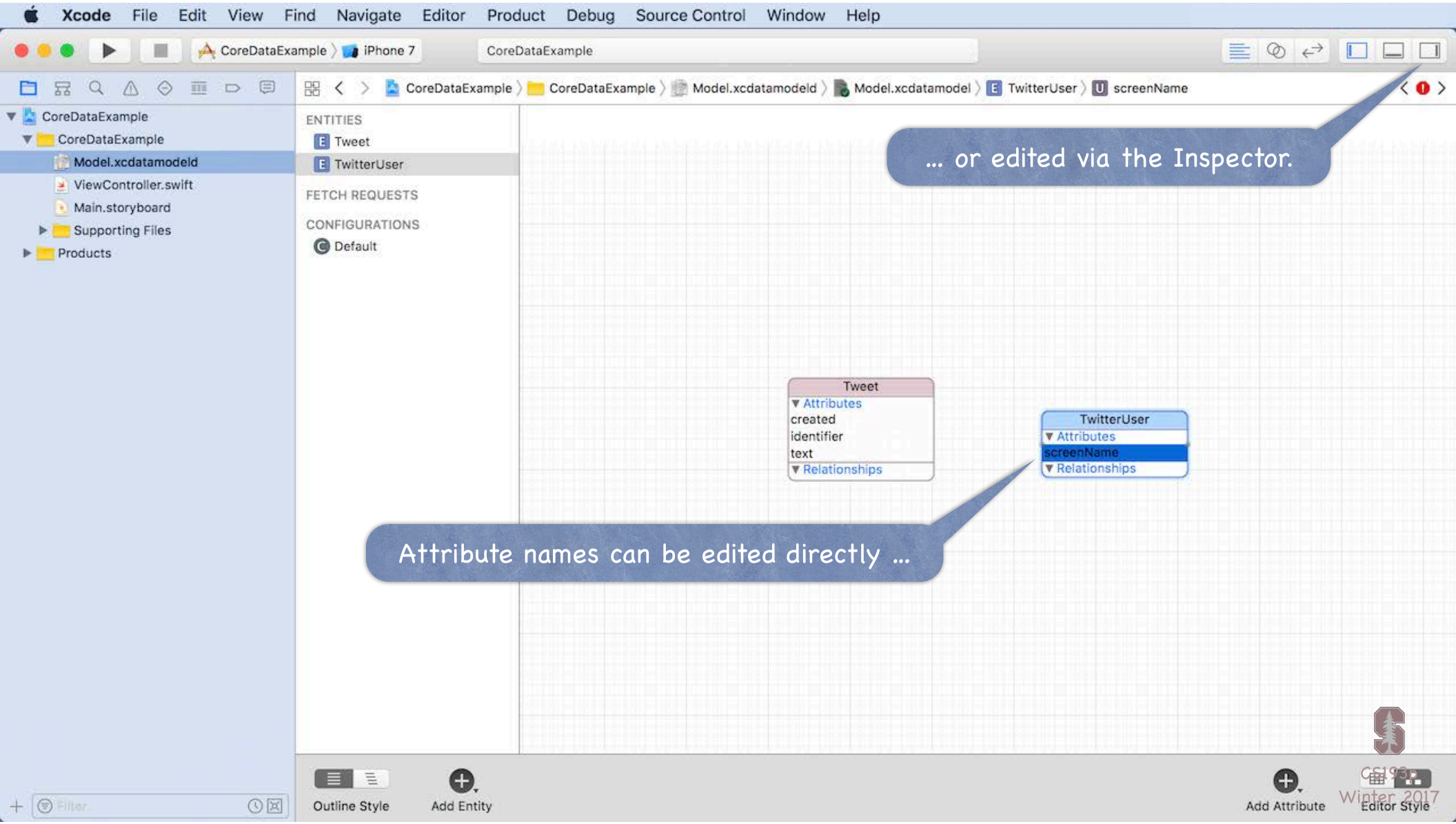

Xcode File Edit View Find Navigate Editor Product Debug Source Control Window Help
CoreDataExample 〉 iPhone 7
CoreDataExample
CoreDataExample 〉 CoreDataExample 〉 Model.xcdatamodeld 〉 Model.xcdatamodel 〉 Tweet
CoreDataExample
CoreDataExample
Model.xcdatamodeld
ViewController.swift
Main.storyboard
Supporting Files
Products
ENTITIES
Tweet
FETCH REQUESTS
CONFIGURATIONS
Default
Tweet
Attributes
created
identifier
text
Relationships
Let's add another Entity.
Add Entity
Add Fetch Request
Add Configuration
Outline Style
Add Entity
Add Attribute
Editor Style

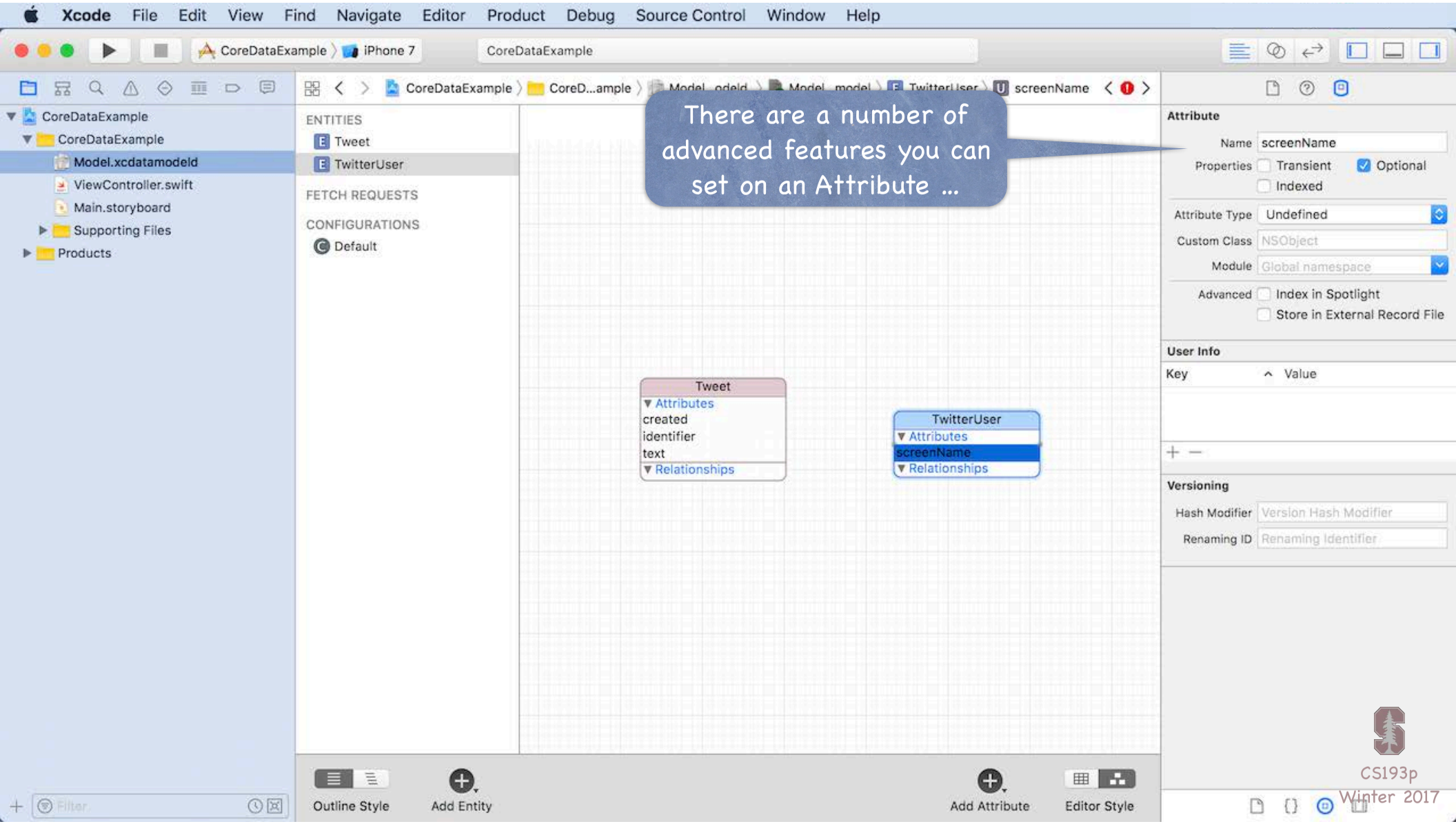

Xcode File Edit View Find Navigate Editor Product Debug Source Control Window Help
CoreDataExample 〉 iPhone 7
CoreDataExample
CoreDataExample 〉 CoreDataExample 〉 Model.xcdatamodeld 〉 Model.xcdatamodel 〉 Entity
CoreDataExample
CoreDataExample
Model.xcdatamodeld
ViewController.swift
Main.storyboard
Supporting Files
Products
ENTITIES
TwitterUser
Tweet
FETCH REQUESTS
CONFIGURATIONS
Default
And set its name.
Tweet
Attributes
created
identifier
text
Relationships
Entity
Attributes
Relationships
Outline Style
Add Entity
Add Attribute
Editor Style

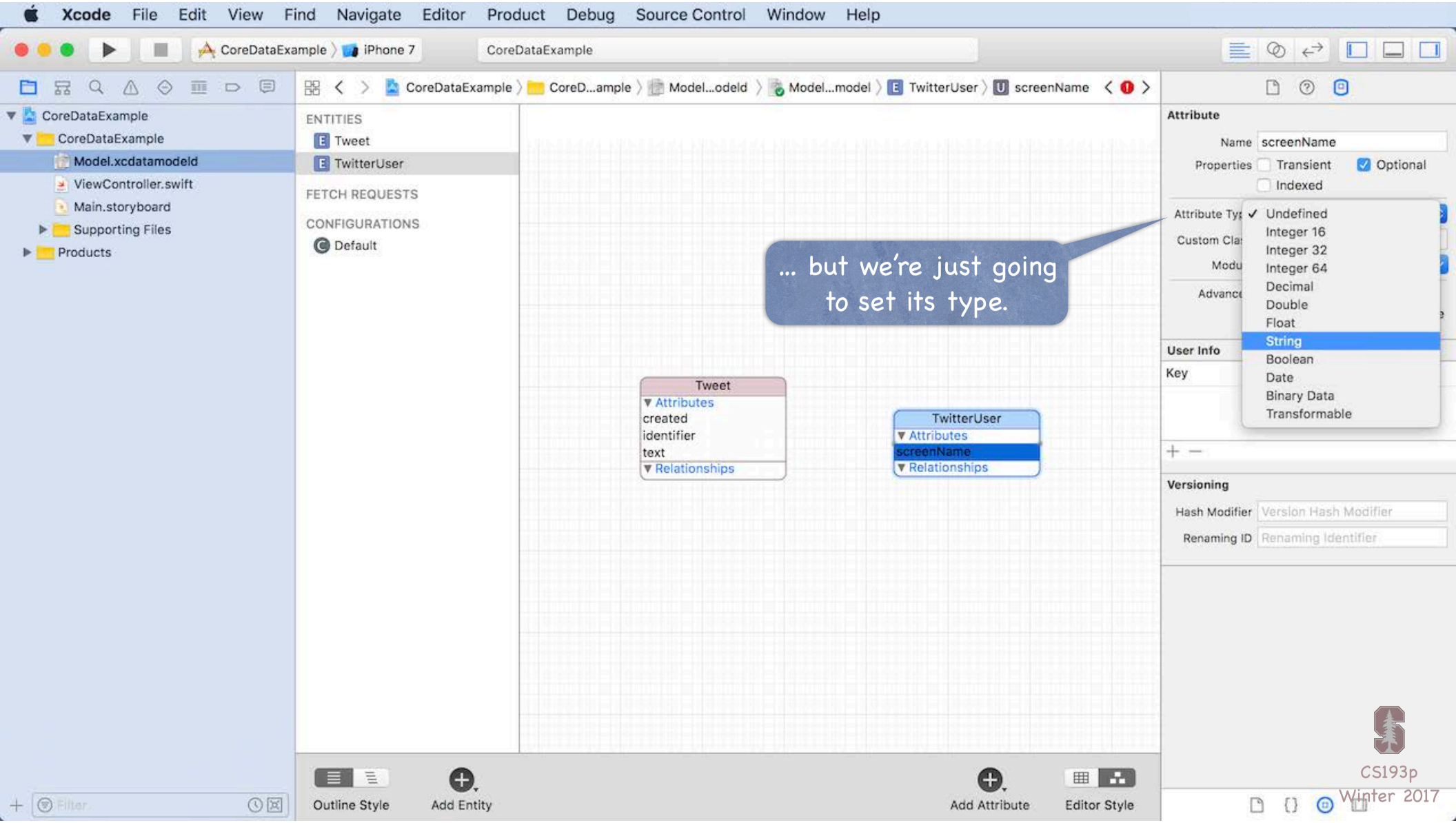


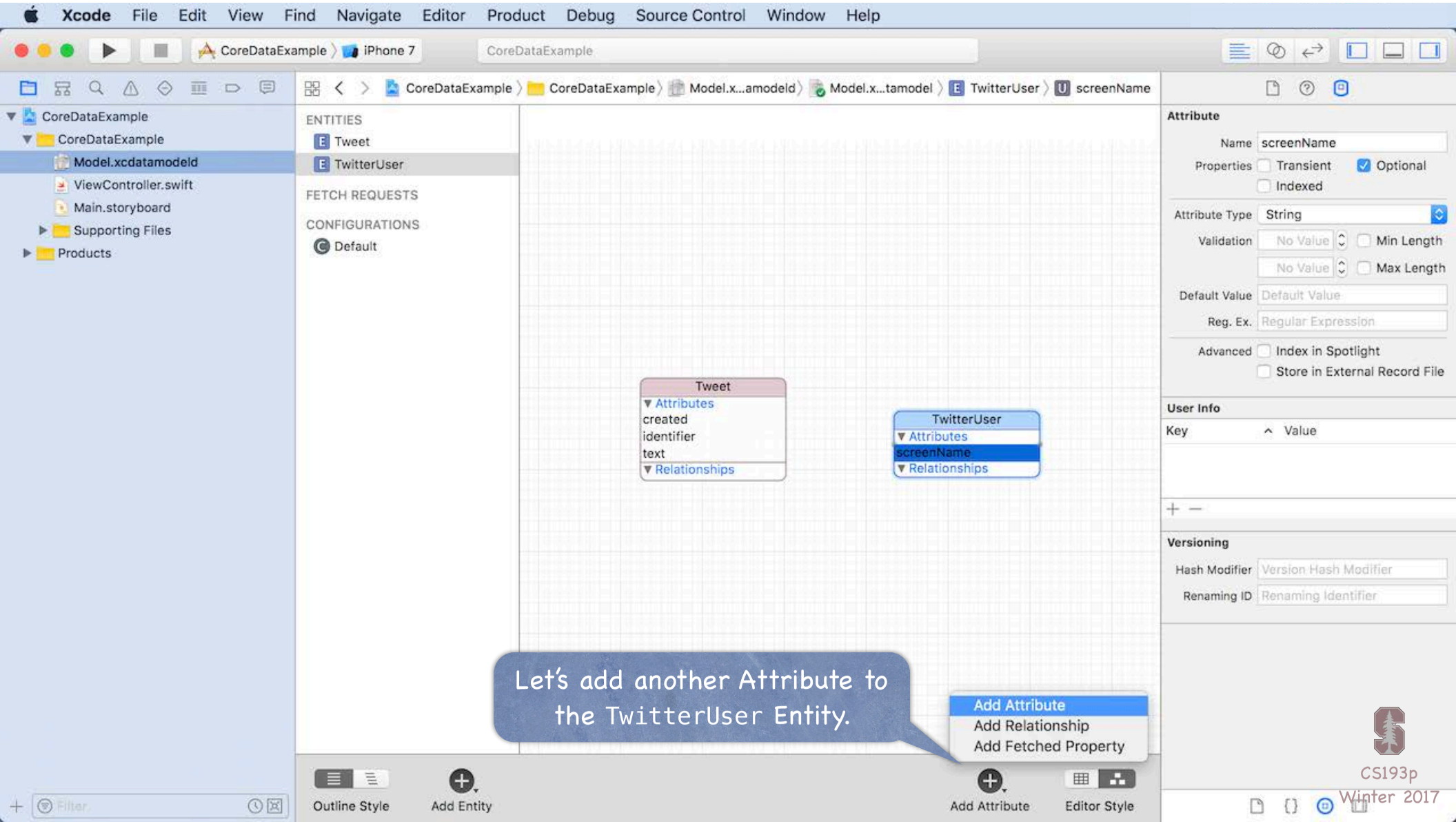

Xcode File Edit View Find Navigate Editor Product Debug Source Control Window Help
CoreDataExample 〉 iPhone 7
CoreDataExample
CoreDataExample 〉 CoreDataExample 〉 Model.xcdatamodeld 〉 Model.xcdatamodel 〉 TwitterUser
CoreDataExample
CoreDataExample
Model.xcdatamodeld
ViewController.swift
Main.storyboard
Supporting Files
Products
ENTITIES
Tweet
TwitterUser
FETCH REQUESTS
CONFIGURATIONS
Default
Tweet
Attributes
created
identifier
text
Relationships
TwitterUser
Attributes
Relationships
Attributes can be added in this editor style as well.
Add Attribute
Add Relationsh
Add Fetched Pr ty
Outline Style
Add Entity
Add Attribute
Editor Style
Winter 2017

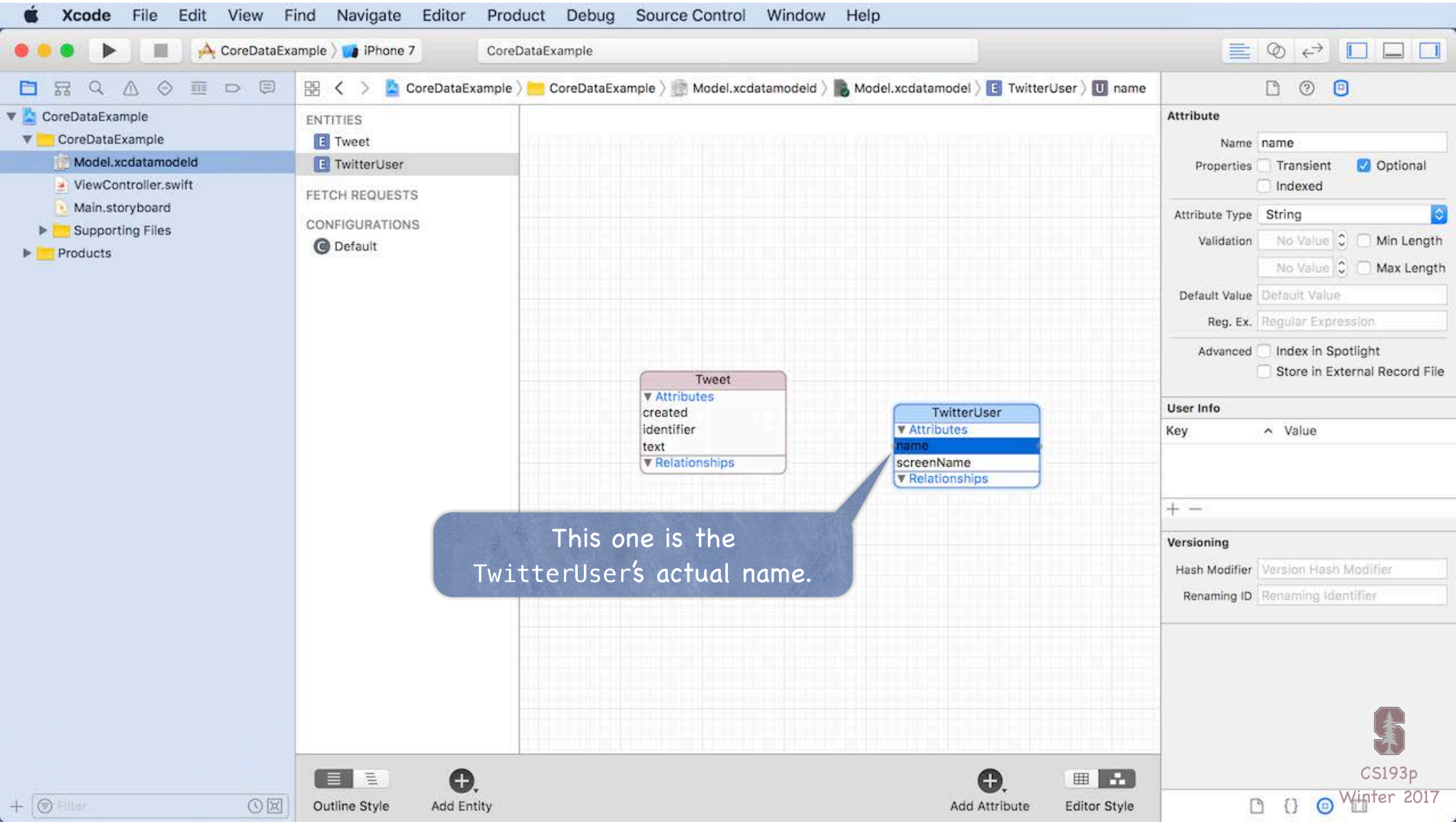

Xcode File Edit View Find Navigate Editor Product Debug Source Control Window Help
CoreDataExample 〉 iPhone 7
CoreDataExample
CoreDataExample 〉 CoreDataExample 〉 Model.xcdatamodeld 〉 Model.xcdatamodel 〉 TwitterUser
CoreDataExample
CoreDataExample
Model.xcdatamodeld
ViewController.swift
Main.storyboard
Supporting Files
Products
ENTITIES
Tweet
TwitterUser
FETCH REQUESTS
CONFIGURATIONS
Default
Tweet
Attributes
created
identifier
text
Relationships
TwitterUser
Attributes
screenName
Relationships
Attribute names can be edited directly …
Outline Style
Add Entity
Add Attribute
Editor Style
CS193p
Winter 2017

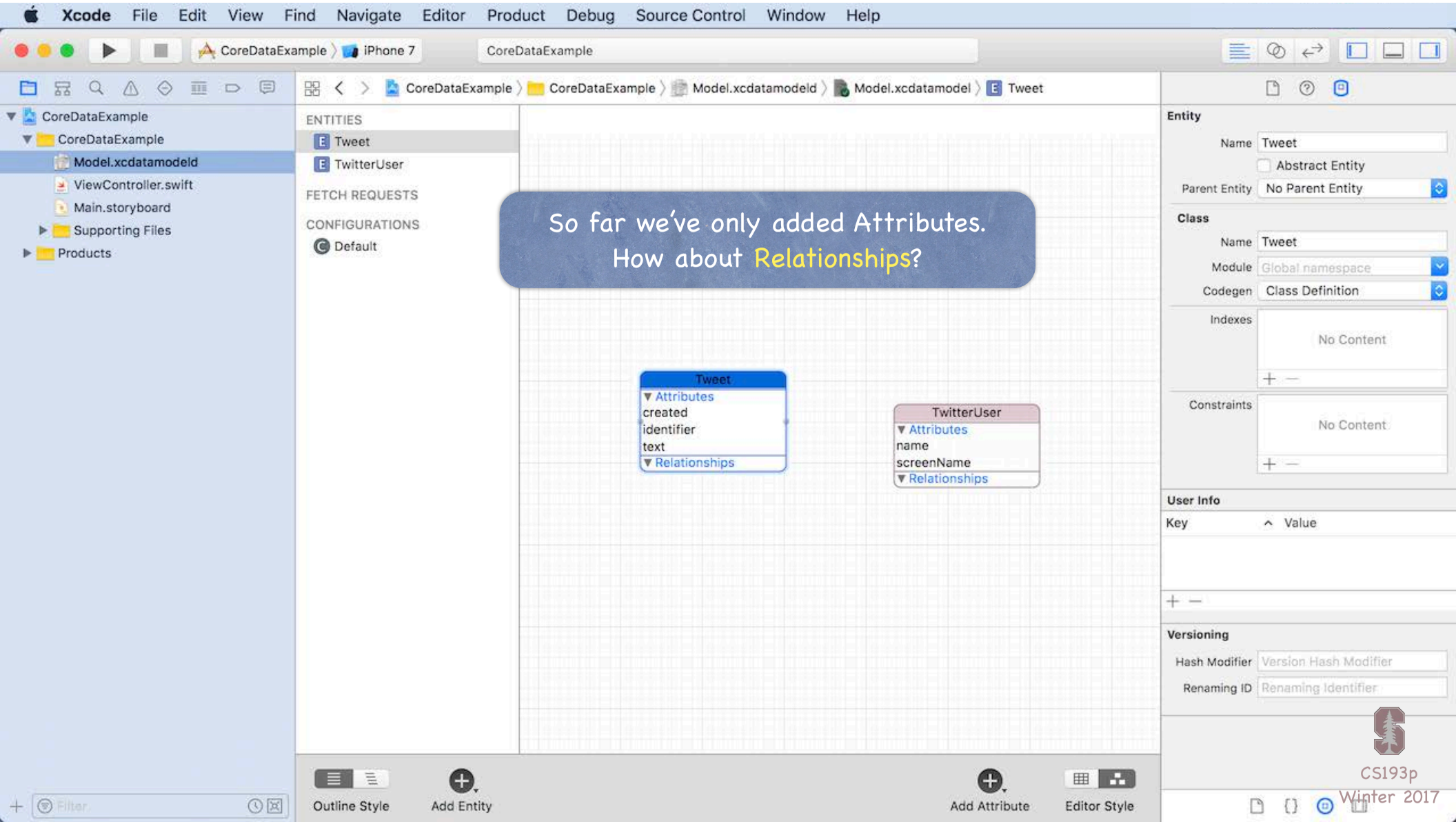


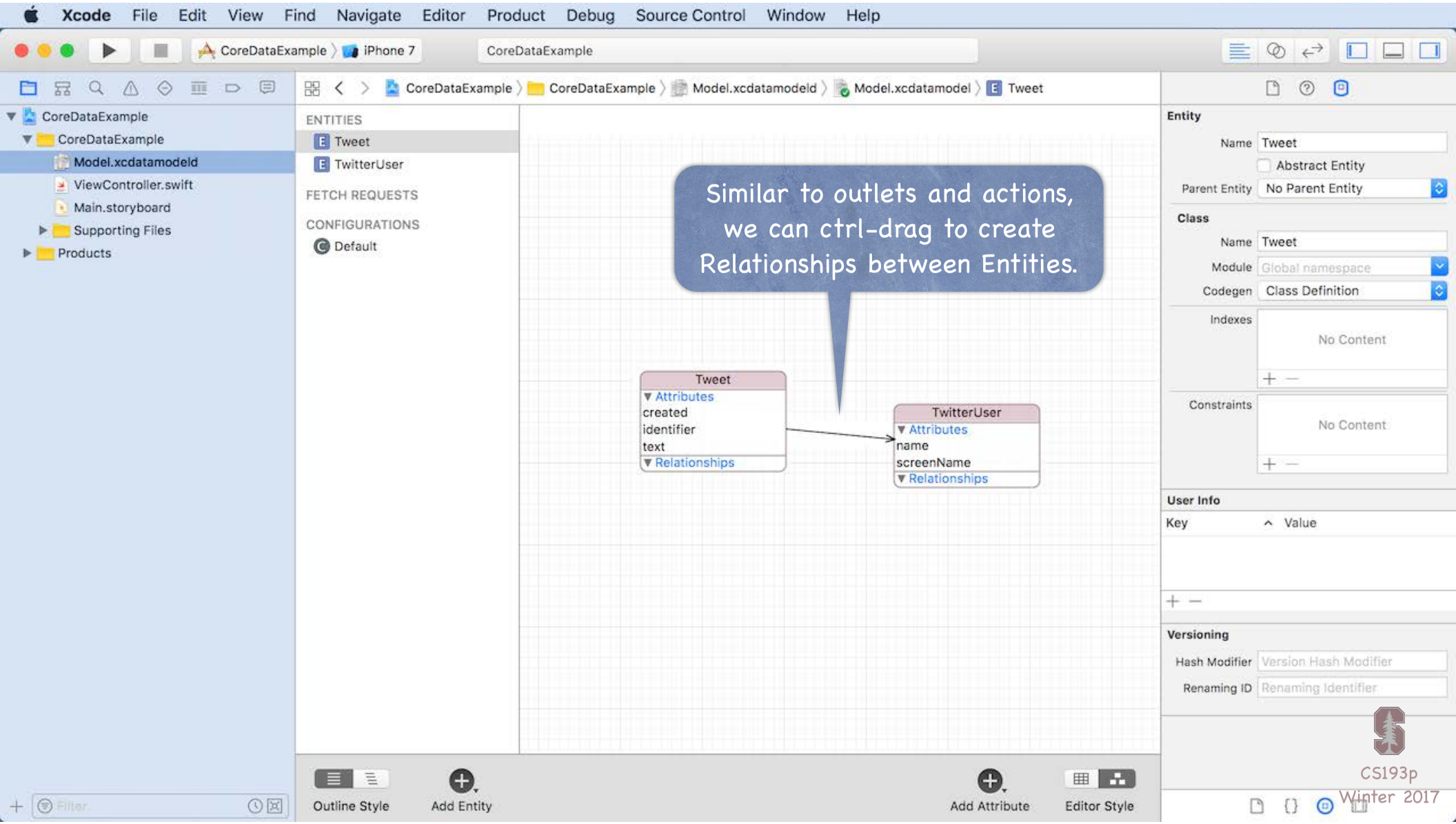

Xcode File Edit View Find Navigate Editor Product Debug Source Control Window Help
CoreDataExample 〉 iPhone 7
CoreDataExample
CoreDataExample 〉 CoreD...ample 〉 Model...odeld 〉 Model...model 〉 TwitterUser 〉 screenName
CoreDataExample
CoreDataExample
Model.xcdatamodeld
ViewController.swift
Main.storyboard
Supporting Files
Products
ENTITIES
Tweet
TwitterUser
FETCH REQUESTS
CONFIGURATIONS
Default
There are a number of advanced features you can set on an Attribute …
Tweet
Attributes
created
identifier
text
Relationships
TwitterUser
Attributes
screenName
Relationships
Attribute
Name screenName
Properties Transient Optional
Indexed
Attribute Type Undefined
Custom Class NSObject
Module Global namespace
Advanced Index in Spotlight
Store in External Record File
User Info
Key Value
Versioning
Hash Modifier Version Hash Modifier
Renaming ID Renaming Identifier
Outline Style
Add Entity
Add Attribute
Editor Style
CS193p
Winter 2017

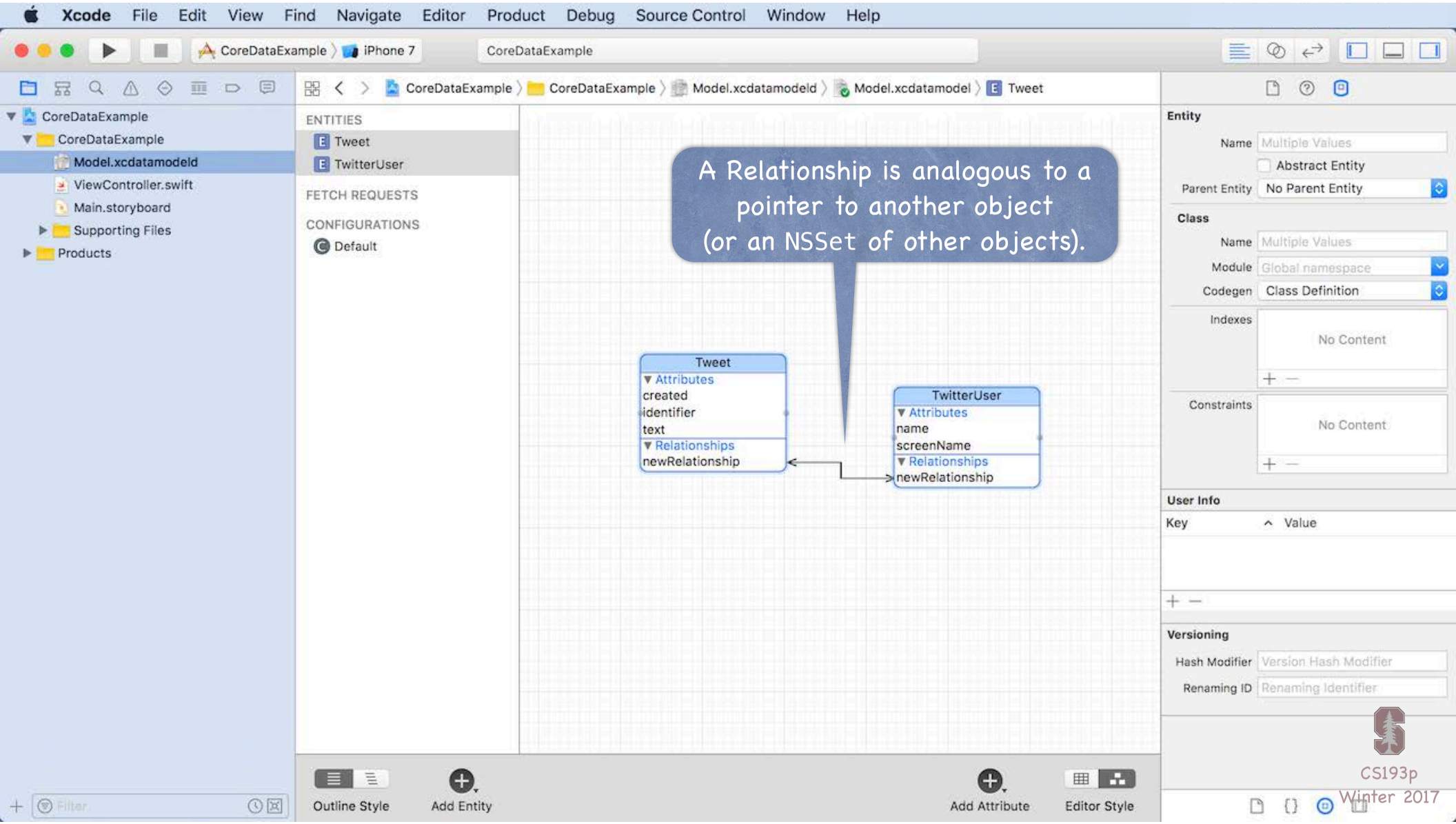

Xcode File Edit View Find Navigate Editor Product Debug Source Control Window Help
CoreDataExample
CoreDataExample
CoreDataExample
Model.xcdatamodeld
ViewController.swift
Main.storyboard
Supporting Files
Products
ENTITIES
Tweet
TwitterUser
FETCH REQUESTS
CONFIGURATIONS
Default
... but we're just going to set its type.
Tweet
Attributes
created
identifier
text
Relationships
TwitterUser
Attributes
screenName
Relationships
Attribute
Name screenName
Properties Transient Optional Indexed
Undefined
Integer 16
Integer 32
Integer 64
Decimal
Double
Float
String
Boolean
Date
Binary Data
Transformable
User Info
Key
Versioning
Hash Modifier
Renaming ID
Outline Style
Add Entity
Add Attribute
Editor Style
CS193p
Winter 2017

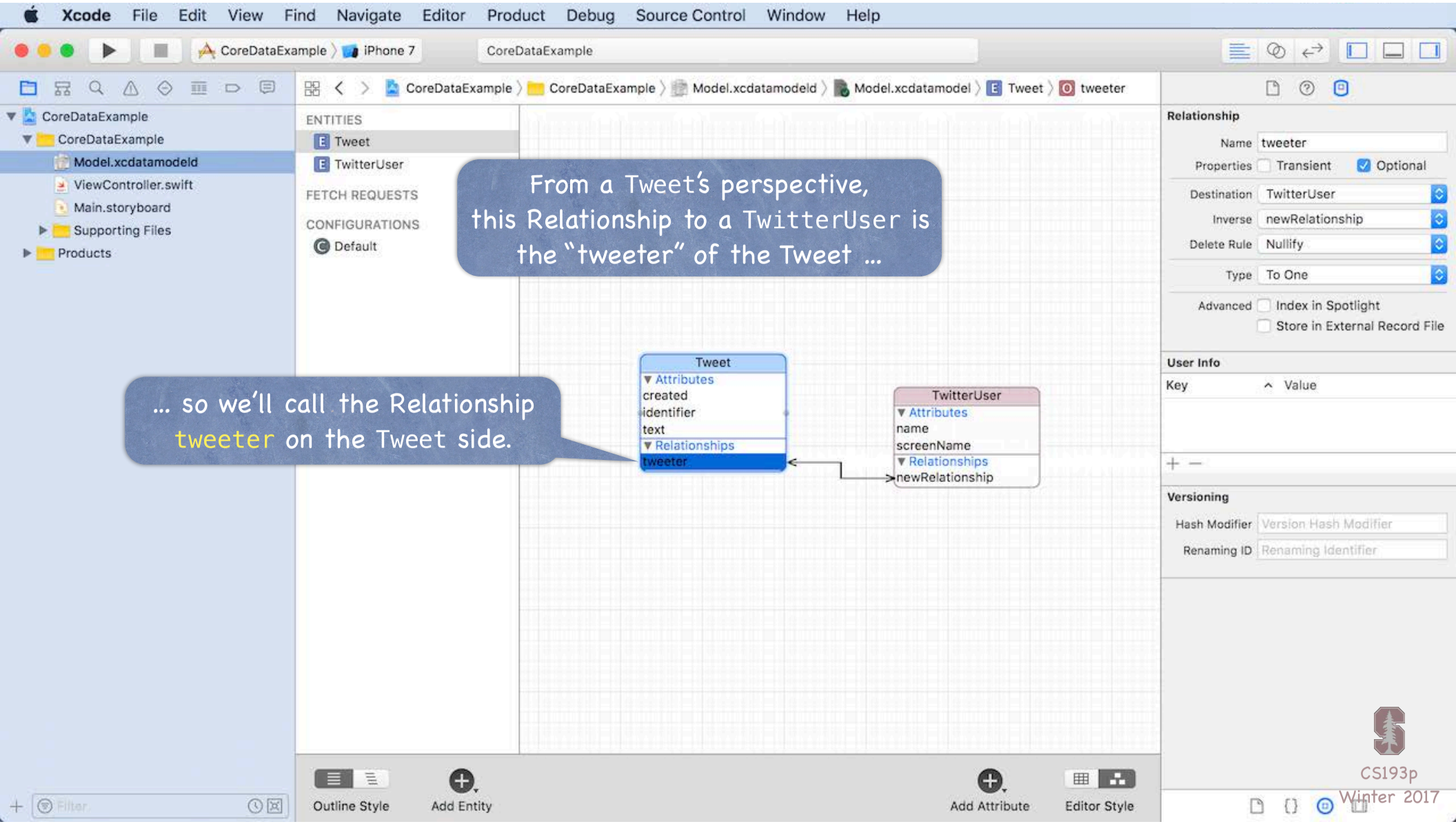

Xcode File Edit View Find Navigate Editor Product Debug Source Control Window Help
CoreDataExample 〉 iPhone 7
CoreDataExample
CoreDataExample 〉 CoreDataExample 〉 Model.x...amodeld 〉 Model.x...tamodel 〉 TwitterUser 〉 screenName
CoreDataExample
CoreDataExample
Model.xcdatamodeld
ViewController.swift
Main.storyboard
Supporting Files
Products
ENTITIES
Tweet
TwitterUser
FETCH REQUESTS
CONFIGURATIONS
Default
Tweet
Attributes
created
identifier
text
Relationships
TwitterUser
Attributes
screenName
Relationships
Attribute
Name screenName
Properties Transient Optional
Indexed
Attribute Type String
Validation No Value Min Length
No Value Max Length
Default Value Default Value
Reg. Ex. Regular Expression
Advanced Index in Spotlight
Store in External Record File
User Info
Key Value
Versioning
Hash Modifier Version Hash Modifier
Renaming ID Renaming Identifier
Let's add another Attribute to the TwitterUser Entity.
Add Attribute
Add Relationship
Add Fetched Property
Outline Style
Add Entity
Add Attribute
Editor Style
CS193p
Winter 2017

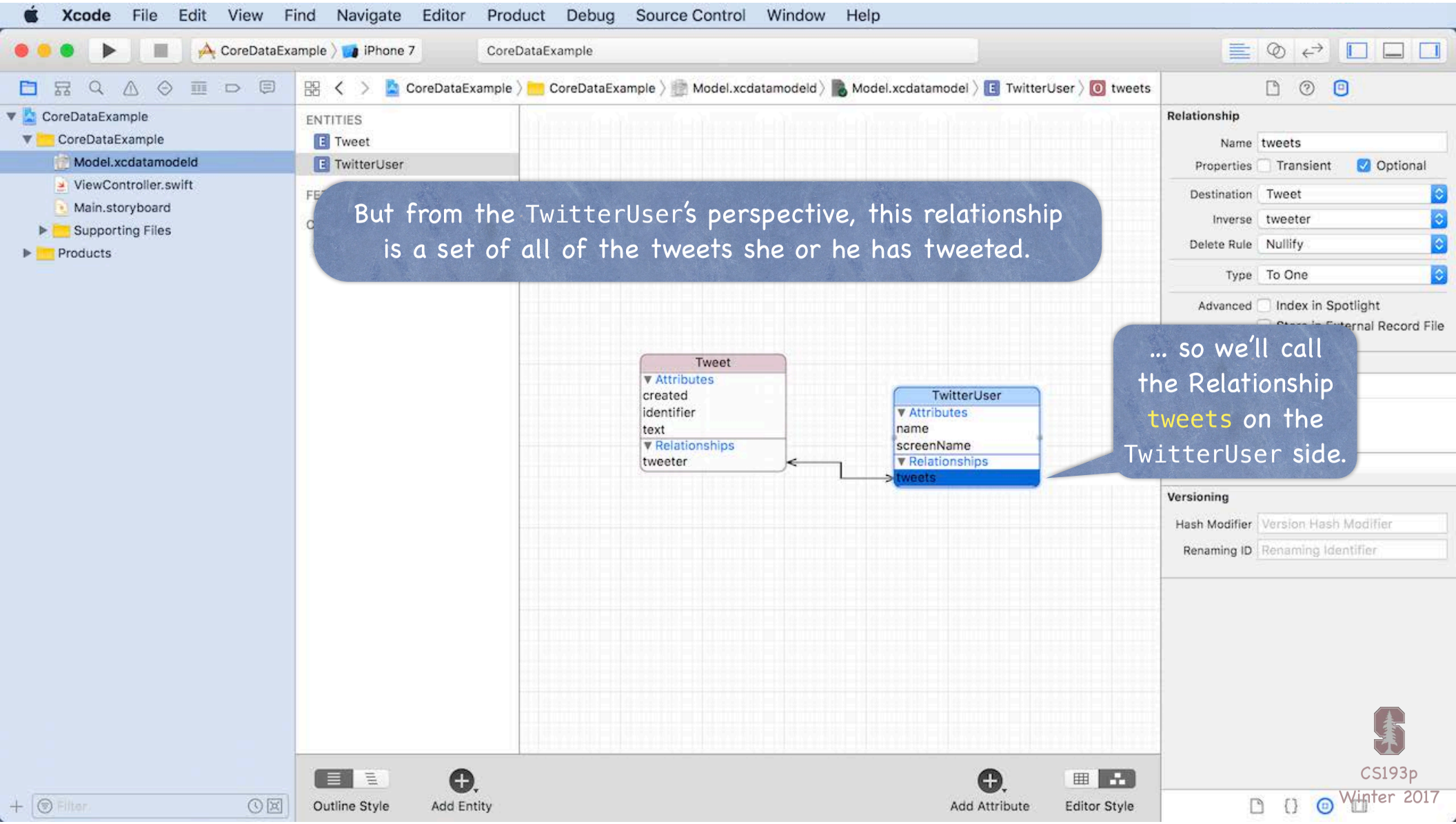

Xcode File Edit View Find Navigate Editor Product Debug Source Control Window Help
CoreDataExample 〉 iPhone 7
CoreDataExample
CoreDataExample 〉 CoreDataExample 〉 Model.xcdatamodeld 〉 Model.xcdatamodel 〉 TwitterUser 〉 name
CoreDataExample
CoreDataExample
Model.xcdatamodeld
ViewController.swift
Main.storyboard
Supporting Files
Products
ENTITIES
Tweet
TwitterUser
FETCH REQUESTS
CONFIGURATIONS
Default
Tweet
Attributes
created
identifier
text
Relationships
TwitterUser
Attributes
name
screenName
Relationships
This one is the TwitterUser's actual name.
Attribute
Name name
Properties Transient Optional Indexed
Attribute Type String
Validation No Value Min Length
No Value Max Length
Default Value Default Value
Reg. Ex. Regular Expression
Advanced Index in Spotlight
Store in External Record File
User Info
Key Value
Versioning
Hash Modifier Version Hash Modifier
Renaming ID Renaming Identifier
Outline Style
Add Entity
Add Attribute
Editor Style
CS193p
Winter 2017

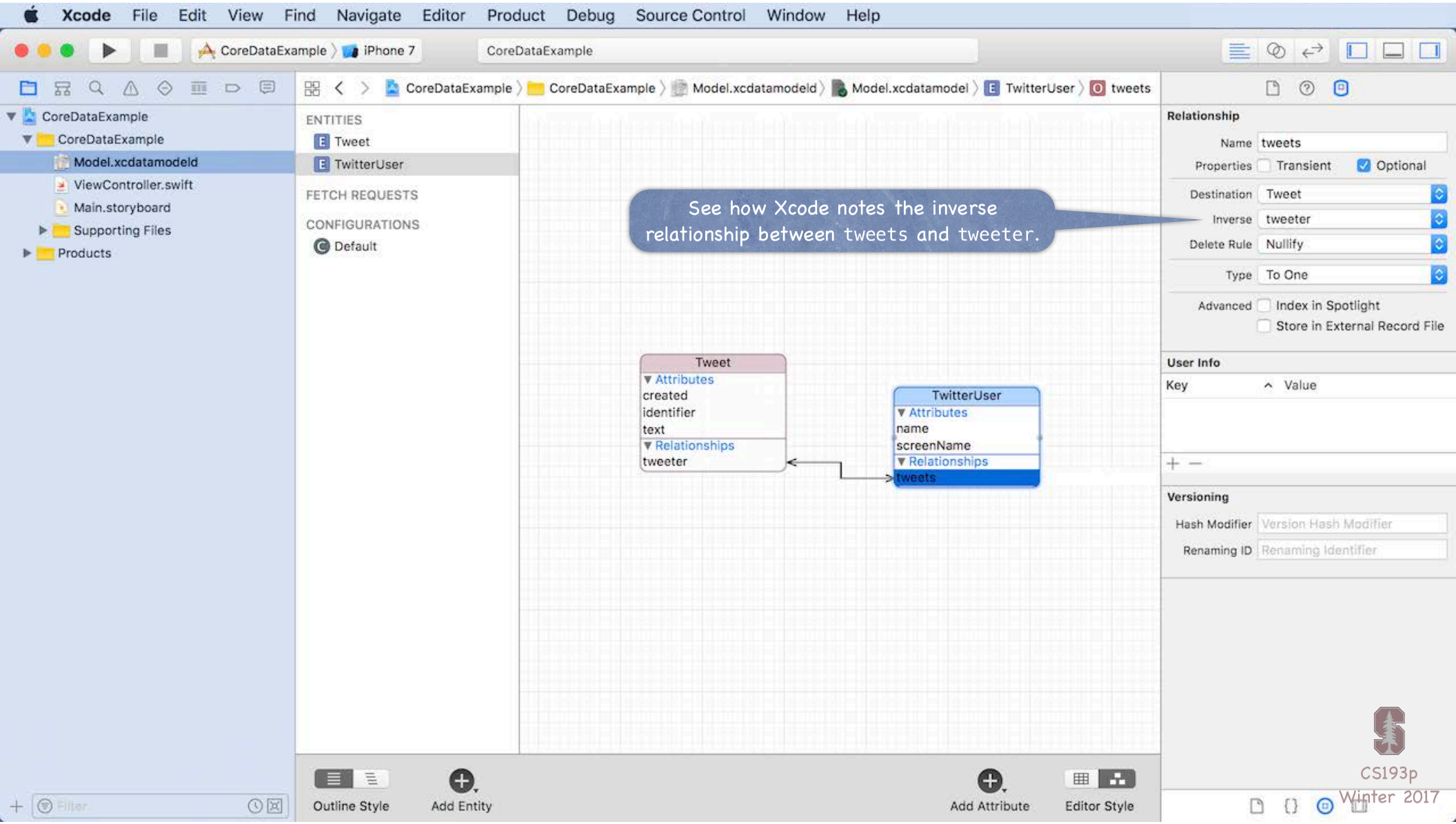

Xcode File Edit View Find Navigate Editor Product Debug Source Control Window Help
CoreDataExample 〉 iPhone 7
CoreDataExample
CoreDataExample 〉 CoreDataExample 〉 Model.xcdatamodeld 〉 Model.xcdatamodel 〉 Tweet
CoreDataExample
CoreDataExample
Model.xcdatamodeld
ViewController.swift
Main.storyboard
Supporting Files
Products
ENTITIES
Tweet
TwitterUser
FETCH REQUESTS
CONFIGURATIONS
Default
So far we've only added Attributes.
How about Relationships?
Tweet
Attributes
created
identifier
text
Relationships
TwitterUser
Attributes
name
screenName
Relationships
Entity
Name Tweet
Abstract Entity
Parent Entity No Parent Entity
Class
Name Tweet
Module Global namespace
Codegen Class Definition
Indexes
No Content
Constraints
No Content
User Info
Key Value
Versioning
Hash Modifier Version Hash Modifier
Renaming ID Renaming Identifier
Outline Style
Add Entity
Add Attribute
Editor Style
CS193p
Winter 2017

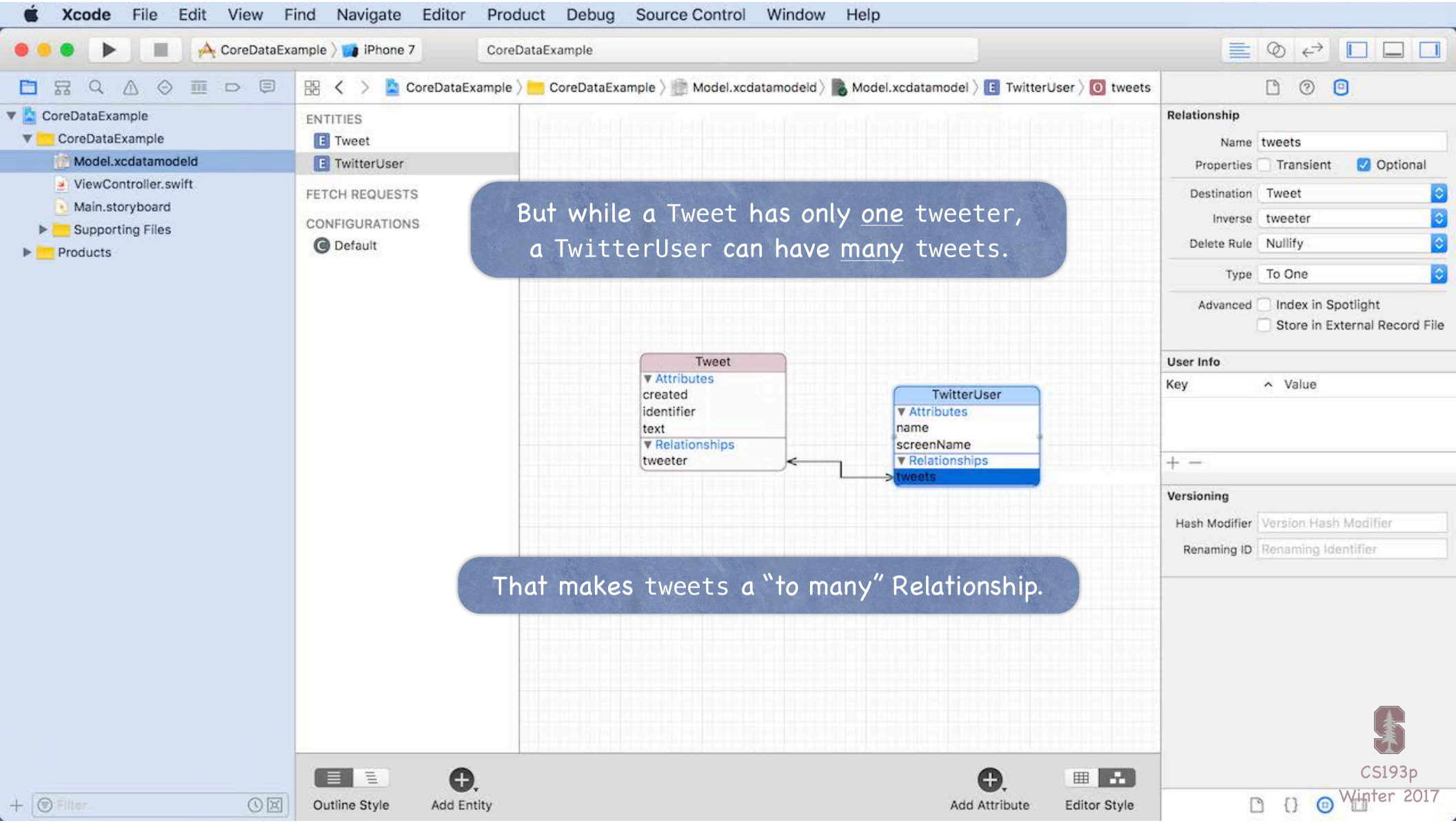

Xcode File Edit View Find Navigate Editor Product Debug Source Control Window Help
CoreDataExample 〉 iPhone 7
CoreDataExample
CoreDataExample 〉 CoreDataExample 〉 Model.xcdatamodeld 〉 Model.xcdatamodel 〉 Tweet
CoreDataExample
CoreDataExample
Model.xcdatamodeld
ViewController.swift
Main.storyboard
Supporting Files
Products
ENTITIES
Tweet
TwitterUser
FETCH REQUESTS
CONFIGURATIONS
Default
Similar to outlets and actions, we can ctrl-drag to create Relationships between Entities.
Tweet
Attributes
created
identifier
text
Relationships
TwitterUser
Attributes
name
screenName
Relationships
Entity
Name Tweet
Abstract Entity
Parent Entity No Parent Entity
Class
Name Tweet
Module Global namespace
Codegen Class Definition
Indexes
No Content
Constraints
No Content
User Info
Key Value
Versioning
Hash Modifier Version Hash Modifier
Renaming ID Renaming Identifier
Outline Style
Add Entity
Add Attribute
Editor Style
CS193p
Winter 2017

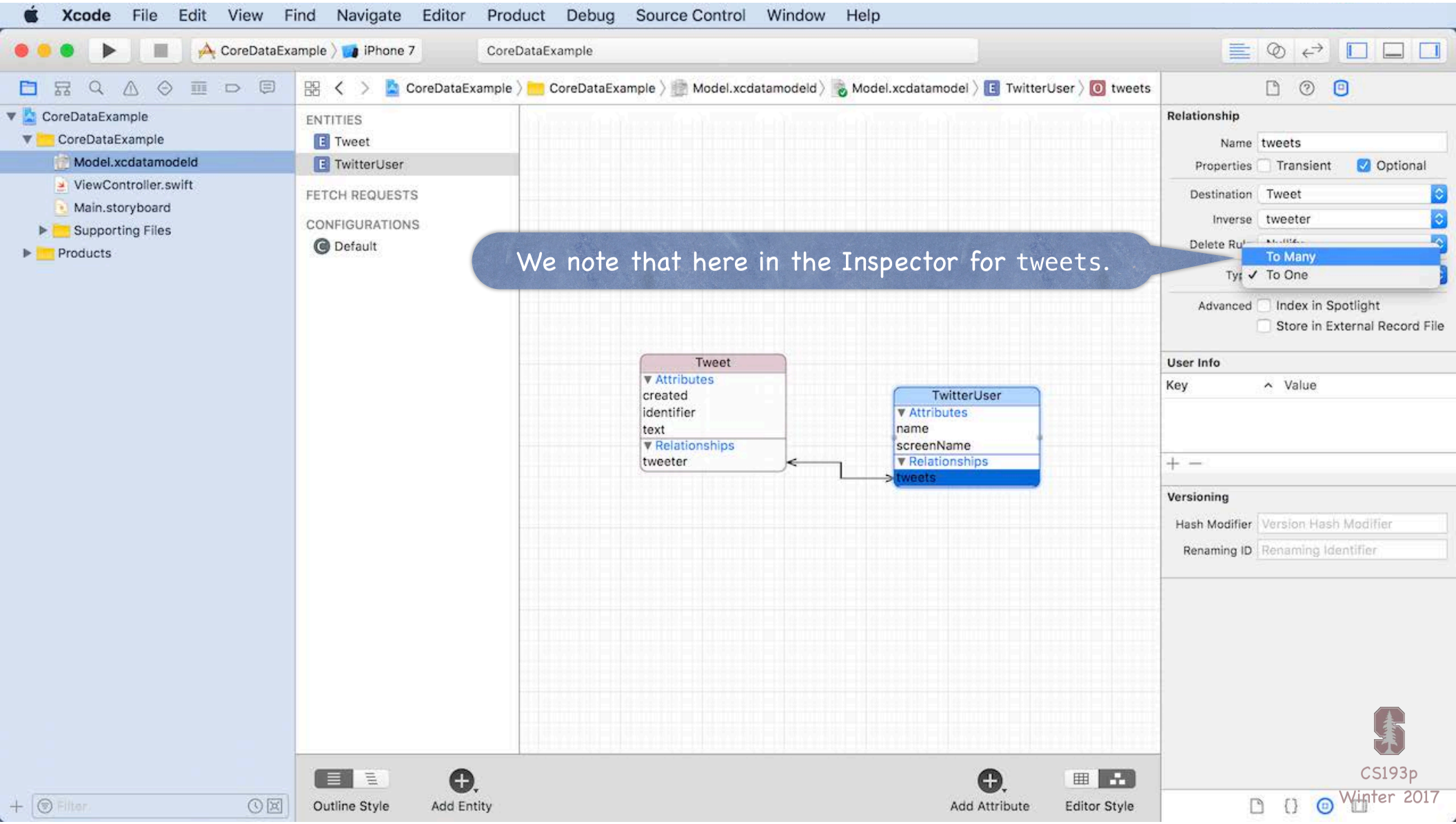

Xcode File Edit View Find Navigate Editor Product Debug Source Control Window Help
CoreDataExample 〉 iPhone 7
CoreDataExample
CoreDataExample 〉 CoreDataExample 〉 Model.xcdatamodeld 〉 Model.xcdatamodel 〉 Tweet
CoreDataExample
CoreDataExample
Model.xcdatamodeld
ViewController.swift
Main.storyboard
Supporting Files
Products
ENTITIES
Tweet
TwitterUser
FETCH REQUESTS
CONFIGURATIONS
Default
A Relationship is analogous to a pointer to another object (or an NSSet of other objects).
Tweet
Attributes
created
identifier
text
Relationships
newRelationship
TwitterUser
Attributes
name
screenName
Relationships
newRelationship
Entity
Name Multiple Values
Abstract Entity
Parent Entity No Parent Entity
Class
Name Multiple Values
Module Global namespace
Codegen Class Definition
Indexes No Content
Constraints No Content
User Info
Key Value
Versioning
Hash Modifier Version Hash Modifier
Renaming ID Renaming Identifier
Outline Style
Add Entity
Add Attribute
Editor Style
CS193p
Winter 2017

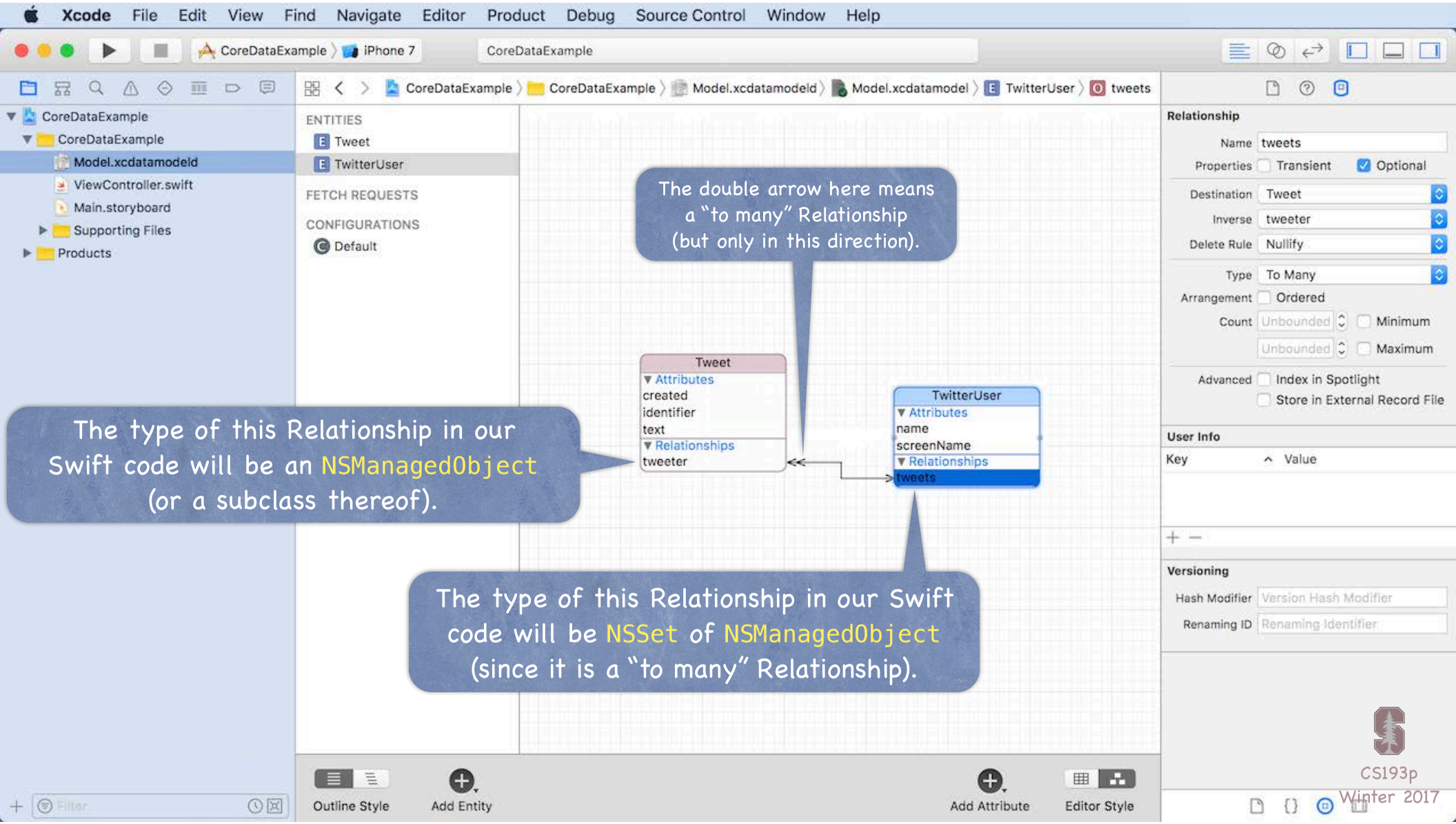

Xcode File Edit View Find Navigate Editor Product Debug Source Control Window Help
CoreDataExample iPhone 7
CoreDataExample
CoreDataExample CoreDataExample Model.xcdatamodeld Model.xcdatamodel Tweet tweeter
CoreDataExample
CoreDataExample
Model.xcdatamodeld
ViewController.swift
Main.storyboard
Supporting Files
Products
ENTITIES
Tweet
TwitterUser
FETCH REQUESTS
CONFIGURATIONS
Default
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.
Tweet
Attributes
created
identifier
text
Relationships
tweeter
TwitterUser
Attributes
name
screenName
Relationships
newRelationship
Relationship
Name tweeter
Properties Transient Optional
Destination TwitterUser
Inverse newRelationship
Delete Rule Nullify
Type To One
Advanced Index in Spotlight
Store in External Record File
User Info
Key Value
Versioning
Hash Modifier Version Hash Modifier
Renaming ID Renaming Identifier
Outline Style
Add Entity
Add Attribute
Editor Style
CS193p
Winter 2017

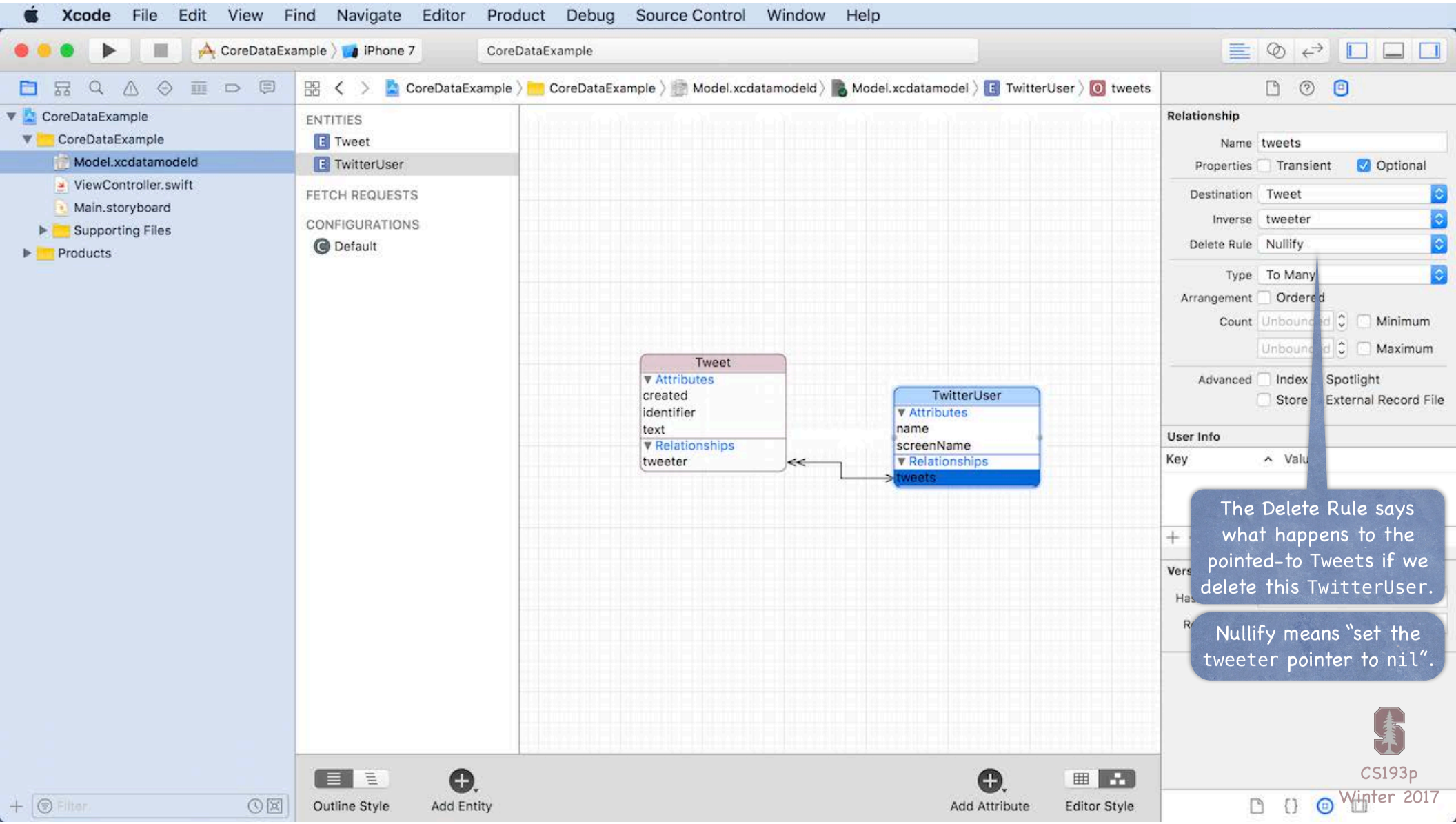

Xcode File Edit View Find Navigate Editor Product Debug Source Control Window Help
CoreDataExample iPhone 7
CoreDataExample
CoreDataExample CoreDataExample Model.xcdatamodeld Model.xcdatamodel TwitterUser tweets
CoreDataExample
CoreDataExample
Model.xcdatamodeld
ViewController.swift
Main.storyboard
Supporting Files
Products
ENTITIES
Tweet
TwitterUser
But from the TwitterUser's perspective, this relationship is a set of all of the tweets she or he has tweeted.
Tweet
Attributes
created
identifier
text
Relationships
tweeter
TwitterUser
Attributes
name
screenName
Relationships
tweets
... so we'll call the Relationship tweets on the TwitterUser side.
Relationship
Name tweets
Properties Transient Optional
Destination Tweet
Inverse tweeter
Delete Rule Nullify
Type To One
Advanced Index in Spotlight
Versioning
Hash Modifier Version Hash Modifier
Renaming ID Renaming Identifier
Outline Style
Add Entity
Add Attribute
Editor Style
CS193p
Winter 2017


Xcode File Edit View Find Navigate Editor Product Debug Source Control Window Help
CoreDataExample 〉 iPhone 7
CoreDataExample
CoreDataExample 〉 CoreDataExample 〉 Model.xcdatamodeld 〉 Model.xcdatamodel 〉 TwitterUser 〉 tweets
CoreDataExample
CoreDataExample
Model.xcdatamodeld
ViewController.swift
Main.storyboard
Supporting Files
Products
ENTITIES
Tweet
TwitterUser
FETCH REQUESTS
CONFIGURATIONS
Default
See how Xcode notes the inverse relationship between tweets and tweeter.
Tweet
Attributes
created
identifier
text
Relationships
tweeter
TwitterUser
Attributes
name
screenName
Relationships
tweets
Relationship
Name tweets
Properties Transient Optional
Destination Tweet
Inverse tweeter
Delete Rule Nullify
Type To One
Advanced Index in Spotlight
Store in External Record File
User Info
Key Value
Versioning
Hash Modifier Version Hash Modifier
Renaming ID Renaming Identifier
Outline Style
Add Entity
Add Attribute
Editor Style
CS193p
Winter 2017


Xcode File Edit View Find Navigate Editor Product Debug Source Control Window Help
CoreDataExample 〉 iPhone 7
CoreDataExample
CoreDataExample 〉 CoreDataExample 〉 Model.xcdatamodeld 〉 Model.xcdatamodel 〉 TwitterUser 〉 tweets
CoreDataExample
CoreDataExample
Model.xcdatamodeld
ViewController.swift
Main.storyboard
Supporting Files
Products
ENTITIES
Tweet
TwitterUser
FETCH REQUESTS
CONFIGURATIONS
Default
But while a Tweet has only one tweeter,
a TwitterUser can have many tweets.
Tweet
Attributes
created
identifier
text
Relationships
tweeter
TwitterUser
Attributes
name
screenName
Relationships
tweets
That makes tweets a "to many" Relationship.
Relationship
Name tweets
Properties Transient Optional
Destination Tweet
Inverse tweeter
Delete Rule Nullify
Type To One
Advanced Index in Spotlight
Store in External Record File
User Info
Key Value
Versioning
Hash Modifier Version Hash Modifier
Renaming ID Renaming Identifier
Outline Style
Add Entity
Add Attribute
Editor Style
CS193p
Winter 2017


Xcode File Edit View Find Navigate Editor Product Debug Source Control Window Help
CoreDataExample 〉 iPhone 7
CoreDataExample
CoreDataExample 〉 CoreDataExample 〉 Model.xcdatamodeld 〉 Model.xcdatamodel 〉 TwitterUser 〉 tweets
CoreDataExample
CoreDataExample
Model.xcdatamodeld
ViewController.swift
Main.storyboard
Supporting Files
Products
ENTITIES
Tweet
TwitterUser
FETCH REQUESTS
CONFIGURATIONS
Default
We note that here in the Inspector for tweets.
Tweet
Attributes
created
identifier
text
Relationships
tweeter
TwitterUser
Attributes
name
screenName
Relationships
tweets
Relationship
Name tweets
Properties Transient Optional
Destination Tweet
Inverse tweeter
To Many
To One
Advanced Index in Spotlight
Store in External Record File
User Info
Key Value
Versioning
Hash Modifier Version Hash Modifier
Renaming ID Renaming Identifier
Outline Style
Add Entity
Add Attribute
Editor Style
CS193p
Winter 2017


Xcode File Edit View Find Navigate Editor Product Debug Source Control Window Help
CoreDataExample 〉 iPhone 7
CoreDataExample
CoreDataExample 〉 CoreDataExample 〉 Model.xcdatamodeld 〉 Model.xcdatamodel 〉 TwitterUser 〉 tweets
CoreDataExample
CoreDataExample
Model.xcdatamodeld
ViewController.swift
Main.storyboard
Supporting Files
Products
ENTITIES
Tweet
TwitterUser
FETCH REQUESTS
CONFIGURATIONS
Default
The double arrow here means a "to many" Relationship (but only in this direction).
Tweet
Attributes
created
identifier
text
Relationships
tweeter
TwitterUser
Attributes
name
screenName
Relationships
tweets
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).
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 in Spotlight
Store in External Record File
User Info
Key Value
Versioning
Hash Modifier Version Hash Modifier
Renaming ID Renaming Identifier
Outline Style
Add Entity
Add Attribute
Editor Style
CS193p
Winter 2017


Xcode File Edit View Find Navigate Editor Product Debug Source Control Window Help
CoreDataExample 〉 iPhone 7
CoreDataExample
CoreDataExample 〉 CoreDataExample 〉 Model.xcdatamodeld 〉 Model.xcdatamodel 〉 TwitterUser 〉 tweets
CoreDataExample
CoreDataExample
Model.xcdatamodeld
ViewController.swift
Main.storyboard
Supporting Files
Products
ENTITIES
Tweet
TwitterUser
FETCH REQUESTS
CONFIGURATIONS
Default
Tweet
Attributes
created
identifier
text
Relationships
tweeter
TwitterUser
Attributes
name
screenName
Relationships
tweets
Relationship
Name tweets
Properties Transient Optional
Destination Tweet
Inverse tweeter
Delete Rule Nullify
Type To Many
Arrangement Ordered
Count Minimum
Maximum
Advanced Index Spotlight
Store External Record File
User Info
Key
The Delete Rule says what happens to the pointed-to Tweets if we delete this TwitterUser.
Nullify means "set the tweeter pointer to nil".
Outline Style
Add Entity
Add Attribute
Editor Style
CS193p
Winter 2017

Core Data

- There are lots of other things you can do
  - But we are going to focus on 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 a context?
  - You get one out of an `NSPersistentContainer`.
  - The code that the `Use Core Data` button adds creates one for you in your `AppDelegate`.
  - (You could easily see how to create multiple of them by looking at that code.)

  You can access that `AppDelegate var` like this ...

```
(UIApplication.shared.delegate as! AppDelegate).persistentContainer
```

Core Data

- Getting the NSManagedObjectContext

  We get the context we need from the persistentContainer using its viewContext var.

  This returns an NSManagedObjectContext suitable (only) for use on the main queue.

```
let container = (UIApplication.shared.delegate as! AppDelegate).persistentContainer
let context: NSManagedObjectContext = container.viewContext
```

Core Data

- Convenience

```
(UIApplication.shared.delegate as! AppDelegate).persistentContainer
```

… is a kind of messy line of code.

So sometimes we'll add a `static` version to `AppDelegate` …

```
static var persistentContainer: NSPersistentContainer {
    return (UIApplication.shared.delegate as! AppDelegate).persistentContainer
}
```

… so you can access the container like this …

```
let coreDataContainer = AppDelegate.persistentContainer
```

… and possibly even add this `static var` too …

```
static var viewContext: NSManagedObjectContext {
    return persistentContainer.viewContext
}
```

… so that we can do this …

```
let context = AppDelegate.viewContext
```

Core Data

- Okay, we have an NSManagedObjectContext, now what?

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

- Inserting objects into the database

```
let context = AppDelegate.viewContext
let tweet: NSManagedObject =
    NSEntityDescription.insertNewObject(forEntityName: "Tweet", into: context)
```

  Note that this NSEntityDescription class method returns an NSManagedObject instance.
  All objects in the database are represented by NSManagedObjects or subclasses thereof.
  An instance of NSManagedObject is a manifestation of an Entity in our Core Data Model*.
  Attributes of a newly-inserted object will start out nil (unless you specify a default in Xcode).

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

Core Data

- How to access Attributes in an NSManagedObject instance
  - You can access them using the following two NSKeyValueCoding protocol methods …
  - `func value(forKey: String) -> Any?`
  - `func setValue(Any?, forKey: String)`
  - Using `value(forKeyPath:)/setValue(_,forKeyPath:)` (with dots) will follow your Relationships!
  - `let username = tweet.value(forKeyPath: "tweeter.name") as? String`
- 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).
  - Numbers are `Double`, `Int`, etc. (if Use Scalar Type checked in Data Model Editor in Xcode).
  - Binary data values are `NSData`s.
  - Date values are `NSDate`s.
  - "To-many" relationships are `NSSet`s but can be cast (with `as?`) to `Set<NSManagedObject>`
  - "To-one" relationships are `NSManagedObject`s.

Core Data

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

  You must explicitly save any changes to a context, but note that this throws.

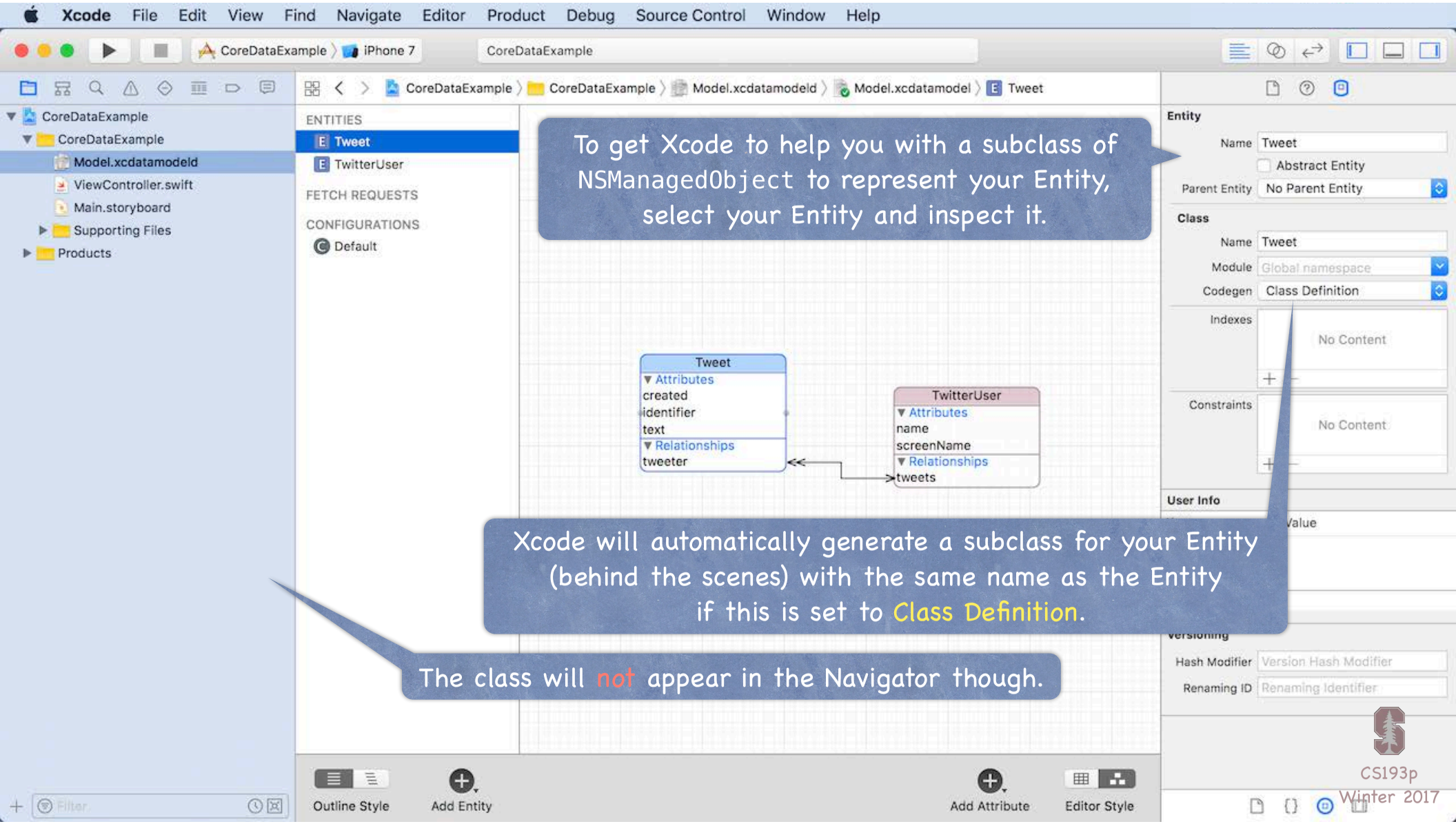
```
do {
    try context.save()
} catch {     // note, by default catch catches any error into a local variable called error
    // deal with error
}
```

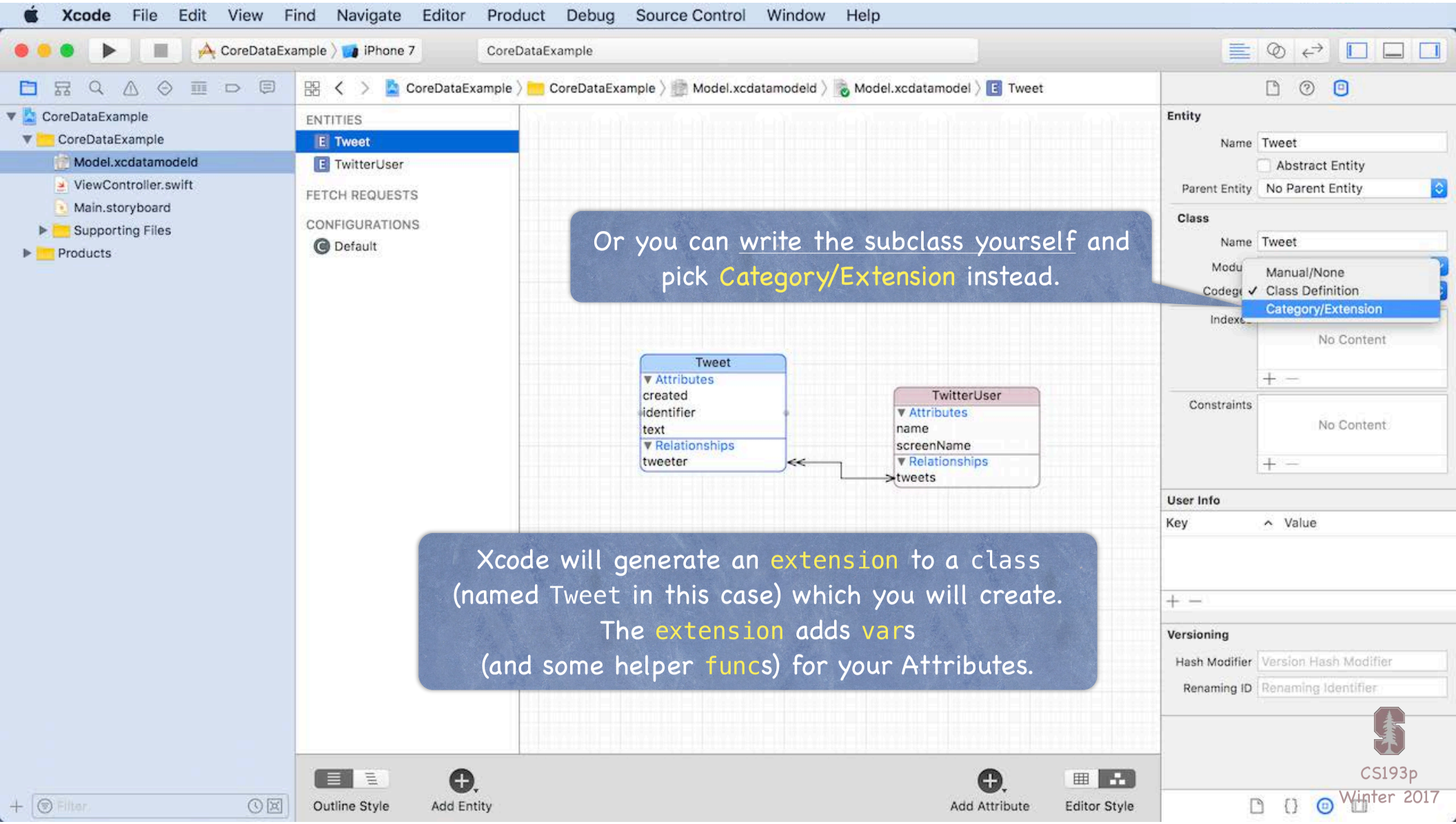
  Don't forget to save your changes any time you touch the database!

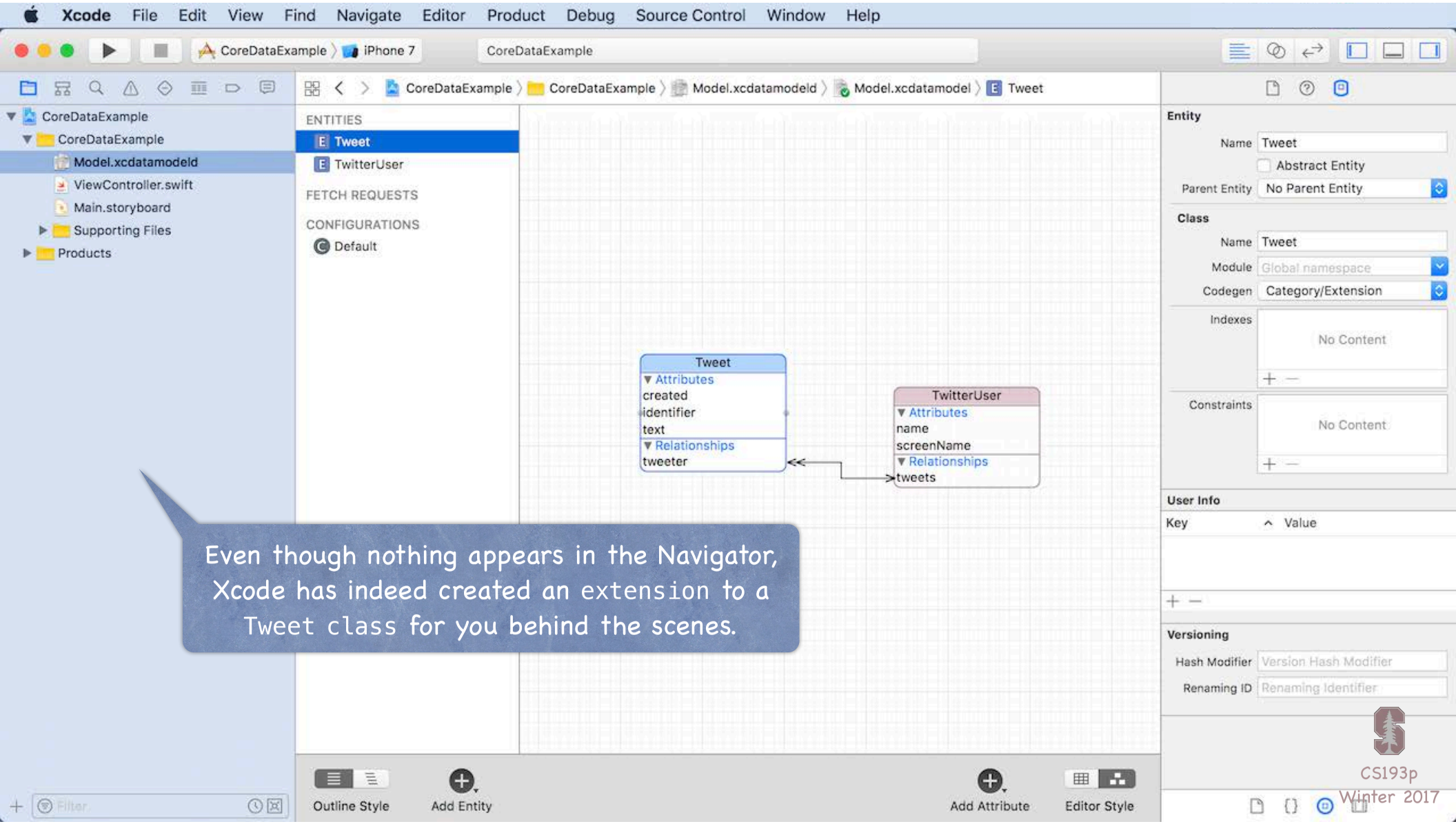
  Of course you will want to group up as many changes into a single save as possible.

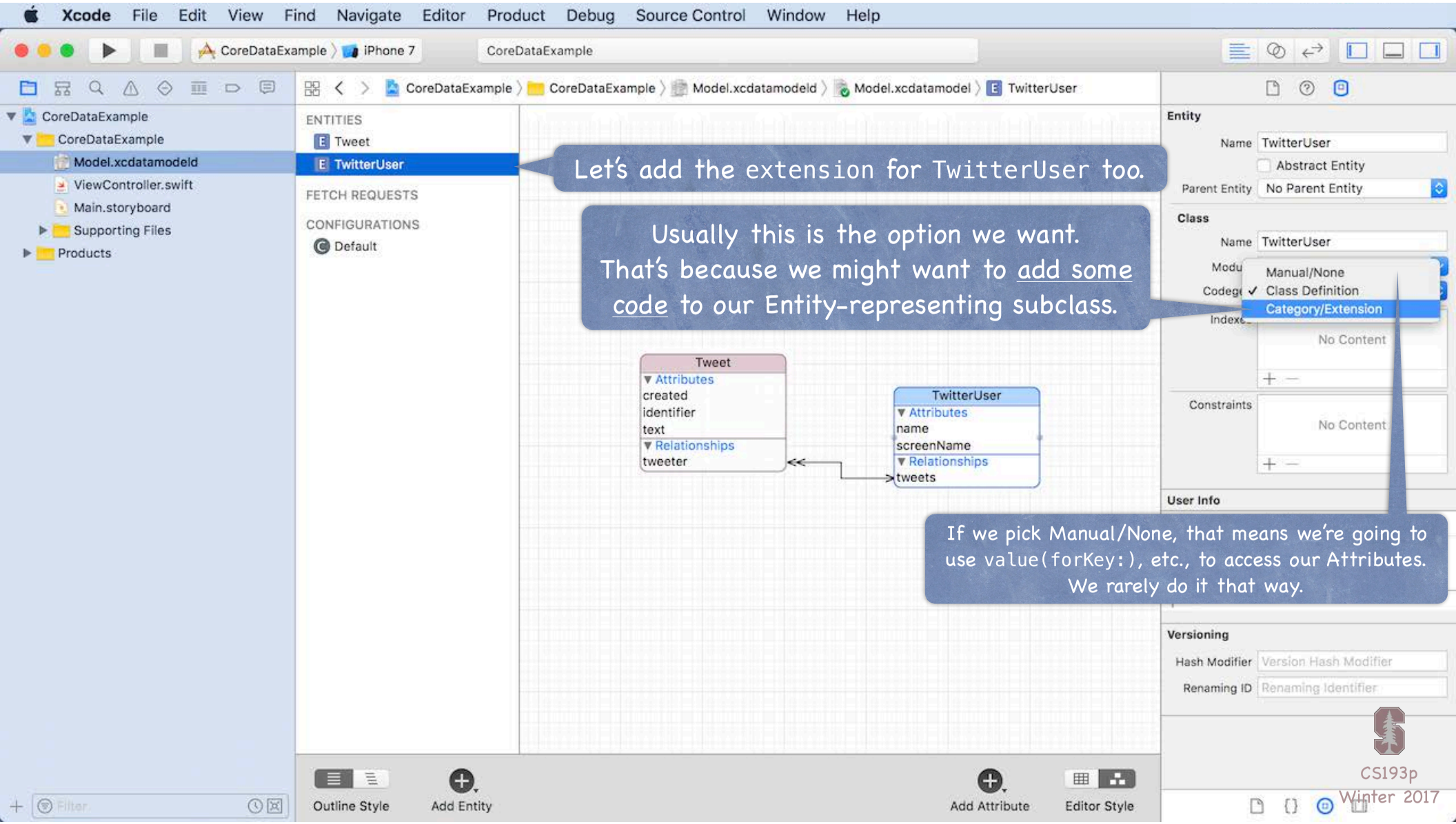
Core Data

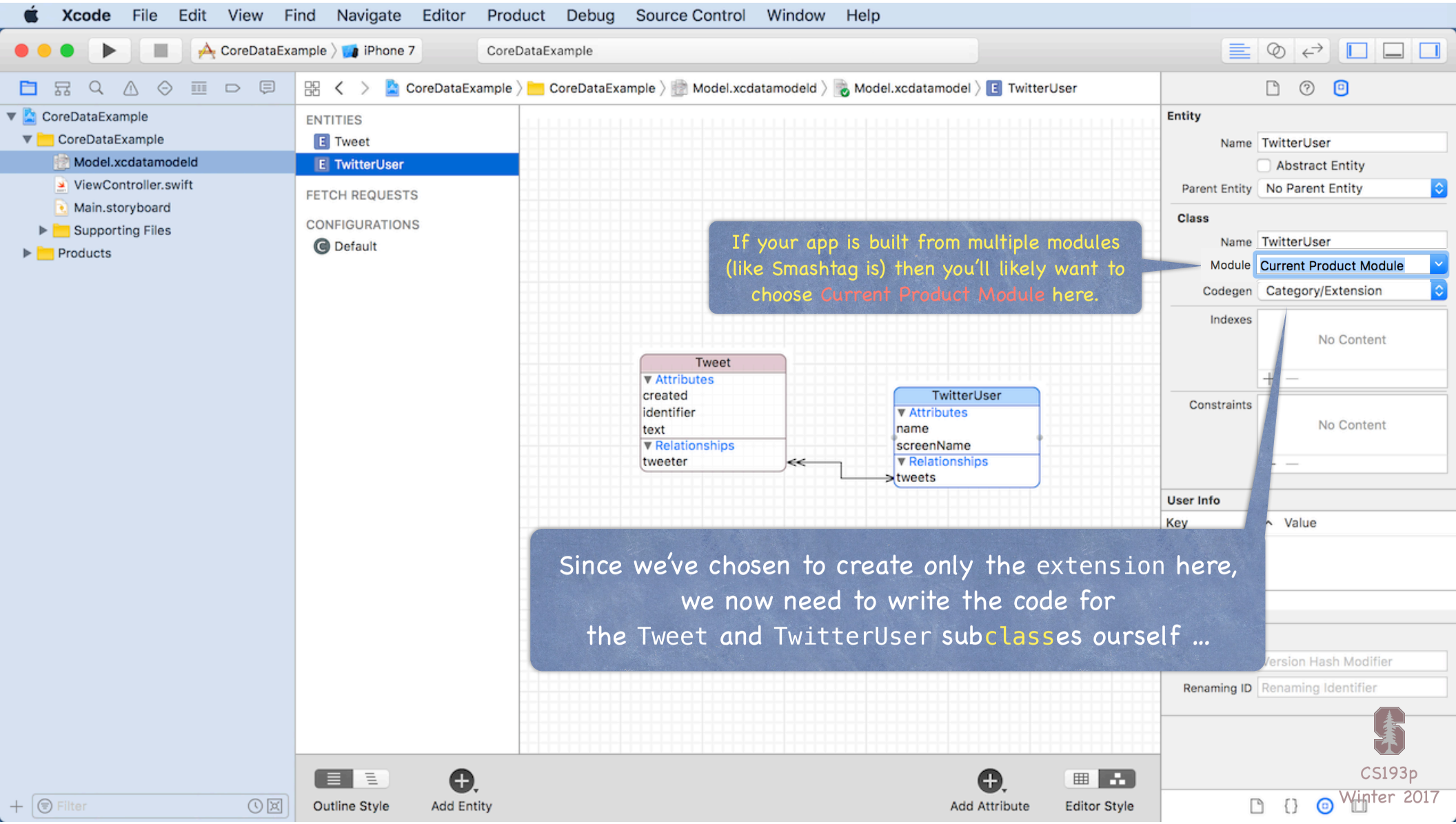
- But calling `value(forKey:)/setValue(_,forKey:)` is pretty ugly
  - There's no type-checking.
  - And you have a lot of literal strings in your code (e.g. "`created`").
- What we really want is to set/get using vars!
- No problem … we just create a subclass of `NSManagedObject`
  - The subclass will have vars for each attribute in the database.
  - We name our subclass the same name as the Entity it matches (not strictly required, but do it).
  - We can get Xcode to generate all the code necessary to make this work.

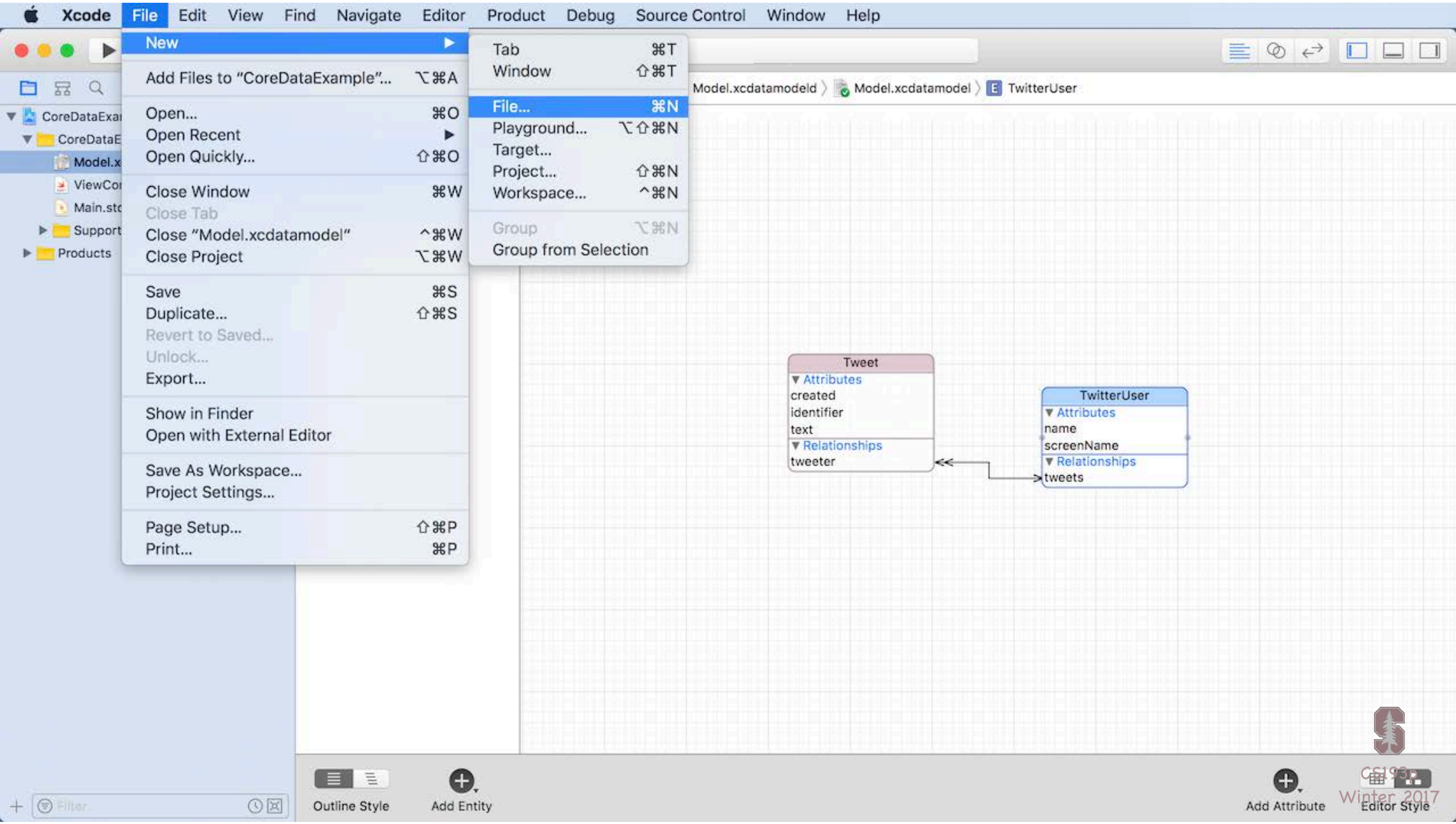

Xcode File Edit View Find Navigate Editor Product Debug Source Control Window Help
CoreDataExample
CoreDataExample
ENTITIES
Tweet
TwitterUser
FETCH REQUESTS
CONFIGURATIONS
Default
To get Xcode to help you with a subclass of NSManagedObject to represent your Entity, select your Entity and inspect it.
Tweet
Attributes
created
identifier
text
Relationships
tweeter
TwitterUser
Attributes
name
screenName
Relationships
tweets
Entity
Name Tweet
Abstract Entity
Parent Entity No Parent Entity
Class
Name Tweet
Module Global namespace
Codegen Class Definition
Indexes
No Content
Constraints
No Content
User Info
Xcode will automatically generate a subclass for your Entity (behind the scenes) with the same name as the Entity if this is set to Class Definition.
The class will not appear in the Navigator though.
Hash Modifier Version Hash Modifier
Renaming ID Renaming Identifier
Outline Style
Add Entity
Add Attribute
Editor Style
CS193p
Winter 2017

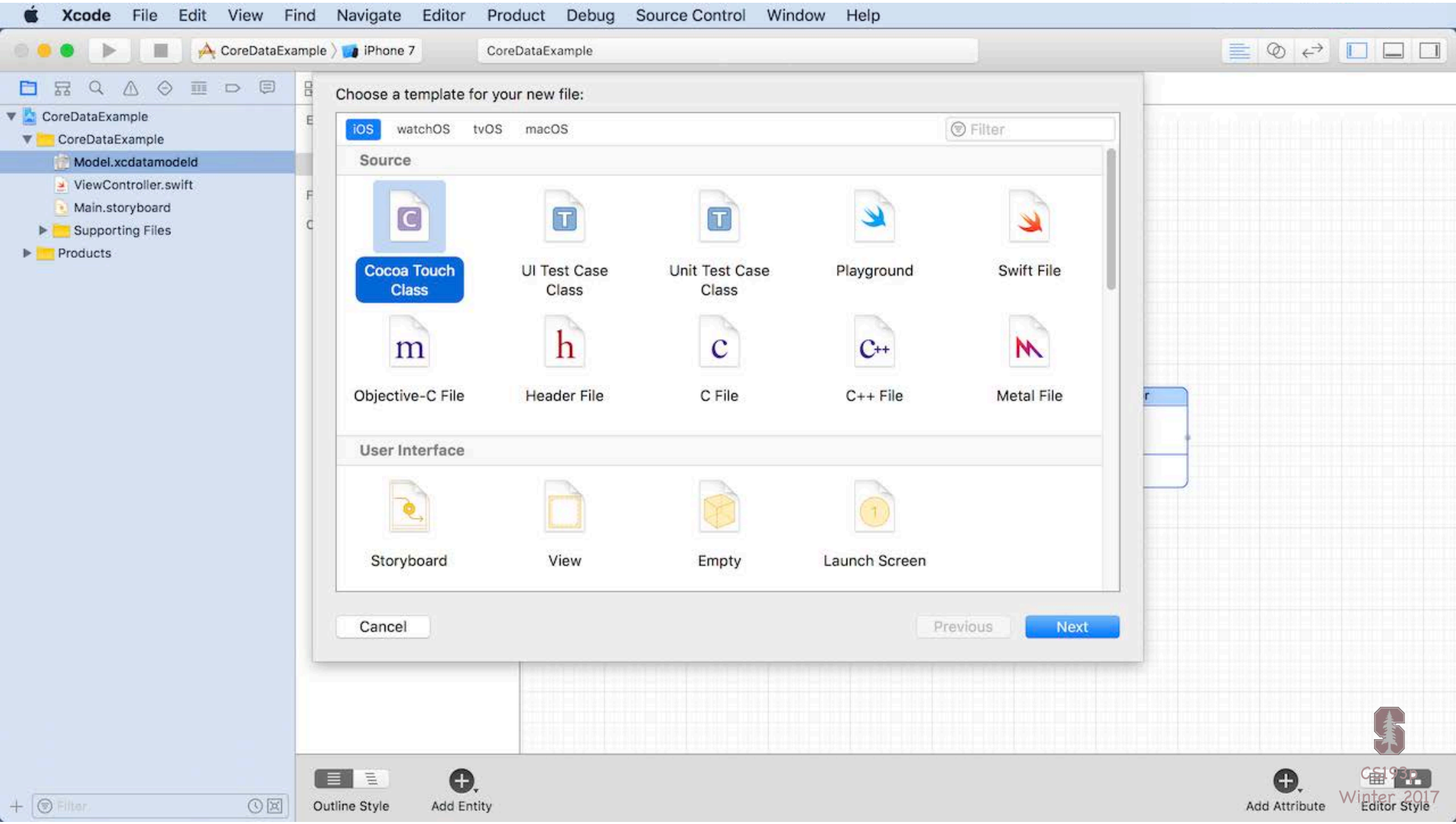

Xcode File Edit View Find Navigate Editor Product Debug Source Control Window Help
CoreDataExample
CoreDataExample
CoreDataExample
Model.xcdatamodeld
ViewController.swift
Main.storyboard
Supporting Files
Products
ENTITIES
Tweet
TwitterUser
FETCH REQUESTS
CONFIGURATIONS
Default
Or you can write the subclass yourself and pick Category/Extension instead.
Tweet
Attributes
created
identifier
text
Relationships
tweeter
TwitterUser
Attributes
name
screenName
Relationships
tweets
Xcode will generate an extension to a class (named Tweet in this case) which you will create. The extension adds vars (and some helper funcs) for your Attributes.
Entity
Name Tweet
Abstract Entity
Parent Entity No Parent Entity
Class
Name Tweet
Manual/None
Class Definition
Category/Extension
No Content
Constraints
No Content
User Info
Key
Value
Versioning
Hash Modifier
Renaming ID
Outline Style
Add Entity
Add Attribute
Editor Style
CS193p
Winter 2017

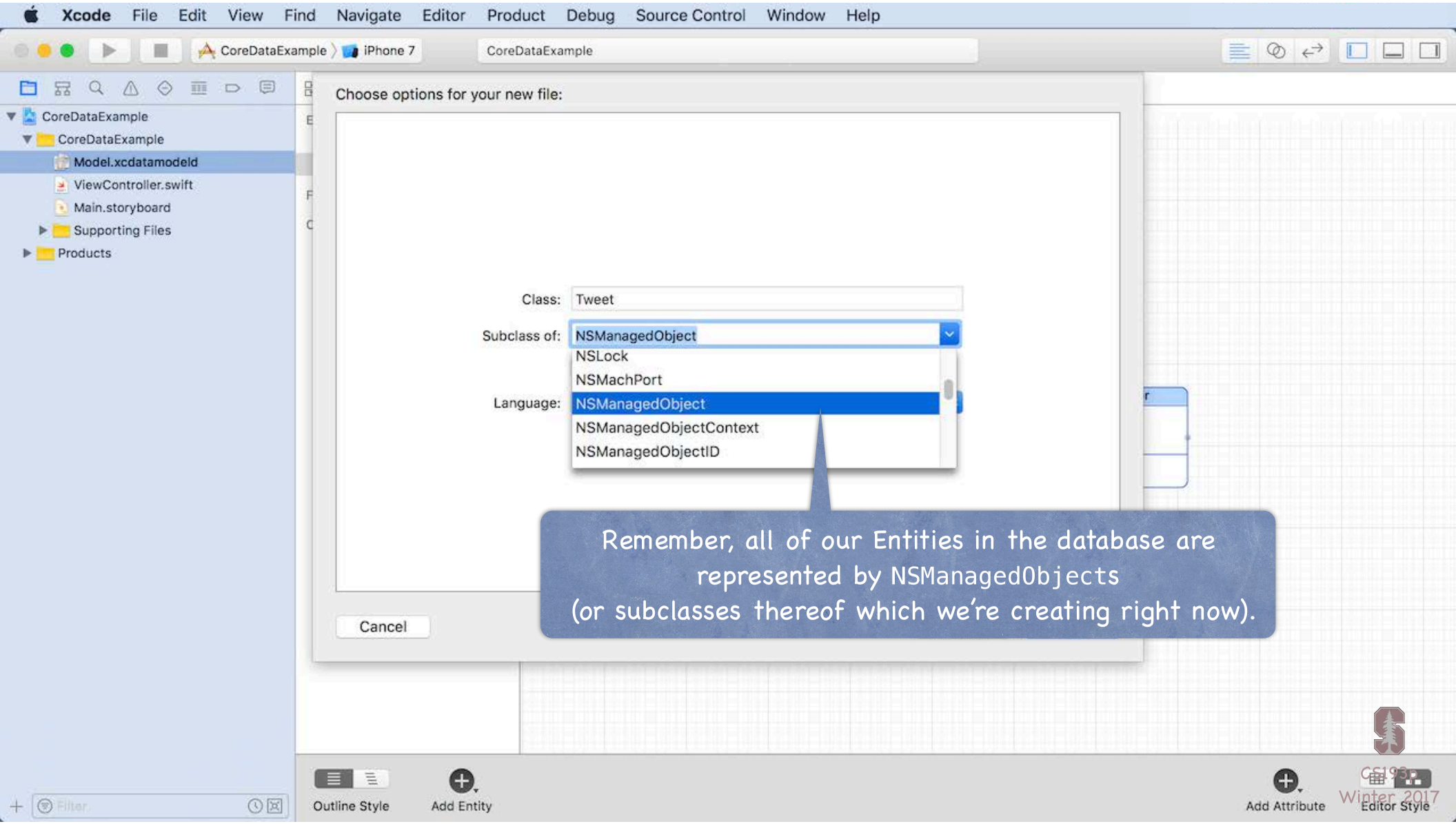

Xcode File Edit View Find Navigate Editor Product Debug Source Control Window Help
CoreDataExample 〉 iPhone 7
CoreDataExample
CoreDataExample 〉 CoreDataExample 〉 Model.xcdatamodeld 〉 Model.xcdatamodel 〉 Tweet
CoreDataExample
CoreDataExample
Model.xcdatamodeld
ViewController.swift
Main.storyboard
Supporting Files
Products
ENTITIES
Tweet
TwitterUser
FETCH REQUESTS
CONFIGURATIONS
Default
Tweet
Attributes
created
identifier
text
Relationships
tweeter
TwitterUser
Attributes
name
screenName
Relationships
tweets
Even though nothing appears in the Navigator, Xcode has indeed created an extension to a Tweet class for you behind the scenes.
Entity
Name Tweet
Abstract Entity
Parent Entity No Parent Entity
Class
Name Tweet
Module Global namespace
Codegen Category/Extension
Indexes
No Content
Constraints
No Content
User Info
Key Value
Versioning
Hash Modifier Version Hash Modifier
Renaming ID Renaming Identifier
Outline Style
Add Entity
Add Attribute
Editor Style
CS193p
Winter 2017

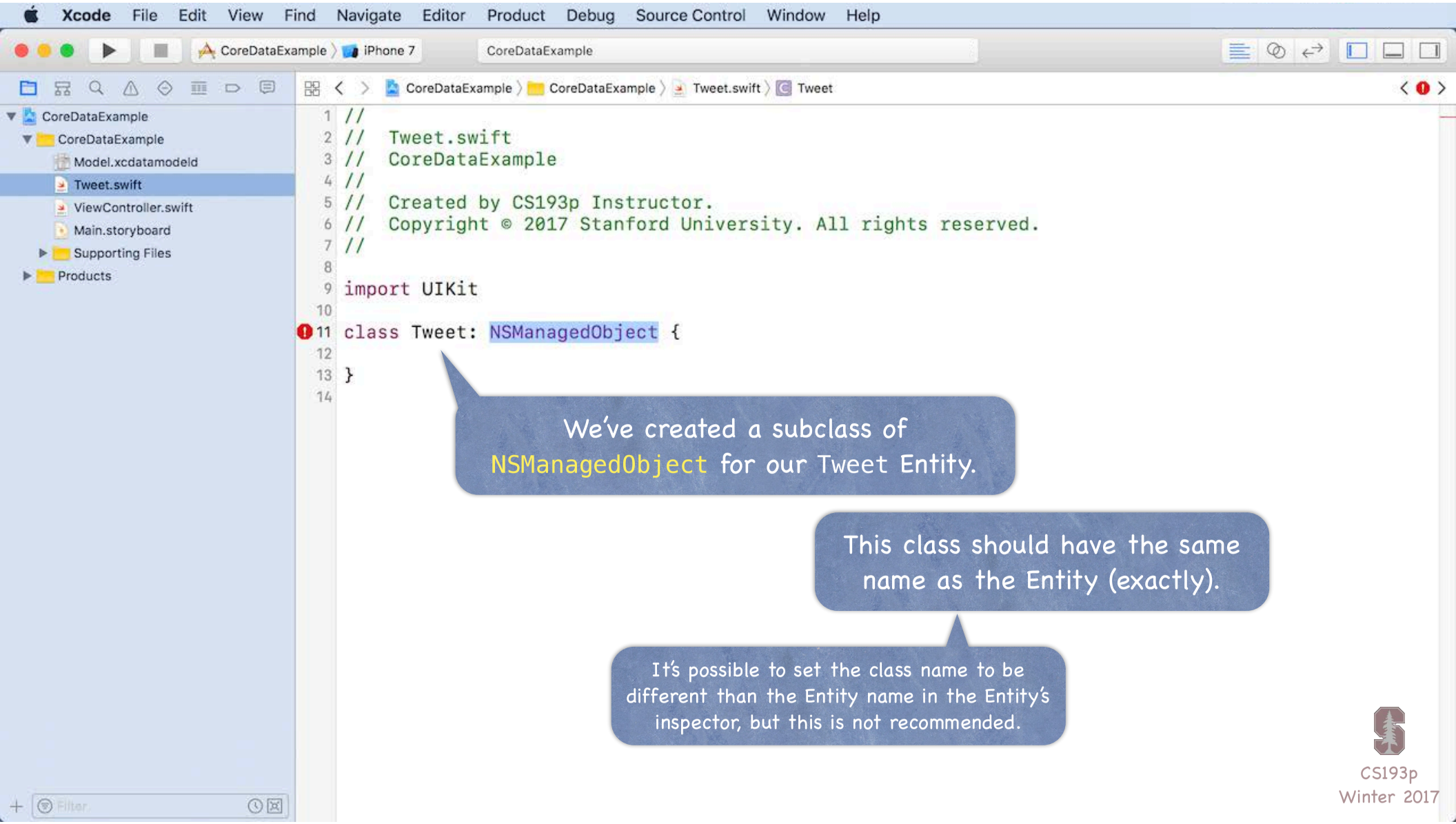

Xcode File Edit View Find Navigate Editor Product Debug Source Control Window Help
CoreDataExample 〉 iPhone 7
CoreDataExample
CoreDataExample 〉 CoreDataExample 〉 Model.xcdatamodeld 〉 Model.xcdatamodel 〉 TwitterUser
CoreDataExample
CoreDataExample
Model.xcdatamodeld
ViewController.swift
Main.storyboard
Supporting Files
Products
ENTITIES
Tweet
TwitterUser
FETCH REQUESTS
CONFIGURATIONS
Default
Let's add the extension for TwitterUser too.
Usually this is the option we want.
That's because we might want to add some code to our Entity-representing subclass.
Tweet
Attributes
created
identifier
text
Relationships
tweeter
TwitterUser
Attributes
name
screenName
Relationships
tweets
Entity
Name TwitterUser
Abstract Entity
Parent Entity No Parent Entity
Class
Name TwitterUser
Manual/None
Class Definition
Category/Extension
No Content
Constraints
No Content
User Info
If we pick Manual/None, that means we're going to use value(forKey:), etc., to access our Attributes. We rarely do it that way.
Versioning
Hash Modifier Version Hash Modifier
Renaming ID Renaming Identifier
Outline Style
Add Entity
Add Attribute
Editor Style
CS193p
Winter 2017

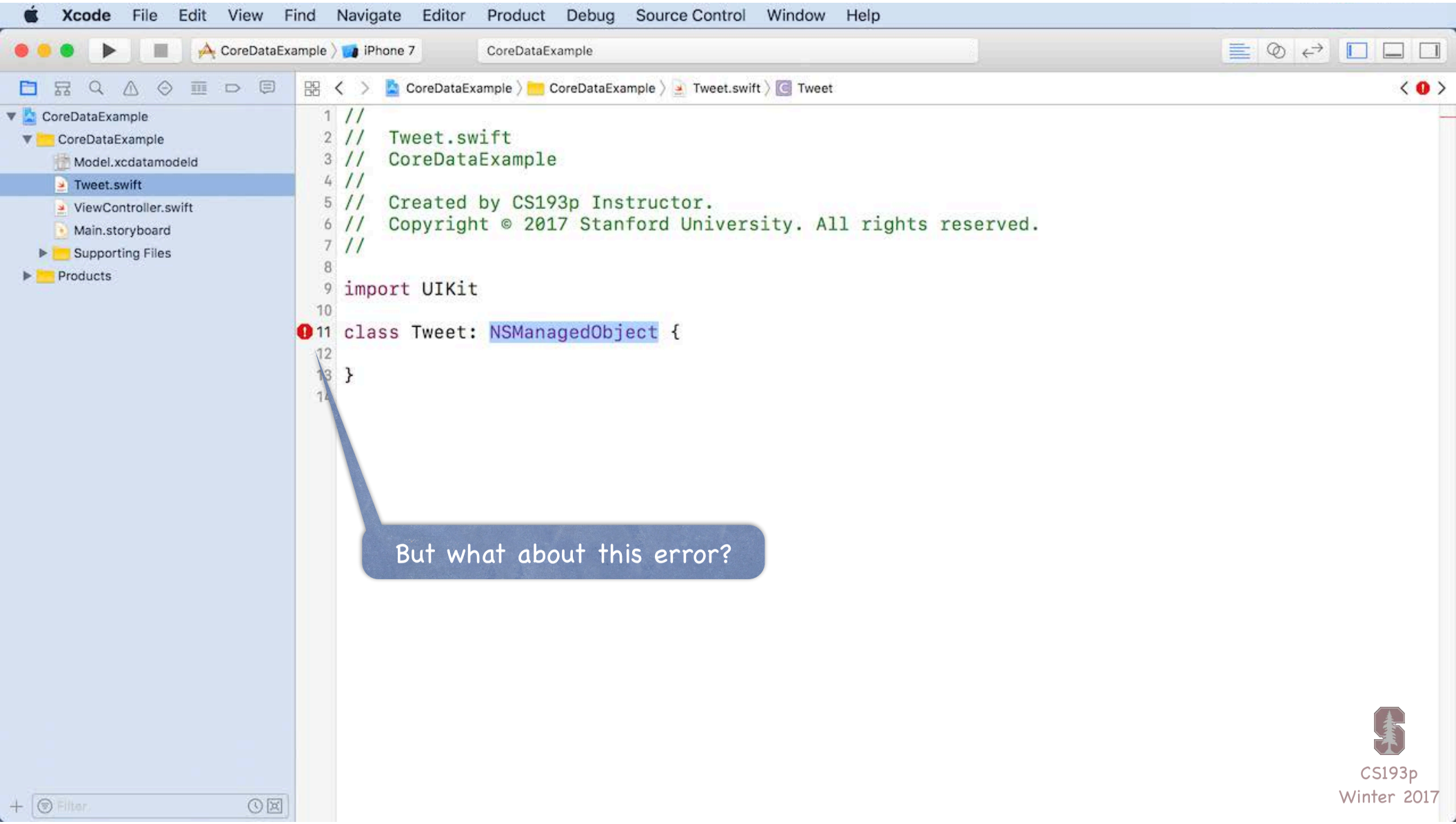

Xcode File Edit View Find Navigate Editor Product Debug Source Control Window Help
CoreDataExample 〉 iPhone 7
CoreDataExample
CoreDataExample 〉 CoreDataExample 〉 Model.xcdatamodeld 〉 Model.xcdatamodel 〉 TwitterUser
CoreDataExample
CoreDataExample
Model.xcdatamodeld
ViewController.swift
Main.storyboard
Supporting Files
Products
ENTITIES
Tweet
TwitterUser
FETCH REQUESTS
CONFIGURATIONS
Default
If your app is built from multiple modules (like Smashtag is) then you'll likely want to choose Current Product Module here.
Tweet
Attributes
created
identifier
text
Relationships
tweeter
TwitterUser
Attributes
name
screenName
Relationships
tweets
Since we've chosen to create only the extension here, we now need to write the code for the Tweet and TwitterUser subclasses ourself …
Entity
Name TwitterUser
Abstract Entity
Parent Entity No Parent Entity
Class
Name TwitterUser
Module Current Product Module
Codegen Category/Extension
Indexes
No Content
Constraints
No Content
User Info
Key
Value
Version Hash Modifier
Renaming ID Renaming Identifier
Outline Style
Add Entity
Add Attribute
Editor Style
CS193p
Winter 2017


Xcode
File
Edit
View
Find
Navigate
Editor
Product
Debug
Source Control
Window
Help
New
Add Files to "CoreDataExample"... ⌥⌘A
Open... ⌘O
Open Recent
Open Quickly... ⇧⌘O
Close Window ⌘W
Close Tab
Close "Model.xcdatamodel" ^⌘W
Close Project ⌥⌘W
Save ⌘S
Duplicate... ⇧⌘S
Revert to Saved...
Unlock...
Export...
Show in Finder
Open with External Editor
Save As Workspace...
Project Settings...
Page Setup... ⇧⌘P
Print... ⌘P
Tab ⌘T
Window ⇧⌘T
File... ⌘N
Playground... ⌥⇧⌘N
Target...
Project... ⇧⌘N
Workspace... ^⌘N
Group ⌥⌘N
Group from Selection
Model.xcdatamodeld
Model.xcdatamodel
TwitterUser
Tweet
Attributes
created
identifier
text
Relationships
tweeter
TwitterUser
Attributes
name
screenName
Relationships
tweets
Outline Style
Add Entity
Add Attribute
Editor Style
CS193p
Winter 2017

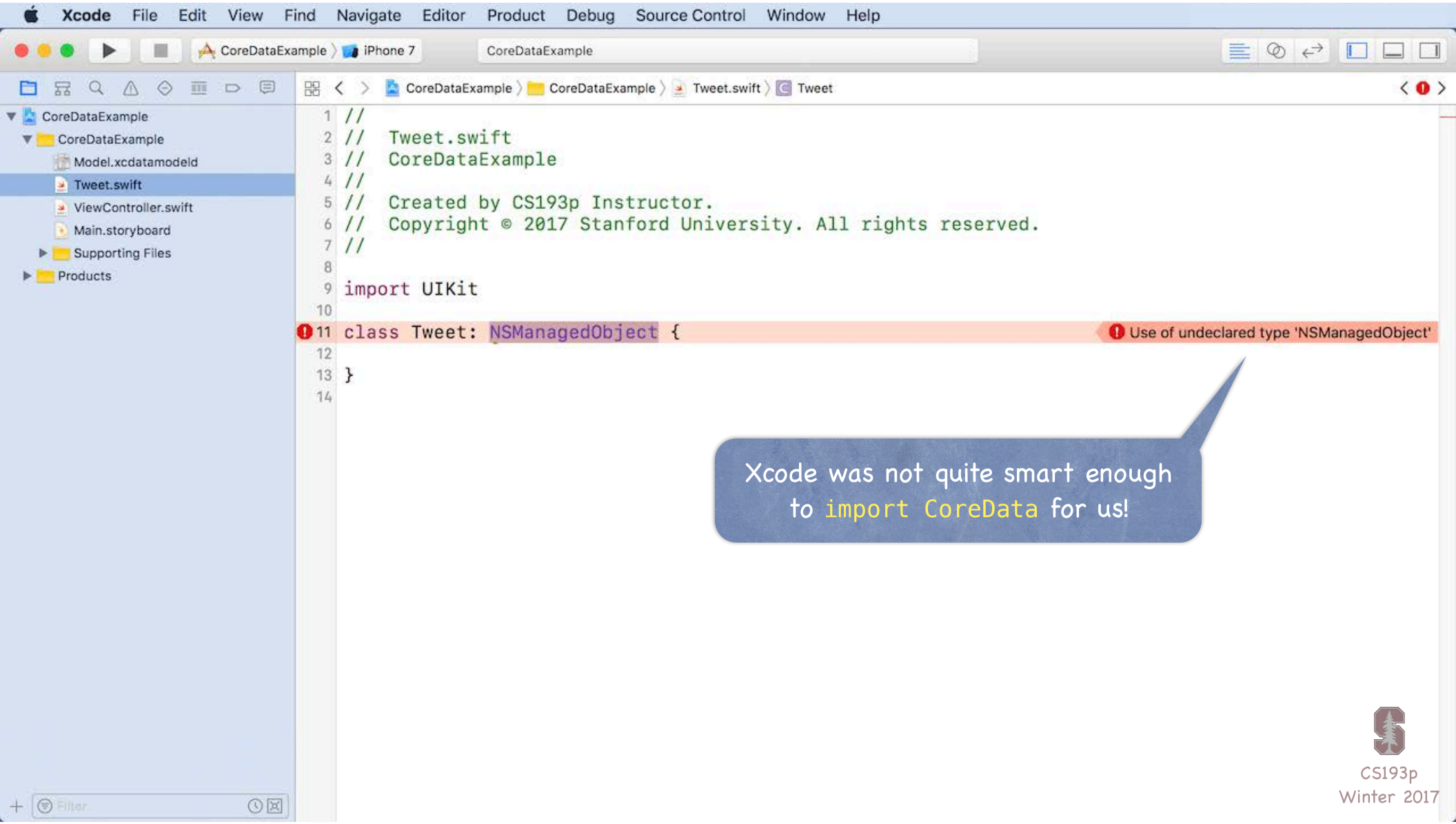


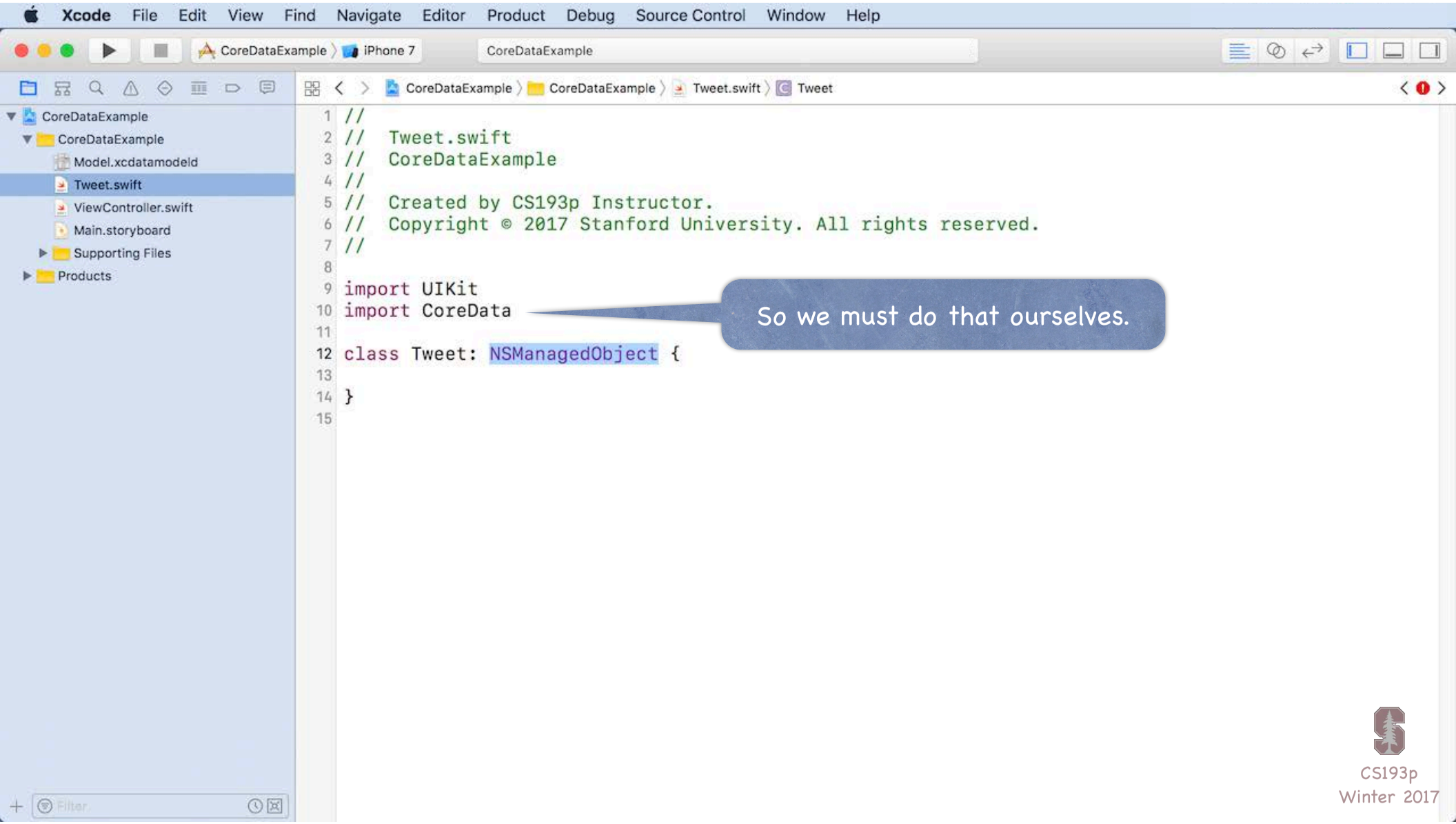

Xcode File Edit View Find Navigate Editor Product Debug Source Control Window Help
CoreDataExample iPhone 7
CoreDataExample
Choose options for your new file:
CoreDataExample
CoreDataExample
Model.xcdatamodeld
ViewController.swift
Main.storyboard
Supporting Files
Products
Class: Tweet
Subclass of: NSManagedObject
NSLock
NSMachPort
Language: NSManagedObject
NSManagedObjectContext
NSManagedObjectID
Cancel
Remember, all of our Entities in the database are represented by NSManagedObjects (or subclasses thereof which we're creating right now).
Outline Style
Add Entity
Add Attribute
Editor Style

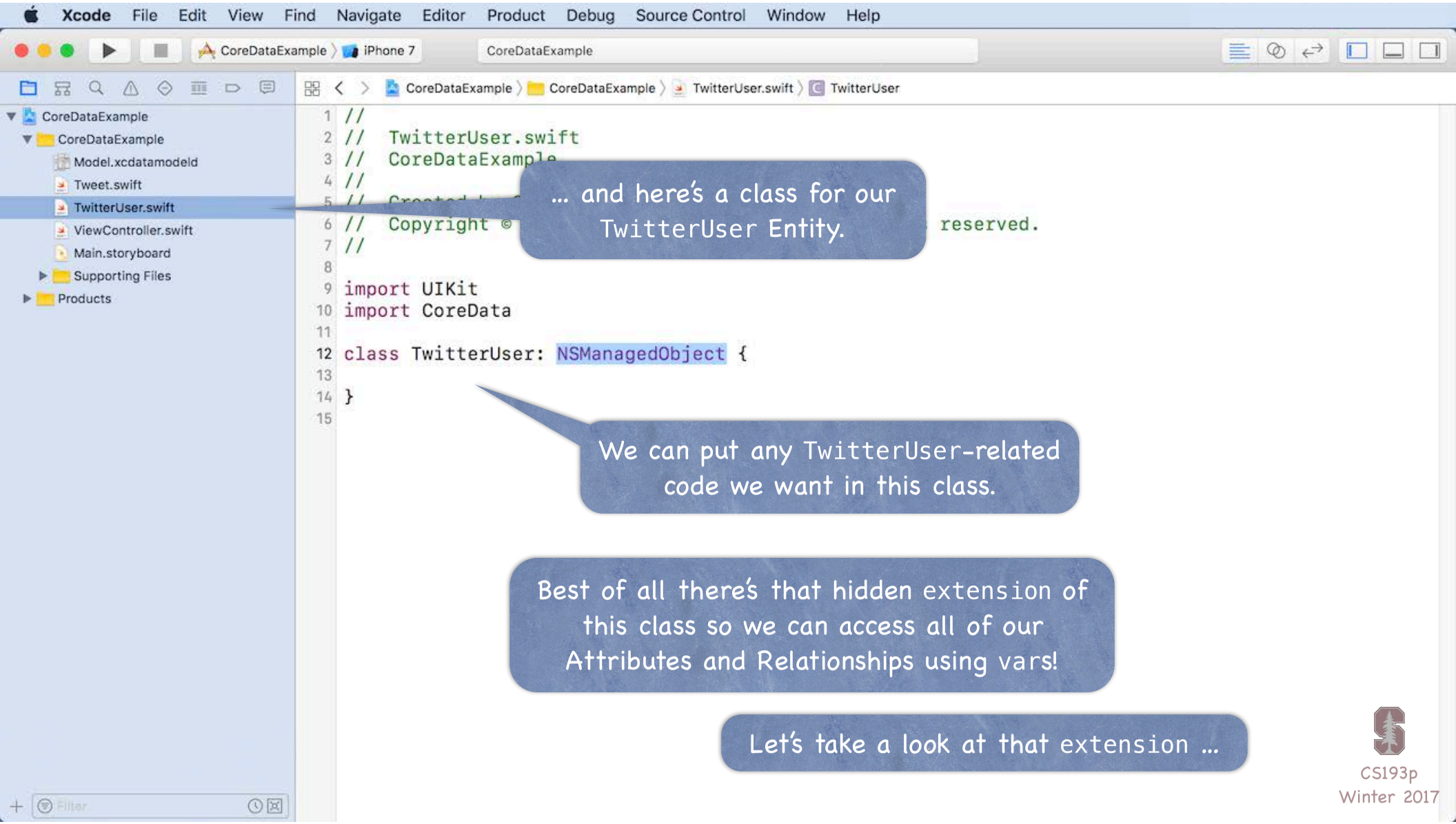

Xcode File Edit View Find Navigate Editor Product Debug Source Control Window Help
CoreDataExample iPhone 7
CoreDataExample
CoreDataExample CoreDataExample Tweet.swift Tweet
CoreDataExample
CoreDataExample
Model.xcdatamodeld
Tweet.swift
ViewController.swift
Main.storyboard
Supporting Files
Products
1 //
2 // Tweet.swift
3 // CoreDataExample
4 //
5 // Created by CS193p Instructor.
6 // Copyright © 2017 Stanford University. All rights reserved.
7 //
8
9 import UIKit
10
11 class Tweet: NSManagedObject {
12
13 }
14
We've created a subclass of NSManagedObject for our Tweet Entity.
This class should have the same name as the Entity (exactly).
It's possible to set the class name to be different than the Entity name in the Entity's inspector, but this is not recommended.

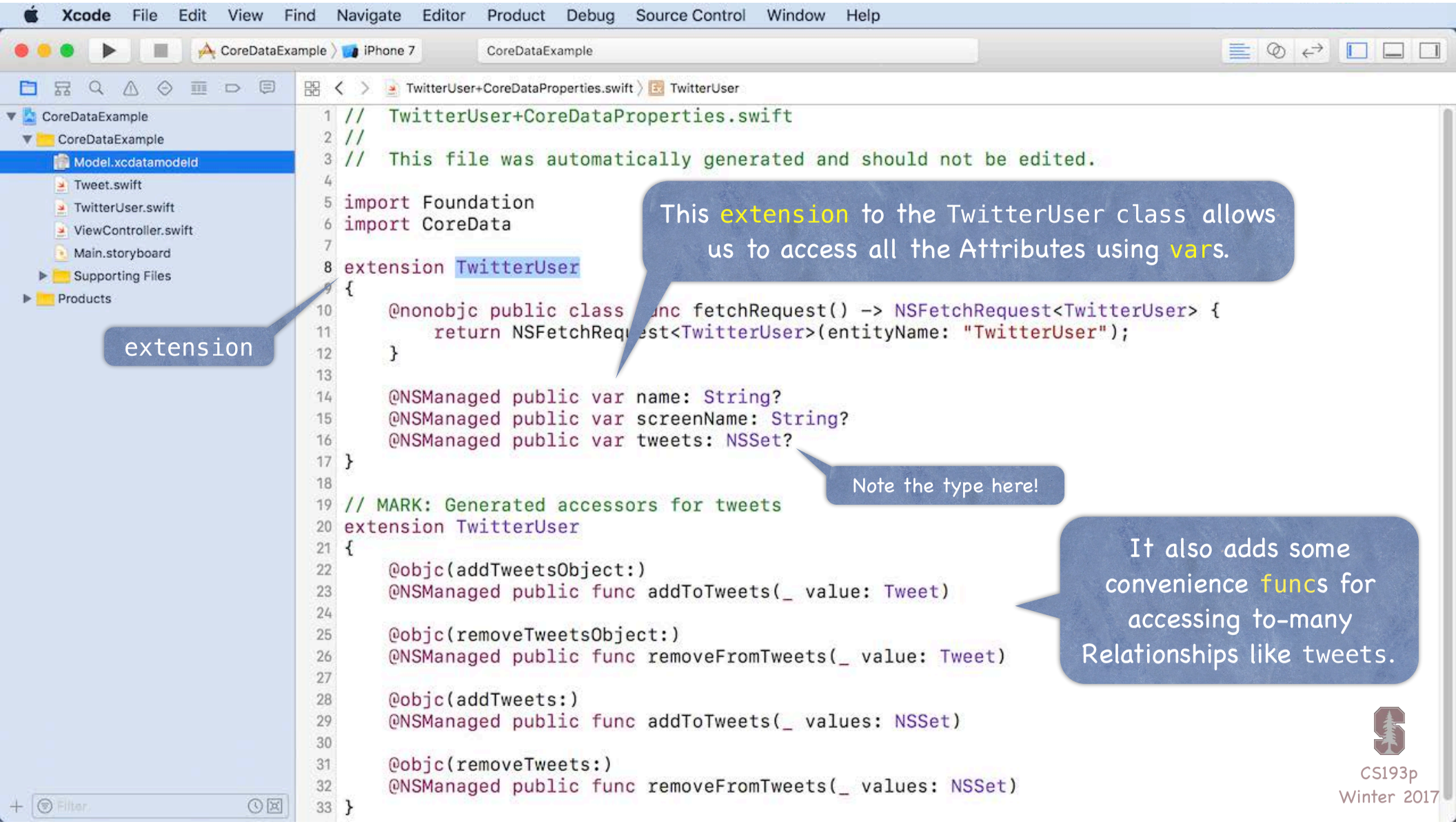


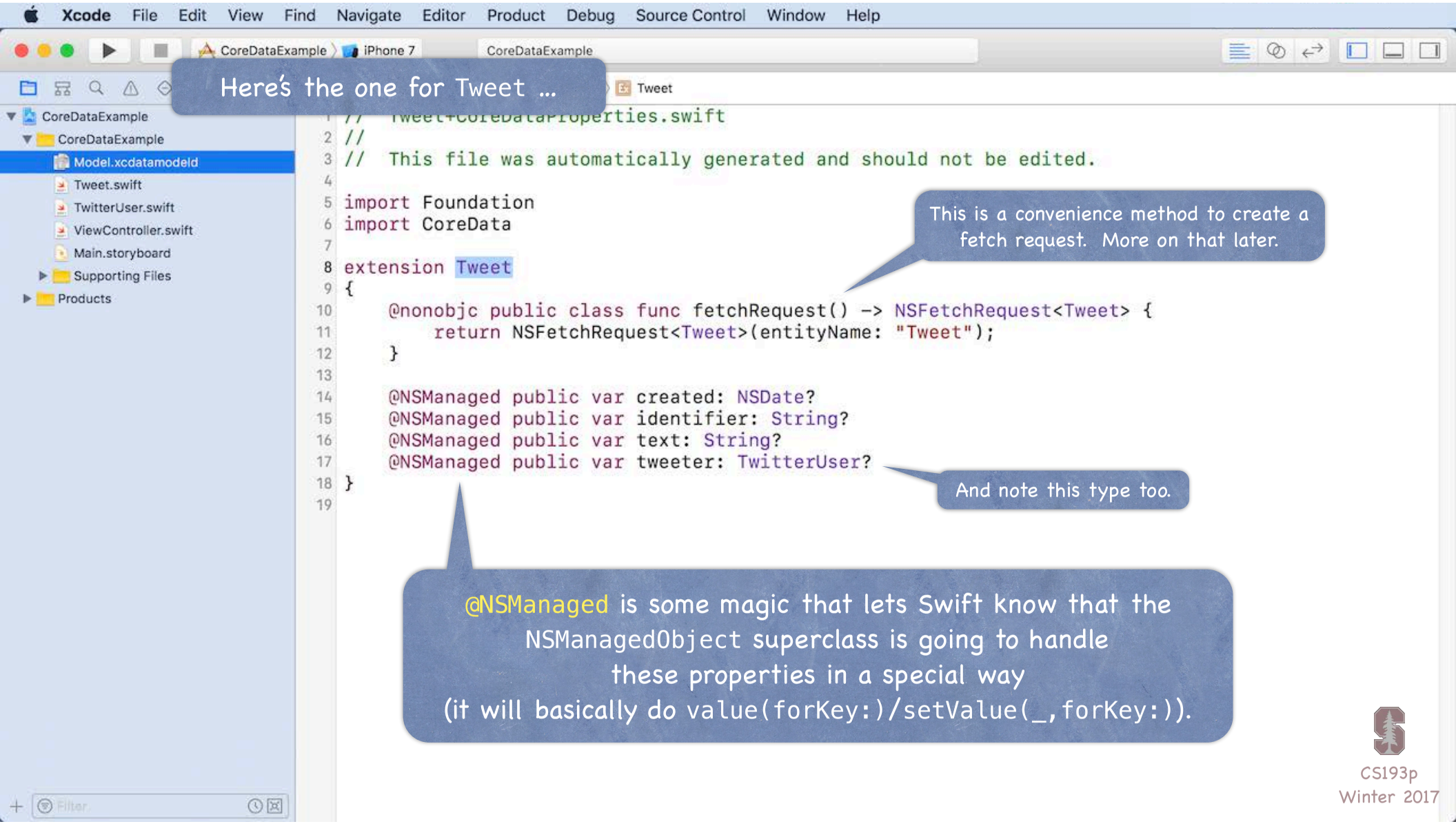
CS193p
Winter 2017


Xcode File Edit View Find Navigate Editor Product Debug Source Control Window Help
CoreDataExample 〉 iPhone 7
CoreDataExample
CoreDataExample 〉 CoreDataExample 〉 Tweet.swift 〉 Tweet
CoreDataExample
CoreDataExample
Model.xcdatamodeld
Tweet.swift
ViewController.swift
Main.storyboard
Supporting Files
Products
1 //
2 // Tweet.swift
3 // CoreDataExample
4 //
5 // Created by CS193p Instructor.
6 // Copyright © 2017 Stanford University. All rights reserved.
7 //
8
9 import UIKit
10
11 class Tweet: NSManagedObject {
12
13 }
14
Use of undeclared type 'NSManagedObject'
Xcode was not quite smart enough to import CoreData for us!


Xcode File Edit View Find Navigate Editor Product Debug Source Control Window Help
CoreDataExample 〉 iPhone 7
CoreDataExample
CoreDataExample 〉 CoreDataExample 〉 Tweet.swift 〉 Tweet
CoreDataExample
CoreDataExample
Model.xcdatamodeld
Tweet.swift
ViewController.swift
Main.storyboard
Supporting Files
Products
1 //
2 // Tweet.swift
3 // CoreDataExample
4 //
5 // Created by CS193p Instructor.
6 // Copyright © 2017 Stanford University. All rights reserved.
7 //
8
9 import UIKit
10 import CoreData
11
12 class Tweet: NSManagedObject {
13
14 }
15
So we must do that ourselves.


Xcode File Edit View Find Navigate Editor Product Debug Source Control Window Help
CoreDataExample iPhone 7
CoreDataExample
CoreDataExample CoreDataExample TwitterUser.swift TwitterUser
CoreDataExample
CoreDataExample
Model.xcdatamodeld
Tweet.swift
TwitterUser.swift
ViewController.swift
Main.storyboard
Supporting Files
Products
1 //
2 // TwitterUser.swift
3 // CoreDataExample
4 //
6 // Copyright © reserved.
7 //
9 import UIKit
10 import CoreData
12 class TwitterUser: NSManagedObject {
14 }
… and here's a class for our TwitterUser Entity.
We can put any TwitterUser-related code we want in this class.
Best of all there's that hidden extension of this class so we can access all of our Attributes and Relationships using vars!
Let's take a look at that extension …


Xcode File Edit View Find Navigate Editor Product Debug Source Control Window Help
CoreDataExample 〉 iPhone 7
CoreDataExample
TwitterUser+CoreDataProperties.swift 〉 TwitterUser
CoreDataExample
CoreDataExample
Model.xcdatamodeld
Tweet.swift
TwitterUser.swift
ViewController.swift
Main.storyboard
Supporting Files
Products
extension
// TwitterUser+CoreDataProperties.swift
//
// This file was automatically generated and should not be edited.
import Foundation
import CoreData
extension TwitterUser
{
@nonobjc public class func fetchRequest() -> NSFetchRequest<TwitterUser> {
return NSFetchRequest<TwitterUser>(entityName: "TwitterUser");
}
@NSManaged public var name: String?
@NSManaged public var screenName: String?
@NSManaged public var tweets: NSSet?
}
// MARK: Generated accessors for tweets
extension TwitterUser
{
@objc(addTweetsObject:)
@NSManaged public func addToTweets(_ value: Tweet)
@objc(removeTweetsObject:)
@NSManaged public func removeFromTweets(_ value: Tweet)
@objc(addTweets:)
@NSManaged public func addToTweets(_ values: NSSet)
@objc(removeTweets:)
@NSManaged public func removeFromTweets(_ values: NSSet)
}
This extension to the TwitterUser class allows us to access all the Attributes using vars.
Note the type here!
It also adds some convenience funcs for accessing to-many Relationships like tweets.
CS193p
Winter 2017


Xcode File Edit View Find Navigate Editor Product Debug Source Control Window Help
CoreDataExample iPhone 7
CoreDataExample
Here's the one for Tweet ...
Tweet
CoreDataExample
CoreDataExample
Model.xcdatamodeld
Tweet.swift
TwitterUser.swift
ViewController.swift
Main.storyboard
Supporting Files
Products
// Tweet+CoreDataProperties.swift
//
// This file was automatically generated and should not be edited.
import Foundation
import CoreData
extension Tweet
{
@nonobjc public class func fetchRequest() -> NSFetchRequest<Tweet> {
return NSFetchRequest<Tweet>(entityName: "Tweet");
}
@NSManaged public var created: NSDate?
@NSManaged public var identifier: String?
@NSManaged public var text: String?
@NSManaged public var tweeter: TwitterUser?
}
This is a convenience method to create a fetch request. More on that later.
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 value(forKey:)/setValue(_,forKey:)).

Core Data

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

```
// let's create an instance of the Tweet Entity in the database ...
let context = AppDelegate.viewContext
if let tweet = Tweet(context: context) {
    tweet.text = "140 characters of pure joy"
    tweet.created = Date() as NSDate
    let joe = TwitterUser(context: tweet.managedObjectContext)
    tweet.tweeter = joe
    tweet.tweeter.name = "Joe Schmo"
}
```

Note that we don't have to use that ugly `NSEntityDescription` method to create an Entity.

Core Data

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

```
// let's create an instance of the Tweet Entity in the database ...
let context = AppDelegate.viewContext
if let tweet = Tweet(context: context) {
    tweet.text = "140 characters of pure joy"
    tweet.created = Date() as NSDate
    let joe = TwitterUser(context: tweet.managedObjectContext)
    tweet.tweeter = joe
    tweet.tweeter.name = "Joe Schmo"
}
```

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

Core Data

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

```
// let's create an instance of the Tweet Entity in the database ...
let context = AppDelegate.viewContext
if let tweet = Tweet(context: context) {
    tweet.text = "140 characters of pure joy"
    tweet.created = Date() as NSDate
    let joe = TwitterUser(context: tweet.managedObjectContext)
    tweet.tweeter = joe
    tweet.tweeter.name = "Joe Schmo"
}
```

This is nicer than `setValue(Date() as NSDate, forKey: "created")`

And Swift can type-check to be sure you're actually passing an `NSDate` here (versus the value being `Any?` and thus un-type-checkable).

Core Data

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

```
// let's create an instance of the Tweet Entity in the database ...
let context = AppDelegate.viewContext
if let tweet = Tweet(context: context) {
    tweet.text = "140 characters of pure joy"
    tweet.created = Date() as NSDate
    let joe = TwitterUser(context: tweet.managedObjectContext)
    tweet.tweeter = joe
    tweet.tweeter.name = "Joe Schmo"
}
```

Setting the value of a Relationship is no different than setting any other Attribute value.

And this will automatically add this `tweet` to `joe`'s `tweets` Relationship too!

```
if let joesTweets = joe.tweets as? Set<Tweet> {      // joe.tweets is an NSSet, thus as
    if joesTweets.contains(tweet) { print("yes!") } // yes!
}
```

Core Data

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

```
// let's create an instance of the Tweet Entity in the database ...
let context = AppDelegate.viewContext
if let tweet = Tweet(context: context) {
    tweet.text = "140 characters of pure joy"
    tweet.created = Date() as NSDate
    let joe = TwitterUser(context: tweet.managedObjectContext)
    tweet.tweeter = joe is the same as joe.addToTweets(tweet)
    tweet.tweeter.name = "Joe Schmo"
}
```

Xcode also generates some convenience functions for "to-many" relationships.
For example, for `TwitterUser`, it creates an `addToTweets(Tweet)` function.
You can use this to add a `Tweet` to a `TwitterUser`'s `tweets` Relationship.

Core Data

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

```
// let's create an instance of the Tweet Entity in the database ...
let context = AppDelegate.viewContext
if let tweet = Tweet(context: context) {
    tweet.text = "140 characters of pure joy"
    tweet.created = Date() as NSDate
    let joe = TwitterUser(context: tweet.managedObjectContext)
    joe.addToTweets(tweet)
    tweet.tweeter.name = "Joe Schmo"
}
```

Every NSManagedObject knows the managedObjectContext it is in.

So we could use that fact to create this TwitterUser in the same context as the tweet is in.

Of course, we could have also just used context here.

Core Data

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

```
// let's create an instance of the Tweet Entity in the database ...
let context = AppDelegate.viewContext
if let tweet = Tweet(context: context) {
    tweet.text = "140 characters of pure joy"
    tweet.created = Date() as NSDate
    let joe = TwitterUser(context: tweet.managedObjectContext)
    joe.addToTweets(tweet)
    tweet.tweeter.name = "Joe Schmo"
}
```

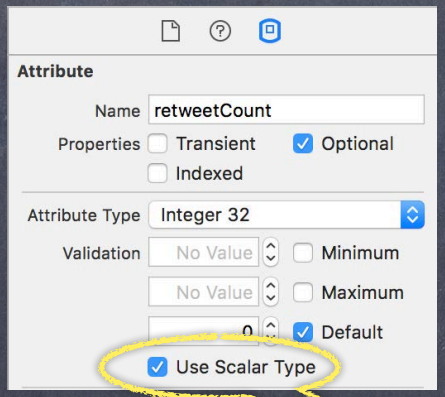
Relationships can be traversed using "dot notation."

`tweet.tweeter` is a `TwitterUser`, so `tweet.tweeter.name` is the `TwitterUser`'s name.

This is much nicer that `value(forKeyPath:)` because it is type-checked at every level.

Scalar Types

- Scalars
  - By default Attributes come through as objects (e.g. `NSNumber`)
  - If you want as normal Swift types (e.g. `Int32`), inspect them in the Data Model and say so



This will usually be the default for numeric values.

Deletion

- Deletion

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

  ```
  managedObjectContext.delete(_ object: tweet)
  ```

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

  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()
  {
      // if this method were in the Tweet class
      // we wouldn't have to remove ourselves from tweeter.tweets (that happens automatically)
      // but if TwitterUser had, for example, a "number of retweets" attribute,
      // and if this Tweet were a retweet
      // then we might adjust it down by one here (e.g. tweeter.retweetCount -= 1).
  }
  ```

Querying

- So far you can …
  - Create objects in the database: `NSEntityDescription` or `Tweet(context: …)`
  - Get/set properties with `value(forKey:)/setValue(_,forKey:)` or vars in a custom subclass.
  - Delete objects using the `NSManagedObjectContext` `delete()` 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`.
- Three important things involved in creating an `NSFetchRequest`
  1. `Entity` to fetch (required)
  2. `NSSortDescriptor`s to specify the order in which the `Array` of fetched objects are returned
  3. `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<Tweet> = Tweet.fetchRequest()
request.sortDescriptors = [sortDescriptor1, sortDescriptor2, …]
request.predicate = ...
```

Querying

- Specifying the kind of Entity we want to fetch

```
let request: NSFetchRequest<Tweet> = Tweet.fetchRequest()
```

(note this is a rare circumstance where Swift cannot infer the type)

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).
`NSFetchRequest` is a generic type so that the `Array<Tweet>` that is fetched can also be typed.

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(_:)) // can skip this
)
```

  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).
  It is usually just compare:, but for NSString there are other options (see documentation).
  It also has to be exposed to the Objective-C runtime (thus NSString, not String).
  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, then, within that sort, by another.
  Example: [lastNameSortDescriptor, firstNameSortDescriptor]

Querying

- NSPredicate

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

  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 predicates = [predicate1, predicate2]
let andPredicate = NSCompoundPredicate(andPredicateWithSubpredicates: predicates)
```

  - This andPredicate is "predicate1 AND predicate2". OR available too, of course.
- Function Predicates
  - 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.

Querying

- Putting it all together

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

```
let request: NSFetchRequest<TwitterUser> = TwitterUser.fetchRequest()
```

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

```
let yesterday = Date(timeIntervalSinceNow:-24*60*60) as NSDate
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

```
let context = AppDelegate.viewContext
let recentTweeters = try? context.fetch(request)
```

The `try?` 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` method ...
Returns an empty `Array` (not `nil`) if it succeeds and there are no matches in the database.
Returns an `Array` of `NSManagedObject`s (or subclasses thereof) if there were any matches.

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 recentTweeters {
    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 has already fetched them.
  - But if you did this …

```
for user in recentTweeters {
    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.
  - Often we use only the main queue and its AppDelegate.viewContext, so it's not an issue.
- Thread-Safe Access to an NSManagedObjectContext

```
context.performBlock {  // or performBlockAndWait until it finishes
    // do stuff with context (this will happen in its safe Q (the Q 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 Core Data code using this.
  - Although if you have no multithreaded code at all in your app, you can probably skip it.
  - It won't cost anything if it's not in a multithreaded situation.

Core Data Thread Safety

- Convenient way to do database stuff in the background

  The `persistentContainer` has a simple method for doing database stuff in the background

```
AppDelegate.persistentContainer.performBackgroundTask { context in
    // do some CoreData stuff using the passed-in context
    // this closure is not the main queue, so don't do UI stuff here (dispatch back if needed)
    // and don't use AppDelegate.viewContext here, use the passed context
    // you don't have to use NSManagedObjectContext's perform method here either
    // since you're implicitly doing this block on that passed context's thread
    try? context.save() // don't forget this (and catch errors if needed)
}
```

  This would generally only be needed if you're doing a big update.

  You'd want to see that some Core Data update is a performance problem in Instruments first.

  For small queries and small updates, doing it on the main queue is fine.

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 an NSFRC to answer all of your UITableViewDataSource protocol's questions!

- Implementation of UITableViewDataSource …

```
var fetchedResultsController = NSFetchedResultsController… // more on this in a moment

func numberOfSectionsInTableView(sender: UITableView) -> Int {
    return fetchedResultsController?.sections?.count ?? 1
}

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

NSFetchedResultsController

- Implementing tableView(_, cellForRowAt indexPath:)

  What about cellForRowAt?

  You'll need this important NSFetchedResultsController method …

```
func object(at indexPath: NSIndexPath) -> NSManagedObject
```

  Here's how you would use it …

```
func tableView(_ tv: UITableView, cellForRowAt indexPath: NSIndexPath) -> UITableViewCell
{
    let cell = tv.dequeue…
    if let obj = fetchedResultsController.object(at: indexPath) {
        // load up the cell based on the properties of the obj
        // obj will be an NSManagedObject (or subclass thereof) that fetches into this row
    }
    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<Tweet> = Tweet.fetchRequest()
request.sortDescriptors = [NSSortDescriptor(key: “created” ...)]
request.predicate = NSPredicate(format: “tweeter.name = %@”, theName)
let frc = NSFetchedResultsController<Tweet>(        // note this is a generic type
            fetchRequest: request,
    managedObjectContext: context,
      sectionNameKeyPath: keyThatSaysWhichAttributeIsTheSectionName,
                cacheName: “MyTwitterQueryCache”)   // careful!
```

  Be sure that any cacheName you use is always associated with exactly the same request.

  It's okay to specify nil for the cacheName (no cacheing of fetch results in that case).

  It is critical that the sortDescriptor matches up with the keyThatSaysWhichAttribute…

  The results must sort such that all objects in the first section come first, second second, etc.

  If keyThatSaysWhichAttributeIsTheSectionName is nil, your table will be one big section.

NSFetchedResultsController

- NSFetchedResultsController also "watches" Core Data
  - And automatically will notify your UITableView if something changes that might affect it!
  - When it notices a change, it sends message like this to its delegate …

```
func controller(NSFetchedResultsController,
                didChange: Any,
                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 …
}
```

- FetchedResultsTableViewController
  - Our demo today (and Assignment 5) will include a class FetchedResultsTableViewController
  - If you make your controller be a subclass of it, you'll get the "watching" code for free

Core Data and UITableView

- Things to remember to do …

  1. Subclass `FetchedResultsTableViewController` to get `NSFetchedResultsControllerDelegate`
  2. Add a `var` called `fetchedResultsController` initialized with the `NSFetchRequest` you want
  3. Implement your `UITableViewDataSource` methods using this `fetchedResultsController` `var`

  You can get the code for #3 from the slides of this presentation (or from the demo).

- Then …

  After you set the value of your `fetchedResultsController` …

  ```
  try? fetchedResultsController?.performFetch() // would be better to catch errors!
  tableView.reloadData()
  ```

  Your table view should then be off and running and tracking changes in the database!

  To get those changes to appear in your table, set yourself as the NSFRC's delegate:

  ```
  fetchedResultsController?.delegate = self
  ```

  This will work if you inherit from `FetchedResultsTableViewController`.