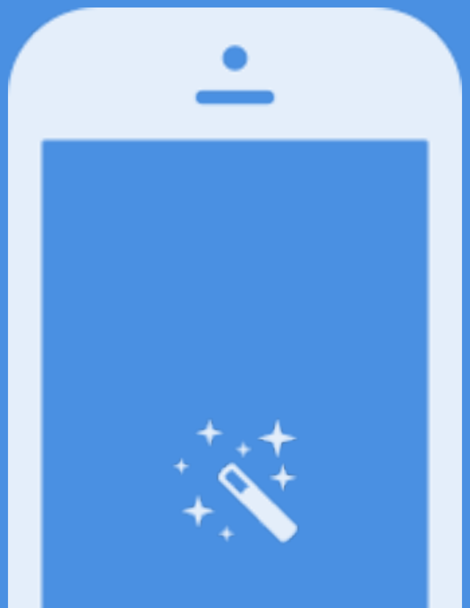
iOS Crash Course

Session Eight

Janum Trivedi

iOSCrashCour.se



Session 7

- Session 7 on Tuesday was just a review session
- If you didn’t attend, we didn’t introduce any new material

Follow Up

- No office hours this weekend (I'll be mentoring at HackFSU)
- Office hours will start next week, however
- Questions?

Session 8 Overview

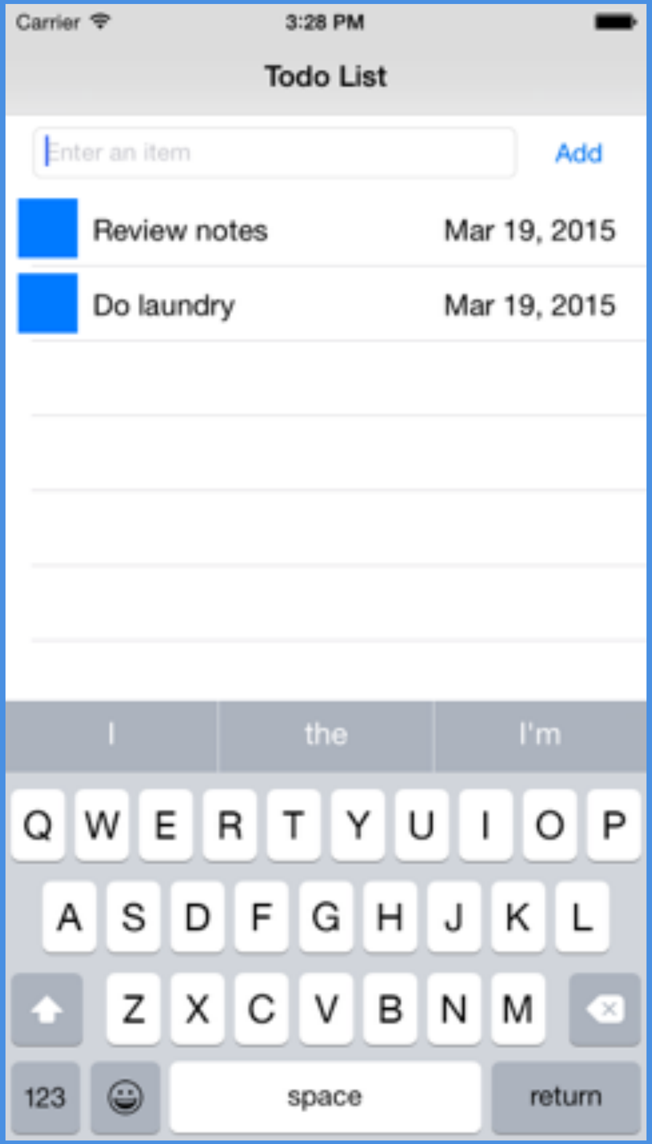
- Today we'll be learning **persistence**
- New concept + we will implement it to our todo application
- Please download these slides **and** the latest Xcode project from **Session 5's GitHub** if you do not have a working/up-to-date project

Persistence

- So far, we've learned:
  - How to **model** our data (like subclassing NSObject to create our Item class)
  - How to **manipulate** our data (setting the properties of our objects)
  - How to **show** the data (using UIKit, UITableView, etc.)

Persistence

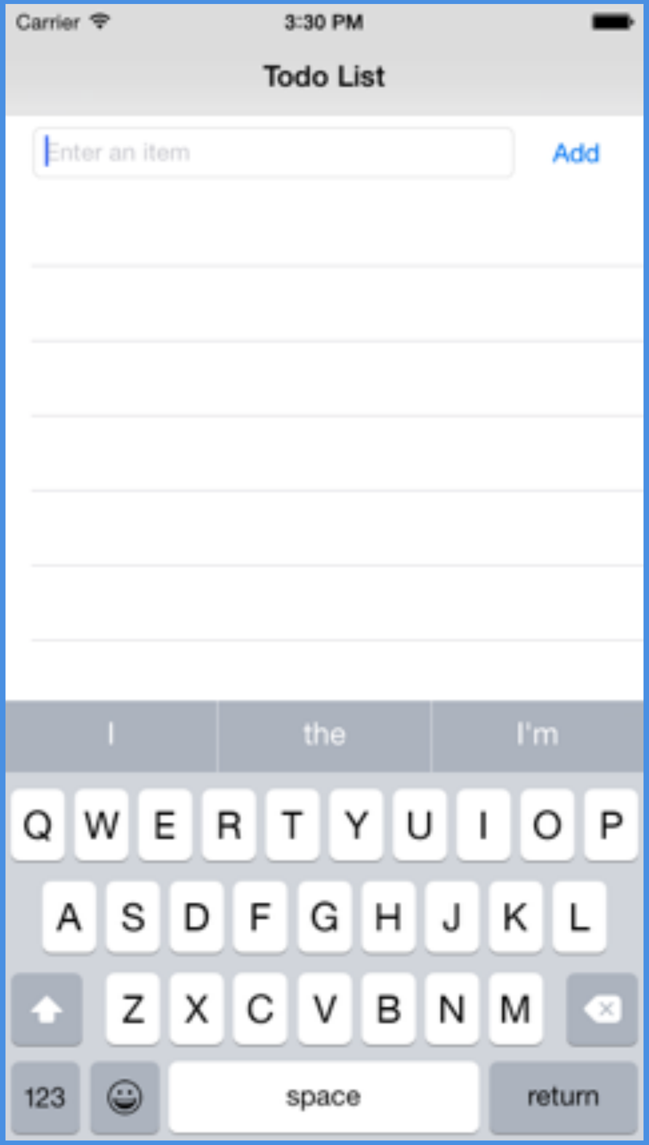
- In our todo app, we can launch the app and add some items



Review notes	Mar 19, 2015
Do laundry	Mar 19, 2015

Persistence

- But as soon as we close the app and re-launch it, our items disappeared
- As if none of the changes we made, or data we created, were stored, or *persisted*



What is persistence?

- Persistence is being able to save our data across multiple sessions (i.e., not lose our data upon exit)

Automatic persistence?

- Why doesn’t our app persist data automatically or by default?
  - 1. When we launch our app, create a new Item object, and add it to our items array, those variables are stored only **in memory**
    - When the app exits, all the memory it “owned” and used is cleared
    - All the data kept in memory is thus lost

Automatic persistence?

- Why doesn’t our app persist data automatically or by default?
  - 2. Our app/compiler doesn’t know **what** to persist, much less **how** to do it
    - We must be explicit

Why?

- Why do we need to support persistence in our apps?
- Imagine if every time you re-launched an app, it kept none of your data
- Persistence helps our apps maintain a user state
    - ex., when I log into an app, it **remembers** my information

Persistence

- The idea behind persistence is that at various points in our application’s lifecycle, we can **take important data** (that’s currently stored in memory), and **write it out to the filesystem**
    - i.e., save to the disk (technically, flash storage on iOS devices)
- We can write-out right when a change is made, or immediately before the app exits

Persistence

- Then, when the app re-launches, it can retrieve data from that file and restore it
- Much like saving a document on our computer
    - ex., if we don't "Save As…" to some folder, we can't re-open the document if the application suddenly quits

Persistence Methods

- i.e., ways to persist
- There are a few ways of adding persistence:
  - NSUserDefaults
  - NSCoding/NSKeyedArchiver
  - SQLite
  - Core Data

Persistence Methods

- i.e., ways to persist
- There are a few ways of adding persistence:
  - **NSUserDefaults**
  - **NSCoding/NSKeyedArchiver**
  - SQLite
  - Core Data

NSUserDefaults

- Most basic, primitive type of persistence
- NSUserDefaults is a Foundation class that lets you store and retrieve values from a **property list** (.plist file)
- Similar structure to a dictionary or hash table
    - Who has heard of either of those?

What is a dictionary?

- In Objective-C, a dictionary is a **data structure** that has a list of **key-value pairs**
  - i.e., every object we want to store in a dictionary has a unique "key" identifier
  - Maps unique keys to values

What is a dictionary?

- In Objective-C, a dictionary is a **data structure** that has a list of **key-value pairs**
    - i.e., every object we want to store has a unique "key" identifier
    - Maps keys to values

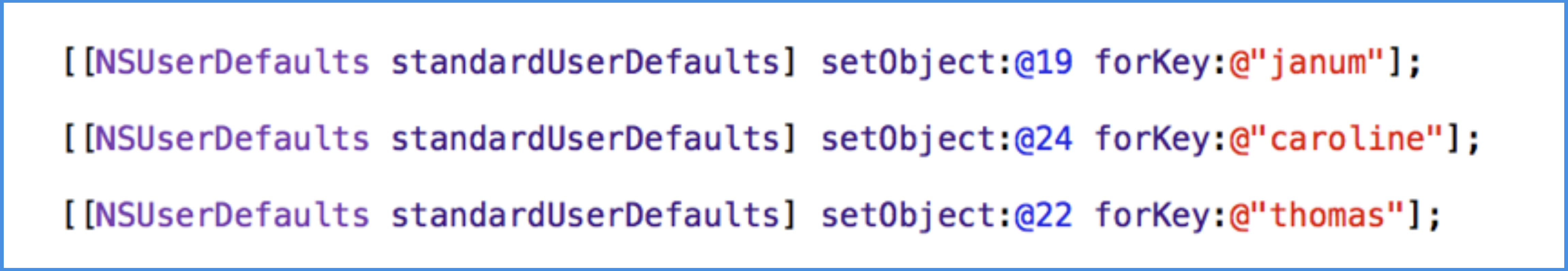
Key	**Value**
`"janum"` →	19
`"caroline"` →	24
`"thomas"` →	22

What is a property list?

- A **property list** (which is what NSUserDefaults writes to) is like a dictionary
- It's simply a file that has a list of key/value pairs
- NSUserDefaults let's us access a property list file that exists for **our app only**
  - In iOS, apps are **sandboxed**, which means your app cannot access the data or files of another app
  - An app can only access **its own** data

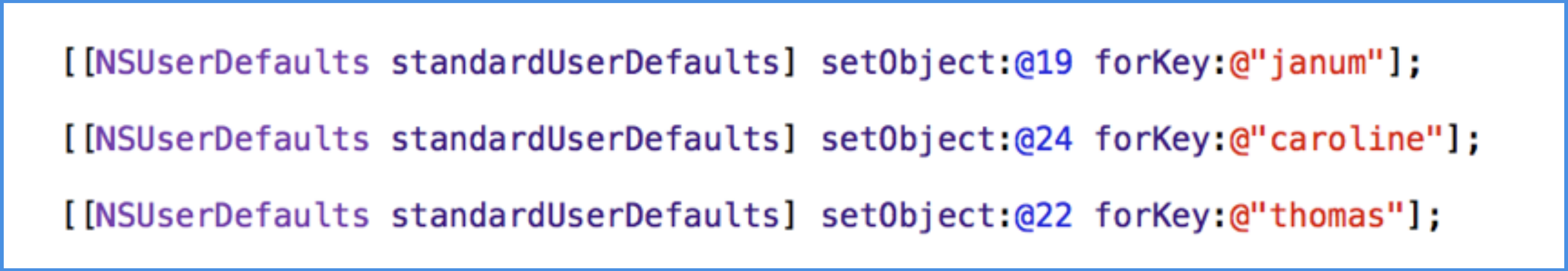
What is a property list?

- We can tell NSUserDefaults to store a key-value pair in that file
- How?



```
[[NSUserDefaults standardUserDefaults] setObject:@19 forKey:@"janum"];
[[NSUserDefaults standardUserDefaults] setObject:@24 forKey:@"caroline"];
[[NSUserDefaults standardUserDefaults] setObject:@22 forKey:@"thomas"];
```

How to store an object

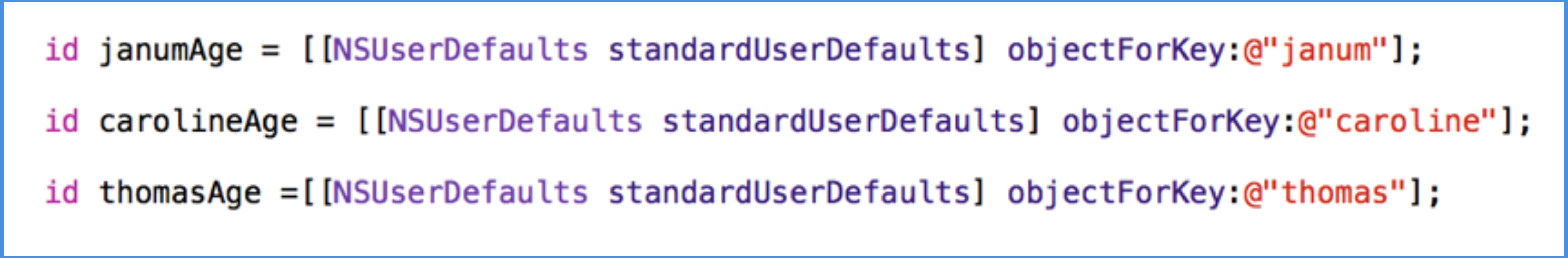


```
[[NSUserDefaults standardUserDefaults] setObject:@19 forKey:@"janum"];
[[NSUserDefaults standardUserDefaults] setObject:@24 forKey:@"caroline"];
[[NSUserDefaults standardUserDefaults] setObject:@22 forKey:@"thomas"];
```

- I’m calling the **`setObject:`** method on the **`NSUserDefaults`** object that **`[NSUserDefaults standardUserDefaults]`** returns

- I’m giving it an **`NSString`** key, and some object of type **`id`** (can be an NSString, NSDate, NSNumber, NSArray, etc.)

How to retrieve an object



```
id janumAge = [[NSUserDefaults standardUserDefaults] objectForKey:@"janum"];

id carolineAge = [[NSUserDefaults standardUserDefaults] objectForKey:@"caroline"];

id thomasAge =[[NSUserDefaults standardUserDefaults] objectForKey:@"thomas"];
```

- “Ask” for the value that corresponds to some key using **`objectForKey:`**
- **Core concept: once we store a value, we can access it later, even if the app exits or is killed**
- It will **persist** between sessions in the filesystem

Limitations of NSUserDefaults

- NSUserDefaults can only store a handful of object types by default (primary Foundation types)
    - NSString, NSNumber, NSDate, NSArray, NSDictionary, etc.
    - Can only store objects that can be represented in a property list
    - For unsupported types, you have to save the object as binary data using NSData

Limitations of NSUserDefaults

- Secondly, NSUserDefaults is *very slow*
- Fine for storing some occasional, small pieces of data
- But, you would quickly run into performance problems if you tried storing an NSArray with 1000 NSStrings in it

Then what good is it?

- If NSUserDefaults is both inflexible and slow at scale, then why have it?
- It’s good for storing things like user preferences that you want your app to remember
  - i.e., that data is **small** and **few in number**
  - Things like settings don’t really belong in a traditional object-graph

Questions?

NSCoding/NSKeyedArchiver

- Let’s move on to something more robust: NSKeyedArchiver
- Again, similar key-value store design
- We give it an object and key, and NSKeyedArchiver will archive it to the disk
- Slightly more complex, but faster and more flexible than NSUserDefaults

NSCoding/NSKeyedArchiver

- If we want to **serialize** an object (such that we can store it to disk), **the object has to conform to the NSCoding protocol**

- Recall: When we want to implement a UITableView, our view controller must conform to UITableViewDelegate/UITableViewDataSource

- Means the class must **implement any required methods** defined by the delegate/protocol

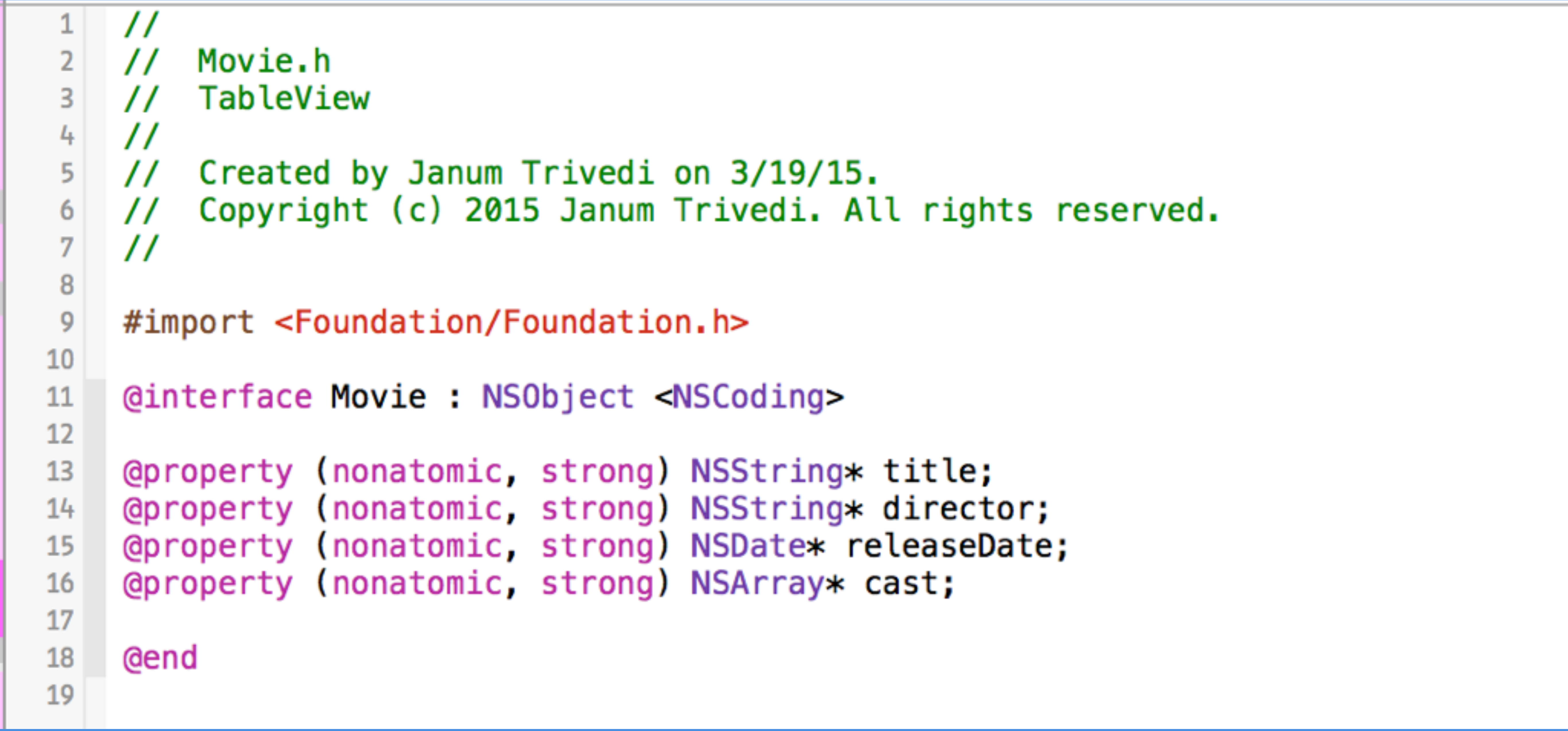
Conforming to NSCoding

- The NSCoding methods tell NSKeyedArchiver *how* to serialize an object (especially if it is a custom object, like **Item**)
- The class must implement 2 functions: -initWithCoder: and -encodeWithCoder

Conforming to NSCoding

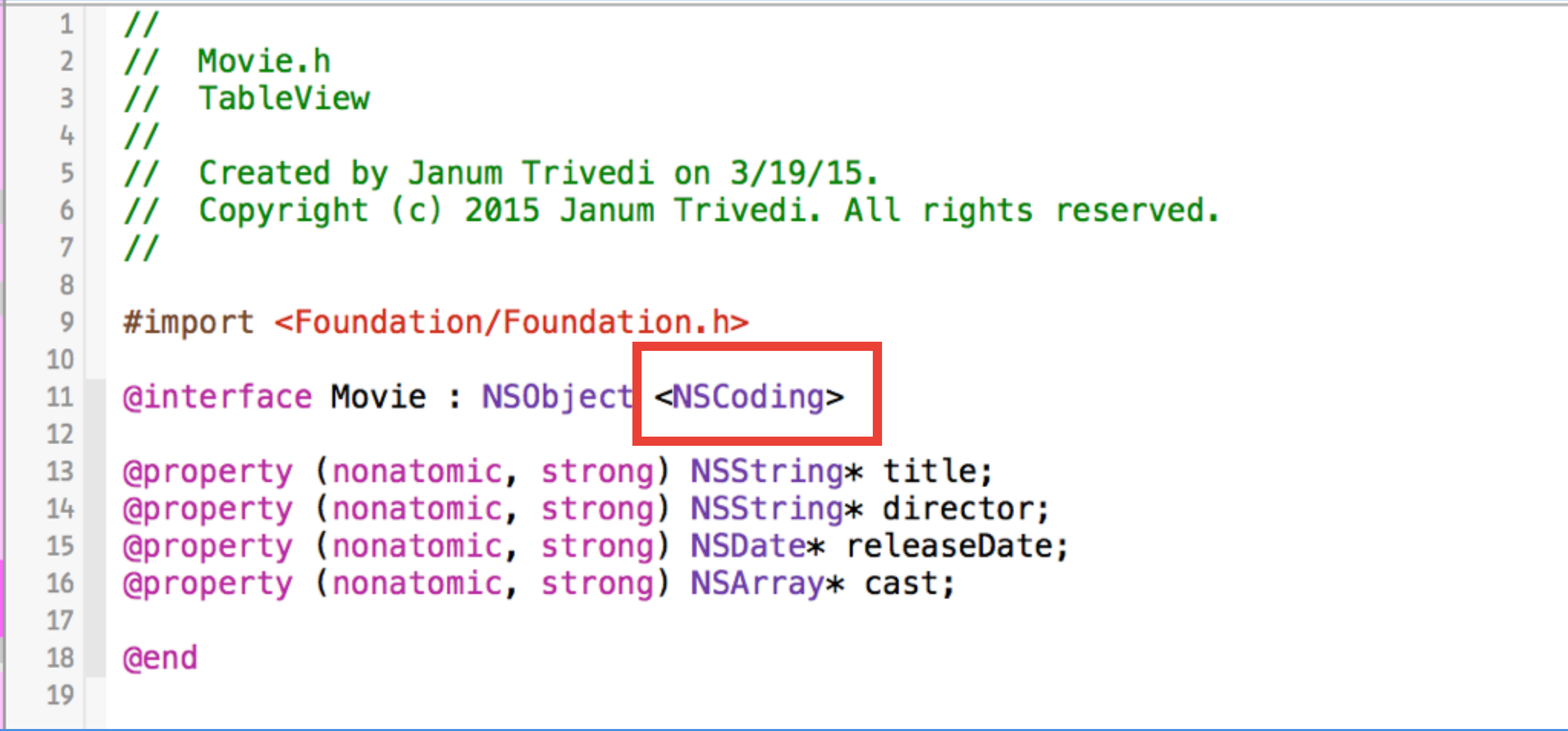
- Say I have a class called Movie (subclass of NSObject)
- Because it is a custom class, I need to implement the 2 NSCoding methods for Movie objects to be serializable
- Movie is a **composite** type (a type that contains properties of different types)
- The code is mostly boilerplate!

Movie.h

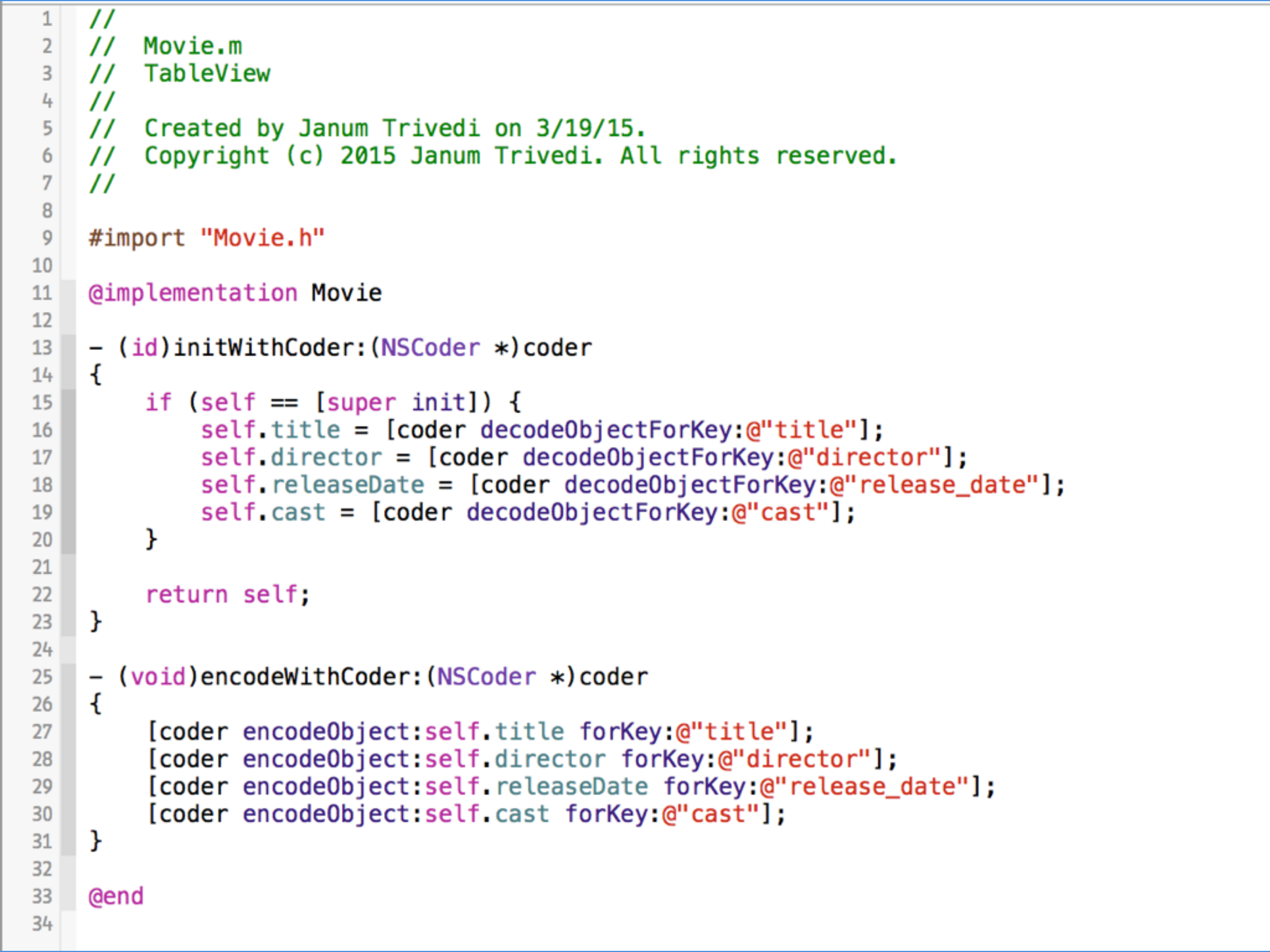


```
//
//  Movie.h
//  TableView
//
//  Created by Janum Trivedi on 3/19/15.
//  Copyright (c) 2015 Janum Trivedi. All rights reserved.
//

#import <Foundation/Foundation.h>

@interface Movie : NSObject <NSCoding>

@property (nonatomic, strong) NSString* title;
@property (nonatomic, strong) NSString* director;
@property (nonatomic, strong) NSDate* releaseDate;
@property (nonatomic, strong) NSArray* cast;

@end

```

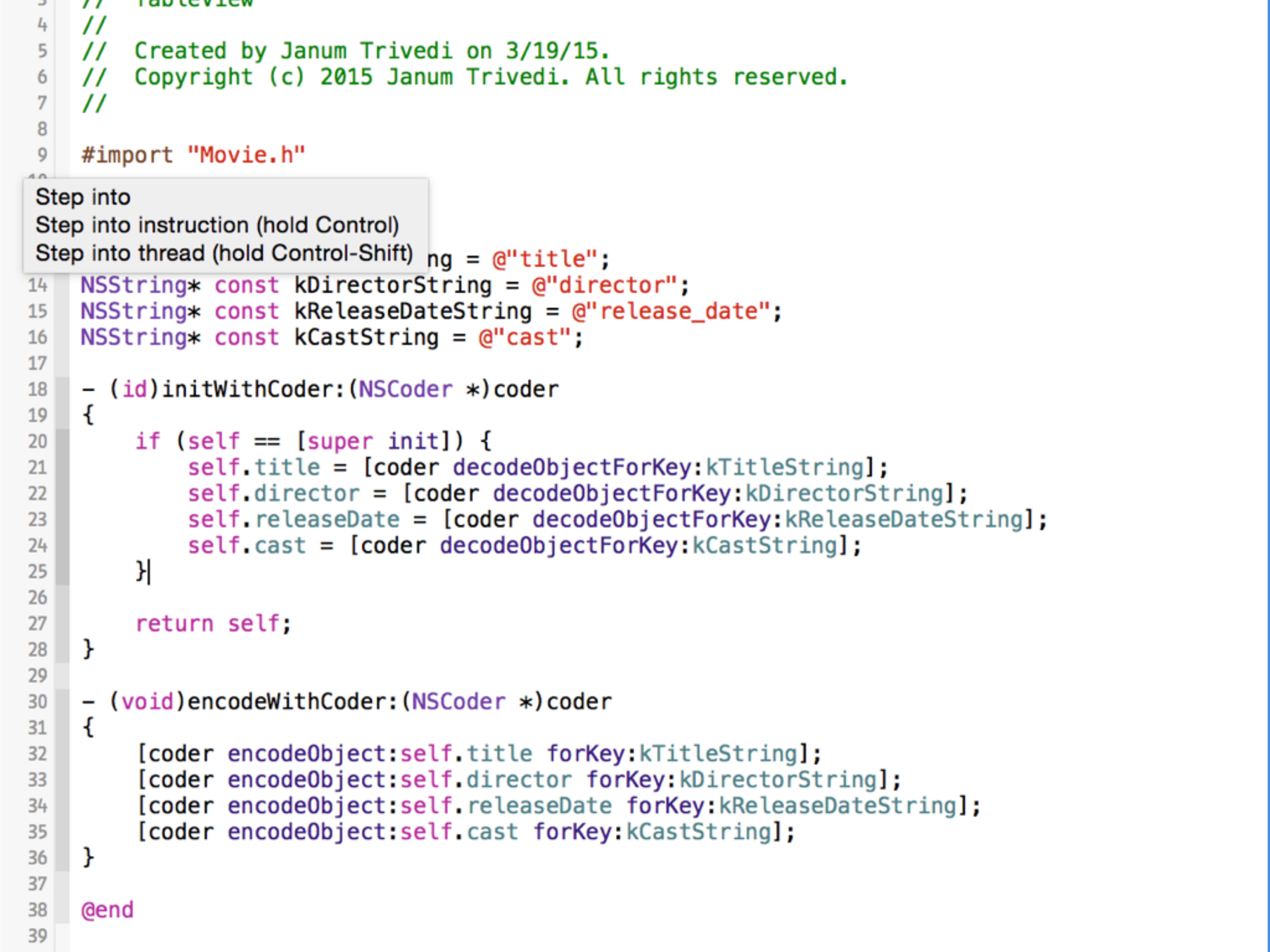
Movie.h



```
//
//  Movie.h
//  TableView
//
//  Created by Janum Trivedi on 3/19/15.
//  Copyright (c) 2015 Janum Trivedi. All rights reserved.
//

#import <Foundation/Foundation.h>

@interface Movie : NSObject <NSCoding>

@property (nonatomic, strong) NSString* title;
@property (nonatomic, strong) NSString* director;
@property (nonatomic, strong) NSDate* releaseDate;
@property (nonatomic, strong) NSArray* cast;

@end
```



```
//
//  Movie.m
//  TableView
//
//  Created by Janum Trivedi on 3/19/15.
//  Copyright (c) 2015 Janum Trivedi. All rights reserved.
//

#import "Movie.h"

@implementation Movie

- (id)initWithCoder:(NSCoder *)coder
{
    if (self == [super init]) {
        self.title = [coder decodeObjectForKey:@"title"];
        self.director = [coder decodeObjectForKey:@"director"];
        self.releaseDate = [coder decodeObjectForKey:@"release_date"];
        self.cast = [coder decodeObjectForKey:@"cast"];
    }

    return self;
}

- (void)encodeWithCoder:(NSCoder *)coder
{
    [coder encodeObject:self.title forKey:@"title"];
    [coder encodeObject:self.director forKey:@"director"];
    [coder encodeObject:self.releaseDate forKey:@"release_date"];
    [coder encodeObject:self.cast forKey:@"cast"];
}

@end

```


Step into
Step into instruction (hold Control)
Step into thread (hold Control-Shift)

```
//  Tableview
//
//  Created by Janum Trivedi on 3/19/15.
//  Copyright (c) 2015 Janum Trivedi. All rights reserved.
//

#import "Movie.h"

                             ng = @"title";
NSString* const kDirectorString = @"director";
NSString* const kReleaseDateString = @"release_date";
NSString* const kCastString = @"cast";

- (id)initWithCoder:(NSCoder *)coder
{
    if (self == [super init]) {
        self.title = [coder decodeObjectForKey:kTitleString];
        self.director = [coder decodeObjectForKey:kDirectorString];
        self.releaseDate = [coder decodeObjectForKey:kReleaseDateString];
        self.cast = [coder decodeObjectForKey:kCastString];
    }

    return self;
}

- (void)encodeWithCoder:(NSCoder *)coder
{
    [coder encodeObject:self.title forKey:kTitleString];
    [coder encodeObject:self.director forKey:kDirectorString];
    [coder encodeObject:self.releaseDate forKey:kReleaseDateString];
    [coder encodeObject:self.cast forKey:kCastString];
}

@end
```

Serialization

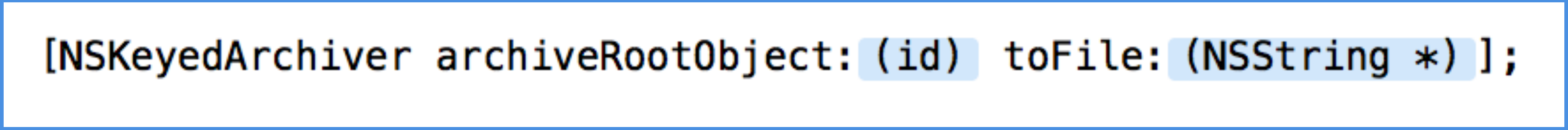
- We learned how to implement the required NSCoding methods, but how do we actually *tell* iOS to *do* the serialization?

- NSKeyedArchiver is the class that handles serialization/deserialization

- Has a method called:

- **`archiveRootObject:(id) toFile:(NSString *)path`**

Serialization



```
[NSKeyedArchiver archiveRootObject:(id) toFile:(NSString *)];
```

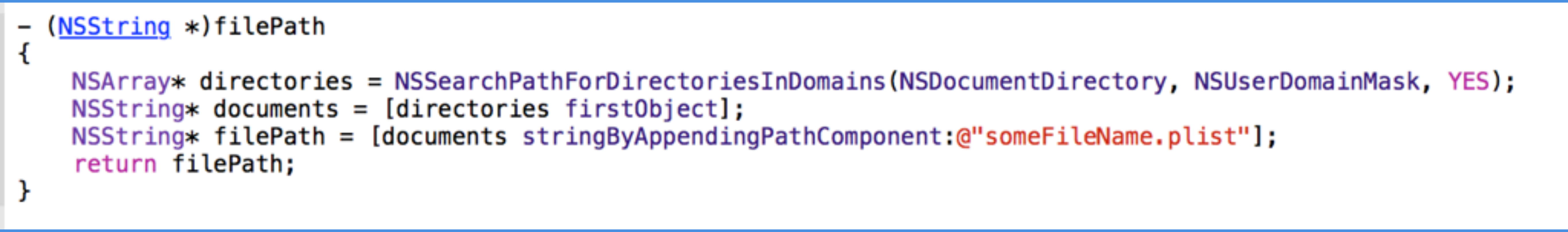
- Accepts 2 parameters: the object to serialize, and a file path to store it at
- But, we don't want to hardcode the entire path- instead, we want iOS to dynamically create the path at runtime
- Then, just append the file name to the path

Creating the file path

```
[NSKeyedArchiver archiveRootObject:(id) toFile:(NSString *)];
```

- Every app has a documents folder that only that app can access
- We want the path to that specific documents folder
- Just copy and use the filePath function in the next slide (don't roll your own)

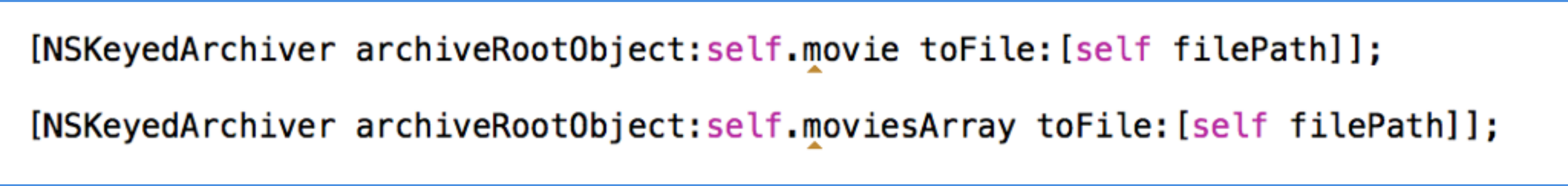
filePath function



```
- (NSString *)filePath
{
    NSArray* directories = NSSearchPathForDirectoriesInDomains(NSDocumentDirectory, NSUserDomainMask, YES);
    NSString* documents = [directories firstObject];
    NSString* filePath = [documents stringByAppendingPathComponent:@"someFileName.plist"];
    return filePath;
}
```

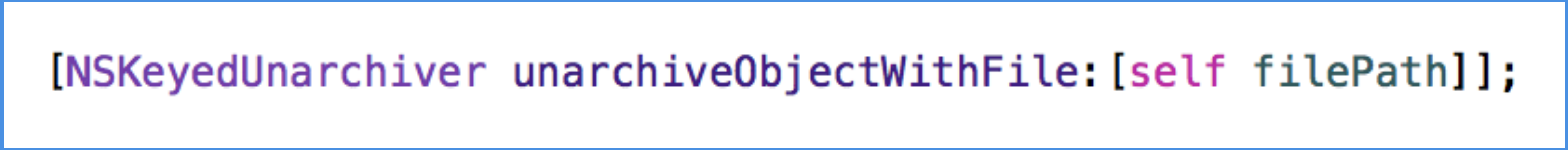
Serializing/Deserializing

- Now, we can archive ("save") some object (in this example, a movie) to the file path returned by the our **`filePath`** method



```
[NSKeyedArchiver archiveRootObject:self.movie toFile:[self filePath]];

[NSKeyedArchiver archiveRootObject:self.moviesArray toFile:[self filePath]];
```

- And unarchive as well (this loads + returns the object):



```
[NSKeyedUnarchiver unarchiveObjectWithFile:[self filePath]];
```

Xcode Time

- Now, we're going to add persistence support to our todo app
  - Download the S5 project if needed
- Find a partner(s) and work in groups of 2-3
- Goal: Use NSCoding + NSKeyedArchiver to **save** and **load** your items array

- **Goal**: We want to use NSCoding and NSKeyedArchiver to save and load the items array

- **Hints**:

  - If we are trying to persist an NSArray of Item objects, Item needs to conform to NSCoding

  - When a new Item is added, re-persist ("re-save" the array)

  - When our app launches, we should load our array with NSKeyedArchiver, then refresh the table view

  - **Highly recommended:** Write 2 helper functions: loadItems and persistItems

    - with loadItems, **check for a nil-result**