CS 329E
Elements of Mobile Computing

Fall 2017
University of Texas at Austin

Lecture 10

Agenda

- User Defaults
- Core Data
- Homework 5

User Defaults

User Defaults

What are user defaults?

- A way to store simple key/value pairs
  - Similar to a dictionary, but with persistence
- Very easy mechanism, but limited capabilities
- Typically used for application configuration data
- Data types you can store in NSUserDefaults:
  - NSData
  - NSString; String
  - NSNumber; UInt/Int/Float/Double/Bool
  - NSDate
  - NSArray; Array
  - NSDictionary; Dictionary
- You use a singleton object to access the user defaults store

User Defaults

Getting the NSUserDefaults singleton object:

```
let defaults = NSUserDefaults.standardUserDefaults()
```

User Defaults

Writing to user defaults:

- Get the singleton NSUserDefaults object
- Call the appropriate method for the data you want to store - providing key and value
- Synchronize to guarantee the data was written to the hard drive

User Defaults

Writing to user defaults:

```
// Define keys for the values to store
let kUserIdKey = "userId"
let kTotalKey = "total"
let kNameKey = "name"
```

```
// Define the values to store
let userId = 900
let total = 1275.55
let name = "University of Texas"

// Get a reference to the global user defaults object
let defaults = NSUserDefaults.standardUserDefaults()

// Store various values
defaults.setInteger(userId, forKey: kUserIdKey)
defaults.setDouble(total, forKey: kTotalKey)
defaults.setObject(name, forKey: kNameKey)

defaults.synchronize()  // force the write to the device
```

User Defaults

Writing to user defaults:

- There are several convenience methods for writing

func **setBool**(value: Bool, forKey defaultName: String)
func **setInteger**(value: Int, forKey defaultName: String)
func **setFloat**(value: Float, forKey defaultName: String)
func **setDouble**(value: Double, forKey defaultName: String)
func **setObject**(value: AnyObject?, forKey defaultName: String)
func **setURL**(url: NSURL, forKey defaultName: String)

User Defaults

Reading from user defaults:

```
// Retrieve the previously stored values
let retrievedUserId = defaults.integerForKey(kUserIdKey)
let retrievedTotal = defaults.doubleForKey(kTotalKey)
let retrievedName = defaults.objectForKey(kNameKey)
```

User Defaults

Reading from user defaults:

- There are several convenience methods for reading

func **boolForKey**(defaultName: String) -> Bool
func **integerForKey**(defaultName: String) -> Int
func **floatForKey**(defaultName: String) -> Float
func **doubleForKey**(defaultName: String) -> Double
func **objectForKey**(defaultName: String) -> AnyObject?
func **URLForKey**(defaultName: String) -> NSURL?
func **dataForKey**(defaultName: String) -> NSData?
func **stringForKey**(defaultName: String) -> String?
func **stringArrayForKey**(defaultName: String) -> [AnyObject]?
func **arrayForKey**(defaultName: String) -> [AnyObject]?
func **dictionaryForKey**(defaultName: String) -> [NSObject : AnyObject]?

User Defaults

Some caveats:

- The methods that return Bool, Int, Float, and Double do not return Optionals. In those cases, they return values appropriate to their types.
- Methods that return *AnyObject* indicates you'll need to cast to the correct type before using.

User Defaults

Demo:

- TestUserDefaultsSwift
- TestUserDefaults2Swift

Core Data

Core Data

What is Core Data?

An iOS framework that provides powerful functionality to query and persist non-trivial data locally.

It lets developers store and retrieve data in a database in an object-oriented way.

You can easily map the objects in your apps to the table records in the database without even knowing any SQL.

Core Data

What is Core Data?

That said, it is much more than a storage mechanism:

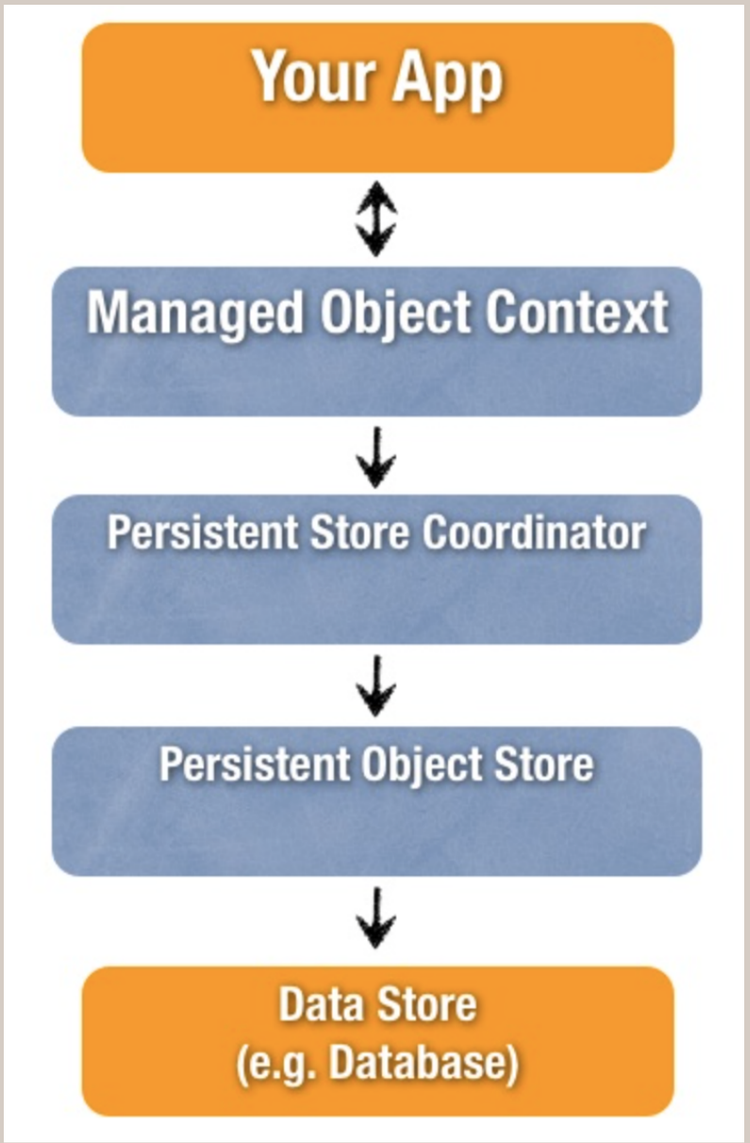
- Manages data object life cycles
- Tracks changes to the data
- Effortless undo support
- Saves data to disk

Core Data

- One of the ways to persist data locally in an iOS app
- Some consider it the best for non-trivial storage
- Makes it simple and fast to work with large amounts of data
- It's not a ‘database' per se but you can do *database like* stuff
- Multiple choices for the 'backing store’ (the technology that actually stores the data)
  - Sqlite (default)
  - XML
  - Binary

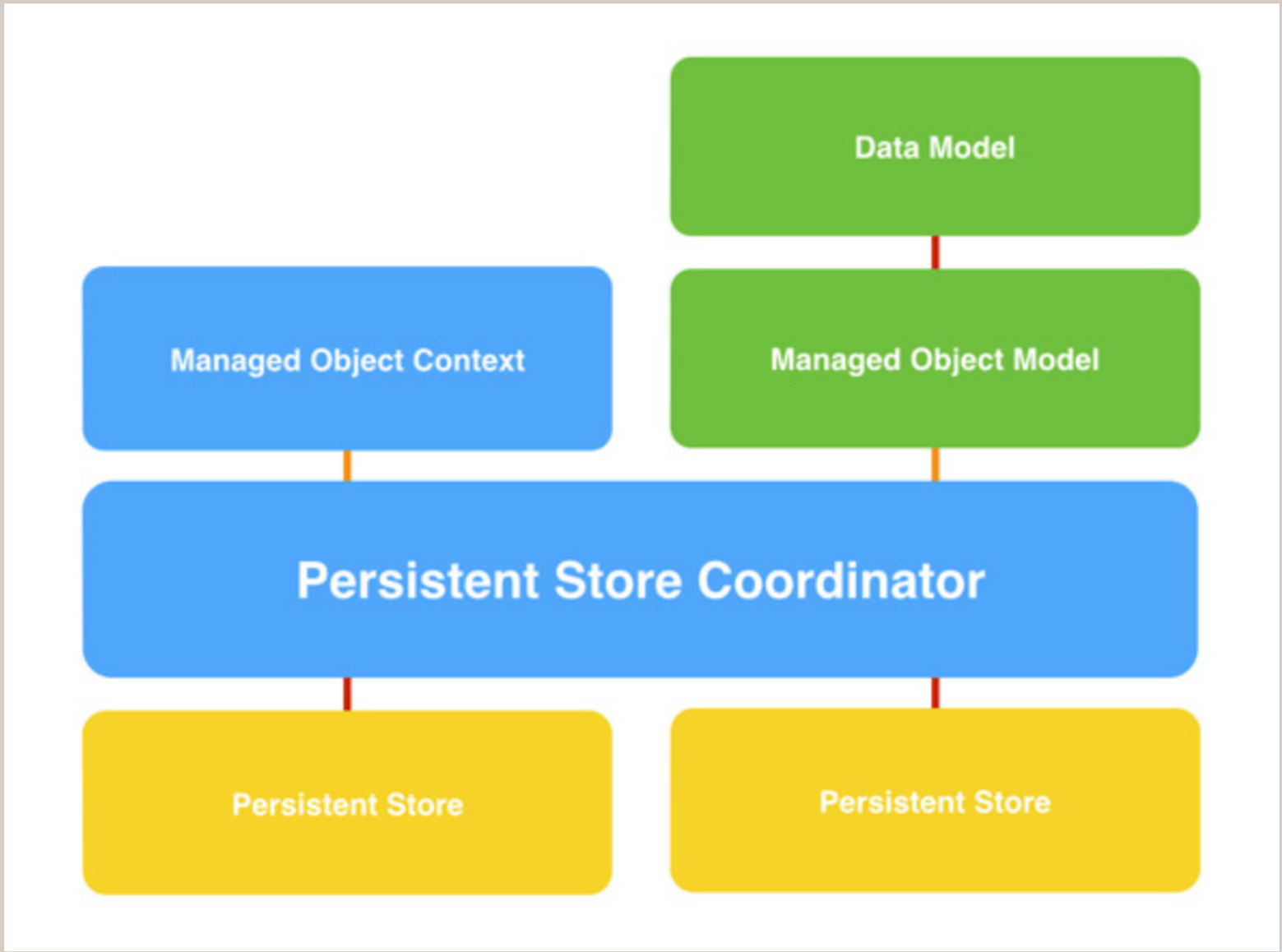
Core Data

Architectural Overview:



Core Data

Architectural Overview (continued):



Core Data

Architectural Overview (continued):

- Managed Object Model
  - Defines the structure of the data - the data schema
    - Data type, relationships
    - Use the data model design tool in Xcode to define the models
- Managed Object Context
  - Acts as a temporary scratch space in memory
  - Objects fetched from the persistent store are stored in the context - where we can manipulate them
  - Changes are monitored here

Core Data

Architectural Overview (continued):

- Entities
  - Instances of the data models
- Properties
  - Attributes
    - Values stored in the entities
  - Relationships
    - Connections between entities
  - Fetched properties
    - Similar to relationships, but used to model weak one-way connections between entities

Core Data

Architectural Overview (continued) - some more formal descriptions:

- An **entity** is a class definition in Core Data
  - Example: An Employee or a Company. In a relational database, an entity corresponds to a *table*. (Think of a table like an Excel spreadsheet, with columns and rows)
- An **attribute** is a piece of information attached to a particular entity
  - Example: An Employee entity could have attributes for the employee's name, position and salary. In a relational database, an attribute corresponds to a particular column in a table.
- A **relationship** is a link between multiple entities
  - In Core Data, relationships between two entities are called *to-one* relationships, while those between one and many entities are called *to-many* relationships
  - Example: A Manager can have a to-many relationship with a set of employees, whereas an individual Employee will have a to-one relationship with his/her manager.

Core Data

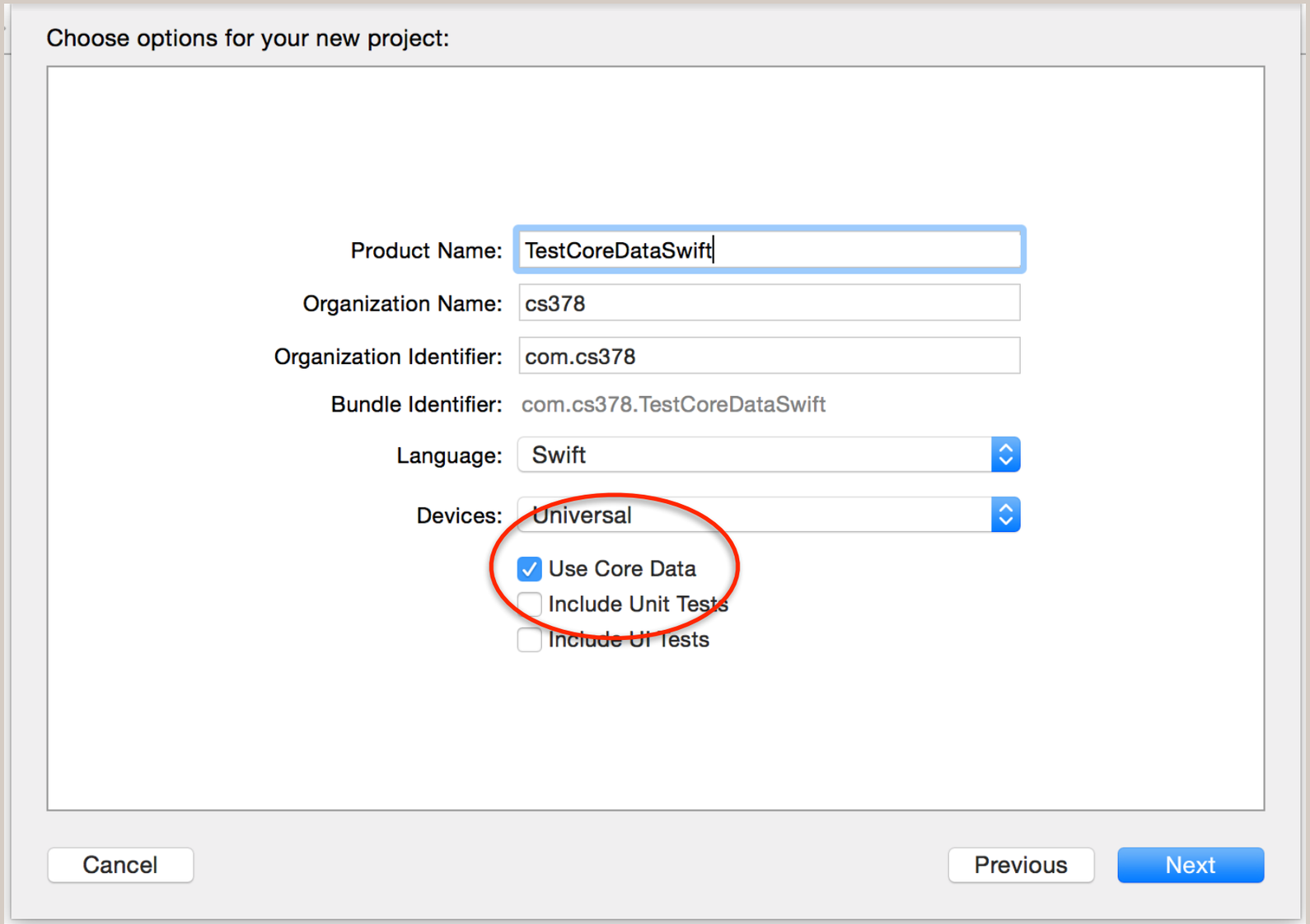
Architectural Overview (continued):

- Relationships
  - One-to-One
  - One-to-Many, Many-to-One
  - Many-to-Many
    - Not naturally represented in database-like storage

Core Data

When creating your project, make sure to check **Use Core Data**

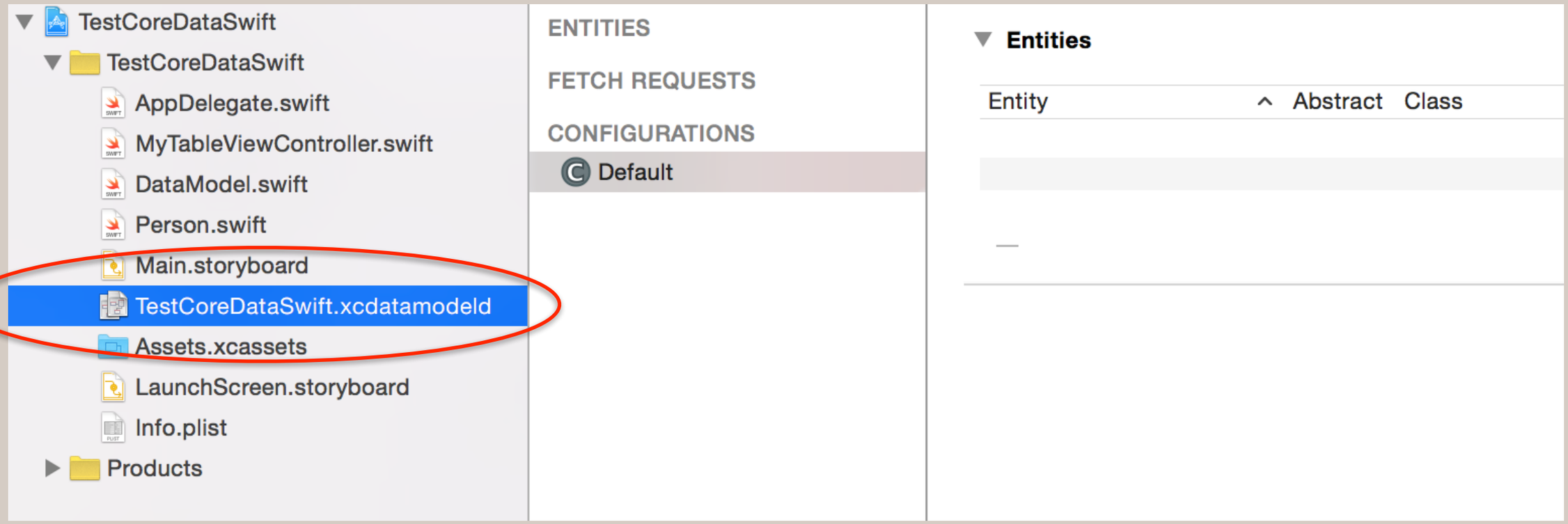
- Includes code in AppDelegate.swift



Core Data

The Core Data data model:

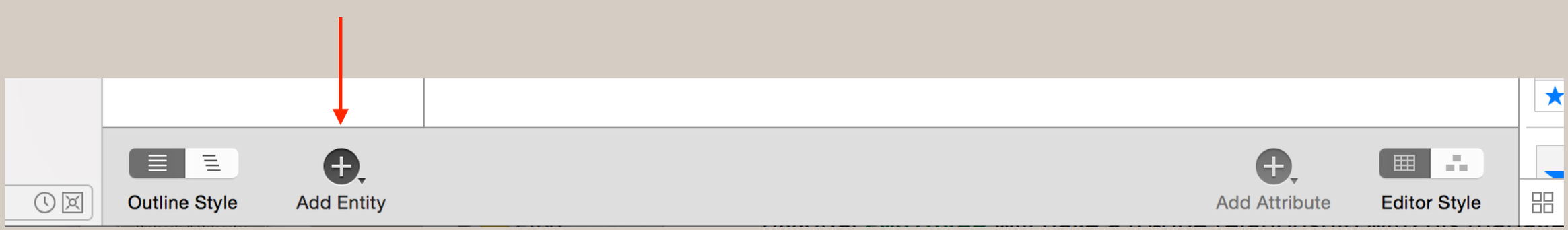
- By checking ‘Use Core Data’ Xcode automatically creates an **xcdatamodeld** file
- This is where you define the attributes of your model, using the data model tool in Xcode



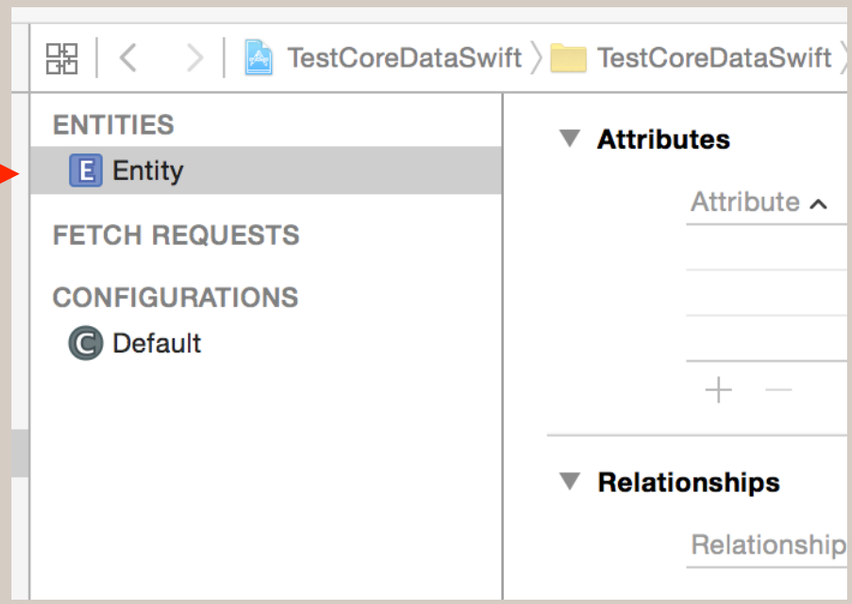
Core Data

The Core Data data model:

- To start you need to add an Entity
  - Think of this like the Person class we defined in an earlier sample project



You will end up with this

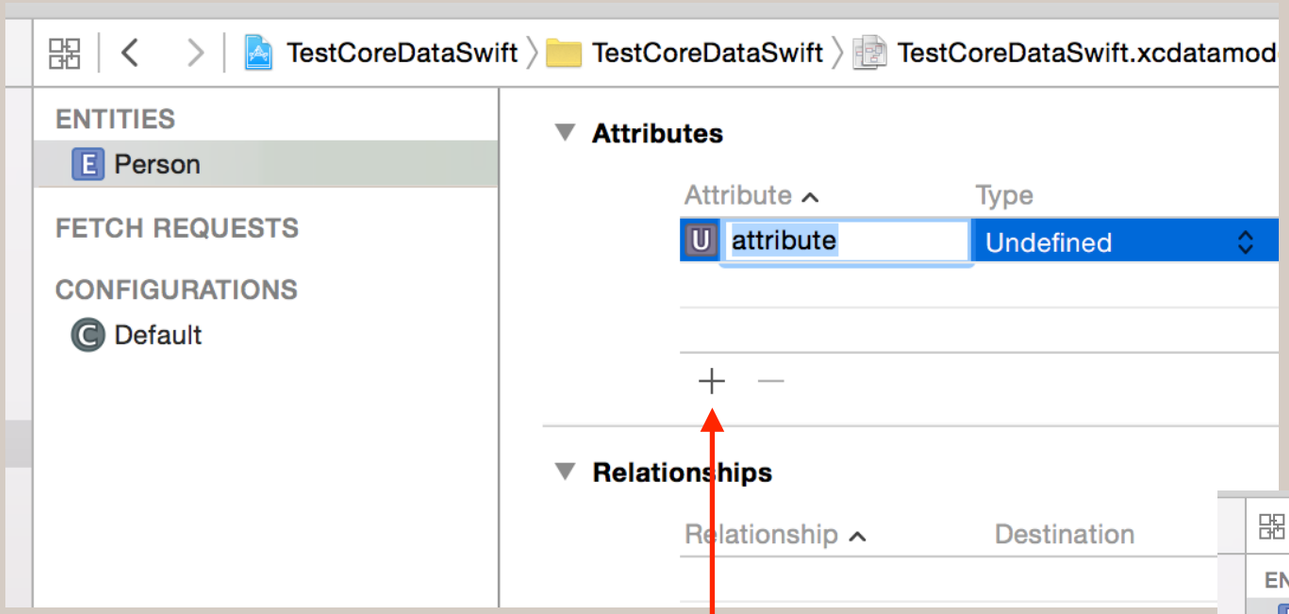


Double-click to change the name

Core Data

The Core Data data model:

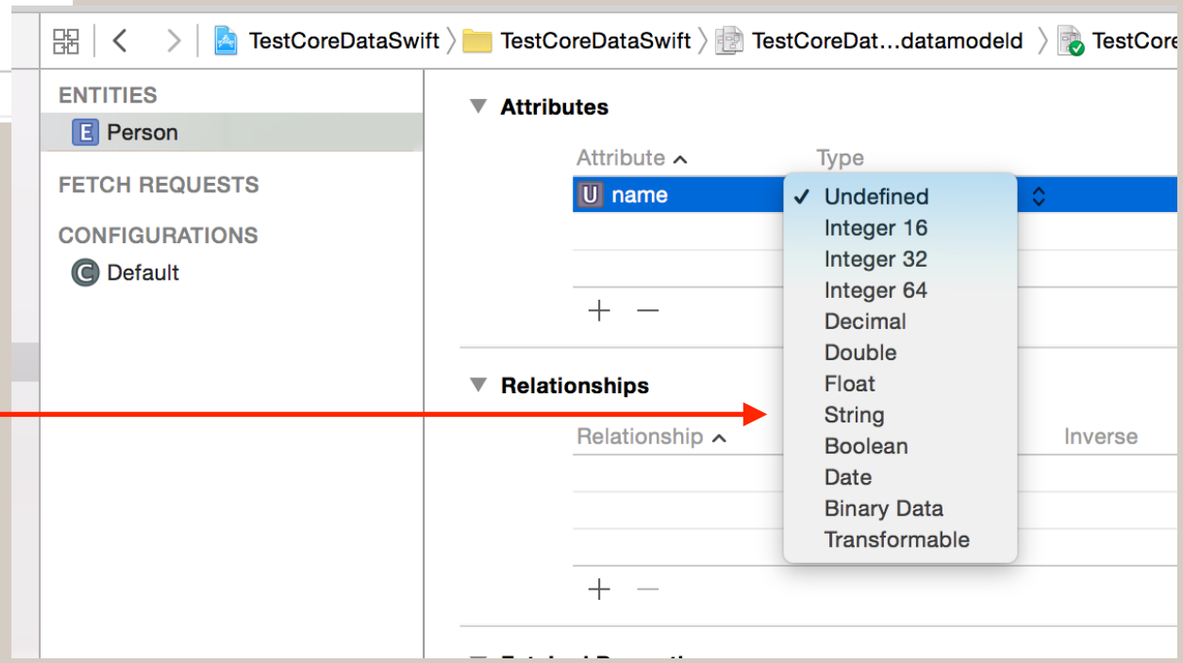
- You'll then want to add attributes



Set the name

To add an attribute

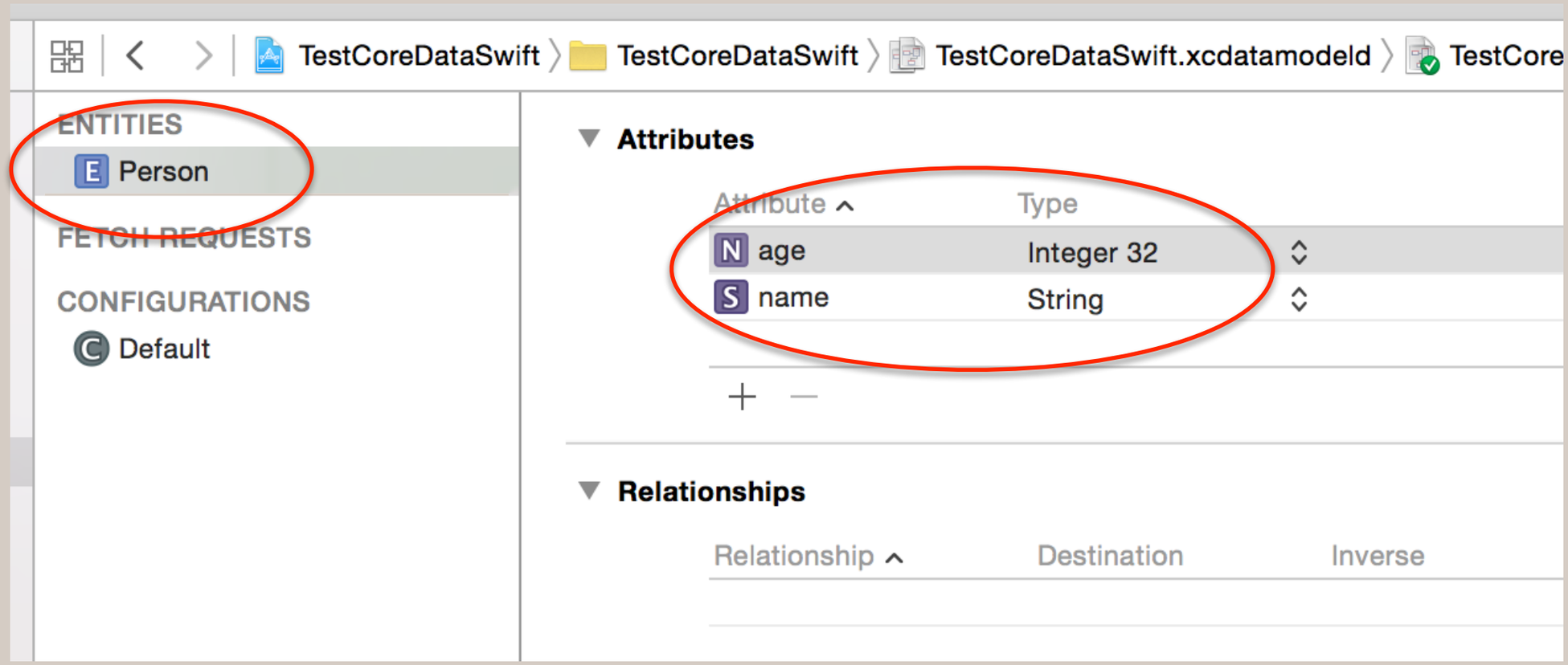
Set the data type



Core Data

The Core Data data model:

- After adding the name and age attributes



Core Data

Then define a property you will use to get access to the managed objects that will contain your data

- In our case, we have an array of managed objects, since we'll have more than one data object to manage
- You'll then use the *people* property to read and write your data

```
// Core Data object
// Holds instances of NSManagedObject rather than our Person class objects
var people = [NSManagedObject]()
```

Core Data

Writing using Core Data:

```
func savePerson(name: String, age: String) {
    let appDelegate = UIApplication.shared.delegate as! AppDelegate
    let managedContext = appDelegate.managedObjectContext
    // Create the entity we want to save
    let entity =  NSEntityDescription.entity(forEntityName: "Person", in: managedContext)

    let person = NSManagedObject(entity: entity!, insertInto:managedContext)

    // Set the attribute values
    person.setValue(name, forKey: "name")
    person.setValue(Int(age), forKey: "age")

    // Commit the changes.
    do {
        try managedContext.save()
    } catch {
        // what to do if an error occurs?
        let nserror = error as NSError
        NSLog("Unresolved error \(nserror), \(nserror.userInfo)")
        abort()
    }

    // Add the new entity to our array of managed objects
    people.append(person)
}
```

Core Data

Reading using Core Data:

```
let appDelegate = UIApplication.shared.delegate as! AppDelegate

let managedContext = appDelegate.managedObjectContext

//
let fetchRequest = NSFetchRequest<NSFetchRequestResult>(entityName:"Person")

//
var fetchedResults:[NSManagedObject]? = nil

do {
    try fetchedResults = managedContext.fetch(fetchRequest) as? [NSManagedObject]
} catch {
    // what to do if an error occurs?
    let nserror = error as NSError
    NSLog("Unresolved error \(nserror), \(nserror.userInfo)")
    abort()
}

if let results = fetchedResults {
    people = results
} else {
    print("Could not fetch")
}
```

Core Data

Demo:

- TestCoreDataSwift

Core Data

Core Data tutorial:

Ray Wenderlich - Getting Started with Core Data Tutorial

https://www.raywenderlich.com/145809/getting-started-core-data-tutorial

In-Class Exercise

In-Class Exercise

Create an application that uses Core Data.

Homework

Homework

Homework 5:

- Create an application that uses:
  - Navigation controller
  - Table view controller
  - Single view controller
  - Segmented control
  - Core data