Lecture 20:
Chapter 27: Savings Locations

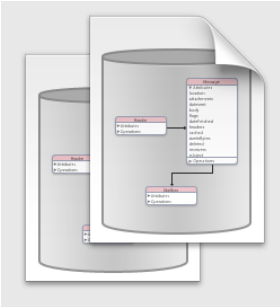
Ruth Torres Castillo

EE443/ EE593
Mobile Application Development

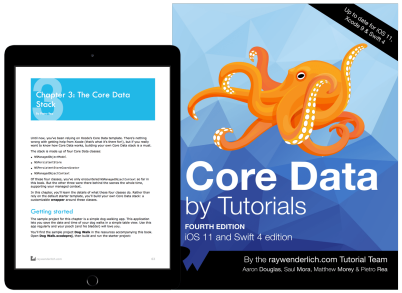
Spring 2018

Outline

- Core Data overview
- Add Core Data
- The data store
- Pass the context
- Browse the data
- Save the locations
- Handle Core Data errors



Core Data overview



- An object persistence framework
- Describes data with a high level data model in terms of entities and their relationships.
- Perform multiple queries over the same data.
- Serialize/deserialize big data structures
- Automatic validation of property values
- Decreases by 50 to 70 percent the amount of code

Plist vs Core Data

- Property lists should be used for data that consists primarily of strings and numbers. They are very inefficient when used with large blocks of binary data.
- Plist will get you up and running faster

- If your data model starts getting richer with additional attributes and relationships, then Core Data would provide better scaling and better support.
- Core Data will give you lots of power when the time comes.

Outline

- Core Data overview
- Add Core Data
- The data store
- Pass the context
- Browse the data
- Save the locations
- Handle Core Data errors

Add Core Data

1. Create the data model
2. Generate the code

Create the data model

- Attributes for the location entity
  - latitude, type Double
  - longitude, type Double
  - date, type Date
  - locationDescription, type String
  - category, type String
  - placemark, type Transformable

Generate the code

- The **Codegen** to Manual/None
- Choose **Editor --> Create NSManagedObject Subclass**
- Two files are created
  - **Location+CoreDataProperties.swift**
  - **Location+CoreDataProperties.swift**

Outline

- Core Data overview
- Add Core Data
- The data store
- Pass the context
- Browse the data
- Save the locations
- Handle Core Data errors

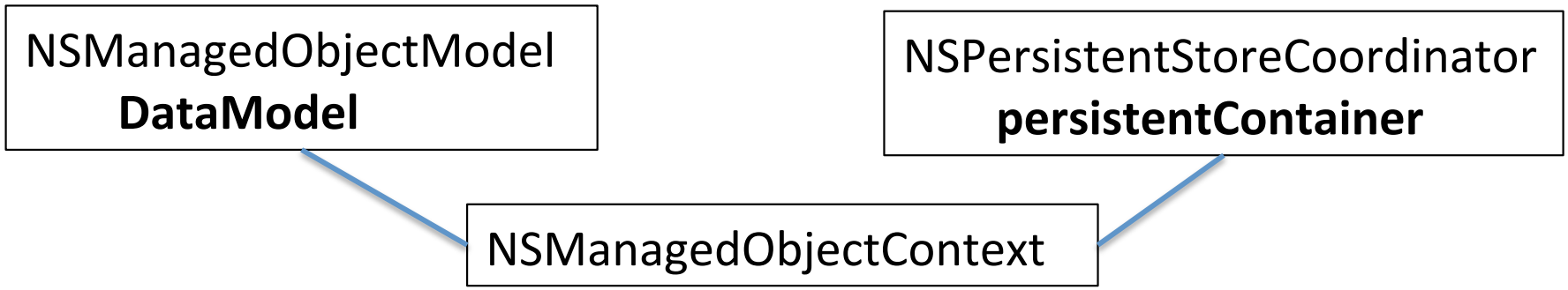
The data store

- Core Data stores its data into an SQLite database
- On **AppDelegate.swift** we need to initialize the data store
- Import the Core Data Framework

```
import CoreData
```

The data store: Core Data stack

- NSManagedObjectModel object: represents the data model during running time
- NSPersistentStoreCoordinator object: is in charge of the SQLite database
- NSManagedObjectContext object: the connection to the persistent store coordinator



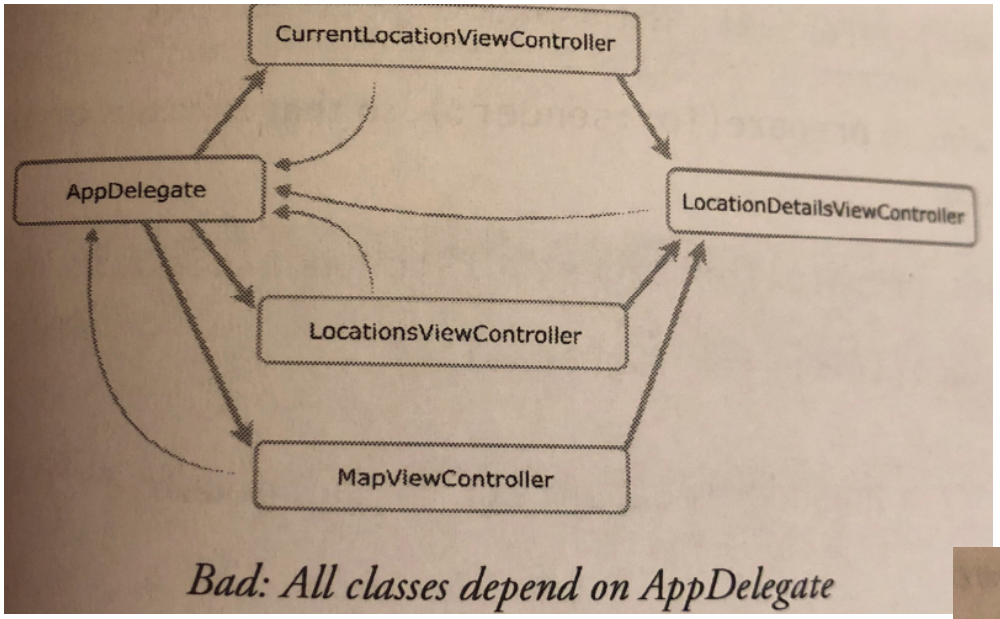
Outline

- Core Data overview
- Add Core Data
- The data store
- Pass the context
- Browse the data
- Save the locations
- Handle Core Data errors

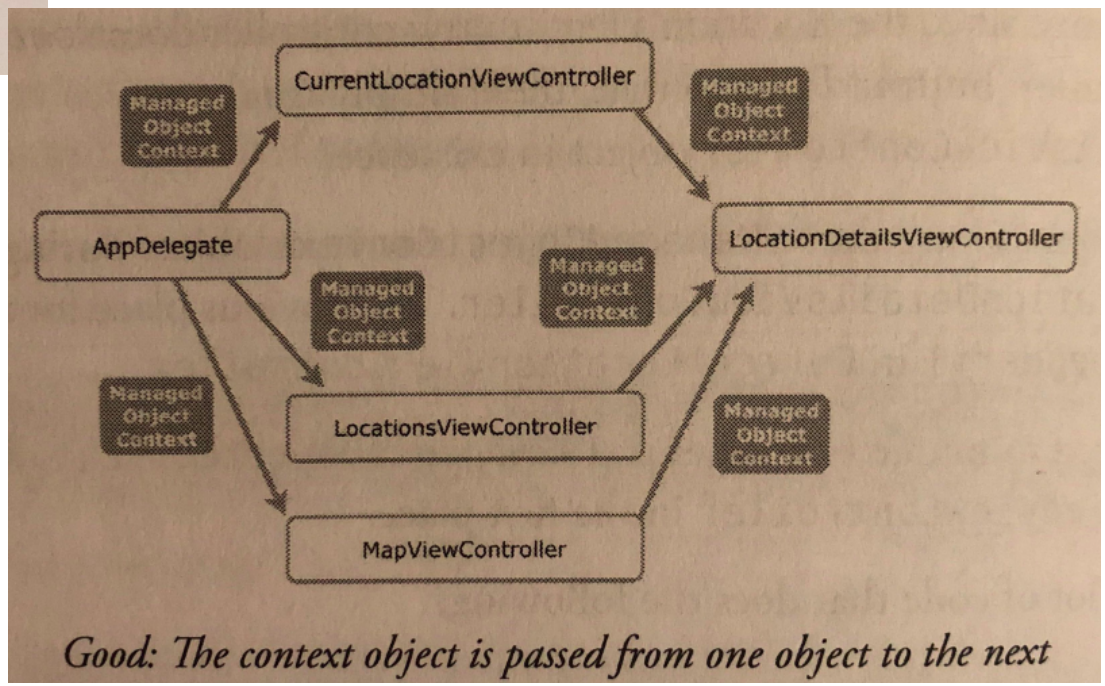
Pass the context

1. Get the context
2. Pass the context from the App Delegate

Dependency Injection



Bad: All classes depend on AppDelegate



Good: The context object is passed from one object to the next

Get the context

- Import Core Data and add a new instance variable in both LocationDetailsViewController.swift & CurrentLocationViewController.swift

```
import CoreData
var managedObjectContext: NSManagedObjectContext!
```

Get the context

- Pass on the context to the Tag Location screen

```
override func prepare(for segue: UIStoryboardSegue,
sender: Any?) {
if segue.identifier == "TagLocation" {
    . . .
 // New code
 controller.managedObjectContext = managedObjectContext
  }
 }
```

Pass the context from the App Delegate

- In the AppDelegate.swift file change the application(_:didFinishLaunchingWithOptions:) method to:

```
let tabController = window!.rootViewController as!
UITabBarController
if let tabViewControllers =
tabController.viewControllers {
    let navController = tabViewControllers[0] as!
UINavigationController
    let controller = navController.viewControllers.first as!
CurrentLocationViewController
    controller.managedObjectContext = managedObjectContext}
```

Outline

- Core Data overview
- Add Core Data
- The data store
- Pass the context
- Browse the data
- Save the locations
- Handle Core Data errors

Browse the data

1. Find Core Data data store location
2. Browse the Core Data store using a GUI app
3. Troubleshoot Core Data issues

Find Core Data data store location

- Add finding paths code to **Functions.swift** and **AppDelegate.swift**
- Open the path in finder to see the files

Core Data store using a GUI app

For this app we will use Liya software, http://cutedgesystems.com/software/liya/

- Start Liya. Under Database Type choose SQLite and Login to the file from Finder Window.
- We can also use SQLiteStudio, sqlitestudio.pl

Troubleshoot Core Data issues

- Scheme defines a collection of targets to build, a configuration to use when building, and a collection of tests to execute
- Choose **Edit Scheme** from **MyLocation > iPhone** menu
- Add arguments in the **Arguments Passed On Launch** section

  -com.apple.CoreData.SQLDebug 1

  -com.apple.CoreData.Logging.stderr 1

Outline

- Core Data overview
- Add Core Data
- The data store
- Pass the context
- Browse the data
- Save the locations
- Handle Core Data errors

Save the locations

- Add the instance variable for date to **LocationDetailsViewController.swift**:

```
var date = Date()
```

- Change the line that sets the dateLabel's text to:

```
dateLabel.text = format(date: date)
```

Save the locations

- The magic in the done( ) method:

```
@IBAction func done() {
…
// Create a new Location instance
   let location = Location(context:
managedObjectContext)
// Once you create the instance,
you can use it like any other
object
location.locationDescription =
descriptionTextView.text
   location.category = categoryName
   location.latitude =
coordinate.latitude
   location.longitude =
coordinate.longitude
   location.date = date
   location.placemark = placemark
// Save the context, so you are
able to look in the data
do {
    try
managedObjectContext.save()
…
    }
   } catch {
// Output the error
    fatalCoreDataError(error)
   }
}
```

Save the locations

- Run the app and tag a Location. Enter a description and press the **Done** button.
- In **Liya**, refresh the contents of the **ZLOCATION** table pressing the **GO** button

Outline

- Core Data overview
- Add Core Data
- The data store
- Pass the context
- Browse the data
- Save the locations
- Handle Core Data errors

Handle Cora Data errors

```
do {
    try managedObjectContext.save()
…
    }
  } catch {
// Output the error
    fatalCoreDataError(error)
  }
```

Fake errors for testing purposes

- Uncheck the Optional flag on the placemark attribute on **DataModel.xcdatamodeld** file
- Remove the DataModel.sqlite file as well as the –shm and –wal files and run the app again

Fake errors for testing purposes

- We can also comment the line:

```
self.placemark = p.last!
```

inside

```
locationManager(_:didUpdateLocations: )
```

- Try to beat the reverse geocoder and the Tag Location will have a placemark as nil.
- The app will crash, so we need to stop since it does not provide any alert.

Alert the user about crashes

- New function to **Functions.swift**

```
…

func fatalCoreDataError(_ error: Error) {
    print("*** Fatal error: \(error)")
    NotificationCenter.default.post(
 name: CoreDataSaveFailedNotification, object:nil)
}
```

- Replace the error handling code in the done( ) action in **LocationDetailsViewController.swift** with:

```
. . .
} catch {
        fatalCoreDataError(error)
}
```

Alert the user about crashes

- NotificationCenter in **AppDelegate.swift**

```
//Tell NotificationCenter that you want to be notified
NotificationCenter.default.addObserver(forName:
CoreDataSaveFailedNotification, object: nil, queue:
OperationQueue.main, using: { notification in

//Set up the error message
      let message = """
There was a fatal error in the app and it ...
"""
//Create a UIAlertController
let alert = UIAlertController(title: "Internal Error", message: message,
preferredStyle: .alert)
```

Alert the user about crashes

```
//Add an action for the alert’s OK button
let action = UIAlertAction(title: "OK", style: .default) { _ in
        let exception = NSException(name:
NSExceptionName.internalInconsistencyException, reason: "Fatal Core Data
error", userInfo: nil)
        exception.raise()
}
…)
```

Alert the user about crashes

- Add the call in the **AppDelegate.swift** class
  `listenForFatalCoreDataNotifications()`
- Try to tag a location before the street address has been obtained

REMEMBER!

- In the data model, set the placemark attribute back to optional and uncomment the line if did in **CurrentLocationViewController.swift**

Recapitulating

- Core Data overview
- Add Core Data
  - Create the data model
    - Entity
    - Attribute
  - Generate the code
    - Location+CoreDataProperties.swift
    - Location+CoreDataProperties.swift
- The data store: Core Data stack
  - Database was loaded into memory and the Core Data stack was set up

Recapitulating

- Pass the context
  - Get the context:
    - LocationDetailsViewController.swift
    - CurrentLocationViewController.swift
  - Pass the context from App Delegate
    - Database was created for Core Data
- Browse the data
  - Core Data data store location
  - Browse the Core Data store using a GUI app
  - Troubleshoot Core Data issues

Recapitulating

- Save the locations
- Handle Core Data errors
    - Fake errors for testing purposes
    - Alert the user about crashes