

# Chapter 23

## Core Data

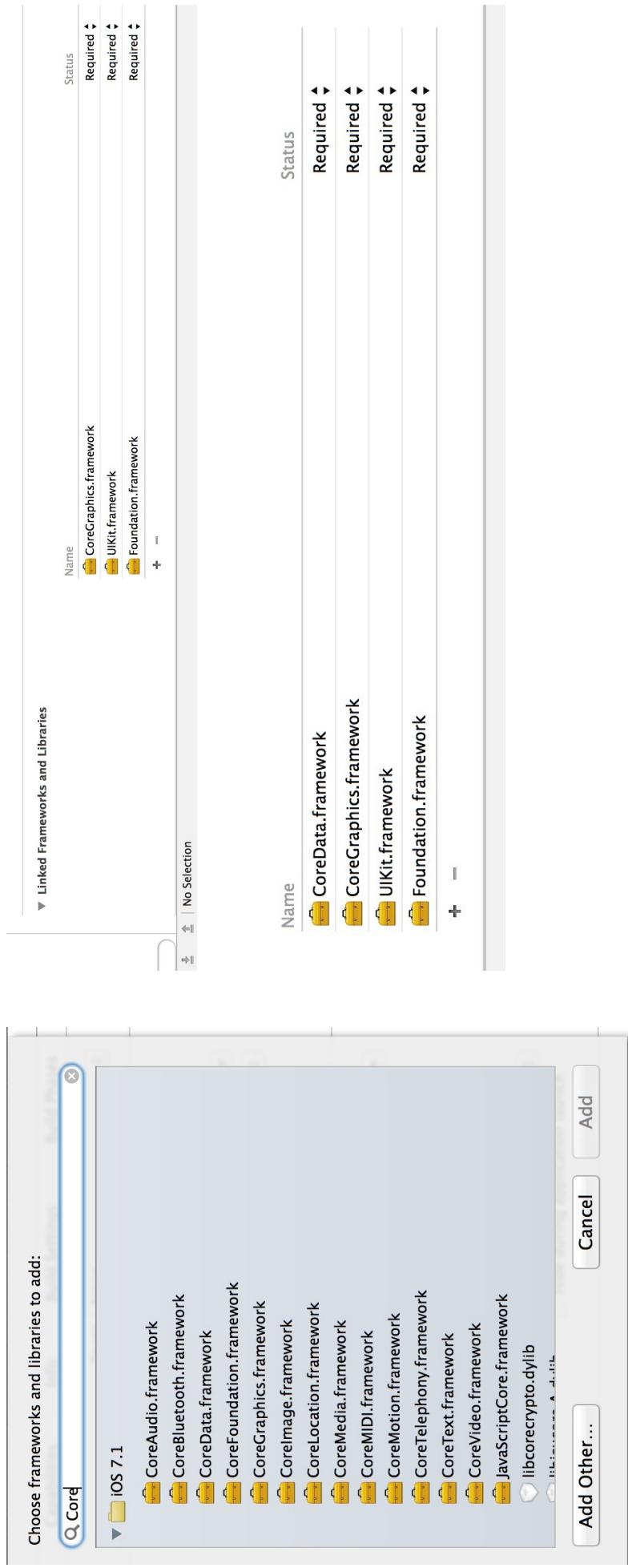
- Object-Relational Mapping
- Moving Homeowner to Core Data

# Core Data

- Core Data is a framework that provides object-relational mapping.
- Core Data can turn Objective-C objects into data that is stored in a SQLite database file and vice-versa.
- SQLite is a relational database that is stored in a single file.
- SQLite is the library that manages the database file
- SQLite is not a full-fledged relational database server
- Core Data gives us the ability to fetch and store data in a relational database without having to know SQL.
- We do have to understand a bit about how relational databases work.

# Moving Homeowner to Core Data

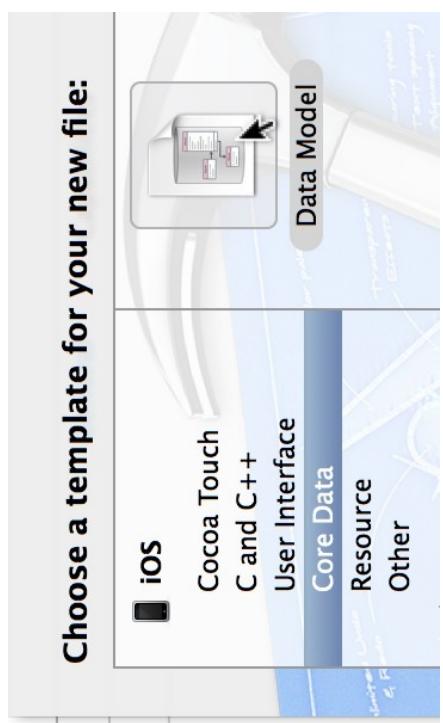
- Copy the latest version of Homeowner application to a new location
- Core Data is a framework that needs to be added to your application



# The Model File

- In relational database we use the terms entity and attribute as the conceptual names for the information that we store in tables and columns, Objective-C uses the Entity/Attribute and Relation terminology too.
- A Core Data model file is the description of every entity with all its attributes.
- Entities in Core Data, as in relational databases, can be related to each other.
- Core Data supports one-to-one and one-to-many:
  - every instance in INIItem will map to one instance of INIAssetType
  - every instance in INIAssetType will map to many instances of INIItem, that is to an instance on NSSet of INIItems

# Creating Model Files in Core Data



The screenshot shows the Xcode Core Data editor. At the top, there is a search bar with 'As: Homeowner.xcdatamodeld' and a 'Create' button. Below the search bar, there are sections for 'args:' and 'Data Model'. In the 'Data Model' section, a 'Homeowner' entity is listed. The main workspace shows the 'Homeowner.xcdatamodeld' file structure. On the left, there are tabs for 'ENTITIES', 'FETCH REQUESTS', and 'CONFIGURATIONS', with 'Default' selected under 'CONFIGURATIONS'. The central area lists files like 'Homeowner.xib', 'INIDetailViewController.xib', 'Homeowner', 'DataStore', and various .h and .m files for different view controllers and delegates. On the right, there are sections for 'Images.xcassets', 'Supporting Files', and 'Products'. At the bottom right, there are buttons for 'Outline Style', 'Add Entity', 'Editor Style', 'Add Attribute', and 'Add Product'.

# The model file NItem for Homeowner

The screenshot shows the Xcode Model Editor with the following details:

- Model:** Homeowner.xcdatamodeld
- Entity:** NItem (selected)
- Attributes:**
  - Thumbnail (Transformable, Type: T)
  - dateCreated (Date, Type: D)
  - itemKey (String, Type: S)
  - itemName (String, Type: S)
  - serialNumber (String, Type: S)
  - valueInDollars (Integer 32, Type: N)
- Relationships:**
  - Relationship: Des (Default, Entity: Des, Type: S)
- Buttons:**
  - Add (+)
  - Remove (-)
  - Fetch Property ▲

The screenshot shows the Xcode Model Editor with the following details and annotations:

- Model:** Homeowner.xcdatamodeld
- Entity:** NItem (selected)
- Attributes:**
  - itemName (String, Type: S) - highlighted in green.
  - valueInDollars (Integer 32, Type: N) - highlighted in green.
- Relationships:**
  - Relationship: Inv (Default, Entity: Inv, Type: S)
- Buttons:**
  - Add (+)
  - Remove (-)
- Annotations:**
  - A callout bubble points to the 'valueInDollars' attribute with the text: "orderingValue used to keep track of the wa items are ordered in th table".

# Handling The thumbnail Attribute

- Core Data can only store certain data types in its store.

- UIImage is not one of these types that is why we declared the thumbnail as transformable.

- With a transformable attribute, Core Data will convert the object into NSData when saving, and convert the NSData back into the original object when loading it from the file system.

- For Core Data to do this for the thumbnail attribute we need to supply it with an NSValueTransformer subclass to handles these conversions.



```
9 #import "UIIImageTransformer.h"
10 @implementation UIIImageTransformer
11 +(Class)transformedValueClass
12 {
13     return [NSData class];
14 }
15 -(id)transformedValue:(id)value
16 {
17     if (!value){
18         return nil;
19     }
20     if ([value isKindOfClass:[NSData class]]){
21         return value;
22     }
23     return UIImagePNGRepresentation(value);
24 }
25 }
26 }
27 }
28 -(id)reverseTransformedValue:(id)value
29 {
30     if ([value isKindOfClass:[NSData class]]){
31         return [UIImage imageWithData: value];
32     }
33 }
34 }
```

```

9 #import "INImageTransformer.h"
10 @implementation INImageTransformer
11 + (Class)transformedValueClass
12 {
13     return [NSData class];
14 }
15 -(id)transformedValue:(id)value
16 {
17     if (!value) {
18         return nil;
19     }
20     if ([value isKindOfClass:[NSData class]]) {
21         return value;
22     }
23     return UIImagePNGRepresentation(value);
24 }
25 -(id)reverseTransformedValue:(id)value
26 {
27 }
28 -(id)reverseTransformedValue:(id)value
29 {
30 }
31     return [UIImage imageWithData: value];
32 }
33 @end
34
35
36
37
38
39
39
40
41
42
43
44
45

```

The class method `transformedValueClass` tells the transformer what type of object it will receive from the `transformedValue` method, namely `NSData`.  
The `transformedValue`: method will be called when your transformable variable is to be saved to the file system. It expects a `UIImage` as input and it will return an instance of `NSData` that can be written to the filesystem.  
`reverseTransformedValue`: method is called when the thumbnail data is loaded from the file system, and your implementation will create the `UIImage` from the `NSData` that was stored.

**UIImagePNGRepresentation**

Returns the data for the specified image in PNG format

```
NSData * UIImagePNGRepresentation (
    UIImage * image
);
```

**Parameters**

*image*  
The original image data.

**Return Value**  
A data object containing the PNG data, or `nil` if there was a problem generating the data. This function may return `nil` if the image has no data or if the underlying `CGImageRef` contains data in an unsupported bitmap format.

**Discussion**  
If the image object's underlying image data has been purged, calling this function forces that data to be reloaded into memory.

**Availability**  
Available in iOS 2.0 and later.

**Related Sample Code**

- Core Data Transformable Attributes
- iPhoneCoreDataRecipes
- MVCNetworking
- Declared In

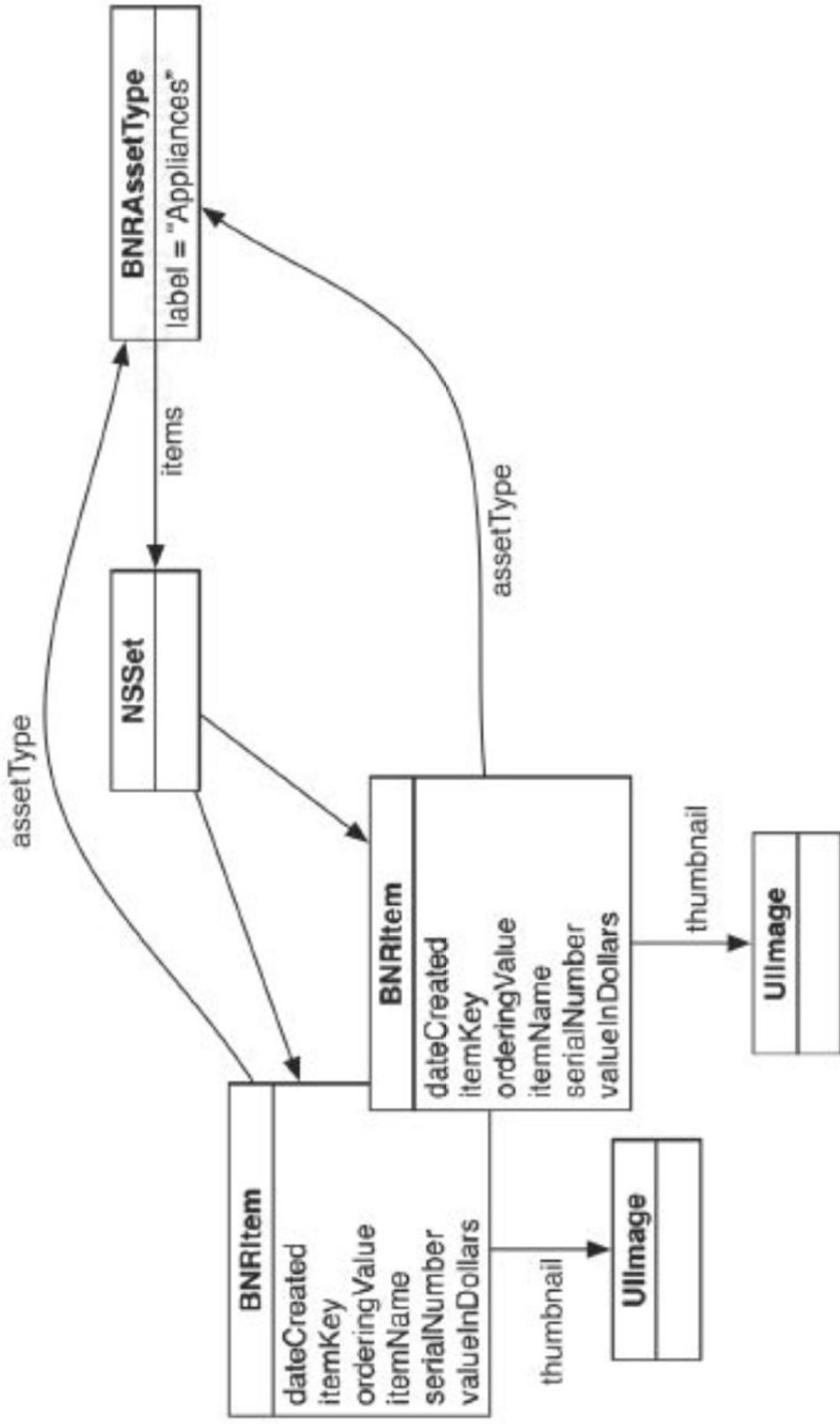
**User Info**

<b>Key</b>	<b>Value</b>
------------	--------------

# Entity Relationships

- One of the benefits to using Core Data is that entities can be related to one another.
- We will add a new entity called `INIAssetType` that describes a category of items.
  - `INIAssetType` will be an entity in the model file, and each row of that table will be mapped to an Objective-C object at runtime.
  - We need to establish relationships between `INIAssetType` and `INItem`.
  - Relationships between entities are represented by pointers between objects.
- There are two kinds of relationships:
  - to-one, each instance of this entity will have a pointer to an instance in the entity it has a relationship to. An item will have a relationship to one type (painting points to art)
  - to-many, each instance of this entity has a pointer to an `NSSet` that contains the instances of the entity that it has a relationship with (art points to a set of all paintings, sculptures ...)

# Entities in Homeowner



# Assigning Relationships

The screenshot shows the Xcode Model Editor interface with the following details:

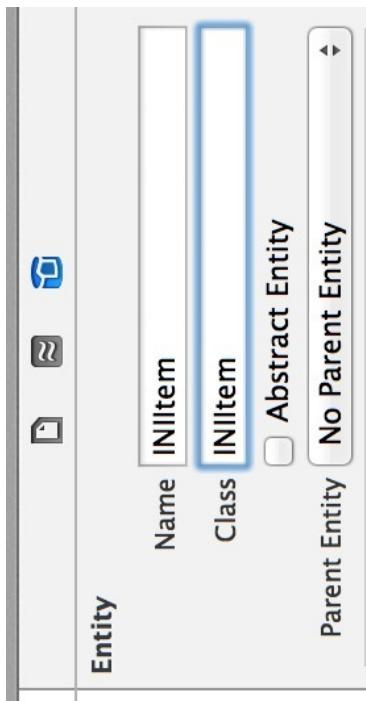
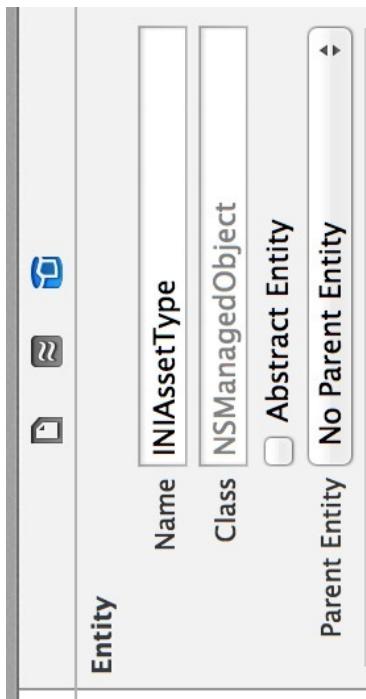
- ENTITIES** section:
  - E INIAssetType**: Selected entity.
  - E INIItem**: Another entity listed.
- ATTRIBUTES** section for INIAssetType:
  - Attribute ▲**: Type String.
  - S label**: Value type String.
- CONFIGURATIONS** section:
  - C Default**: Configuration selected.
- FETCH REQUESTS** section:
  - E INIAssetType**: Fetch request selected.
  - E INIItem**: Another fetch request listed.
- ENTITIES** section for INIItem:
  - E INIAssetType**: Selected entity.
  - E INIItem**: Another entity listed.
- ATTRIBUTES** section for INIAssetType:
  - D dateCreated**: Type Date.
  - S itemKey**: Type String.
  - S itemName**: Type String.
  - N orderingValue**: Type Double.
  - S serialNumber**: Type String.
  - T thumbnail**: Type Transformable.
  - N valueInDollars**: Type Integer.
- CONFIGURATIONS** section:
  - C Default**: Configuration selected.
- RELATIONSHIPS** section:
  - Relationship ▲**: Destination INIItem.
  - M items**: Relationship type To Many.
  - Relationship ▲**: Destination INIAssetType.
  - M assetType**: Relationship type To One.

Relationship configuration details shown in the top right:

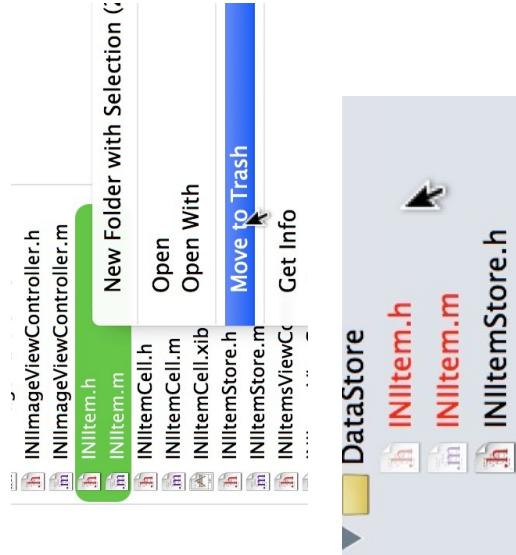
- Relationship** section:
  - Name**: items
  - Type**: To Many
  - Inverse**: To One
- Properties** section:
  - Transient
  - Optional
- Destination** section:
  - INIItem**
- Advanced** section:
  - Index in Spotlight
  - Store in External Record File
- User Info** section:
  - Key**: Value

# NSManagedObject

- When Core Data fetches objects from a table, it stores the information in an instance of NSManagedObject.
- NSManagedObject is a smart class that knows how to cooperate with Core Data and it holds a key-value pair for every property (attribute or relationship) in the entity.
  - If you want to add additional behavior to what is retrieved by Core Data, you should subclass NSManagedObject and have the returned object by Core Data be an instance of this subclass.
  - In our example, we let Xcode generate a subclass of NSManagedObject as our new INIItem class then we add our desired behavior to this new class.



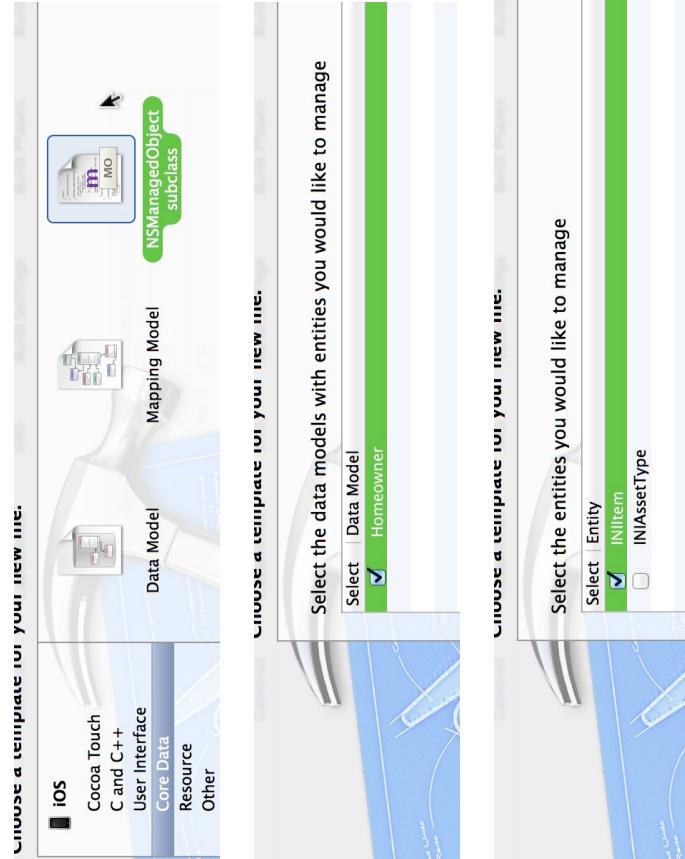
# NSManagedObject and subclasses



```

8 #import <Foundation/Foundation.h>
9
10 #import <CoreData/CoreData.h>
11
12 @interface INIItem : NSManagedObject
13
14 @property (nonatomic, retain) NSDate *dateCreated;
15 @property (nonatomic, retain) NSString *itemKey;
16 @property (nonatomic, retain) NSString *itemName;
17 @property (nonatomic) double orderingValue;
18 @property (nonatomic) NSString *serialNumber;
19 @property (nonatomic, retain) UIImage *thumbnail;
20 @property (nonatomic, strong) NSNumber *valueInDollars;
21 @property (nonatomic, retain) NSManagedObject *assetType;
22
23 -(void) setThumbnailFromImage:(UIImage *) image;
24
25 @end
26
27
28
29
30
31

```



7/16/14

CECS 590, I. Imam

//Figure out a scaling ratio to make sure we maintain the same float ratio = MAX(newRect.size.width /origImageSize.width, new

## awakeFromInsert

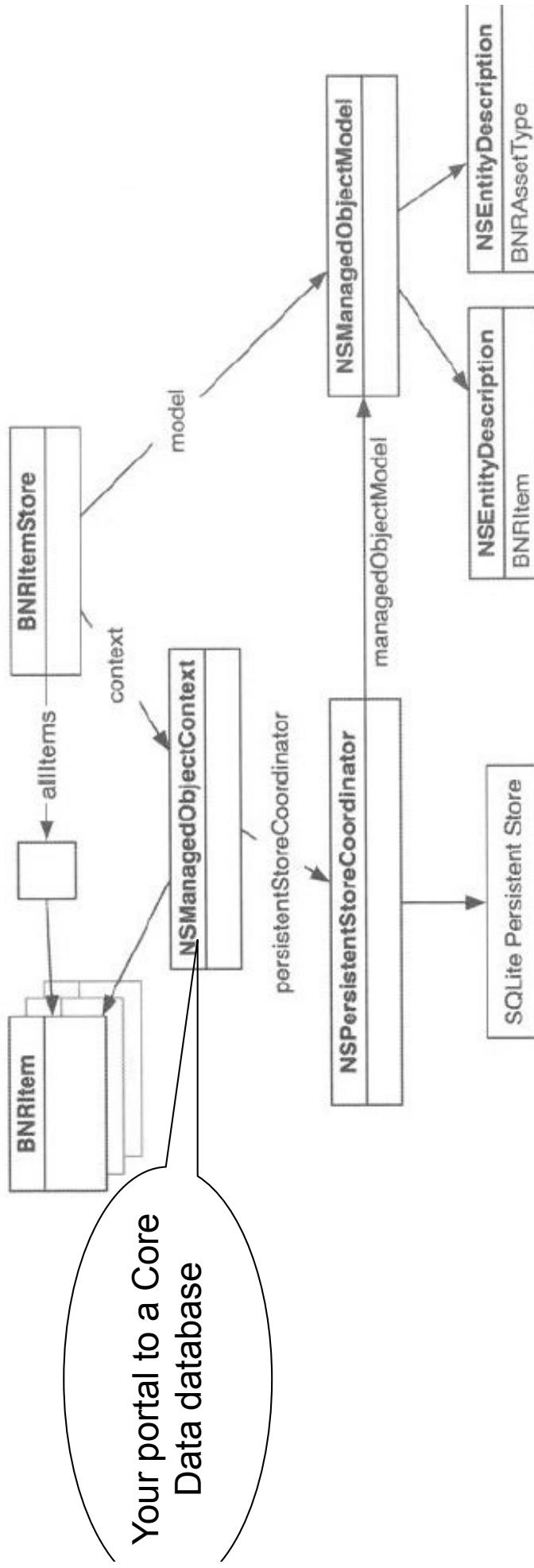
- Of course, when you first launch an application, there are no saved items or asset types.
- When the user creates a new `INItem` instance, it will be added to the database.
  - When objects are added to the database, they are sent the message `awakeFromInsert`.
  - In `awakeFromInsert` is where you will set the `dateCreated` and `itemKey` properties of a `INItem`.

```
59
60 -(void)awakeFromInsert
61 {
62     [super awakeFromInsert];
63     self.dateCreated = [NSDate date];
64     //Create an NSUUID object -and get its string representation
65     NSUUID *uuid = [[NSUUID alloc] init];
66     NSString *key = [uuid UUIDString];
67     self.itemKey = key;
68 }
```

# Updating INIItemStore

- The portal through which you talk to the database is the `NSManagedObjectContext`.
- The `NSManagedObjectContext` uses a persistence store coordinator named `NSPersistentStoreCoordinator`.
- We request the persistent store coordinator to open a SQLite database at a particular filename (connect to the database).
- The persistent store coordinator uses the model file in the form of an instance of `NSManagedObjectModel`.

# Talking to the database in Core Data



A context without a coordinator is not fully functional as it cannot access a model except through a coordinator.

The coordinator needs to know:

- The entities, attributes and relations which are in the Model
- Where to store the data in the persistence store

# Updating NSItemStore

```
#import "INIItemStore.h"
#import "INIIImageStore.h"
#import <CoreData/CoreData.h>

@interface INIItemStore : NSObject

@property (nonatomic) NSMutableArray *privateItems;
@property (nonatomic, strong) NSMutableArray *allAssetTypes;
@property (nonatomic, strong) NSManagedObjectContext *context;
@property (nonatomic, strong) NSManagedObjectModel *model;

@end

~ // Here is the real (secret) initializer
- (instancetype) initPrivate
{
    self = [super init];
    if (self) {
        NSString *path = [self itemArchivePath];
        _privateItems = [NSKeyedUnarchiver unarchiveObjectWithFile: path];
        // If the array hadn't been saved previously, create a new empty one
        if (_privateItems) {
            _privateItems = [[NSMutableArray alloc] init];
        }
        _privateItems = [_privateItems copy];
    }
    return self;
}

// Here is the real (secret) initializer
- (instancetype) initPrivate
{
    self = [super init];
    if (self) {
        // Read in Homeowner.xcdatamodeld
        model = [[NSManagedObjectModel alloc] initWithContentsOfURL: nil];
        NSPersistentStoreCoordinator *psc = [[NSPersistentStoreCoordinator alloc] initWithManagedObjectModel: _model];
        // Where does the SQLite file go?
        NSString *path = self.itemArchivePath;
        NSURL *storeURL = [NSURL fileURLWithPath: path];
        NSError *error;
        if (! [psc addPersistentStoreWithType: NSSQLiteStoreType
                                         configuration: nil
                                           URL: storeURL
                                         options: nil
                                           error:&error]) {
            [NSErrorException raise:@"Open Failure"
                           format:@Reason:@"%@, [error localizedDescription]];
        }
        // Create the managed object context
        _context = [[NSManagedObjectContext alloc] init];
        _context.persistentStoreCoordinator = psc;
        [_self loadAllItems];
    }
    return self;
}

- (NSDocumentDirectory) documentDirectories
{
    NSArray *documentDirectories = NSSearchPathForDirectoriesInDomains(NSDocumentDirectory,
YES);
    NSString *documentDirectory = [documentDirectories firstObject];
    return [documentDirectory stringByAppendingPathComponent:@"store.data"];
}

- (BOOL) saveChanges
{
    BOOL successful = [self.context save:&error];
    if (!successful){
        NSLog(@"Error saving: %@", [error localizedDescription]);
    }
    return successful;
}
```

## NSFetchRequest and NSPredicate

- In this application we will fetch all of the items in store.data the first time you need them.
- To get objects back from the NSManagedObjectContext we must prepare and execute an NSFetchedRequest (SQL Select).
- After a fetch request is executed, we will get an array of all the objects that match the parameters of that request.
- A fetch request needs an entity description that defines which entity you want to get objects from.
- To fetch INItem instances, we specify the INItem entity.
- We can also set the request's sort descriptors to specify the order of the objects in the array.
- A sort descriptor has a key that maps to an attribute of the entity and a BOOL that indicates if the order should be ascending or descending.

# Implementing the NSFetchedRequest and NSPredicate

```
-(void)loadAllItems
{
    if (!self.privateItems){
        NSFetchedRequest *request = [[NSFetchRequest alloc] init];
        NSEntityDescription *e = [NSEntityDescription entityForName:@"INIIItem" inManagedObjectContext: self.context];
        request.entity = e;
        NSSortDescriptor *sd = [NSSortDescriptor sortDescriptorWithKey:@"orderingValue" ascending: YES];
        request.sortDescriptors = @*[sd];
        NSError *error;
        NSArray *result = [self.context executeFetchRequest: request error:& error];
        if (! result){
            [NSErrorException raise:@"Fetch failed"
                format:@"Reason:%@", [error localizedDescription]];
        }
        self.privateItems = [[NSMutableArray alloc] initWithArray: result];
    }
}
```

- A predicate contains a condition that can be true or false.  
NSPredicate \* p = [ NSPredicate predicateWithFormat:@" valueInDollars > 50" ];  
[ request setPredicate: p];
- The format string for a predicate can be very long and complex.
- Predicates can also be used to filter the contents of an array  
NSArray \* expensiveStuff = [ allItems filteredArrayUsingPredicate: p];

# Adding and Deleting Items

```
- (INItem *) createItem
{
    double order;
    if ([self.allItems count] == 0) {
        order = 1.0;
    } else {
        order = [[self.privateItems lastObject] orderingValue] + 1.0;
    }
    NSLog(@"Adding after %lu items, order = %f", (unsigned long)[self.privateItems count], order);

    INItem *item = [NSEntityDescription insertNewObjectForEntityForName:@"INItem"
inManagedObjectContext: self.context];
    item.orderingValue = order;
    [self.privateItems addObject: item];
    return item;
}

- (void) removeItem:(INItem *) item
{
    NSString *key = item.itemKey;
    [[INImageStore sharedStore] deleteImageForKey: key];
    [self.context deleteObject: item];
    [self.privateItems removeObjectIdenticalTo: item];
}

- (void) moveItemAtIndex:(NSUInteger) fromIndex toIndex:(NSUInteger) toIndex
{
    if ( fromIndex == toIndex ) {
        return;
    }

    INItem *item = self.privateItems[fromIndex];
    [self.privateItems removeObjectAtIndex: fromIndex];
    [self.privateItems insertObject: item atIndex: toIndex];

    // Computing a new orderingValue for the object that was moved

    double lowerBound = 0.0;
    // Is there an object before it in the array?
    if (toIndex > 0){
        lowerBound = [self.privateItems[(toIndex - 1)] orderingValue];
    } else {
        lowerBound = [self.privateItems[1] orderingValue] -2.0;
    }

    double upperBound = 0.0;
    // Is there an object after it in the array?
    if (toIndex < [self.privateItems count] -1){
        upperBound = [self.privateItems[(toIndex + 1)] orderingValue];
    } else {
        upperBound = [self.privateItems[(toIndex -1)] orderingValue] + 2.0;
    }

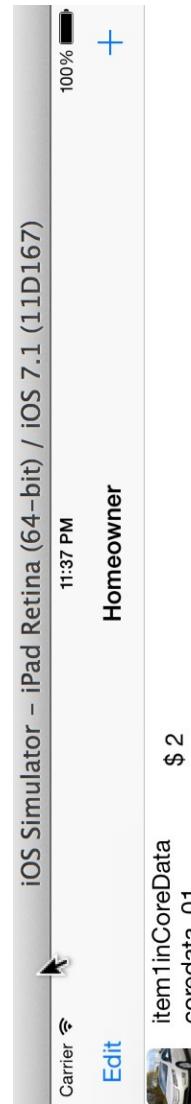
    double newOrderValue = (lowerBound + upperBound)/2.0;
    NSLog(@"%@", moving to order %f", newOrderValue);
    item.orderingValue = newOrderValue;
}
```

The orderingValue is a double to make calculating new order value easier.

# Running The App



```
2014-07-15 23:34:45.293 Homeowner[1280:60b] Adding after 0 items, order = 1.00
2014-07-15 23:34:45.344 Homeowner[1280:60b] Error: unable to find /Users/iniamam/Library/Application Support/iPhone Simulator/7.1-64/Applications/
4D69E3AD-3BA5-4F52-BFF4-7440BE5086E0/Documents/E4D2F59F-6FA8-4B0F-BF91-4CE98BB81CD5A
2014-07-15 23:35:34.330 Homeowner[1280:60b] I am in showImage
2014-07-15 23:35:34.332 Homeowner[1280:60b] Going to show image for <INItem: 0x10d926470> (entity: INItem; id: 0x10d929a40 <x-coredata://INItem/
t4C45B15-31E7-4B18-BE55-E3252ACD07352>; data: {
    assetType = nil;
    dateCreated = "2014-07-16 03:34:45 +0000";
    itemKey = "E4D2F59F-6FA8-4B0F-BF91-4CE98BB81CD5A";
    itemName = item1inCoreData;
    orderingValue = 1;
    serialNumber = "coredata_01";
    thumbnail = "<UIImage: 0x10d9dec050>";
    valueInDollars = 1;
})
```



# Adding INIAssetTypes to Homeowner

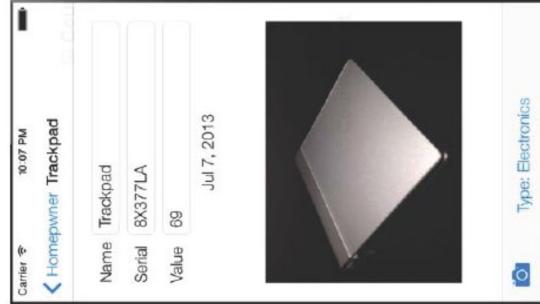
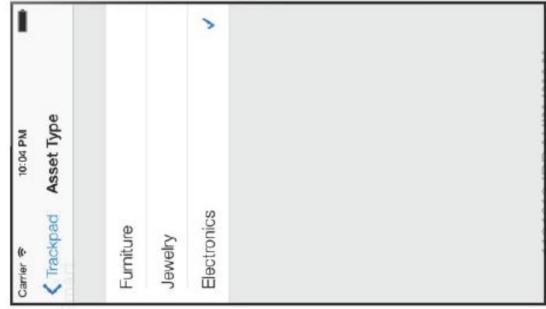
- We need to create the missing entity `INIAssetType` and to set the asset type for items in our `INItems` entity.
- `INItemStore` needs to be able to retrieve assets' types

## Modify `INItemStore`:

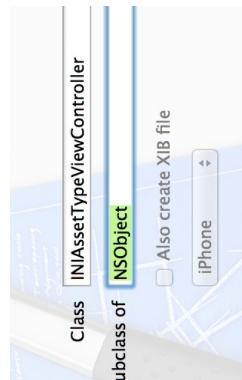
```
15 - (NSArray *)allAssetTypes
16 {
17     if (!_allAssetTypes) {
18         NSFetchRequest *request = [[NSFetchRequest alloc] init];
19         NSEntityDescription *entityForName = [NSEntityDescription entityForName:@"INIAssetType"
20                                             inManagedObjectContext:self.context];
21         request.entity = entity;
22         NSError *error = nil;
23         NSArray *result = [self.context executeFetchRequest:request error:&error];
24         if (!result) {
25             [NSErrorException raise:@"Fetch failed"
26             format:@"Reason: %@", [error localizedDescription]];
27         }
28     }
29     _allAssetTypes = [result mutableCopy];
30 }
31
32 // Is this the first time the program is being run?
33 if ([_allAssetTypes count] == 0) {
34     NSManagedObject *type;
35
36     type = [NSEntityDescription insertNewObjectForEntityForName:@"INIAssetType"
37                                         inManagedObjectContext:self.context];
38     [type setValue:@"Furniture" forKey:@"label"];
39     [_allAssetTypes addObject:type];
40
41     type = [NSEntityDescription insertNewObjectForEntityForName:@"INIAssetType"
42                                         inManagedObjectContext:self.context];
43     [type setValue:@"Jewelry" forKey:@"label"];
44     [_allAssetTypes addObject:type];
45
46     type = [NSEntityDescription insertNewObjectForEntityForName:@"INIAssetType"
47                                         inManagedObjectContext:self.context];
48     [type setValue:@"Electronics" forKey:@"label"];
49     [_allAssetTypes addObject:type];
50
51 }
52
53 return _allAssetTypes;
54 }
```

# Creating a view to list assets type

- We want to create a table view controller to show a list of the available asset types.
- This view will be shown when we tap a button on the `INIDetailViewController`'s view.



```
9 #import <Foundation/Foundation.h>
0 @class NIItem;
1
2 @interface INIAssetTypeViewController : UITableViewController
3
4 @property (nonatomic, strong) NIItem *item;
5
6 @end
```



## INIAssetTypeViewController

```

#import "INIAssetTypeViewController.h"
#import "INIAssetStore.h"
#import "INIAsset.h"

@implementation INIAssetTypeviewController

- (instancetype)init
{
    return [super initWithStyle:UITableViewStylePlain];
}

- (instancetype)initWithStyle:(UITableViewStyle)style
{
    return [self init];
}

- (void)viewDidLoad
{
    [super viewDidLoad];
}

- (void)viewDidUnload
{
    [super viewDidUnload];
}

- (void)viewDidLoad
{
    [super viewDidLoad];
}

- (void)tableView:(UITableView *)tableView numberOfRowsInSection:(NSInteger)section
{
    NSInteger *tableview = (NSInteger)tableView numberOfRowsInSection:(NSInteger)section;
    if (tableview == nil)
        return [[[INIAssetStore sharedStore] allAssetTypes] count];
}

- (UITableViewCell *)tableView:(UITableView *)tableView cellForRowAtIndexPath:(NSIndexPath *)indexPath
{
    UITableViewCell *cell = [tableView dequeueReusableCellWithIdentifier:@"UITableViewCell"];
    if (cell == nil)
        cell = [[UITableViewCell alloc] initWithStyle:UITableViewCellStyleDefault reuseIdentifier:@"UITableViewCell"]];
}

- (void)tableView:(UITableView *)tableView didSelectRowAtIndexPath:(NSIndexPath *)indexPath
{
    NSManagedObject *asset = [[[INIAssetStore sharedStore] allAssetTypes] objectAtIndex:indexPath.row];
    NSString *assetLabel = [asset valueForKey:@"label"];
    cell.textLabel.text = assetLabel;
    cell.accessoryType = asset.accessoryType;
    if (asset.accessoryType == self.item.assetType)
        cell.accessoryType = UITableViewCellAccessoryCheckmark;
    else
        cell.accessoryType = UITableViewCellAccessoryNone;
}

- (void)tableView:(UITableView *)tableView didSelectRowAtIndexPath:(NSIndexPath *)indexPath
{
    UITableViewCell *cell = [tableView cellForRowAtIndexPath:indexPath];
    cell.accessoryType = UITableViewCellAccessoryCheckmark;
}

- (void)tableView:(UITableView *)tableView didSelectRowAtIndexPath:(NSIndexPath *)indexPath
{
    NSManagedObject *asset = [[[INIAssetStore sharedStore] allAssetTypes] objectAtIndex:indexPath.row];
    self.item.assetType = assetType;
    [self.navigationController popViewControllerAnimated:YES];
}

```

# Connecting Item's Details to Asset Type

- In INIDetailViewController.xib, we drag a UIBarButtonItem onto the toolbar.
  - Create an outlet named assetTypeButton to this button by Control-dragging to the class extension of INIDetailViewController.m.
  - Create an action from this button in the same way by dragging implementation section and name it showAssetTypePicker.

```
22 @implementation INIDetailViewController
23 @property (weak, nonatomic) IBOutlet UIImageView *imageView;
24 @property (weak, nonatomic) IBOutlet UIToolbar *toolbar;
25 @property (weak, nonatomic) IBOutlet UIBarButtonItem *cameraButton;
26 @property (strong, nonatomic) UIPopoverController *imagePickerController;
27 @property (weak, nonatomic) IBOutlet UIBarButtonItem *assetTypeButton;
28
29 @end
30
31 @implementation INIDetailViewController
32 - (IBAction)showAssetTypePicker:(id)sender {
33 }
34
```

# Finally

```
④ 34 - (IBAction)showAssetTypePickerController:(id)sender
④ 35 {
④ 36 [self.view endEditing:YES];
④ 37
④ 38 INITAssetTypeViewController *avc = [[INITAssetTypeViewController alloc] init];
④ 39 avc.item = self.item;
④ 40
④ 41 [self.navigationController pushViewController:avc
④ 42 animated:YES];
④ 43 }
```

The screenshot shows an iPhone X displaying an application interface. At the top, there is a navigation bar with the text "Carrier" and a signal icon on the left, and "1:37 PM" on the right. Below the navigation bar, the title "Homeowner" is displayed. On the left side of the screen, there is a blue button labeled "Edit". The main content area displays a table with two rows. The first row contains the text "item1inCoreData" and "coredata\_01" followed by a price of "\$ 2". The second row contains a small image of a red high-heeled shoe, the text "Furniture", and a blue circular icon with a white question mark. To the right of the main content area, there is a sidebar with three sections: "Furniture" (with a red high-heeled shoe icon), "Jewelry" (with a diamond ring icon), and "Electronics" (with a laptop icon). A blue arrow points from the sidebar towards the main content area.