Stanford CS193p

Developing Applications for iPhone 4, iPod Touch, & iPad

Fall 2010

Today

- More Core Data
  - What does the code for the custom NSManagedObject subclasses generated by Xcode look like?
  - Querying for (fetching) objects via NSFetchRequest.
- Core Data and Table Views
  - NSFetchedResultsController (hooking up your Core Data objects to a UITableView)
  - CoreDataTableViewController (plugs NSFetchedResultsController into a UITableViewController)
- Demo
  - Core Data
  - Using an NSFetchedResultsController to drive a UITableView

Core Data

- What does the generated code look like in a header file?
  Easy, it's just `@property` entries for each attribute.
  (Relationships will have some extra code (inside `#ifdef 0`) which you can delete.)

```
@interface Photo : NSManagedObject {
}
@property (nonatomic, retain) NSString * thumbnailURL;
@property (nonatomic, retain) NSData * thumbnailData;
@property (nonatomic, retain) NSManagedObject * whoTook;
@end

@interface Photographer : NSManagedObject {
}
@property (nonatomic, retain) NSSet * photos;
@end
```

Core Data

- How about on the implementation side?

  New Objective-C keyword you have not seen before ... `@dynamic`.

```
@implementation Photo

@dynamic thumbnailURL;
@dynamic thumbnailData;
@dynamic whoTook;

@end
```

(Ditto about deleting the `#ifdef 0` code for now, it's not necessary for most situations.)

`@dynamic` means "my class will figure out how to respond to this at runtime."
Uses a runtime mechanism for an object to intercept messages it normally wouldn't respond to.
In this case, `NSManagedObject` turns these into calls to `valueForKey:`/`setValueForKey:`.

The bottom line is that you can use property dot notation to access your database Entity, e.g.,

```
Photo *photo = [NSEntityDescription insertNewObjectForEntityForName:@"Photo" inManage...];
NSString *myThumbnail = photo.thumbnailURL;
photo.thumbnailData = [FlickrFetcher imageDataForPhotoWithURLString:myThumbnail];
photo.whoTook = ...; // a Photographer object we created or got by querying
```

Core Data

- So far you can ...
  Create objects in the database with insertNewObjectForEntityForName:inManagedObjectContext:
  Get/set properties with valueForKey:/setValueForKey: or @propertys in a custom subclass.

- 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

- Four important things involved in creating an NSFetchRequest
  1. NSEntityDescription of which Entity to fetch (required)
  2. NSPredicate specifying which of those Entities to fetch (optional, default is all of them)
  3. NSSortDescriptors to specify the order of objects in the returned array (optional, random)
  4. How many objects to fetch at a time and/or maximum to fetch (optional, all)

Core Data

- Creating an NSFetchRequest

  We'll consider each of these lines of code one by one ...

```
NSFetchRequest *request = [[NSFetchRequest alloc] init];
request.entity = [NSEntityDescription entityForName:@"Photo" inManagedObjectContext:ctxt];
request.fetchBatchSize = 20;
request.fetchLimit = 100;
request.sortDescriptors = [NSArray arrayWithObject:sortDescriptor];
request.predicate = ...;
```

- Getting a description of the kind of Entity we want to fetch

```
request.entity = [NSEntityDescription entityForName:@"Photo" inManagedObjectContext:ctxt];
```

  We are just asking the class NSEntityDescription to create an instance to describe the Entity.

- Setting fetch sizes/limits

  If you created a fetch that would match 1000 objects, the request above faults 20 at a time.
  And it would stop fetching after it had fetched 100 of the 1000.

Core Data

- NSSortDescriptor

  When we execute our fetch request, it's going to return an NSArray of NSManagedObjects.
  NSArrays are "ordered," so we usually want to specify the order when we fetch.
  We do that by giving the fetch request a list of "sort descriptors" that describe what to sort by.

```
NSSortDescriptor *sortDescriptor =
    [[NSSortDescriptor alloc] initWithKey:@"thumbnailURL"
                                ascending:YES
                                 selector:@selector(localizedCaseInsensitiveCompare:)];
```

  There's another version with no selector: argument (default is the method compare:).
  The selector: argument is just a method (conceptually) sent to each key to compare it to others.
  Some of these "methods" might be smart (i.e. they can happen on the database side).

  There are also class methods that return an autoreleased descriptor (we use those most often).

  We give a list of these to the NSFetchRequest because sometimes we want to sort first by one key (e.g. last name), then, within that sort, sort by another (e.g. first name).

Core Data

- NSPredicate

  This is the guts of how we specify exactly which objects we want from the database.
  Creating one looks a lot like creating an NSString, but the contents have semantic meaning.

```
NSString *serverName = @“flickr-5”;
NSPredicate *predicate =
    [NSPredicate predicateWithFormat:@“thumbnailURL contains %@”, serverName];
```

- What can this predicate format look like?

  Very powerful. Examples of predicateWithFormat: arguments …

```
@“uniqueId == %@”, [flickrInfo objectForKey:@“id”]
@“%@ in tags”, (NSManagedObject *)    // tags is a to-many relationship
@“viewed > %@”, (NSDate *)            // viewed is a Date attribute in the data mapping
@“name contains[c] %@”, (NSString *)  // matches the string in name attribute case insensitively
```

  Many more options. Look at the class documentation for NSPredicate.

Core Data

- NSCompoundPredicate

  We can also combine predicates with ands and ors.

```
NSArray *array = [NSArray arrayWithObjects:predicate1, predicate2, nil];
NSPredicate *predicate = [NSCompoundPredicate andPredicateWithSubpredicates:array];
```

  The predicate is "predicate1 AND predicate2". Or also available, of course.

Core Data

- Putting it all together

```
// Let's say Photographer is a custom NSManagedObject subclass
// And that we've implemented photographerWithName:inManagedContext: to get one from the db
NSManagedObject *photographer = [Photographer photographerWithName:@"George" inManage...];
// (we said NSManagedObject *photographer but we could have said Photographer *photographer)

// Now let's create a fetch request to find all photos this photographer has taken
NSFetchRequest *request = [[NSFetchRequest alloc] init];
request.entity = [NSEntityDescription entityForName:@"Photo" inManagedObjectContext:ctxt];
request.fetchBatchSize = 20;
request.sortDescriptors =
    [NSArray arrayWithObject:[NSSortDescriptor sortDescriptorWithKey:@ "title" ascen...]];
request.predicate =
    [NSPredicate predicateWithFormat:@"whoTook = %@", photographer];
```

Core Data

- So how do we actually get this fetch request to happen?

  We use the method `executeFetchRequest:error:` in `NSManagedObjectContext`

```
NSFetchRequest *request = ...;
NSError **error = nil;
NSArray *results = [mangedObjectContext executeFetchRequest:request error:&error];
```

  Returns `nil` if there is an error (check the `NSError` for details).
  Returns an empty array (not `nil`) if there are no matches in the database.
  Returns an array of `NSManagedObjects` (or subclasses thereof) if there were any matches.
  You can pass `NULL` for `error:` if you don't care why it fails.

- Examples

  Assuming the `results` above ...

```
NSManagedObject *photo = [results objectAtIndex:0];
for (Photo *photo in results) { ... }
Photo *photo = [results lastObject];   // convenient if fetch should return 0 or 1 object only
```

  If you assign the objects from `results` to a `Photo *`, be sure you're fetching in the Photo Entity!

Core Data

- Deleting objects

  Simple (sort of). `[managedObjectContext deleteObject:(NSManagedObject *)anObject];`
  There are considerations when objects have relationships to each other.
  E.g., what if I delete the last Photo that a Photographer has taken? Delete the Photographer too?
  There are settings to control this (check out "Creating and Deleting Managed Objects" in the doc).
  For your homework, you do not have to delete any objects if you don't want to.

- 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?)
  Observing changes (like NSFetchedResultsController is doing automatically for you)
  Overriding value setting/getting in custom subclasses (that's what the `#ifdef 0` code is about).

NSFetchedResultsController

- Hooks an NSFetchRequest up to a UITableView

  It can answer all the "questions" the UITableView's dataSource protocol asks.

- Examples

```
- (NSInteger)numberOfSectionsInTableView:(UITableView *)tableView {
    return [[fetchedResultsController sections] count];
}

- (UITableViewCell *)tableView:(UITableView *)tv cellForRowAtIndexPath:(NSIndexPath *)ip
{
    UITableViewCell *cell = ...;
    NSManagedObject *managedObject = [fetchedResultsController objectAtIndexPath:ip];
    // load up the cell based on the properties of the managedObject
    // of course, if you had a custom subclass, you'd be using dot notation to get them
    return cell;
}
```

NSFetchedResultsController

- It also "watches" changes in Core Data and auto-updates table

  It does this via its own delegate, sending messages like:

```
- (void)controller:(NSFetchedResultsController *)controller
    didChangeObject:(id)anObject
    atIndexPath:(NSIndexPath *)indexPath
    forChangeType:(NSFetchedResultsChangeType)type
    newIndexPath:(NSIndexPath *)newIndexPath
{
    // here you would call appropriate UITableView methods to update rows
}
```

CoreDataTableViewController

- How to use all this functionality is shown in its documentation
  All you need to do is copy/paste the example code shown there into your UITableViewController
- But that's a bit of a pain
  So ... let us copy/paste that code for you!
  We've done that to create `CoreDataTableViewController`.
  Just connects an `NSFetchedResultsController` to a `UITableViewController`.
  Download it along with your homework assignment.
  Familiarize yourself with its API (it just calls methods you override and has properties you set).
- Easy to use
  Be sure to set its `fetchedResultsController` property in the initializer for your subclass of it.
  Set the appropriate properties which say which key to use in your Entity as the title/subtitle.
  Override the method that gets called when a row is selected to do your pushing or whatever.

NSFetchedResultsController

- How do you create an NSFetchedResultsController?
  Just need the fetch request that is going to drive it.
  Can also specify a key in your Entity which says which section each row is in.

```
NSSortDescriptor *sortDescriptor = [NSSortDescriptor sortDescriptorWithKey:@“title” ascen...];
NSFetchRequest *request = [[NSFetchRequest alloc] init];
request.entity = [[NSEntityDescription entityForName:@“Photo” inManagedObjectContext:context];
request.sortDescriptors = [NSArray arrayWithObject:sortDescriptor];
request.predicate = [NSPredicate predicateWithFormat:@“whoTook = %@”, photographer];
request.fetchBatchSize = 20;
NSFetchedResultsController *frc = [[NSFetchedResultsController alloc]
    initWithFetchRequest:(NSFetchRequest *)request
    managedObjectContext:(NSManagedObjectContext *)context
      sectionNameKeyPath:(NSString *)keyThatSaysWhichSectionEachManagedObjectIsIn
               cacheName:@“MyPhotoCache”;   // if not nil, don’t reuse frc or modify request
[request release];
```

Coming Up

- Demo
  - Fetch a list of photos from Flickr
  - Display a table view full of the photographers who took those photos
  - Push a list of that photographer's photos when the photographer is clicked on
- Homework
  - Add a Favorites tab to your Places application
  - Use Core Data to store your Favorites information (and your Recents)
  - Cache the user's favorite photos' image data in the file system
- Next Week
  - Blocks and Multithreading
  - Final Project Guidelines