Assignment VI:

Core Data Places

Objective

We continue working on our Places application this week. The primary goal this time is to get experience with creating a Core Data model and then creating objects in the associated database and querying the database via `NSFetchedResultsController` in a table view environment.

You'll do this by adding a new "Favorites" tab to your application and by reimplementing your Recents tab to use the database instead of `NSUserDefaults`.

Even though this application is substantially similar to last week's it is recommended that you start fresh with a completely new application because your underlying data structure is will be completely different. You will still want to drag some of your classes from last week in on occasion (like your view controller that shows a photo in a scrollview and your view controller(s) that showed the top places from Flickr and the list of photos in a top place).

Be sure to check out the Hints section below!

Also, check out the latest in the Evaluation section to make sure you understand what you are going to be evaluated on with this assignment.

Materials

- The class `CoreDataTableViewController` is provided.
- You will need your Flickr API key & the `FlickrFetcher/JSON` code from last week.

Required Tasks

1. Create a Core Data-managed object model in Xcode to store all the information from Flickr that you feel you will need to do all the rest of the required tasks in this assignment. You must give some thought to what entities will be in the model and what attributes and relationships those objects will have so that you can accomplish the rest of the required tasks.
2. Add a button in the existing part of your user-interface that shows an photo (i.e. the one that displays an image in a scrollview) to mark that photo as "a favorite." The button should toggle so that if the photo is already a favorite, the button makes it stop being a favorite.
3. Add a tab to your tab bar controller for Favorites. This tab should display a list of **PLACES** which, when clicked on, will show you a list of the favorite photos taken in that place (as chosen by the user with the user-interface described in Required Task #2 above). The user can then click on one of the photos in this list to see the photo again.
4. Store all of your recents information in the Core Data database and **not** in `NSUserDefaults`. The list still needs to be in chronological order (most recent first) of viewing. It should not show photos viewed more than 48 hours ago.
5. Cache the image data returned from Flickr for your favorites (only) in your application's sandbox (i.e. the file system). Do not store your image data in Core Data. Do not waste cache space on photos that are not marked as favorites.
6. Make sure your list of places in your Top Places tab (from last assignment) is sorted alphabetically by the name of the place (you may have already done this last week, in which case, you have less work to do this week!).

Hints

1. When you create your New Project ... for this assignment, be sure to click the box that says "Use Core Data for storage." This will give you the very important application delegate property `managedObjectContext` which you will need to create and query objects from the database.
2. Hint #1 will also create a (blank, obviously) `.xcdatamodel` file for you in your Resources folder.
3. Do not use an attribute in your data model named `description`. This will be tempting, but it will cause problems (because of `NSObject`'s `description` method).
4. Do not use an attribute in your data model named `id`. You can imagine the problems this might cause in Objective-C.
5. While you will have to reimplement your Recents tab for this assignment, you will not have to reimplement your Top Places tab (though it might require some minor modification).
6. Remember that in order to load up a table view with objects of a certain type from Core Data, there must be a representation for that object (an Entity) in the managed object model you create in Xcode's data modeler. Or, in database terms, there must be a "table" for it. For example, you cannot fetch a list of photos into a table view if there is no Entity that represents a photo in the database.
7. If you change your object model (and you will likely do that numerous times as you iterate on your schema), be sure to delete your application from your device or simulator before running it with a new schema so that your old database (with the old schema) gets deleted. You do this by pressing and holding on the application icon on the home screen of the device or simulator until it jiggles, then pressing the X that appears in the corner of the icon. If you fail to do this when you change your schema, your program will crash (with complaints in the console about incompatible database descriptions).
8. It is generally recommended to create custom subclasses for each of your Entities. You can do this using the New File ... menu item (as described in class) once you have your database model created. This subclass is a pretty good place to put a class method that creates one of your Entity in the database using `NSEntityDescription`'s `insertNewObjectForEntityForName:inManagedObjectContext:` method (but you should first check to make sure the Entity you want doesn't already exist with a call to `executeFetchRequest:error:` or you might end up with lots of duplicate objects).
9. Modify your scrolling photo-viewing controller to use a photo object from the database as its Model instead of a dictionary of Flickr information. You will thus

need to be sure the database representation of a photo has the information needed by your photo-viewing controller.

10. You'll also need to modify your Top Places code (only slightly) so that when it is time for it to ask for a photo to be shown by your photo-viewing controller, it has all the information it needs to create a photo object in the database. Namely an `NSManagedObjectContext` and a place object in the database that it can use to create the proper relationship between a photo and the place it was taken.
11. Even though you display a list of photos in Top Places (when a place is chosen) and in Favorites (when a place with favorite photos is chosen), these will be different implementations because the Top Places list of photos is an array of dictionaries from Flickr and the Favorites list of photos is a list of photos from your database. Keep the code from last week for Top Places (modified slightly as described above).
12. The `CoreDataTableViewController` class's implementation is mostly just copy/pasted from the documentation of `NSFetchedResultsController`. It's very easy to subclass and use. Usually there are only 3 things you need to do:
    a. Create an `NSFetchedResultsController` for it and set it using the `fetchedResultsController` property.
    b. Set the keys that are going to represent the `title` and (if appropriate) the `subtitle`. These are the names of attributes in your `NSManagedObject`.
    c. Push an appropriate view controller onto the navigation stack when a given `NSManagedObject` is selected from the table (`managedObjectSelected:`).
13. There are a few other things a subclass of `CoreDataTableViewController` can do. Check out the header file. You will only need those extra things if you are doing the extra credit.
14. You will need at most three `CoreDataTableViewController` subclasses for this assignment (`Recents`, `FavoritePlaces` and `FavoritePhotosInAFavoritePlace`). You might be able to get away with just two (i.e. your `Recents` and `FavoritePhotosInAFavoritePlace` controllers are awfully similar--they vary only by a predicate here and a sort descriptor there) or you might want to have three with one of the three being a subclass of one of the others. How you arrange this code is up to you, but try to apply the best possible object-oriented practices that you know.
15. Your Recents tab's view controller is sorted by "when a photo in the database was last viewed," so it might be a good idea to have an attribute for that in your schema and make sure it gets set properly.
16. It is probably also a good idea to have an attribute somewhere in the database which marks that a given place has some (i.e. at least one) favorite photo(s). You should set it anytime a photo from that place gets marked as a favorite. You'll need to use an inverse relationship to make sure that this attribute gets cleared in a place when the last favorite photo in that place stops being a favorite. Inverse relationship properties

are just `NSSet`s of `NSManagedObject`s (or subclasses thereof). The code for this should probably go in a custom subclass of one of your `NSManagedObject`s (i.e., create methods to set and clear the favorite status of a photo and have them do the right thing in the corresponding place).

17. If you find yourself writing a lot of code to make your table views work, then you are probably headed down the wrong path. Let `CoreDataTableViewController` do all the work for you. You probably only need two methods in each of your subclasses of `CoreDataTableViewController` (initialization and handling clicks in the table).
18. Recall that you can use "dot notation" in your `NSPredicate` to query attributes of objects through a relationship in the database. For example, in a photo, `place.name` might represent the name of the place the photo was taken (assuming `place` is a relationship in the database between the photo and the place it was taken, and assuming `name` is an attribute in place objects in the database).
19. Your sorting in all of your Core Data-driven tables should happen automatically as part of an appropriately created `NSFetchRequest`. Just make sure it has the right `NSSortDescriptor`.
20. Note that whenever you change your Core Data database (e.g. you mark an object as a favorite) things will automatically update in the entire user-interface without your having to do anything. That is the wonder of `NSFetchedResultsController` and Core Data's use of the key-value observing mechanism. Enjoy it!
21. Don't forget that even after you've created a custom subclass for an Entity in the database, as you continue to edit your data model, Xcode provides a way to "copy and paste" the declarations from the data model editor. Just select the attributes you want to copy/paste in the data model editor, then click the menu item Design -> Data Model -> Copy Obj-C 2.0 Method Declarations to Clipboard, then go paste them into your `Photo.h`. Then do it again with Copy Obj-C 2.0 Method Implementation to Clipboard for your `Photo.m` file. The main thing that will get copied and pasted is the @property for the attribute and the @dynamic in the implementation. It might copy some other code that is #ifdef 0'd out which (as the comments will say) you can just delete.
22. If you keep your custom subclass's `@property` and `@dynamic` code up to date as you modify your data model, then you can access the attributes of your database objects using Objective-C dot notation. Of course, to do this, you will have to cast any `NSManagedObject` you are handed to your custom subclass. For example, `Photo *photo = (Photo *)selectedManagedObject`.

Evaluation

In all of the assignments this quarter, writing quality code that builds without warnings or errors, and then testing the resulting application and iterating until it functions properly is the goal.

Here are the most common reasons assignments are marked down:

- Project does not build.
- Project does not build without warnings.
- One or more items in the Required Tasks section was not satisfied.
- A fundamental concept was not understood.
- Code is sloppy and hard to read (e.g. indentation is not consistent, etc.).
- Assignment was turned in late (you get 3 late days per quarter, so use them wisely).
- Code is too lightly or too heavily commented.
- Code crashes.
- Code leaks memory!

Extra Credit

If you do any Extra Credit items, don't forget to note what you did in your submission `README`.

1. Allow users to "delete"items from the Recents tab and the Favorites tab. Don't worry about actually deleting objects from the database, just modify the objects so that they no longer satisfy the `NSPredicate` the table view is using. Again, you don't need to do anything other than change the attribute. Updating of the database and the table view will happen automatically. Check out `CoreDataTableViewController`'s header file. This is a pretty easy one.
2. Enhance the Favorites tab to have a button which shows "Only Favorites" versus "All Places" (i.e. all places that the user has ever viewed a photo in, regardless of whether they marked it as a favorite). Don't go so far as to store all the places that come back as `topPlaces` in your database though, that'll be a waste of database space and clutter in the user's interface. Just show the ones the user was interested enough in to click on at least one photo there.
3. Enhance one or more of your `CoreDataTableViewController` subclasses to allow searching. This can be done with one line of code using the `searchKey` property in `CoreDataTableViewController`.
4. Make your application work on the iPad with appropriate user-interface idioms on that platform.