Stanford CS193p

Developing Applications for iOS
Fall 2011

Today

- Core Data Thread Safety
  `NSManagedObjectContext` is not thread-safe.
  What to do about that.
- Core Data and Table View
  Very common way to view data from a Core Data database
  So it is made very easy with NSFetchedResultsController
- Big ol' demo
  Photomania
  Browse recent photos by Photographer who took them

Core Data Thread Safety

- NSManagedObjectContext is not thread safe

  Luckily, Core Data access is usually very fast, so doing it in the main thread is mostly fine.
  Always safe to access from the thread in which it (or its UIManagedDocument) was created.
  Feel free to take this approach for your homework (it's the most straightforward).

- Another Approach

  ```
  [context performBlock:^{   // or performBlockAndWait:
      // do stuff with context
  }];
  ```

  This will make sure that the code in the block happens on the context's safe thread.
  Note that this might well be the main thread, so you're not necessarily getting "multithreaded."

- Advanced Approach

  Some contexts (including Core Data ones) have a parentContext (a @property on NSMOC).
  The parentContext will almost always have its own thread, so you can performBlock: on it.
  But it is a different context, so you'll have to save and then refetch to see the changes.

Core Data and Table View

- NSFetchedResultsController

  Hooks an NSFetchRequest up to a UITableViewController

  NSFetchedResultsController can answer all of the UITableViewDataSource protocol's questions!

- For example ...

```
- (NSUInteger)numberOfSectionsInTableView:(UITableView *)sender
{
    return [[self.fetchedResultsController sections] count];
}
- (NSUInteger)tableView:(UITableView *)sender numberOfRowsInSection:(NSUInteger)section
{
    return [[[self.fetchedResultsController sections] objectAtIndex:section] numberOfObjects];
}
```

NSFetchedResultsController

- Very important method … objectAtIndexPath:

```
- (NSManagedObject *)objectAtIndexPath:(NSIndexPath *)indexPath;
```

Here's how you would use it in, for example, tableView:cellForRowAtIndexPath:

```
- (UITableViewCell *)tableView:(UITableView *)sender
         cellForRowAtIndexPath:(NSIndexPath *)indexPath
{
    UITableViewCell *cell = …;
    NSManagedObject *managedObject =
        [self.fetchedResultsController objectAtIndexPath:indexPath];
    // 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

- How do you create an NSFetchedResultsController?

  Just need the NSFetchRequest to drive it (and a NSManagedObjectContext to fetch from).

```
NSFetchRequest *request = [NSFetchRequest fetchRequestWithEntityName:@“Photo”];
NSSortDescriptor *sortDescriptor = [NSSortDescriptor sortDescriptorWithKey:@“title” …];
request.sortDescriptors = [NSArray arrayWithObject:sortDescriptor];
request.predicate = [NSPredicate predicateWithFormat:@“whoTook.name = %@”, photogName];
NSFetchedResultsController *frc = [[NSFetchedResultsController alloc]
    initWithFetchRequest:(NSFetchRequest *)request
    managedObjectContext:(NSManagedObjectContext *)context
      sectionNameKeyPath:(NSString *)keyThatSaysWhichSectionEachManagedObjectIsIn
               cacheName:@“MyPhotoCache”;   // careful!
```

  Be sure that any cacheName you use is always associated with exactly the same request.
  It's okay to specify nil for the cacheName (no cacheing of fetch results in that case).

  It is critical that the sortDescriptor matches up with the keyThatSaysWhichSection…
  The results must sort such that all objects in the first section come first, second second, etc.

NSFetchedResultsController

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

  Uses a key-value observing mechanism.

  When it notices a change, it sends message like this to its delegate …

```
- (void)controller:(NSFetchedResultsController *)controller
   didChangeObject:(id)anObject
       atIndexPath:(NSIndexPath *)indexPath
     forChangeType:(NSFetchedResultsChangeType)type
      newIndexPath:(NSIndexPath *)newIndexPath
{
    // here you are supposed call appropriate UITableView methods to update rows
    // but don't worry, we're going to make it easy on you …
}
```

CoreDataTableViewController

- NSFetchedResultsController's doc shows how to do all this
  In fact, you're supposed to copy/paste the code from the doc into your table view subclass.
  But that's all a bit of a pain, so ...
- Enter CoreDataTableViewController!
  We've copy/pasted the code from NSFetchedResultsController into a subclass of UITVC for you!
- How does CoreDataTableViewController work?
  It's just a UITableViewController that adds an NSFetchedResultsController as a @property.
  Whenever you set it, it will immediately start using it to fill the contents of its UITableView.
- Easy to use
  Download it along with your homework assignment.
  Just subclass it and override the methods that load up cells and/or react to rows being selected
  (you'll use the NSFetchedResultsController method objectAtIndexPath: mentioned earlier).
  Then just set the fetchedResultsController @property and watch it go!

Demo

- Photomania

  Gets recent photos from Flickr.
  Shows a list of photographers who took all the photos.
  Select a photographer -> shows a list of all the photos that photographer took.
  Core Data Entities: `Photographer` and `Photo`.

- Watch for ...

  How we open/create a `UIManagedDocument` to hold our Core Data database.
  How we define our database schema graphically in Xcode.
  How we create `NSManagedObject` subclasses and then add categories to them.
  Especially how we use categories to create "factory" methods to create/initialize database objects.
  How we use `CoreDataTableViewController` to hook the table views up to the database.

Coming Up

- **Homework**

  Virtual Vacation

  Let the user create a "virtual" vacation and then go on a vacation any time they want!

  Builds on your Fast Map Places application from last week

- **Friday Section**

  Mike Ghaffary

  Director of Business Development at Yelp!

  Also co-founder of BarMax, the most expensive iPhone/iPad app on the AppStore

  Topic: Building Apps that People Want

  - Understanding Market Opportunity
  - Building a Prototype
  - Financing a Company or Team
  - Getting User Feedback
  - Distribution through the AppStore