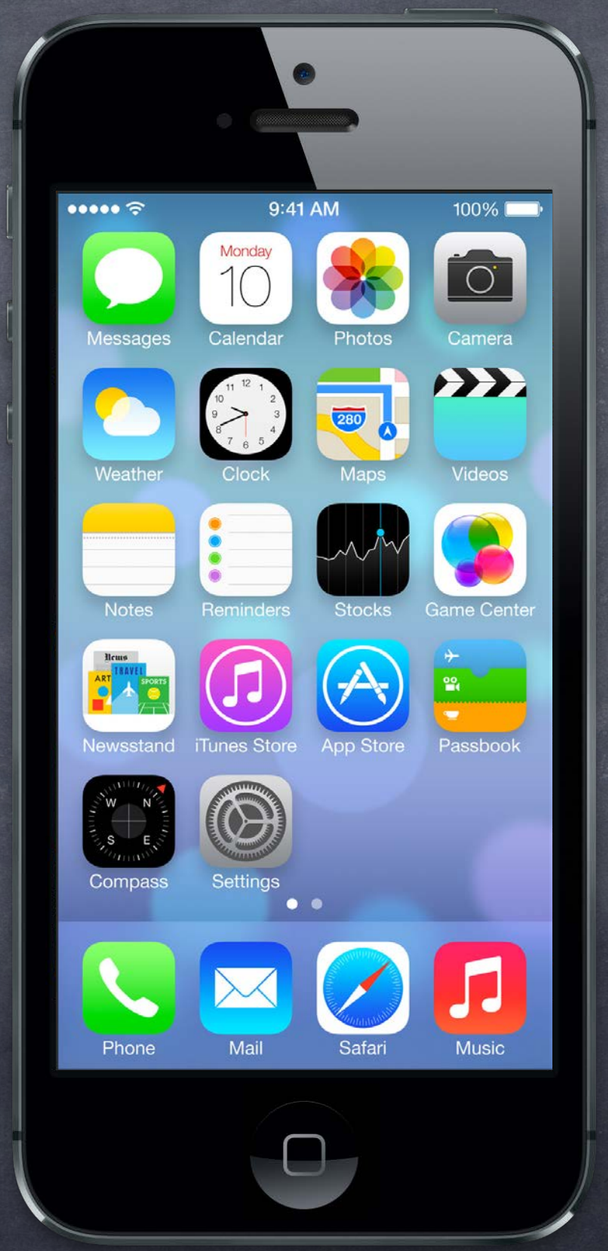
# Stanford CS193p

## Developing Applications for iOS
## Fall 2013-14



# Today

- Final Project Requirements
- Core Data and `UITableView`
- Core Data Demo

  Photomania

# Final Project

- Proposal due immediately!
  - And must be received no later than next Wednesday.
  - Send PDF of your proposal to your CA (the one who has graded your latest assignment).
  - Proposal must say not only what you are doing, but also what parts of SDK will be featured.
- Project (including Keynote) due on Friday, December 6th.
  - Use normal submission process (put the keynote file at the top level where README is).
  - NO LATE DAYS (last two assignments are the last opportunity to use free late days).
- Required presentation during final exam period
  - Thursday, December 12th at 12:15pm in this room.
  - 2-minute Keynote (not PowerPoint) presentation (more on this in a moment).
  - 1280x720 aspect ratio (not 1024x768 or 800x600).

  - Alternate presentation time on Thursday, December 5th (w/Keynote due by Tuesday, December 3rd).
  - If you need/want the alternate presentation time, let us know immediately (via class staff e-mail).

# Final Project

- Scope is the same as about three weeks of homework
  - Luckily, you'll have about three weeks to do it (counts as approximately 35% of your overall grade).
  - P/NC students must pass both homework and final project segments separately.
- Must work on hardware!
  - Bring your hardware to final exam to demo to TA (if not used during your presentation).
  - iPad or iPhone or iPod Touch okay.
- Only iOS SDK code "counts"
  - Don't waste your time writing server-side code
  - Okay to "simulate" a server-side interaction to make your code demonstrable.

# Final Project

- You'll be graded on proper use of SDK
  - Hackery will count against you. Use good object-oriented programming technique.
  - Must have at least one feature which was NOT taught in lecture/demo/homework assignment.
  - Breadth is VERY important. Don't get stuck down a rathole.
  - Only need to show depth in one or two areas. Breadth is more important.
- Aesthetics of your user-interface matter
  - (although we do not expect professional graphic designer quality graphics)
  - Sloppy layouts will be graded down.
  - Lots of places to get graphics from on the internet.
- Be careful not to get side-tracked on non-iOS-code
  - Some students in the past have spent 80% of their time working on stuff that didn't demonstrate their mastery of the class material.
  - (e.g. preparing some large database or working on graphics too much, etc.)
  - In the end, this is an iOS PROGRAMMING course, so we want to see how well you can program on this platform.

# Final Project

- Presentation Quality Matters
  - A (tiny) portion of your grade will be related to the quality of your presentation.
  - Not okay to just put up a recording of you or of your application and say nothing.
  - Being able to make a live presentation is a valuable skill.
  - Practice your presentation before you show up.
  - You only get 2 minutes (strictly enforced), so make 'em count.
- Live demo?
  - All iOS 7 devices (iPad2+, iPhone4S, iPhone5) can mirror their screen to the projector here.
  - Live demos are perilous, as you saw all quarter :), but effective!
  - You must, at worst, show screen shots of your application.
  - Keynote/Quicktime has some tools to "animate" screen shots (better than static).
  - Video (screen capture) of your app in action can be good also.

# Sample Proposal

- Section 1: What am I doing?

  I will be building a "Shakespeare Director" application.

  It will have the following features:

  - A table for choosing a Shakespearean play from a list downloaded from Folio*.
  - A custom view for laying out the blocking of a chosen Shakespearean play.
  - A dialogue-learning mode.

  * Folio is an on-line database of all of Shakespeare's works.

  The custom view will be simple (only rectangles and circles with colors for stroke/fill, and text).

  Photos (from Camera or Library) can be put in rectangles in the blocking view.

  The blocking can change from line to line in the dialog (but no more often than that).

  Blocking can be stepped through, line by line, or played back in "time lapse" mode.

  The dialogue-learning mode will step through all the dialog line by line.

  Users can record the dialog for other parts (as prompts for them to learn their own part).

  iPad only.

# Sample Proposal

- Section 2: What parts of iOS will it use?
  - `UITableView` for choosing plays and stepping through dialog
  - Custom `UITableViewCell` prototypes (for dialog, including speaker, blocking instructions)
  - Custom `UIView` with `drawRect:` for scene-setting
  - Camera/Photo Library for putting images in blocking rectangles
  - `UITextField` in a `UIPopoverController` for text labels in the scene-setting view
  - `UIPopoverController` for choosing stroke and fill color and shape in scene-setting mode
  - Scroll view to zoom in/pan around in blocking view
  - `AVFoundation` for record/playback of dialog
  - `NSTimer` for "time lapse playback" of entire play with dialog/blocking linked
  - Core Data to store the scene-setting and dialog
    - Play entity
    - Scene entity
    - BlockingElement entity
    - LineOfDialog entity
  - Printing of blocking to AirPrint printers (this is the NOT COVERED IN LECTURE feature)

# Sample Proposal

- What to notice about this sample proposal?
  - Clear description of what the application will do (section 1).
  - Clear list of the iOS features that will be used (section 2).
  - Lots of breadth (not necessarily that much depth in any one area).
  - Clearly delineates the NOT COVERED IN LECTURE feature.
  - Specifies platform (iPad only sacrifices breadth, but makes sense for this project).
  - It's creative (it's not just Matchismo or Top Places recycled).

# Core Data and UITableView

- How to hook these up
  - As you can imagine, they were (probably literally) made for each other!
  - The magic to doing this? `NSFetchedResultsController` …

# Core Data and UITableView

- NSFetchedResultsController

  Simply hooks an NSFetchRequest up to a UITableViewController

  Usually you'll have an NSFetchedResultsController @property in your UITableViewController.

  It will be hooked up to an NSFetchRequest that returns the data you want to show in your table.

  Then use it to answer all your 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:

NSFetchedResultsController method …

```
- (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 =  // or, e.g., Photo *photo = (Photo *) …
        [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).
  Let's say we want to show all photos taken by someone with the name photogName in our table:

```
NSFetchRequest *request = [NSFetchRequest fetchRequestWithEntityName:@“Photo”];
request.sortDescriptors = @[[NSSortDescriptor sortDescriptorWithKey:@“title” ...]];
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

- NSFRC 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 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.
  - The Application Delegate (finally!)
  - `NSManagedObjectContext`
  - Background Fetching
  - Background URL Sessions
  - `NSNotification` posting and listening
  - How we use `CoreDataTableViewController` to hook the table views up to the database.

# Coming Up

- Homework
  - Last one!
  - Due next Wednesday.
- Friday
  - Instruments (performance monitoring in Xcode).
  - (This is actually at risk. Watch Piazza for whether it's going to come together.)
- Next Week
  - More Multitasking
  - Advanced Segueing
  - Map Kit?