Announcements

Paul’s Office Hours
E-mail is best way to get help
Or in this room before and after Monday’s class

Sonali’s Office Hours Changed Again
Tuesday 10am to noon
Thursday 10am to noon

Andreas Taking Over Sonali’s Friday Time
Monday 6pm to 8pm
Friday 11am to 1pm
Homework

Submission Problems
Disk space was full, so half the class could not submit.
Make sure you delete your build directory before submitting.
Assignment 2 deadline moved to tonight at 11:59pm.
If it still fails, you can submit by e-mail this week.

Try to finish your Homework on Monday
There is no “new information” for homework in Monday’s lectures
In fact, Monday is “moving on to the next thing” day
Lots of “last minute” help on Monday (and now Tuesday morning)
Not so Tuesday night
Don’t be afraid to use e-mail to get help as well (but don’t depend
on getting a response to something sent Tuesday night on Tuesday).
Communication

E-mail
Questions are best sent to cs193p@cs.stanford.edu
Sending directly to instructor or TA's risks slow response.

Web Site
Very Important!
http://cs193p.stanford.edu
All lectures, assignments, code, etc. will be there.
This site will be your best friend when it comes to getting info.
Today’s Topics

- “Under the Hood” of “View-based Application”
- Application Lifecycle
- View Controller Lifecycle
- UINavigationController
- Demo Continuation
Like any C application, your app starts here.

Here's the code from main.m (found in Other Sources folder in Xcode window):

```c
#import <UIKit/UIKit.h>

int main(int argc, char *argv[]) {
    NSAutoreleasePool * pool = [[NSAutoreleasePool alloc] init];
    int retVal = UIApplicationMain(argc, argv, nil, nil);
    [pool release];
    return retVal;
}
```
main()

#import <UIKit/UIKit.h>

int main(int argc, char *argv[]) {

    NSAutoreleasePool * pool = [[NSAutoreleasePool alloc] init];
    int retVal = UIApplicationMain(argc, argv, nil, nil);
    [pool release];
    return retVal;
}

UIApplicationMain() creates a UIApplication object
(despite return value, it never returns).

The third argument is the name of the class of your UIApplication
(nil is the same as @"UIApplication")

The fourth argument is the name of the class which will be your
UIApplication’s delegate (nil means “none,” but it’s going to get set via
the application’s “MainWindow.xib” file--next slide).

We’ll talk about NSAutoreleasePool later (the one above does nothing).
What is this “application object” you speak of?

It is a shared object (singleton).
There is only ever one instance of it.
You can get ahold of it via ...

UIApplication *app = [UIApplication sharedApplication];

Has a lot of stuff you’ll never use

But here are a few of interest ...

@property BOOL networkActivityIndicatorVisible;
@property BOOL statusBarHidden;
@property UIStatusBarStyle statusBarStyle;

Other things that we’ll get into later
Application Lifecycle

- Create UIApplication object
- Application initializes itself
- MainWindow.xib is loaded (delegate is usually set)
- NSAutoreleasePool is created
- Application waits for events to occur
- Events are dispatched to your code
- When done, screen is updated (drawRect: might be called)
- NSAutoreleasePool is drained
- Application waits for events to occur again
- System asks application to terminate and it does
Autorelease Pools

Pool created

Launch app
App initialized
Load main nib
Wait for event
Handle event

Pool created

Exit app

Wednesday, April 14, 2010
Autorelease Pools

Launch app
App initialized
Load main nib
Wait for event
Handle event
Exit app

Pool

Pool created

[object autorelease];

Objects autoreleased here go into pool

Wednesday, April 14, 2010
Autorelease Pools

Launch app → App initialized → Load main nib → Wait for event → Handle event → Exit app

Pool created

Objects autoreleased here go into pool
Pool released

Launch app

App initialized

Load main nib

Wait for event

Handle event

Pool created

[object release];

Objects autoreleased here go into pool

Wednesday, April 14, 2010
Autorelease Pools

Pool

Pool released

Objects autoreleased here go into pool

Pool created

Launch app
App initialized
Load main nib
Wait for event
Handle event
Exit app

Wednesday, April 14, 2010
Application Lifecycle

Parts of the lifecycle you are involved in:

1. MainWindow.xib – you can edit it
2. Application Delegate – you can implement it
3. Event Handling and Drawing – you can implement

The rest is handled automatically for you

We covered #3 on Monday. Let’s look at #1 and #2 now.
MainWindow.xib

- Automatically loaded
- Contains (what is usually the only) UIWindow for your app
- Contains the Application Delegate
  (this is just an NSObject with its class set in the Identity Inspector in IB)
  The delegate implements the UIApplicationDelegate protocol
  All the methods in this protocol are optional

- Xcode creates a MainWindow.xib for you
- Xcode creates a template Application Delegate for you
  It calls it <YourApplicationName>AppDelegate.[mh]
If you choose “View-based Application” when you create your project, Xcode will create a View Controller in MainWindow.xib. It will be called <YourApplicationName>ViewController. And its “NIB Name” property will be <YourApplicationName>ViewController.xib.

The AppDelegate that Xcode creates will have outlets to the window and to the View Controller in MainWindow.xib. For example, here’s the AppDelegate Xcode created for Happiness:

```c
#import <UIKit/UIKit.h>

@class HappinessViewController;

@interface HappinessAppDelegate : NSObject <UIApplicationDelegate> {
    UIWindow *window;
    HappinessViewController *viewController;
}

@property (nonatomic, retain) IBOutlet UIWindow *window;
@property (nonatomic, retain) IBOutlet HappinessViewController *viewController;
@end
```

```c
#import <UIKit/UIKit.h>

@class HappinessViewController;

@interface HappinessAppDelegate : NSObject <UIApplicationDelegate> {
    UIWindow *window;
    HappinessViewController *viewController;
}

@property (nonatomic, retain) IBOutlet UIWindow *window;
@property (nonatomic, retain) IBOutlet HappinessViewController *viewController;
@end
```
If you choose “Window-based Application” when you create your project, Xcode will only create the UIWindow (no View Controller) in MainWindow.xib.

We will do that today in our demo. The AppDelegate will look like this (our demo app today is “Psychologist”):

```objective-c
#import <UIKit/UIKit.h>

@interface PsychologistAppDelegate : NSObject <UIApplicationDelegate> {
    UIWindow *window;
}

@property (nonatomic, retain) IBOutlet UIWindow *window;

@end
```
Application Delegate

Application Delegate is involved in the entire lifecycle

- (void)applicationDidFinishLaunching:
- (void)applicationDidBecomeActive:
- (BOOL)application:handleOpenURL:
- (void)applicationDidReceiveMemoryWarning:
- (void)applicationWillResignActive:
- (void)applicationDidResignActive:
- (void)application:willChangeStatusBarOrientation:duration:
- (void)application:didChangeStatusBarOrientation:
- (void)applicationWillTerminate:

... and more
Application Delegate

applicationDidFinishLaunching:
This is only called after MainWindow.xib is loaded.
Note the property in UIViewController called view.
All UIViewController all have a view property which is their top-level view.
It is either built in UIViewController's loadView or from its .xib file.
In this case, it is wired up to the top-level view in HappinessViewController.xib
(not in MainWindow.xib).
We simply add it as a subview of the main window below.

@implementation HappinessAppDelegate
@synthesize window;
@synthesize viewController;

- (void)applicationDidFinishLaunching:(UIApplication *)application {
    // Override point for customization after app launch
    [window addSubview:viewController.view];
    [window makeKeyAndVisible];
}

- (void)dealloc {
    [viewController release]; // this should be self.viewDidLoad = nil
    [window release];         // this should be self.window = nil
    [super dealloc];
}
@end
**Application Delegate**

**applicationWillTerminate:**
There is no way to stop the termination, you just have a chance to clean up. Don't do anything that takes very long here! If you have a lot of data to write out or something, do as you go, not here. Here's an example which writes out NSUserDefaults.

```objective-c
@implementation HappinessAppDelegate

-(void)applicationWillTerminate:(UIApplication *)application
{
    [[NSUserDefaults standardUserDefaults] synchronize];
}
@end
```
Application Delegate

**applicationDidReceiveMemoryWarning:**
Now is the time to clean up big memory hogs:
Image data
Large data sets
Sounds or other media
Anything that can be reconstructed from a permanent store like disk

View Controllers also receive a version of this, and they are even more likely to be able to do something about it than the Application Delegate so many times there is no implementation here at all.
View Controller

It’s an MVC Controller with a property pointing to its View
@property (nonatomic, retain) UIView *view;
Note that it retains its view, so you won’t have to.
We’ve seen that property as an outlet (and ignored it) in IB a few times.

It has a “lifecycle” too, which starts with creating it

Its designated initializer specifies .xib file to use
- (id)initWithNibName:(NSString *)nibName bundle:(NSBundle *)nibBundle
If nibBundle is nil, it gets it out of the Resources folder in Xcode.
Usually you just call init which looks for a .xib of the same name as class.
If there is no .xib, then you must subclass UIViewController and implement:
- (void)loadView
If you don’t use a .xib, this method MUST set the view property.
Don’t specify nibName and also implement loadView, it’s one or the other
(it’s undefined what it means to do both).
**View Controller**

- Lifecycle continued ...

- Once it is created, it sends `viewDidLoad` to itself

  ```c
  -(void)viewDidLoad
  {
    // set up properties and such of the view here
    // but don't start loading data or other expensive operations
    view.someProperty = 35;
  }
  ```

- Just before it appears on screen, it sends

  ```c
  -(void)viewWillAppear:(BOOL)animated
  {
    [super viewWillAppear:animated]; // can be called anywhere in here
    // kick off data loading now, but don't block (use threads)
    [self beginLoadingDataFromTheWeb];
    // maybe set up UI to indicate data loading in progress
  }
  ```
View Controller

Lifecycle continued ...

Just before it disappears from the screen

- (void)viewWillDisappear:(BOOL)animated
  {
    [super viewWillDisappear:animated]; // can be called anywhere in here
    // be nice to user in case they come back to this view
    [self rememberScrollPosition];
    // do other things that make sense now that we’re going away
    [self saveDataToPermanentStore];
    // again, be careful not to do anything time-consuming here
  }

Also:

- (void)viewDidAppear:(BOOL)animated
- (void)viewDidDisappear:(BOOL)animated
View Controller

Lifecycle continued ...

If we are low on memory

- (void)viewDidUnload
  {
    self.outlet = nil; // release outlets so they can get deallocated
  }

By the time this is called, the view property is nil.
This almost never happens (i.e. when memory is low, this is rarely called).
There are bigger memory fish to fry.
What else can a View Controller do?

- It knows what container it is in
  Might be in a Tab Bar or a Navigation Controller or a Split View, et. al.
  @property (readonly) UINavigationController *navigationController;
  @property (readonly) UITabBarController *tabBarController;
  @property (readonly) UISplitViewController *splitViewController;
  Might be being displayed modally by another View Controller
  @property (readonly) UIViewController *modalViewController;

- Can control whether it can be rotated by the user
  - (BOOL)shouldAutorotateToInterfaceOrientation:(UIInterfaceOrientation)
  By default, returns NO.
  Return YES to any orientation your View is willing to support.
  Other methods will notify you when rotation happens, e.g.
  - (void)willRotateToInterfaceOrientation:(UIInterfaceOrientation)
  - (void)didRotateFromInterfaceOrientation:(UIInterfaceOrientation)
What else can a View Controller do?

Has a property which is an object which contains all sorts of info about how to display the top bar if the View Controller is in a UINavigationController

@property (readonly) UINavigationItem *navigationItem;
(note that it automatically creates one for you if it doesn’t exist, that’s why it is a readonly property with no setter)

UINavigationItem has a lot of properties, e.g.
@property NSString *title;
@property UIBarButtonItem *leftBarButtonItem;
@property UIBarButtonItem *rightBarButtonItem;
@property UIView *titleView; // replaces title with an arbitrary view

Lots of other stuff. Check out documentation.
Controllers of Controllers

There are a handful of special kinds of UIViewController in UIKit.

They are UIViewController whose view is an arrangement of the views of other UIViewController

UINavigationController
Hierarchical flow of views

UITabBarController
Independently tabbable views

UISplitViewController
Side-by-side master-detail view
UINavigationController

- Manages a “stack” of View Controllers

- The view belonging to the View Controller on the top of the stack is displayed

- What controls are in the top and bottom bar of the UINavigationController is also controlled by the View Controller on top of the stack

- When a selection of some sort is made in the top View Controller’s view, a new View Controller can be pushed onto the stack and become the new top

- There is always a “back” button in the upper left corner of the view for popping off the stack (i.e. going back)
UINavigationController's Parts

- view of the UIViewController which is currently on the top of the stack
UINavigationController’s Parts

- view of the UIViewController which is currently on the top of the stack
- title (an NSString property on UIViewController) of the one on top of the stack
UINavigationController’s Parts

- view of the UIViewController which is currently on the top of the stack
- title (an NSString property on UIViewController) of the one on top of the stack
- title of the UIViewController which is next down on the stack
UINavigationController’s Parts

- view of the UIViewController which is currently on the top of the stack
- title (an NSString property on UIViewController) of the one on top of the stack
- title of the UIViewController which is next down on the stack
- toolbarItems (an NSArray of UIBarButtonItem) of the UIViewController which is on top of the stack
UIViewController

- Designated initializer is `init`
  But there is a convenience initializer to start stack off
  - `initWithRootViewController:(UIViewController *)vc`

- Pushing UIViewController onto the stack
  - `(void)pushViewController:(UIViewController *)vc animated:(BOOL)animated`

- UIViewController know which UINavigationController they are in (if any), so it’s easy to push the next one on the stack from the one that is currently on the stack
  - `(void)someAction:(UIButton *)sender`
    {
      UIViewController *vcToPush = ...;
      [self.navigationController pushViewController:vcToPush animated:YES];
    }

By the way, animated is YES always except very first push
UIKit NavigationController

Notice that the space available for your view is smaller inside a UINavigationController (because of the bars on top and bottom)

So make sure you have your autosizing set up properly in Interface Builder
There are also methods in UIView to set autosizing
How you pass data between views on the stack is important.

Don’t use global variables
Your Application Delegate is a global variable!

So how to do it?
Set up the data in a UIViewController before pushing it.
Imagine you have an action method in the top VC on the stack (visible).

```c
-(void)someAction:(UIButton *)sender
{
    MoreDetailViewController *vcToPush = [self getVCToPush];
    vcToPush.someProperty = ...;
    vcToPush.otherProperty = ...;
    [self.navigationController pushViewController:vcToPush animated:YES];
}
```

VCs on the stack should not have pointers to the VC that pushed them on.
If you absolutely need data from the VC that pushed you onto the stack, use a protocol to get it and have that pushing VC set itself as your delegate.
Demo

* We’ll take a look at HappinessAppDelegate and MainWindow.nib

* Create a new project not using “View-based App.”

* We’ll reuse HappinessViewController and FaceView by dragging them into our new project

* Next we’ll create a new UIViewController which will push a HappinessViewController

* Finally, we’ll create a UINavigationController and call addSubview: with its view on our main window
Homework

 créer un nouveau projet qui n’est pas “View-based”
 Vous devrez implémenter votre propre applicationDidFinishLaunching:

 Importer (reutiliser) le code d’un autre projet
 Vous utiliserez votre CalculatorBrain et votre CalculatorViewController

 Créer une vue personnalisée avec sa propre drawRect:

 Créer un UIViewController

 Créer un UINavigationController

 Poussez un UIViewController dans le contexte d’un UINavigationController