CS193P - Lecture 9

iPhone Application Development

Data in Your iPhone App

Announcements

- Presence 2 is due on Tuesday 5/5 at 11:59pm
  - Questions?

Announcements

- Friday optional lectures will now be recorded
  - Should become available on iTunes U if all goes well!
  - This Friday: Loren Brichter (http://www.atebits.com)

Announcements

- Final projects!
  - Groups of 1-2 people
  - Final 3 weeks of the course
  - **Due Sunday 6/7 at 11:59PM**
  - **Presentations on Monday 6/8 from 12:15-3:15PM**
- You **must** send us your proposal for approval
  - **Approval deadline is Monday 5/11**
  - But the earlier you're approved, the earlier you can start...
- Students retain ownership of final projects

Frequently Encountered Issues

- “Calling a method on object X doesn’t do anything!”
  - Remember that in Objective-C, **messaging nil is allowed**
  - If the method has a return value, it will return zero or nil
  - **Try inspecting the value of the variable with NSLog() or gdb**

Frequently Encountered Issues

- "My IBOutlet variables are nil!"
  - Remember that view controllers don't load their NIBs right away
  - Don't try to access IBOutlet variables in -init methods
  - Instead, **use -viewDidLoad or -viewWillAppear:**
    - -viewDidLoad called once after loading the view
    - -viewWillAppear: called every time the view comes onscreen

Today's Topics

- Data in Your iPhone App
  - Saving & loading **local data**
  - Accessing **remote data** over the Internet

Today's Topics

- Property Lists
- iPhone's File System
- Archiving Objects
- The Joy of SQLite
- Web Services
- App Data Flow

Property Lists

Property Lists

- Convenient way to store a **small amount of data**
  - Arrays, dictionaries, strings, numbers, dates, raw data
  - Human-readable XML or binary format
- NSUserDefaults class uses property lists under the hood

When Not to Use Property Lists

- More than a few hundred KB of data
  - Loading a property list is all-or-nothing
- Complex object graphs
- Custom object types

Reading & Writing Property Lists

- NSArray and NSDictionary convenience methods
- Operate recursively

```
// Writing
- (BOOL)writeToFile:(NSString *)aPath atomically:(BOOL)flag;
- (BOOL)writeToURL:(NSURL *)aURL atomically:(BOOL)flag;

// Reading
- (id)initWithContentsOfFile:(NSString *)aPath;
- (id)initWithContentsOfURL:(NSURL *)aURL;
```

Writing an Array to Disk

```
NSArray *array = [NSArray arrayWithObjects:@“Foo”,
                  [NSNumber numberWithBool:YES],
                  [NSDate dateWithTimeIntervalSinceNow:60],
                  nil];
[array writeToFile:@“MyArray.plist” atomically:YES];
```

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST 1.0//EN"
"http://www.apple.com/DTDs/PropertyList-1.0.dtd">
<plist version="1.0">
<array>
      <string>Foo</string>
      <true/>
      <date>2009-04-29T15:26:18Z</date>
</array>
</plist>
```

Writing a Dictionary to Disk

```
NSDictionary *dict = [NSDictionary dictionaryWithObjectsAndKeys:
                      @“Name”, @“Evan”,
                      @“Lecture”, [NSNumber numberWithInt:9],
                      nil];
[dict writeToFile:@“MyDict.plist” atomically:YES];
```

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST 1.0//EN"
"http://www.apple.com/DTDs/PropertyList-1.0.dtd">
<plist version="1.0">
<dict>
      <key>Name</key>
      <string>Evan</string>
      <key>Lecture</key>
      <integer>10</integer>
</dict>
</plist>
```

NSPropertyListSerialization

- Allows finer-grained control
  - File format
  - More descriptive errors
  - Mutability

```
// Property list to NSData
+ (NSData *)dataFromPropertyList:(id)plist
                          format:(NSPropertyListFormat)format
                errorDescription:(NSString **)errorString;

// NSData to property list
+ (id)propertyListFromData:(NSData *)data
          mutabilityOption:(NSPropertyListMutabilityOptions)opt
                    format:(NSPropertyListFormat *)format
          errorDescription:(NSString **)errorString;
```

More on Property Lists

- “Property List Programming Guide for Cocoa”
  http://developer.apple.com/documentation/Cocoa/Conceptual/PropertyLists/

iPhone’s File System

Keeping Applications Separate

Image (cc) by davidsilver on Flickr

Why Keep Applications Separate?

- Security
- Privacy
- Cleanup after deleting an app

Home Directory Layout

- Each app has its **own set of directories**
- <Application Home>
  - MyApp.app
    - MyApp
    - MainWindow.nib
    - SomeImage.png
  - Documents
  - Library
    - Caches
    - Preferences
- Applications only read and write within their home directory
- Backed up by iTunes during sync (mostly)

File Paths in Your Application

```
// Basic directories
NSString *homePath = NSHomeDirectory();
NSString *tmpPath = NSTemporaryDirectory();

// Documents directory
NSArray *paths =
NSSearchPathForDirectoriesInDomains(NSDocumentDirectory,
                                    NSUserDomainMask, YES);
NSString *documentsPath = [paths objectAtIndex:0];

// <Application Home>/Documents/foo.plist
NSString *fooPath =
[documentsPath stringByAppendingPathComponent:@“foo.plist”];
```

Including Writable Files with Your App

- Many applications want to include some starter data
- But application bundles are code signed
  - You can't modify the contents of your app bundle
- To include a writable data file with your app...
  - Build it as part of your app bundle
  - On first launch, **copy it to your Documents directory**

Archiving Objects

Archiving Objects

- Next logical step from property lists
  - Include arbitrary classes
  - Complex object graphs
- Used by Interface Builder for NIBs

Making Objects Archivable

- Conform to the <NSCoding> protocol

```
// Encode an object for an archive
- (void)encodeWithCoder:(NSCoder *)coder
{
   [super encodeWithCoder:coder];
   [coder encodeObject:name forKey:@“Name”];
   [coder encodeInteger:numberOfSides forKey:@“Sides”];
}

// Decode an object from an archive
- (id)initWithCoder:(NSCoder *)coder
{
   self = [super initWithCoder:coder];
   name = [[coder decodeObjectForKey:@“Name”] retain];
   numberOfSides = [coder decodeIntegerForKey:@“Side”];
}
```

Archiving & Unarchiving Object Graphs

- Creating an archive

```
NSArray *polygons = ...;
NSString *path = ...;
BOOL result = [NSKeyedArchiver archiveRootObject:polygons
                                          toFile:path];
```

- Decoding an archive

```
NSArray *polygons = nil;
NSString *path = ...;
polygons = [NSKeyedUnarchiver unarchiveObjectWithFile:path];
```

More on Archiving Objects

- “Archives and Serializations Programming Guide for Cocoa”
  http://developer.apple.com/documentation/Cocoa/Conceptual/Archiving/

The Joy of SQLite

SQLite

- Complete SQL database in an ordinary file
- Simple, compact, fast, reliable
- No server
- Great for embedded devices
  - Included on the iPhone platform

“And just as you have received SQLite for free, so also freely give, paying the debt forward.”

D. Richard Hipp

When Not to Use SQLite

- Multi-gigabyte databases
- High concurrency (multiple writers)
- Client-server applications
- “Appropriate Uses for SQLite”
  http://www.sqlite.org/whentouse.html

SQLite C API Basics

- Open the database

```
int sqlite3_open(const char *filename, sqlite3 **db);
```

- Execute a SQL statement

```
int sqlite3_exec(sqlite3 *db, const char *sql,
                 int (*callback)(void*,int,char**,char**),
                 void *context, char **error);

// Your callback
int callback(void *context, int count,
             char **values, char **columns);
```

- Close the database

```
int sqlite3_close(sqlite3 *db);
```

Demo:
Simple SQLite

More on SQLite

- “SQLite in 5 Minutes Or Less”
  http://www.sqlite.org/quickstart.html
- “Intro to the SQLite C Interface”
  http://www.sqlite.org/cintro.html

Core Data

- Object-graph management and persistence framework
  - Makes it easy to save & load model objects
    - Properties
    - Relationships
  - Higher-level abstraction than SQLite or property lists
- Available on the Mac OS X desktop
- Not available on iPhone OS 2.x...

Web Services

Your Application & The Cloud

- Store & access remote data
- May be under your control or someone else's
- Many Web 2.0 apps/sites provide developer API

“I made a location-based user-generated video blogging mashup... for pets!”

Integrating with Web Services

- **Non-goal** of this class: teach you all about web services
  - Plenty of tutorials accessible, search on Google
- Many are exposed via RESTful interfaces with XML or JSON
- High level overview of parsing these types of data

XML

Options for Parsing XML

- libxml2
  - Tree-based: easy to parse, entire tree in memory
  - Event-driven: less memory, more complex to manage state
  - Text reader: fast, easy to write, efficient
- NSXMLParser
  - Event-driven API: simpler but less powerful than libxml2

More on Parsing XML

- Brent Simmons, "libxml2 + xmlTextReader on Macs"
  http://inessential.com/?comments=1&postid=3489
  - Includes example of parsing Twitter XML!
- Big Nerd Ranch, "Parsing XML in Cocoa"
  http://weblog.bignerdranch.com/?p=48
  - Covers the basics of NSXMLReader

JSON

JavaScript Object Notation

- More lightweight than XML
- Looks a lot like a property list
  - Arrays, dictionaries, strings, numbers
- Open source json-framework wrapper for Objective-C

What does a JSON string look like?

```
{
  “instructor” : “Evan Doll”,
  “students” : 60,
  “itunes-u” : true,
  “midterm-exam” : null,
  “assignments” : [ “WhatATool”,
                    “HelloPoly”,
                    “Presence” ]
}
```

Using json-framework

- Reading a JSON string into Foundation objects

```
#import <JSON/JSON.h>

// Get a JSON string from the cloud
NSString *jsonString = ...;

// Parsing will result in Foundation objects
// Top level may be an NSDictionary or an NSArray
id object = [jsonString JSONValue];
```

Using json-framework

- Writing a JSON string from Foundation objects

```
// Create some data in your app
NSDictionary *dictionary = ...;

// Convert into a JSON string before sending to the cloud
jsonString = [dictionary JSONRepresentation];
```

Demo:
Flickr API with JSON

More on JSON

- “JSON Parser/Generator for Objective-C”
  http://code.google.com/p/json-framework/
- “Introducing JSON”
  http://www.json.org/

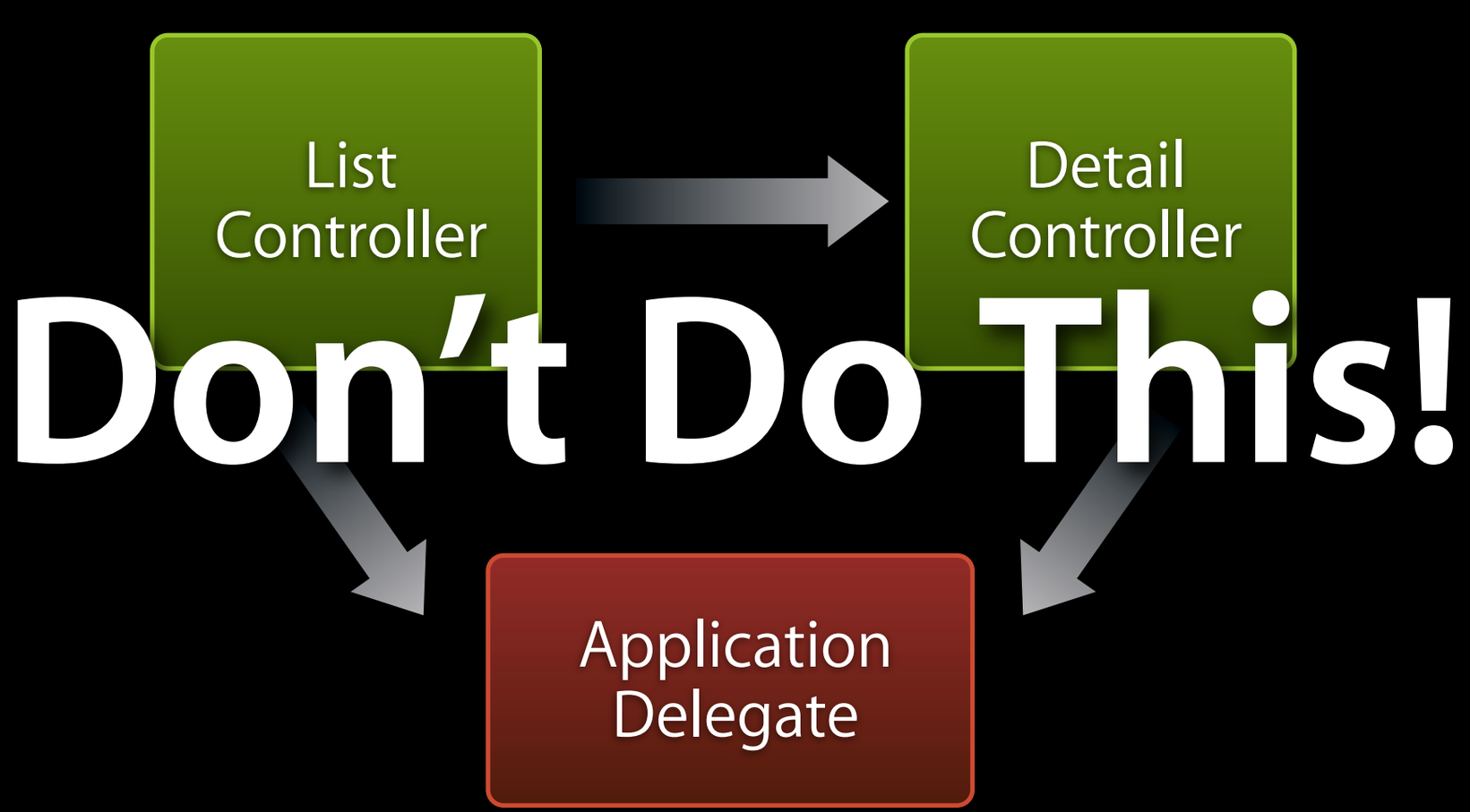
App Data Flow

Data Flow in Your Application

- Your objects may need to share data
- Don’t let this lead to tightly interwoven code

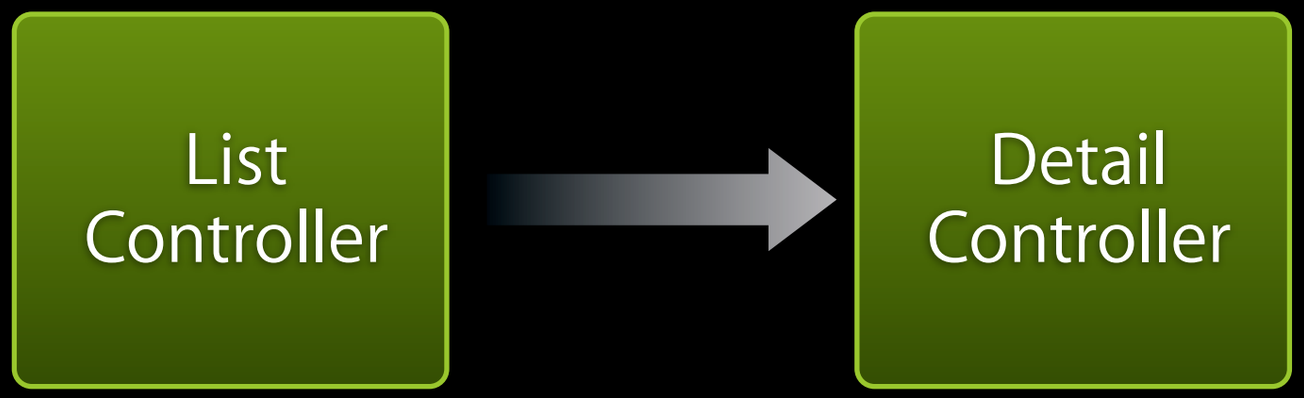
How Not To Share Data

- Global variables or singletons
  - This includes your **application delegate**!
- Direct dependencies make your code less reusable
  - And more difficult to debug & test



Best Practices for Data Flow

- Figure out **exactly** what needs to be communicated
- **Define input parameters** for your objects
- For communicating back up the hierarchy, **use loose coupling**
  - Define a generic interface (like delegation)



Delegates and Memory Management

- Delegates should be assigned, not retained, to **avoid cycles**

```
@property (assign) id delegate;
```

- When deallocating, if you're another object's delegate, unset it to **avoid leaving an invalid reference**

```
- (void)dealloc
{
  if (otherObject.delegate == self) {
    otherObject.delegate = nil;
  }
  [otherObject release];

  [super dealloc];
}
```

Recap

- Property lists
  - Quick & easy, but limited
- Archived objects
  - More flexible, but require writing a lot of code
- SQLite
  - Elegant solution for many types of problems
- XML and JSON
  - Low-overhead options for talking to “the cloud”
- Design your data flow thoughtfully

Questions?