CS193E
Lecture 4

Model View Controller
AppKit Framework Overview
Delegation
Archiving

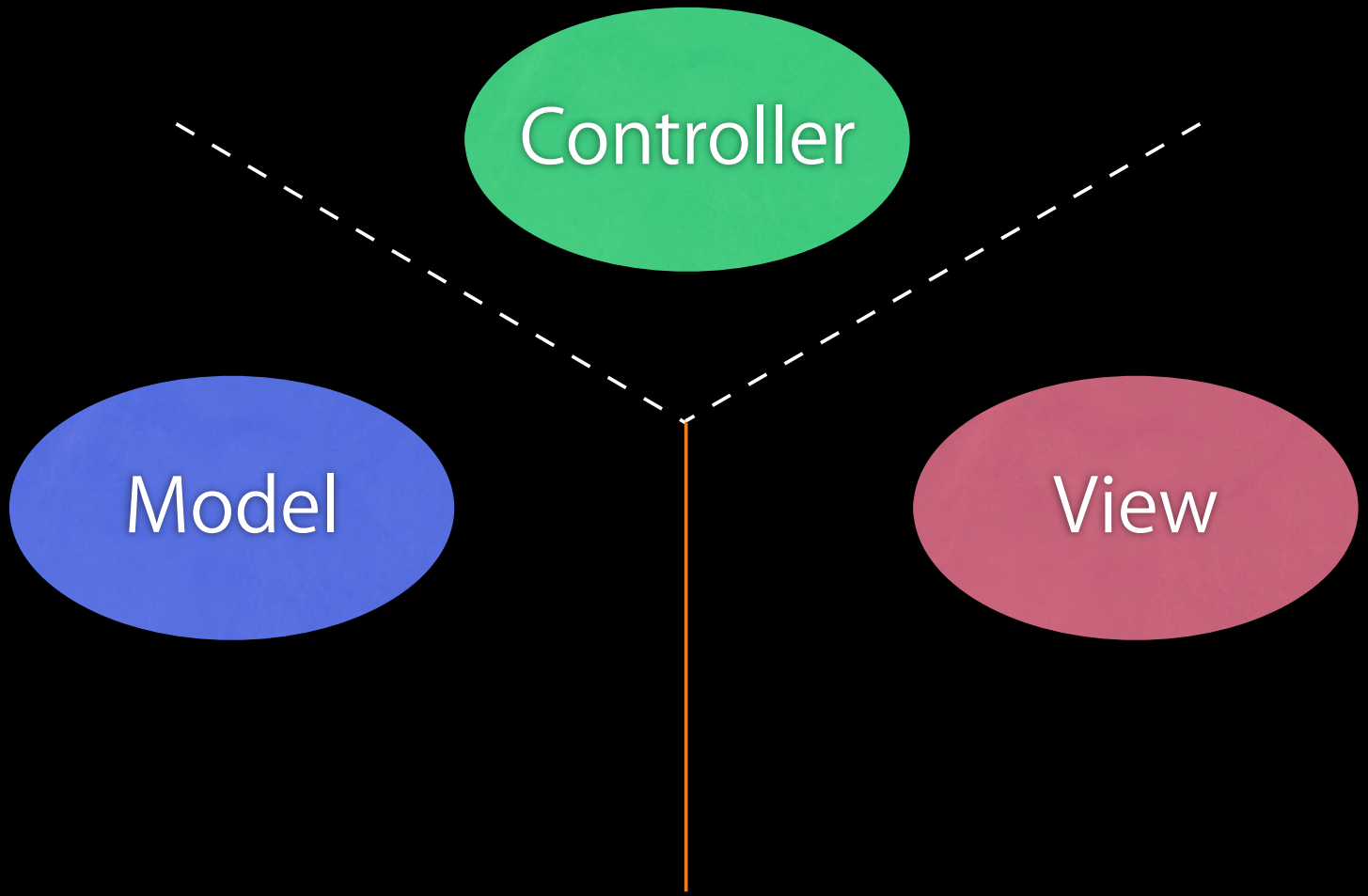
Today's Topics

- Questions about the assignments?
- Model-View-Controller design pattern
- AppKit Framework Overview
- Delegation
- Archiving

Different apps have a lot in common

- Some way to allow user to view and manipulate information
- Some defined domain of data the application views or edits
- Some way to keep the data in sync with the model
- Expected functionality and interface
  - Asking if you want to save before closing
  - Copy, cut and paste
  - Quit menu item and ⌘Q quits the app
  - etc.
- AppKit and its design patterns work to provide the common pieces, so you can focus on the application-specific pieces

Model, View, Controller



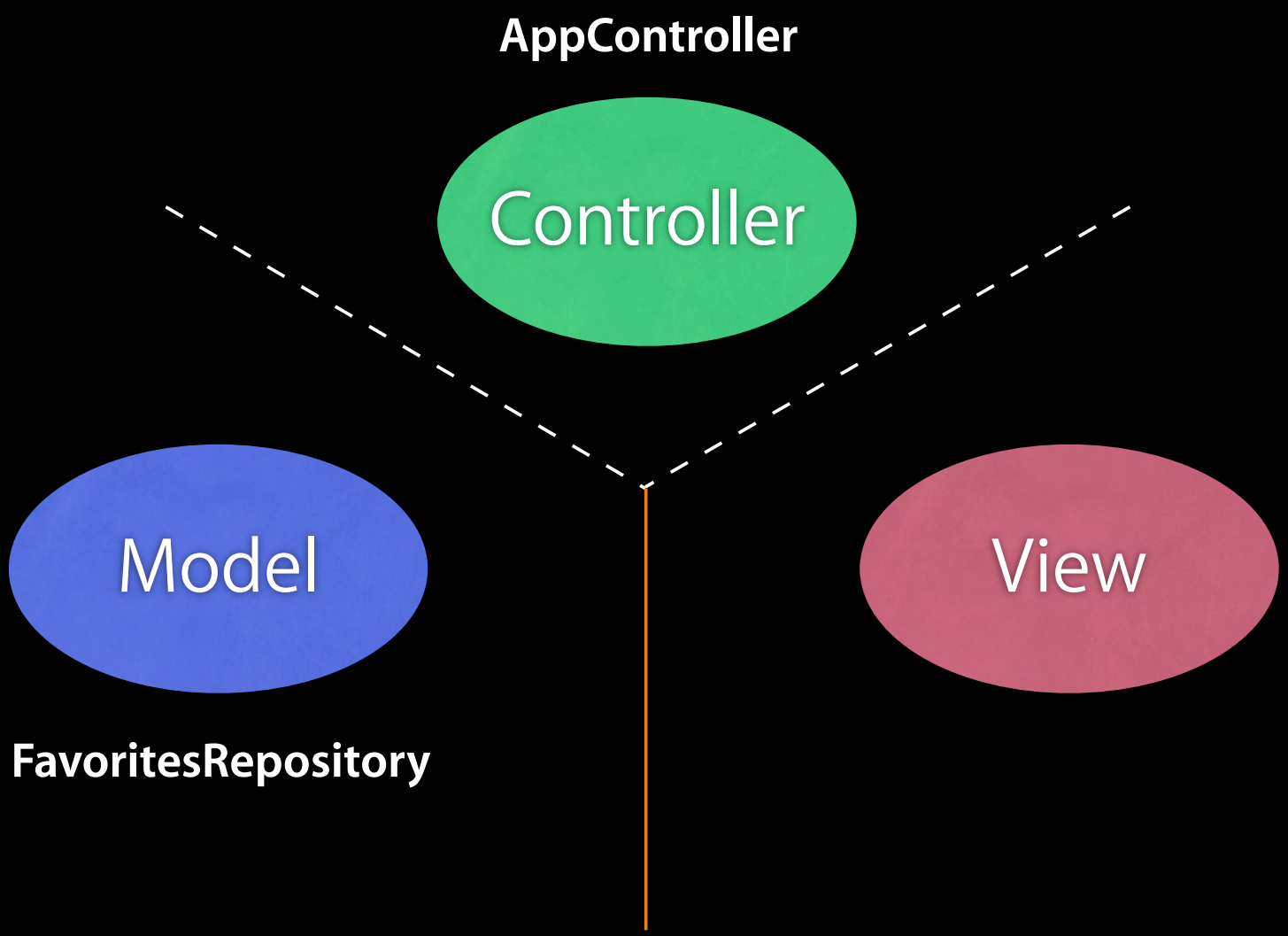
Model, View, Controller

- Divides an application into 3 main functional pieces
- Model
  - Manages the app data and state, not concerned with UI or presentation
  - Often persists somewhere
- View:
  - Displays the model objects to the user
  - Allows user to manipulate data
- Controller:
  - Coordinates the model and the view, keeps the view updated when model changes, etc. Typically where app "logic" is.

Favorite Things

- This week's assignment is a full MVC application
- Next week's assignment will flesh it out further
- It is not designed to be a complex application, but rather to provide a series of small studies of all of the fundamentals of a Cocoa application

Model, View, Controller

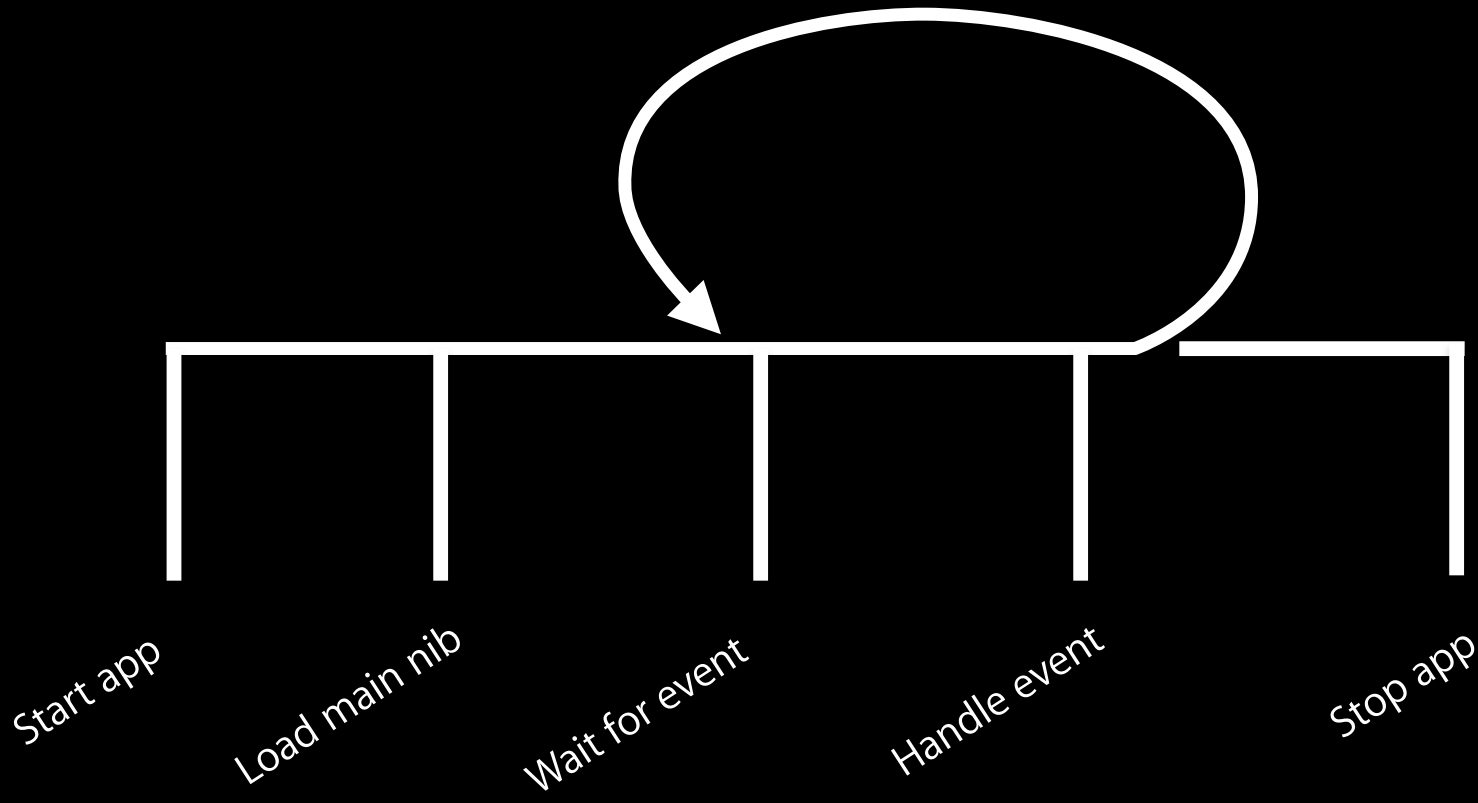


Application Demo and Tear Apart

What makes this app tick?

- Anatomy of an application
  - Compiled code
  - Nib files
  - Info.plist file
- Where does all this functionality come from?

App Lifecycle



Nib Loading

- Instantiates archived objects
  - Objects have same values/settings as in Interface Builder
  - Ensures all outlets and actions are connected
- If loading the nib automatically creates objects, where do I customize?

-awakeFromNib

- Control point to implement any additional logic after nib loading
- You often implement it in your controller class
- Guaranteed everything has been unarchived from nib, and all connections are made before -awakeFromNib is called

Info.plist file

- A property list containing various pieces of application-specific information
- Can edit most pieces of information using interface in Xcode

AppKit

- Your goal is to work with the AppKit framework, not against it
- Lots of things are built-in, but you have to know about them - use the documentation or ask us!
- If you're writing tons of code, step back and make sure you're not overlooking something

AppKit Classes Overview

NSApplication

- Manages the application lifecycle
- Runs the main event loop of the application
- Manages the main menu bar
- Keeps track of windows in the app
- Delegates many tasks and provides hooks into various application behaviors
- Rarely subclassed

NSWindow

- Manages the frame of a window
- Keeps track of views inside of the window
- Dispatches events to appropriate views
- Coordinates drawing of views in windows

NSView

- Manages a “rectangle” in a window
- Handles events directed at it
- Draws the contents of itself
- Always subclassed
- Views are arranged in a tree
  - Ordering is back-to-front, so children lie on top of parents

NSControl

- Draws widgets on the screen
- Handles user inputs and events
- Sends action methods
- Intended to be subclassed for new control types
- Works in conjunction with NSCell

NSCell

- Light-weight drawn object
- Think of it as a “stencil”
- Knows how to draw text and images
- Often subclassed for specific drawing needs

Working with AppKit

- Subclassing
- Delegation
- Objective-C Categories

Subclassing AppKit Classes

- Lots of classes are usable out of the box

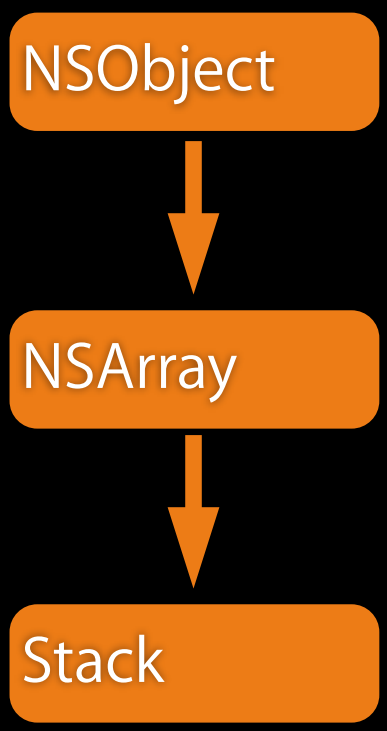
  Examples: NSButton, NSSlider, NSTableView
- Some classes are meant to be subclassed

  Examples: NSObject, NSView, NSDocument
- Foundation classes usually don't get subclassed; AppKit classes frequently do
- Look for "Subclassing Notes" in docs
- Subclassing is powerful…but can be abused

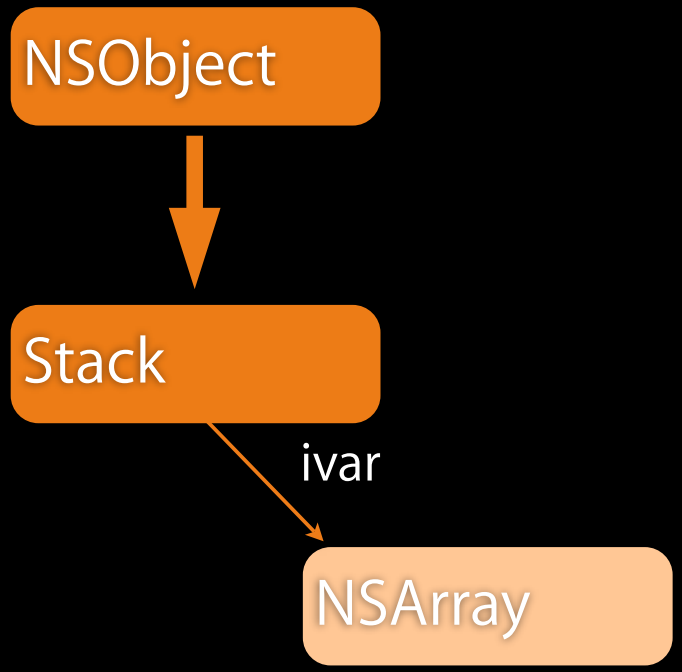
Subclass vs. Ownership

- Consider whether subclassing naturally extends the superclass
- Sometimes it's better to own an object than to subclass from it

A Stack Class



Not good!



Good!

Subclassing

- You typically subclass NSView, NSCell, and NSObject (of course!)
- You rarely (or never) subclass controls such as NSTextField, NSSlider and so on
- You never subclass Foundation collection classes or string classes

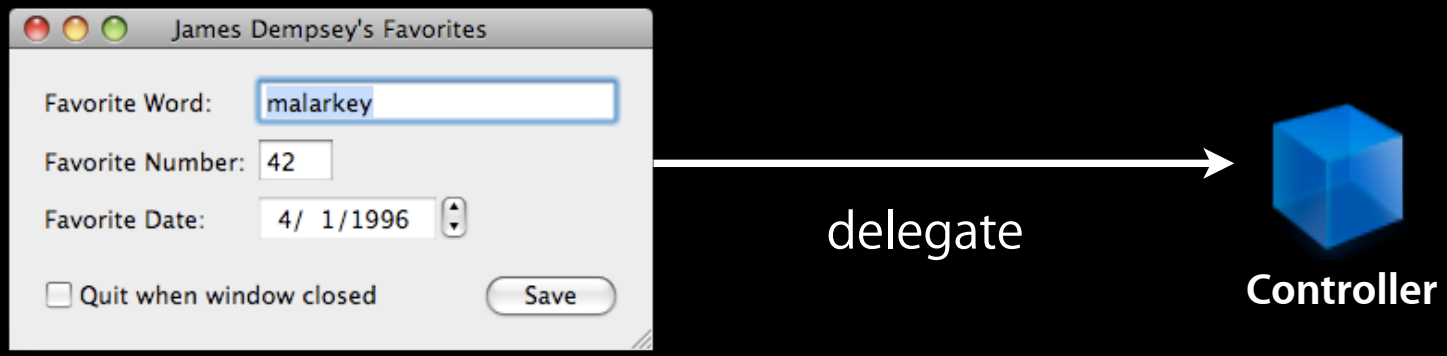
Delegation

Delegation

- Allows your object to control or fine tune behavior of an AppKit object
- Doesn't require subclassing
- Many AppKit classes have delegates

  Examples: NSApplication, NSWindow, NSTableView
- Several types of delegate uses:
  - Approval that something should be done
  - Inform an object that something will/did happen
  - Hand off tasks that may be application-specific

NSWindow Delegate




```
- (BOOL)windowShouldClose:(id)sender;
- (BOOL)windowShouldZoom:(NSWindow *)window
   toFrame:(NSRect)newFrame;
- (NSSize)windowWillResize:(NSWindow *)sender
   toSize:(NSSize)frameSize;
```

NSApplication Delegate

- Can handle opening or printing files
- Consulted when the user tries to quit
- Informed when app is activated/deactivated or hidden/unhidden
- Allowed to modify various behaviors when launching and quitting

How Delegates Work

- Your object is the delegate for the window, application, whatever
- You implement the delegate methods that you are interested in
- At runtime, AppKit objects call the delegate methods you implement to modify their behavior

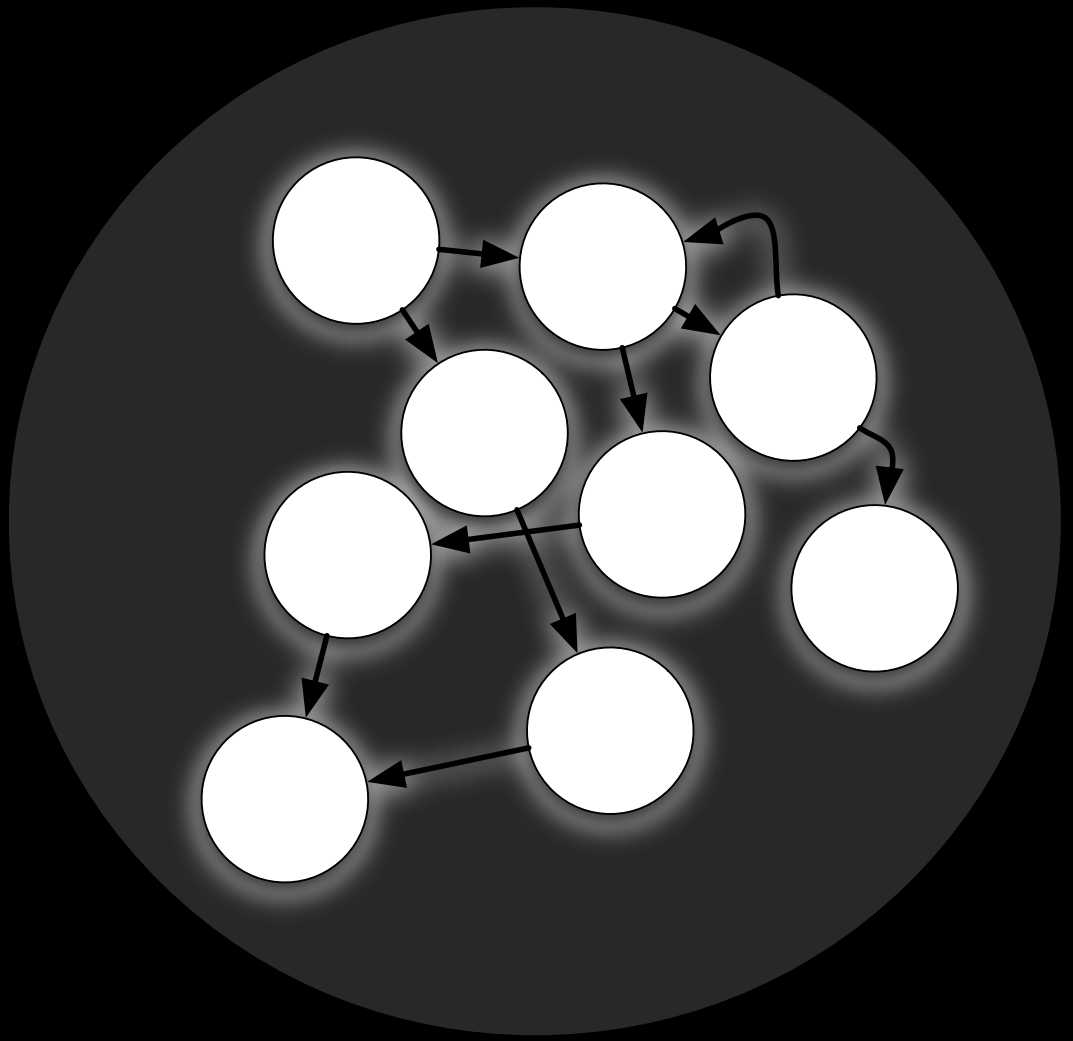
Finding Delegate Info

- Look at the documentation for delegate information
- In header files, delegate methods are usually grouped together

Archiving

Archiving Objects

Q: How do you turn a graph of objects like this into an NSData?



A: You use an NSKeyedArchiver

Keyed Archiving

- Archivers (an NSCoder subclass) are objects that help convert a graph of objects into NSData
- Traverses object graph, tells each object to “encode” or “decode” itself
- Objects store their specific data by key
- Archiving is really hard, archivers make it easy
- Archivers handle all the dirty work of dealing with cycles, versioning, weak references, etc.
- Classes implement the NSCoding protocol to be archivable
- Many framework classes implement NSCoding
  - Collection classes
  - Value classes

Archiving objects to/from data

- Archivers drive the whole process, you just tell it where to begin
- Encoding from objects to data:

```
NSData *data =
  [NSKeyedArchiver archivedDataWithRootObject:tree];
```

- Decoding data into objects:

```
id obj =
  [[NSKeyedUnarchiver unarchiveObjectWithData:data] retain];
```

- Pay very careful attention to retain/release!

Archiving objects to/from a file

- Convenience methods to work with files
- Encoding from objects to a file:

```
BOOL success =
  [NSKeyedArchiver archivedRootObject:obj toFile:path];
```

- Decoding data from a file into objects:

```
id obj =
  [[NSKeyedUnarchiver unarchiveObjectWithFile:file] retain];
```

Archiving Objects

- Archivers drive the whole process, you just tell it where to begin
- Encoding from objects to data:

```
NSData *data =
  [NSKeyedArchiver archivedDataWithRootObject:tree];
```

- Decoding data into objects:

```
id obj =
  [[NSKeyedUnarchiver unarchiveObjectWithData:data] retain];
```

- Pay very careful attention to retain/release!

Supporting Archiving

- Archiveable objects implement two methods of the NSCoding protocol:
- Encoding:

```
- (void)encodeWithCoder:(NSCoder *)coder;
```

- Decoding:

```
- (id)initWithCoder:(NSCoder *)coder;
```

- “coder” argument is the NSKeyedArchiver that’s doing the archiving

Encoding example

```
- (void)encodeWithCoder:(NSCoder *)coder
{
    [coder encodeInt:number forKey:@"number"];
    [coder encodeRect:bounds forKey:@"bounds"];
    [coder encodeObject:color forKey:@"color"];
    [coder encodeObject:name forKey:@"name"];
}
```

Decoding example

```
- (id)initWithCoder:(NSCoder *)coder
{
  if (self = [super init]) {

    number = [coder decodeIntForKey:@"number"];
    bounds = [coder decodeRectForKey:@"bounds"];
    color = [[coder decodeObjectForKey:@"color"] retain];
    name = [[coder decodeObjectForKey:@"name"] retain];
  }
  return self;
}
```

Archiving FAQs

- Can I encode C types along with objects?
  Yes. Check the docs for NSCoder.
- Does it matter what order I encode things?
  No. Everything is kept track of by key which gives a lot of flexibility.
- Why does my application crash after unarchiving?
  More than likely you didn't retain something in an initWithCoder: method.

Miscellaneous

Bits and pieces to help with Assignment 2

NSFileManager

- Class for performing common file operations
  - Check for a file's existence
  - Create a new folder/directory
- Typically you use the object returned by +defaultManager
- Uses NSString for path arguments - use in conjunction with NSString's various path methods

NSWindow document edited indicator

- Indicates to the user that window contains unsaved changes
- NSWindow API allows it to be set:

```
- (BOOL)isDocumentEdited;
- (void)setDocumentEdited:(BOOL)flag;
```

not edited

is edited

Naming Conventions

- Use prefixes on class names
  - Examples: NSString, NSArray

    ABPerson, ABRecord

    WebView, WebFrame
- Protect against class name collisions
- Differentiate functional areas
- Don’t use a prefix used by Apple frameworks (especially not ‘NS’)

Naming Conventions

- Setter methods use set, getters don't use use get

```
setColor:                color
setEditable:             isEditable
setDrawsBackground:      drawsBackground
```

Naming Conventions

- ‘get’ means ‘return by reference’

```
- (void)getWhite:(float *)white alpha:(float *)alpha;
```

```
NSColor *color = [NSColor grayColor];
float white, alpha;
[color getWhite:&white alpha:&alpha];
```

Questions?