

HIT3329 / HIT8329
Creating Data Driven Mobile Applications

Lecture 3

The Cocoa Touch Framework

Presented by Paul Chapman

Adjunct Industry Fellow F-ICT
Director, Long Weekend LLC
Swinburne University of Technology

Last Lecture Reviewed

- 1. NSObject
 - 2. Properties
 - 3. Memory Management
- Questions?**

What's On For Today?

1. Protocols
2. Delegates & Datasources
3. Interface Builder
4. Application Object
5. Collections
6. File IO

1.0 Protocols

An Objective-C Protocol :

- Is like an interface in Java
- Can define required methods, optional methods or both
- A group of method prototype declarations (contains no operative code)

1.1 Protocols

Example Protocol:

If it walks like a duck & quacks like a duck, it must be a duck!

```
@protocol DuckProtocol
```

```
@required  
-(void) quack;
```

```
@optional  
-(void) fly;    // Not all ducks can fly
```

```
@end
```

**Place your protocol at the top of your .h header file*

1.2 Protocols Are Not About Types

Imagine you want to write a "*duck hunting*" game. In the game, humans can also "*quack*" using duck calls.

Instead of worrying about an object's type, like this:

```
if ([object isKindOfClass:[Duck class]])
{
    [object quack];    // Quack if it is a duck
}
else if ([object isKindOfClass:[Human class]])
{
    [object useDuckCall]; // For Humans use a duck call
}
```

1.3 Protocols Are About Behaviour

Protocols only care about behaviour, like this:

```
if ([object conformsToProtocol:@protocol(DuckProtocol)])  
{  
    [object quack]; // Quack if it acts like a duck  
}
```

Conclusion: Protocols decouple your objects

1.4 Implementing the Duck Protocol

To make it work we tell the compiler the Human class "conforms" to the DuckProtocol

```
// Human.h
@interface Human : NSObject <DuckProtocol>
- (void) shootAtDuck:(id<DuckProtocol>)duck;
- (void) safelyShootAtDuck:(Duck*)duck;
@end
```

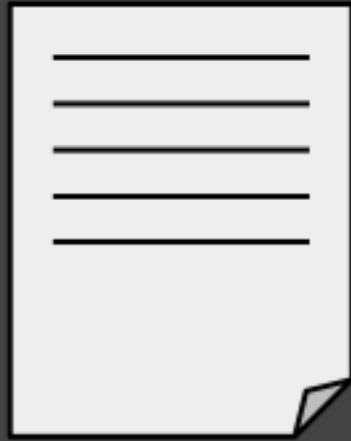
And implement the quack method:

```
// Human.m
@implementation Human
- (void) quack {
    // Humans can quack too!
    [self useDuckCall];
}
/* ... other methods ... */
```



See: http://en.wikipedia.org/wiki/Duck_typing

1.5 How It Works So Far



DuckProtocol
(says object should quack)



Human <DuckProtocol>
- (void) quack;

1.6 Protocols - Formal Definition

Protocols declare methods that may be implemented by any class. Protocols are useful in *at least 3 situations*:

1. To declare the interface to an object while concealing its class (*e.g. our duck example just now*)
2. To declare methods that others are expected to implement (*e.g. coming next - delegate*)
3. To capture similarities among classes that are not hierarchically related

Source: <http://developer.apple.com/library/mac/#documentation/Cocoa/Conceptual/ObjectiveC/Chapters/ocProtocols.html>

What's On For Today?

1. Protocols
- 2. Delegates & Datasources**
3. Interface Builder
4. Application Object
5. Collections
6. File IO

2.0 Delegates & Datasources



*Design
Pattern*

A delegate is:

- A common design pattern in Cocoa Touch
- A method for plugging in *ready made* functionality into your code

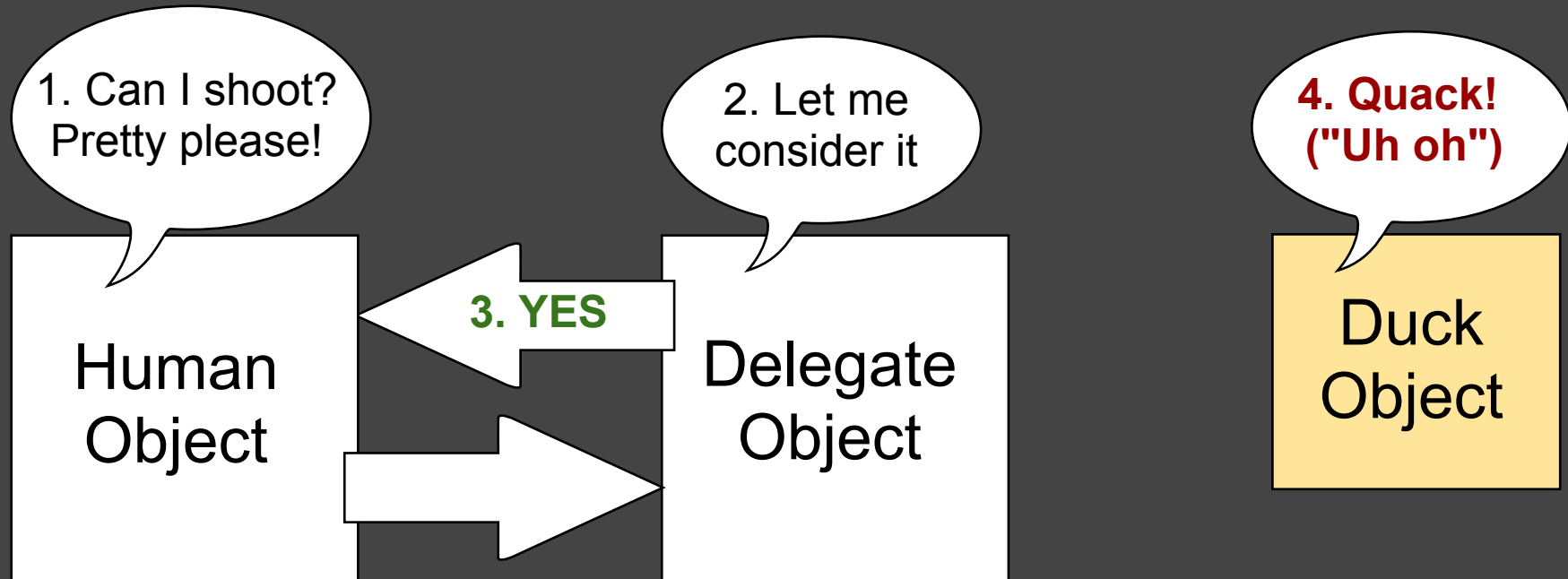
2.1 Delegates Explained

Imagine: Due to gun safety regulations, our duck hunting game has a new requirement.

A Human must get permission before shooting something. If no one is available to give permission, they must not shoot.

The Human class should have knowledge of ***how*** to shoot (implementation), but we ***delegate*** decision making (logic) to another object.

2.2 Duck Hunter Delegate Block Diagram



Note: We don't define a specific class for the delegate object. Instead, we declare how we want a delegate object to behave using a *protocol*.

2.3 Creating the Delegate's Protocol

1. Decide how the delegate should behave
2. Make a protocol to define that behaviour

```
@protocol HumanShootingDelegate
@required
-(BOOL) shouldShootAtDuck:(Human*)human;
@end
```

Now any object acting as a delegate for Human must implement the `shouldShootAtDuck` method.

2.4 Adding a Delegate Property

When implementing DuckProtocol in the Human class, we declare a delegate property and ask it for permission.

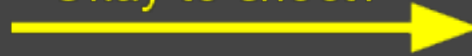
```
// Human.h
@interface Human : NSObject <DuckProtocol>
@property id<HumanShootingDelegate> delegateObj;
@end
```

```
// Human.m
@implementation
@synthesize delegateObj;
- (void) shootAtDuck:(id<DuckProtocol>)duck {
    if ([[self delegateObj] shouldShootAtDuck:self]) {
        // actual implementation of shooting
    }
}
@end
```

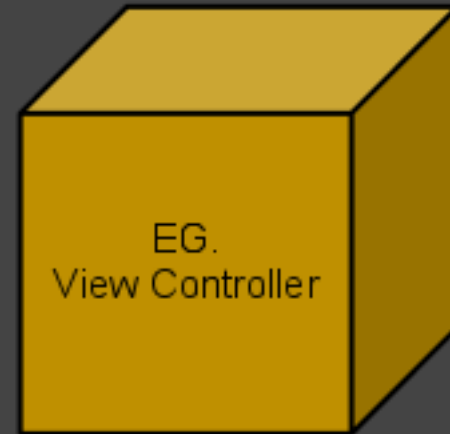
2.5 How It Works Now



Okay to shoot?



Yes, shoot em!



Human <DuckProtocol>
@property humanShootingDelegate;
- (void) quack;

Delegate Object <HumanShootingDelegate>
- (BOOL) shouldShootAtDuck;



DuckProtocol
(says object should quack)

2.6 Reviewing Delegate-Protocol Example

1. DuckProtocol - defines how a duck behaves
2. HumanShootingDelegate - defines when humans can shoot
3. Make Human class conform to DuckProtocol
4. Add HumanShootingDelegate property to Human class

Compiler confirms Human objects can quack and their delegate knows how to respond properly with YES or NO when they ask for permission to shoot.

2.7 Delegates - Formal Definition

"In software engineering, the delegation pattern is a technique where an object outwardly expresses certain behaviour but in reality delegates responsibility for implementing that behavior to an associated object in an Inversion of Responsibility." http://en.wikipedia.org/wiki/Delegation_pattern

"... is often used instead of subclassing: instead of subclassing a class to change its behaviour, you supply a delegate that responds to the appropriate methods." <http://stackoverflow.com/questions/2232147/whats-the-difference-between-data-source-and-delegate>

The **Delegation Pattern** is used *very frequently* in Cocoa.

2.8 How Cocoa Touch Uses Delegates

Your custom objects act can be delegated responsibility for certain behaviour of *Cocoa Touch* objects.

For example, it allows you to:

- Populate a UITableView with data
- Respond to touches on a table cell
- Read user input from a UITextView
- *Much more ...*

2.9 Ex: UITableView Delegate/Datasource Protocol

The delegate of a UITableView object must adopt the UITableViewDelegate protocol.

```
// MyTableVC.h
@interface MyTableVC : UIViewController
    <UITableViewDelegate, UITableViewDataSource>
@property (retain, nonatomic) IBOutlet UITableView *tableView;
@end
```

```
// MyTableVC.m
@implementation MyTableVC
- (void)viewDidLoad {
    [super viewDidLoad];
    tableView.delegate = self; // set VC as table's delegate
    tableView.dataSource = self; // set VC as table's datasource
}
@end
```

2.10 UITableView Delegate/Datasource

Methods

// Called when the table loads each cell

```
- (UITableViewCell *)tableView:(UITableView*)tableView  
    cellForRowAtIndexPath:(NSIndexPath*)indexPath;
```

// Called when the table loads the header section

```
- (NSString *)tableView:(UITableView*)tableView  
    titleForHeaderInSection:(NSInteger)section;
```

// Called when a cell is selected/tapped

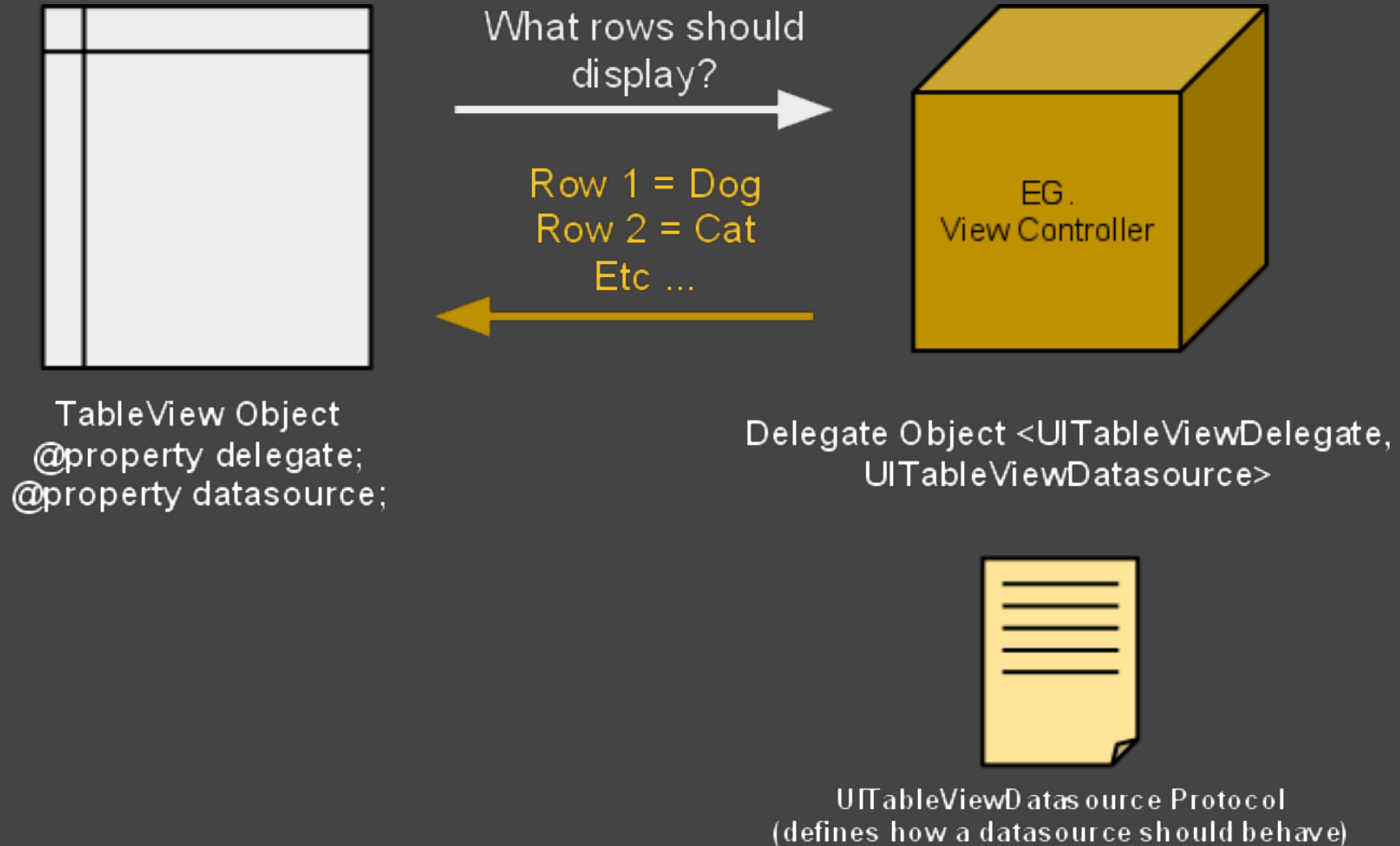
```
- (void)tableView:(UITableView*)tableView  
    didSelectRowAtIndexPath:(NSIndexPath*)indexPath;
```

Reference:

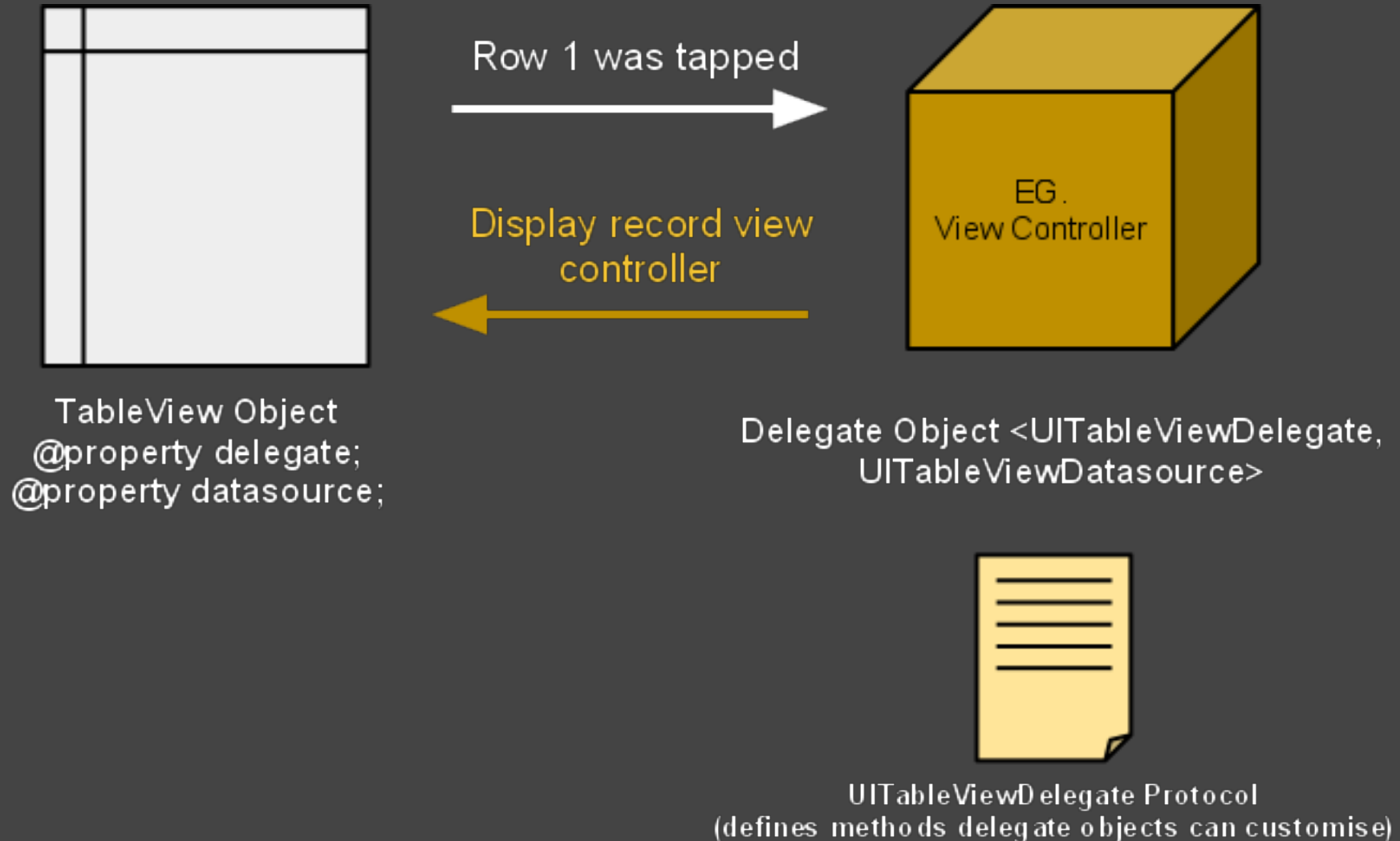
http://developer.apple.com/library/ios/#documentation/UIKit/Reference/UITableViewDelegate_Protocol/Reference/Reference.html

http://developer.apple.com/library/ios/#documentation/UIKit/Reference/UITableViewDataSource_Protocol/Reference/Reference.html

2.11 How UITableView Datasource Works



2.12 How UITableView Delegate Works



2.13 Ex: UITextViewDelegate Protocol

This protocol defines a set optional methods to receive editing-related messages for UITextView objects

```
// MyVC.h
@interface MyTableVC : UIViewController <UITextViewDelegate>
@property (nonatomic, retain) IBOutlet UITextView *textView;
@end

// MyVC.m
@implementation MyTableVC
@synthesize textView;

- (void)viewDidLoad {
    [super viewDidLoad];
    // tells text view this class is the delegate obj
    self.textView.delegate = self;
}
@end
```

2.14 UITextField Delegate Methods

// Called when text field gets the focus

- (void)textViewDidBeginEditing:(UITextView *)textView;

// Called when text in the field changes

- (void)textViewDidChange:(UITextView *)textView;

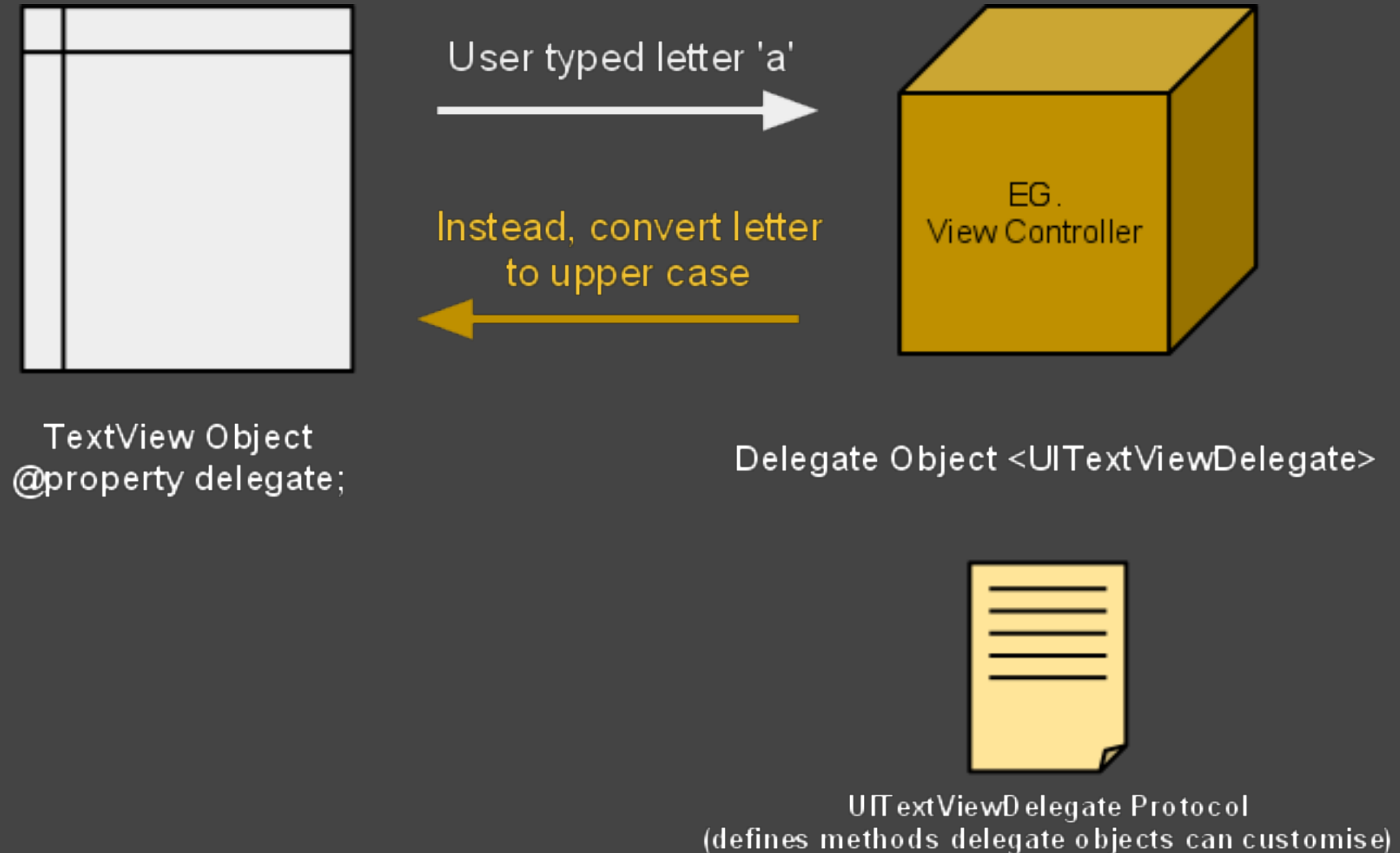
// Called when the text selection changes

- (void)textViewDidChangeSelection:(UITextView *)textView;

Reference:

http://developer.apple.com/library/ios/#documentation/UIKit/Reference/UITextViewDelegate_Protocol/Reference/UITextViewDelegate.html

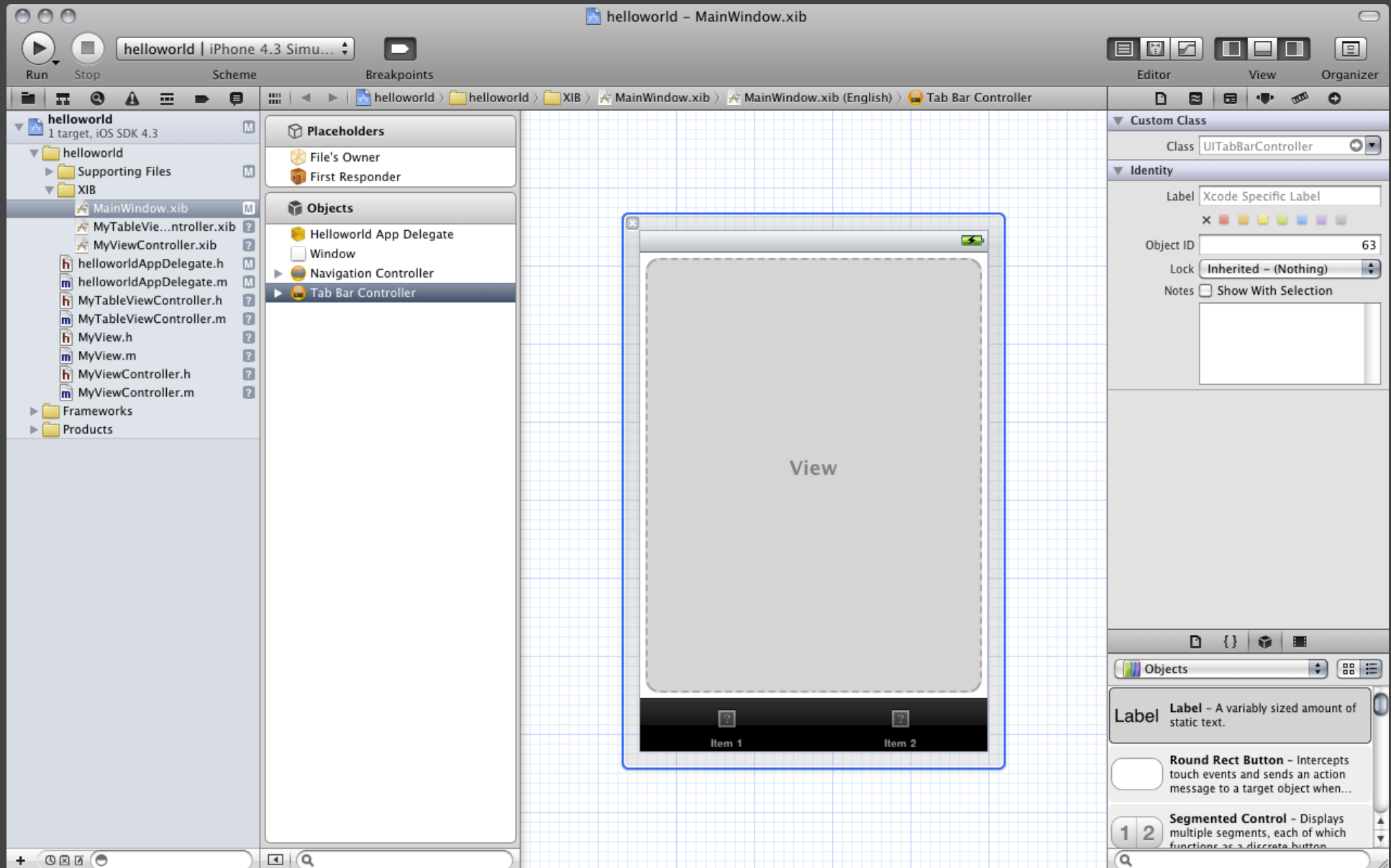
2.15 How UITextView Delegate Works



What's On For Today?

1. Protocols
2. Delegates & Datasources
- 3. Interface Builder**
4. Application Object
5. Collections
6. File IO

3.0 Interface Builder



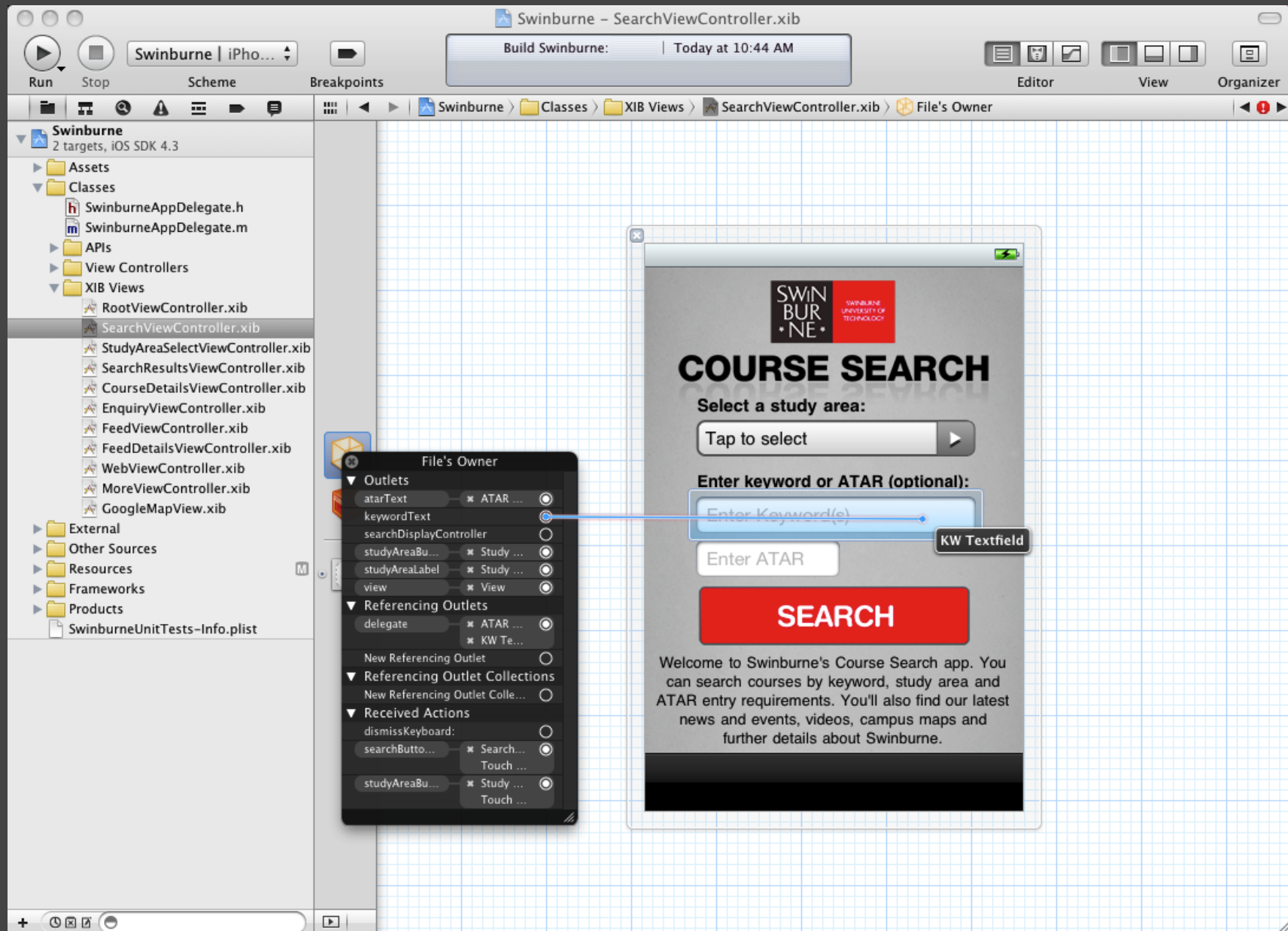
3.1 What is Interface Builder (IB)?

- IB lets you create user interfaces visually and produces NIB/XIB files
- XIBs are unarchived into memory at runtime
- XIB contents are described as:
 - *"frozen objects"*
 - *"freeze dried objects"*
 - *"ready made objects"*

How do we integrate custom code with a XIB file's archived objects?

3.2 What is an IB Outlet?

- Outlets connect *IB objects* to *source code*
- Each IB object has one or more outlets
- We visually connect objects to outlets
- Only connectable to certain class properties



3.2.1 Interface Builder Outlets

3.3 Connecting Outlets to Properties

1. Add IBOutlet metadata tag to a *declared property*
2. It appears in the list of available *IBOutlets* for a class*

For Example:

```
@property (nonatomic, retain) IBOutlet UITextView *txtView;
```

** IBOutlet is not a data type, it's a "metadata hint" declaring the properties IB can use.*

3.4 Connecting Actions to Methods

1. Set method return type to `IBAction` in header file
2. Method will appear in list of available *IBAction* event handler outlets for a class*

For Example:

```
-(IBAction)onButtonClicked:(id)sender;
```

How do we associate a XIB with a particular class?

** IBAction is also not a data type, it is metadata and resolves to void when compiled*

3.5 "File's Owner" Property

- XIB file property defining owning source file
- Each *XIB file* has one *File's Owner*
- Owning source file usually provides logic for controlling / responding to objects in the XIB

3.6 Customising XIB/NIBs in Code

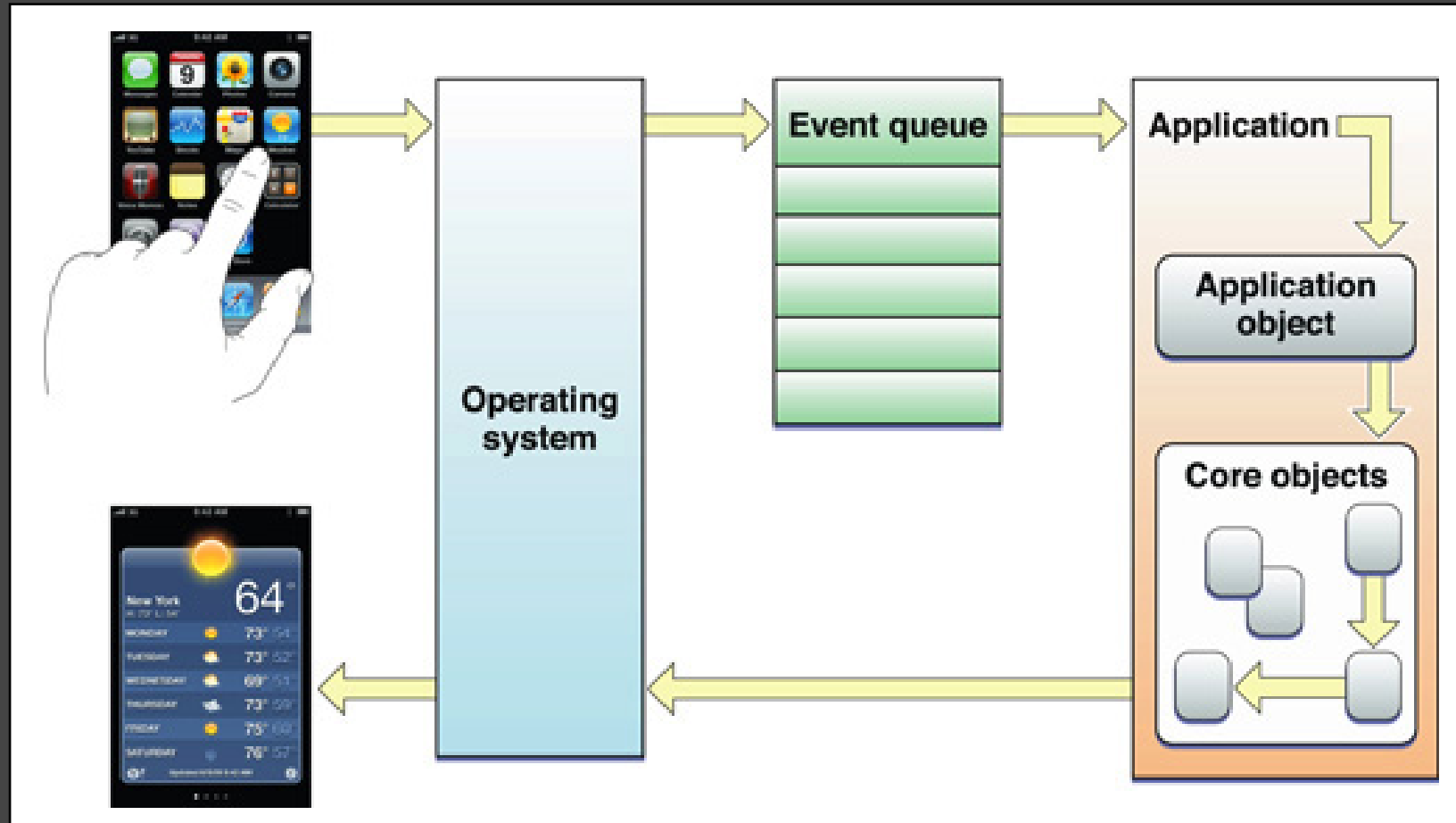
```
- (void)awakeFromNib  
{  
    // perform customizations  
}
```

This method is called after the XIB/NIB's archived objects have been instantiated and connected to their IBOutlets/IBActions.

What's On For Today?

1. Protocols
2. Delegates & Datasources
3. Interface Builder
- 4. Application Object**
5. Collections
6. File IO

4.0 Application Object



Source: <http://developer.apple.com/library/ios/#DOCUMENTATION/General/Conceptual/Devpedia-CocoaApp/MainEventLoop.html>

4.1 What is the Application Object?

- Each app has one Application Object
- Application Object is created in main()
- App behaviour is driven by external events
- (eg. touches, movement, etc...)
- Run loop reads events from a FIFO queue

Read More About the App Lifecycle: <http://www.codeproject.com/KB/iPhone/ApplicationLifeCycle.aspx>

4.2 App Delegate

- This is the actual point where your code starts executing in ANY iOS app
- Can respond to events like start up, shut down, wake from background, etc...
- Usually declared in `<MyName>AppDelegate.m/h`

4.3 Anatomy of an App Delegate I

```
//  
// ExampleAppDelegate.h  
//
```


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

```
@interface AppDelegate : NSObject <UIApplicationDelegate> {  
    UIWindow *window;  
    UIViewController *viewController;  
}
```

Public property to
Window object



View controller
loaded on startup



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

4.4 Anatomy of an App Delegate II

```
// ExampleAppDelegate.m
#import "ExampleAppDelegate.h"
#import "SomeViewController.h"; // Defined elsewhere
```

```
@implementation ExampleAppDelegate
@synthesize window, viewController;
```

```
- (void)applicationDidFinishLaunching:(UIApplication*)application {
    self.viewController = [[SomeViewController alloc]
                           initWithNibName:nil bundle:nil];
    [window addSubview:viewController.view];
    [window makeKeyAndVisible];
}

- (void)dealloc {
    [viewController release];
    [window release];
    [super dealloc];
}
@end
```

4.5 Anatomy of an App Delegate III

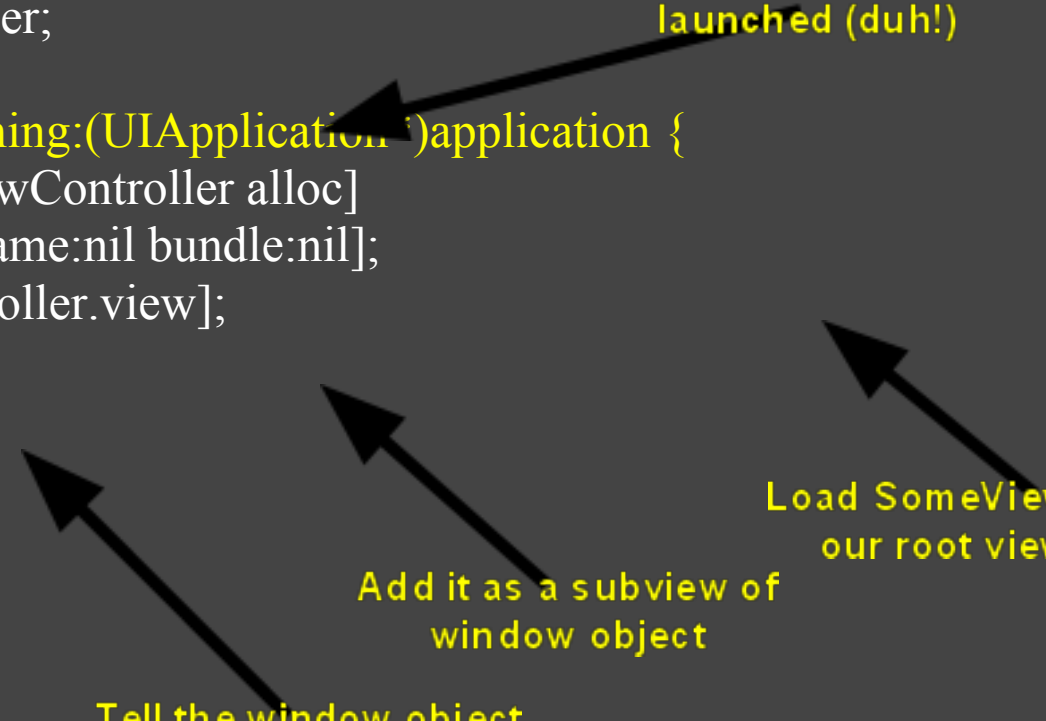
```
// ExampleAppDelegate.m
#import "ExampleAppDelegate.h"
#import "SomeViewController.h"; // Defined elsewhere
```

```
@implementation ExampleAppDelegate
@synthesize window, viewController;
```

```
- (void)applicationDidFinishLaunching:(UIApplication *)application {
    self.viewController = [[SomeViewController alloc]
                           initWithNibName:nil bundle:nil];
    [window addSubview:viewController.view];
    [window makeKeyAndVisible];
}
```

```
- (void)dealloc {
    [viewController release];
    [window release];
    [super dealloc];
}
@end
```

Called when app has
launched (duh!)



Load SomeViewController as
our root view controller

Add it as a subview of
window object

Tell the window object
to appear (draw)

What's On For Today?

1. Protocols
2. Delegates & Datasources
3. Interface Builder
4. Application Object
- 5. Collections**
6. File IO

5.0 Collections

ObjC supports well featured collection objects

- **Arrays:** NSArray, NSMutableArray
- **Dictionaries:** NSDictionary, NSMutableDictionary
- **Sets:** NSSet, NSMutableSet, NSCountedSet

Characteristics:

- **Arrays** - zero indexed, stores ordered objects
- **Dictionaries** - indexed by keys, keys can be numbers, string, any NSObject (true!)
- **Sets** - stores unique, unordered objects

5.1 Enumeration (Loops)

Traversing collections in Objective-C is easy!

```
for (NSString* tmpString in [someObject collectionOfObjects]) {  
    // do something with object  
}
```

5.2 Enumeration (continued)

- Arrays, Dictionaries and Sets each conform to the `NSFastEnumeration` protocol, so they work with 'for' loops
- 'For' loops are optimised by the ObjC Runtime
- The *collection expression* is evaluated only once, at the start of the loop, so it is quite efficient

5.3 Collections Documentation

Get to know, love and endure the documentation ...

NSArray

http://developer.apple.com/library/ios/#documentation/cocoa/reference/foundation/Classes/NSArray_Class/NSArray.html

NSMutableArray

□ http://developer.apple.com/library/ios/#documentation/Cocoa/Reference/Foundation/Classes/NSMutableArray_Class/Reference/Reference.html

NSDictionary

□ http://developer.apple.com/library/ios/#documentation/Cocoa/Reference/Foundation/Classes/NSDictionary_Class/Reference/Reference.html

NSMutableDictionary

□ http://developer.apple.com/library/ios/#documentation/Cocoa/Reference/Foundation/Classes/NSMutableDictionary_Class/Reference/Reference.html

NSSet

□ http://developer.apple.com/library/ios/#documentation/Cocoa/Reference/Foundation/Classes/NSSet_Class/Reference/Reference.html

NSMutableSet

□ http://developer.apple.com/library/ios/#documentation/Cocoa/Reference/Foundation/Classes/NSMutableSet_Class/Reference/Reference.html

5.4 What You Need To Know?

Make sure you know how to:

- Create a collection
- Add and remove objects
- Check if an object is in a collection
- Iterate/loop collections

Details are in the docs.

5.5 Adding To Collections

- Retain/Release

// Add to collection idiom

```
NSObject *myObj = [[NSObject alloc] init]; // retainCount is 1  
[myArray addObject:myObj];                // retainCount is 2  
[someObject release];                     // retainCount is 1 again
```

Remember:

Objects are retained by their owning collections.

Be sure to *relinquish ownership* in your code after adding to a collection.

What's On For Today?

1. Protocols
2. Delegates & Datasources
3. Interface Builder
4. Application Object
5. Collections
- 6. File IO**

6.0 File IO

- File system access via NSFileManager
- Apps run in a security sandbox
- File contexts:
 - **Application Bundle** - Read Only access
 - **Documents Directory** - R&W access
 - **Library Directory** - R&W access
- Apps cannot see file storage of other apps

6.1 File IO: Path to File in Bundle

An *Application Bundle* is created by Xcode at build time. It is a special directory for grouping together resources relating to a single program.

```
// Request path to file in bundle
NSString *filePath =
[[NSBundle mainBundle] pathForResource:@"mydata" ofType:@"txt"];
```

NB: The application bundle is read only.

6.2 File IO: Path to File in Documents

Each app has one *Documents Folder* for storing temporary or editable file based resources.

```
// Get documents directory (first object in array)
```

```
NSArray *paths = NSSearchPathForDirectoriesInDomains(NSDocumentDirectory, NSUserDomainMask, YES);
```

```
NSString *docDirPath = [paths objectAtIndex:0];
```

```
// Append file name to documents folder path
```

```
NSString *filePath = [docDirPath stringByAppendingPathComponent:@"mydata.txt"];
```

NB: *The documents folder contents are not deleted when upgrading or re-installing an existing app.*

6.3 File IO: Read Text Data

```
NSString *filePath =  
    [[NSBundle mainBundle] pathForResource:@"mydata" ofType:@"txt"];  
  
NSString *textData =  
    [NSString stringWithContentsOfFile:filePath  
        encoding: NSUTF8StringEncoding error:nil];  
if (textData)  
{  
    // now do something  
}
```

6.4 File IO: Read Binary Data

```
NSString *filePath =  
    [[NSBundle mainBundle] pathForResource:@"mydata" ofType:@"dat"];  
  
NSData *fileData = [NSData dataWithContentsOfFile:filePath];  
  
if (fileData)  
{  
    // now do something  
}
```

6.5 File IO: Write to a File

Only Document & Library directories can be written to

```
// Create NSError pointer
```

```
NSError *error;
```

```
// Get documents directory (first object in array)
```

```
NSArray *paths = NSSearchPathForDirectoriesInDomains(NSDocumentDirectory, NSUserDomainMask, YES);
```

```
NSString *docDirPath = [paths objectAtIndex:0];
```

```
// Append file name to documents folder path
```

```
NSString *filePath = [docDirPath stringByAppendingPathComponent:@"mydata.txt"];
```

```
// Create a string to write
```

```
NSString *str = [NSString stringWithString:@"Write this string to my file!"];
```

```
// Write string to file
```

```
[str writeToFile:filePath atomically:YES encoding:NSUTF8StringEncoding error:&error];
```

6.6 File IO: Delete a File

Files in Document & Library directories can be deleted

```
// Get documents directory (first object in array)
```

```
NSArray *paths = NSSearchPathForDirectoriesInDomains(NSDocumentDirectory, NSUserDomainMask, YES);
```

```
NSString *docDirPath = [paths objectAtIndex:0];
```

```
NSString *filePath = [docDirPath stringByAppendingPathComponent:@"mydata.txt"];
```

```
// Ask shared NSFileManager object to delete file
```

```
[[NSFileManager defaultManager] removeItemAtPath:filePath error:nil];
```

What We Covered Today

1. Protocols
2. Delegates & Datasources
3. Interface Builder
4. Application Object
5. Collections
6. File IO

End of Lecture 3

1. Lab
2. Assignments
3. Get Your Snack On