Announcements

- **Lab 1 is due tonight by midnight**
- **Lab 2 is now posted**
  - Due Monday Feb 9th
- **On Monday we will meet in the Mac Lab**
  - Two sessions 8:30 AM and 10 AM

Lab 2

Today's Topics

- Autorelease
- Automatic Reference Counting (ARC)
- Objective-C Properties
- Quick Intro to Debugging
- Application Lifecycle
- Model, View, Controller design
- Controls and Target-Action

Autorelease

Returning a newly created object

```
-(NSString *)fullName {
  NSString *result;

  result = [[NSString alloc] initWithFormat:@"%@ %@",
    firstName, lastName];

  return result;
}
```

- Wrong: result is leaked!

Returning a newly created object

```
-(NSString *)fullName {
  NSString *result;

  result = [[NSString alloc] initWithFormat:@"%@ %@",
    firstName, lastName];

  [result release];
  return result;
}
```

- Wrong: result is released too early!
- Uncertain what method returns

Returning a newly created object

```
-(NSString *)fullName {
  NSString *result;

  result = [[NSString alloc] initWithFormat:@"%@ %@",
    firstName, lastName];

  [result autorelease];
  return result;
}
```

- Just right: result is released, but not right away!
- Caller gets valid object and could retain if needed

Autoreleasing Objects

- Calling -autorelease flags an object to be sent release at some point in the future
- Let's you fulfill your retain/release obligations while allowing an object some additional time to live
- Makes it much more convenient to manage memory
- Very useful in methods which return a newly created object

Method Names & Autorelease

- Methods whose names includes **alloc** or **copy** return a retained object that the caller needs to release

```
NSMutableString *string = [[NSMutableString alloc] init];

// We are responsible for calling -release or –autorelease
[string autorelease];
```

- All other methods return autoreleased objects

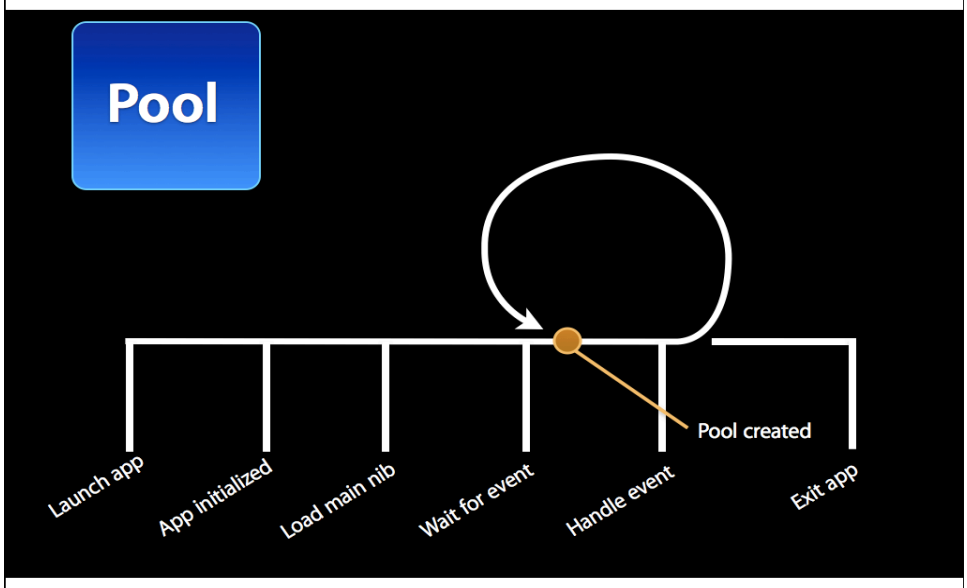
```
NSMutableString *string = [NSMutableString string];

// The method name doesn't indicate that we need to release it
// So don't- we're cool!
```

- This is a convention
  - follow it in methods you define

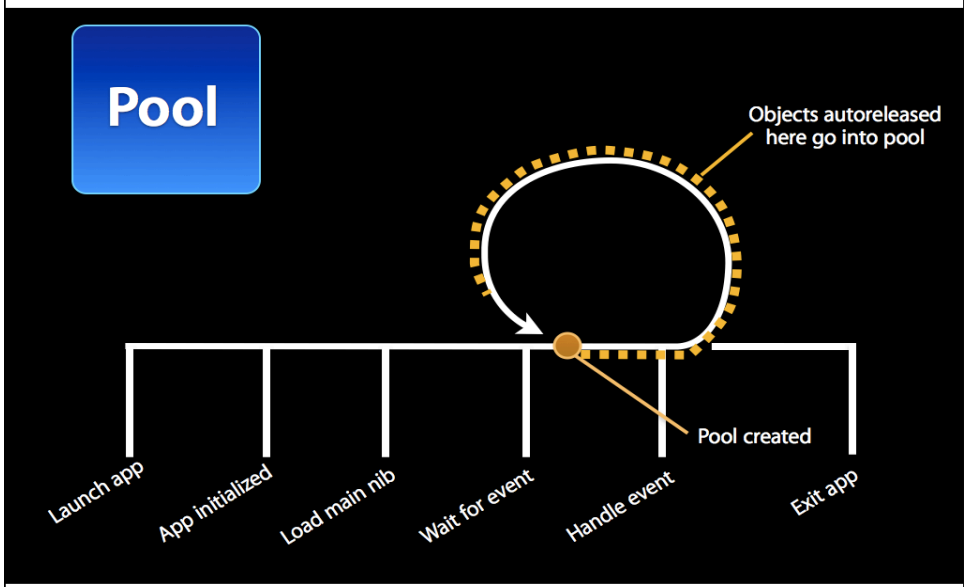
How does -autorelease work?

- Object is added to current autorelease pool
- Autorelease pools track objects scheduled to be released
  - When the pool itself is released, it sends -release to all its objects
- UIKit automatically wraps a pool around every event dispatch

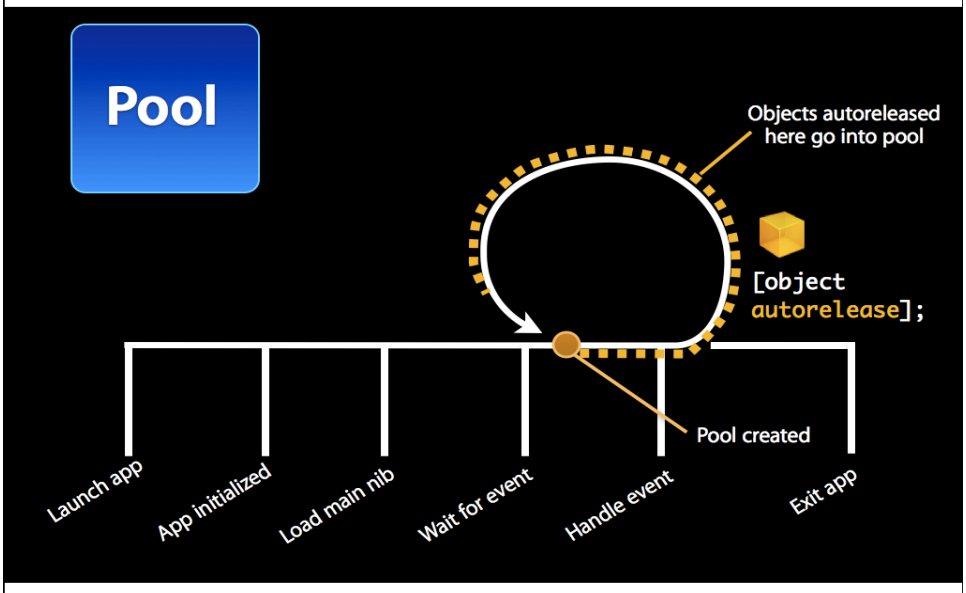
Autorelease Pools (from cs193p slides)



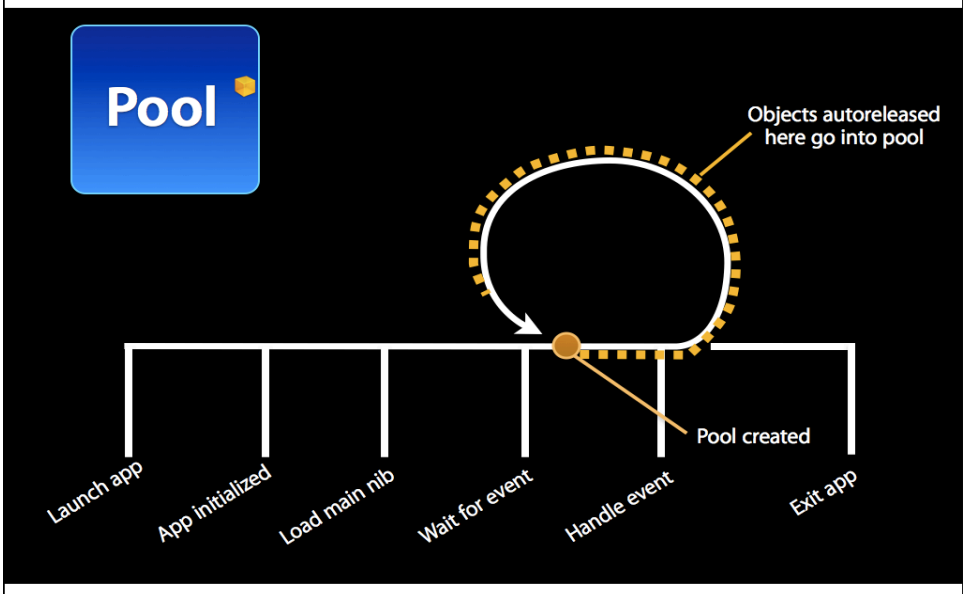
Autorelease Pools (from cs193p slides)



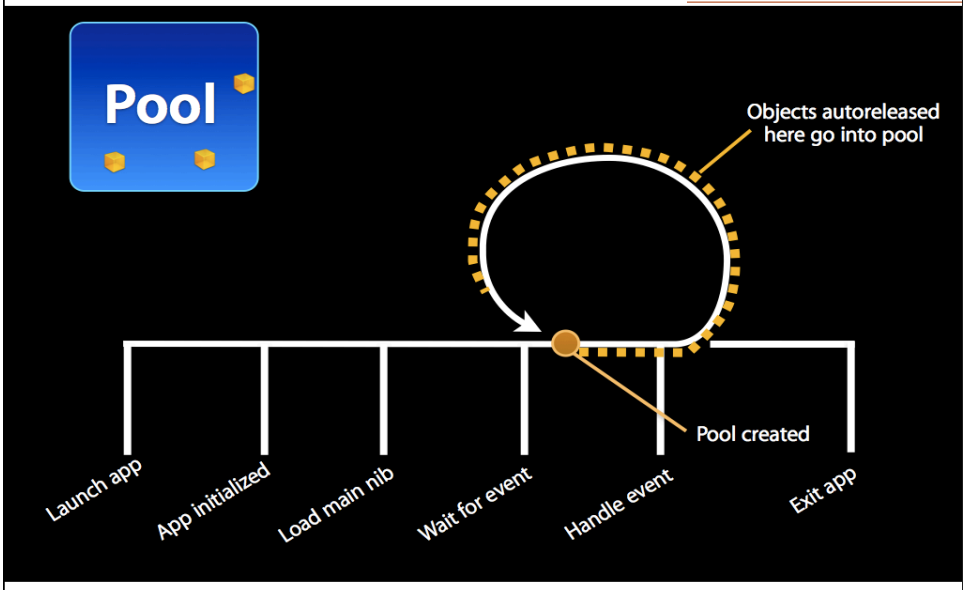
Autorelease Pools (from cs193p slides)



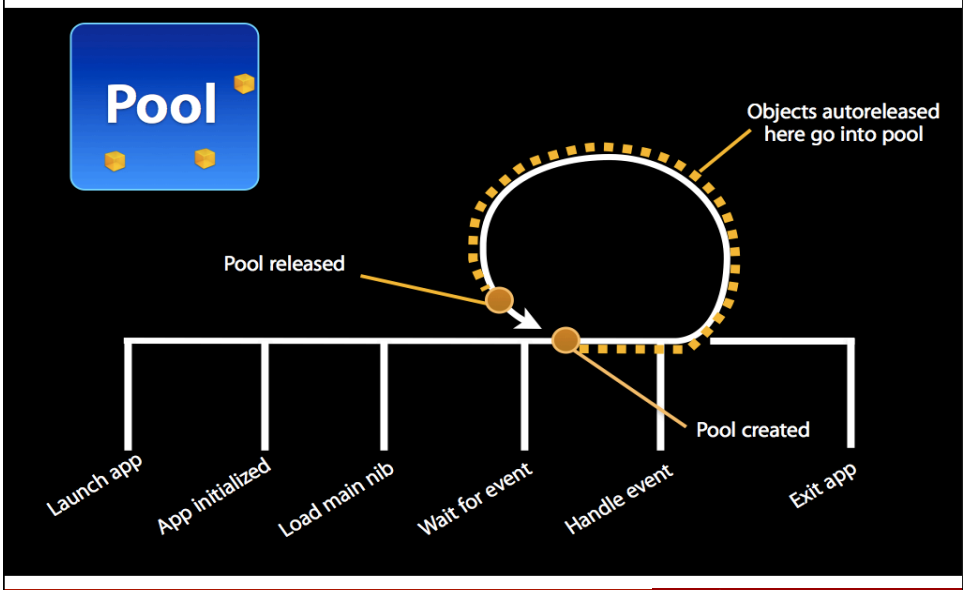
Autorelease Pools (from cs193p slides)



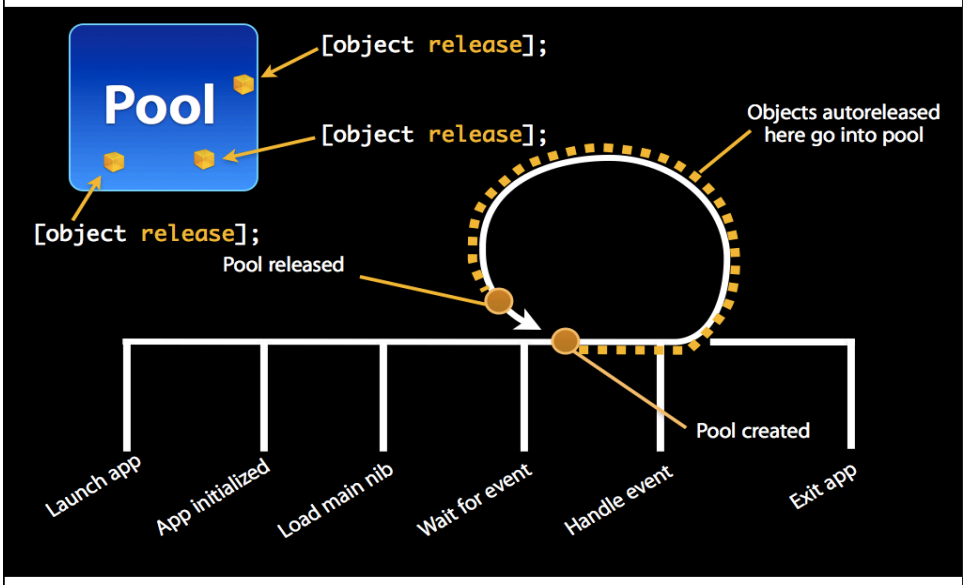
Autorelease Pools (from cs193p slides)



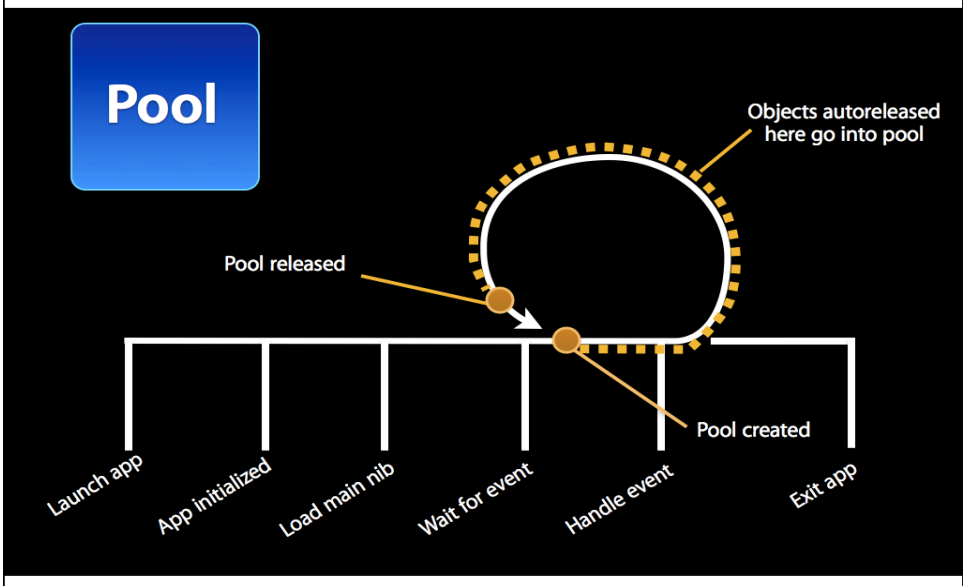
Autorelease Pools (from cs193p slides)



Autorelease Pools (from cs193p slides)



Autorelease Pools (from cs193p slides)



Hanging Onto an Autoreleased Object

- **Many methods return autoreleased objects**
  - Remember the naming conventions...
  - They're hanging out in the pool and will get released later
- **If you need to hold onto those objects you need to retain them**
  - Bumps up the retain count before the release happens

```
name = [NSMutableString string];

// We want to name to remain valid!
[name retain];
// ...
// Eventually, we'll release it (maybe in our -dealloc?)
[name release];
```


Side Note: Garbage Collection

- **Autorelease is not garbage collection**
- **Objective-C on iPhone OS (iOS) does not have garbage collection**

Automatic Reference Counting (ARC)

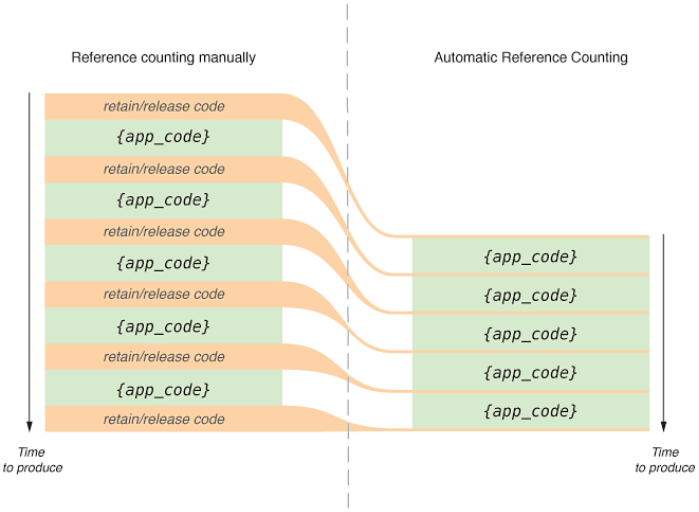
Automatic Reference Counting (ARC)

- **The new and "improved" way to manage memory**
  - All objects are either strong or weak
- **Strong**
  - Keep me around until I no longer need this memory
- **Weak**
  - Keep me around as long as some other object needs this memory

Automatic Reference Counting

- **By default all objects allocated when using ARC are strong**
  - NSNumber *myNumber = [NSNumber alloc] init];
- **Weak references are often used when pointing to objects on a storyboard**
  - UIButton, UILabel, UIImage
    - These objects are already instantiated when the storyboard loads
    - We just want a pointer to them while they are alive

Automatic Retain/Release



Properties

- Provide access to object attributes
- Shortcut to implementing getter/setter methods
- Also allow you to specify:
  - read-only versus read-write access
  - memory management policy

Defining Properties

```
#import<Foundation/Foundation.h>
@interface Person : NSObject
{
// instance variables
   NSString *name;
   int age;
}

// method declarations
-(NSString *)name;
-(void)setName:(NSString *)value;
-(int)age;
-(void)setAge:(int)age;
-(BOOL)canLegallyVote;

-(void)castBallot;
@end
```

Defining Properties

```
#import<Foundation/Foundation.h>
@interface Person : NSObject
{
// instance variables
   NSString *name;
   int age;
}

// method declarations
-(NSString *)name;
-(void)setName:(NSString *)value;
-(int)age;
-(void)setAge:(int)age;
-(BOOL)canLegallyVote;

-(void)castBallot;
@end
```


Defining Properties

```
#import<Foundation/Foundation.h>
@interface Person : NSObject
{
// instance variables
   NSString *name;
   int age;
}

// method declarations
-(NSString *)name;
-(void)setName:(NSString *)value;
-(int)age;
-(void)setAge:(int)age;
-(BOOL)canLegallyVote;

-(void)castBallot;
@end
```

Defining Properties

```
#import<Foundation/Foundation.h>
@interface Person : NSObject
{
// instance variables
   NSString *name;
   int age;
}

// property declarations
@property int age;
@property (copy) NSString  *name;
@property (readonly) BOOL canLegallyVote;

-(void)castBallot;
@end
```


Synthesizing Properties

```
@implementation Person

-(int)age {
 return age;
 }

-(void)setAge:(int)value {
  age = value;
 }

-(NSString *)name {
  return name;
 }

-(void)setName:(NSString *)value {
   if (value != name) {
      [value release];
      name = [value copy];
   }

- (BOOL)canLegallyVote { …
```

Synthesizing Properties

```
@implementation Person

-(int)age {
  return age;
 }

-(void)setAge:(int)value {
   age = value;
 }

-(NSString *)name {
   return name;
 }

-(void)setName:(NSString *)value {
    if (value != name) {
        [value release];
        name = [value copy];
    }

- (BOOL)canLegallyVote { …
```


Synthesizing Properties

```
@implementation Person

-(int)age {
  return age;
 }

-(void)setAge:(int)value {
   age = value;
 }

-(NSString *)name {
   return name;
 }

-(void)setName:(NSString *)value {
    if (value != name) {
        [value release];
        name = [value copy];
    }

- (BOOL)canLegallyVote { …
```

Synthesizing Properties

```
@implementation Person

@synthesize age;
@synthesize name;
-    (BOOL)canLegallyVote {
   return (age > 17);
}

@end
```


iOS Property Attributes

- **Use strong and weak instead of retain and assign**

```
@property (retain) NSString *name; // retain called
@property (strong) NSString *name;// new way


@property (assign) NSString *name; // pointer assignment
@property (weak) NSString *name; //new way
```

Property Names vs. Instance Variables

- Property name can be different than instance variable

```
@interface Person : NSObject {
  int numberOfYearsOld;
}

@property int age;

@end


@implementation Person

@synthesize age = numberOfYearsOld;

@end
```


Properties

- Mix and match synthesized and implemented properties

```
@implementation Person

@synthesize age;
@synthesize name;

(void)setAge:(int)value {
    age = value;
}
@end
```

- Setter method explicitly implemented
- Getter method still synthesized

Properties In Practice

- **Newer APIs use @property**
- **Older APIs use getter/setter methods**
- **Properties used heavily throughout UIKit APIs**
  - Not so much with Foundation APIs
- **You can use either approach**
  - Properties mean writing less code, but "magic" can sometimes be non-obvious

Further Reading

- **Objective-C 2.0 Programming Language**
  - "Defining a Class"
  - "Declared Properties"
- **Memory Management Programming Guide for Cocoa**

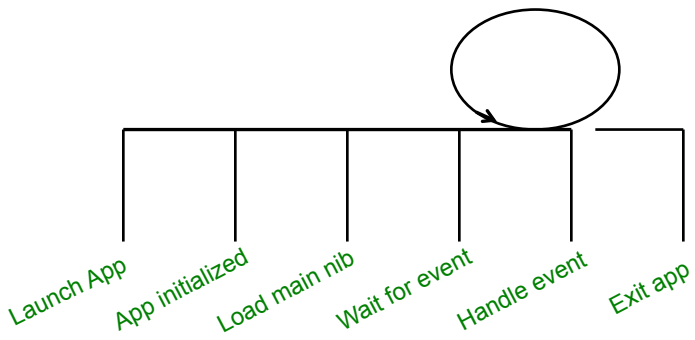
ARC and Properties Demo

Building an Application

Anatomy of an Application

- **Compiled code**
  - Your code
  - Frameworks
- **Nib files**
  - UI elements and other objects
  - Details about object relationships
- **Resources (images, sounds, strings, etc)**
- **Info.plist file (application configuration)**

App Lifecycle



UIKit Framework

- **Provides standard interface elements**
- **UIKit and you**
  - Don't fight the frameworks
  - Understand the designs and how you fit into them

UIKit Framework

- **Starts your application**
- **Every application has a single instance of UIApplication**
  - Singleton design pattern

```
@interface UIApplication
 + (UIApplication *)sharedApplication
@end
```

- **Orchestrates the lifecycle of an application**
- **Dispatches events**
- **Manages status bar, application icon badge**
- **Rarely subclassed**
  - Uses delegation instead

Delegation

- Control passed to delegate objects to perform application-specific behavior
- Avoids need to subclass complex objects
- Many UIKit classes use delegates
  - UIApplication
  - UITableView
  - UITextField

UIApplicationDelegate

- Xcode project templates have one set up by default
- Object you provide that participates in application lifecycle
- Can implement various methods which UIApplication will call
- Examples:

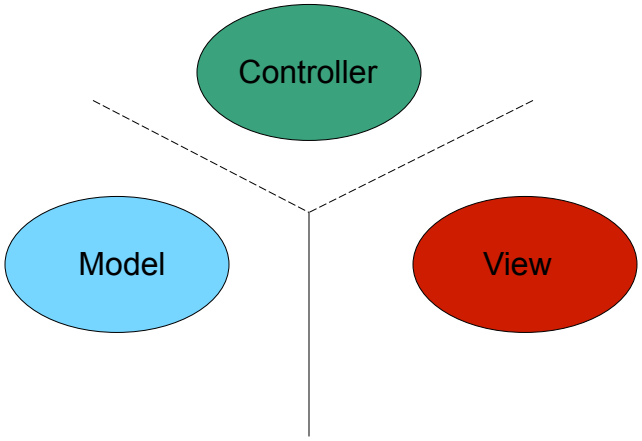
```
-(void)applicationDidFinishLaunching:(UIApplication *)application;
-(void)applicationWillTerminate:(UIApplication *)application;

-(void)applicationWillResignActive:(UIApplication *)application;
-(BOOL)application:(UIApplication *)application handleOpenURL:(NSURL *)url;

-(void)applicationDidReceiveMemoryWarning:(UIApplication *)application;
```

Info.plist file

- **Property List (often XML), describing your application**
  - Icon appearance
  - Status bar style (default, black, hidden)
  - Orientation
  - Uses Wifi networking
  - System Requirements
- **Can edit most properties in Xcode**
  - Project > Edit Active Target "Foo" menu item
  - On the properties tab

Model, View, Controller

Model, View, Controller



Model

- Manages the app data and state
- Not concerned with UI or presentation
- Often persists somewhere
- Same model should be reusable, unchanged in different interfaces

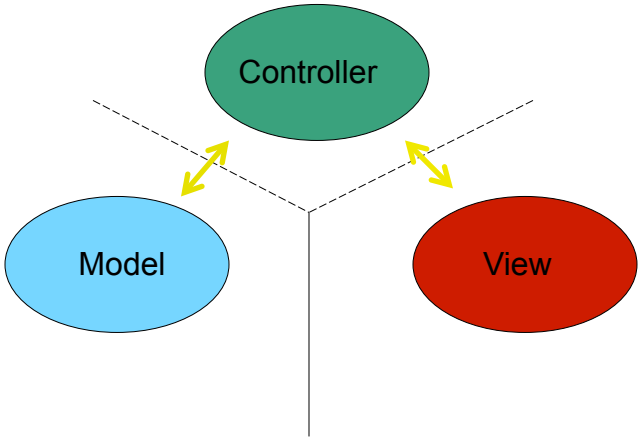
View

- Present the Model to the user in an appropriate interface
- Allows user to manipulate data
- Does not store any data
  - (except to cache state)
- Easily reusable & configurable to display different data

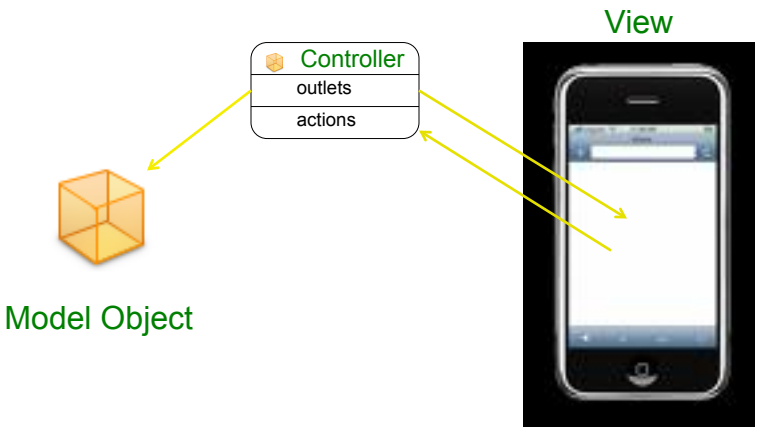
Controller

- Intermediary between Model & View
- Updates the view when the model changes
- Updates the model when the user manipulates the view
- Typically where the app logic lives

Model, View, Controller



Model, View, Controller

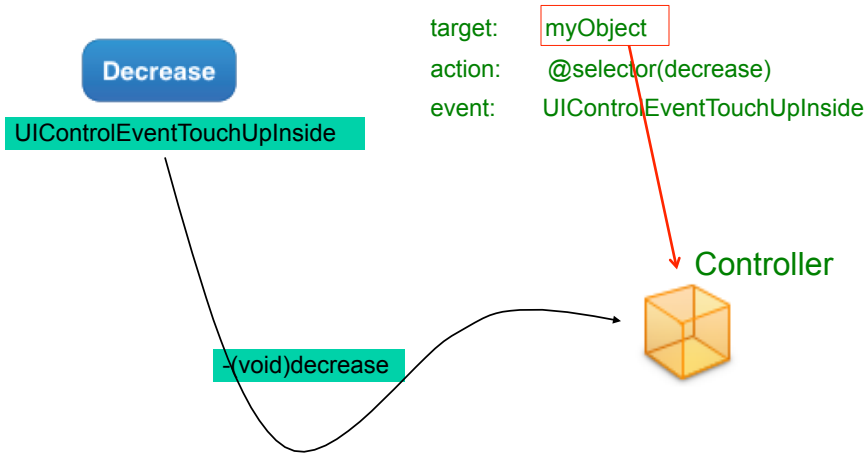


Controls and Target-Action

Controls - Events

- View objects that allows users to initiate some type of action
- Respond to variety of events
  - Touch events
    - touchDown
    - touchDragged (entered, exited, drag inside, drag outside)
    - touchUp (inside, outside)
  - Value changed
  - Editing events
    - editing began
    - editing changed
    - editing ended

Controls - Target/Action



Action Methods

- **3 different flavors of action method selector types**

```
-(void)actionMethod;
-(void)actionMethod:(id)sender;
-(void)actionMethod:(id)sender withEvent:(UIEvent *)event;
```

- **UIEvent contains details about the event that took place**

Action Method Variations

- **Simple no-argument method**

```
-(void)increase {
  // bump the number of sides of the polygon up
     polygon.numberOfSides += 1;
  }
```

- **Single argument method - control is 'sender'**

```
// for example, if control is a slider...
-(void)adjustNumberOfSides:(id)sender {
   polygon.numberOfSides = [sender value];
 }
```


Action Method Variations

- **Two-arguments in method (sender & event)**

```
-(void)adjustNumberOfSides:(id)sender
  withEvent:(UIEvent *)event{
  // could inspect event object if you  needed to
}
```

Manual Target-Action

- Same information IB would use
- API and UIControlEvents found in UIControl.h
- UIControlEvents is a bitmask

```
@interface UIControl
-(void)addTarget:(id)target action:(SEL)action forControlEvents:
    (UIControlEvents)controlEvents;

-(void)removeTarget:(id)target action:(SEL)action forControlEvents:
    (UIControlEvents)controlEvents;@end
```


Multiple target-actions

- Controls can trigger multiple actions on different targets in response to the same event
- Different than Cocoa on the desktop where only one target-action is supported
- Different events can be setup in IB

Questions?

Debugging Intro

iPhone University Program

How do I run my apps on my phone?

Outline of Steps Required

- **Create an Apple ID**
  - Most of you have done this already
  - Necessary to download SDK
- **Enroll in the iPhone University Program**
- **Generate a Certificate Signing Request (CSR)**
- **Send me your iPhone or iPod Touch UDID**
- **Download a provisioning profile**

iPhone University Program

- **Three levels of membership**
  - Team Agent
  - Team Admin
  - Team Members
- **All members have the ability to run apps on iPhones**
- **Team Agent and Team Admin have very similar roles**
  - Add new members and admins
  - Add devices to the group
  - Approve CSRs
  - Create provisioning profiles
- **Team Members**
  - Download provisioning profiles
  - Submit CSR
- **I am one of twoTeam Admins for WashU**
- **Rick Tyler is the WashU Team Agent**

Digital Certificate

- **All iPhone applications must be signed by a valid certificate before they can run on an Apple device**
- **In order to sign applications for testing, Team Members need an iPhone Development Certificate**
- **This is accomplished through a digital identity**
  - Consists of a secret "private key" and a shared "public key"
  - The private key allows Xcode to sign your iPhone OS application binary
- **The digital certificate associates your digital identity with other information**
  - name, email address, business..
- **The iPhone development certificate is restricted to application development only and valid for a limited time**

Requesting a Development Certificate

- **Generate a Certificate Signing Request (CSR)**
  - Use Keychain Access in Max OS X Leopard
  - http://developer.apple.com/iphone/download.action?path=/iphone/iphone_developer_program_at_a_glance/at_a_glance_v1.pdf
- **Upload the CSR to the iPhone University Website Portal**
- **I will then approve the certificate request and your will receive a confirmation email**

iPhone UDID

- **I also need your iPhone or iPod Touch devices Unique Device Identifier (UDID)**
  - 40 character string tied to a single device.
- **Locate the UDID in Xcode**
  - Windows -> Organizer
  - email me this ID and I will add your device