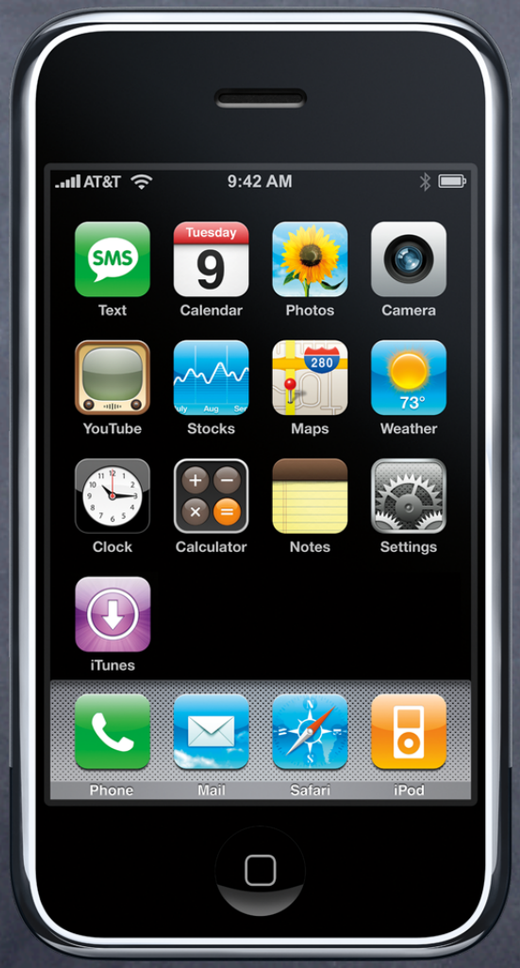
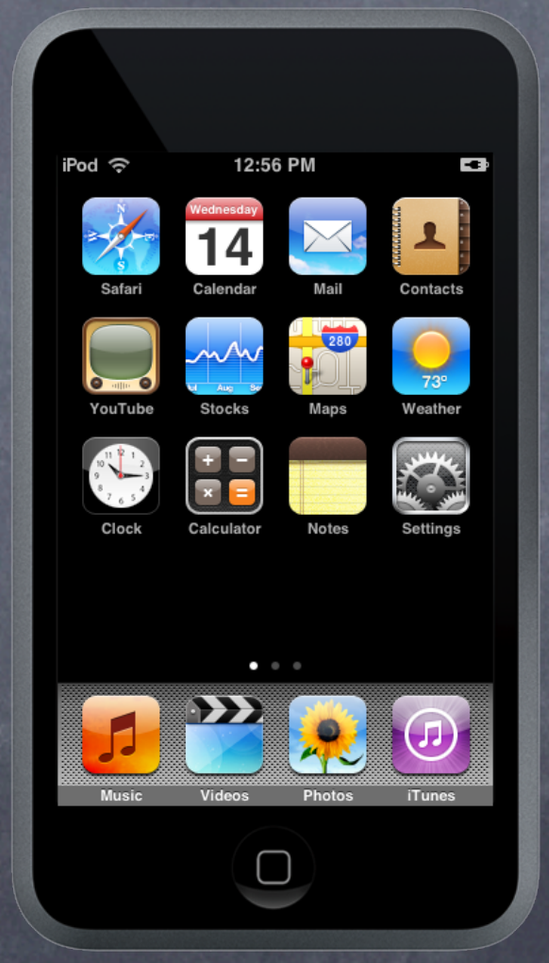
CS193p

Spring 2010





Announcements

- Get your Axess situation right.
- If you have not turned in homework, e-mail us.
  As they say in Air Traffic Control: state your intentions.
- Homework
  Any questions about the homework?

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

- Foundation Framework
  NSArray, NSDictionary, NSSet
  NSUserDefaults, etc.
- Objective-C
  Protocols and Delegates
- Memory Management
  Allocating and initializing objects
  Reference Counting
- Demo!

NSArray

- NSArray

  Ordered collection of objects

  Cannot be modified once created

  Important methods:

```
- (int)count
- (id)objectAtIndex:(int)index
- (void)makeObjectsPerformSelector:(SEL)aSelector
- (NSArray *)sortedArrayUsingSelector:(SEL)aSelector
```

- NSMutableArray

  Modifiable version of NSArray

```
- (void)addObject:(id)object
- (void)insertObject:(id)object atIndex:(int)index
- (void)removeObjectAtIndex:(int)index
- (void)replaceObjectAtIndex:(int)index withObject:(id)object
```

NSDictionary

- NSDictionary

  Cannot be modified once created!

  Look up a value using a key (aka a "hash table")

  A key must implement `- (NSUInteger)hash` and `- (BOOL)isEqual:(NSObject *)obj`

  Usually keys are `NSString` objects (since that implements those two)

  Important methods:

  ```
  - (int)count
  - (id)objectForKey:(id)key
  - (NSArray *)allKeys
  - (NSArray *)allValues
  ```

- NSMutableDictionary

  Modifiable version of `NSDictionary`

  ```
  - (void)setObject:(id)object forKey:(id)key
  - (void)removeObjectForKey:(id)key
  - (void)addEntriesFromDictionary:(NSDictionary *)dictionary
  ```

NSSet

- NSSet

  Unordered collection of objects without duplicates

  Cannot be modified once created

  Important methods:

  ```
  - (int)count
  - (BOOL)containsObject:(id)object
  - (id)anyObject
  - (void)makeObjectsPerformSelector:(SEL)aSelector
  - (id)member:(id)object (uses isEqual: and returns a matching object)
  ```

- NSMutableSet

  Modifiable version of NSSet

  ```
  - (void)addObject:(id)object
  - (void)removeObject:(id)object
  - (void)unionSet:(NSSet *)otherSet
  - (void)minusSet:(NSSet *)otherSet
  - (void)intersectSet:(NSSet *)otherSet
  ```

Enumeration

- NSArray of NSString objects

```
NSArray *myArray = ...; // known to only have NSString objects inside
for (NSString *string in myArray) {
    double value = [string doubleValue]; // crash if not NSString
}
```

- NSArray of id

```
NSArray *myArray = ...;  // no idea what kind of objects are inside
for (id obj in myArray) {
    < do something with obj here, but make sure you don't
      send it a message it doesn't respond to >
    if ([obj isKindOfClass:[NSString class]]) {
        // send NSString messages to obj with impunity!
    }
}
```

Enumeration

- NSDictionary's keys

```
NSDictionary *myDict = ...;
for (id key in [myDict allKeys]) {
  < do something with the key >
}
```

- NSDictionary's values

```
NSDictionary *myDict = ...;
for (id value in [myDict allValues]) {
  < do something with the value >
}
```

Property Lists

- The term "Property List" just means "one of the collection classes which contains only more collection classes (nothing else)."

  `NSArray`, `NSDictionary`, `NSNumber`, `NSString`, `NSDate`, `NSData`

- So an `NSArray` is a Property List as long as all the objects in it are also Property Lists.

- An `NSDictionary` is a Property List as long as all the keys and all the values are Property Lists.

- Why make this distinction? The SDK has a number of methods here and there which read/write Property Lists.

Other Foundation

- NSUserDefaults
  - `- (void)setDouble:(double)aDouble forKey:(NSString *)key`
  - `- (NSInteger)integerForKey:(NSString *)key`
  - `- (void)setObject:(id)obj forKey:(NSString *)key`

    (obj must be a Property List)
  - `- (NSArray *)arrayForKey:(NSString *)key`

    (if the object stored for that key is not an `NSArray`, this returns `nil`)
  - `- (void)synchronize` // writes to permanent storage

- NSNotification
- NSTimer
- NSThread
- NSFileManager
- Undo Manager

Protocols

- Very similar to `@interface`, but no implementation

```
@protocol Foo
- (void)doSomething;
@optional
- (int)getSomething;
@required
- (NSArray *)getManySomethings:(int)howMany;
@end
```

- Classes then proclaim they implement a protocol

```
@interface MyClass : NSObject <Foo>
    ...
@end
```

Protocols

- Declaring arguments to require a protocol

```
- (void)giveMeTheObject:(id <Foo>)anObjectImplementingFoo
```

- Declaring variables to require a protocol

```
id <Foo> obj = [[MyClass alloc] init];
[obj doSomething]; // will not warn (and should be okay)
```

- Compiler will warn of misbehavior

  Class says it implements protocol Foo, but doesn't implement required methods

  Assigning an object which does not implement Foo to a variable like obj above

  Passing an object which does not implement Foo through an argument which requires it (like above)

Delegate

- Very common in SDK to have a property which is a "delegate"
- Used to pass off responsibility to another object
- The property will be declared (approximately) like this ...

```
@property id <MyClassDelegate> delegate;
```

- Convenient for maintaining MVC boundaries but still have documented interfaces between things

  For example, the `UITableView` delegates both the provision of the data and the content of what is drawn to other objects while implementing the core of the user interaction itself

Creating Objects

- Allocating and initializing
- Send `+ (id)alloc` to the class
- Send appropriate initializer to what you get back
- `alloc` allocates space for the instance variables
- Default initializer (for `NSObject` and subclasses) is
  `- (id) init`
- `NSObject`'s `init` sets all instance variables to zero
- Subclasses of `NSObject` might define new initializers with more arguments
  Initializers with fewer args should call those with more args (usually)

```
CalculatorBrain *brain = [[CalculatorBrain alloc] init];

UIView *view = [[UIView alloc] initWithFrame:aRect];
UIView *view = [[UIView alloc] init]; // some default frame

MyClass *obj = [MyClass alloc];  // ack! no init! don't do this!
```

Creating Objects

- Goofy implementation of initializers

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

@implementation MyView

- (id)initWithFrame:(CGRect)aRect
{
    if (self = [super initWithFrame:aRect]) {
        // initialize my class here
    }
    return self;
}

@end
```

initWithFrame: is UIView's "designated initializer"
so we, as a subclass, must call it in our DI.
We don't have to override it, but we should
if it makes any sense whatsoever to our class.

Creating Objects

- Asking other objects to create an object for you

```
NSString *s = [otherString stringByAppendingString:@”hi”];
NSArray *keys = [dictionary allKeys];
NSString *lowerString = [string lowercaseString];
NSNumber *n = [NSNumber numberWithFloat:9.0];
NSDate *date = [NSDate date]; // returns the date/time now
```

Memory Management

- When does the memory get freed?
- Garbage Collection!
- NO, sorry.

- Reference Counting

Reference Counting

- Objects take ownership for other objects.
- Multiple owners is okay.
- Mechanism for taking "temporary" ownership.
- When last object gives up ownership, deallocate.

Object Ownership

- When you call `alloc`, you take ownership.
- When you ask another object to create an object for you, you are not taking ownership (with a couple of exceptions).
- But if you want to access that object outside the method you are in, you must take ownership.
- You take ownership by sending the object you want to own the message "`retain`."
- When you are done owning the object, send it "`release`."

  If you are the last owner, object will be freed.

  Messages sent to that object after that will crash your application.

autorelease

- What if you want to give an object to someone?
  Usually you are doing this by returning it from one of your methods.

- Before you return it to them, send the object the message "autorelease" first.

- UIKit will automatically send it release at some later time (but not until call stack unwinds).
  We'll talk more about how this works when we get to threads.

- All those NSString, NSArray, etc. methods are returning "autoreleased" objects.

autorelease

- Example

```
- (Money *)showMeTheMoney:(double)amount
{
    Money *theMoney = [[Money alloc] init:amount];
    [theMoney autorelease];
    return theMoney;
}
```

autorelease

- Example

```
- (Money *)showMeTheMoney:(double)amount
{
    Money *theMoney = [[Money alloc] init:amount];
    return [theMoney autorelease];
}
```

autorelease

- Mutable collection class `autorelease` creators

```
[NSMutableString string];
[NSMutableArray array];
[NSMutableDictionary dictionary];
```

- Create them, load them up, and return them

```
- (NSString *)showMeTheMoney:(double)amount
{
    NSMutableString *s = [NSMutableString string];
    [s appendString:@"The Money:"];
    [s appendFormat:@" %g", amount];
    return s;
}
```

Note there is no autorelease here!

autorelease

- Immutable "with" creators

```
[NSString stringWith... ];
[NSArray arrayWith... ];
[NSDictionary dictionaryWith...];
```

```
[NSString stringWithFormat:@"%@ %d", ...];
[NSArray arrayWithObjects:obj1, obj2, nil];
[NSDictionary dictionaryWithObjectsAndKeys:...];
[NSArray arrayWithContentsOfFile:(NSString *)path];
[NSDictionary dictionaryWithContentsOfFile:...];
[NSString stringWithContentsOfFile:encoding:error:];
```

Other Ownership Rules

- When you put an object in an `NSArray` or `NSDictionary`, they do take ownership. When you take an object out, they `release` ownership.

- You also implicitly `retain` if you copy an object
  This is done using the copy method.

- Methods whose names start with `alloc`, `copy` or `new` return an object you own
  So you must release it at some point down the road

- You should release an object as soon as possible
  That is to say, the instant you are done with it.

Deallocation

- What happens when the last owner releases?
- A special method, `- (void)dealloc`, is called.
- You should override this method and release any instance variables you own.
- And then be sure to call `[super dealloc]` to let your superclass release it's owned objects.
- NEVER call `dealloc`. It is called automatically when the last owner releases.

Properties

- Remember `@property`?
- Does the `@synthesized` getter method return `autoreleased` object?
- NO! That getter is returning an instance variable. If the caller doesn't `retain` it, then when your object is deallocated, that caller will have a bad pointer (assuming you properly `releases` your instance variables in your `dealloc`).

Properties

- Remember `@property`?
- Does the `@synthesized` setter do a `retain` when it is called?
- You get to decide:

```
@property (retain) NSString *name;

@synthesize will create a setter equivalent to this …

- (void)setName:(NSString *)aString
{
    [name release];
    name = [aString retain];
}
```

Properties

- Remember `@property`?
- Does the `@synthesized` setter do a `retain` when it is called?
- You get to decide:

```
@property (copy) NSString *name;
```

`@synthesize` will create a setter equivalent to this …

```
- (void)setName:(NSString *)aString
{
    [name release];
    name = [aString copy];
}
```

Properties

- Remember `@property`?
- Does the `@synthesized` setter do a `retain` when it is called?
- You get to decide:

```
@property (assign) NSString *name;
```

`@synthesize` will create a setter equivalent to this …

```
- (void)setName:(NSString *)aString
{
    name = aString;
}
```

Wake up!

- What about objects that come out of nib files?
- They are archived in IB, then unarchived in your running app.
- So where do you initialize if not in `init`?

```
- (void)awakeFromNib      // override this NSObject method
- (void)viewDidLoad       // only for UIViewController subclasses
```

viewDidUnload

- And what about `release`-ing `IBOutlets` in `UIViewControllers`?
- Don't do it in `dealloc` because your controller's views are allowed to be "unloaded" to save memory when not on-screen.
- Create `@properties (retain)` for all of them and then set them to `nil` in the method `viewDidUnload` …

```
@property (retain, nonatomic) IBOutlet UILabel *myOutlet;

- (void)viewDidUnload
{
    self.myOutlet = nil;    // this will release because property is retain
    [super viewDidUnload]; // probably not necessary unless you think it is
}
```

- When/if the view is reloaded, your outlets will get hooked back up and `- (void)viewDidLoad` will get called (again).

Demo

- MVC: Collector
- NSArray, NSDictionary
- Instrospection
- @property
- Memory Management

Homework

- Make your CalculatorBrain support variables
- Mostly NSArray, NSDictionary work
- Some Instrospection
- Get Memory Management right!
  Not too difficult, but a new concept. Be diligent!

Next Week

- Custom Views, Navigation Controllers