iPhone Programming

- History of Objective-C
- Objects
- Messaging
- Classes
- Foundation Class Essentials
Swift Programming Language

let myString = "Hello"

Objective-C Programming Language

[myString setStringValue:@"Hello"];
Objective-C
• Objective-C is an object oriented language.

• follows ANSI C style coding with methods from Smalltalk

• There is no formal written standard
  • Relies mostly on libraries written by others

• Flexible almost everything is done at runtime.
  • Dynamic Binding
  • Dynamic Typing
  • Dynamic Linking
Objective-C was invented by two men, Brad Cox and Tom Love.

Both were introduced to Smalltalk at ITT in 1981

Cox thought something like Smalltalk would be very useful to application developers

Cox modified a C compiler and by 1983 he had a working Object-oriented extension to C called OOPC.
• Tom Love acquired a commercial copy of Smalltalk-80 while working for Schlumberger Research

• With direct access Smalltalk, Love added more to OOPC making the final product, Objective-C.

• In 1986 they release Objective-C through their company “Stepstone”
• In 1988 Steve Jobs acquires Objective-C license for NeXT

• Used Objective-C to build the NeXTSTEP Operating System

• Objective-C made interface design for NeXTSTEP much easier

• NeXTSTEP was derived from BSD Unix

• In 1995 NeXT gets full rights to Objective-C from Stepstone
• Developed in 1993 by NeXT and Sun

• An effort to make NeXTSTEP-like Objective-C implementation available to other platforms.

• In order to be OS independent
  • Removed dependency on Mach Kernel
  • Made low-level data into classes

• Paved the way for Mac OS X, GNUstep
• NeXT is taken over by Apple in 1996 and put Steve Jobs and his Objective-C libraries to work

• Redesigned Mac OS to use objective-C similar to that of NeXTSTEP

• Developed a collection of libraries named “Cocoa” to aid GUI development

• Release Mac OS X (ten), which was radically different than OS 9, in March 2001
• Primarily the most frequently used frameworks nowadays.

• Developed by Apple from NeXTSTEP and OPENSTEP

• Has a set of predefined classes and types such as NNumber, NSString, NSDate, etc.

• **NS** stands for *NeXT-Sun* (or *NeXTSTEP*, depending upon whom you believe!)

• Includes a root class NSObject where words like alloc, retain, and release come from
• In October 2007, Apple Inc. released Objective-C 2.0 for Mac OS 10.5 (Leopard)

• Adds automatic garbage collection

• Instance Methods (public functions) are defined differently using `@property`
Almost everything is done at runtime

Uses dynamic typing, linking, and binding

This allows for greater flexibility

Minimizes RAM and CPU usage
Objects
Object

Thing
Message

```
Thing

"doSomething"

doSOMething
```
Other Objects As State

Thing
- flag
- count
- helper

behavior
- doSomething
- doSomethingElse

Other Thing
- performDifficultTask
Object Basics
OOP Vocabulary

- **Class**: defines the grouping of data and code, the “type” of an object
- **Instance**: a specific allocation of a class
- **Method**: a “function” that an object knows how to perform
- **Instance Variable (or “ivar”)**: a specific piece of data belonging to an object
OOP Vocabulary

• Encapsulation
  - keep implementation private and separate from interface

• Polymorphism
  - different objects, same interface

• Inheritance
  - hierarchical organization, share code, customize or extend behaviors
Inheritance

- Hierarchical relation between classes
- Subclass “inherit” behavior and data from superclass
- Subclasses can use, augment or replace superclass methods
OOP with ObjC
Classes and Instances

- In Objective-C, classes and instances are both objects
- Class is the blueprint to create instances
Classes and Objects

- Classes declare state and behavior
- State (data) is maintained using instance variables
- Behavior is implemented using methods
- Instance variables typically hidden
  - Accessible only using getter/setter methods
OOP From ObjC Perspective

- Everybody has their own spin on OOP
  - Apple is no different
- For the spin on OOP from an ObjC perspective:
  - Read the “Object-Oriented Programming with Objective-C” document
Messaging syntax
Class and Instance Methods

• Instances respond to instance methods
  - (id)init;
  - (float)height;
  - (void)walk;

• Classes respond to class methods
  + (id)alloc;
  + (id)person;
  + (Person *)sharedPerson;
Message syntax

[receiver message]

[receiver message:argument]

[receiver message:arg1 andArg:arg2]
Message examples

Person *voter; //assume this exists

[voter castBallot];

int theAge = [voter age];

[voter setAge:21];

if ([voter canLegallyVote]) {
    // do something voter-y
}

[voter registerForState:@"NY" party:@"Independant"];

NSString *name = [[voter spouse] name];
Terminology

- Message expression
  
  [receiver method: argument]

- Message
  
  [receiver method: argument]

- Selector
  
  [receiver method: argument]

- Method
  
  The code selected by a message
Dot Syntax

- Objective-C 2.0 introduced dot syntax
- Convenient shorthand for invoking accessor methods
  ```
  float height = [person height];
  float height = person.height;
  ```
  ```
  [person setHeight:newHeight];
  person.height = newHeight;
  ```
- Follows the dots...
  ```
  [[[person child] setHeight:newHeight]];
  // exactly the same as
  person.child.height = newHeight;
  ```
Objective-C Types
Dynamic and static typing

• Dynamically-typed object
  
  id anObject
  
  ▪ Just id
  ▪ Not id * (unless you really, really mean it...)

• Statically-typed object

  Person *anObject

• Objective-C provides compile-time, not runtime, type checking
• Objective-C always uses dynamic binding
The null object pointer

- Test for nil explicitly
  ```
  if (person == nil) return;
  ```
- Or implicitly
  ```
  if (!person) return;
  ```
- Can use in assignments and as arguments if expected
  ```
  person = nil;
  ```
  ```
  [button setTarget: nil];
  ```
- Sending a message to nil?
  ```
  person = nil;
  ```
  ```
  [person castBallot];
  ```
**BOOL typedef**

- When ObjC was developed, C had no boolean type (C99 introduced one)
- ObjC uses a typedef to define BOOL as a type
  ```
  BOOL flag = NO;
  ```
- Macros included for initialization and comparison: YES and NO
  ```
  if (flag == YES)  
  if (flag)        
  if (!flag)       
  if (flag != YES) 
  flag = YES;     
  flag = 1;
  ```
Selectors identify methods by name

- A selector has type SEL
  
  ```swift
  SEL action = [button action];
  [button setAction:@selector(start:)];
  ```

- Conceptually similar to function pointer

- Selectors include the name and all colons, for example:
  
  ```swift
  -(void)setName:(NSString *)name age:(int)age;
  ```

  would have a selector:

  ```swift
  SEL sel = @selector(setName:age:);
  ```
Working with selectors

- You can determine if an object responds to a given selector
  
  ```
  id obj;
  SEL sel = @selector(start:);
  if ([obj respondsToSelector:sel]) {
    [obj performSelector:sel withObject:self]
  }
  ```

- This sort of introspection and dynamic messaging underlies many Cocoa design patterns
  
  ```
  -(void)setTarget:(id)target;
  -(void)setAction:(SEL)action;
  ```
Working with Classes
Class Introspection

• You can ask an object about its class

    Class myClass = [myObject class];
    NSLog(@"My class is %@", [myObject class]);

• Testing for general class membership (subclasses included):

    if ([myObject isKindOfClass:[UIControl class]]) {
        // something
    }

• Testing for specific class membership (subclasses excluded):

    if ([myObject isMemberOfClass:[NSString class]]) {
        // something string specific
    }
Working with Objects
Identity versus Equality

- Identity—testing equality of the pointer values
  
  ```
  if (object1 == object2) {
      NSLog(@"Same exact object instance");
  }
  ```

- Equality—testing object attributes
  
  ```
  if ([object1 isEqual: object2]) {
      NSLog(@"Logically equivalent, but may be different object instances");
  }
  ```
-description

- NSObject implements -description
  - (NSString *)description;

- Objects represented in format strings using %@

- When an object appears in a format string, it is asked for its description
  
  `[NSString stringWithFormat: @"The answer is: %@", myObject];`

- You can log an object’s description with:

  `NSLog([anObject description]);`

- Your custom subclasses can override description to return more specific information
Cocoa Touch Architecture

Cocoa Touch

UIKit
- User interface elements
- Application runtime
- Event handling
- Hardware APIs

Foundation
- Utility classes
- Collection classes
- Object wrappers for system services
- Subset of Foundation in Cocoa
Foundation Classes
Foundation Framework

- Value and collection classes
- User defaults
- Archiving
- Notifications
- Undo manager
- Tasks, timers, threads
- File system, pipes, I/O, bundles
**NSObject**

- Root class
- Implements many basics
  - Memory management
  - Introspection
  - Object equality
NSString

• General-purpose Unicode string support
  - Unicode is a coding system which represents all of the world’s languages
• Consistently used throughout Cocoa Touch instead of “char *”
• Without doubt the most commonly used class
• Easy to support any language in the world with Cocoa
String Constants

• In C constant strings are
  "simple"
• In ObjC, constant strings are
  @"just as simple"
• Constant strings are NSString instances
  NSString *aString = @"Hello World!";
Format Strings

• Similar to printf, but with %@ added for objects

```plaintext
NSString *aString = @”Johnny”;
NSString *log = [NSString stringWithFormat: @”It’s ‘%@’”, aString];
```

log would be set to It’s Johnny

• Also used for logging

```plaintext
NSLog(@”I am a %@, I have %d items”, [array className], [array count]);
```

would log something like:

```plaintext
I am a NSArray, I have 5 items
```
**NSString**

- Often ask an existing string for a new string with modifications
  - `(NSString *)stringByAppendingString:(NSString *)string;`
  - `(NSString *)stringByAppendingFormat:(NSString *)string;`
  - `(NSString *)stringByDeletingPathComponent;`

- Example:
  ```
  NSString *myString = @“Hello”;
  NSString *fullString;
  fullString = [myString stringByAppendingString:@“ world!”];
  ```

  `fullString` would be set to **Hello world!**
**NSString**

- **Common NSString methods**
  - `(BOOL)isEqualToString:(NSString *)string;
  - `(BOOL)hasPrefix:(NSString *)string;
  - `(int)intValue;
  - `(double)doubleValue;

- **Example:**

```objective-c
NSString *myString = @"Hello";
NSString *otherString = @"449";
if ([myString hasPrefix:@"He"]) {
    // will make it here
}
if ([otherString intValue] > 500) {
    // won’t make it here
}
```
**NSMutableString**

- **NSMutableString** subclasses **NSString**
- Allows a string to be modified
- **Common NSMunableString methods**
  
  ```
  + (id)string;
  -(void)appendString:(NSString *)string;
  -(void)appendFormat:(NSString *)format, ...;
  ```

  ```
  NSString *newString = [NSMutableString string];
  [newString appendString:@"Hi"];
  [newString appendFormat:@"", my favorite number is: %d",
        [self favoriteNumberNumber]];
  ```
Collections

- **Array** - ordered collection of objects
- **Dictionary** - collection of key-value pairs
- **Set** - unordered collection of unique objects
- Common enumeration mechanism
- Immutable and mutable versions
  - Immutable collections can be shared without side effect
  - Prevents unexpected changes
  - Mutable objects typically carry a performance overhead
NSArray

- Common NSArray methods
  
  + arrayWithObjects:(id)firstObj, ...; // nil terminated!!!
  - (unsigned)count;
  - (id)objectAtIndex:(unsigned)index;
  - (unsigned)indexForObject:(id)object;

-NotFoundError returned for index if not found
  
  NSArray *array = [NSArray arrayWithObjects:@"Red", @"Blue", @"Green", nil];
  
  if ([array indexOfObject:@"Purple"] == NotFound) {
    NSLog (@"No color purple");
  }

- Be careful of the nil termination!!!
**NSMUTABLEARRAY**

- **NSMUTABLEARRAY** subclasses **NSArray**
- **So, everything in NSArray**
- **Common NSMUTABLEARRAY Methods**
  
```objective-c
+(.DeserializeObject *)array;
-(void)addObject:(id)object;
-(void)removeObject:(id)object;
-(void)removeAllObjects;
-(void)insertObject:(id)object atIndex:(unsigned)index;
```

```objective-c
NSArray *array = [[NSArray alloc] array];
[array addObject:@"Red"];
[array addObject:@"Green"];
[array addObject:@"Blue"];
[array removeObjectAtIndex:1];
```
NSDictionary

• Common NSDictionary methods
  + dictionaryWithObjectsAndKeys: (id)firstObject, ...;
  - (unsigned)count;
  - (id)objectForKey:(id)key;

• nil returned if no object found for given key

```objective-c
NSDictionary *colors = [[NSDictionary
    dictionaryWithObjectsAndKeys:@"Color 1", @"Red",
    @"Color 2", @"Green", @"Color 3", @"Blue", nil];

NSString *firstColor = [colors objectForKey:@"Color 1"];

if ([colors objectForKey:@"Color 8"]) {
    // won’t make it here
}
```
NSMmutableDictionary

- NSMmutableDictionary subclasses NSDdictionary
- Common NSMmutableDictionary methods
  
  + (NSMmutableDictionary *)dictionary;
  - (void)setObject:(id)object forKey:(id)key;
  - (void)removeObjectForKey:(id)key;
  - (void)removeAllObjects;

  NSMmutableDictionary *colors = [NSMmutableDictionary dictionary];

  [colors setObject:@"Orange" forKey:@"HighlightColor"];
NSSet

• Unordered collection of objects
• Common NSSet methods
  
  + setWithObjects:(id)firstObj, ...;   // nil terminated
  - (unsigned)count;
  - (BOOL)containsObject:(id)object;
**NSMutableSet**

- **NSMutableSet** subclasses **NSSet**
- Common **NSMutableSet** methods
  
  ```
  +(NSMutableSet *)set;
  -(void)addObject:(id)object;
  -(void)removeObject:(id)object;
  -(void)removeAllObjects;
  -(void)intersectSet:(NSSet *)otherSet;
  -(void)minusSet:(NSSet *)otherSet;
  ```
Enumeration

- Consistent way of enumerating over objects in collections
- Use with NSArray, NSDictionary, NSSet, etc.

```c
NSArray *array = ... ; // assume an array of People objects

// old school
Person *person;
int count = [array count];
for (i = 0; i < count; i++) {
    person = [array objectAtIndex:i];
    NSLog([person description]);
}

// new school
for (Person *person in array) {
    NSLog([person description]);
}
```
**NSNumber**

- In Objective-C, you typically use standard C number types
- `NSNumber` is used to wrap C number types as objects
- Subclass of `NSValue`
- No mutable equivalent!
- Common `NSNumber` methods
  
  ```
  + (NSNumber *)numberWithInt:(int)value;
  + (NSNumber *)numberWithDouble:(double)value;
  - (int)intValue;
  - (double)doubleValue;
  ```
Other Classes

- **NSData / NSMmutableData**
  - Arbitrary sets of bytes
- **NSDate / NSCalendarDate**
  - Times and dates
Getting some objects

- Until we talk about memory management:
  - Use class factory methods
    - NSString’s `+stringWithFormat`:
    - NSArray’s `+array`
    - NSDictionary’s `+dictionary`
  - Or any method that returns an object except alloc/init or copy.
• Oh, yes, and don’t forget our book for V22.0480-002:

Programming in Objective-C