

# CS193P - Lecture 2

## iPhone Application Development

Objective-C  
Foundation Framework

# Announcements

- Enrollment process is almost done
- Shooting for end of day Friday
- Please drop the class in Axess if you are not enrolled.

# Office Hours

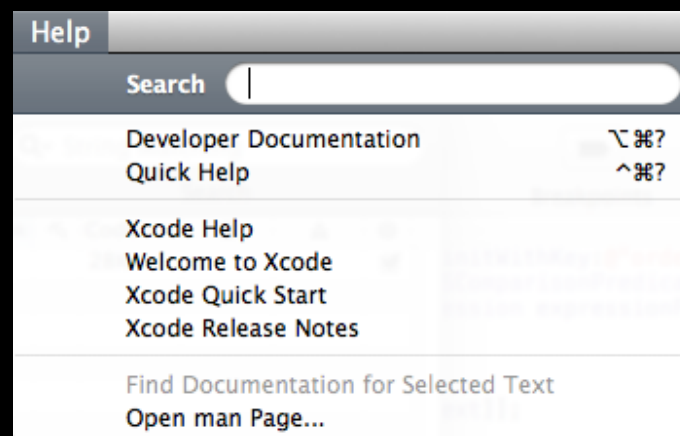
- David Jacobs
  - Mondays 4-6pm: Gates 360
- Paul Salzman
  - Some time. Best to try all possible times until you hit it
  - Some place, probably in Gates. Just come by and yell real loud

# iPhone SDK

- Enrolled students will be invited to developer program
  - Login to Program Portal
  - Request a Certificate
  - Download and install the SDK
- Will need your Device UDIDs - details to come
- Auditors will need to sign up for Developer Program independently
  - Free for Simulator development
  - \$99 for on-device development

# Getting Help

- The assignment walks you through it
- Key spots to look
  - API & Conceptual Docs in Xcode
  - Class header files
  - Docs, sample code, tech notes on Apple Developer Connection (ADC) site
    - <http://developer.apple.com>
    - Dev site uses Google search



# Today's Topics

- Questions from Tuesday or Assignments?
- Object Oriented Programming Overview
- Objective-C Language
- Common Foundation Classes

# Object Basics

# OOP Vocabulary

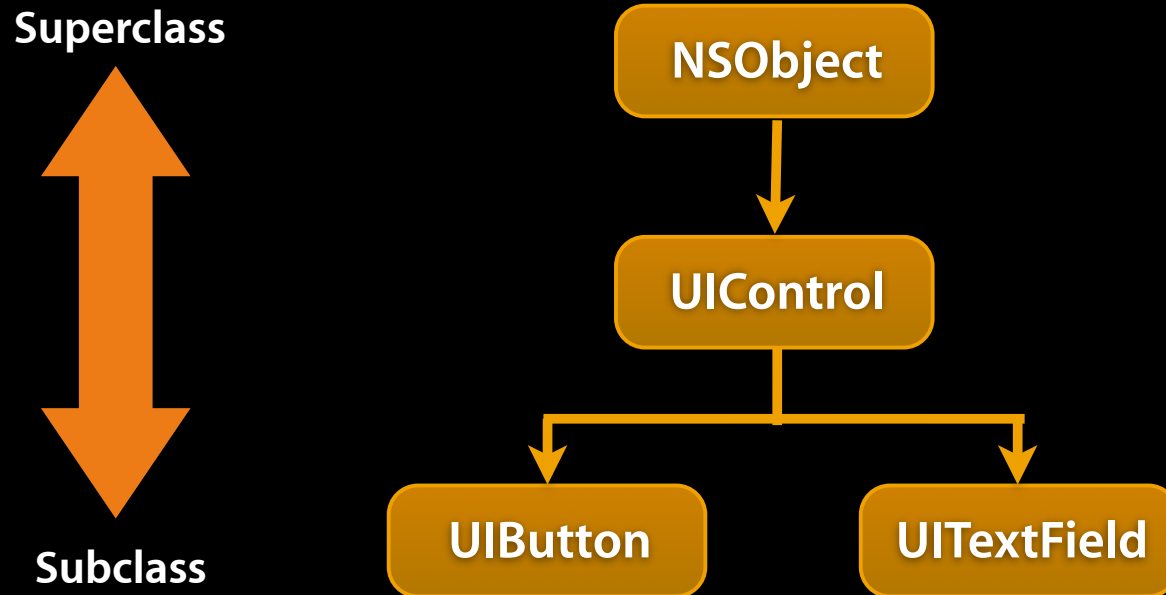
- **Class**: defines the grouping of data and code, the “type” of an object
- **Instance**: a specific allocation of a class
- **Method**: a “function” that an object knows how to perform
- **Instance Variable (or “ivar”)**: a specific piece of data belonging to an object



# OOP Vocabulary

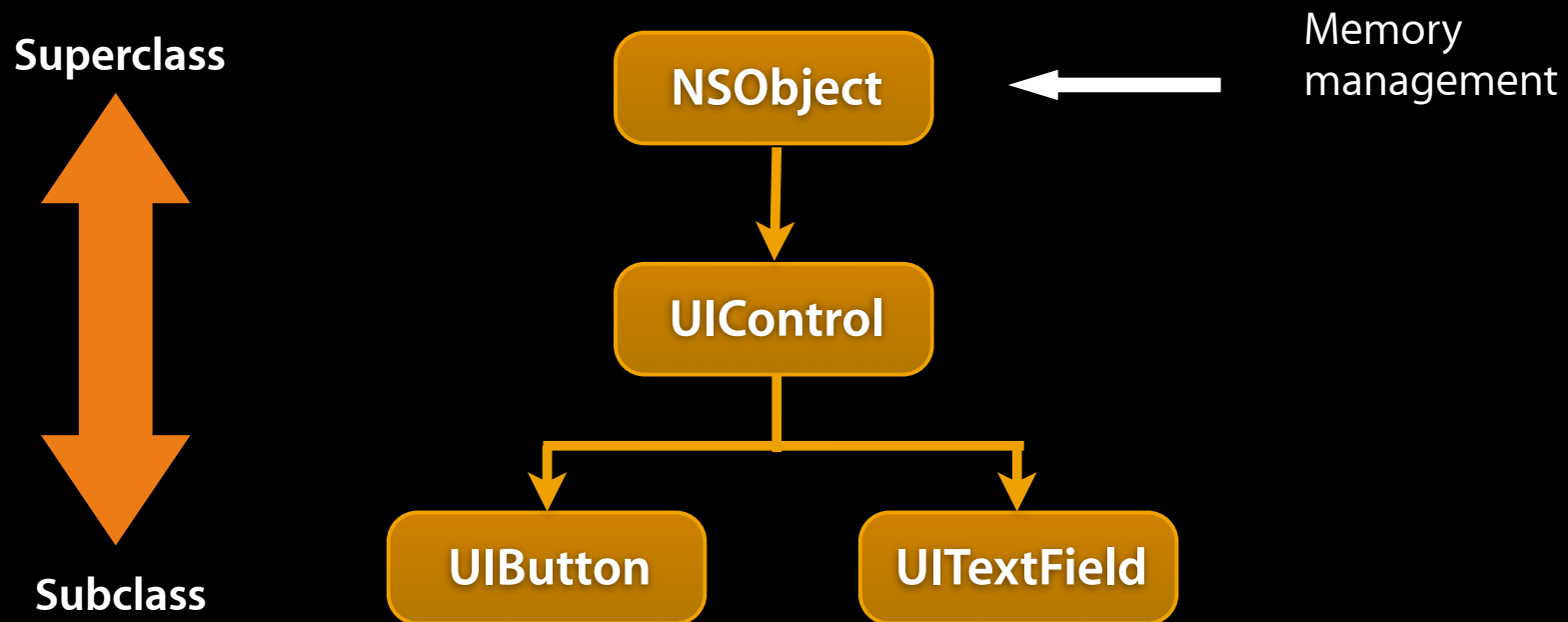
- Encapsulation
  - keep implementation private and separate from interface
- Polymorphism
  - different objects, same interface
- Inheritance
  - hierarchical organization, share code, customize or extend behaviors

# Inheritance



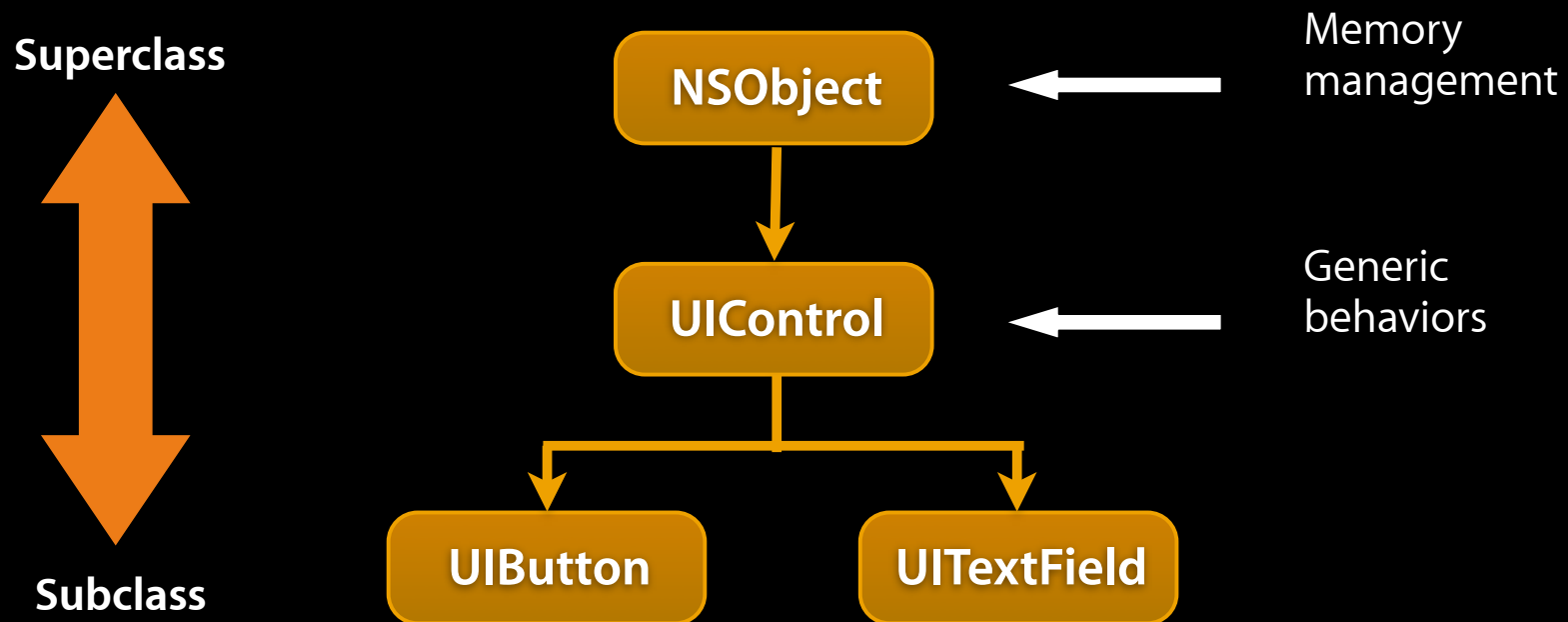
- Hierarchical relation between classes
- Subclass “inherit” behavior and data from superclass
- Subclasses can use, augment or replace superclass methods

# Inheritance



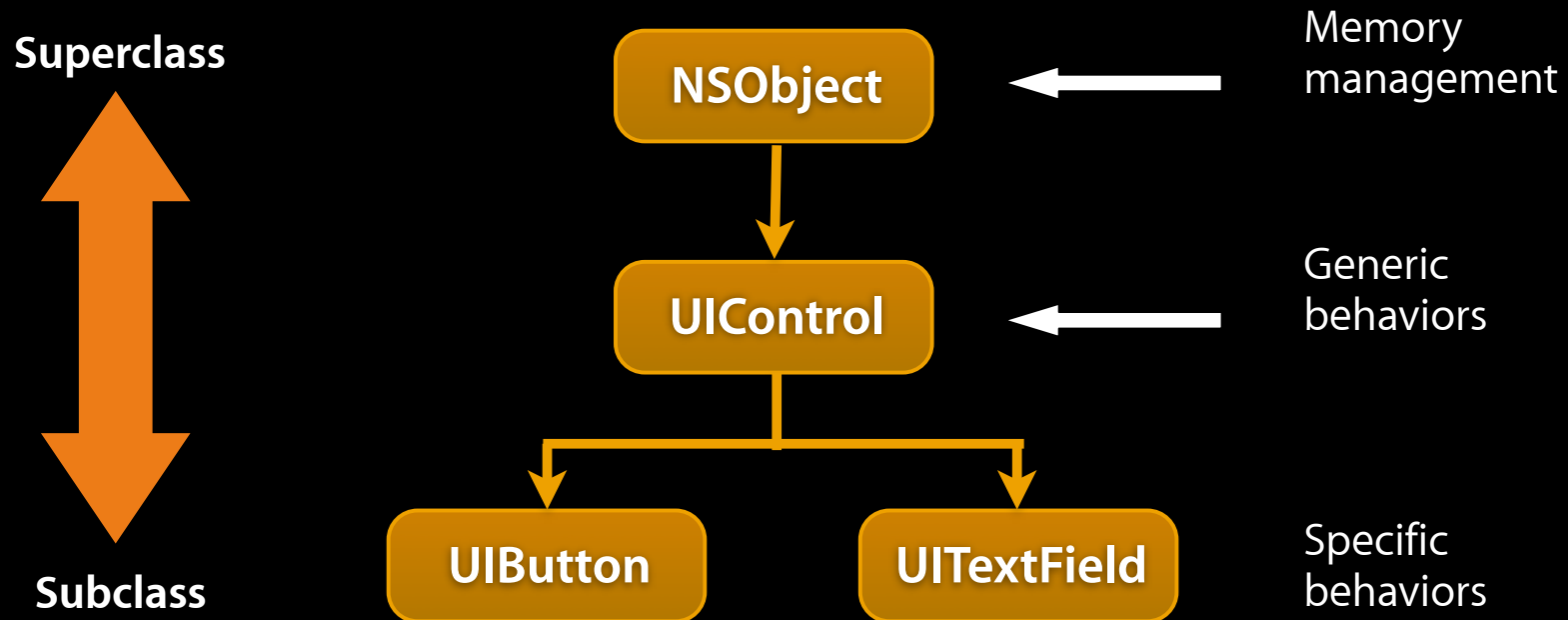
- Hierarchical relation between classes
- Subclass “inherit” behavior and data from superclass
- Subclasses can use, augment or replace superclass methods

# Inheritance



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

# Inheritance



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

# More OOP Info?

- Drop by office hours to talk about basics of OOP
- Tons of books and articles on OOP
- Most Java or C++ book have OOP introductions
- Objective-C 2.0 Programming Language
  - <http://developer.apple.com/documentation/Cocoa/Conceptual/ObjectiveC>

# Objective-C

# Objective-C

- Strict superset of C
  - Mix C with ObjC
  - Or even C++ with ObjC (usually referred to as ObjC++)
- A very simple language, but some new syntax
- Single inheritance, classes inherit from one and only one superclass
- Protocols define behavior that cross classes
- Dynamic runtime
- Loosely typed, if you'd like



# Syntax Additions

- Small number of additions
- Some new types
  - Anonymous object
  - Class
  - Selectors
- Syntax for defining classes
- Syntax for message expressions

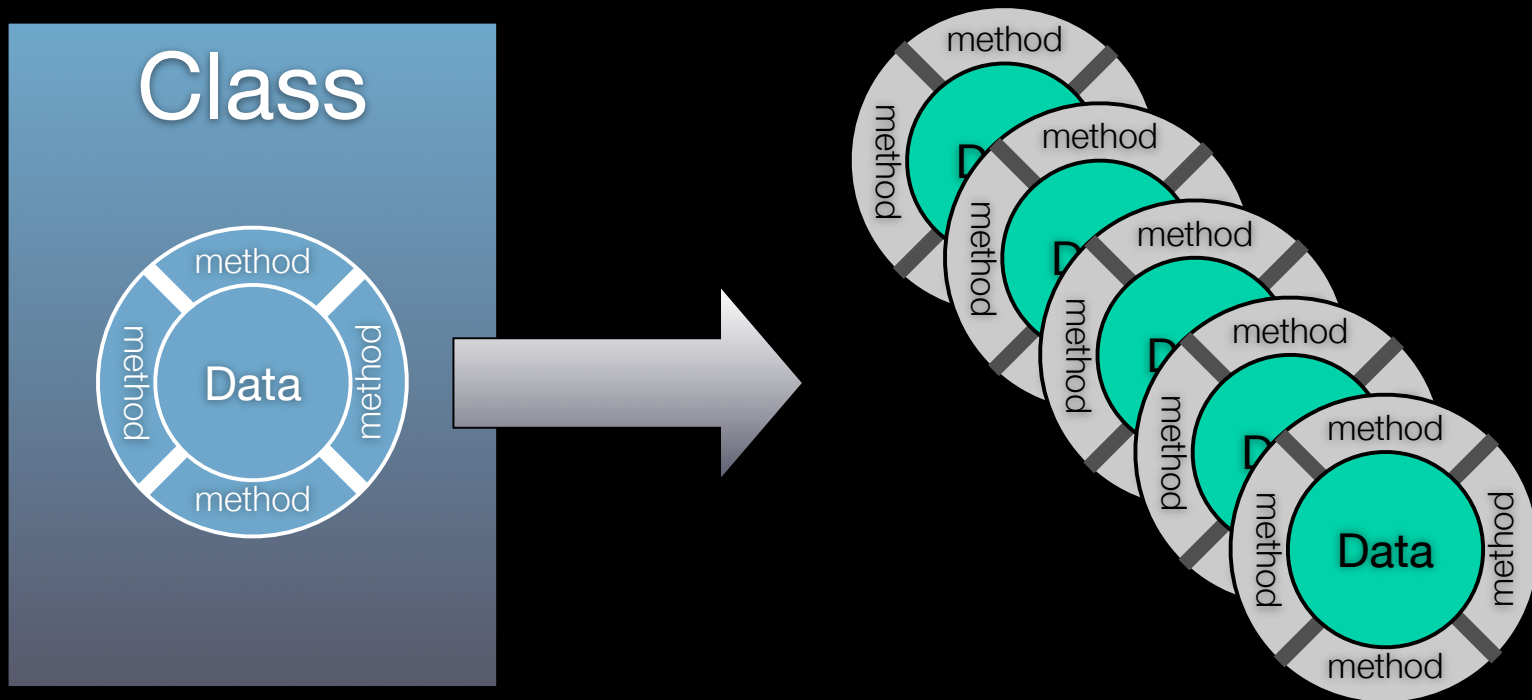
# Dynamic Runtime

- Object creation
  - All objects allocated out of the heap
  - No stack based objects
- Message dispatch
- Introspection

# 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
  - [http://developer.apple.com/iphone/library/documentation/Cocoa/Conceptual/OOP\\_ObjC](http://developer.apple.com/iphone/library/documentation/Cocoa/Conceptual/OOP_ObjC)

# 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:@"CA" party:@"Independant"];
```

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

# Method definition examples

```
Person *voter; //assume this exists
- (void)castBallot;
[voter castBallot];
- (int)age;
int theAge = [voter age];
- (void)setAge:(int)age;
[voter setAge:21];
- (BOOL)canLegallyVote;
if ([voter canLegallyVote]) {
    // do something voter-y
}
- (void)registerForState:(NSString*)state
                    party:(NSString*)party;
[voter registerForState:@"CA" party:@"Independant"];
- (Person*)spouse;
- (NSString*)name;
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

```
SEL action = [button action];  
[button setAction:@selector(start:)];
```

- Conceptually similar to function pointer

- Selectors include the name and all colons, for example:

```
-(void)setName:(NSString *)name age:(int)age;
```

would have a selector:

```
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 className]);
```

- Testing for general class membership (subclasses included):

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

- Testing for specific class membership (subclasses excluded):

```
if ([myObject isKindOfClass:[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



# 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

```
NSString *aString = @"Johnny";  
NSString *log = [NSString stringWithFormat: @"It's '%@'", aString];
```

log would be set to It's 'Johnny'

- Also used for logging

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

would log something like:

```
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:

```
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 NSMutableString methods

```
+ (id)string;
```

```
- (void)appendString:(NSString *)string;
```

```
- (void)appendFormat:(NSString *)format, ...;
```

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

```
[newString appendString:@"Hi"];
```

```
[newString appendFormat:@", my favorite number is: %d",  
    [self favoriteNumber]];
```

# 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)indexOfObject:(id)object;
```

- NSNotFound returned for index if not found

```
NSArray *array = [NSArray arrayWithObjects:@"Red", @"Blue",  
@"Green", nil];  
  
if ([array indexOfObject:@"Purple"] == NSNotFound) {  
    NSLog(@"No color purple");  
}
```

- Be careful of the nil termination!!!

# NSMutableArray

- NSMutableArray subclasses NSArray
- So, everything in NSArray
- Common NSMutableArray Methods

```
+ (NSMutableArray *)array;  
- (void)addObject:(id)object;  
- (void)removeObject:(id)object;  
- (void)removeAllObjects;  
- (void)insertObject:(id)object atIndex:(unsigned)index;
```

```
NSMutableArray *array = [NSMutableArray 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

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

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

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

# NSMutableDictionary

- NSMutableDictionary subclasses NSDictionary
- Common NSMutableDictionary methods

```
+ (NSMutableDictionary *)dictionary;  
- (void)setObject:(id)object forKey:(id)key;  
- (void)removeObjectForKey:(id)key;  
- (void)removeAllObjects;
```

```
NSMutableDictionary *colors = [NSMutableDictionary dictionary];
```

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

# NSSet

- Unordered collection of objects
- Common NSSet methods

```
+ initWithObjects:(id)firstObj, ...; // nil terminated  
- (unsigned)count;  
- (BOOL)containsObject:(id)object;
```

# NSMutableSet

- NSMutableSet subclasses NSMutableSet
- 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.

```
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 / NSMutableData**
  - Arbitrary sets of bytes
- **NSDate / NSDate**
  - 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`.

# More ObjC Info?

- <http://developer.apple.com/documentation/Cocoa/Conceptual/ObjectiveC>
- Concepts in Objective C are applicable to any other OOP language

# Questions?