SPARC iOS Training

Lesson 2: Intro to Objective-C and Cocoa

Goals

Agenda

- Basics of Objective-C and Cocoa
- Basic Syntax and Conventions
- Coding Demo

Obj-C vs Cocoa

Objective-C is the language.

Cocoa is the framework.

What is Objective-C?

- Developed in the early 1980s and licensed by NeXT for the NeXTSTEP Operating System.
- Superset of C
- It is the main programming language used by Apple for the OS X and iOS operating systems and their respective APIs, Cocoa and Cocoa Touch.

Common Items

- .h and .m files
- Pointers (id)
- *
- @
- Lots of brackets

What is a Pointer

- A pointer is a reference to an instance of an object in memory
- It “points” to something

Obj-C Data Types

- String = NSString
- Number = NSNumber
- Array = NSArray
- Dictionary = NSDictionary
- Primitives = BOOL, int, float, double

Mutable vs Immutable

- You cannot change the encapsulated values of immutable objects; once such an object is created, the value it represents remains the same throughout the object's life (NSArray, etc.)
- You can change the encapsulated value of a mutable object at any time (NSMutableArray, etc.)

Conventions

Class Names

- Cocoa encourages expressive, clear, non-ambiguious names.
- Class names are always capitalized.

```
UIButton
UITableView
UIColor
```

Conventions

Variable Names

- Variable names start with lower-case letters, but are internally capitalized wherever a new word appears
- Non-ambiguous

```
NSString *hostName;
NSNumber *latency;
NSArray *users;
MyCustomDataObject *dataFromServer
```

Creating Objects

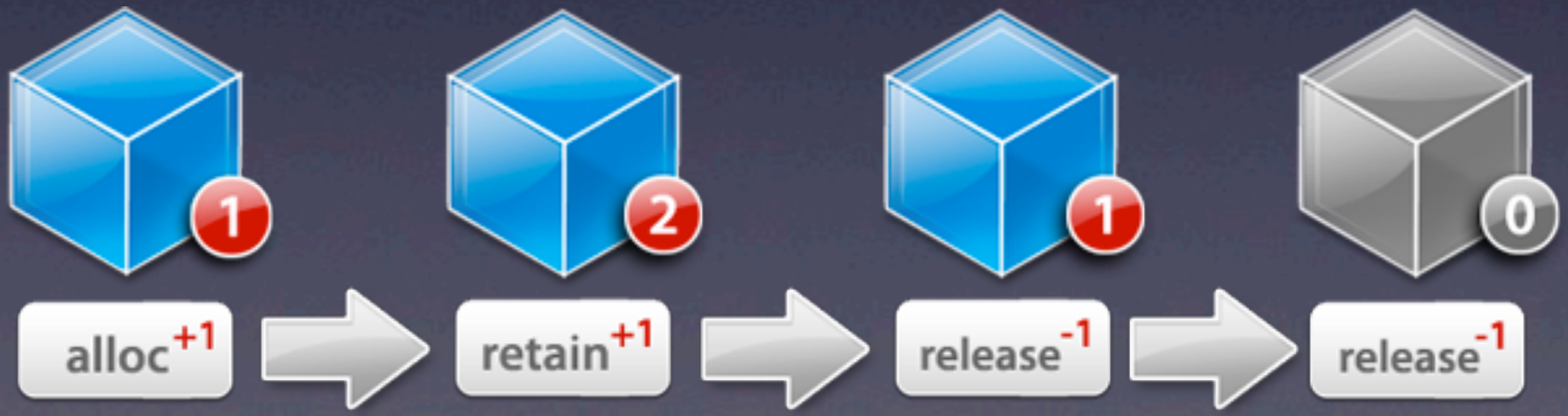
```
NSString *greeting = @"Hello SPARC!";
```

```
NSNumber *answerToLife = [NSNumber numberWithInt:42];
```

```
UIView *view = [[UIView alloc] init];
```

Memory Management

- Retain counts
- Anytime you alloc, copy or retain, you are adding 1 to the retain count
- For every +1, you need to -1 by calling release



ARC
Automatic Reference Counting

- Handles the releases for you
- ARC is available in iOS 5.0+
- Can have ARC and Non-ARC within the same app but you have to manage the non-ARC portions.

Calling Methods

```
[object method];
```

```
[object methodWithInput: input];
```

Conventions

Method Names

- Objective-C and Cocoa are designed to read well. Reading a message as a phrase is a good way to test your method name.

```
fileWrapper.write(path, true, true);
```

```
[fileWrapper writeToFile: path atomically: YES updateFilenames: YES];
```

Method Declarations

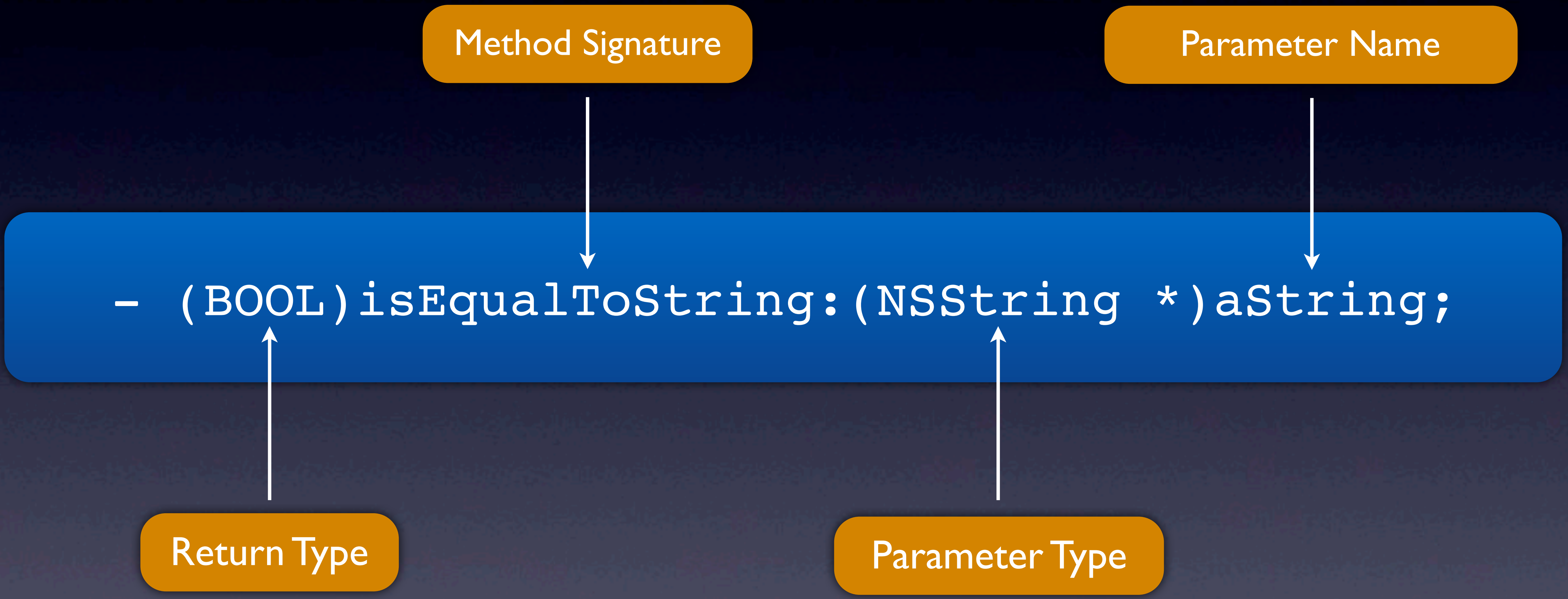
```
- (void)display;
```

```
- (NSString *)uppercaseString;
```

```
- (BOOL)isEqualToString:(NSString *)aString;
```

Method Declaration

Breakdown



```
- (BOOL)isEqualToString:(NSString *)aString;
```

Using our method

```
NSString *user = @”LeeLoo Dallas”;
```

```
NSString *greeting = [user
stringByAppendingString:@” Multipass”];
```

```
greeting is equal to:
“LeeLoo Dallas Multipass”
```

Nesting Methods

```
[[cardCounter addNumber:1] subtractNumber:2];
```

2 Types of Methods

Class Method

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

Instance Method

```
- (id)init;
```

Instance Methods

Declaration

```
- (id)initWithString:(NSString *)aString
```

In Use

```
NSString *myString = [[NSString alloc]
    initWithString:@"SPARC!"];
```

Class Methods

Declaration

```
+ (id)stringWithString:(NSString *)aString
```

In Use

```
NSString *myString = [NSString stringWithString:@”SPARC!”];
```

Literal Syntax

Strings

We use:

```
NSString *myString = @”SPARC!”;
```

Translates to:

```
NSString *myString = [NSString stringWithCString:"SPARC!"
        encoding:NSUTF8StringEncoding];
```

Literal Syntax

Integers

We use:

```
NSNumber *myNumber = @42;
```

Translates to:

```
NSNumber *myNumber = [NSNumber numberWithInt:42];
```

Literal Syntax

Doubles

We use:

```
NSNumber *myNumber = @3.1415926;
```

Translates to:

```
NSNumber *myNumber = [NSNumber numberWithDouble:3.1415926];
```

Literal Syntax

Floats

We use:

```
NSNumber *myNumber = @2.718f;
```

Translates to:

```
NSNumber *myNumber = [NSNumber numberWithFloat:2.718f];
```

Syntax

Arrays

We use:

```
NSArray *array1 = [NSArray arrayWithObjects:@"foo",
@42, @3.14, nil];
```

Translates to:

```
[NSArray arrayWithObjects:@"foo",
[NSNumber numberWithInt:42],
[NSNumber numberWithDouble:3.14], nil];
```

Literal Syntax

Arrays

We use:

```
NSArray *array1 = [NSArray arrayWithObjects:@"foo",
@42, @3.14, nil];
```

New Hotness:

```
NSArray *array1 = @[@"foo", @42, @3.14];
```

Properties

- a simple way to declare and implement an object’s getter and setter methods.
- The property declaration provides a clear, explicit specification of how the accessor methods behave.
- The compiler can synthesize accessor methods for you, according to the specification you provide in the declaration.
- Properties are represented syntactically as identifiers and are scoped, so the compiler can detect use of undeclared properties.

Properties

```
@property(nonatomic, weak) UIButton *button
```

```
@property(nonatomic, weak) IBOutlet UILabel *nameLabel
```

```
@property(nonatomic, strong) NSArray *users
```

Properties

Setter

```
@property(nonatomic, weak) IBOutlet UILabel *nameLabel
```

```
nameLabel.text = @”Label Text”;
```

```
[nameLabel setText:@”Label Text”]
```

Properties

Getter

```
NSString *labelString = nameLabel.text;
```

```
-(NSString *) text {
  return text;
}
```

Categories

- Add functionality to existing classes without subclassing

```
-(BOOL)isBlank;
```

```
-(BOOL)isBlank {
  if([[self stringByStrippingWhitespace] isEqualToString:@""])
    return YES;
  return NO;
}
```

Demo