Pace University - PCBT

Advanced iOS Class Spring 2015

OBJECTIVE-C INTRODUCTION

- This lecture will provide a background and context for using the OBJECTIVE-C language in iOS.
- We will discuss the history, purpose, and features.
- This lecture will also summarize the key language grammar and technical aspects.
- The lecture should be accompanied by an XCode exercise integrating Swift and Objective-C.

Origins of C/C++/Objective-C

- The initial development of C occurred at AT&T Bell Labs between 1969 and 1973;[3] according to Ritchie, the most creative period occurred in 1972. It was named "C" because its features were derived from an earlier language called "B", which according to Ken Thompson was a stripped-down version of the BCPL programming language.

- The origin of C is closely tied to the development of the Unix operating system, originally implemented in assembly language on a PDP-7 by Ritchie and Thompson, incorporating several ideas from colleagues. Eventually they decided to port the operating system to a PDP-11. B's inability to take advantage of some of the PDP-11's features, notably byte addressability, led to the development of an early version of C.

Origins of C/C++/Objective-C

- Bjarne Stroustrup began work on "C with Classes" in 1979.[3] The idea of creating a new language originated from Stroustrup's experience in programming for his Ph.D. thesis. Stroustrup found that Simula had features that were very helpful for large software development, but the language was too slow for practical use, while BCPL was fast but too low-level to be suitable for large software development. When Stroustrup started working in AT&T Bell Labs, he had the problem of analyzing the UNIX kernel with respect to distributed computing. Remembering his Ph.D. experience, Stroustrup set out to enhance the C language with Simula-like features. C was chosen because it was general-purpose, fast, portable and widely used. Besides C and Simula, some other languages that inspired him were ALGOL 68, Ada, CLU and ML. At first, the class, derived class, strong type checking, inlining, and default argument features were added to C via Stroustrup's C++ to C compiler, Cfront. The first commercial implementation of C++ was released on October 14, 1985.[3]

Origins of Objective-C

- Objective-C was created primarily by Brad Cox and Tom Love in the early 1980s at their company Stepstone. Both had been introduced to Smalltalk while at ITT Corporation's Programming Technology Center in 1981. The earliest work on Objective C traces back to around that time.

- After Steve Jobs left Apple Inc., he started the company NeXT. In 1988, NeXT licensed Objective-C from StepStone (the owner of the Objective-C trademark) and released its own Objective-C compiler and libraries on which the NeXTstep user interface and interface builder were based. While the NeXT workstations failed to make a great impact in the marketplace, the tools were widely lauded in the industry. This led NeXT to drop hardware production and focus on software tools, selling NeXTstep (and OpenStep) as a platform for custom programming.

Origins of Objective-C

- After acquiring NeXT in 1996, Apple Computer used OpenStep in its new operating system, Mac OS X. This included Objective-C and NeXT's Objective-C based developer tool, Project Builder (later replaced by Xcode), as well as its interface design tool, Interface Builder. Most of Apple's present-day Cocoa API is based on OpenStep interface objects, and is the most significant Objective-C environment being used for active development.

OODesign in Objective-C

- Like an object oriented language Objective C revolves around objects. It has three parts:
- **Interface:**
  - Interface of a class is generally defined in header file suffixed .h. It is a declaration of a class.
- **Implementation:**
  - Actual code is written in implementation of a class is generally defined in file of suffixed .m. It is a definition of a class.
- **Instantiation:**
  - After declaring and defining class we can be instantiated by allocating memory to the new object of the class.

Classes and Methods

- Because of objective-C is the extension of ANSI-C and it follows an
  object oriented approach so provides classes and objects. The way to declare and define classes and creation of object is little bit different from C and C++.

- To declare a new class objective-C uses **@interface directive.**

Interface Example

```
#import"SuperClass.h"

#import<headerFile.h>

@interface ClassName:SuperClass

{

    variable declaration;

    variable declaration;

}

method declaration;

method declaration;

@end
```

Calling Methods

- The basic syntax for calling a method on an object is this:

```
[object method];

[object methodWithInput:input];

// Methods can return a value:

output = [object methodWithOutput];

output = [object
methodWithInputAndOutput:input];
```

Calling Methods

- You can call methods on classes too, which is how you create objects. In the example below, we call the string method on the NSString class, which returns a new NSString object:
- id myObject = [NSString existingstring];
- The id type means that the myObject variable can refer to any kind of object, so the actual class and the methods it implements aren't known when you compile the app.
- In this example, it's obvious the object type will be an NSString, so we can change the type:
- NSString* myString = [NSString string];
- This is now an NSString variable, so the compiler will warn us if we try to use a method on this object which NSString doesn't support.
- Notice that there's a asterisk to the right of the object type. All Objective-C object variables are pointers types. The id type is predefined as a pointer type, so there's no need to add the asterisk.

Multiple-Input Methods

- Some methods take multiple input values. In Objective-C, a method name can be split up into several segments. In the header, a multi-input method looks like this:
- -(BOOL)writeToFile:(NSString *)path atomically: (BOOL)useAuxiliaryFile;
- You call the method like this:
- BOOL result = [myData writeToFile:@"/tmp/log.txt" atomically:NO];
- These are not just named arguments. The method name is actually writeToFile:atomically: in the runtime system.

Accessors

- All instance variables are private in Objective-C by default, so you should use accessors to get and set values in most cases. There are two syntaxes. This is the traditional 1.x syntax:
- [photo setCaption:@"Day at the Beach"];
- output = [photo caption];
- The code on the second line is not reading the instance variable directly. It's actually calling a method named caption. In most cases, you don't add the "get" prefix to getters in Objective-C.

Accessors

- Whenever you see code inside square brackets, you are sending a message to an object or a class.
- Dot Syntax
- The dot syntax for getters and setters is new in Objective-C 2.0, which is part of Mac OS X 10.5:
- photo.caption = @"Day at the Beach"; output = photo.caption;
- You can use either style, but choose only one for each project. The dot syntax should only be used setters and getters, not for general purpose methods.

Inheritance

- Objective-C enables programmer to inherit common methods and properties from other class, known as inheritance. Class from methods and properties are inherited known as Base Class and class that inherits known as Derived Class. Derived class only specifies how it is different with base class and everything else is taken to be the same. Here in the figure given below Vehicle is the base class and both Car and Bike are derived classes so that these classes can use methods and properties of Vehicle class.
- @interface FirstClass:NSObject {
- @interface SecondClass: FirstClass {

Data Types and Expressions

- http://www.techotopia.com/index.php/Objective-C_Operators_and_Expressions
- http://www.techotopia.com/index.php/Objective-C_2.0_Data_Types

Creating Objects

- There are two main ways to create an object. The first is: NSString* myString = [NSString string];
- This is the more convenient automatic style. In this case, you are creating an autoreleased object, which we'll look at in more detail later. In many cases, though, you need to create an object using the manual style:
- NSString* myString = [[NSString alloc] init];
- This is a nested method call. The first is the alloc method called on NSString itself. This is a relatively low-level call which reserves memory and instantiates an object.
- The second piece is a call to init on the new object. The init implementation usually does basic setup, such as creating instance variables. The details of that are unknown to you as a client of the class.
- In some cases, you may use a different version of init which takes input: NSNumber* value = [[NSNumber alloc] initWithFloat:1.0];

Loops

- Using loops to repeat actions in programming is a common task. Objective-C does this chore in more or less the same way as other programming languages.

For and Do Loop Example

```
//For loop
for (int y = 0; y < 3; y++) {
    NSLog(@"y = %i", y);
}
do{
    NSLog(@"x = %i", x);
    x++;
}
```

While Loop Example

```
x = 0;
while (x <= 4 ) {
    NSLog(@"x = %i", x);
    x++;
}
```

Conditionals/Decisions

- A fundamental feature of any programming language lies in its capability to make decisions. Decisions were made when executing the looping statements to determine when to terminate a loop. The Objective-C programming language also provides several other decision- making constructs, which are covered in this chapter:
- The if statement
- The switch statement

If Statement

- The Objective-C programming language provides a general decision-making capability in the form of a language construct known as the if statement. The general format of this statement is
- if (expression) program statement

Switch Statement

- The type of if-else statement chain you encountered in the last program example —where the value of a variable is successively compared against different values—is so commonly used when developing programs that a special program statement exists in the Objective-C language for performing precisely this function. The name of the statement is the switch statement, and its general format is as follows:

Switch Example

```
switch ( expression ) {
    case value1:
        program statement break;
    case value2:
        program statement break;
    default:
        program statement break;
}
```

Access Modifiers

- Access Privileges:
- Access Modifiers
- 1. Default access in objective-C is @protected.
- 2. Like C++ objective-C provide public and private access modifiers as well. 3. @protected accessifier enable access elements in the subclass.

Exception Handling

- Objective-C provides exception handling to handle exceptional conditions so that code can be easier to write, easy to detect exceptions in the code and easier to maintain as well. To take support of exception handling makes sure the -fobj- exceptions flag is turned on.
- These are four compiler directives that are used for exception handling- 1. @try: block of code that can throw an exception.
- 2. @catch: define block of code to handle exception thrown by try block, this is usually an NSException object.
- 3. @finally: defines a block of code that executed whether an exception is thrown or not.
- 4. @throw: once your program detects an exception, it must propagate the exception to code that handles it. This code is called the exception handler. This entire process of propagating an exception is referred to as "throwing an exception".

End of Lecture

rpascazio@pace.edu