

Cocoa Exercises

About These Exercises

In order to maximize the instructional potential of these exercises, you should know C, but not necessarily object-oriented programming or application design. These exercises introduce two very common classes: NSString and NSNumber.

Exercise 1: Create an iPhone Window-based App

In this exercise, you will create an iPhone Window-based App named CocoaDemo. CocoaDemo will serve as the foundation for the exercises in this book. At the end of all the exercises, you will submit the completed CocoaDemo Project.

Step 1. Startup Xcode:

1. Click on Finder (the "Smiling Face" icon at the lower left of your desktop).
2. Select Macintosh HD near the top-left of the Finder window.
3. Select the Developer folder on the right-side of the Finder window.
4. Select the Applications folder.
5. Double-click on Xcode.

Stated differently, the path to Xcode is this: /Volumes/Macintosh HD/Developer/Applications/Xcode.app.

Step 2. Use the Xcode main menu to select File > New Project.

You will see the Xcode New Project window appear:

Step 3. On the left-side of the New Project window, under the iPhone OS heading, make sure that Application is selected.

On the right-side of the New Project window, make sure that Window-based Application is selected.

Step 4. Click on the Choose button.

Step 5. In the Save As field, enter CocoaDemo for the project name.

Step 6. Click on the Save button. Xcode will display the project files for the CocoaDemo project.

Cocoa Exercises

Step 7. Near the top-middle of Xcode is a green button labeled Build and Run. Click on that button to build and run the default iPhone Window-based app.

You should see the default iPhone Window-base app running in the iPhone Simulator.

Step 8. Press the HOME key on the iPhone Simulator to stop the default application from running.

Exercise 2: Create instances of NSString

In this exercise, you will add your first code to the existing Objective-C 2.0 code that was generated by the iPhone Window-based application template.

Step 1. With Xcode still open in the CocoaDemo project from the previous exercise, observe the folder titled Other Sources.

Step 2. Click on the small triangle to the left of the Other Sources folder. You should see the list of files displayed.

Step 3. Observe the file named main.m. A file with a .m file extension is an Objective-C source-code file.

Step 4. Click on the file named main.m, displaying it in the Xcode editor pane. You will see code that looks like this:

```
// main.m
// CocoaDemos

#import <UIKit/UIKit.h>

int main(int argc, char *argv[]) {

    NSAutoreleasePool * pool = [[NSAutoreleasePool alloc] init];
    int retVal = UIApplicationMain(argc, argv, nil, nil);
    [pool release];
    return retVal;
}
```

This Objective-C code is common for all iPhone applications. While it is code that you will rarely, if ever change, it is important to understand each line of code, for the concepts used in this file are similar to concepts used in other iPhone Objective-C source files.

Step 5. Use Xcode to edit main.m to match the code shown below.

```
// main.m
// CocoaDemos

#import <UIKit/UIKit.h>

int main(int argc, char *argv[]) {
```

Cocoa Exercises

```
NSAutoreleasePool * pool = [[NSAutoreleasePool alloc] init];

    // create two new NSStrings
    NSString *string1 = @"Hello World!";
    NSString *string2 = @"Hello Cocoa!";

    // display the original strings
    NSLog( @"string1 = %@", string1 );
    NSLog( @"string2 = %@", string2 );

    int retVal = UIApplicationMain(argc, argv, nil, nil);
    [pool release];
    return retVal;
}
```

Step 6. After entering the code shown above, use the Xcode main menu to select Run > Console. You should see the Xcode console appear.

Step 7. In the Xcode console, click on the Clear Log button to clear the output on the console.

Step 8. In the Xcode console, click on the green Build and Run button to build and run the code, observing the output on the console:

```
2010-08-26 16:03:10.577 CocoaDemo[99684:207] string1 = 'Hello World!'
2010-08-26 16:03:10.578 CocoaDemo[99684:207] string2 = 'Hello Cocoa!'
```

Exercise 3: Comparing NSStrings

In this exercise, you will use NSString’s `-compare:` instance method to determine whether the second string is less than, equal to, or greater than the first string. In this context, we are referring to the [lexicographical order](#) of the strings (dictionary order, or alphabetical order).

From Xcode, select `Help > Developer Documentation`.



In the Search field, type in `NSString` and press `Return`. On the left under `API`, double click on the first instance of `NSString` to select the `NSString Class Reference`. Scroll down to find the section titled “Identifying and Comparing Strings” listed under `Tasks`. Here you will find a variety of methods provided to you for comparing strings. Click on “`- compare:`” to jump to the description of this particular method. You will see that the syntax for calling this method is:

```
– (NSComparisonResult)compare:(NSString *)aString
```

This syntax should be familiar from the Objective-C Exercises you recently completed. We can see that this is an instance method that takes a single argument, an

Cocoa Exercises

NSString. It returns something called NSComparisonResult. What exactly is an NSComparisonResult?

Use the Search field to type in and search for NSComparisonResult, and you will find the following enumeration:

```
enum {
    NSOrderedAscending = -1, // assigned a value of -1
    NSOrderedSame,         // defaults to -1 + 1 = 0
    NSOrderedDescending   // defaults to 0 + 1 = 1
};
typedef NSInteger NSComparisonResult;
```

These constants are used to indicate how items in a request are ordered, from the first one given in a method invocation to the last (that is, left to right in code). The possible values that NSComparisonResult can hold are:

```
-1    The first string and the second string are in ascending order.
0     The first string and the second string are equal or same order.
1     The first string and the second string are in descending order.
```

By testing the return value, we can easily determine the lexicographical order of any two strings.

Step 1. Assuming that you are continuing from the previous exercise, add the switch statement as shown below.

```
...

    // compare the strings and display the result
switch ( [string1 compare:string2] ) {
    case -1:
        NSLog( @"'%@' comes before '%@'.", string1, string2 );
        break;
    case 0:
        NSLog( @"'%@' is equal to '%@'.", string1, string2 );
        break;
    case 1:
        NSLog( @"'%@' comes after '%@'.", string1, string2 );
        break;
}
```

Step 2. After entering the code, use the Xcode main menu to select Run > Console, then click on the green Build and Run, observing the output in the console.

Cocoa Exercises

```
... 'Hello World!' comes after 'Hello Cocoa!'.
```

This is the expected output, as lexicographically ‘W’ (in “World”) comes after ‘C’ (in “Cocoa”).

Try altering the constants assigned to `string1` and `string2` to test this method and the `switch` statement. Make sure you understand how this all ties together.

Exercise 4: lowercaseString and uppercaseString

In this exercise, you will use the NSString instance methods `-lowercaseString:` and `-uppercaseString:` to convert a string to all upper- or all lower-case letters.

Returning to the NSString Class Reference in the Developer Documentation, locate the description of the instance method `-lowercaseString:`

```
- (NSString *)lowercaseString
```

From this prototype, we can tell that this is an instance method that does not accept any arguments, and it returns an NSString with each character from the receiver changed to its corresponding lowercase value.

Locate the `-uppercaseString:` method in the NSString Class Reference and note that it shares a similar syntax, but returns an NSString with each character from the receiver changed to its corresponding *uppercase* value.

Step 1. Assuming that you are continuing from the previous exercise, add the following two statements:

```
...
// convert our strings to upper- and lower-case
NSLog( @"string1 in upper-case: '%@'", [string1 uppercaseString] );
NSLog( @"string2 in lower-case: '%@'", [string2 lowercaseString] );
```

Step 2. After entering the code, use the Xcode main menu to select `Run > Console`, then click on the green `Build and Run`, observing the output in the console.

```
...
... string1 in upper-case: 'HELLO WORLD!'
... string2 in lower-case: 'hello cocoa!'
```

Exercise 5: NSNumber

In this exercise, you will learn about another common Foundation class, NSNumber. Open the NSNumber Class Reference by double-clicking on NSNumber under API in the left pane.

As you read through the Overview, you'll see that NSNumber is a convenience class for handling C numeric types (e.g. short, int, long). [So why would you want to use NSNumber instead of a C primitive scalar?](#) In a nutshell, if you wish to pass numeric values to other Foundation classes, you will need to use an NSNumber.

Enter NSNumber in the Developer Documentation Search field and press Return. Scroll down until you locate the section "Retrieving String Representations". Of particular interest to us for this exercise is the instance method, -stringValue:

```
- (NSString *)stringValue
```

As you can see from this prototype, this is an instance method that accepts no arguments and returns an NSString. Its function is to return the receiver's value as a human-readable string.

Step 1. Assuming that you are continuing from the previous exercise, edit main.m to add the following:

```
...
    // we need a number, so let's use pi
    NSNumber *pi = [[NSNumber alloc] initWithDouble: 3.14159];
    NSLog( @"pi is equal to %@", [pi stringValue] );
```

The first line of code allocates and initializes a new instance of NSNumber with the double value 3.14159.

The second line of code calls the instance method -stringValue: to display the value of this object as an NSString.

Step 2. After entering the code, use the Xcode main menu to select Run > Console, then click on the green Build and Run, observing the output in the console.

```
...
... pi is equal to 3.14159.
```

Exercise 6: Releasing Memory

Because we have allocated memory for two instances of NSString and one instance of NSNumber, we must release this memory prior to terminating our program to avoid a memory leak.

Step 1. Assuming that you are continuing from the previous exercise, edit main.m to add the following:

```
    ...
    // release the memory used by our objects
    [string1 release];
    [string2 release];
    [pi release];

    int retVal = UIApplicationMain( argc, argv, nil, nil );
    [pool release];
    return retVal;
}
```

Submission

Step 1. Use Xcode Build > Clean All Targets.

Step 2. From Finder, locate your project directory, Ctrl-Click (or Right-Click) on the project directory, and select Compress <directoryName>.

Step 3. Submit the archive to the Angel Drop box for this assignment.