Introduction to Objective-C (Other DataTypes(NSString, NSDate, NSNumber), Protocols, Category, Extensions)

Sisoft Technologies Pvt Ltd
SRC E7, Shipra Riviera Bazar, Gyan Khand-3, Indirapuram, Ghaziabad
Website: www.sisoft.in Email:info@sisoft.in
Phone: +91-9999-283-283

The id Type

- The id type is the generic type for all Objective-C objects.
- The word ‘id’ indicates an identifier for an object much like a pointer in c

- This uses dynamic typing

- For example, if Pen is a class…

```
extern id Pen;

id myPen;

myPen = [Pen new];
```

id work like generic pointers to objects.

The SEL Type

- The SEL data type is used to store selectors, which are Objective-C's internal representation of a method name
- For example, the following snippet stores a method called sayHello in the someMethod variable.
- This variable could be used to dynamically call a method at runtime.
- SEL someMethod = @selector(sayHello);

The classType

- Objective-C classes are represented as objects themselves, using a special data type called Class.
- This lets you, for example, dynamically check an object's type at runtime.
- All classes implement a class-level method called class that returns its associated class object
- This object can be used for introspection, which we see with the isKindOfClass: method

NSNumber

- The NSNumber class is a lightweight, object-oriented wrapper around C's numeric primitives.
- It's main job is to store and retrieve primitive values, and it comes with dedicated methods for each data type

NSNumber

- NSNumber *aBool = [NSNumber numberWithBool:NO];
- NSNumber *aChar = [NSNumber numberWithChar:'z'];
- NSNumber *aUChar = [NSNumber numberWithUnsignedChar:255];
- NSNumber *aShort = [NSNumber numberWithShort:32767];
- NSNumber *aUShort = [NSNumber numberWithUnsignedShort:65535];
- NSNumber *anInt = [NSNumber numberWithInt:2147483647];
- NSNumber *aUInt = [NSNumber numberWithUnsignedInt:4294967295];
- NSNumber *aLong = [NSNumber numberWithLong:92233720368547758O7];
- NSNumber *aULong = [NSNumber numberWithUnsignedLong:18446744073709551615];
- NSNumber *aFloat = [NSNumber numberWithFloat:26.99f];
- NSNumber *aDouble = [NSNumber numberWithDouble:26.99];

NSNumber

- NSNumber *aBool = [NSNumber numberWithBool:NO];
- NSNumber *aChar = [NSNumber numberWithChar:'z'];
- NSNumber *aUChar = [NSNumber numberWithUnsignedChar:255];
- NSNumber *aShort = [NSNumber numberWithShort:32767];
- NSNumber *aUShort = [NSNumber numberWithUnsignedShort:65535];
- NSNumber *anInt = [NSNumber numberWithInt:2147483647];
- NSNumber *aUInt = [NSNumber numberWithUnsignedInt:4294967295];
- NSNumber *aLong = [NSNumber numberWithLong:92233720368547758O7];
- NSNumber *aULong = [NSNumber numberWithUnsignedLong:18446744073709551615];
- NSNumber *aFloat = [NSNumber numberWithFloat:26.99f];
- NSNumber *aDouble = [NSNumber numberWithDouble:26.99];

NSNumber

- NSLog(@"%@", [aBool boolValue] ? @"YES" : @"NO");
- NSLog(@"%c", [aChar charValue]);
- NSLog(@"%hhu", [aUChar unsignedCharValue]);
- NSLog(@"%hi", [aShort shortValue]);
- NSLog(@"%hu", [aUShort unsignedShortValue]);
- NSLog(@"%i", [anInt intValue]);
- NSLog(@"%u", [aUInt unsignedIntValue]);
- NSLog(@"%li", [aLong longValue]);
- NSLog(@"%lu", [aULong unsignedLongValue]);
- NSLog(@"%f", [aFloat floatValue]);
- NSLog(@"%f", [aDouble doubleValue]);

NSNumber: Comparison

- Methods
- isEqualToNumber (Returns Boolean true/false)
  - if ([anInt isEqualToNumber:sameInt])
- Compare
  - it returns an NSComparisonResult, which is an enum that defines the relationship between the operands:

Return Value	**Description**
NSOrderedAscending	receiver < argument
NSOrderedSame	receiver == argument
NSOrderedDescending	receiver > argument

NSDecimalNumber

- The NSDecimalNumber class provides fixed-point arithmetic capabilities to Objective-C programs.
- They’re designed to perform base-10 calculations without loss of precision and with predictable rounding behavior.
- The following snippet creates the value 15.99 using both methods.

```
NSDecimalNumber *price;
price = [NSDecimalNumber decimalNumberWithMantissa:1599
                                              exponent:-2    isNegative:NO];
price = [NSDecimalNumber decimalNumberWithString:@"15.99"];
```

String Object

- A string object is implemented as an array of Unicode characters
- An immutable string is a text string that is defined when it is created and subsequently cannot be changed. To create and manage an immutable string, use the **NSString** class
- To construct and manage a string that can be changed after it has been created, use **NSMutableString**
- The term *C string* refers to the standard C char * type.

NSString

- Creating String
  - Using @ contruct
    - NSString *theMessage = @”hello world”;
  - Using C String data
    - initWithUTF8String
    - NSString n1 = [[NSString alloc] initWithUTF8String: cStr]
    - initWithCString
    - NSString n1 = [[NSString alloc] initWithCString: cStr]
  - Using format specifier (stringWithFormat: or initWithFormat:)
    - NSString *msg = [NSString stringWithFormat:@”This is %@”, theMessage] ;

NSString: Format Specifier

Specifier	Description
%@	Objective-C object, printed as the string returned by descriptionWithLocale: if available, or description otherwise. Also works with CFTypeRef objects, returning the result of the CFCopyDescription function.
%%	'%' character.
%d, %D	Signed 32-bit integer (int).
%u, %U	Unsigned 32-bit integer (unsigned int).
%x	Unsigned 32-bit integer (unsigned int), printed in hexadecimal using the digits 0–9 and lowercase a–f.
%X	Unsigned 32-bit integer (unsigned int), printed in hexadecimal using the digits 0–9 and uppercase A–F.
%o, %O	Unsigned 32-bit integer (unsigned int), printed in octal.
%f	64-bit floating-point number (double).
%e	64-bit floating-point number (double), printed in scientific notation using a lowercase e to introduce the exponent.
%E	64-bit floating-point number (double), printed in scientific notation using an uppercase E to introduce the exponent.
%g	64-bit floating-point number (double), printed in the style of %e if the exponent is less than –4 or greater than or equal to the precision, in the style of %f otherwise.
%G	64-bit floating-point number (double), printed in the style of %E if the exponent is less than –4 or greater than or equal to the precision, in the style of %f otherwise.
%c	8-bit unsigned character (unsigned char), printed by NSLog() as an ASCII character, or, if not an ASCII character, in the octal format \\ddd or the Unicode hexadecimal format \\udddd, where d is a digit.
%C	16-bit Unicode character (unichar), printed by NSLog() as an ASCII character, or, if not an ASCII character, in the octal format \\ddd or the Unicode hexadecimal format \\udddd, where d is a digit.
%s	Null-terminated array of 8-bit unsigned characters. Because the %s specifier causes the characters to be interpreted in the system default encoding, the results can be variable, especially with right-to-left languages. For example, with RTL, %s inserts direction markers when the characters are not strongly directional. For this reason, it's best to avoid %s and specify encodings explicitly.
%S	Null-terminated array of 16-bit Unicode characters.
%p	Void pointer (void *), printed in hexadecimal with the digits 0–9 and lowercase a–f, with a leading 0x.
%a	64-bit floating-point number (double), printed in scientific notation with a leading 0x and one hexadecimal digit before the decimal point using a lowercase p to introduce the exponent.
%A	64-bit floating-point number (double), printed in scientific notation with a leading 0X and one hexadecimal digit before the decimal point using a uppercase P to introduce the exponent.
%F	64-bit floating-point number (double), printed in decimal notation

NSString

- Enumerating String
  - Number of characters in a string (length)
    - NSUInteger charCount = [theMessage length];
  - Character at a given index (characterAtIndex:)
    - Unichar char = [str characterAtIndex:i];
- Comparing
  - Test if 2 strings equal
    - if([string_var_1 isEqual: string_var_2])
      { //code for equal case }
- Getting C String
  - UTF8String returns const char *

NSStrings : Code fragment

```
NSString *hello = @”Hello World” ;
  NSLog (@“%@”, hello);
char *ulike = "I like you" ;
NSString *n1 ;
n1 = [[NSString alloc] initWithUTF8String:ulike];
  NSLog(@"%@", n1) ;
NSString *n2 ;
n2 = [[NSString alloc] initWithCString:ulike];
  NSLog(@"%@", n2) ;
const char *u2 = [n2 UTF8String] ;
 printf("C Style %s\n", u2) ;
```

NSMutableString ---Mutable

- String whose content can be changed without forming any new object
- NSMutableString inherits from NSString, so all the methods of NSString will apply here

```
NSMutableString *ms =  [[NSMutableString alloc] initWithString:@“hello”];
                       [ms appendString:digit];
```

- (NSMutableString *)stringWithCapacity:(NSUInteger)*capacity*
  - Returns an empty NSMutableString object with initial storage for a given number of characters.
- (NSMutableString *)initWithCapacity:(NSUInteger)*capacity*
  - Returns an NSMutableString object initialized with initial storage for a given number of characters

NSDate

- NSDate allows you to represent an absolute point in time.
- Date objects allow you to store absolute points in time which are meaningful across locales, calendars and timezones
- To get current time,
  - Allocate NSDate object and initialize it with init
  - Use the date method to create date object
  - eg
    - NSDate *now1 = [[NSDate alloc] init]
    - NSDate *now2 = [NSDate date]

NSDate

- To get time other than the current time, NSDate's initWithTimeInterval... or dateWithTimeInterval... Methods should be used

```
NSTimeInterval secondsInWeek = 7 * 24 * 60 * 60;
NSDate *nextWeek = [[NSDate alloc] initWithTimeIntervalSinceNow:secondsInWeek];
NSDate *nextWeek = [NSDate dateWithTimeInterval:secondsInWeek sinceDate:now];
```

- To compare dates, you can use the
  - isEqualToDate: Returns a Boolean value that indicates whether a given object is an NSDate object and exactly equal the receiver.
  - compare: Returns an NSComparisonResult value that indicates the temporal ordering of the receiver and another given date
  - laterDate: Returns the later of the receiver and another given date
  - earlierDate: Returns the earlier of the receiver and another given date

NSDate

- **Create a Date From Day, Month and Year**

```
NSDateComponents* comps = [[NSDateComponents
alloc]init];
comps.year = 2014;
comps.month = 3;
comps.day = 31;

NSCalendar* calendar = [NSCalendar currentCalendar];

NSDate* date = [calendar dateFromComponents:comps];
```

NSDate

- **Convert a Date to a String:**

```
NSDate* date = [NSDate date];
NSDateFormatter* formatter = [[NSDateFormatter alloc]init];
formatter.dateFormat = @"MMMM dd, yyyy";
NSString* dateString = [formatter stringFromDate:date];
```

- **Convert a String to Date:**

```
NSDateFormatter* formatter = [[NSDateFormatter alloc]init];
formatter.dateFormat = @"MMMM dd, yyyy";
NSDate* date = [formatter dateFromString:@"August 02, 2014"];
```

OOPs Concepts
(Protocols, Category, Extensions)

Sisoft Technologies Pvt Ltd
SRC E7, Shipra Riviera Bazar, Gyan Khand-3, Indirapuram, Ghaziabad
Website: www.sisoft.in Email:info@sisoft.in
Phone: +91-9999-283-283

Protocols

- *A protocol is a defined set of methods that a class can choose to implement*
- Objective-C protocols are the equivalent of Java interfaces
- A protocol is effectively an interface that's not tied to a class. It declares a set of methods, listing their arguments and their returns.
- Classes can then state that they're using the protocol in their own @interface statements
- Protocol declare methods that can implemented by any class
- Declare methods that others are expected to implement

Protocols Declaration

- Protocol names are enclosed in angle brackets
- A protocol is declared by declaring its methods between @protocol and @end compiler directives
- For the <DrawableItem> protocol, the declaration looks like this:

```
@protocol DrawableItem
- (void) drawItem;
-(int) boundingBox;
- (void) setColor:(NSColor*) color;
@end
```

Protocols Declaration contd…

- The protocol declaration goes in a header file, so you could put this declaration in a header file named. *anyClass.h.There is no corresponding implementation file*
- Objective-C 2.0 allows you to mark protocol methods as either optional (@optional) or required(@required):
  - A class that adopts a protocol must implement all of the protocol's required methods.
  - A class that adopts a protocol is free to implement or not implement any of the protocol's optional methods.

Adopting a Protocol

- A class can adopt a protocol by adding the protocol name, enclosed in angle brackets, to the class's @interface line:

```
@interface myClass : NSObject <protocolName>
```

- A class can adopt more than one protocol. The protocols are listed, separated by commas, between a single set of angle brackets.

```
@interface myClass : NSObject <protocolName1,
                               protocolName2,protocolName3>
```

A class adopts a protocol by implementing all the protocol's required methods and any or none of the protocol's optional methods.

Category

- Categories let you add methods to an existing class without subclassing it and without requiring access to the class's source code
- Using a category to extend the behavior of a class is a much lighter-weight procedure than subclassing
- New method declaration is added in the category @interface section and code for method in @implementation section

Category -Declaration

```
#import <Foundation/Foundation.h>
@interface NSString (CamelCase)
-(NSString*) camelCaseString;
@end
```

- One big difference between a category and a subclass is that a category cannot add any variables to a class. The header file reflects this: It has no instance variable section

Class Extensions

- A class extension bears some similarity to a category, but it can only be added to a class for which you have the source code at compile time (the class is compiled at the same time as the class extension).
- The methods declared by a class extension are implemented in the @implementation block for the original class so you can't, for example, declare a class extension on a framework class, such as a Cocoa or Cocoa Touch class like NSString.

Class Extensions

- The syntax to declare a class extension is similar to the syntax for a category, and looks like this:

```
@interface ClassName ()
@end
```

- Because no name is given in the parentheses, class extensions are often referred to as *anonymous categories* .