OBJECTIVE-C CLASSES & OBJECTS

The main purpose of Objective-C programming language is to add object orientation to the C programming language and classes are the central feature of Objective-C that support object-oriented programming and are often called user-defined types.

A class is used to specify the form of an object and it combines data representation and methods for manipulating that data into one neat package. The data and methods within a class are called members of the class.

Objective-C characteristics

- The class is defined in two different sections namely **@interface** and **@implementation**.
- Almost everything is in form of objects.
- Objects receive messages and objects are often referred as receivers.
- Objects contain instance variables.
- Objects and instance variables have scope.
- Classes hide an object's implementation.
- Properties are used to provide access to class instance variables in other classes.

Objective-C Class Definitions:

When you define a class, you define a blueprint for a data type. This doesn't actually define any data, but it does define what the class name means, that is, what an object of the class will consist of and what operations can be performed on such an object.

A class definition starts with the keyword **@interface** followed by the interface*class* name; and the class body, enclosed by a pair of curly braces. In Objective-C, all classes are derived from the base class called **NSObject**. It is the superclass of all Objective-C classes. It provides basic methods like memory allocation and initialization. For example, we defined the Box data type using the keyword **class** as follows:

```
@interface Box:NSObject
{
    //Instance variables
    double length;   // Length of a box
    double breadth;  // Breadth of a box
}
@property(nonatomic, readwrite) double height; // Property

@end
```

The instance variables are private and are only accessible inside the class implementation.

Allocating and initializing Objective-C Objects:

A class provides the blueprints for objects, so basically an object is created from a class. We declare objects of a class with exactly the same sort of declaration that we declare variables of basic types. Following statements declare two objects of class Box:

```
Box box1 = [[Box alloc]init];     // Create box1 object of type Box
Box box2 = [[Box alloc]init];     // Create box2 object of type Box
```

Both of the objects box1 and box2 will have their own copy of data members.

Accessing the Data Members:

The properties of objects of a class can be accessed using the direct member access operator . . Let us try the following example to make things clear:

```
#import <Foundation/Foundation.h>

@interface Box:NSObject
{
    double length;   // Length of a box
    double breadth;  // Breadth of a box
    double height;   // Height of a box
}
@property(nonatomic, readwrite) double height; // Property

-(double) volume;

@end

@implementation Box

@synthesize height;

-(id)init
{
   self = [super init];
   length = 1.0;
   breadth = 1.0;
   return self;
}

-(double) volume
{
   return length*breadth*height;
}

@end

int main( )
{
   NSAutoreleasePool * pool = [[NSAutoreleasePool alloc] init];
   Box *box1 = [[Box alloc]init];    // Create box1 object of type Box
   Box *box2 = [[Box alloc]init];    // Create box2 object of type Box

   double volume = 0.0;     // Store the volume of a box here

   // box 1 specification
   box1.height = 5.0;

   // box 2 specification
   box2.height = 10.0;

   // volume of box 1
   volume = [box1 volume];
   NSLog(@"Volume of Box1 : %f", volume);
   // volume of box 2
   volume = [box2 volume];
   NSLog(@"Volume of Box2 : %f", volume);
   [pool drain];
   return 0;
}
```

When the above code is compiled and executed, it produces the following result:

```
2013-09-22 21:25:33.314 ClassAndObjects[387:303] Volume of Box1 : 5.000000
2013-09-22 21:25:33.316 ClassAndObjects[387:303] Volume of Box2 : 10.000000
```

Properties:

Properties are introduced in Objective-C to ensure that the instance variable of the class can be accessed outside the class.

The various parts are the property declaration are as follows.

- Properties begin with **@property**, which is a keyword
- It is followed with access specifiers, which are nonatomic or atomic, readwrite or readonly and strong, unsafe_unretained or weak. This varies based on the type of the variable. For any pointer type, we can use strong, unsafe_unretained or weak. Similarly for other types we can use readwrite or readonly.
- This is followed by the datatype of the variable.
- Finally, we have the property name terminated by a semicolon.
- We can add synthesize statement in the implementation class. But in the latest XCode, the synthesis part is taken care by the XCode and you need not include synthesize statement.

It is only possible with the properties we can access the instance variables of the class. Actually, internally getter and setter methods are created for the properties.

For example, let's assume we have a property **@property** ***nonatomic, readonly*** **BOOL isDone**. Under the hood, there are setters and getters created as shown below.

```
-(void)setIsDone(BOOL)isDone;
-(BOOL)isDone;
```