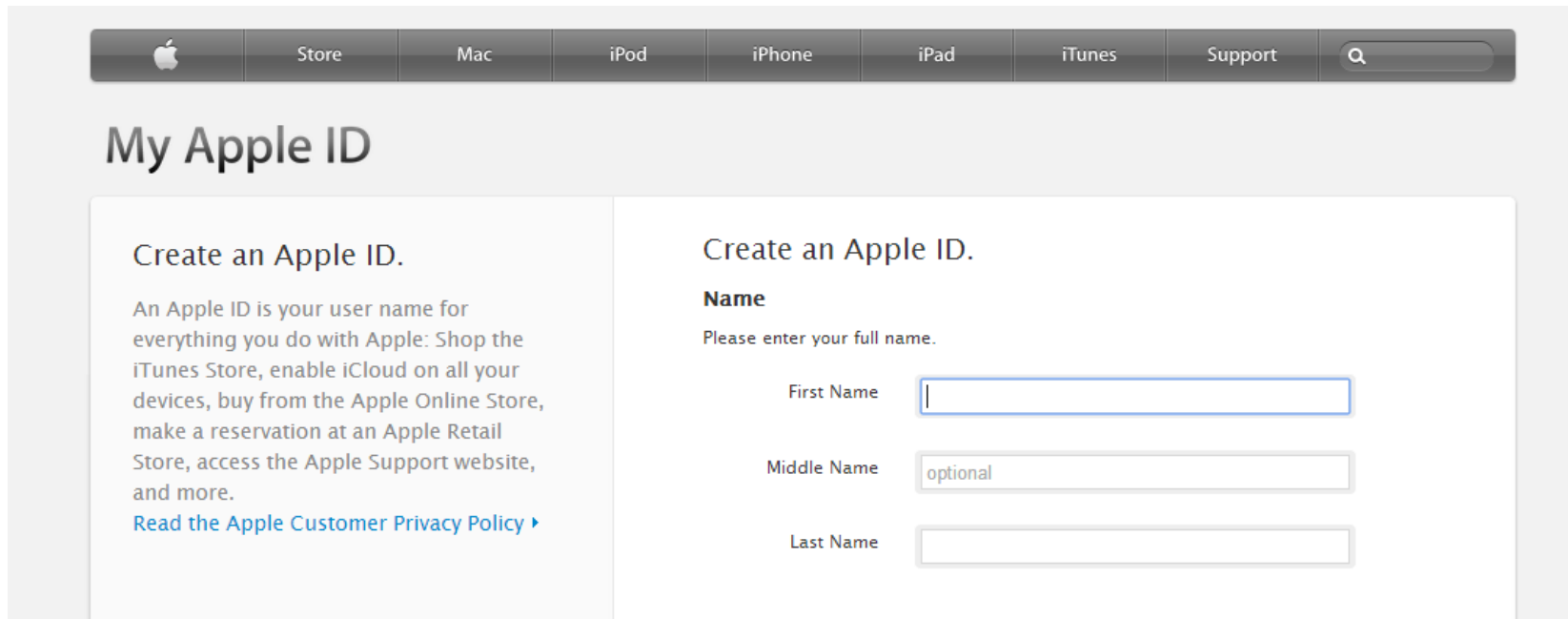


iPhone Apps Development using Objective C & Xcode

By Dannis Mok

Create an Apple ID

- One must have the Apple ID to download the Development Tools
(<https://appleid.apple.com/account>)



The screenshot shows the Apple website's navigation bar with links for Store, Mac, iPod, iPhone, iPad, iTunes, and Support, along with a search icon. Below the navigation bar is the heading "My Apple ID". The main content area is split into two columns. The left column contains the text "Create an Apple ID." followed by a paragraph explaining the benefits of an Apple ID and a link to "Read the Apple Customer Privacy Policy". The right column also has the heading "Create an Apple ID." and a section titled "Name" with the instruction "Please enter your full name." Below this are three input fields: "First Name" (empty), "Middle Name" (with the placeholder text "optional"), and "Last Name" (empty).

Register as Apple Developer

- One must register as Apple Developer to deploy the apps onto the device for testing and also posted the apps to App Store for selling.
(<https://developer.apple.com/register/index.action>)



Environment Setup

- Download the Xcode IDE from Apple and install onto the MAC.
(<https://developer.apple.com/downloads/index.action>)

The screenshot shows the Apple Developer website interface. At the top, there is a navigation bar with the Apple Developer logo, links for Technologies, Resources, Programs, Support, and Member Center, and a search bar. Below the navigation bar, the main heading is "Downloads for Apple Developers". On the right side of this heading, it says "Hi, Dannis Mok" with links for "My Profile" and "Sign out".

On the left side, there is a "Categories" section with a search bar and a list of categories with checkboxes:

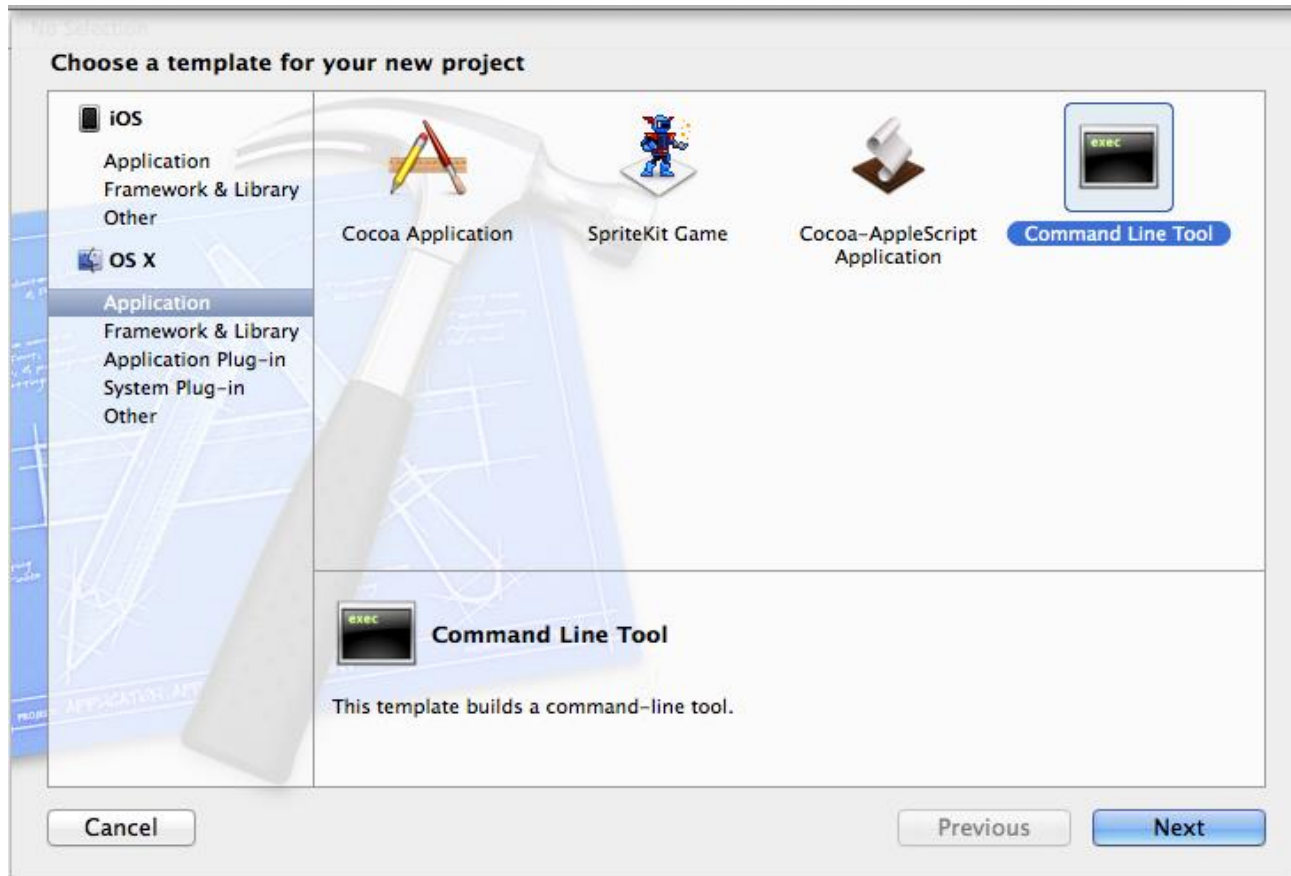
- Applications (11)
- Developer Tools (196)
- iOS (13)
- OS X (65)
- OS X Server (9)
- Safari (1)

The main content area displays a table of downloads. The table has two columns: "Description" and "Release Date". The first row shows "Graphics Tools for Xcode - Late August 2014" with a release date of "Aug 18, 2014". The second row shows "Command Line Tools (OS X 10.9) for Xcode - Late August 2014" with a release date of "Aug 18, 2014". The third row shows "Hardware IO Tools for Xcode - Late August 2014" with a release date of "Aug 18, 2014". The fourth row shows "Command Line Tools (OS X 10.9) for Xcode - August 2014" with a release date of "Aug 4, 2014".

Description	Release Date
▶ Graphics Tools for Xcode - Late August 2014	Aug 18, 2014
▶ Command Line Tools (OS X 10.9) for Xcode - Late August 2014	Aug 18, 2014
▶ Hardware IO Tools for Xcode - Late August 2014	Aug 18, 2014
▶ Command Line Tools (OS X 10.9) for Xcode - August 2014	Aug 4, 2014

New Command Line Project

- Use Xcode to create a new command line project



New Command Line Project

- Fill in the Project name and create

No selection

Choose options for your new project:

Product Name

Organization Name

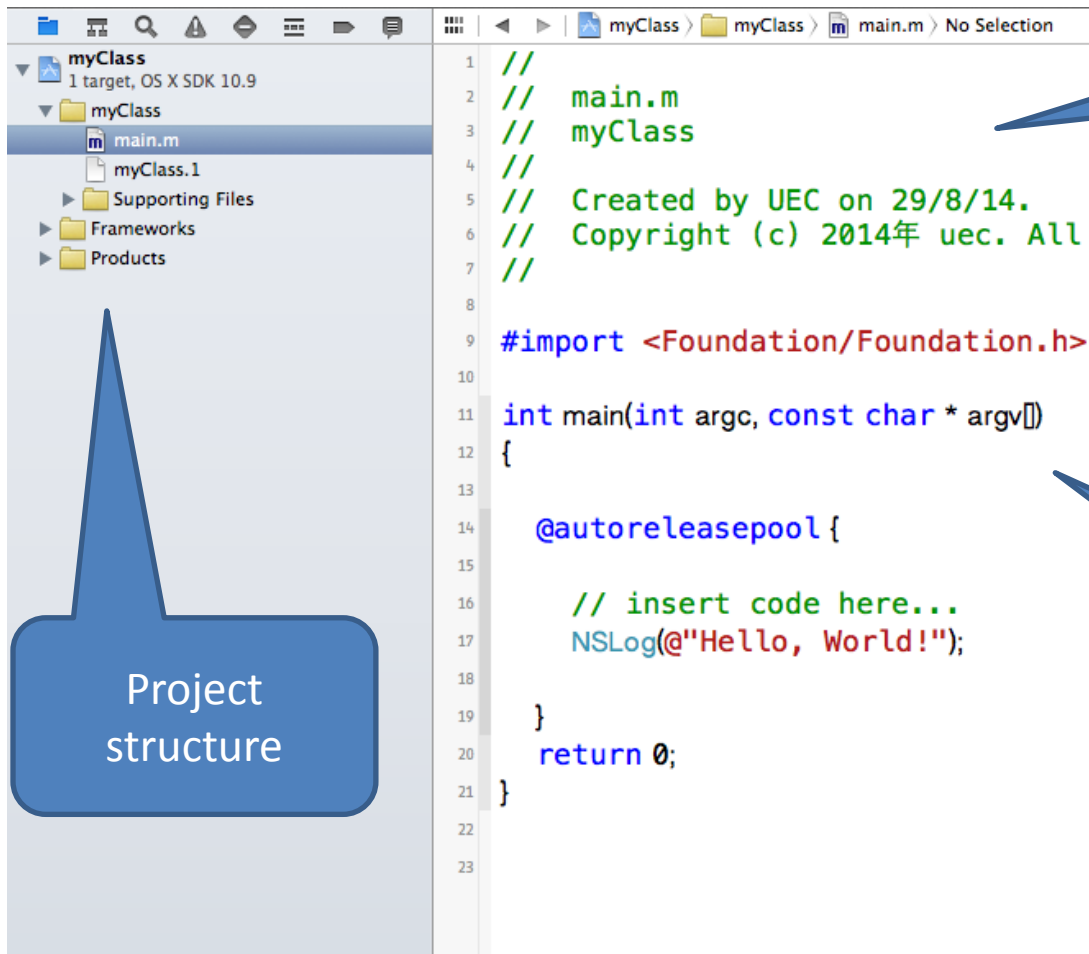
Company Identifier

Bundle Identifier

Type

Cancel Previous Next

New Command Line Project



Comments begin with //

Project structure

Program Structure. Main() is the entry point of the program

Objective-C Introduction

- It is based on traditional C programming language and syntax.
- It add the object-oriented features to the C language and have some new syntax.
- Its root is from NEXT computer, a company owned by Steve Jobs

Objective-C – (Variables)

- Variables
 - Variable name must start with letter or underline (_).
 - Variables are case sensitive.
 - Variable name cannot be the reserved keywords

auto	else	long	switch
break	enum	register	typedef
case	extern	return	union
char	float	short	unsigned
const	for	signed	void
continue	goto	sizeof	volatile
default	if	static	while
do	int	struct	_Packed
double	protocol	interface	implementation
NSObject	NSInteger	NSNumber	CGFloat
property	nonatomic;	retain	strong
weak	unsafe_unretained;	readwrite	readonly

Objective-C – (Data Types)

- Variables must be defined to contain certain type of data.
- e.g `int k = 10;`

Type	Storage size	Value range
char	1 byte	-128 to 127 or 0 to 255
unsigned char	1 byte	0 to 255
signed char	1 byte	-128 to 127
int	2 or 4 bytes	-32,768 to 32,767 or -2,147,483,648 to 2,147,483,647
unsigned int	2 or 4 bytes	0 to 65,535 or 0 to 4,294,967,295
short	2 bytes	-32,768 to 32,767
unsigned short	2 bytes	0 to 65,535
long	4 bytes	-2,147,483,648 to 2,147,483,647
unsigned long	4 bytes	0 to 4,294,967,295

Objective-C – (Data Types)

- Data Types and their usage.

Type	Description
char	Typically a single octet (one byte). This is an integer type.
int	The most natural size of integer for the machine.
float	A single-precision floating point value.
double	A double-precision floating point value.
void	Represents the absence of type.



A special data type

Objective-C – (Input / Output)

- NSLog is used to print out the content to the log file. Its main use is for debugging in development.

e.g

```
NSLog(@"Hello World \n Welcome !");
```



Must use @ before a string



New Line characters

Objective-C – (Input / Output)

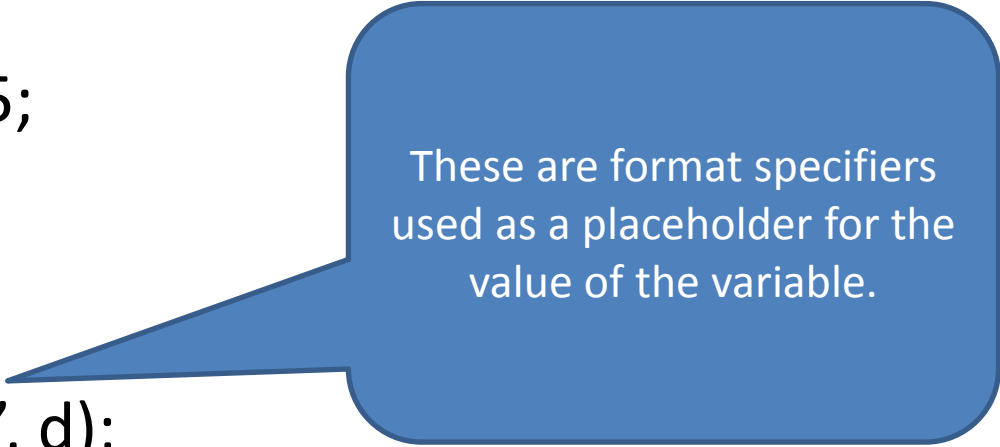
e.g.

```
double d= 123.45;
```

```
char ch = 'd';
```

```
NSLog(@"d = %f", d);
```

```
NSLog(@"ch=%c", ch);
```



These are format specifiers used as a placeholder for the value of the variable.

Objective-C – (Input / Output)

- Common format specifiers

Specifier	Descriptions
%d, %D, %i	Signed 32 bit integer (int)
%u, %U	Unsigned 32 bit integer (unsigned int)
%x, %X	Unsigned 32 bit integer, printed in hexadecimal
%o, %O	Unsigned 32 bit integer, printed in octal
%f	64 bit floating point number (double)
%e, %E, %g, %G	64 bit floating point number (double) with scientific notation
%c	8 bit unsigned character
%@	Objective C object, like String

Objective-C (Input / Output)

- Traditional C's printf() and scanf() can also be used for input and output.

e.g.

```
float amount;  
NSLog(@"Please enter your amount");  
scanf("%f", &amount);
```

Address Operator must
be added

```
NSLog(@"Your amount is %f", amount);  
printf("Welcome for your help");
```

No need to use @

Objective-C – (Operators)

- An operator is a symbol that tells the compiler to perform specific mathematical or logical manipulations
- Types of Operators
 - Arithmetic Operators
 - Relational Operators
 - Logical Operators
 - Bitwise Operators
 - Assignment Operators
 - Misc Operators

Objective-C – (Operators)

- Arithmetic Operators

Operator	Description	Example
+	Adds two operands	A + B will give 30
-	Subtracts second operand from the first	A - B will give -10
*	Multiplies both operands	A * B will give 200
/	Divides numerator by denominator	B / A will give 2
%	Modulus Operator and remainder of after an integer division	B % A will give 0
++	Increment operator increases integer value by one	A++ will give 11
--	Decrement operator decreases integer value by one	A-- will give 9

Objective-C – (Operators)

- Relational Operators

Operator	Description	Example
==	Checks if the values of two operands are equal or not; if yes, then condition becomes true.	(A == B) is not true.
!=	Checks if the values of two operands are equal or not; if values are not equal, then condition becomes true.	(A != B) is true.
>	Checks if the value of left operand is greater than the value of right operand; if yes, then condition becomes true.	(A > B) is not true.
<	Checks if the value of left operand is less than the value of right operand; if yes, then condition becomes true.	(A < B) is true.
>=	Checks if the value of left operand is greater than or equal to the value of right operand; if yes, then condition becomes true.	(A >= B) is not true.
<=	Checks if the value of left operand is less than or equal to the value of right operand; if yes, then condition becomes true.	(A <= B) is true.

Objective-C – (Operators)

- Logical Operators

Operator	Description	Example
&&	Called Logical AND operator. If both the operands are non zero then condition becomes true.	(A && B) is false.
	Called Logical OR Operator. If any of the two operands is non zero then condition becomes true.	(A B) is true.
!	Called Logical NOT Operator. Use to reverses the logical state of its operand. If a condition is true, then Logical NOT operator will make false.	!(A && B) is true.

Objective-C – (Operators)

- Bitwise Operators

p	q	p & q	p q	p ^ q
0	0	0	0	0
0	1	0	1	1
1	1	1	1	0
1	0	0	1	1

e.g.

Assume if A = 60; and B = 13; now in binary format they will be as follows:

A = 0011 1100

B = 0000 1101

A&B = 0000 1100

A|B = 0011 1101

A^B = 0011 0001

~A = 1100 0011

Objective-C – (Operators)

- Assignment Operators

Operator	Description	Example
=	Simple assignment operator, Assigns values from right side operands to left side operand	$C = A + B$ will assign value of $A + B$ into C
+=	Add AND assignment operator, It adds right operand to the left operand and assigns the result to left operand	$C += A$ is equivalent to $C = C + A$
-=	Subtract AND assignment operator, It subtracts right operand from the left operand and assigns the result to left operand	$C -= A$ is equivalent to $C = C - A$
*=	Multiply AND assignment operator, It multiplies right operand with the left operand and assigns the result to left operand	$C *= A$ is equivalent to $C = C * A$
/=	Divide AND assignment operator, It divides left operand with the right operand and assigns the result to left operand	$C /= A$ is equivalent to $C = C / A$
%=	Modulus AND assignment operator, It takes modulus using two operands and assigns the result to left operand	$C \% = A$ is equivalent to $C = C \% A$

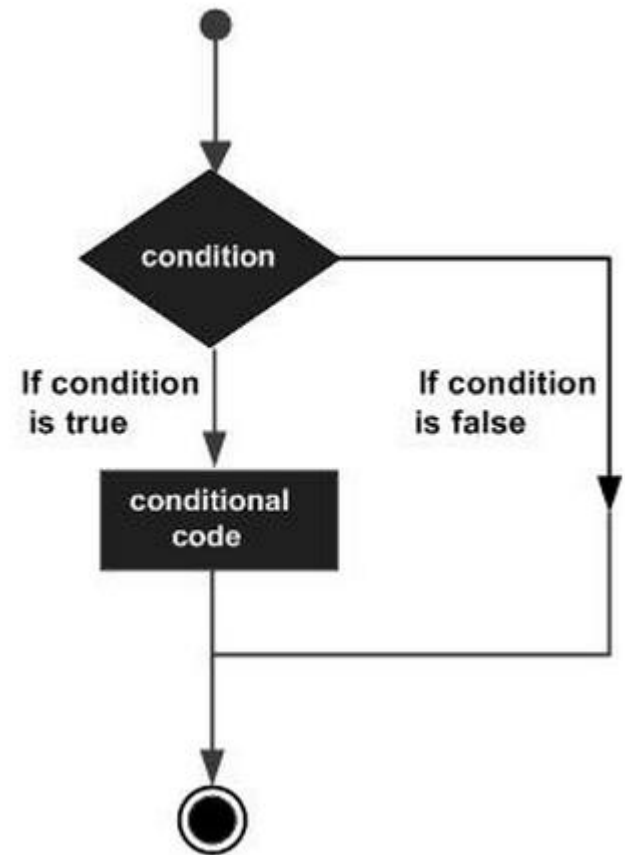
Objective-C – (Operators)

- Misc Operators

Operator	Description	Example
sizeof()	Returns the size of an variable.	sizeof(a), where a is integer, will return 4.
&	Returns the address of an variable.	&a; will give actual address of the variable.
*	Pointer to a variable.	*a; will pointer to a variable.
? :	Conditional Expression	If Condition is true ? Then value X : Otherwise value Y

Objective-C – (Decision Making)

- Condition will be true if it is non-zero or non-null value
- Condition will be false if it is zero or null value



Objective-C – (Decision Making)

- Syntax examples
 - If condition

```
if(boolean_expression)
{
    /* statement(s) will execute if the boolean expression is true */
}
```

- If else condition

```
if(boolean_expression)
{
    /* statement(s) will execute if the boolean expression is true */
}
else
{
    /* statement(s) will execute if the boolean expression is false */
}
```


Objective-C – (Decision Making)

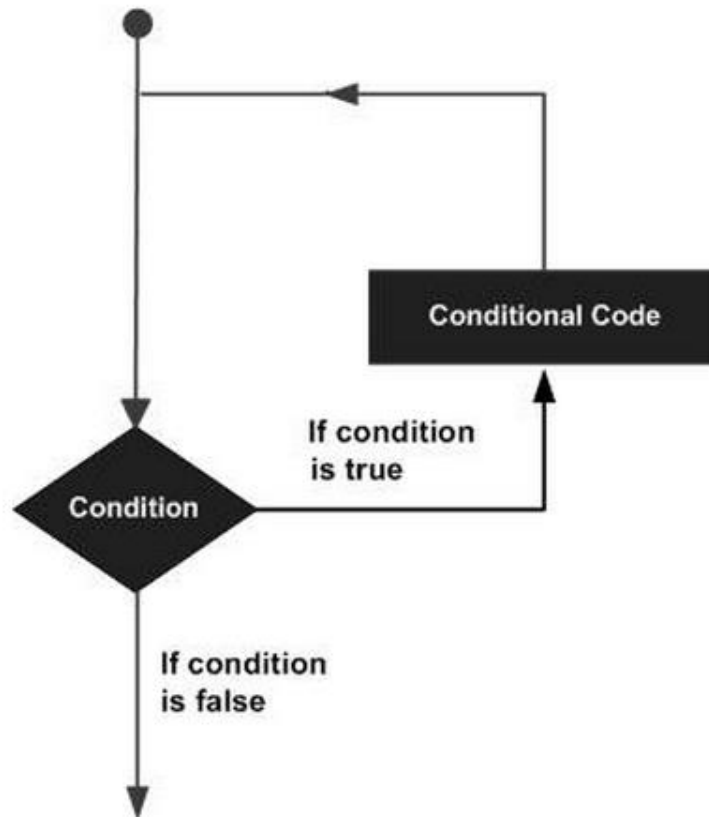
- switch-case

```
switch(expression) {  
    case constant-expression :  
        statement(s);  
        break; /* optional */  
    case constant-expression :  
        statement(s);  
        break; /* optional */  
  
    /* you can have any number of case statements */  
    default : /* Optional */  
        statement(s);  
}
```

Remind to add break statement

Objective-C – (Loop)

- A loop statement allows us to execute a statement or group of statements multiple times



Objective-C – (Loop)

- Syntax examples
 - while loop

```
while(condition)
{
    statement(s);
}
```

- do while loop

```
do
{
    statement(s);
}while( condition );
```

Objective-C – (Loop)

- for loop

```
for ( init; condition; increment )  
{  
    statement(s);  
}
```

- nested loop

```
for ( init; condition; increment )  
{  
    for ( init; condition; increment )  
    {  
        statement(s);  
    }  
    statement(s);  
}
```

```
while(condition)  
{  
    while(condition)  
    {  
        statement(s);  
    }  
    statement(s);  
}
```

Objective-C – (Functions)

- A function is a group of statements that together perform a task
- Every Objective-C program has one C function, which is **main()**
- A function **declaration** tells the compiler about a function's name, return type, and parameters
- A function **definition** provides the actual body of the function.

Objective-C – (Functions)

- In Objective-C, we called functions as methods
- Method syntax

```
- (return_type) method_name:( argumentType1 )argumentName1  
joiningArgument2:( argumentType2 )argumentName2 ...  
joiningArgumentn:( argumentTypen )argumentNamen  
{  
    body of the function  
}
```

e.g.

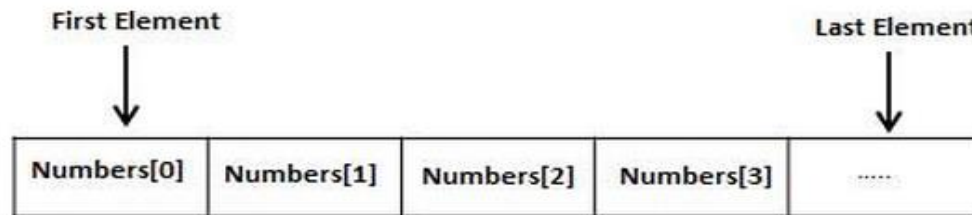
```
- (int) max: (int) num1 secondNumber: (int) num2 {  
    .....  
}
```

Objective-C – (Functions)

- A function declaration
 - (int) max: (int) num1 secondNumber: (int) num2;
- A function implementation
 - (int) max: (int) num1 secondNumber: (int) num2 {
.....
}
- A function call
 - [*ObjectName* max:30 secondNumber:20];

Objective-C – (Array)

- Array is a data structure used to store a collection of data



- Declaring Array
`int balance[10];`
- Initializing Array
`int balance[5] = {10,20,30,40,50};`
- Accessing Array value
`amount = balance[3];`

Objective-C – (Pointers)

- Each variable has a memory location and the memory address can be accessed by & operator.

e.g.

```
int x = 10;
```

```
NSLog(@"Address of x is %f", &x);
```

Output can be – Address of x is 1c083ff

Objective-C – (Pointers)

- A **pointer** is a variable whose value is address of the memory location

e.g

```
int *p1    // p1 is a pointer to integer
float *p2  // p2 is a pointer to a float
double *p3 // p3 is a pointer to a double
```

- Assign NULL to a pointer for initialization

```
int *p1 = NULL;
```

Objective-C – (Pointers)

- How to use pointer ?

```
int  var = 20;    /* actual variable declaration */
int  *ip;        /* pointer variable declaration */

ip = &var;      /* store address of var in pointer variable*/

NSLog(@"Address of var variable: %x\n", &var );

/* address stored in pointer variable */
NSLog(@"Address stored in ip variable: %x\n", ip );

/* access the value using the pointer */
NSLog(@"Value of *ip variable: %d\n", *ip );
```

```
Address of var variable: 337ed41c
Address stored in ip variable: 337ed41c
Value of *ip variable: 20
```

Objective-C – (String)

- A string is represented by NSString or NSMutableString

e.g



Must use a pointer variable

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

```
NSLog(@"Message %@", greeting);
```

Objective-C – (String)

- Common String methods

1	- (NSString *)capitalizedString; Returns a capitalized representation of the receiver.
2	- (unichar)characterAtIndex:(NSUInteger)index; Returns the character at a given array position.
3	- (double)doubleValue; Returns the floating-point value of the receiver's text as a double.
4	- (float)floatValue; Returns the floating-point value of the receiver's text as a float.
5	- (BOOL)hasPrefix:(NSString *)aString; Returns a Boolean value that indicates whether a given string matches the beginning characters of the receiver.
6	- (BOOL)hasSuffix:(NSString *)aString; Returns a Boolean value that indicates whether a given string matches the ending characters of the receiver.
7	- (id)initWithFormat:(NSString *)format ...; Returns an NSString object initialized by using a given format string as a template into which the remaining argument values are substituted.
8	- (NSInteger)integerValue; Returns the NSInteger value of the receiver's text.

Objective-C – (String)

9	- (BOOL)isEqualToString:(NSString *)aString; Returns a Boolean value that indicates whether a given string is equal to the receiver using a literal Unicode-based comparison.
10	- (NSUInteger)length; Returns the number of Unicode characters in the receiver.
11	- (NSString *)lowercaseString; Returns lowercased representation of the receiver.
12	- (NSRange)rangeOfString:(NSString *)aString; Finds and returns the range of the first occurrence of a given string within the receiver.
13	- (NSString *)stringByAppendingFormat:(NSString *)format ...; Returns a string made by appending to the receiver a string constructed from a given format string and the following arguments.
14	- (NSString *)stringByTrimmingCharactersInSet:(NSCharacterSet *)set; Returns a new string made by removing from both ends of the receiver characters contained in a given character set.
15	- (NSString *)substringFromIndex:(NSUInteger)anIndex ; Returns a new string containing the characters of the receiver from the one at a given index to the end.

Objective-C – (String)

- String usage example

```
NSString *str1 = @"Hello";
NSString *str2 = @"World";
NSString *str3;
int len ;

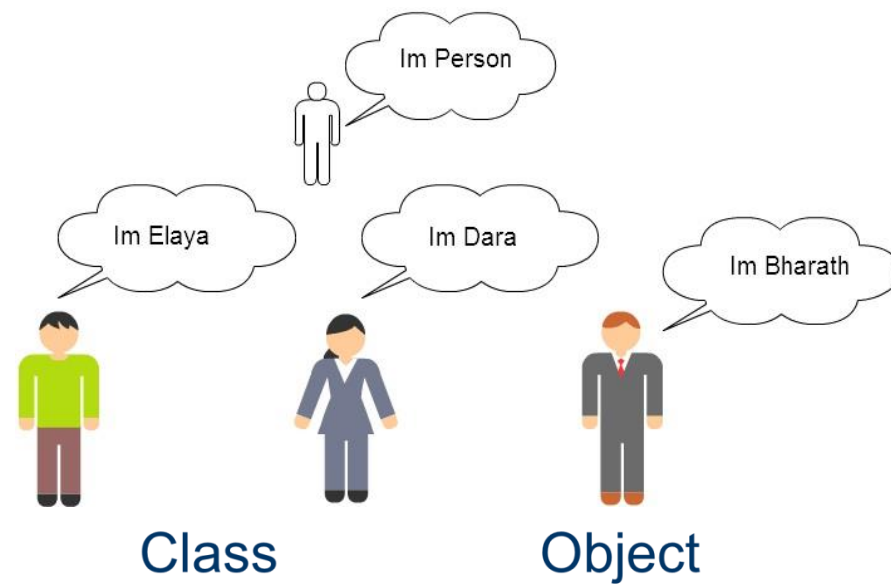
/* uppercase string */
str3 = [str2 uppercaseString];
NSLog(@"Uppercase String : %@\n", str3 );

/* concatenates str1 and str2 */
str3 = [str1 stringByAppendingFormat:@"World"];
NSLog(@"Concatenated string:  %@\n", str3 );

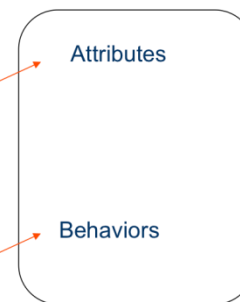
/* total length of str3 after concatenation */
len = [str3 length];
NSLog(@"Length of Str3 : %d\n", len );
```

Objective-C – (Class/Objects)

- A class is a template or blueprint for creating objects.
- A class combines both data and methods together into a single package.
- Data and methods are called members of the class.



- A blueprint for objects of a particular type
- Defines the structure (number, types) of the attributes
- Defines available behaviors of its objects



Objective-C – (Class/Objects)

- Class definition
 - A class starts with @interface followed by the interface(class) name; and the class body, enclosed by a pair of curly braces.
 - All classes are derived from the base class called **NSObject**

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

Data of the class.
Called the instance
variables

Method of the class. Called
the instance methods

Objective-C – (Class/Objects)

- Class implementation is the 2nd part of the class definition. It will start with `@implementation` and end with `@end`.
- The interface part and the implementation part be in the same file or in the separate files.

```
@implementation Box

@synthesize height;

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

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

@end
```

Objective-C – (Class/Objects)

- Creating Objects

- An object is created from the class by allocating memory and then initialized

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

- Both objects will have its own data members and is independent

Objective-C – (Class/Objects)

- Property is added to allow instance variable of the class can be accessed outside the class.
- It will generate the getter and setter methods automatically.
- Need to add synthesize statement in the implementation class. But can be ignored in latest XCode

Objective-C – (Class/Objects)

- An instance variable with property can be accessed by the dot notation.

e.g `box1.height = 5.0;`

- An instance variable with property can be read by the following method.

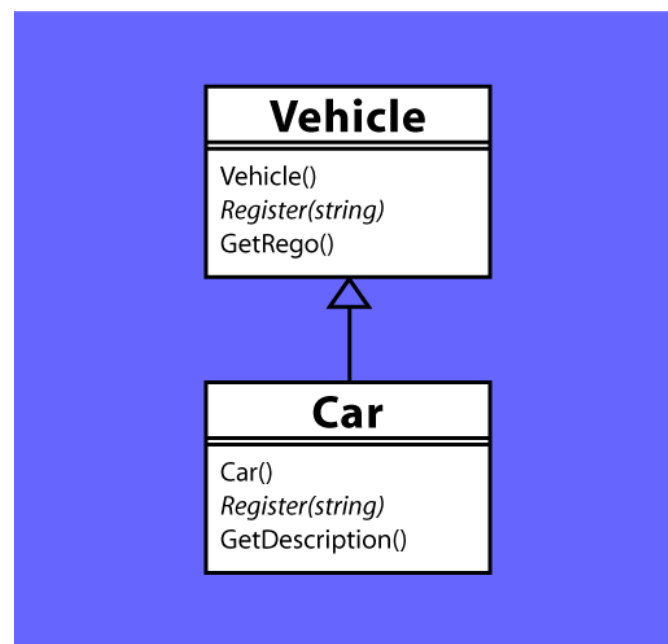
e.g `float myheight = [box1 height];`

- An instance variable with property can be set by the following method.

e.g `[box1 setHeight:10.0];`

Objective-C – (Inheritance)

- Inheritance allows us to define a class in terms of another class which makes it easier to create and maintain an application.



```
@interface derived-class: base-class
```

Objective-C – (Inheritance)

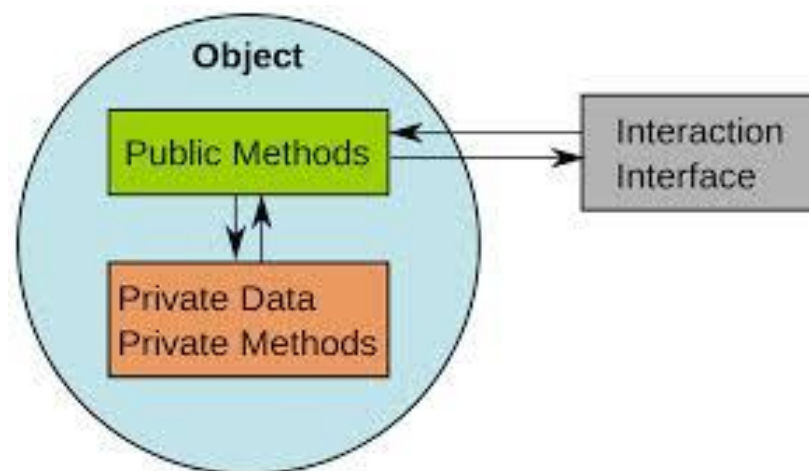
- Inheritance Access Control

A derived class inherits all base class methods and variables with the following exceptions:

- Variables declared in implementation file
- Methods declared in implementation file
- In case the inherited class implements the method in base class, then the method in derived class is executed

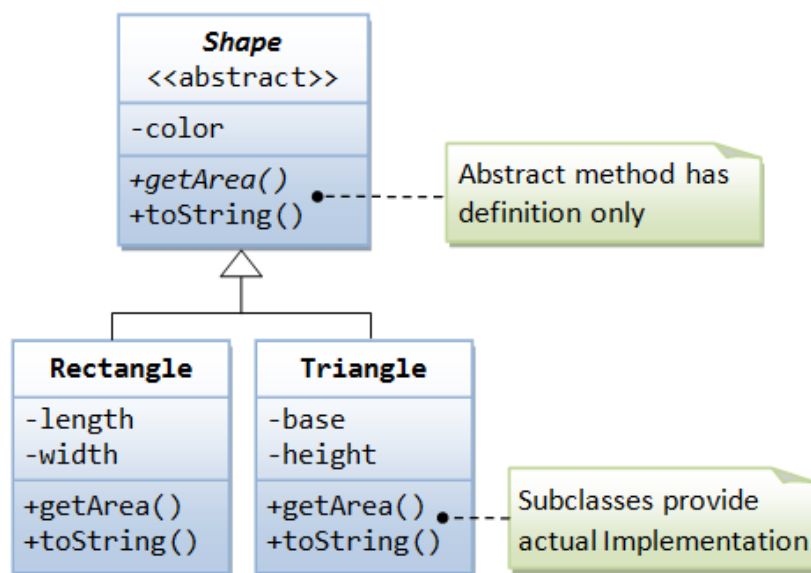
Objective-C – (Encapsulation)

- Encapsulation is a concept that binds together the data and functions that manipulate the data and that keeps both safe from outside interference and misuse
- Objective-C supports the properties of encapsulation and data hiding through the creation of the **classes**.



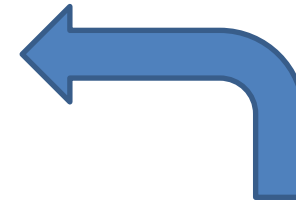
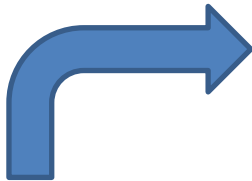
Objective-C – (Polymorphism)

- **Polymorphism** means having “many forms”. Typically, polymorphism occurs when there is a hierarchy of classes and they are related by inheritance.
- Objective-C polymorphism means that a call to a member function will cause a different function to be executed depending on the type of object that invokes the function.



Objective-C – (Polymorphism)

```
@implementation Shape  
  
- (void)printArea{  
    NSLog(@"The area is %f", area);  
}  
  
- (void)calculateArea{  
  
}
```



```
@implementation Square  
  
- (id)initWithSide:(CGFloat)side{  
    length = side;  
    return self;  
}  
  
- (void)calculateArea{  
    area = length * length;  
}
```

```
@implementation Rectangle  
  
- (id)initWithLength:(CGFloat)rLength andBreadth:(CGFloat)rBreadth{  
    length = rLength;  
    breadth = rBreadth;  
    return self;  
}  
  
- (void)calculateArea{  
    area = length * breadth;  
}  
  
@end
```

```
Shape *square = [[Square alloc] initWithSide:10.0];  
[square calculateArea];  
[square printArea];  
Shape *rect = [[Rectangle alloc]  
initWithLength:10.0 andBreadth:5.0];  
[rect calculateArea];  
[rect printArea];
```

Polymorphism happens
here

Objective-C – (Protocol)

- Objective-C allows you to define protocols, which declare the methods expected to be used for a particular situation. Protocols are implemented in the classes conforming to the protocol.

```
@protocol ProtocolName
@required
// list of required methods
@optional
// list of optional methods
@end
```

```
@interface MyClass : NSObject <MyProtocol>
...
@end
```

Objective-C – (Protocol)

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

@protocol PrintProtocolDelegate

- (void)processCompleted;

@end

@interface PrintClass :NSObject
{
    id delegate;
}

- (void) printDetails;
- (void) setDelegate:(id)newDelegate;
@end

@implementation PrintClass

- (void)printDetails{
    NSLog(@"Printing Details");
    [delegate processCompleted];
}

- (void) setDelegate:(id)newDelegate{
    delegate = newDelegate;
}

@end
```

In the example we have seen how the delegate methods are called and executed. It starts with startAction, once the process is completed, the delegate method processCompleted is called to intimate the operation is completed.

Objective-C – (Protocol)

```
@interface SampleClass:NSObject<PrintProtocolDelegate>

- (void)startAction;

@end

@implementation SampleClass

- (void)startAction{
    PrintClass *printClass = [[PrintClass alloc]init];
    [printClass setDelegate:self];
    [printClass printDetails];
}

- (void)processCompleted{
    NSLog(@"Printing Process Completed");
}

@end

int main(int argc, const char * argv[])
{
    NSAutoreleasePool * pool = [[NSAutoreleasePool alloc] init];
    SampleClass *sampleClass = [[SampleClass alloc]init];
    [sampleClass startAction];
    [pool drain];
    return 0;
}
```

Objective-C – (Category)

- By using category, one can add new functions to an existing class.

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

e.g

```
@interface NSString (MyAdditions)  
  
+(NSString *)getCopyrightString;  
  
@end
```

New Method is added to
NSString class

```
NSString *copyrightString = [NSString getCopyrightString];
```

New method is called

Frameworks

- A framework is a collection of predefined classes and methods.
- Mac OS X has provided over 80 frameworks for developers
- 3 frameworks used in every app are
 - Foundation framework
 - Application Kit framework
 - UI Kit framework

Foundation Framework

- In this framework, we can find the following popular classes
 - NSNumber,
 - NSString, NSMutableString
 - NSArray, NSMutableArray
 - NSDictionary, NSMutableDictionary
 - NSFileManager and more

NSNumber – Methods

- Popular Methods

1	+ (NSNumber *)numberWithBool:(BOOL)value Creates and returns an NSNumber object containing a given value, treating it as a BOOL.
2	+ (NSNumber *)numberWithChar:(char)value Creates and returns an NSNumber object containing a given value, treating it as a signed char.
3	+ (NSNumber *)numberWithDouble:(double)value Creates and returns an NSNumber object containing a given value, treating it as a double.
4	+ (NSNumber *)numberWithFloat:(float)value Creates and returns an NSNumber object containing a given value, treating it as a float.
5	+ (NSNumber *)numberWithInt:(int)value Creates and returns an NSNumber object containing a given value, treating it as a signed int.
6	+ (NSNumber *)numberWithInteger:(NSInteger)value Creates and returns an NSNumber object containing a given value, treating it as an NSInteger.

NSNumber – Methods

7	- (BOOL)boolValue Returns the receiver's value as a BOOL.
8	- (char)charValue Returns the receiver's value as a char.
9	- (double)doubleValue Returns the receiver's value as a double.
10	- (float)floatValue Returns the receiver's value as a float.
11	- (NSInteger)integerValue Returns the receiver's value as an NSInteger.
12	- (int)intValue Returns the receiver's value as an int.
13	- (NSString *)stringValue Returns the receiver's value as a human-readable string.

NSNumber – Methods

Methods	Description
-(BOOL) isEqualToNumber: (NSNumber *) aNumber	Return YES if equal to aNumber Return NO if not equal to aNumber
-(NSComparisonResult) compare: (NSNumber *) aNumber	Return NSOrderedAscending if < aNumber Return NSOrderedSame if = aNumber Return NSOrderedDescending if > aNumber

Return Name	Value Represented
NSOrderedAscending	-1
NSOrderedSame	0
NSOrderedDescending	1

NSNumber - Example

```
NSNumber *n1 = [NSNumber numberWithInt:100];
```

```
NSNumber *n2 = [NSNumber numberWithChar: 'b'];
```

```
NSNumber *n3 = [NSNumber numberWithDouble:200.3];
```

```
NSNumber *n4 = [NSNumber numberWithFloat:100.5];
```

```
NSNumber *n5 = [NSNumber numberWithFloat:300.6];
```

```
float f1 = [n4 floatValue];
```

```
float f2 = [n5 floatValue];
```

```
float result = f1 + f2;
```

```
NSNumber *n6 = [NSNumber numberWithFloat:result];
```

```
NSLog(@"The final out is %i, %c, %f and %@",[n1 intValue], [n2 charValue], [n3 doubleValue], [n6 stringValue]);
```

NSNumber - Example

```
NSNumber *a1 = [[NSNumber alloc] initWithInt:900];  
NSNumber *a2 = [[NSNumber alloc] initWithInt:1000];  
if([a1 isEqualToNumber:a2] == YES) {  
    NSLog(@"a1 is larger");  
} else {  
    NSLog(@"a1 is smaller");  
}  
NSComparisonResult c = [a1 compare:a2];  
if(c == NSOrderedAscending)  
    NSLog(@"a1 is larger than a2");  
else if(c == NSOrderedDescending)  
    NSLog(@"a1 is smaller than a2");  
else  
    NSLog(@"a1 is equal to a2");
```

NSString - Methods

1	- (NSString *)capitalizedString; Returns a capitalized representation of the receiver.
2	- (unichar)characterAtIndex:(NSUInteger)index; Returns the character at a given array position.
3	- (double)doubleValue; Returns the floating-point value of the receiver's text as a double.
4	- (float)floatValue; Returns the floating-point value of the receiver's text as a float.
5	- (BOOL)hasPrefix:(NSString *)aString; Returns a Boolean value that indicates whether a given string matches the beginning characters of the receiver.
6	- (BOOL)hasSuffix:(NSString *)aString; Returns a Boolean value that indicates whether a given string matches the ending characters of the receiver.
7	- (id)initWithFormat:(NSString *)format ...; Returns an NSString object initialized by using a given format string as a template into which the remaining argument values are substituted.

NSString - Methods

8	- (NSInteger)integerValue; Returns the NSInteger value of the receiver's text.
9	- (BOOL)isEqualToString:(NSString *)aString; Returns a Boolean value that indicates whether a given string is equal to the receiver using a literal Unicode-based comparison.
10	- (NSUInteger)length; Returns the number of Unicode characters in the receiver.
11	- (NSString *)lowercaseString; Returns lowercased representation of the receiver.
12	- (NSRange)rangeOfString:(NSString *)aString; Finds and returns the range of the first occurrence of a given string within the receiver.
13	- (NSString *)stringByAppendingFormat:(NSString *)format ...; Returns a string made by appending to the receiver a string constructed from a given format string and the following arguments.
14	- (NSString *)stringByTrimmingCharactersInSet:(NSCharacterSet *)set; Returns a new string made by removing from both ends of the receiver characters contained in a given character set.
15	- (NSString *)substringFromIndex:(NSUInteger)anIndex ; Returns a new string containing the characters of the receiver from the one at a given index to the end.

NSMutableString - Methods

Methods	Description
<code>+(id) stringWithCapacity:(NSUInteger)capacity</code>	Create a string which can store capacity number of characters
<code>-(void)appendString(NSString *)aString</code>	Append aString behind
<code>-(void)deleteCharactersInRange:(NSRange)aRange</code>	Delete part of the string specified by NSRange
<code>-(void)insertString(NSString *)aString atIndex:(NSUInteger) anIndex</code>	Insert string into existing string at specified index.

NSString - Example

```
NSString *s1 = @"Apple iPhone";  
  
s1 = [s1 stringByAppendingString:@" and iOS"];  
  
NSLog(@"%c", [s1 characterAtIndex:3]);  
  
NSLog(@"%@@", [s1 substringWithRange:NSMakeRange(5,10)]);  
  
NSString *s4 = @"Apple";  
  
if([s1 isEqualToString:s4] == YES) {  
    NSLog(@"The same");  
}  
  
NSLog(@"%@@", [s4 uppercaseString]);
```

NSString - Example

```
int s2 = 100;

NSString *s3 = [NSString stringWithFormat:@"%i", s2];

double d1;

d1 = [s3 doubleValue];

NSMutableString *s5 = [NSMutableString stringWithCapacity:10];

[s5 appendString:@"iPhone 6"];

[s5 deleteCharactersInRange:NSMakeRange(2,3) ];

[s5 insertString:@" I like it" atIndex: 2];

NSLog(@"%@ ",s5) ;
```

NSArray - Methods

Methods	Description
<code>+(id) arrayWithObjects(id)firstObj,</code>	Create an array putting objects inside, end with a nil. e.g. str1, str2, str3, nil
<code>-(NSUInteger) count</code>	Return the length of the array
<code>-(NSUInteger) indexOfObject(id)anObject</code>	If the array contains anObject, return its index. Otherwise, return NSNotFound
<code>-(id)objectAtIndex(NSUInteger) index</code>	Return the object at the index specified
<code>-(void) makeObjectsPerformSelector: (SEL)aSelector</code>	Request all the objects inside the array to perform the aSelector methods.

NSMutableArray - Methods

Methods	Description
-(void)addObject:(id) anObject	Add anObject into the array
-(void)removeObject: (id) anObject	Remove anObject from the array
-(void)sortUsingSelector: (SEL) comparator	Using the comparator method to sort the array
-(void)replaceObjectAtIndex:(NSUInteger) index withObject: (id) anObject	Replace the object with anObject at the specified index position
-(void)exchangeObjectAtIndex:(NSUInteger) idx1 withObjectAtIndex:(NSUInteger) idx2	Exchange the 2 objects the idx1 and idx2

NSArray - Examples

```
NSString *k1 = @"Peter";
NSString *k2 = @"Mary";

NSArray *names = [NSArray arrayWithObjects: k1, k2, nil];

int count = [names count];

for(int i=0; i<count; i++) {

    NSLog(@"Index: %i, value: %@", i, [names objectAtIndex: i]);

}

NSString *find = @"Mary";
NSInteger index;
if((index = [names indexOfObject::find]) != NSNotFound)

    NSLog(@"Found");

else
    NSLog(@"Not Found");
```

NSArray - Examples

```
NSMutableArray *students = [[NSMutableArray alloc] init];
```

```
NSString *k3 = @"John";
```

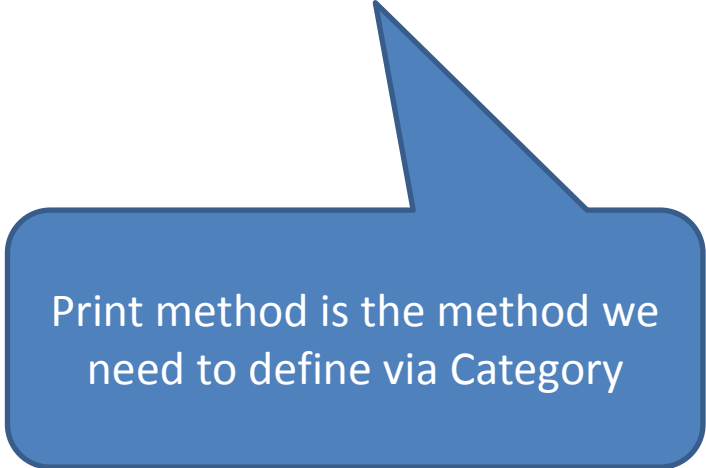
```
NSString *k4 = @"Sam";
```

```
[students addObject:k3];
```

```
[students addObject:k4];
```

```
[students removeObject:k3];
```

```
[students makeObjectsPerformSelector:@selector(print)];
```



Print method is the method we need to define via Category

NSDictionary - Methods

Methods	Description
<code>+(id)dictionaryWithObjects:(NSArray *) objects forKeys:(NSArray *)keys</code>	Create a dictionary object with objects and keys as a pair
<code>-(NSUInteger) count</code>	Return the number of objects in the dictionary
<code>-(id) objectForKey: (id) aKey</code>	Return the object for the key aKey

NSMutableDictionary - Methods

Methods	Description
<code>+(id) dictionaryWithCapacity:(NSUInteger) numItems</code>	Create a dictionary object with numItems capacity
<code>-(id) initWithCapacity:(NSUInteger) numItems</code>	Initialize a dictionary object with numItems capacity
<code>-(void) setObject:(id)anObject forKey:(id) aKey</code>	Put anObject into the dictionary with aKey as the identifier
<code>-(void) removeObjectForKey:(id) aKey</code>	Remove the object identified by the aKey

NSDictionary - Examples

```
NSArray *key = [NSArray arrayWithObjects:@"Peter",@"John",@"Mary",nil];

NSArray *object = [NSArray arrayWithObjects:@"He is tall",@"He is fat",@"She is
    thin",nil];

NSDictionary *dic = [NSDictionary dictionaryWithObject:object forKey:key];

NSLog(@"The count is %i", [dic count]);

for(NSString *temp in dic)
    NSLog(@"%@ : %@", temp, [dic objectForKey:temp]);
```