

Unit -8

Introduction to iOS Programming [8 Hrs]

Introduction to iOS and iOS Programming

iOS (formerly iPhone OS) is a mobile operating system created and developed by Apple Inc. exclusively for its hardware. It is the operating system that presently powers many of the company's mobile devices, including the iPhone, and iPod Touch; it also powered the iPad prior to the introduction of iPadOS in 2019. It is the second most popular mobile operating system globally after Android.

Originally unveiled in 2007 for the iPhone, iOS has been extended to support other Apple devices such as the iPod Touch (September 2007) and the iPad (January 2010). As of March 2018, Apple's App Store contains more than 2.1 million iOS applications, 1 million of which are native for iPads. These mobile apps have collectively been downloaded more than 130 billion times.

The iOS user interface is based upon direct manipulation, using multi-touch gestures. Interface control elements consist of sliders, switches, and buttons. Interaction with the OS includes gestures such as swipe, tap, pinch, and reverse pinch, all of which have specific definitions within the context of the iOS operating system and its multi-touch interface. Internal accelerometers are used by some applications to respond to shaking the device (one common result is the undo command) or rotating it in three dimensions (one common result is switching between portrait and landscape mode). Apple has been significantly praised for incorporating thorough accessibility functions into iOS, enabling users with vision and hearing disabilities to properly use its products.

Major versions of iOS are released annually. On all recent iOS devices, iOS regularly checks on the availability of an update, and if one is available, will prompt the user to permit its automatic installation. The current version, iOS 13 was released to the public on 19 September 2019, introducing user interface tweaks and a dark mode, along with features such as a redesigned Reminders app, a swipe keyboard, and an enhanced Photos app. iOS 13 does not support devices with less than 2 GB of RAM, including the iPhone 5s, iPod Touch (6th generation), and the iPhone 6 and iPhone 6 Plus, which still make up over 10% of all iOS devices. iOS 13 is exclusively for the iPhone and iPod touch as the iPad variant is now called iPadOS.

iOS programming is a set of processes and procedures involved in writing software for iPhone developed by Apple Inc.

iOS Platform

The iOS SDK (Software Development Kit) allows for the development of mobile apps on iOS. While originally developing iPhone prior to its unveiling in 2007, Apple's then-CEO Steve Jobs did not intend to let third-party developers build native apps for iOS, instead directing them to make web applications for the Safari web browser. However, backlash from developers prompted the company to reconsider with Jobs announcing in October

2007 that Apple would have a software development kit available for developers by February 2008. The SDK was released on March 6, 2008.

The SDK is a free download for users of Mac personal computers. It is not available for Microsoft Windows PCs. The SDK contains sets giving developers access to various functions and services of iOS devices, such as hardware and software attributes. It also contains an iPhone simulator to mimic the look and feel of the device on the computer while developing. New versions of the SDK accompany new versions of iOS. In order to test applications, get technical support, and distribute apps through App Store, developers are required to subscribe to the Apple Developer Program.

Combined with Xcode, the iOS SDK helps developers write iOS apps using officially supported programming languages, including Swift and Objective-C. Other companies have also created tools that allow for the development of native iOS apps using their respective programming languages.

The history of Apple iOS versions is discussed below:

- Apple iOS was originally known as iPhone OS. The company released three versions of the mobile OS under that name before iOS 4 debuted in June 2010.
- On Oct. 12, 2011, Apple released iOS 5, which expanded the number of available applications to over 500,000. This iOS version also added the Notification Center, a camera app, Siri and more.
- Unveiled on June 11, 2012, iOS 6 included a Maps application and the Passbook ticket storage and loyalty program application.
- Released on Sept. 18, 2013, iOS 7 featured an entirely redesigned user interface. In September 2014, iOS 8 introduced Continuity, a cross-platform system that allows users of multiple Apple devices to pick up on one where they left off from another. Other new features included the Photos app and Apple Music.
- Apple iOS 9 and iOS 10 -- released, respectively, on Sept. 16, 2015, and Sept. 13, 2016 -- featured upgrades such as a revamped notifications section, improved iMessage capabilities and Siri integration with third-party apps.

Environment Setup

For development of iOS applications, we need to install Xcode editor, its apple integrated development environment (IDE). The Xcode editor is the primary development tool for any type of apple platforms like OSX Software's, iPhones App, Mac, etc.

The requirement of the Xcode editor is that we should have minimum Apple Laptop or Mac Computer for development of iOS application. We can download Xcode editor from **Apple website** or **Apple app store** and this editor is completely free we don't need to pay anything for Apple.

Creating an Xcode Project

With Xcode installed, it is time to launch it for the first time. If all went well, you should see the **Welcome to Xcode** window, which contains a few useful links and helps you create a new application. To create your first iOS application, select **Create a new Xcode project** from the list of options.

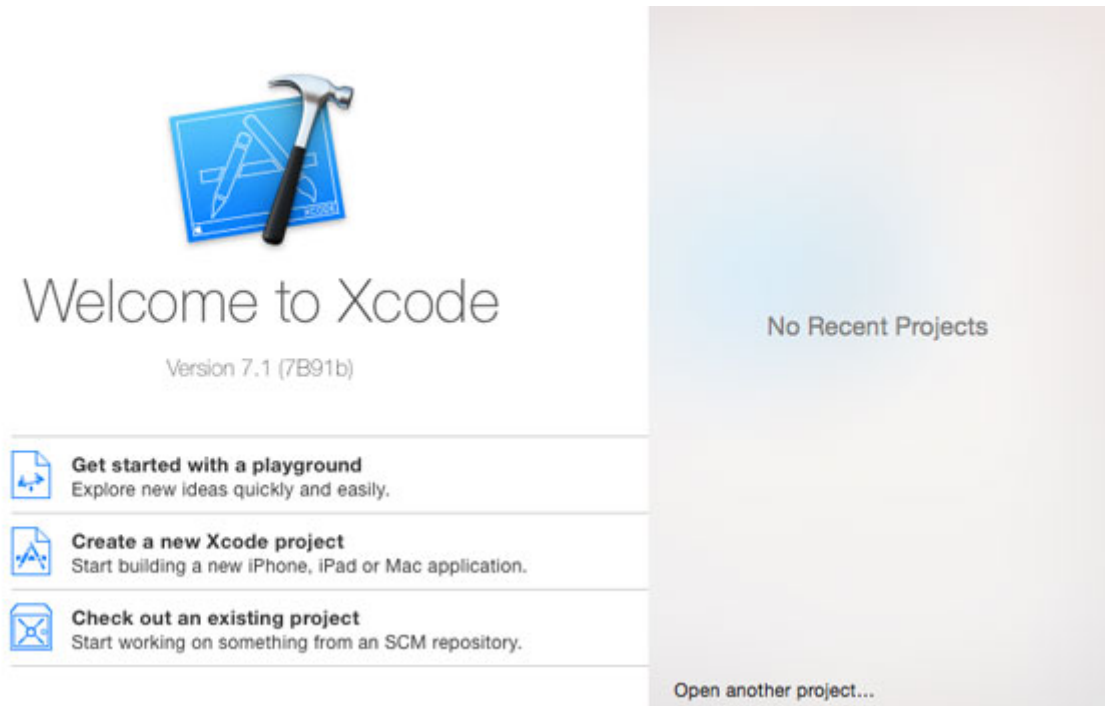


Figure 8-1: Xcode Homepage

Xcode makes it easy to create a new Xcode project by offering a handful of useful project templates. The **Single View Application** template is a good choice for your first application. Select it from the list of **iOS > Application** templates and click **Next**.

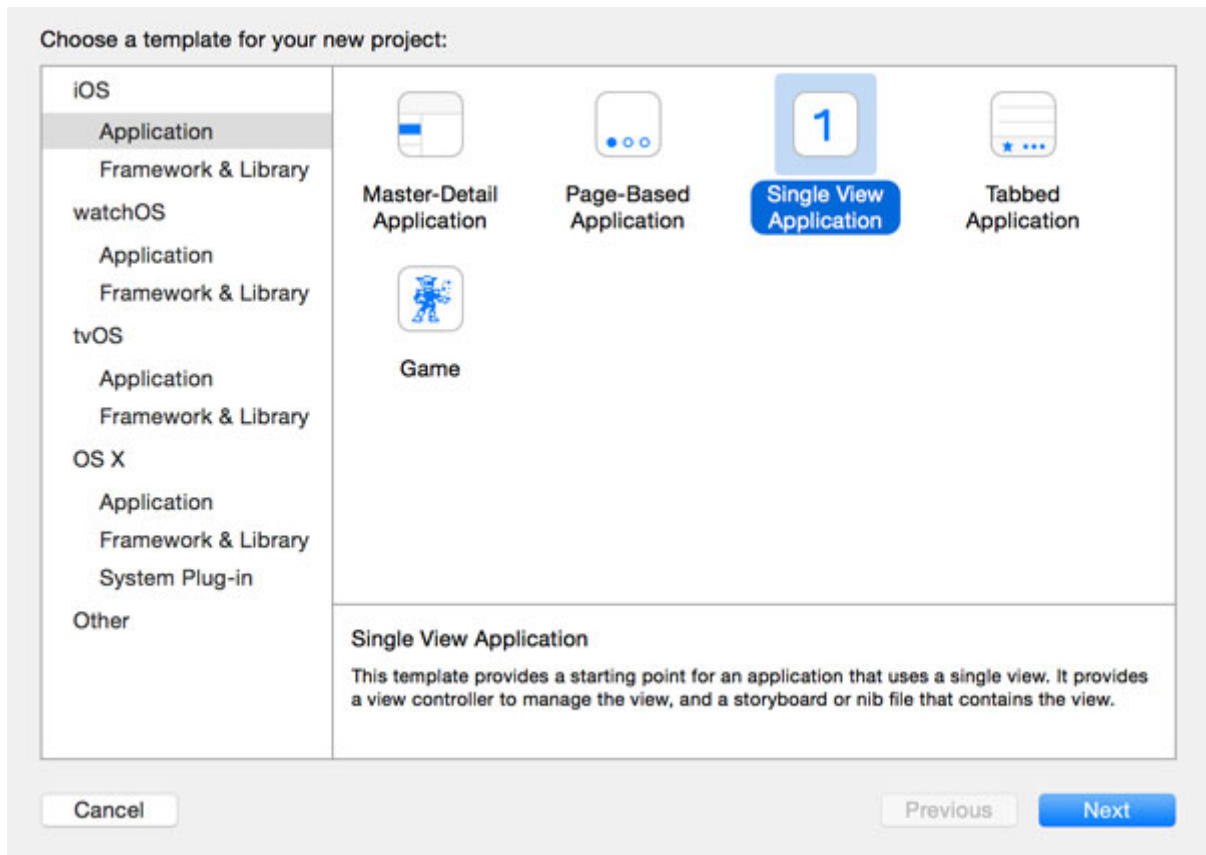


Figure 8-2: Creating a Project

The next window lets you configure your Xcode project. Fill in the fields as shown in the screenshot below and click **Next**.

Choose options for your new project:

Product Name:

Organization Name:

Organization Identifier:

Bundle Identifier:

Language:

Devices:

☐ Use Core Data

☐ Include Unit Tests

☐ Include UI Tests

Figure 8-3: Configuring a new Project

Once we click on Next button new dialog will open in that we need to select the location to save our project. Once you select the location to save project then click on **Create** button like as shown below.

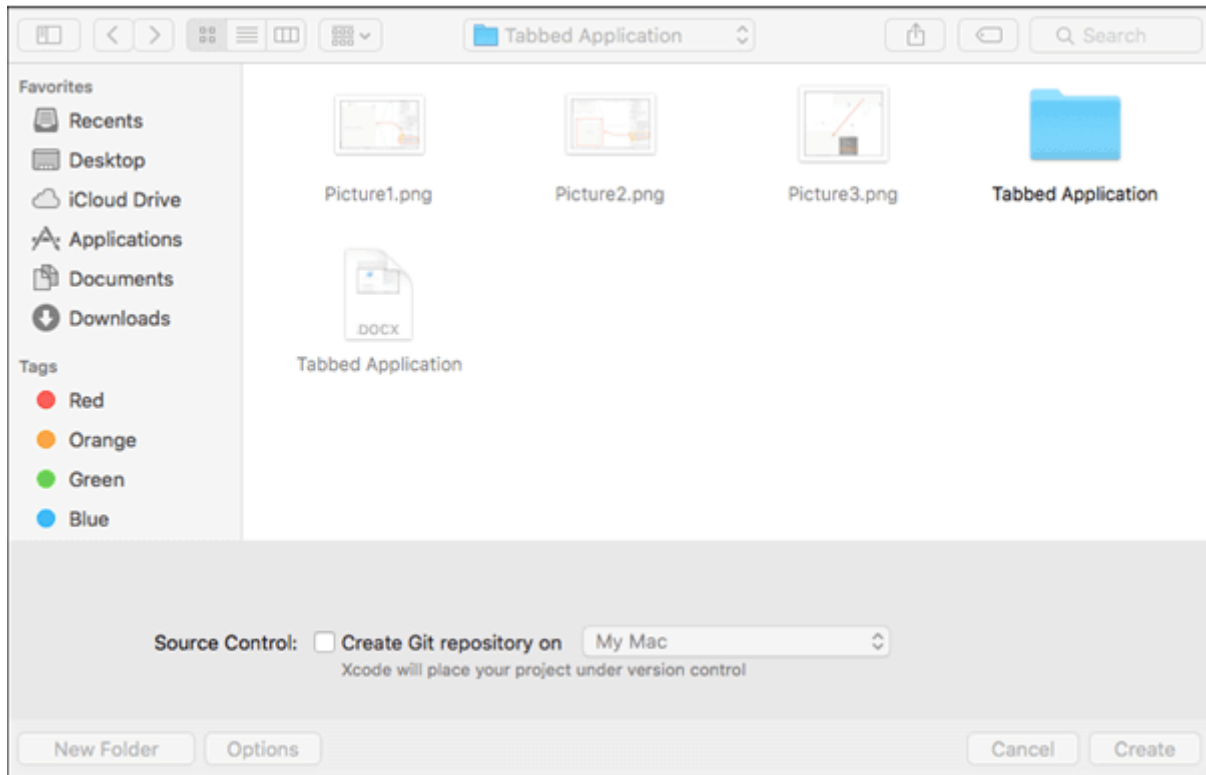


Figure 8-4: Selecting project location

After click on **Create** button the Xcode will create and open a new project. By default, our project structure will be like as shown below.

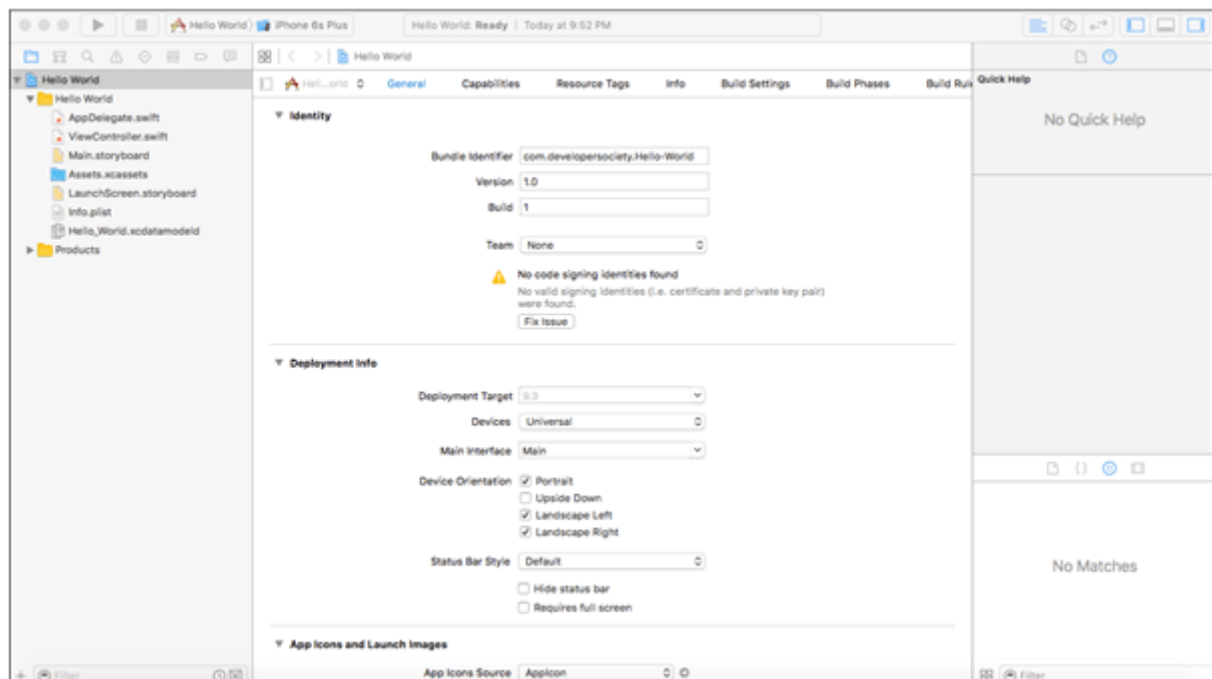


Figure 8-5: Xcode workspace window

Building the Interface

In our project **Main.storyboard** and **ViewController.swift** are the main files which we used to design app user interface and to maintain source code.

- **Main.storyboard** - Its visual interface editor and we will use this file to design our app user interface
- **ViewController.swift** - It contains source code of our application and we use this file to write any code related to our app.

Now in project select **Main.storyboard** file the Xcode will open visual interface editor like as shown below.

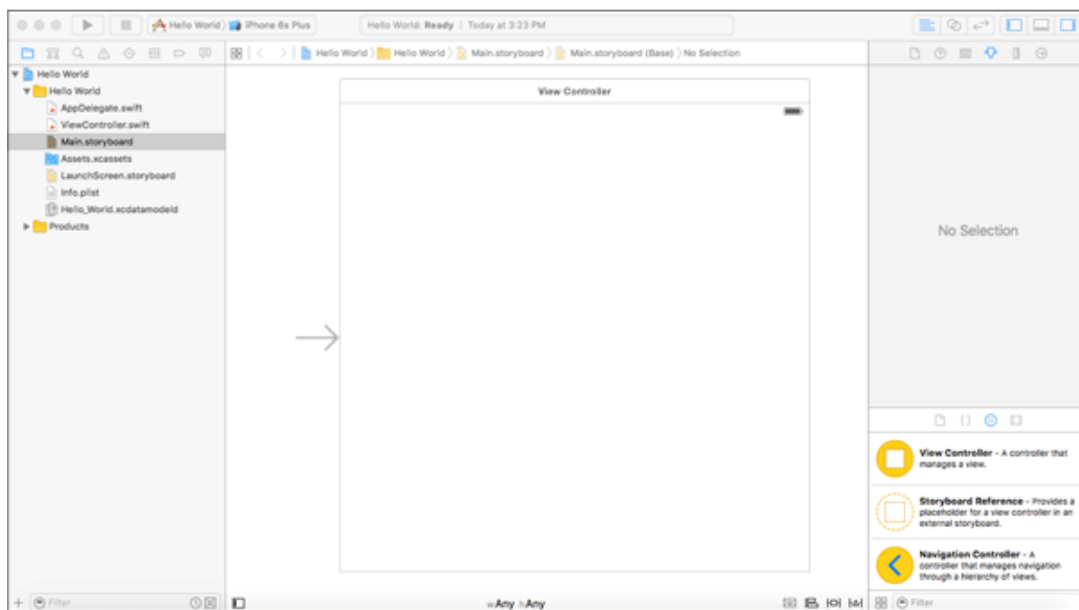


Figure 8-6: First look of Main.storyboard

Now select **ViewController.swift** file in your project that view will be like as shown below.

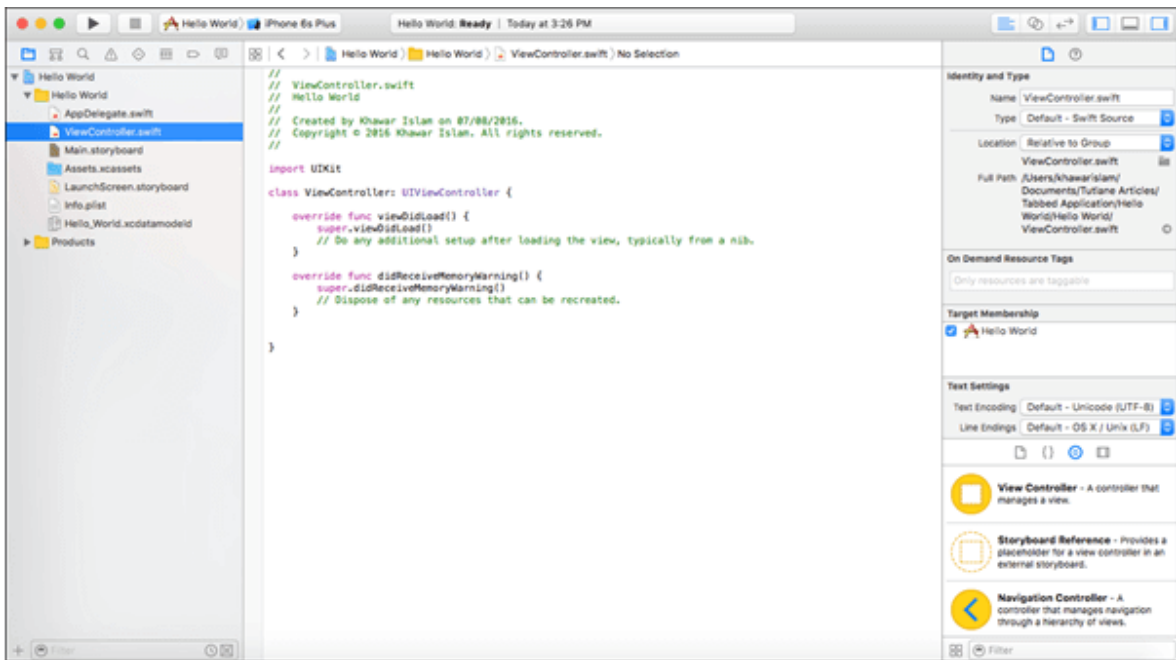


Figure 8-7: First look of ViewController.swift

Now we will add controls to our application for that open Object Library. The Object Library will appear at the bottom of Xcode in right side. In case if you don't find Object library, click on the button which is at the third position from the left in the library selector bar like as shown below. (Alternatively you can choose **View -> Utilities -> Show Object Library**).

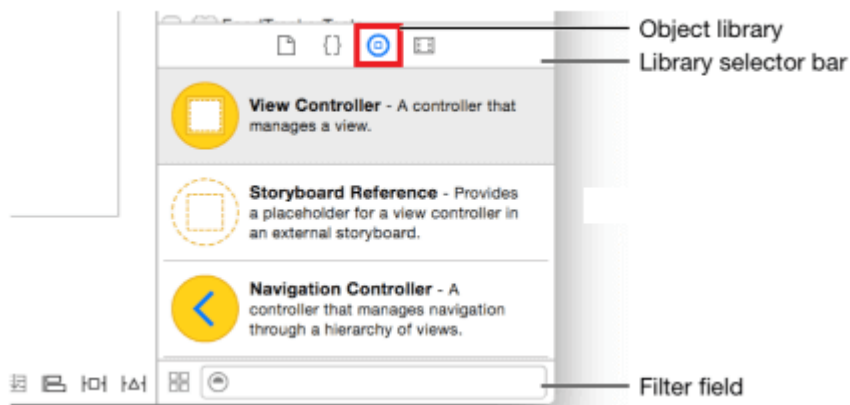


Figure 8-8: Opening object library

As we discussed our user interface will be in Main.storyboard file so open Main.storyboard file. Now in Object library search for the label in Filter field then drag and drop the label into Main.storyboard ViewController like as shown below.

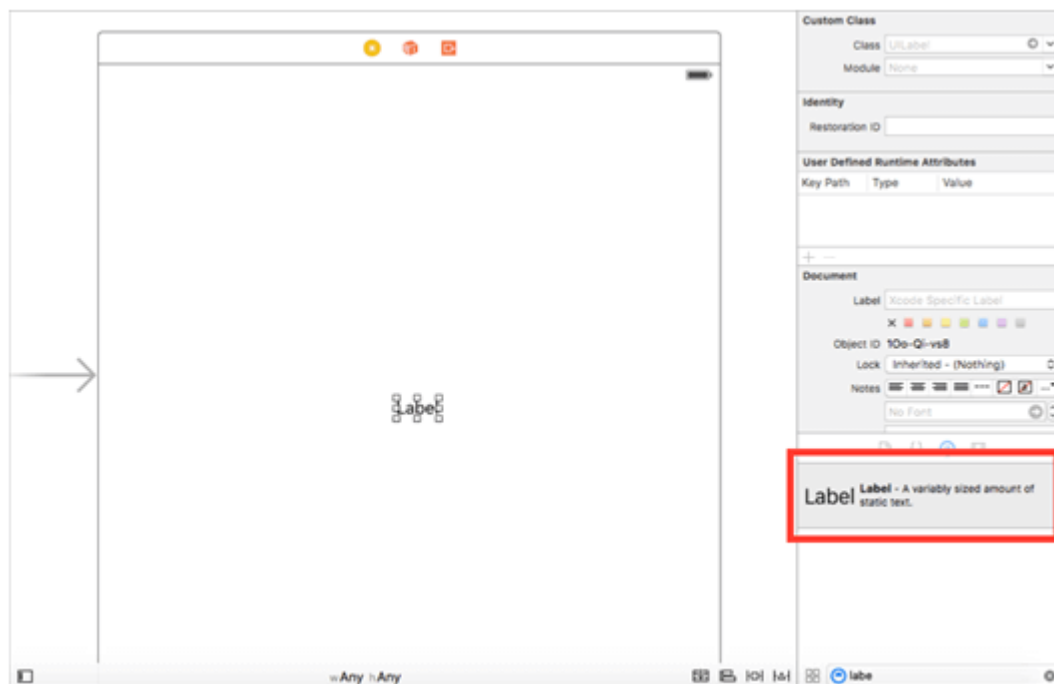


Figure 8-9: Adding label to the project

Now we will change the text of label for that click on label in right side the label properties will open in **Text** property textbox write the “**Hello World**”. Once we change the label text now we will change the position of label control for that select **label** control -> Click on **label** control and just **drag** the **label** control like as shown below.

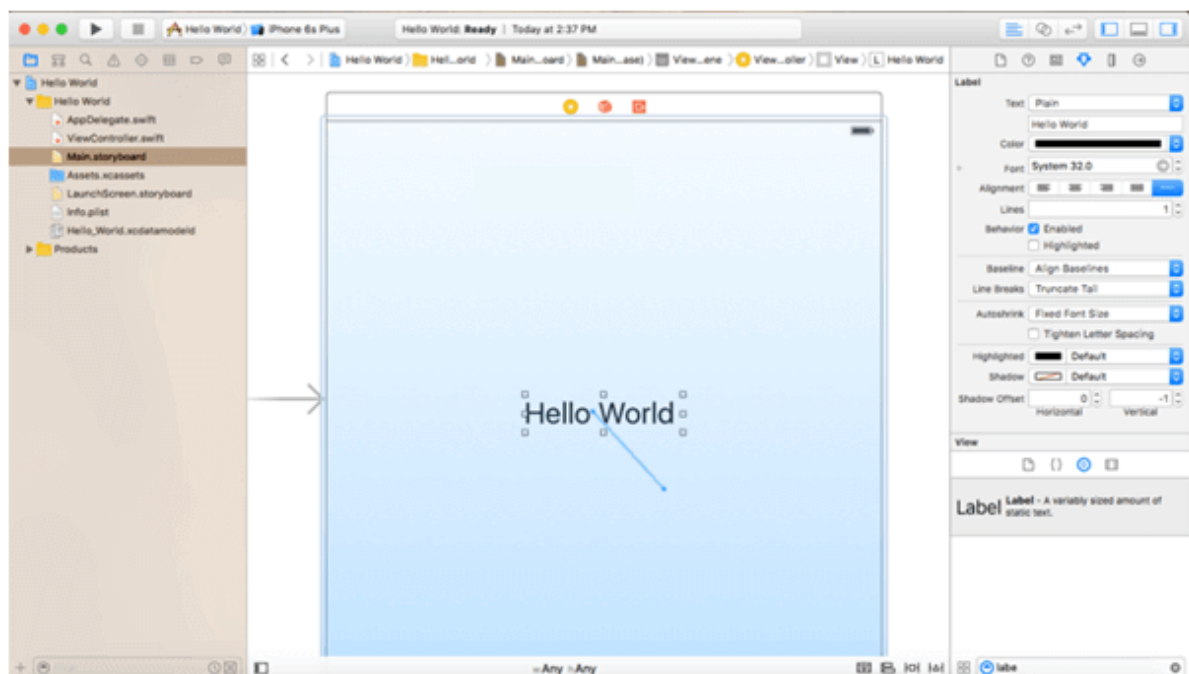


Figure 8-10: Adding text “Hello World” to label

Once we **click** and **drag** the **label** control we can able to see multiple properties like “**Center Horizontally in Container**”, “**Center Vertically in Container**”, etc.

We will set “**Center Horizontally in Container**”, “**Center Vertically in Container**” properties for label control to make our label appeared in centered position. Once we set these two properties we can able to see two lines which appear like cross on our hello world label like as shown below.

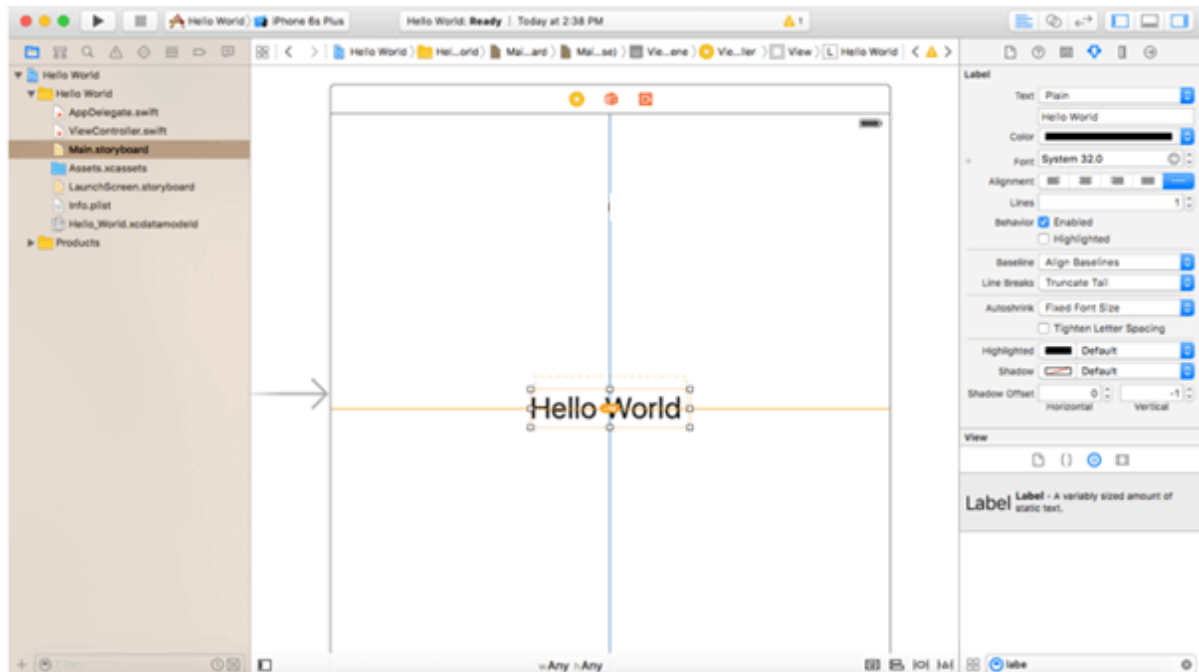


Figure 8-11: Centering the label

Running on the Simulator

To build and run our application we use **Simulator** in Xcode. The **Simulator** will help us to know how our app will look and behave if it is running on device.

The Simulator in Xcode is having different device options to check our app in multiple devices like iPad, iPhone with different screen sizes. By using these options, we can simulate our app in any device and test its design and behavior based on our requirements.

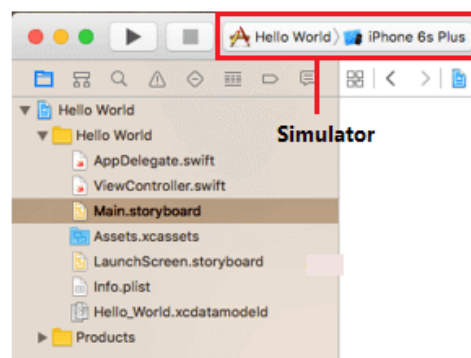


Figure 8-12: Configuring Simulator

Once we select the required simulator then we will run our application by using **Run** button, located in the top-left corner of the Xcode toolbar like as shown below.

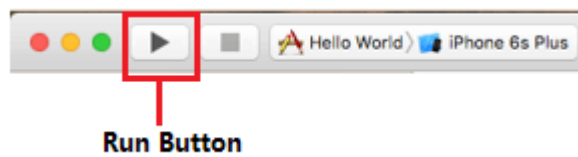


Figure 8-13: Running Application on Simulator

Once we run the iOS hello world application we will get output like as shown below.



Figure 8-14: Output displayed in Simulator

Making Connections

A **connection** lets one object know where another object is in memory so that the two objects can communicate. There are two kinds of connections that you can make in Interface Builder: outlets and actions. An outlet is a reference to an object. An action is a method that gets triggered by a button or some other view that the user can interact with, like a slider or a picker.

For making connections you have to use **iOS Outlets and Actions**. In iOS Outlets and Actions are basically the property symbol of **IBOutlet**s and **IBAction**s where **IB** called **Interface Builder**. Outlet is basically the reference object and action is the method which is used to perform some type of actions.

Following is the syntax of using iOS Actions and Outlets in applications.

```
@IBOutlet weak var labeltxt: UILabel!  
@IBAction func buttonaction(sender: AnyObject) {  
    labeltxt.text = "Hello"  
}
```

If you observe above syntax we used “@IBOutlet” to add reference of label control and we used “@IBAction” to add button click action to change label control text.

To demonstrate connection, let's take an example of an iOS application containing two view controls (one label control and another button control). While clicking on the button control we must be able to change text of label. So, let's begin by adding label and button controls in our storyboard.

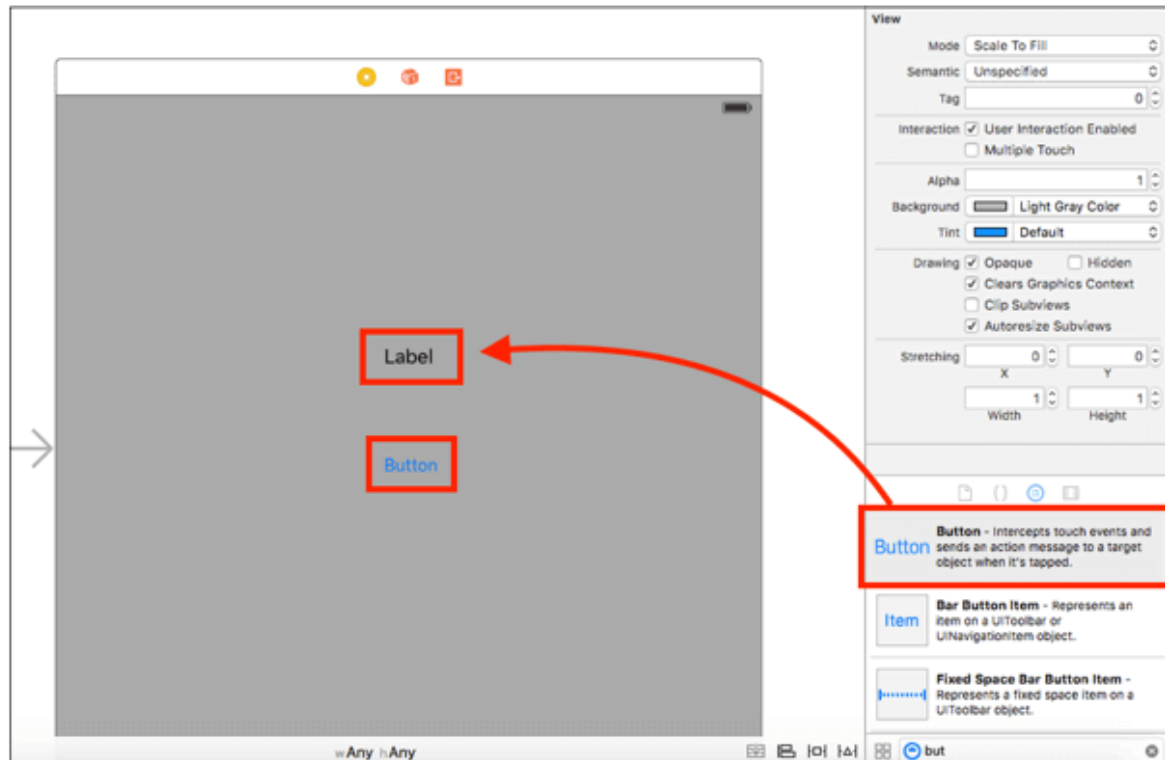


Figure 8-15: Adding label and button

Now we will map our controls to **ViewController.swift** file for that we need to use **Assistant editor** in Xcode. Open the Xcode editor in assistant mode for that click on the **overlap circle** button in Xcode toolbar at top right side like as shown below.

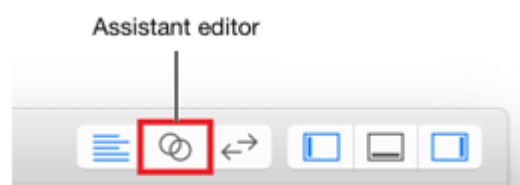


Figure 8-16: Opening Assistant Editor

Now press **Ctrl** button in keyboard and drag the controls from your canvas to the code display in **ViewController.swift** file like as shown below.

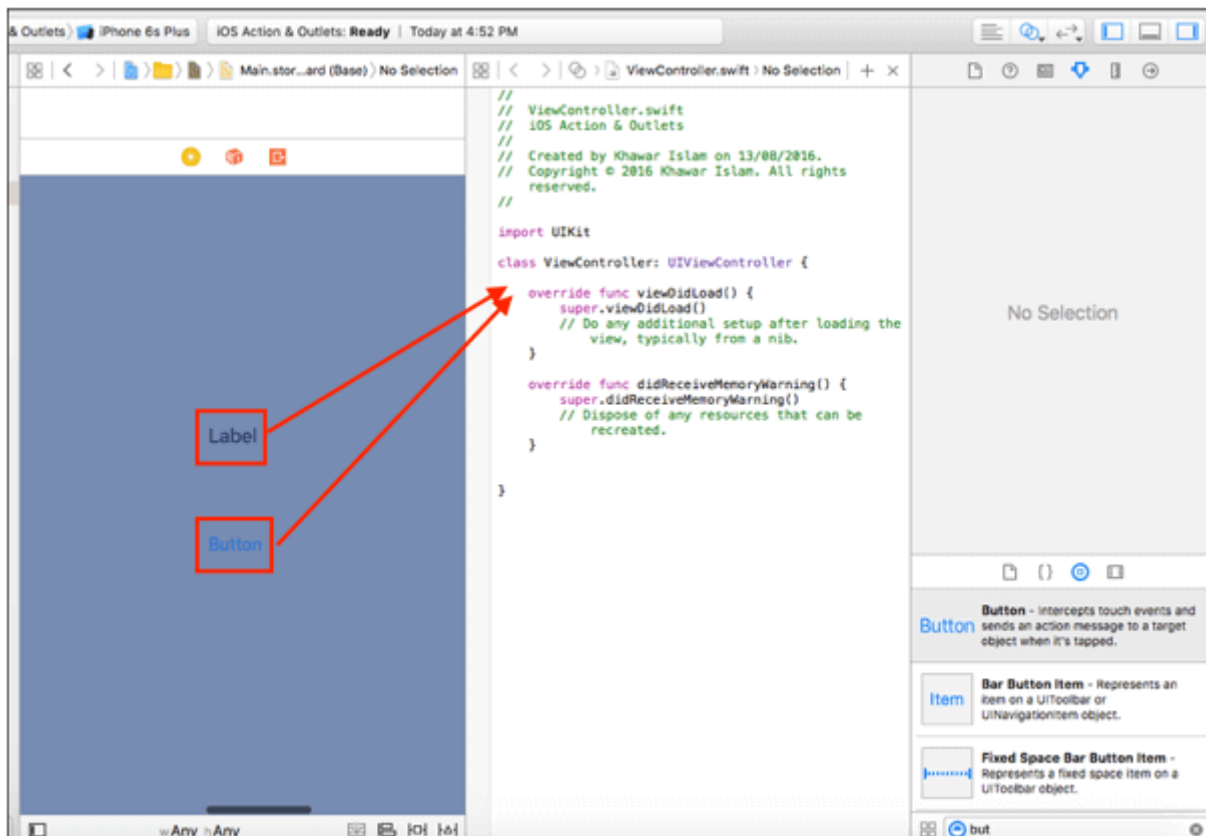


Figure 8-17: Connecting UI controls to application code

Once you drag the controls from viewcontroller to code in **ViewController.swift** file now write code in **@IBAction** to change the label text when click on button. Once we make required changes our **ViewController.swift** file should be like as shown below.

```
import UIKit
class ViewController: UIViewController {
    @IBOutlet weak var labeltxt: UILabel!
    @IBAction func buttonaction(sender: AnyObject) {
        labeltxt.text = "We love BCA "
    }
    override func viewDidLoad() {
        super.viewDidLoad()
        // Do any additional setup after loading the view.
    }
    override func didReceiveMemoryWarning() {
        super.didReceiveMemoryWarning()
        // Dispose of any resources that can be recreated.
    }
}
```

Once we run application click on button it will change the label text. The button performs the action for label to change its outlet text.

Above code produces following output:

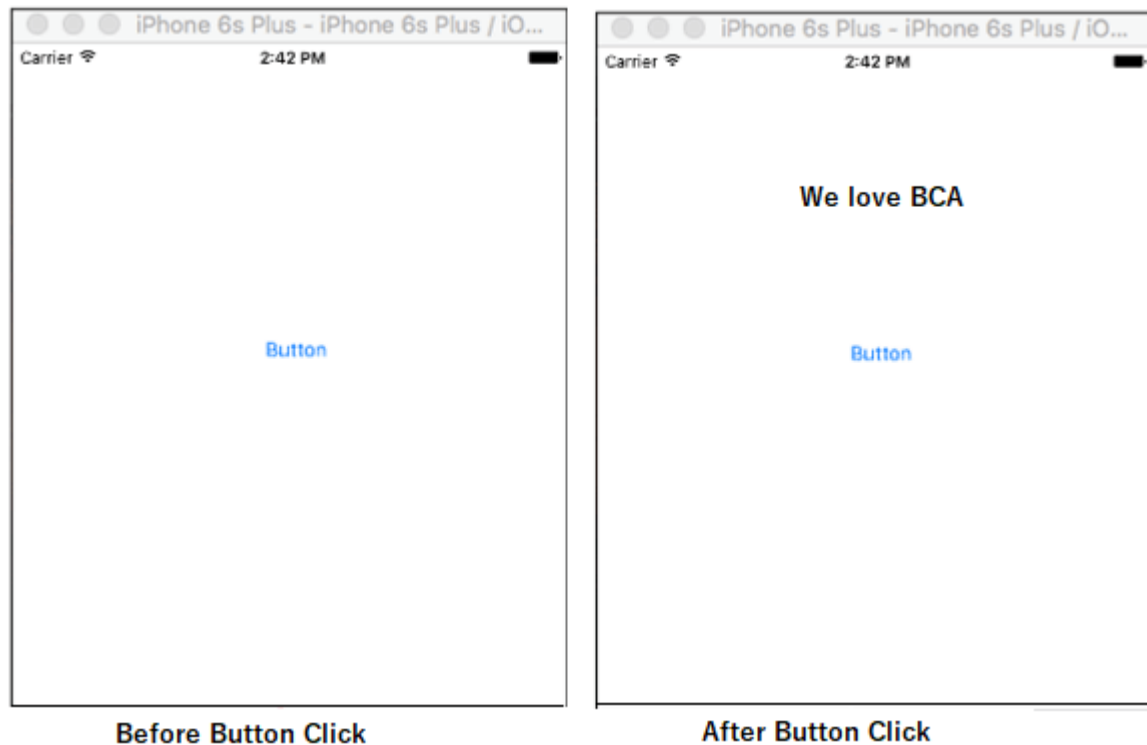


Figure 8-18: Output demonstrating connection

Introduction to Swift Language

Swift is a new language that Apple introduced in 2014. It replaces Objective-C as the recommended development language for iOS and Mac. In this chapter, you are going to focus on the basics of Swift. You will not learn everything, but you will learn enough to get started. Then, as you continue through the book, you will learn more Swift while you learn iOS development.

Swift maintains the expressiveness of Objective-C while introducing a syntax that is safer, succinct, and readable. It emphasizes type safety and adds advanced features such as optionals, generics, and sophisticated structures and enumerations. Most importantly, Swift allows the use of these new features while relying on the same tested, elegant iOS frameworks that developers have built upon for years.

Types in Swift

Swift types can be arranged into three basic groups: structures, classes, and enumerations. All three can have:

- **properties** – values associated with a type
- **initializers** – code that initializes an instance of a type
- **instance methods** – functions specific to a type that can be called on an instance of that type
- **class or static methods** – functions specific to a type that can be called on the type itself

Structures <pre> struct MyStruct { // properties // initializers // methods } </pre>	Enumerations <pre> enum MyEnum { // properties // initializers // methods } </pre>	Classes <pre> class MyClass: SuperClass { // properties // initializers // methods } </pre>
--	--	---

c

Built-in Data Types

The following types of basic data types are most frequently when declaring variables –

- **Int or UInt** – This is used for whole numbers. More specifically, you can use Int32, Int64 to define 32 or 64-bit signed integer, whereas UInt32 or UInt64 to define 32 or 64-bit unsigned integer variables. For example, 42 and -23.
- **Float** – This is used to represent a 32-bit floating-point number and numbers with smaller decimal points. For example, 3.14159, 0.1, and -273.158.
- **Double** – This is used to represent a 64-bit floating-point number and used when floating-point values must be very large. For example, 3.14159, 0.1, and -273.158.
- **Bool** – This represents a Boolean value which is either true or false.
- **String** – This is an ordered collection of characters. For example, "Hello, World!"
- **Character** – This is a single-character string literal. For example, "C"
- **Optional** – This represents a variable that can hold either a value or no value.
- **Tuples** – This is used to group multiple values in single Compound Value.

Variables

You must declare them using **var** keyword as follows –

```
var variableName = <initial value>
```

You can provide a **type annotation** when you declare a variable, to be clear about the kind of values the variable can store. Here is the syntax –

```
var variableName:<data type> = <optional initial value>
```

```

var varA = 42
print(varA)

var varB:Float

varB = 3.14159
print(varB)

```

When we run the above program using playground, we get the following result –

```

42
3.1415901184082

```

Optionals

Swift also introduces **Optionals** type, which handles the absence of a value. Optionals say either "there is a value, and it equals x" or "there isn't a value at all".

Here's an optional Integer declaration –

```
var perhapsInt: Int?
```

Here's an optional String declaration –

```
var perhapsStr: String?
```

The above declaration is equivalent to explicitly initializing it to **nil** which means no value –

```
var perhapsStr: String? = nil
```

Let's take the following example to understand how optionals work-

```
var myString:String? = nil

if myString != nil {
    print(myString)
} else {
    print("myString has nil value")
}
```

When we run the above program using playground, we get the following result –

```
myString has nil value
```

Constants

Constants refer to fixed values that a program may not alter during its execution. Constants can be of any of the basic data types like an *integer constant*, a *floating constant*, a *character constant*, or a *string literal*. There are *enumeration constants* as well.

Constants are treated just like regular variables except the fact that their values cannot be modified after their definition.

Before you use constants, you must declare them using **let** keyword as follows –

```
let constantName = <initial value>
```

Following is a simple example to show how to declare a constant-

```
let constA = 42
print(constA)
```

When we run the above program using playground, we get the following result –

```
42
```

Operators

An operator is a symbol that tells the compiler to perform specific mathematical or logical manipulations. Swift is rich in built-in operators and provides the following types of operators –

- Arithmetic Operators
- Comparison Operators
- Logical Operators
- Bitwise Operators
- Assignment Operators
- Range Operators

- Misc Operators

Arithmetic Operators

The following table shows all the arithmetic operators supported by Swift language. Assume variable **A** holds 10 and variable **B** holds 20, then –

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/float division	B % A will give 0

Comparison Operators

The following table shows all the relational operators supported by Swift language. Assume variable **A** holds 10 and variable **B** holds 20, then –

Operator	Description	Example
==	Checks if the values of two operands are equal or not; if yes, then the 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 the 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 the 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 the 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 the 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 the condition becomes true.	(A <= B) is true.

Logical Operators

The following table shows all the logical operators supported by Swift language. Assume variable **A** holds 1 and variable **B** holds 0, then –

Operator	Description	Example
&&	Called Logical AND operator. If both the operands are non-zero, then the condition becomes true.	(A && B) is false.
	Called Logical OR Operator. If any of the two operands is non-zero, then the 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 the Logical NOT operator will make it false.	!(A && B) is true.
---	--	--------------------

Bitwise Operators

Bitwise operators work on bits and perform bit by bit operation. The truth tables for &, |, and ^ are as follows –

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

Assume A = 60; and B = 13;

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

Bitwise operators supported by Swift language are listed in the following table. Assume variable **A** holds 60 and variable **B** holds 13, then 7–

Operator	Description	Example
&	Binary AND Operator copies a bit to the result, if it exists in both operands.	(A & B) will give 12, which is 0000 1100
	Binary OR Operator copies a bit, if it exists in either operand.	(A B) will give 61, which is 0011 1101
^	Binary XOR Operator copies the bit, if it is set in one operand but not both.	(A ^ B) will give 49, which is 0011 0001
~	Binary Ones Complement Operator is unary and has the effect of 'flipping' bits.	(~A) will give -61, which is 1100 0011 in 2's complement form.
<<	Binary Left Shift Operator. The left operands value is moved left by the number of bits specified by the right operand.	(A << 2 will give 240, which is 1111 0000

>>	Binary Right Shift Operator. The left operands value is moved right by the number of bits specified by the right operand.	A >> 2 will give 15, which is 0000 1111
----	---	---

Assignment Operators

Swift supports the following 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
<<=	Left shift AND assignment operator	C <<= 2 is same as C = C << 2
>>=	Right shift AND assignment operator	C >>= 2 is same as C = C >> 2
&=	Bitwise AND assignment operator	C &= 2 is same as C = C & 2
^=	bitwise exclusive OR and assignment operator	C ^= 2 is same as C = C ^ 2
=	bitwise inclusive OR and assignment operator	C = 2 is same as C = C 2

Range Operators

Swift includes two range operators, which are shortcuts for expressing a range of values. The following table explains these two operators.

Operator	Description	Example
Closed Range	(a...b) defines a range that runs from a to b, and includes the values a and b.	1...5 gives 1, 2, 3, 4 and 5
Half-Open Range	(a.< b) defines a range that runs from a to b, but does not include b.	1.< 5 gives 1, 2, 3, and 4
One-sided Range	a... , defines a range that runs from a to end of elements ...a , defines a range starting from start to a	1... gives 1 , 2,3... end of elements

		...2 gives beginning... to 1,2
--	--	--------------------------------

Misc Operators

Swift supports a few other important operators including **range** and **?** : which are explained in the following table.

Operator	Description	Example
Unary Minus	The sign of a numeric value can be toggled using a prefixed -	-3 or -4
Unary Plus	Returns the value it operates on, without any change.	+6 gives 6
Ternary Conditional	Condition ? X : Y	If Condition is true ? Then value X : Otherwise value Y

Decision Making

Swift provides the following types of decision making statements.

S.N.	Statement	Description	Example
1	if statement	An if statement consists of a Boolean expression followed by one or more statements.	<pre>var varA:Int = 10; /* Check the boolean condition using if statement */ if varA < 20 { /* If condition is true then print the following */ print("varA is less than 20"); } print("Value of variable varA is \(varA)");</pre> <p>Output: varA is less than 20 Value of variable varA is 10</p>
2	if...else statement	An if statement can be followed by an optional else statement, which executes when the Boolean expression is false.	<pre>var varA:Int = 100; /* Check the boolean condition using if statement */ if varA < 20 {</pre>

			<pre> /* If condition is true then print the following */ print("varA is less than 20"); } else { /* If condition is false then print the following */ print("varA is not less than 20"); } print("Value of variable varA is \ (varA)"); </pre> <p>Output: varA is not less than 20 Value of variable varA is 100</p>
3	if...elseif...else Statement	An if statement can be followed by an optional else if...else statement, which is very useful to test various conditions using single if...else if statement.	<pre> var varA:Int = 100; /* Check the boolean condition using if statement */ if varA == 20 { /* If condition is true then print the following */ print("varA is equal to than 20"); } else if varA == 50 { /* If condition is true then print the following */ print("varA is equal to than 50"); } else { /* If condition is false then print the following */ print("None of the values is matching"); } </pre>

			<pre>print("Value of variable varA is \(varA)");</pre> <p>Output: None of the values is matching Value of variable varA is 100</p>
4	Nested if statement	You can use one if or else if statement inside another if or else if statement(s).	<pre>var varA:Int = 100; var varB:Int = 200; /* Check the boolean condition using if statement */ if varA == 100 { /* If condition is true then print the following */ print("First condition is satisfied"); if varB == 200 { /* If condition is true then print the following */ print("Second condition is also satisfied"); } } print("Value of variable varA is \(varA)"); print("Value of variable varB is \(varB)");</pre> <p>Output: First condition is satisfied Second condition is also satisfied Value of variable varA is 100 Value of variable varB is 200</p>

5	switch statement	A switch statement allows a variable to be tested for equality against a list of values.	<pre> var index = 10 switch index { case 100 : print("Value of index is 100") case 10,15 : print("Value of index is either 10 or 15") case 5 : print("Value of index is 5") default : print("default case") } </pre> <p>Output: Value of index is either 10 or 15</p>
---	-------------------------	--	--

Loops

Swift programming language provides the following kinds of loop to handle looping requirements.

S.N.	Statement	Description	Example
1	for-in	This loop performs a set of statements for each item in a range, sequence, collection, or progression.	<pre> var someInts:[Int] = [10, 20, 30] for index in someInts { print("Value of index is \(index)") } </pre> <p>Output: Value of index is 10 Value of index is 20 Value of index is 30</p>
2	while loop	Repeats a statement or group of statements while a given condition is true. It tests the condition before executing the loop body.	<pre> var index = 10 while index < 15 { print("Value of index is \(index)") index = index + 1 } </pre> <p>Output: Value of index is 10 Value of index is 11</p>

			Value of index is 12 Value of index is 13 Value of index is 14
3	repeat...while loop	Like a while statement, except that it tests the condition at the end of the loop body.	<pre>var index = 10 repeat { print("Value of index is \(index)") index = index + 1 } while index < 15</pre> <p>Output: Value of index is 10 Value of index is 11 Value of index is 12 Value of index is 13 Value of index is 14</p>

Arrays

You can create an empty array of a certain type using the following initializer syntax –

```
var someArray = [SomeType]()
```

You can use the following statement to create an empty array of **Int** type having 3 elements and the initial value as zero –

```
var someInts = [Int](count: 3, repeatedValue: 0)
```

Following is one more example to create an array of three elements and assign three values to that array –

```
var someInts:[Int] = [10, 20, 30]
```

You can retrieve a value from an array by using **subscript** syntax, passing the index of the value you want to retrieve within square brackets immediately after the name of the array as follows –

```
var someVar = someArray[index]
```

The following example shows how to create, initialize, and access arrays –

```
var someInts = [Int](count: 3, repeatedValue: 10)

var someVar = someInts[0]
print( "Value of first element is \(someVar)" )
print( "Value of second element is \(someInts[1])" )
print( "Value of third element is \(someInts[2])" )
```

Output:

```
Value of first element is 10
Value of second element is 10
Value of third element is 10
```


Strings

You can create a String either by using a string literal or creating an instance of a String class as follows –

```
// String creation using String literal
var stringA = "Hello, Swift!"
print( stringA )

// String creation using String instance
var stringB = String("Hello, Swift!")
print( stringB )
```

Empty String

You can create an empty String either by using an empty string literal or creating an instance of String class as shown below. You can also check whether a string is empty or not using the Boolean property **isEmpty**.

```
// Empty string creation using String literal
var stringA = ""

if stringA.isEmpty {
    print( "stringA is empty" )
} else {
    print( "stringA is not empty" )
}

// Empty string creation using String instance
let stringB = String()

if stringB.isEmpty {
    print( "stringB is empty" )
} else {
    print( "stringB is not empty" )
}
```

Output:

```
stringA is empty
stringB is empty
```

String Concatenation

You can use the + operator to concatenate two strings or a string and a character, or two characters. Here is a simple example –

```
let constA = "Hello,"
let constB = "World!"

var stringA = constA + constB
print( stringA )
```

Output:

```
Hello,World!
```

String Length

Swift strings do not have a **length** property, but you can use the global `count()` function to count the number of characters in a string. Here is a simple example –

```
var varA = "Hello, Swift 4!"  
  
print( "\(varA), length is \(varA.count)" )
```

Output:

Hello, Swift 4!, length is 15

String Comparison

You can use the `==` operator to compare two strings variables or constants. Here is a simple example –

```
var varA = "Hello, Swift!"  
var varB = "Hello, World!"  
  
if varA == varB {  
    print( "\(varA) and \(varB) are equal" )  
} else {  
    print( "\(varA) and \(varB) are not equal" )  
}
```

Output:

Hello, Swift! and Hello, World! are not equal

Function

A function's arguments must always be provided in the same order as the function's parameter list and the return values are followed by `→`.

```
func funcname(Parameters) -> returntype {  
    Statement1  
    Statement2  
    ---  
    Statement N  
    return parameters  
}
```

Take a look at the following code. The student's name is declared as string datatype declared inside the function 'student' and when the function is called, it will return student's name.

```
func student(name: String) -> String {  
    return name  
}  
  
print(student(name: "First Program"))  
print(student(name: "About Functions"))
```

When we run the above program using playground, we get the following result –

First Program
About Functions

Functions with Parameters

A function is accessed by passing its parameter values to the body of the function. We can pass single to multiple parameter values as tuples inside the function.

```
func mult(no1: Int, no2: Int) -> Int {  
    return no1*no2  
}  
  
print(mult(no1: 2, no2: 20))  
print(mult(no1: 3, no2: 15))  
print(mult(no1: 4, no2: 30))
```

When we run above program using playground, we get the following result –

40
45
120

Functions without Parameters

Following is an example having a function without a parameter –

```
func votersname() -> String {  
    return "Alice"  
}  
print(votersname())
```

When we run the above program using playground, we get the following result –

Alice

Views and the View Hierarchy

A view hierarchy defines the relationships of views in a window to each other. You can think of a view hierarchy as an inverted tree structure with the window being the top node of the tree. Under it come views structurally specified by parent-child relationships. From a visual perspective, the essential fact of a view hierarchy is enclosure: one view contains one or more other views, and the window contains them all.

The view hierarchy is a major part of the responder chain, and it is something that the application frameworks use to determine the layering order of views when they render the content of a window in a drawing pass. The view hierarchy is also the governing concept behind view composition: You construct compound views by adding sub views to a super view. Finally, the view hierarchy is a critical factor in the multiple coordinate systems found in a window.

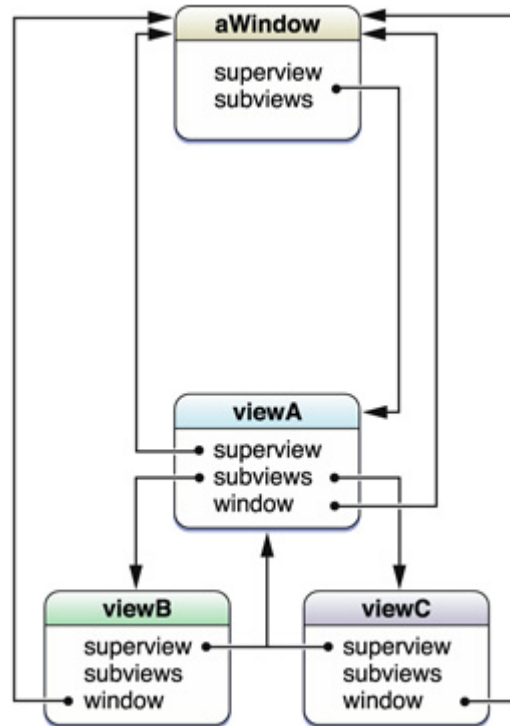
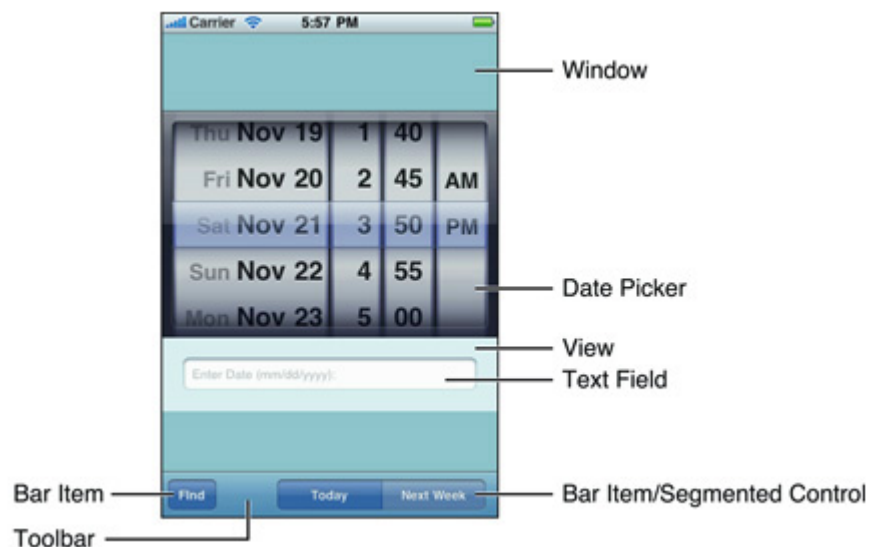


Figure 8-20. View Hierarchy in iOS

A view is related to other views through two properties, and these relationships determine the form of the hierarchy:

- **super view** — The view above a given view in the hierarchy; this is the view that encloses it. All views except the topmost view must have a super view.
- **sub views** — The views below a given view in the hierarchy; these are the views that it encloses. A view may have any number of sub views, or it may have none.



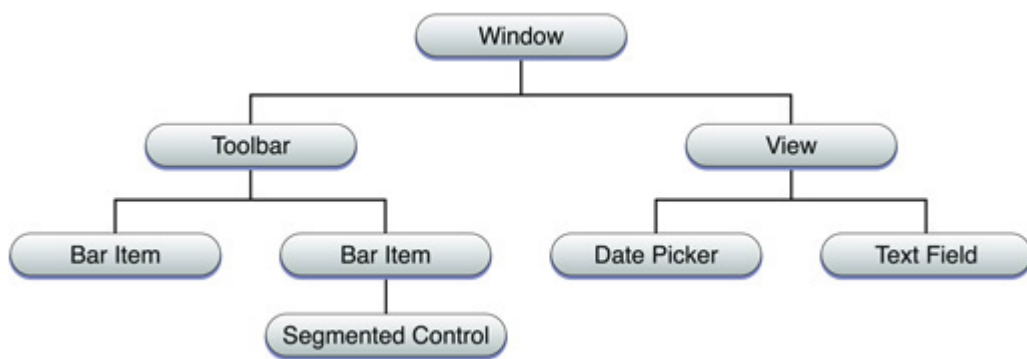


Figure 8-21. Example of View Hierarchy in iOS

Storyboard and View Controllers

Storyboards are an exciting feature first introduced in iOS 5, which save time building user interfaces for your apps. Storyboards allow you to prototype and design multiple view controller views within one file, and also let you create transitions between view controllers.

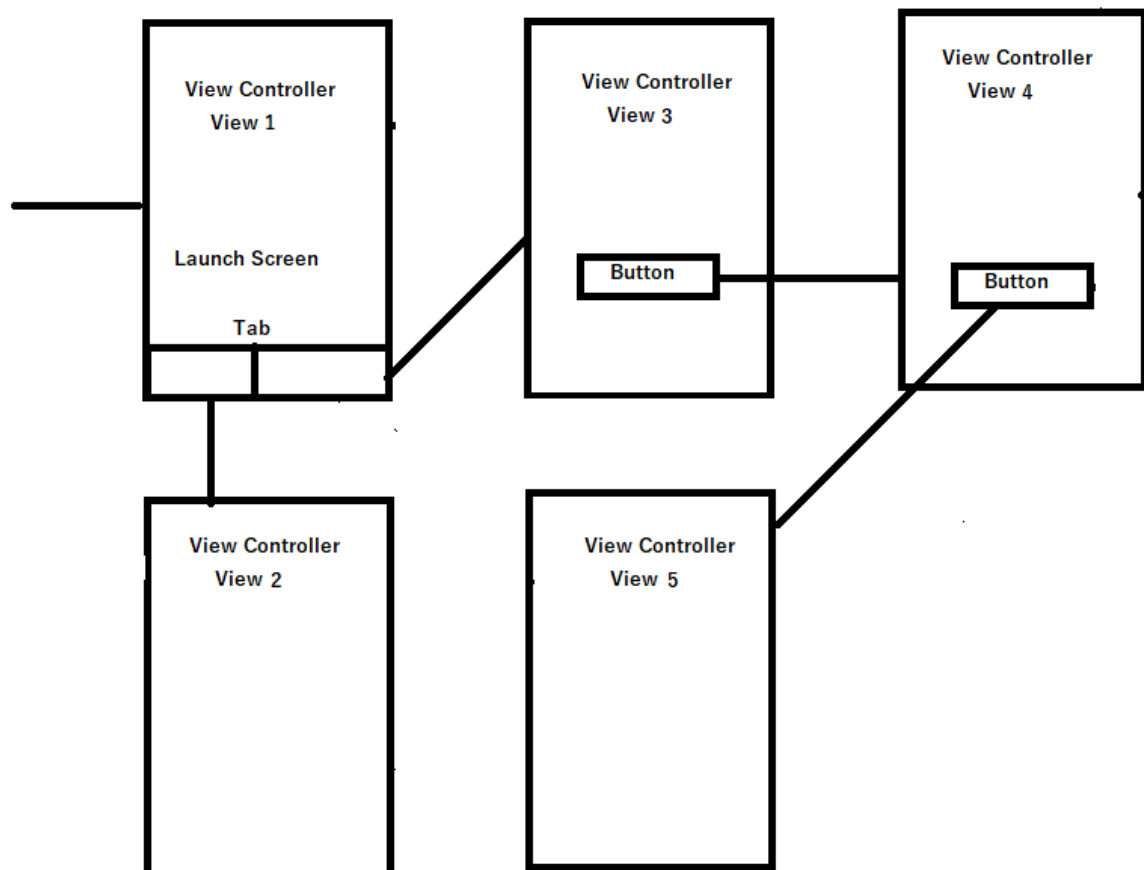


Figure 8-22. Example of Storyboard with multiple view controllers

To demonstrate storyboard now I am going to create a storyboard containing a Tab and multiple view controllers linking with the Tab.

So, let's begin by creating a project with following information:

- Product Name: **MyExample**.
- Organization Name: Fill this in however you like.
- Organization Identifier: The identifier you use for your apps.
- Language: **Swift**.
- User Interface: **Storyboard**.
- Make sure you've unchecked the **Use Core Data, Include Unit Tests** and **UI Tests** options.

Open **Main.storyboard** in the Project navigator to view it in the **Interface Builder** editor:

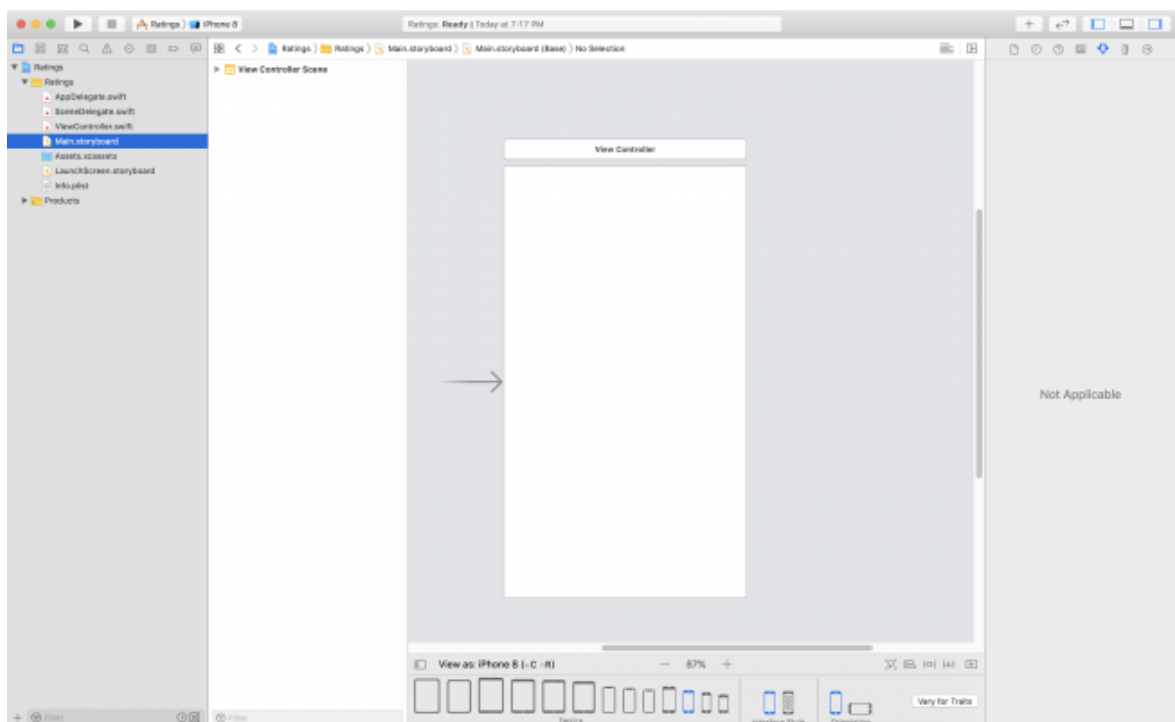


Figure 8-23. First look of Main. Storyboard

Here, you see a single view controller containing an empty view. The arrow pointing to the view controller from the left indicates it's the initial view controller for this storyboard. Xcode enables **Auto Layout** and **Size Classes** by default for storyboards. They allow you to make flexible user interfaces that can resize easily, which is useful for supporting various sizes of iPhones and iPads.

Adding Tab to Storyboard

Drag a **tab bar controller** from the **Object Library** into the canvas. You can filter the list by typing part of the name of the item you're looking for.

Tab bar controller comes pre-configured with two additional view controllers, one for each tab. It's a so-called **container view** controller because it contains one or more other view controllers. Other common containers are the **Navigation controller** and the **Split View controller**.

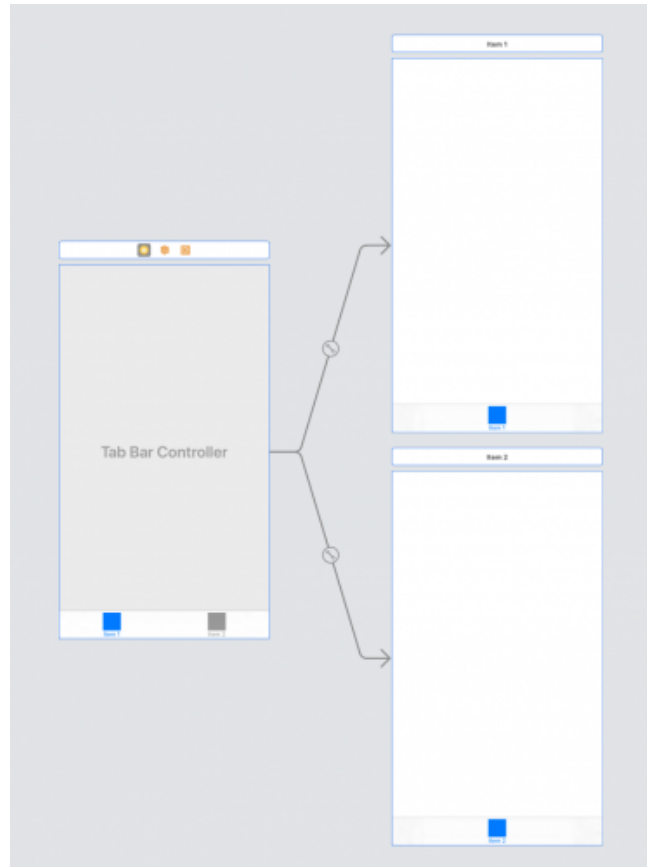


Figure 8-24. Adding Tab Controller to Main. Storyboard

The arrows between the **tab bar controller** and the view controllers it contains represent the container **relationship**. The icon shown below, in the middle of the arrow body, signifies that they have an **embed relationship**.

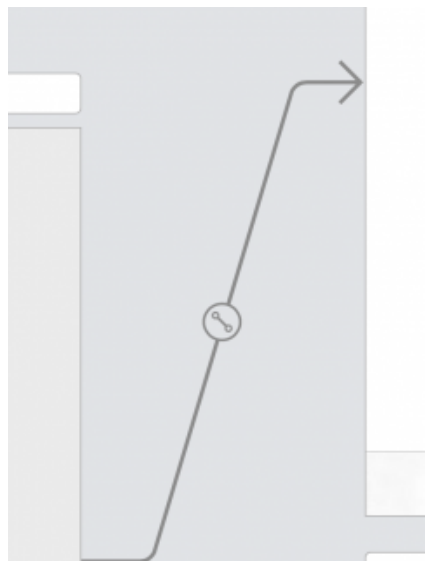


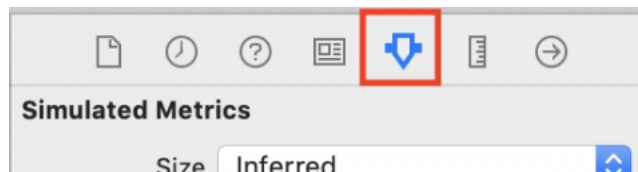
Figure 8-25. Showing Relationship

Build and run and you'll see something like this in the console:

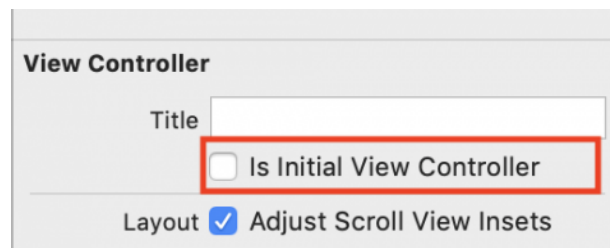
```
MyExample[9912:704408] [WindowScene] Failed to instantiate the default view controller for UIMainStoryboardFile 'Main' - perhaps the designated entry point is not set?
```

This error simply indicates that the app didn't find the initial view controller to show.

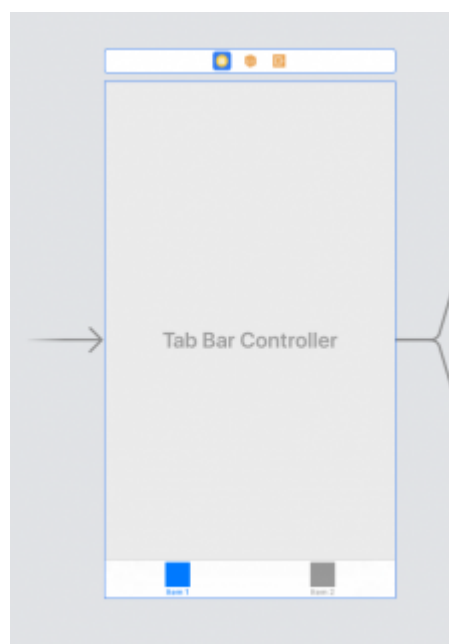
To fix this error, Open **Main.storyboard** and select the **Tab Bar Controller Scene**. On the right, select the **Attribute inspector**.



You'll find a checkbox named **Is Initial View Controller**.



Checking this box will identify the selected view controller as the initial entry point for the storyboard you're on. Also, an arrow will appear on the left of the view controller.






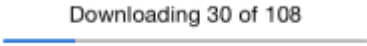
Now, build and run and you'll see an empty view controller with a tab bar that has two items at the bottom.






Figure 8-26. Output demonstrating Tab View Controller

Working with widgets and its attributes

S.N.	UI Controls	Description
1	iOS UI Buttons	<p>In iOS button object is a view that perform some custom actions based on user interactions. When we click or tap a button in iOS it will perform actions which are attached to it and we can easily change the appearance of iOS buttons by adding title or image icons based on our requirements.</p> <p>Button  </p>
2	iOS UI Labels	<p>Label in iOS is a basic UI control which is used to display plain static text or styled text. The content in iOS label control is a read only text we cannot able change or edit the text but we can copy the content of label.</p>

		We can apply custom styles to the content in iOS label control like changing the font style, font size, adding background colors, etc. based on our requirements.
3	iOS UI Text Fields	<p>In iOS Text fields are used to allow users to enter single line of text as an input to an app. The text fields automatically bring up a keyboard to enter a text whenever the user touch or taped it and these text fields are helpful to gather small amounts of text from the user like name, email, etc. By using text fields, we can perform some actions like search operation, based on that text. We can use Text Fields in our iOS application by adding UITextField class reference in our applications.</p> 
4	iOS UI Image View	<p>In iOS, image view is used to show the images in iOS application and it will resize the images automatically to make fit with the current size of view.</p> <p>By using iOS image view, we can show single image or animated sequence of images with transparent or opaque background based on our requirement.</p>
5	iOS Progress Bar (Progress View)	<p>In iOS progress indicators or progress bars are used to show the progress of task in application instead of making people staring at a static screen while performing lengthy data operations.</p> <p>By using iOS progress indicators or bars we can make the people to understand that how long the process will take to finish the task like as shown below:</p>  <p>We can use progress view indicators in our iOS application by adding UIProgressView class reference in our applications.</p>
6	iOS UI DatePicker	In iOS datepicker is a control which is used to select a required date, time, or both and It also provides an interface for a countdown timer. Datepicker is having different modes by using those modes we can specify date and time display formats based on our requirements like only date or time or date and time or countdown timer.
7	iOS UI Switches	In iOS switches are used to let user know the status of option either on or off / active or inactive . Generally, we use switches to toggle an option between on and off .

		<p>By using iOS switches we can implement features like allowing users to turn the settings on or off based on their requirements.</p> <p>The visual representation of iOS switches with on and off options will be like as shown below:</p>  <p>We can use Switches in our iOS application by adding UISwitch class reference in our applications.</p>
8	iOS UI Sliders	<p>In iOS slider is a horizontal bar with thumb control, by using this we can slide between minimum and maximum value. Generally, we use sliders to increase or decrease value like adjust screen brightness level or change the volume of speaker like as shown below:</p>  <p>We can use Sliders in our iOS applications by adding UISlider class reference.</p>
9	iOS UI Segmented Control	<p>In iOS segmented control is a horizontal control which made with a multiple segments and each segment will act as separate view. Our iOS segmented control will be like as shown below:</p>  <p>We can use SegmentedControl in our iOS applications by adding UISegmentedControl class reference.</p>
10	iOS UI WebView	<p>In iOS webview control is used to embed websites within application or show rich HTML web content inside of app and the iOS webview control will act as a HTML iframe to show the website content within an app.</p> <p>We can use Web View in our iOS applications by adding UIWebView class reference.</p>
11	iOS UI Scroll View	<p>In iOS, scroll view is used to show the content which is larger than the scroll view boundaries like viewing larger content in documents, showing multiple images in app. In our iOS application whenever we see the scroll view it is indication that there is extra content other than visible area.</p> <p>We can use Scroll View in our iOS applications by adding UIScrollView class reference.</p>

Creating a Simple iOS Application

Here we going to create a simple application to show the addition of two numbers. Our application contains two Text Fields for inputting two numbers and a Button to trigger click event. Also I will create a Label for displaying the result.

After designing UI our application looks something like this:

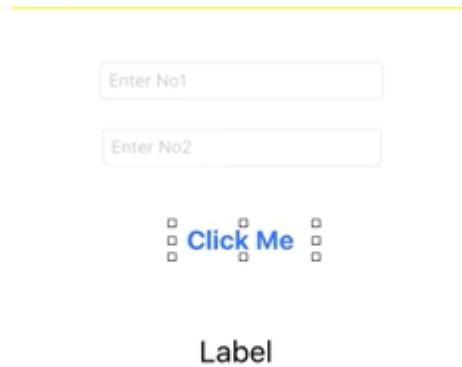


Figure 8-27. UI for adding two numbers

Source code for adding two numbers is shown below:

```
import UIKit
class ViewController: UIViewController {

    @IBOutlet weak var txtFirst: UITextField!
    @IBOutlet weak var txtSecond: UITextField!
    @IBOutlet weak var lblRes: UILabel!
    @IBOutlet weak var btnClick: UIButton!

    @IBAction func btnClick(sender: AnyObject) {
        let first= Double(txtFirst.text!)
        let second= Double(txtSecond.text!)
        let res=Double(first! + second!)
        lblRes.text="Sum is \ \(res)"
    }

    override func viewDidLoad() {
        super.viewDidLoad()
        // Do any additional setup after loading the view, typically
        from a nib.
    }

    override func didReceiveMemoryWarning() {
        super.didReceiveMemoryWarning()
        // Dispose of any resources that can be recreated.
    }
}
```

}

After running our application following output will be produced:

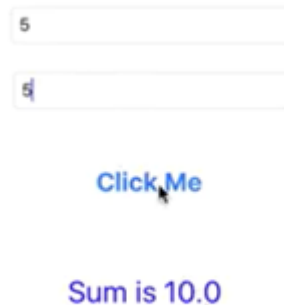


Figure 8-28. Output demonstrating addition of two numbers

Exercise

1. What do you mean by iOS programming? Explain.
2. Explain iOS platform in detail.
3. Explain the procedure for developing hello world application in iOS.
4. What do you mean by connections? How can you connect UI controls to application code? Explain with example.
5. What do you mean by swift language? Explain process of declaring and using variables in swift language.
6. Explain different types of operators used in swift language.
7. Explain branching and looping statements used in swift language in detail.
8. How can you create array and use array in swift language? Explain.
9. Explain view hierarchy in iOS programming in detail.
10. What do you mean by storyboard and view controllers? Explain with example.
11. What are different types of UI controls used in iOS programming? Explain in detail.
12. Develop an iOS application to calculate simple interest.
13. Develop an iOS application to calculate area and perimeter of rectangle.