ikompass

Storyboard

Storyboards are introduced in iOS 5. The storyboard contains the designs for all of your app's screens, and shows how the app goes from one screen to another with big pointy arrows. This initial scene is represented by a square in the middle of the Interface Builder canvas. You may find this odd, considering that iPhones are not actually square. This is a new feature of iOS 8 called "size classes" or "universal storyboards" that lets you design a single storyboard that caters to the different screen sizes of the various iPhone models.

Common terminology:

IBOutlet

In XCode, we design UI elements and the information is kept in a nib file. The corresponding ViewController class is a typical "swift" file or an Objective-C's .h/.m file combination. With this disconnected system in place, a mechanism is required to connect the two ends. IBOutlet is one part of such a mechanism.

An outlet is a property of an object that references another object. An application typically sets outlet connections between its custom controller objects and objects on the user interface. The type qualifier IBOutlet is a tag applied to an property declaration so that the Interface Builder application can recognise the property as an outlet and synchronise the display and connection of it with Xcode.

Note : You create an IBOutlet connection, if you want to access the properties of a UI element, such as if you want to read the text property of a textfield.

IBAction

As discussed, IBOutlet is used to access the properties of UI controls. What if we want to tap the actions associated with them? IBAction completes the full picture. An action is the message a control sends to the target or, from the perspective of the target, the method the target implements to respond to the action message.

IBAction does not designate a data type for a return value; no value is returned. IBAction is a type qualifier that Interface Builder notices during application development to synchronise actions added programmatically with its internal list of action methods defined for a project.

Note : You create an IBAction connection, if you want to respond to a user action / interaction, such as tap on a button.

Objective

In the following exercise, we will create an application using Storyboard feature of Xcode and exhibit and master the following techniques :

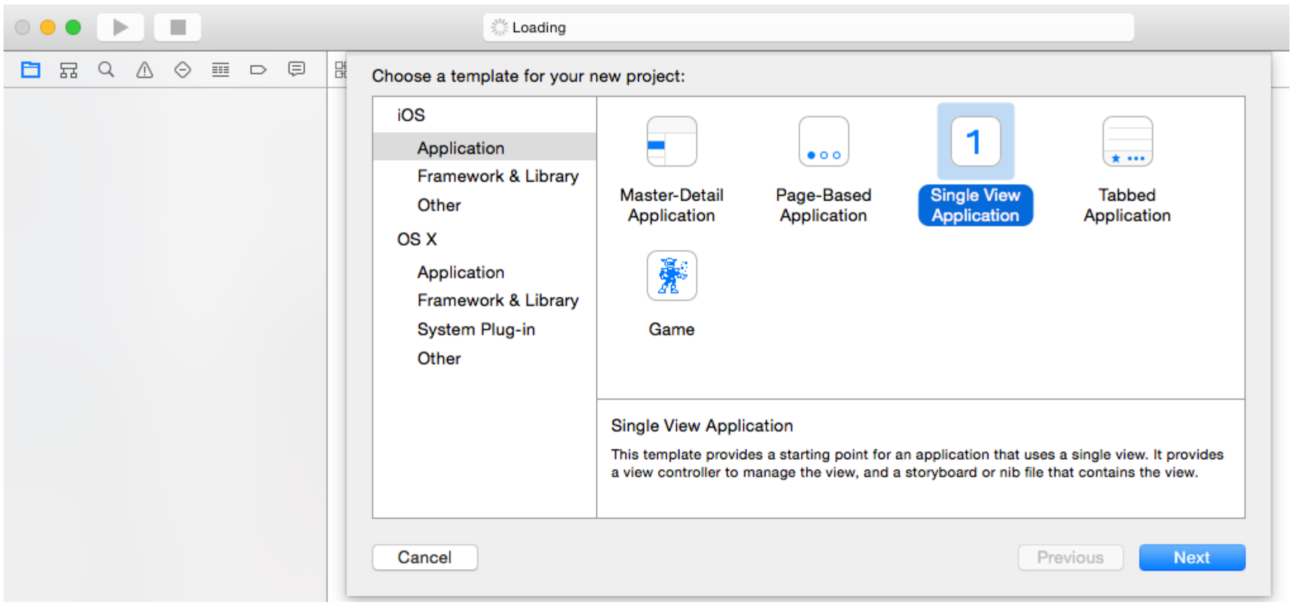
1. Designing a basic user interface.
2. Show / hide various panels of Xcode.
3. Creating an IBOutlet connection.
4. Creating an IBAction connection.
5. Reading values from UI controls.
6. Manipulating values of UI controls.
7. Show / Hide keyboard when the TextField gains or looses focus respectively.

Besides, you will also learn about some of the Foundation Framework classes that are used to manipulate numeric & string values.

Let's start with creating our app and call it : Multiply.

Step 1:- Creating the app and getting familiar with Xcode

1. Launch Xcode and create a new project. On the Choose a template screen, choose Single View Application.



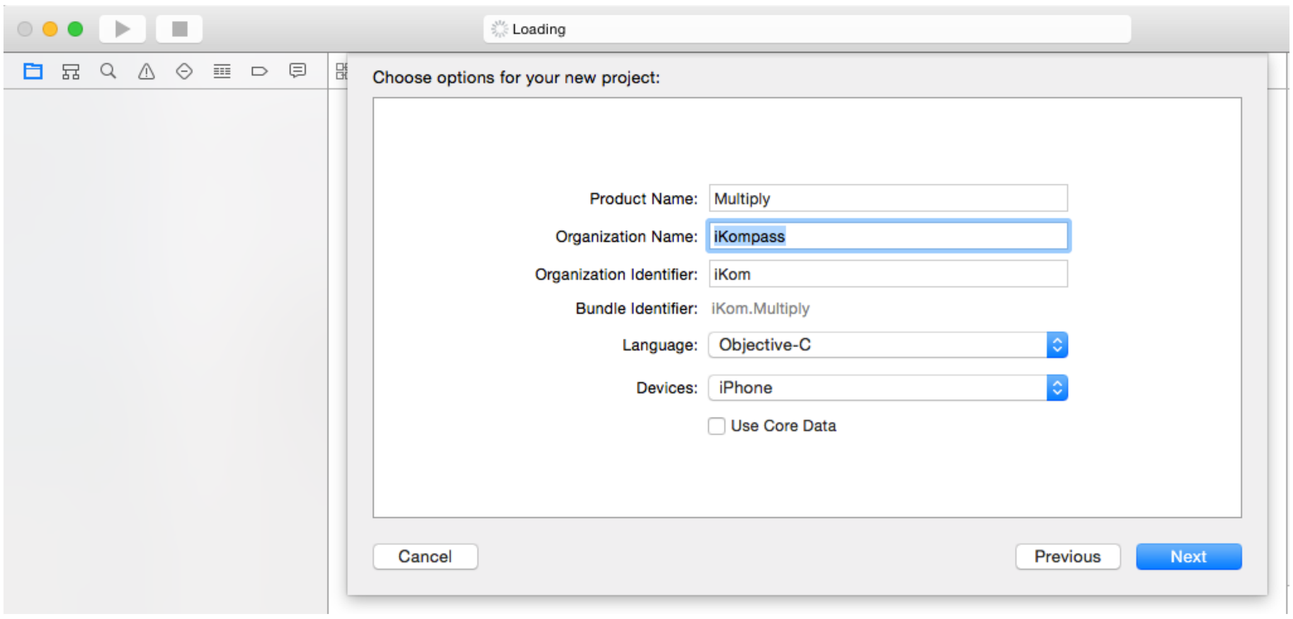
Single View Application

This template provides a starting point for an application that uses a single view. It provides a view controller to manage the view, and a storyboard or nib file that contains the view.

2. Click Next and on the "Choose options for your new project" enter the following information :

Product name	:	Multiply
Organization name	:	iKompass
Organization identifier	:	iKom
Language	:	Objective-C
Devices	:	iPhone

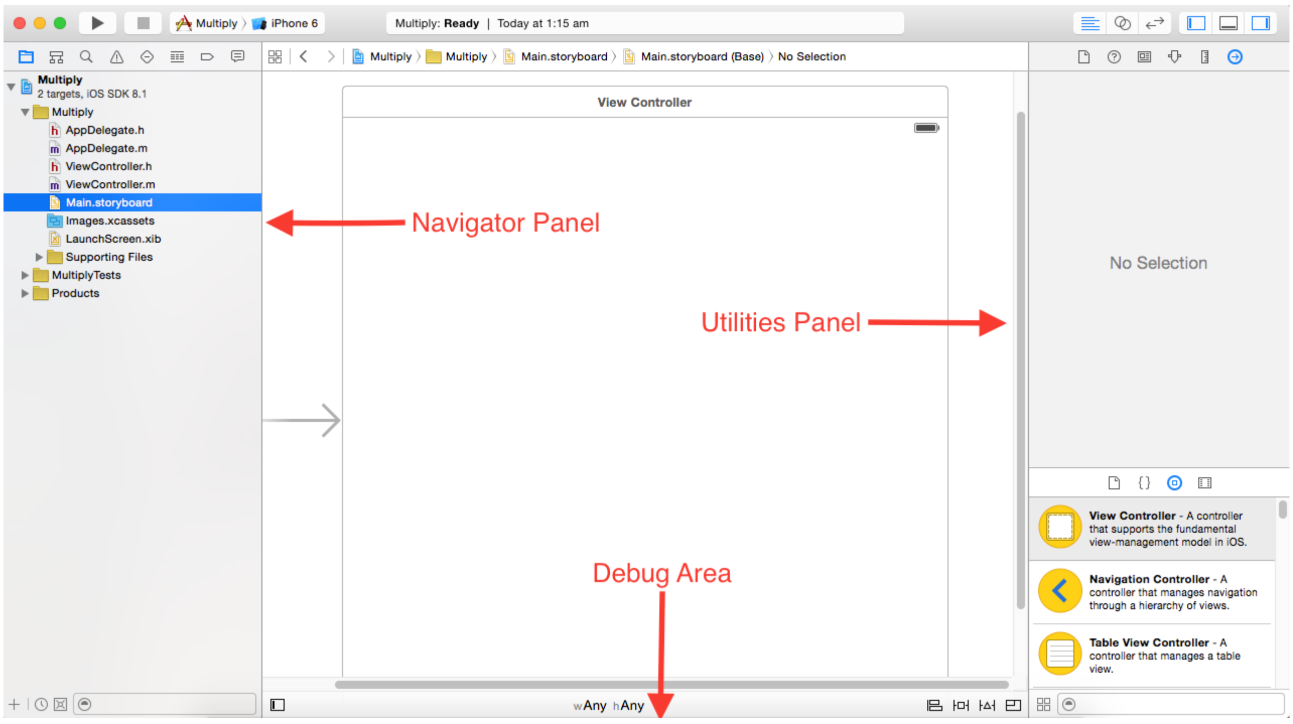
Leave the Use Core Data option, unchecked.



Product Name: Multiply
Organization Name: iKompass
Organization Identifier: iKom
Bundle Identifier: iKom.Multiply
Language: Objective-C
Devices: iPhone
☐ Use Core Data

3. Click "Next" and "Save" your project. Xcode will automatically open your project. Xcode is a very rich IDE and comes loaded with plethora of options and choices. These options and choices are grouped logically and placed in different panel.

Through out the life of your application development, you will be dealing with one or more of these panels and show-hide them as and when required. Thus it is in best of our interest, that, we get familiar with the important panels before we proceed further.

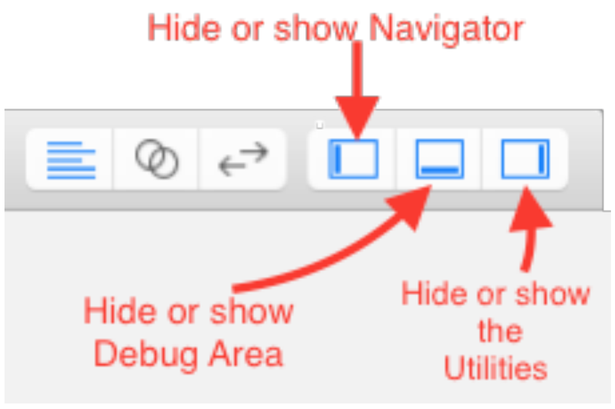

Multiply
iPhone 6
Multiply: Ready | Today at 1:15 am
Multiply > Multiply > Main.storyboard > Main.storyboard (Base) > No Selection
Multiply
2 targets, iOS SDK 8.1
Multiply
AppDelegate.h
AppDelegate.m
ViewController.h
ViewController.m
Main.storyboard
Images.xcassets
LaunchScreen.xib
Supporting Files
MultiplyTests
Products
View Controller
Navigator Panel
Utilities Panel
Debug Area
No Selection
View Controller - A controller that supports the fundamental view-management model in iOS.
Navigation Controller - A controller that manages navigation through a hierarchy of views.
Table View Controller - A controller that manages a table view.
wAny hAny

1.Navigator panel :- This panel along with its multiple tabs, allow the user to access various resources contained in the project.

2. Utilities panel :- This panel gives access to various settings and properties that can be altered or set for various UI controls. Besides, this panel also contains the control libraries that can be dragged and dropped on the scene (View controller). Once again, there are multiple tabs available (exposing different set of controls and options), which we will explore as and when required.

3. Debug area :- This panel contains Console and debug controls such as "Step Over", "Step Into", "Toggle Breakpoints", etc. Besides, it also shows the values of local variables at the time of execution. The information contained in this panel are very helpful during a debug session.

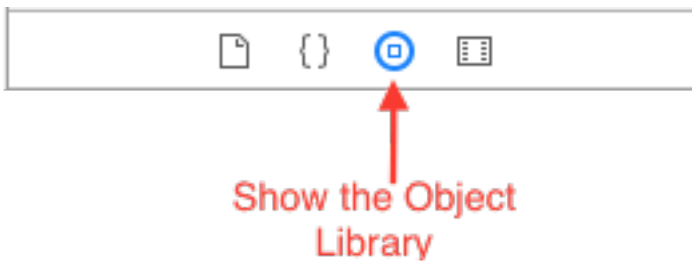
The central piece or the client area which is surrounded by the above three panels is where you will design your view or see the code editor. At times, you may find the client area cramped as the real estate available on the screen is not much. You can toggle the visibility of one or more of these panels as and when required by clicking the respective Show/Hide buttons available on the menu bar.



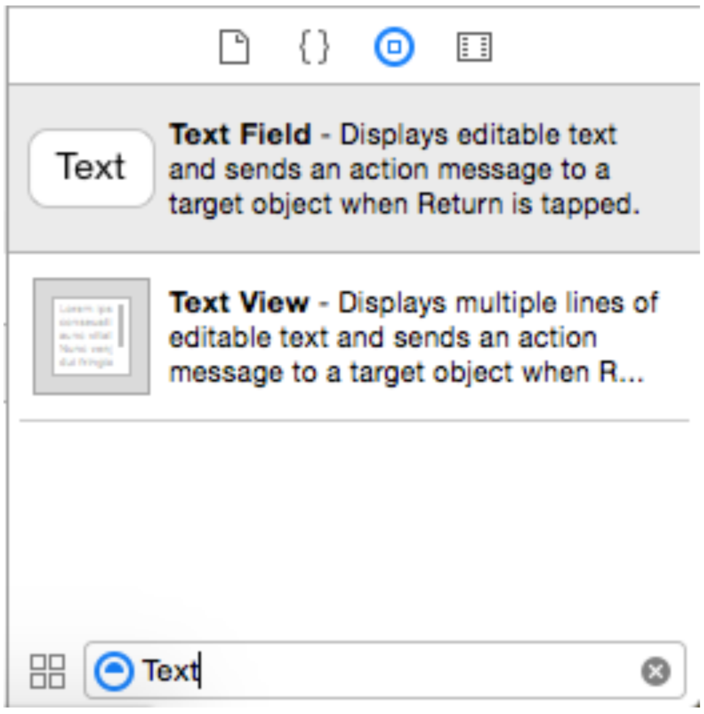
For the time being, lets have all the panel visible.

Step 2:- Add a UITextField to the ViewController and creating the corresponding Outlet

Select the storyboard by clicking on "Main.storyboard" in the Navigator panel. In order to add the UI controls to the scene, we must drag them from the Object library and drop them to the ViewController. The library panel is available on the Utilities panel (at the bottom). Click the "Show the Object Library" button to bring the Object library in front (if it is not already displayed).



Object library panel comes with a very handy search text field, where you can type the name of the object (in our case UI control) and Xcode will list out only those objects which contain that text in their name. Let's try this and type "Text" (without double quotes) in the associated search text field.

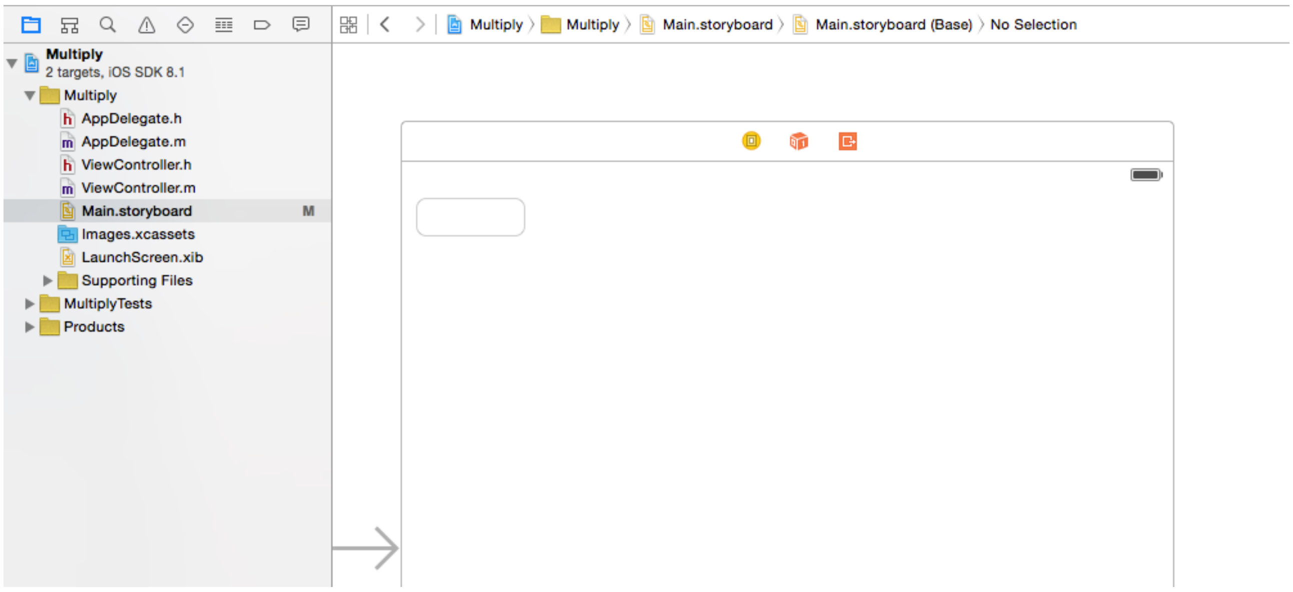


Text Field - Displays editable text and sends an action message to a target object when Return is tapped.

Text View - Displays multiple lines of editable text and sends an action message to a target object when R...

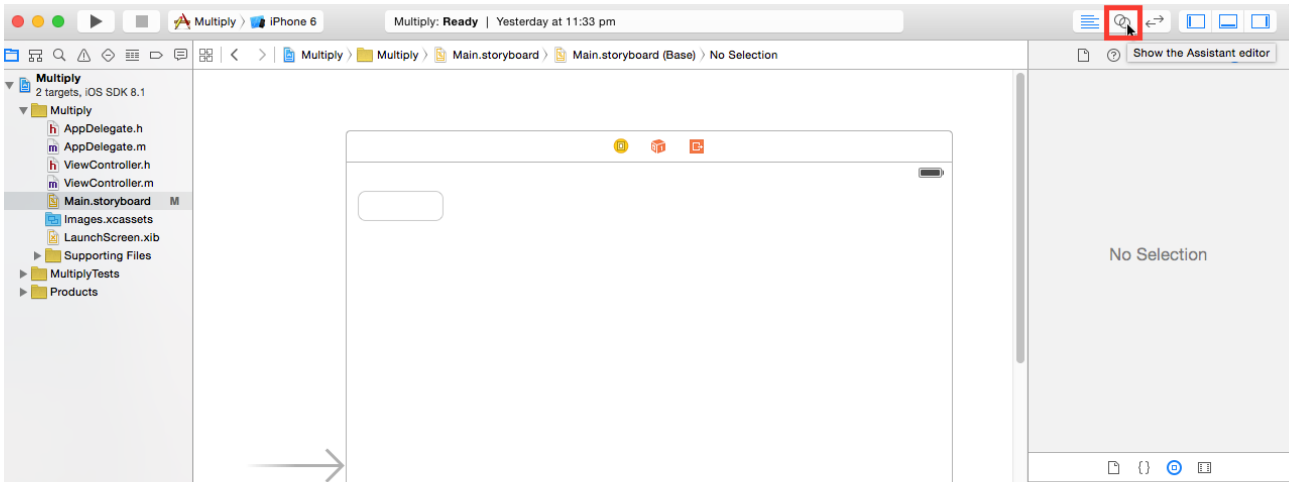
The list is updated almost instantly and shows two controls as shown above. For our purpose, we will drag the Text Field and drop it on the ViewController.

You will notice that when you drag the control on the ViewController, blue guides appear which help you align the control precisely. Use the guides and try to place the text field at the top-left corner.

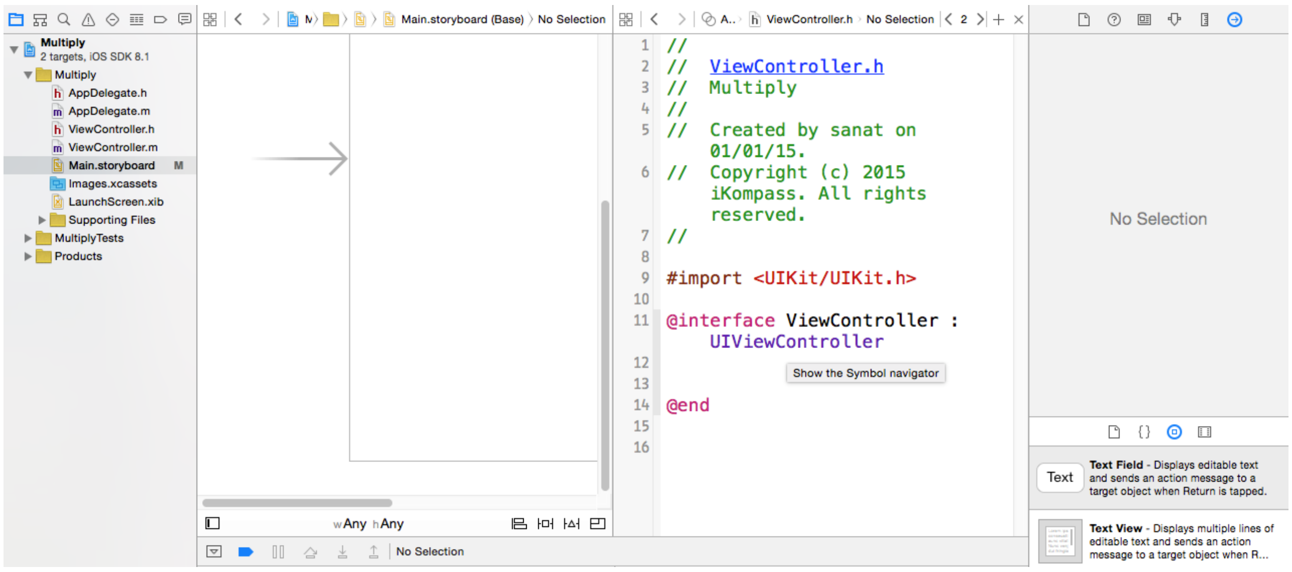


Before we proceed further, wait for a while and think what we are expected to do with a text field and its sole purpose. A text field is used to get inputs from the user. The text entered in a text field is exposed via a property named : text.

In order to access and manipulate the values entered in the textfield, we need to create an IBOutlet connection. As discussed earlier, the IBOutlet connection will connect the object in the scene with the respective Controller's class. The easiest way to achieve this is to open the controller's class and the storyboard side-by-side and "ctrl" drag from the control to the Objective-C class. To do that, let's open the Assistant Editor by clicking the "Show the Assistant Editor" button.



Once the assistant editor is opened, your Xcode may look like as follows:



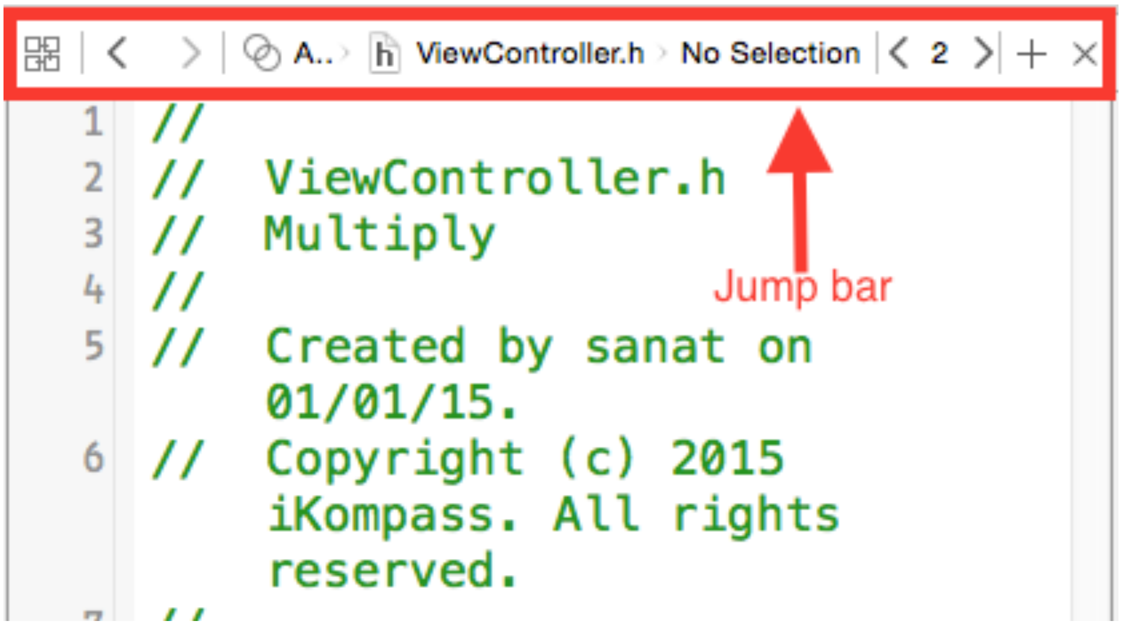
```
//
//  ViewController.h
//  Multiply
//
//  Created by sanat on 01/01/15.
//  Copyright (c) 2015 iKompass. All rights reserved.
//

#import <UIKit/UIKit.h>

@interface ViewController : UIViewController


@end
```

Text Field - Displays editable text and sends an action message to a target object when Return is tapped.

Text View - Displays multiple lines of editable text and sends an action message to a target object when R...

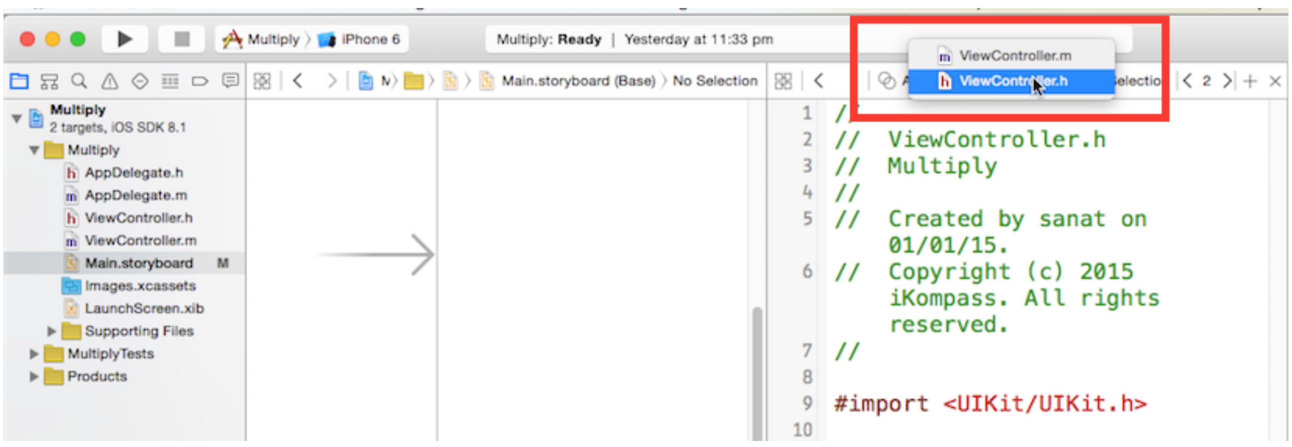
If you feel that there is a space scarcity, go ahead and hide the navigator panel and / or utilities panel.

Sometimes, assistant editor opens the corresponding ".m" file. For this exercise, we will declare the connection IBOutlets in the header file (.h file). In order to bring the header file in the assistant editor, click on the jump bar and just choose the same from the list.



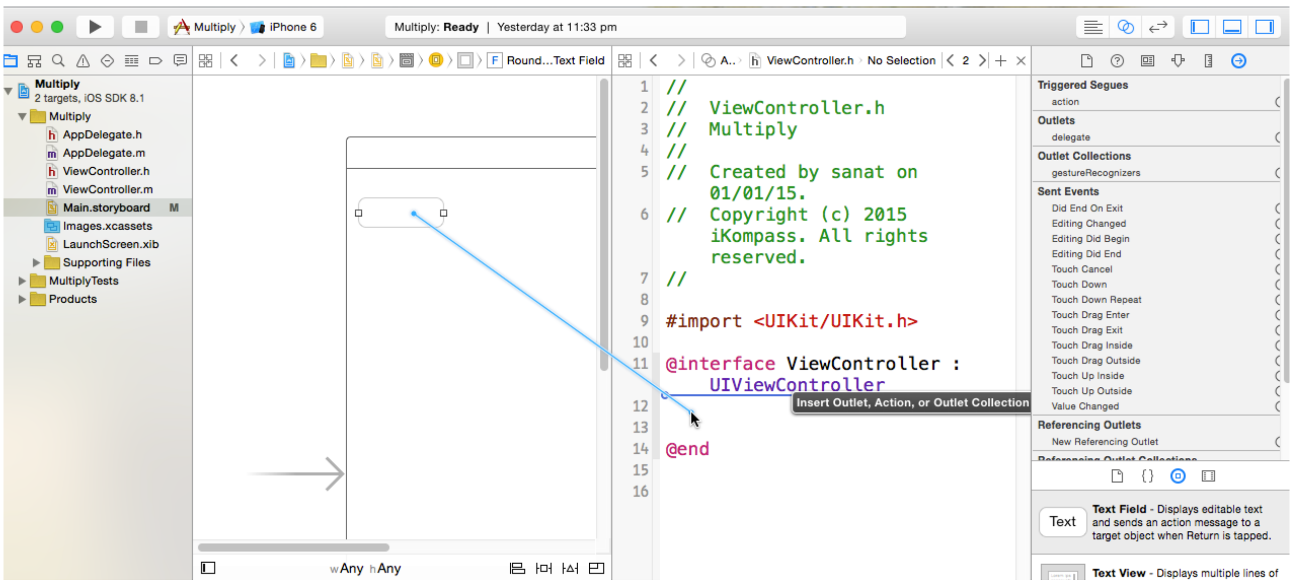
```
//
//  ViewController.h
//  Multiply
//
//  Created by sanat on 01/01/15.
//  Copyright (c) 2015 iKompass. All rights reserved.
```

Be careful when you click on the jump bar and you should click only on the file name that is displayed. The list of options displayed when you click on the jump bar depends upon where you click on it.


Multiply: Ready | Yesterday at 11:33 pm
Main.storyboard (Base) 〉 No Selection
ViewController.m
ViewController.h

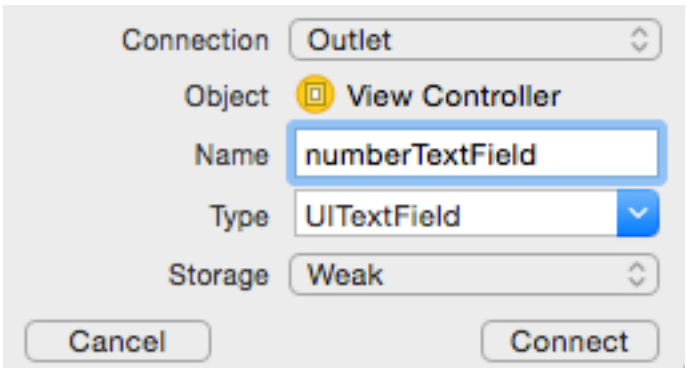
```
//
//  ViewController.h
//  Multiply
//
//  Created by sanat on 01/01/15.
//  Copyright (c) 2015 iKompass. All rights reserved.
//

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

Now, hold the "Ctrl" key and drag from the text field and drop it on "ViewController.h", between the @interface and @end directives.


Multiply: Ready | Yesterday at 11:33 pm
Round...Text Field
ViewController.h 〉 No Selection
Insert Outlet, Action, or Outlet Collection
Triggered Segues
Outlets
Outlet Collections
Sent Events
Referencing Outlets
Text Field - Displays editable text and sends an action message to a target object when Return is tapped.
Text View - Displays multiple lines of

```
//
//  ViewController.h
//  Multiply
//
//  Created by sanat on 01/01/15.
//  Copyright (c) 2015 iKompass. All rights reserved.
//

#import <UIKit/UIKit.h>

@interface ViewController : UIViewController



@end
```

It will open a popup where you can enter the name for the Outlet's property. Enter the following values for the new outlet:



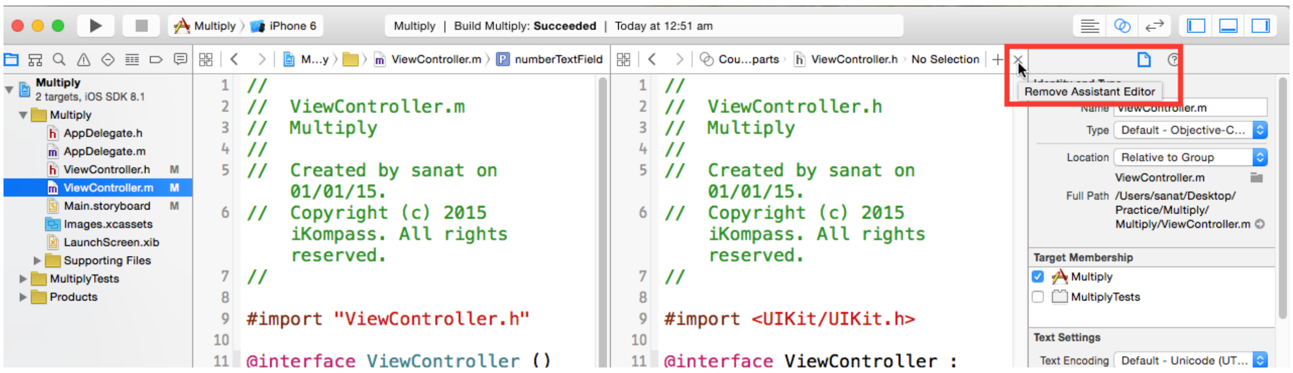
Connection: Outlet
Object: View Controller
Name: numberTextField
Type: UITextField
Storage: Weak

Click "Connect" when done. The following entry will be made in the ViewController's header file :

@property (weak, nonatomic) IBOutlet UITextField *numberTextField;

As the entry suggests, it is an IBOutlet property for a UITextField and named "numberTextField".

You can close the "Assistant Editor" by clicking the "Remove Assistant Editor" button (the small 'X' button present on the jump bar).



Multiply | Build Multiply: **Succeeded** | Today at 12:51 am

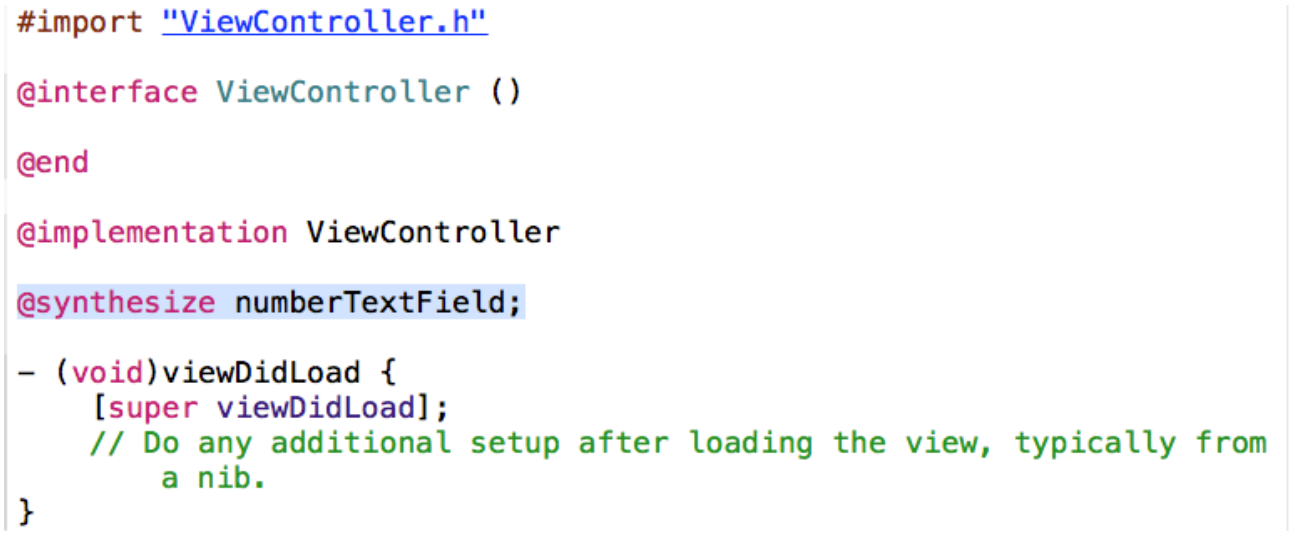
```
//
//  ViewController.m
//  Multiply
//
//  Created by sanat on
    01/01/15.
//  Copyright (c) 2015
    iKompass. All rights
    reserved.
//

#import "ViewController.h"

@interface ViewController ()
```

```
//
//  ViewController.h
//  Multiply
//
//  Created by sanat on
    01/01/15.
//  Copyright (c) 2015
    iKompass. All rights
    reserved.
//

#import <UIKit/UIKit.h>

@interface ViewController :
```

Name: ViewController.m
Type: Default - Objective-C...
Location: Relative to Group
ViewController.m
Full Path: /Users/sanat/Desktop/Practice/Multiply/Multiply/ViewController.m
Target Membership: [✓ Multiply] MultiplyTests
Text Encoding: Default - Unicode (UT...

Let's synthesise the property in the implementation file. To do that, select the "ViewController.m" file in the Navigation panel and add the following piece of code just after the @implementation line:

@synthesize numberTextField;

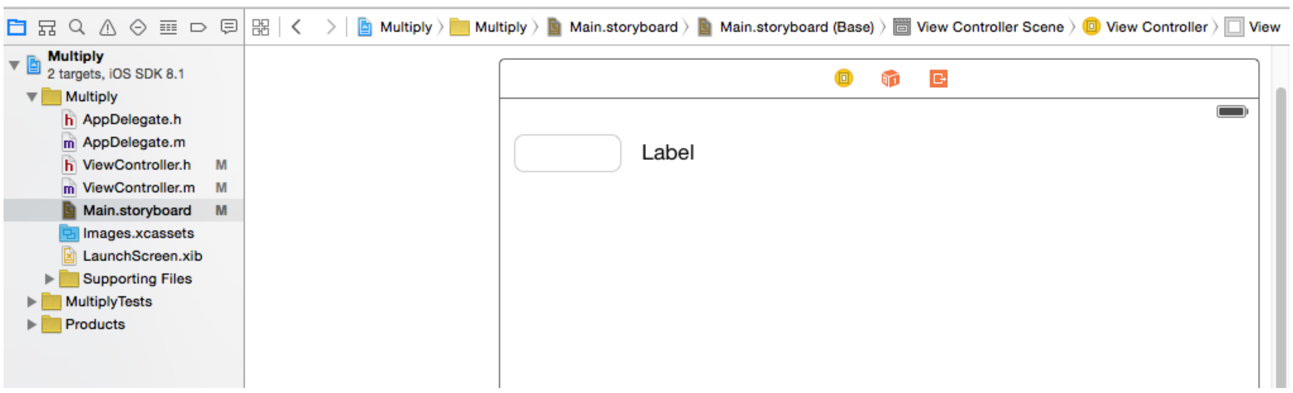


```
#import "ViewController.h"

@interface ViewController ()

@end

@implementation ViewController

@synthesize numberTextField;

- (void)viewDidLoad {
    [super viewDidLoad];
    // Do any additional setup after loading the view, typically from
        a nib.
}
```

Step 3:- Add the multiplier label, set its text property and create the corresponding outlet.

Just like we added the UITextField, we can add the label. A UILabel is usually used to display static text. A UILabel does not expect user interaction.

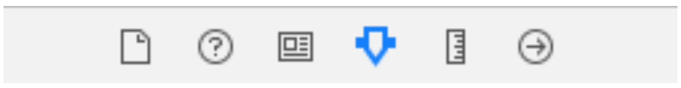
Select Main.storyboard in Navigation panel and, once again use the Object library to find the UILabel. Type "label" in the associated search text field. The Object library will show the Label control. Drag it from there and drop it on the scene, just next to the numberTextField. Align the label horizontally and vertically with respect to the textfield using the blue guides that appear.



In order to change the text value of the label, we need to open the "Attributes Inspector". The Utilities panel contains the following six inspectors (from left to right) :

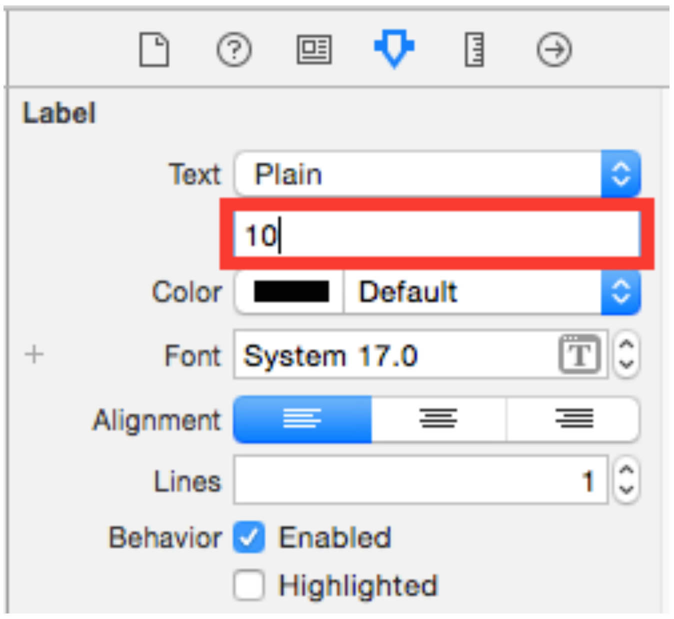
1. File Inspector
2. Quick Help Inspector
3. Identity Inspector
4. Attributes Inspector
5. Size Inspector
6. Connections Inspector

In order to open any one of the inspectors, click on their respective icons in the Utilities panel:



Select the label and click on the Attributes inspector. The attributes inspector opens up in the Utilities panel.

Delete the text "Label" in the Attributes inspector, and enter 10.



Label

Text: Plain

10

Color: Default

Font: System 17.0

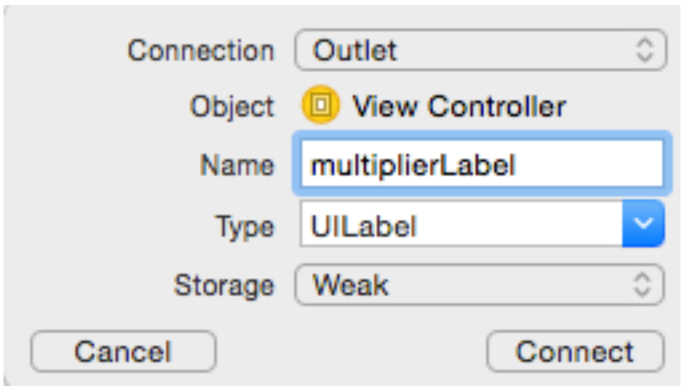
Alignment

Lines: 1

Behavior: ☑ Enabled

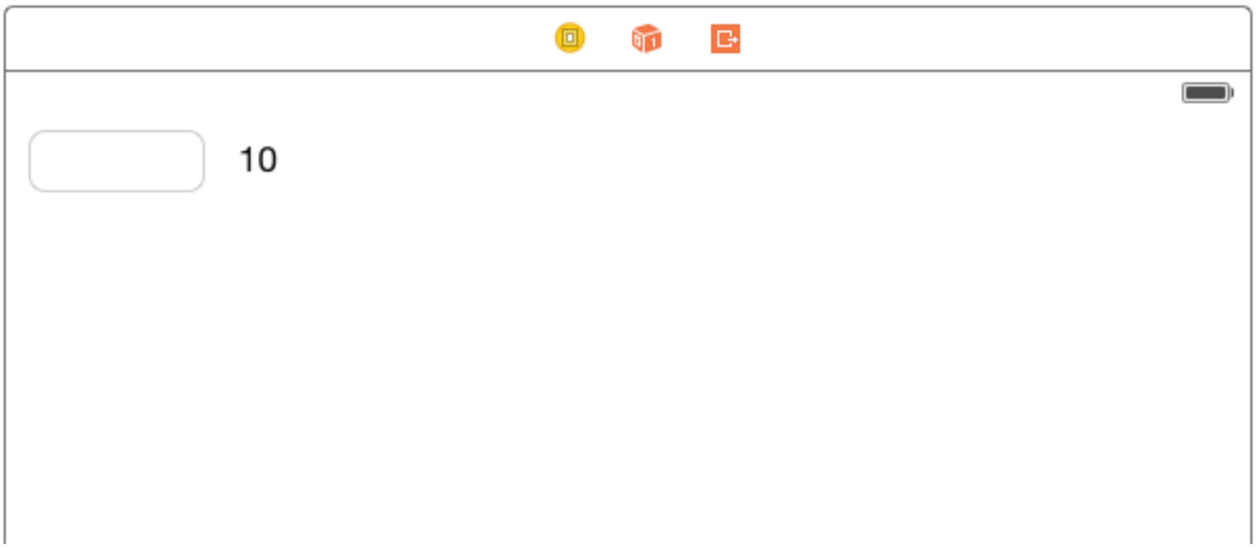
☐ Highlighted

Create the outlet for the label in the same fashion as we did for the textfield and name it : multiplierLabel.



Connection: Outlet
Object: View Controller
Name: multiplierLabel
Type: UILabel
Storage: Weak

At the end of Step 3, your scene should appear as follows :



Step 4:- Add the Results label and create its Outlet.

The best place to show the result of the multiplication is to show it right next to the second operand, as we have been habitual of seeing a mathematical equation in the following format :

a x b = result

Just add the label and create its outlet as done in the earlier steps. Call the outlet of the results label : answerLabel

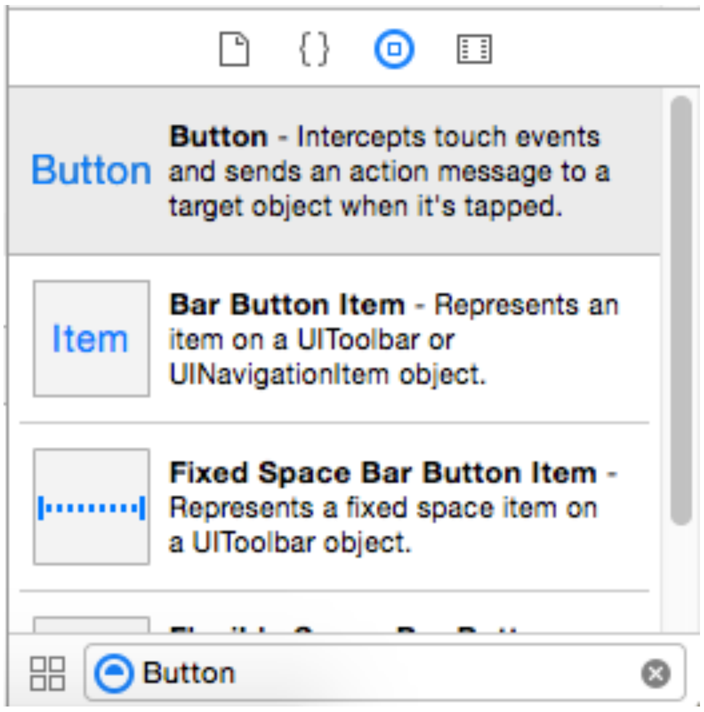
Forget not to delete the text of this label and make it blank as we intend to display the result dynamically.

Step 5:- Add a UIButton and tap its touch event to display the result in the answer label

We have a text field, a multiplier and an answer label. Now we need a trigger to actually calculate the result and display it in the answerLabel. Buttons have traditionally been used to trigger common actions in any UI driven application. We will stick to the same methodology and use a button.

Let's add our button by dragging it from the Object library and placing it below our text field and other controls. Try to place it in the middle of the screen (horizontally).

Type "Button" (without quotes), in the associated Search text field in the Object library and once the Button control is listed, drag and drop it on the scene.

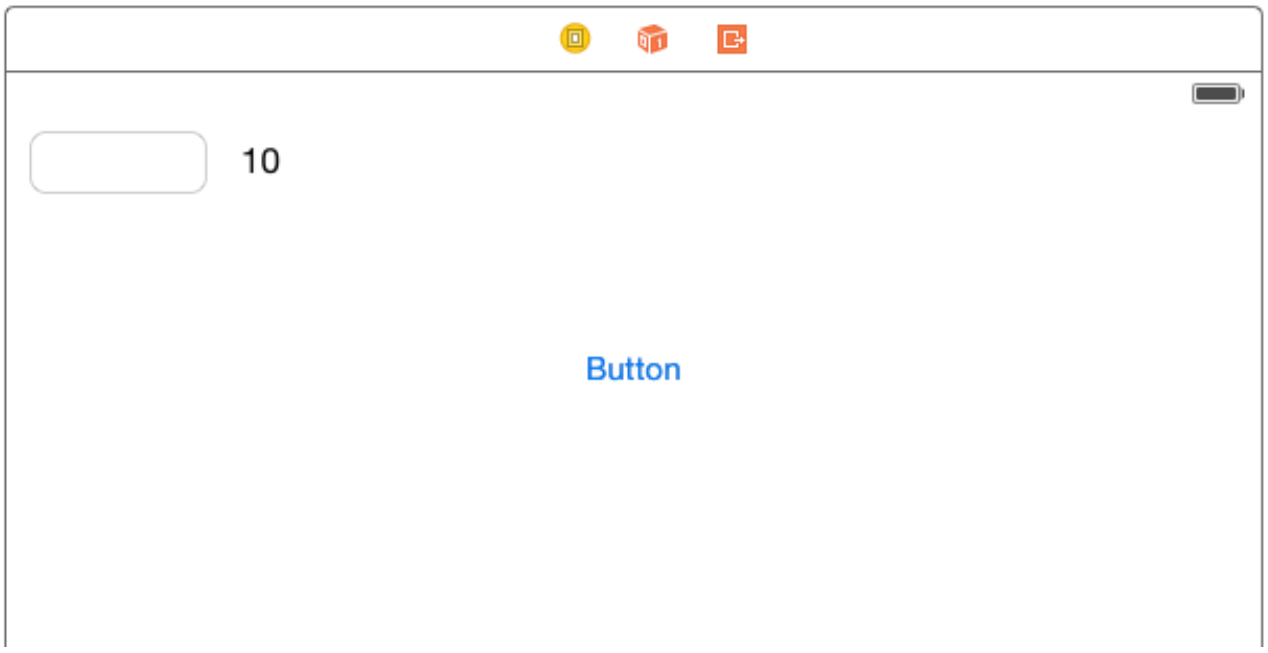


Button - Intercepts touch events and sends an action message to a target object when it's tapped.

Bar Button Item - Represents an item on a UIToolbar or UINavigationItem object.

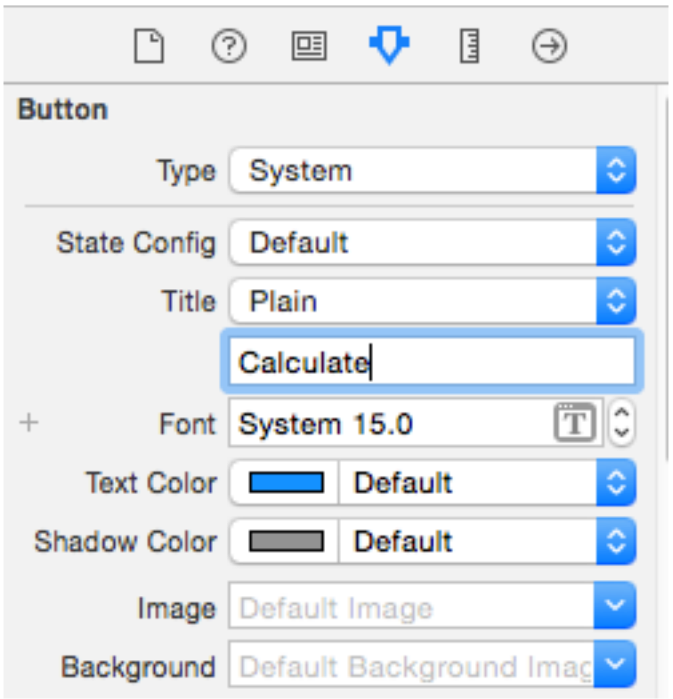
Fixed Space Bar Button Item - Represents a fixed space item on a UIToolbar object.

Once you have placed your button properly, your screen should appear as below :



The default title displayed for a newly added button is "Button" which is not very intuitive. We wish to change it to something more meaningful. As you guessed it right, it can be done through Attributes inspector panel.

So select the button and click the Show Attributes inspector button in the utilities panel. Under the "Title" section, just below the drop-down field, you will find the title text field. Delete the existing text and enter : Calculate.



Button

Type: System

State Config: Default

Title: Plain

Calculate

Font: System 15.0

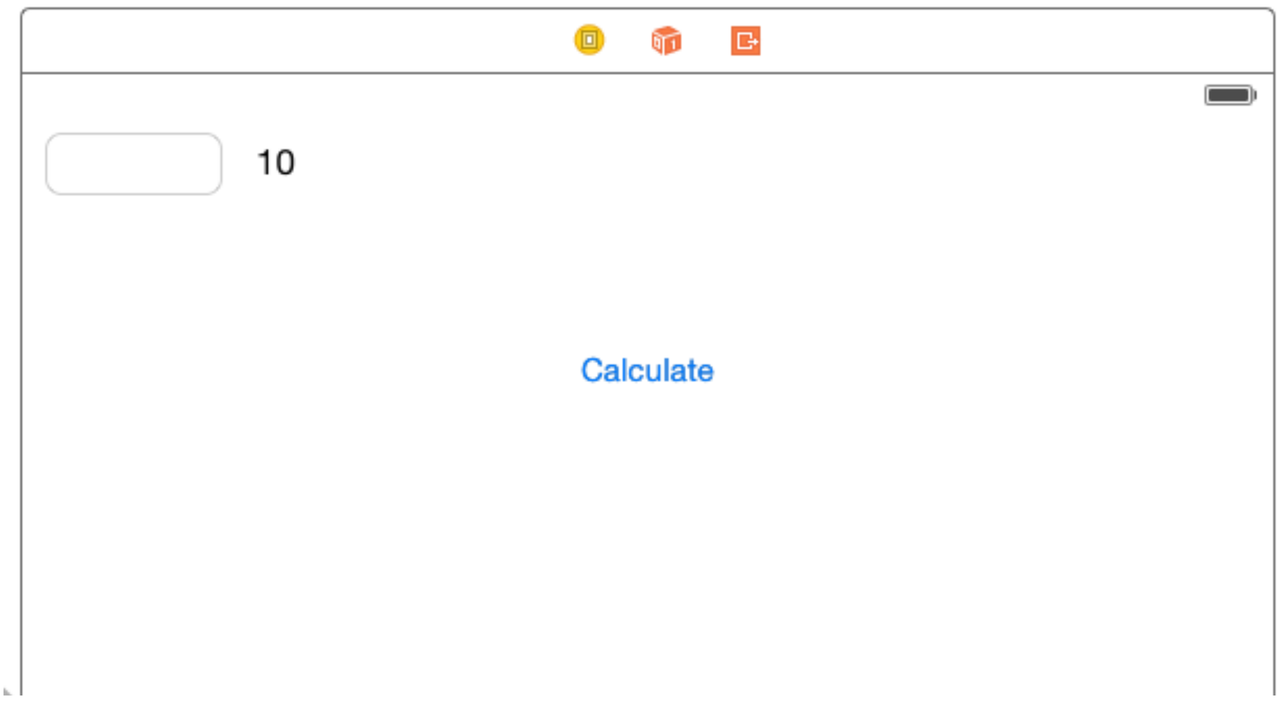
Text Color: Default

Shadow Color: Default

Image: Default Image

Background: Default Background Imag

If your button is not sized properly to show the entire text, you may need to increase its width by dragging the left and right size handles. Remember to readjust the button horizontally so that it fits into the middle of the view controller after you have resized it. Finally, your screen should look like :



<u>**Important :- Synthesize all the IBOutlet properties that you have created**</u>

Adding the IBAction connection for the "Calculate" button

Remember that we created IBOutlet for the labels and UITextField earlier, as we wanted to access their values (properties). However, in case of the button, we are not interested in any of its property such as title, color, etc. What we are more interested in is about how to respond to, when a user touches / taps this button !

iOS apps are event-driven and the flow of the app is determined by events: system events and user actions. The user performs actions on the interface, which trigger events in the app. These events result in the execution of the app's logic and manipulation of its data. As a developer, it is our responsibility to identify exactly which actions a user can perform and what happens in response to those actions.

To do this, we need to create an IBAction method and wire it with one of the events exposed by the button. A button or any other UIControl may respond to more than one event and as such it is very important that we create the connection with the correct "Event". For example, a button may have the following valid events :

i. Touch Up Inside
ii. Touch Up Outside
iii. Touch Down
iv. Touch Cancel

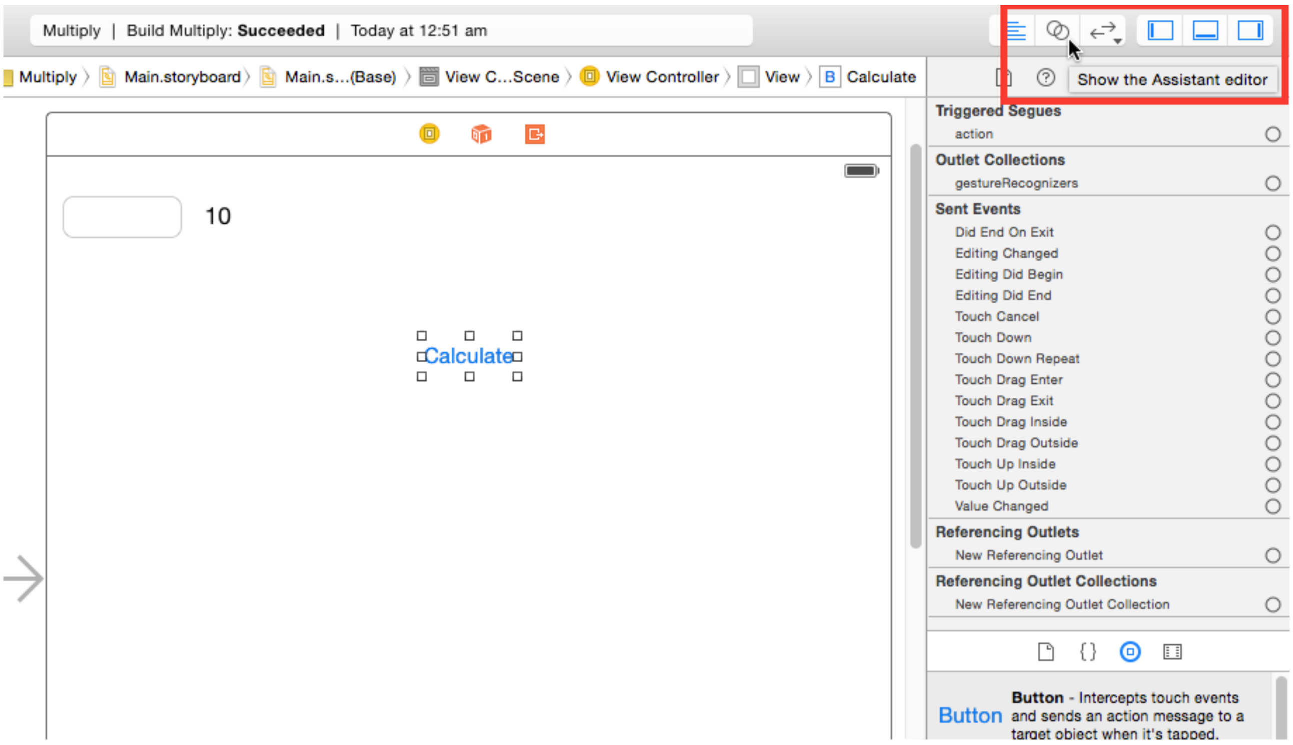
You can easily identify when a particular event is fired by just reading its name. When a user initially touches a finger on a button, for example, the Touch Down Inside event is triggered; Touch Up Inside is sent when the user lifts a finger off the button while still within the bounds of the button's edges. If the user has dragged a finger outside the button before lifting the finger, effectively cancelling the touch, the Touch Up Outside event is triggered.

For a simple button tap the correct event to wire up is : Touch Up Inside. Note that Xcode automatically suggests the default event for any control when you create the IBAction connection for it.

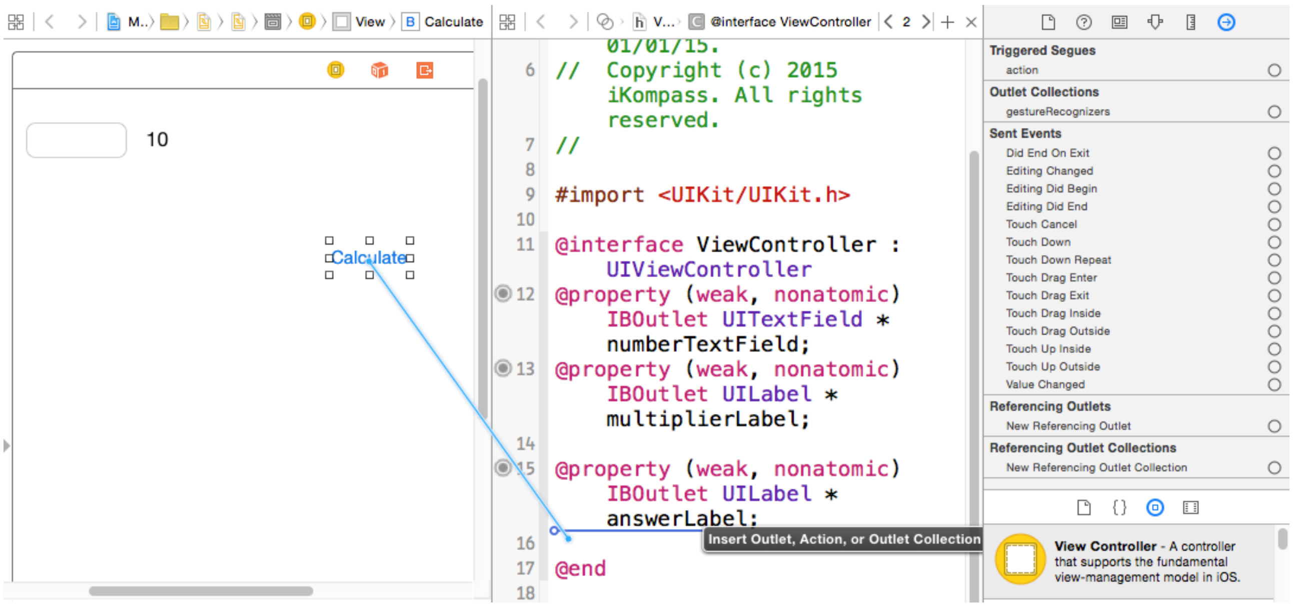
Let's create our first IBAction method of this exercise.

1. Select Main.storyboard in Navigation panel.

2. Select the Calculate button and open the Assistant Editor by clicking the Show the Assistant Editor button.

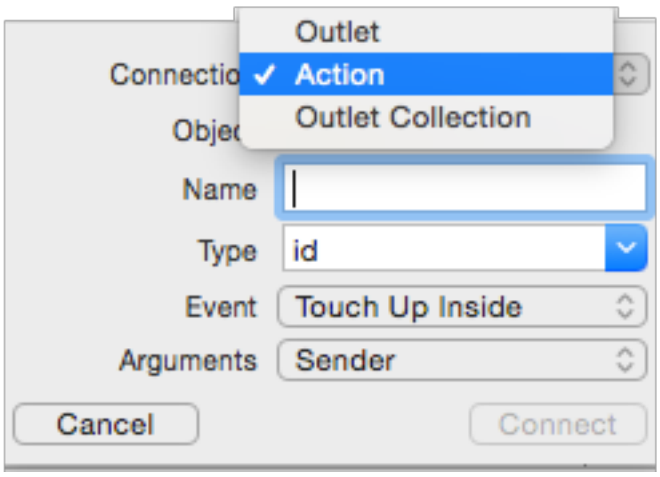

Multiply | Build Multiply: Succeeded | Today at 12:51 am
Multiply › Main.storyboard › Main.s...(Base) › View C...Scene › View Controller › View › Calculate
Show the Assistant editor
10
Calculate
Triggered Segues
action
Outlet Collections
gestureRecognizers
Sent Events
Did End On Exit
Editing Changed
Editing Did Begin
Editing Did End
Touch Cancel
Touch Down
Touch Down Repeat
Touch Drag Enter
Touch Drag Exit
Touch Drag Inside
Touch Drag Outside
Touch Up Inside
Touch Up Outside
Value Changed
Referencing Outlets
New Referencing Outlet
Referencing Outlet Collections
New Referencing Outlet Collection
Button
Button - Intercepts touch events and sends an action message to a target object when it's tapped.

3. While the "Ctrl" key is pressed, drag from the "Calculate" button and drop it on the assistant editor.



```
    01/01/15.
//  Copyright (c) 2015
    iKompass. All rights
    reserved.
//

#import <UIKit/UIKit.h>

@interface ViewController :
    UIViewController
@property (weak, nonatomic)
    IBOutlet UITextField *
    numberTextField;
@property (weak, nonatomic)
    IBOutlet UILabel *
    multiplierLabel;

@property (weak, nonatomic)
    IBOutlet UILabel *
    answerLabel;

@end
```

4. In the resulting popup, choose the connection type as :Action


Outlet
Connectio ✓ Action
Objec Outlet Collection
Name
Type id
Event Touch Up Inside
Arguments Sender
Cancel Connect

5. Enter the name as : onCalculateButtonPressed

Leave all other options unchanged and press the Connect button. The following entry will be made for you :

```
- (IBAction)onCalculateButtonPressed:(id)sender;
```

Save your changes and close the Assistant editor. Select the "ViewController.m" file in Navigation panel. You will notice that the method body is automatically created by Xcode for us. Now, whatever code we write inside this method, that will get executed whenever a user action invokes "Touch Up Inside" event.

Adding business logic to the method

We have to :

1. Read value from the numberTextField
2. Read value from the multiplierLabel
3. Multiply the value obtained from numberTextField with the multiplierLabel's value.
4. Assign the result into answerLabel's text value

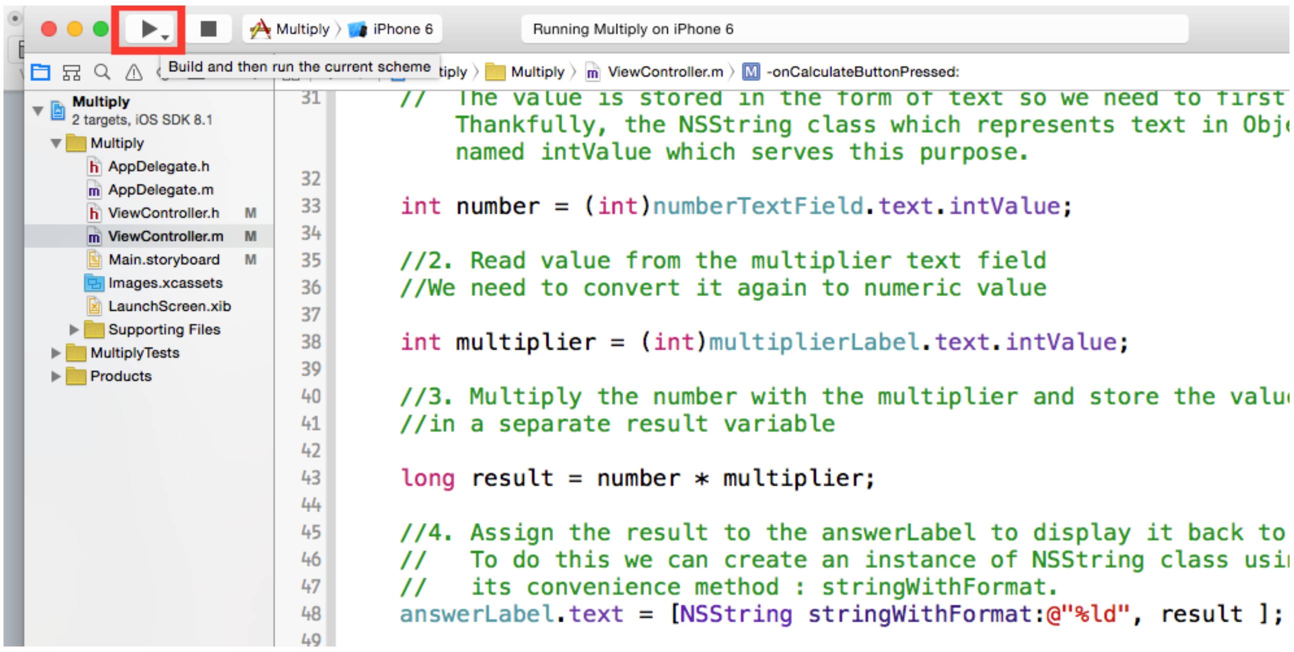
Before we do any of it, we have to understand that text values stored in the labels or textfields are stored in string format which is represented by Foundation framework's class NSString. However, we are talking about numeric calculation which requires numbers as operands. Thus, we must first convert the text values to numeric values. Fortunately, NSString class has a method which can do this job for us.

We also know that the IBOutlet properties are connected back to the controls on the scene and can be used to read and manipulate the properties of their respective controls. Using all this information, let's add the following code :

```
- (IBAction)onCalculateButtonPressed:(id)sender {
    //1. Read value from the text field
    //  The value is stored in the form of text so we need to first convert it into
integer. Thankfully, the NSString class which represents text in Objective-C has a
method named intValue which serves this purpose.

    int number = (int)numberTextField.text.intValue;

    //2. Read value from the multiplier text field
    //We need to convert it again to numeric value

    int multiplier = (int)multiplierLabel.text.intValue;

    //3. Multiply the number with the multiplier and store the value
    //in a separate result variable

    long result = number * multiplier;

    //4. Assign the result to the answerLabel to display it back to the user
    //   To do this we can create an instance of NSString class using one of
    //   its convenience method : stringWithFormat.
    answerLabel.text = [NSString stringWithFormat:@"%ld", result ];

}
```

In the above code snippet, remember that text in green are comments and are only for the sole purpose of making this code more understandable. Any comment added by the user is ignored by the compiler.

Run your application by hitting "Command + R" or press the "Run" button.

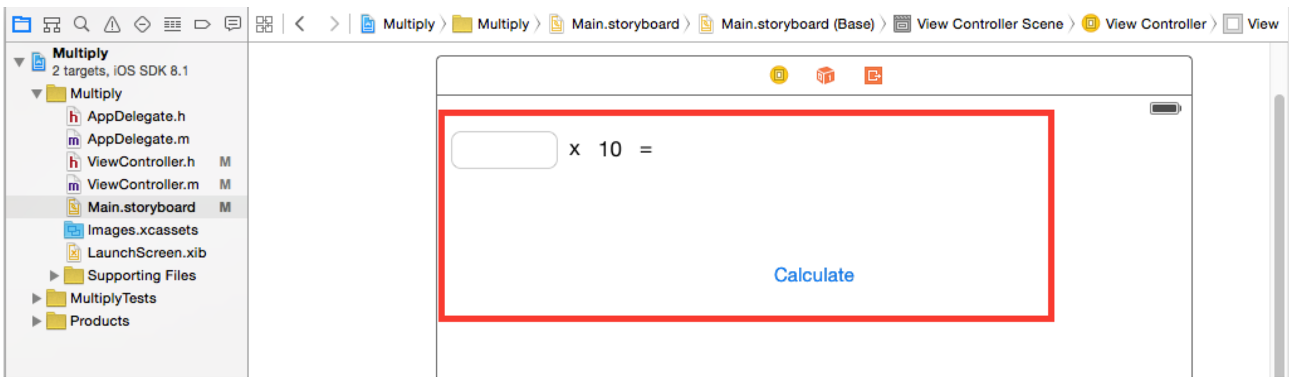


```
    //  The value is stored in the form of text so we need to first
        Thankfully, the NSString class which represents text in Obj
        named intValue which serves this purpose.

    int number = (int)numberTextField.text.intValue;

    //2. Read value from the multiplier text field
    //We need to convert it again to numeric value

    int multiplier = (int)multiplierLabel.text.intValue;

    //3. Multiply the number with the multiplier and store the valu
    //in a separate result variable

    long result = number * multiplier;

    //4. Assign the result to the answerLabel to display it back to
    //   To do this we can create an instance of NSString class usi
    //   its convenience method : stringWithFormat.
    answerLabel.text = [NSString stringWithFormat:@"%ld", result ];
```

Enter a valid numeric value in the text field and tap on "Calculate" button. It is important that for this exercise we stick to entering the numeric value in the textfield because we don't have an error handling mechanism in place.

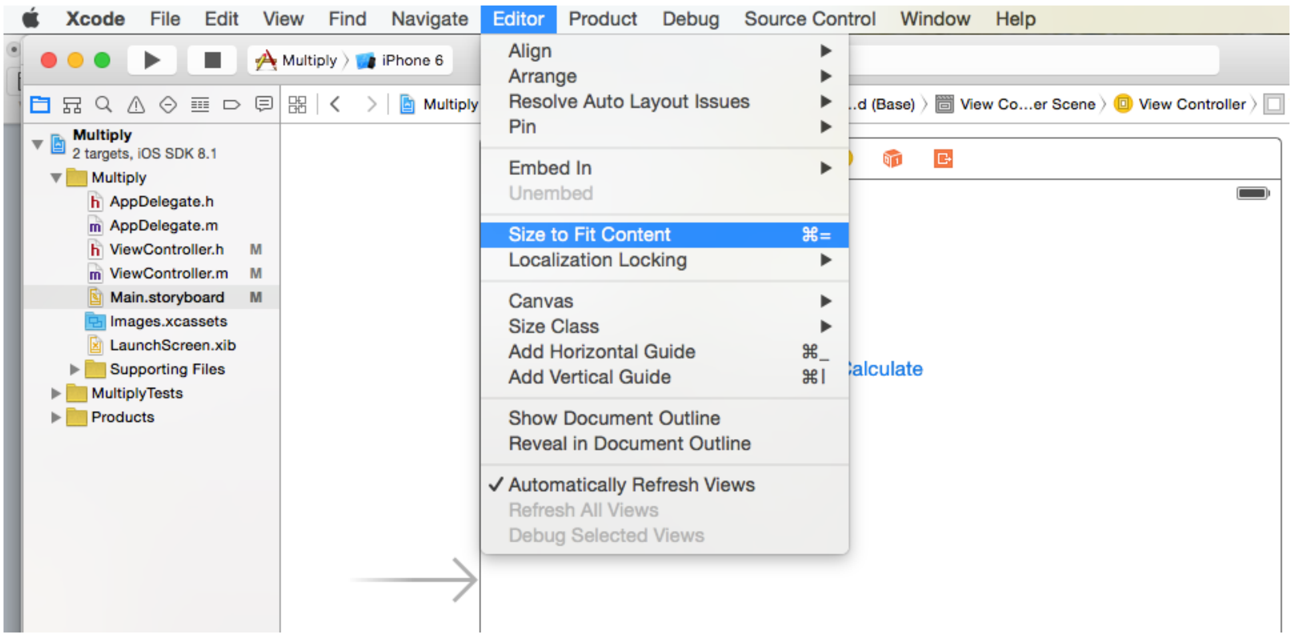
Step 6:- Completing the equation

If you run the application now, you will see the result coming fine but the whole of the equation is not displayed and the end-user (other than you) may not get any idea of what is happening. Let's add some more labels to make the equation appear in the following format :



Add two labels and place them properly. Use the blue guides to align them with each other. The labels containing "x", "10" and "=" are supposed to be sized properly so that they don't waste extra space on the screen and are only as wide as required to display their content. To do that, select your label and click :

Editor -> Size to Fit Content



These newly added labels are only for aesthetic purpose and they don't involve in any business logic, nor do we need to manipulate or read their properties. So it is alright to have them without an IBOutlet connection. Remember that we would create IBAction and / or IBOutlet connections for only those controls, where it is necessary.

Step 7:- Changing color of the ViewController

Colors in Objective-C are represented by the UIKit class : UIColor. It has many methods that allow you to create colors based on RGB values and other color parameters such as hue, saturation, alpha, etc. Besides, there are standard methods to get basic colors such as : Red, Green, White, etc.

We have to :

1. Change the color of ViewController to green if the result is equal to or greater than 20, or
2. If the result is less than 20 we have to make it white.

Let's expand our "onCalculateButtonPressed" method to incorporate this requirement. We will do it by adding an "if…else" block. An "if…else" block is used to test conditions. Depending upon the output of the condition, true or false, the corresponding code block is executed. An if…else block looks like below:

```
if (condition){

        //will be executed if the condition evaluates to true
        ….
        ….
}else{

        //will be executed if the condition evaluates to false
        ….
        ….

}
```

The updated "onCalculateButtonPressed" method is as follows:

```
- (IBAction)onCalculateButtonPressed:(id)sender {
    //1. Read value from the text field
    //  The value is stored in the form of text so we need to first convert it into 
integer. Thankfully, the NSString class which represents text in Objective-C has a 
method named intValue which serves this purpose.

    int number = (int)numberTextField.text.intValue;

    //2. Read value from the multiplier text field
    //We need to convert it again to numeric value

    int multiplier = (int)multiplierLabel.text.intValue;

    //3. Multiply the number with the multiplier and store the value
    //in a separate result variable

    long result = number * multiplier;

    //4. Assign the result to the answerLabel to display it back to the user
    //   To do this we can create an instance of NSString class using one of 
    //   its convenience method : stringWithFormat.
    answerLabel.text = [NSString stringWithFormat:@"%ld", result ];

    //5. Check if the result is equal to or greater than 20
    if(result >= 20){
        //6. If the result is >= 20 then create a green color
        UIColor *bgcolor = [UIColor greenColor];
        //7. Set the ViewController's background color to green
        //   In order to access ViewController itself, we can use the 
        //   "self" pointer. A "self" pointer is a speical pointer in that it always points to 
the current class instance. Using the self pointer, we will access the underlying view 
of the ViewController and set its background color to green
        self.view.backgroundColor = bgcolor;
    }else{
        //8. If result is < 20, then set the background color to white.
        UIColor *bgcolor = [UIColor whiteColor];
        self.view.backgroundColor = bgcolor;
    }
}
```

We are first checking if the result is greater than or equal to 20 or not. If this condition evaluates to true, then we are creating an instance of UIColor :

```
        UIColor *bgcolor = [UIColor greenColor];
```

The ViewController's view property is auto-wired to the scene's main view. So we need to access it in order to get to its background color property. To do that we are using "self" pointer. A "self" pointer always points to the current object and thus has access to all its methods and properties. In this case, the "self" pointer actually is pointing to the ViewController itself. Thus the following construct, sets the background color of the underlying view:

```
self.view.backgroundColor = bgcolor;
```

Similarly we are setting the background color of the view to white, in case the result is less than 20.

Step 8:- Show fizz/buzz/fizzbuzz/number result in the answersLabel as per the following rules :

i. If the calculated result is a multiple of 3, make answerLabel's text: fizz
ii. If the calculated result is a multiple of 5, make answerLabel's text: buzz
iii. If the calculated result is a multiple of 3 and 5, make answerLabel's text: fizz buzz
iv. Otherwise continue to show the number result, as before

In the previous step we had only two conditions to evaluate so a simple if…else block worked for us. But in here, we have to take care of multiple conditions so a simple if…else block would not do much. We will have to use an enhanced "if..else if..else" block instead. An if..else if…else block may have as many "else if" blocks as are required to suit the number of conditions and it looks like :

```
if (condition){
        //will be executed if the condition evaluates to true
        ….
        ….
}else if(condition 1){
        //will be executed if the condition 1 evaluates to true
        ….
        ….

}else if(condition 2){
        //will be executed if the condition 2 evaluates to true
        ….
        ….

}
.
.
.
else{
        //will be executed if none of the conditions evaluates to true
        ….
        ….

}
```

As soon as any of the condition is fulfilled, the rest of the blocks and their conditions are not evaluated and the code within the first fulfilling block is executed. The control of the execution then comes out of the entire construct. So it is very important for us that we keep the more specific conditions at the top and the generic ones at the bottom.

Let's add the required logic in our "onCalculateButtonPressed" method and change the code where we assign the result to the answerLabel (labelled as 4 in the method) as follows :

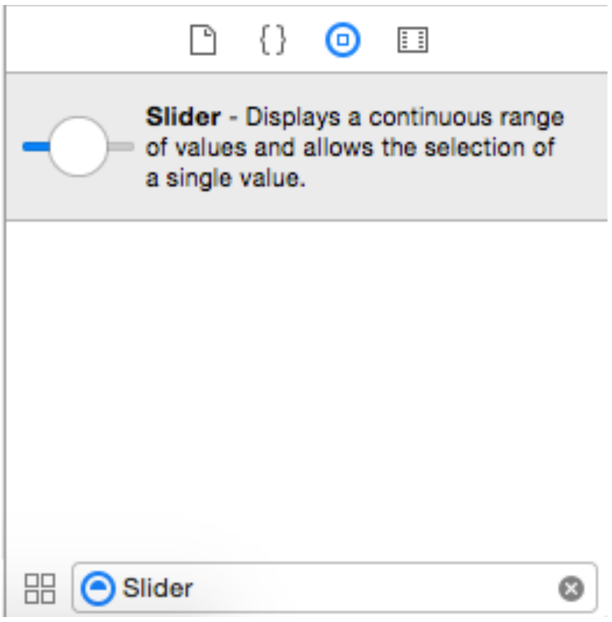
```
//4. Display fizz buzz if the result is a multiple of both 3 & 5
    //   else if the result is only a multiple of 3 then display fizz
    //   else if the result is a multiple of 5 alone, then display buzz
    //   else display the numeric result value. To do this we can create an instance of NSString class using one of
    //   its convenience method : stringWithFormat.
    if(result % 3 == 0 && result % 5 == 0){
        answerLabel.text = @"fizz buzz";
    }else if(result % 3 == 0){
        answerLabel.text = @"fizz";
    }else if(result % 5 == 0){
        answerLabel.text = @"buzz";
    }else{
        answerLabel.text = [NSString stringWithFormat:@"%ld", result ];
    }
```

Run you code and experiment with different values and see if the answerLabel is being updated properly or not.

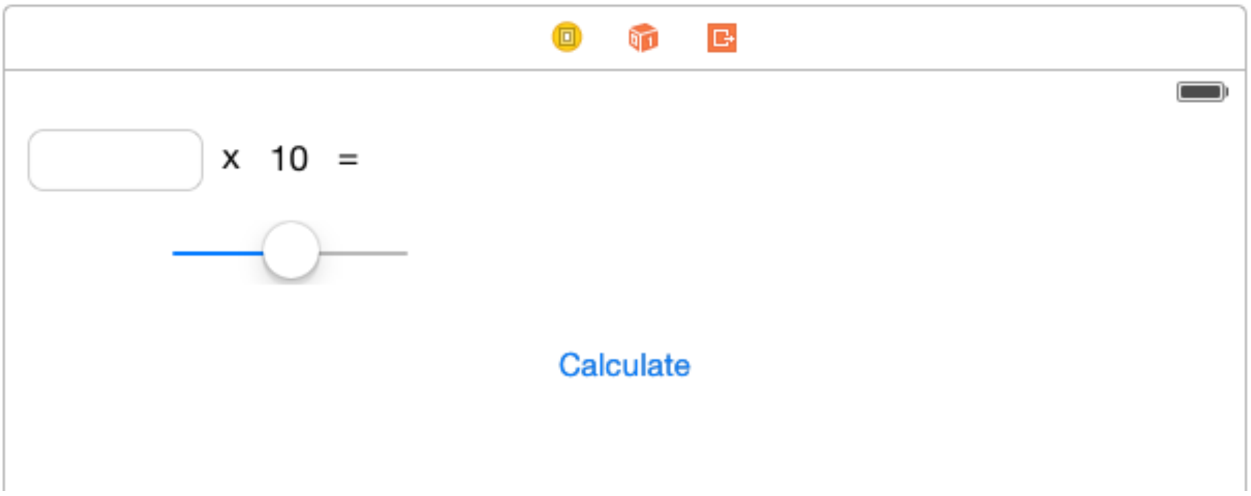
Step 9:- Using a UISlider to change the value of the multiplierLabel

Currently the multiplierLabel has a static value of 10. We want to give the user an option to change this value using a UISlider. Whatever the value of UISlider be, the same will be the value set in the multiplierLabel. However, let's limit the upper limit of the UISlider's value to 10 and lower limit to 0.

1. Select Main.storyboard in the Navigation panel.
2. Type in "Slider" in the Object library and it will filter out the Slider control for you.


Slider - Displays a continuous range of values and allows the selection of a single value.

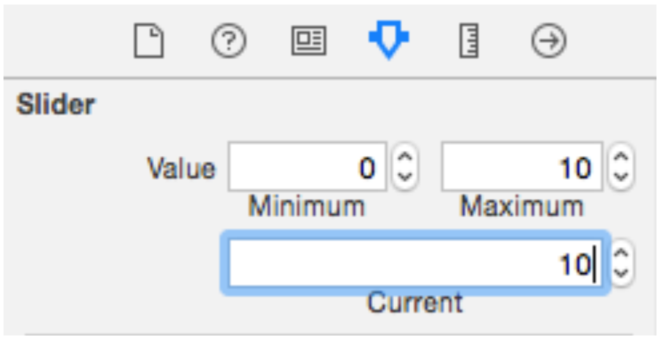
3. Drag and drop the Slider control and align it such that it appears just below the multiplier label.



4. Select the Slider control and open the Attributes inspector in the Utilities panel. Under the Value section set the following :

Minimum	:	0
Maximum	:	10

Current	:	10



Slider

Value: Minimum 0, Maximum 10

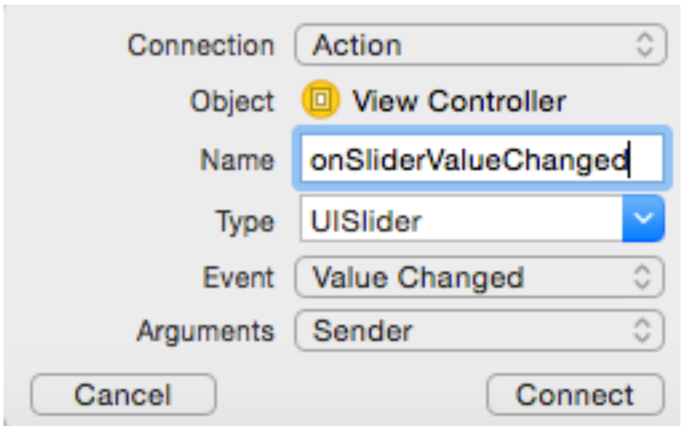
Current: 10

It is important to set the "Current value" of the slider to 10 as this is the value we are displaying initially in the multiplierLabel and it feels good to see the slider's head and the multiplierLabel both set to same value.

5. Whenever a user changes the slider's value by dragging its handle, the same needs to be updated in the multiplierLabel. For this, once again an IBAction wiring for the slider would be required and it should be done for the slider's event :Value Changed. A "Value Changed" event is fired automatically whenever the slider's value is changed.

Select the slider and open the Assistant Editor. "Ctrl-Drag" from the slider and drop it on the editor. Enter the following details in the Connection popup:

Connection	:	Action
Name	:	onSliderValueChanged
Type	:	UISlider
Event	:	Value Changed
Arguments	:	Sender



Connection: Action

Object: View Controller

Name: onSliderValueChanged

Type: UISlider

Event: Value Changed

Arguments: Sender

6. Close the Assistant editor and select ViewController.m file in the Navigation panel.
7. In the "onSliderValueChanged" method, add the logic to update the multiplierLabel's text value as per the slider's value.

```
- (IBAction)onSliderValueChanged:(UISlider *)sender {
    multiplierLabel.text = [NSString stringWithFormat:@"%d", (int)sender.value ];
}
```

UISlider's value is floating-point type but our multiplier is displaying integer value. Hence, first we need to typecast the floating-point value to integer as : (int)sender.value . Then using the integer format specifier (%d), we are creating an instance of NSString which in turn is being assigned to multiplierLabel's text property.

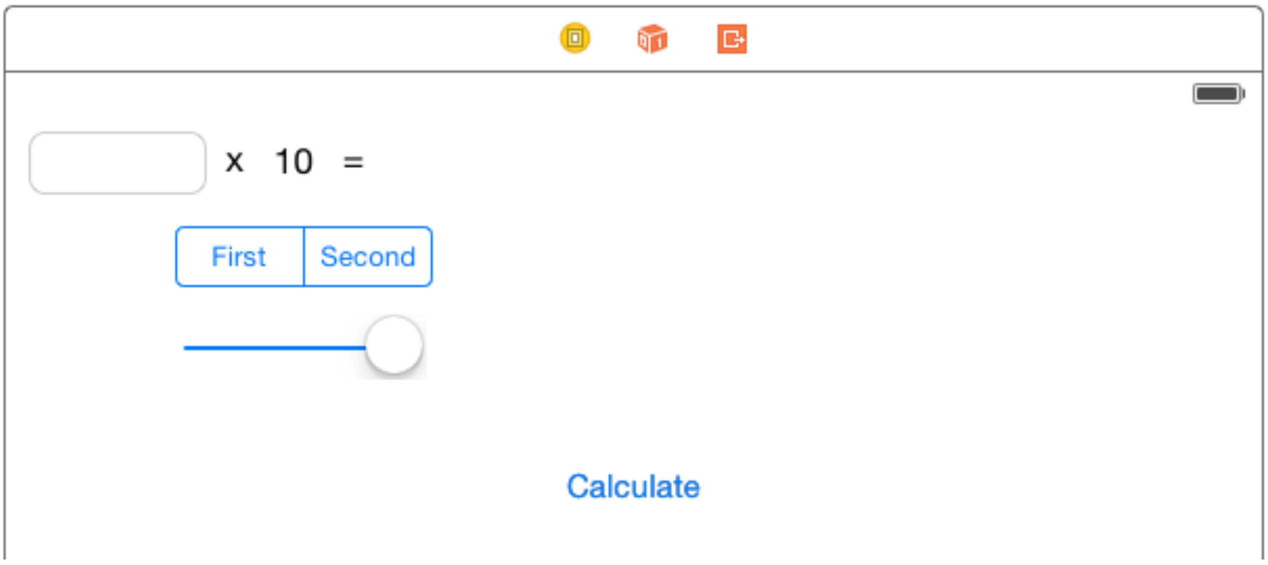
Run the application and move the slider, you will see that the value displayed in the multiplierLabel also changes.

Step 10:- Using the UISegmentedControl to change the operator and calculate the result accordingly.

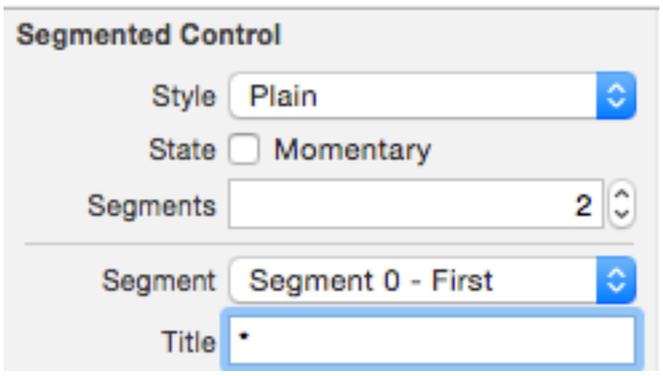
Let's give an option to the user to choose an operator. We will provide the user a UISegmentControl, using which he can decide whether to do multiplication or subtraction.

1. Select Main.storyboard in the Navigation panel.
2. Type in Segment in the Object Explorer's search box. You should see the Segmented Control being filtered out and displayed in the panel.

3. Drag and drop it on the scene. Place it above the Slider and align the two controls vertically. You may need to shift the Calculate button a little down. At the end, your scene should appear as follows:



4. Select the segment control and open the Attributes inspector panel. Ensure that Segment 0 is selected in the Segment drop-down combo. Change its title to "*".



Segmented Control

Style: Plain

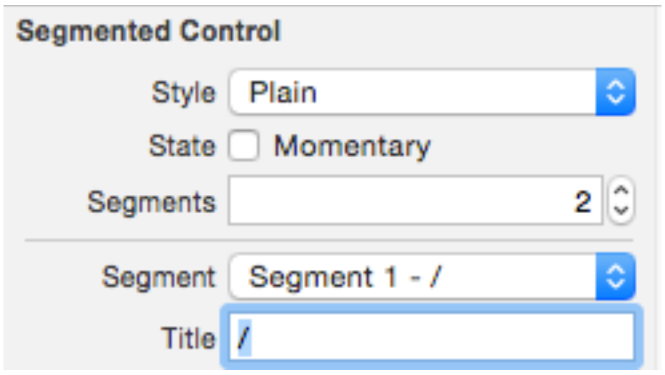
State: ☐ Momentary

Segments: 2

Segment: Segment 0 - First

Title: *

5. Select Segment 1 in the Segment drop-down combo and change the title to "/".

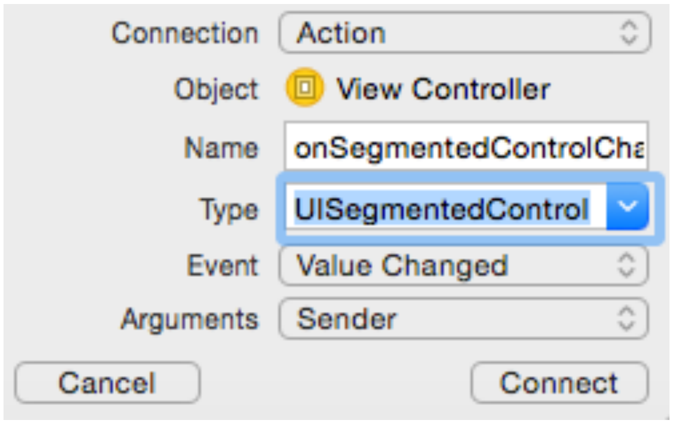


Segmented Control
Style: Plain
State: ☐ Momentary
Segments: 2
Segment: Segment 1 - /
Title: /

6. We want the operator's label to change and display the current symbol as per the selected segment of the Segmented control. We need two things here :
   i. an IBOutlet for the operator label
   ii. an IBAction for the segmented control

Let's first select the operator label and create an IBOutlet for it. Let's call it : operatorLabel. Forget not to synthesize the property.

Next select the segmented control and create an IBAction connection for it and call it : onSegmentedControlChanged. Ensure that you have set the Type to : UISegmentedControl from the default value : id



Connection: Action
Object: View Controller
Name: onSegmentedControlCha
Type: UISegmentedControl
Event: Value Changed
Arguments: Sender

The following entries should be created in the ViewController's header file :

```
@property (weak, nonatomic) IBOutlet UILabel *operatorLabel;
```

```
- (IBAction)onSegmentedControlChanged:(UISegmentedControl *)sender;
```

7. Close the Assistant editor if not already done, and select the ViewController's implementation file and implement the "onSegmentedControlChanged" method:

```
- (IBAction)onSegmentedControlChanged:(UISegmentedControl *)sender {
    //1. Check the selected segment index
    if(sender.selectedSegmentIndex == 0){
        //2. if the selected segment index is 0 then
        //   set the operatorLabel's text to *
        operatorLabel.text = @"*";
    }else{
        //3. else set the operatorLabel's text to /
        operatorLabel.text = @"/";
    }
}
```

Since we have two operators now and user can apply any one of them, so we need to consider the following :

1. when the user has chosen "multiplication", we need to multiply the values of numberTextField and the multiplier.
2. when the user chooses "division", we need to do the division operation. However, the multiplier's value is being set from the slider where 0 is a possibility and division by 0 is illegal, so we need to take care of that scenario. Let's show the text : Invalid whenever there is an attempt made to divide by 0 and change the view's background color to red.

Let's update the onCalculateButtonPressed method accordingly:

```
//3. Multiply or divide as per the operator and store the value
    //in a separate result variable as long as the operations are valid

    long result = 0;
    if(operatorLabel.text == @"*"){
        result = number * multiplier;
    }else{
        if(multiplier == 0){
            answerLabel.text = @"Invalid";
            //let's also set the UIColor of view to red to show an error
            self.view.backgroundColor = [UIColor redColor];
            //since it is an invalid operation, let's skip rest of the piece of code.
            return;
        }else{
            result = number / multiplier;
        }
    }
```

The full method looks as follows :

```
- (IBAction)onCalculateButtonPressed:(id)sender {
    //1. Read value from the text field
    // The value is stored in the form of text so we need to first convert it into
integer. Thankfully, the NSString class which represents text in Objective-C has a
method named intValue which serves this purpose.

    int number = (int)numberTextField.text.intValue;

    //2. Read value from the multiplier text field
    //We need to convert it again to numeric value

    int multiplier = (int)multiplierLabel.text.intValue;

    //3. Multiply or divide as per the operator and store the value
    //in a separate result variable as long as the operations are valid

    long result = 0;
    if(operatorLabel.text == @"*"){
        result = number * multiplier;
    }else{
        if(multiplier == 0){
            answerLabel.text = @"Invalid";
            //let's also set the UIColor of view to red to show an error
            self.view.backgroundColor = [UIColor redColor];
            //since it is an invalid operation, let's skip rest of the piece of code.
            return;
        }else{
            result = number / multiplier;
        }
    }

    //4. Display fizz buzz if the result is a multiple of both 3 & 5
    //   else if the result is only a multiple of 3 then display fizz
    //   else if the result is a multiple of 5 alone, then display buzz
    //   else display the numeric result value. To do this we can create an instance of 
NSString class using one of
    //   its convenience method : stringWithFormat.
    if(result % 3 == 0 && result % 5 == 0){
        answerLabel.text = @"fizz buzz";
    }else if(result % 3 == 0){
        answerLabel.text = @"fizz";
    }else if(result % 5 == 0){
        answerLabel.text = @"buzz";
    }else{
        answerLabel.text = [NSString stringWithFormat:@"%ld", result ];
    }

    //5. Check if the result is equal to or greater than 20
    if(result >= 20){
        //6. If the result is >= 20 then create a green color
        UIColor *bgcolor = [UIColor greenColor];
        //7. Set the ViewController's background color to green
        //   In order to access ViewController itself, we can use the
        //   "self" pointer. A "self" pointer is a speical pointer in that it always points to 
the current class instance. Using the self pointer, we will access the underlying view 
of the ViewController and set its background color to green
        self.view.backgroundColor = bgcolor;
    }else{
        //8. If result is < 20, then set the background color to white.
        UIColor *bgcolor = [UIColor whiteColor];
        self.view.backgroundColor = bgcolor;
    }

}
```

Step 11:- Hide the keyboard when the Calculate button is tapped.

A keyboard is automatically sprung into action by iOS whenever the focus enters a control which accepts keyboard input. However, it is not hidden automatically. Usually we need to handle such a situation. We will do that whenever user presses the Calculate button.

Add the following line of code just after the opening curly braces of the method: onCalculateButtonPressed:

```
[numberTextField resignFirstResponder];
```

The control which gets the focus becomes the first responder, that is, it becomes its responsibility to handle user interaction. By calling the resignFirstResponder method we relive the control from this responsibility. The moment we relieve the textfield, the associated keyboard also is removed by iOS.

BONUS

The numberTextField opens up a keyboard which allows all sorts of input possible. But we expect only the numeric values to be input by the user. We can't expect it all the time unless we enforce it. The simplest way to enforce it is to give the user a number keyboard (instead of a universal / default keyboard).
To do this, open the Main.storyboard and select the numberTextField. Open Attributes inspector and scroll down until you find : Keyboard Type option. Change the keyboard type to : Number Pad

Run your app and tap the numberTextField. A number pad comes instead of default keyboard.