LOREM IPSUM

Introducti on to

Dolor Set Amet

Section 1

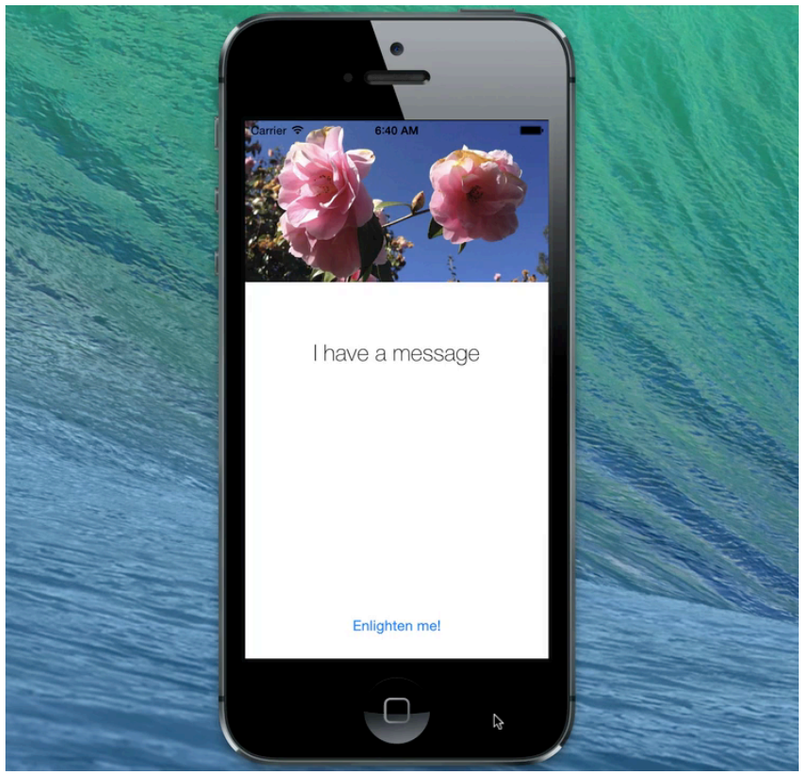
Task 01 - Single Page App

INTENDED LEARNING OUTCOMES

1. Open the development tool XCode
2. Create a single view application for iPhone
3. Add UI components to the root view using StoryBoard
4. Identify the Model, View and Controller in the MVC paradigm
5. Create an “Action” (event handler) for a touch event
6. Create an “Outlet” (pointer) to enable access to UI Components
7. Create a controller class
8. Create a model object (type NSArray) and populate with initial data
9. Describe what is meant by “lazy loading”
10. Write a read accessor (getter) to lazily create a model object
11. Describe the role of the ViewControler in the MVC paradigm

Getting started

The first task is to create an app with a single view. Tap the video below to preview this. Pinch-zoom to view full-screen.



Single View Application that responds to touch and rotation events

TASK - CREATING A SINGLE VIEW APPLICATION

Follow the demonstration from the tutor.

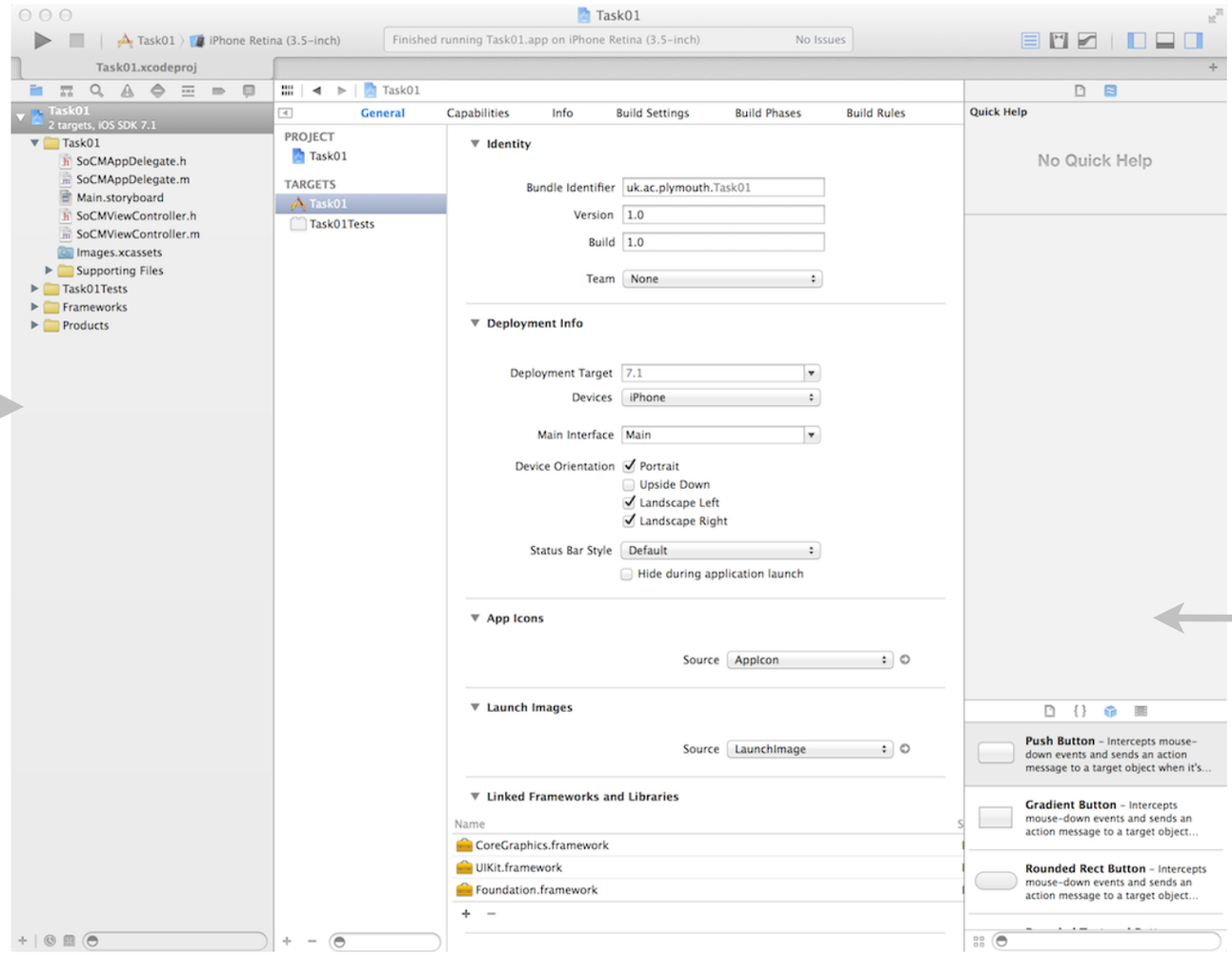
Now do you the following:

- Open XCode
- Create a single view application and save on the desktop as TASK-01
- Run the simulator

Getting started

The first task is to create an app with a single view. Tap the video below to preview this.

Pinch-zoom to view full-screen.



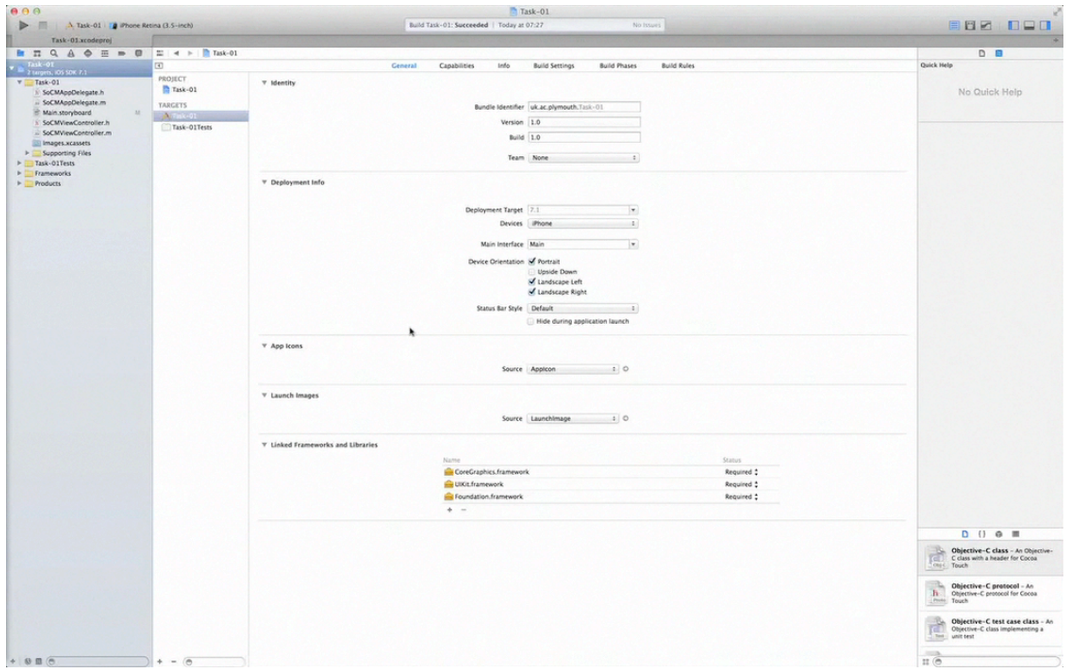
TASK 01-01 ADDING AND LAYING OUT UI ELEMENTS

Follow the demonstration from the tutor.

Now do you the following:

- Open XCode
- Create a single view application and save on the desktop as TASK-01
- Run the simulator
- Watch Movie 1.1
    - Add the button
    - Add the label
    - Add the image
- Run the simulator and test

Movie Introduction to iOS7 Development.1 Adding elements to the root view



Here we use StoryBoard in XCode to lay out a simple view. We add a button and a label. However, as you will observe, the view does not (yet) adapt to changes in device orientation

TASK 01-02 ADDING AN IMAGE

Watch Movie 1.2

Repeat the task.

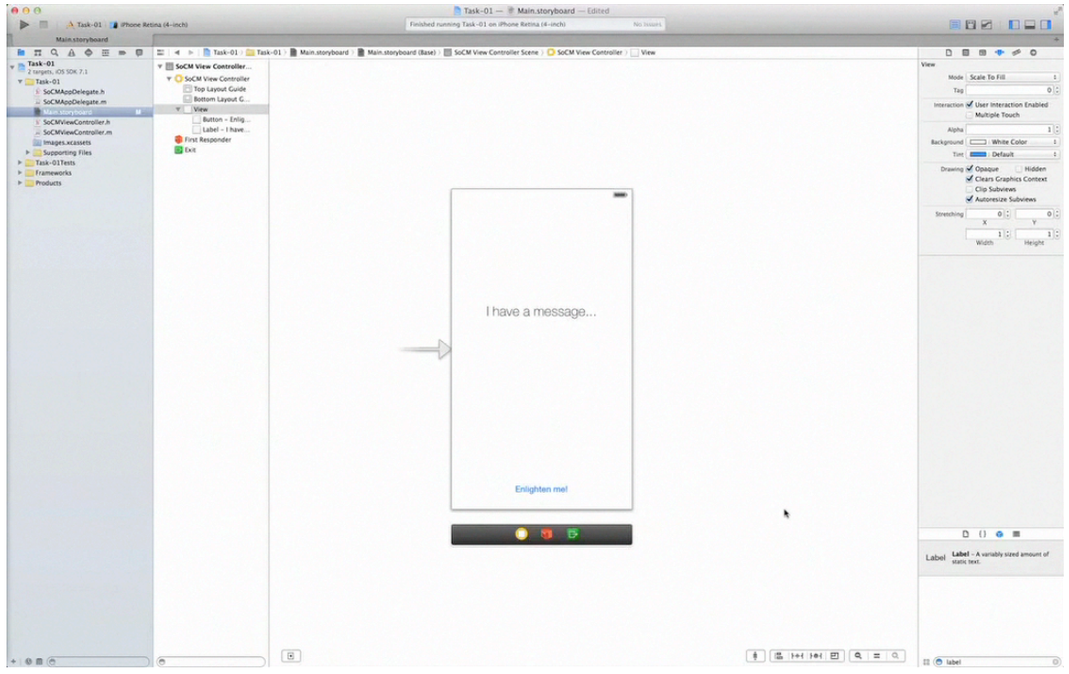
Test using different device orientations.

What Class is the image? Use the inspector to find out

- **A.** NSImage
- **B.** UIImageView
- **C.** UIImage
- **D.** UIViewController

Check Answer

Movie Introduction to iOS7 Development.2 Importing an image and placing as a sub-view



An image (one for retina display) is imported into an asset catalogue and played as a subview

TASK 01-03 USING AUTOLAYOUT TO MANAGE CHANGES IN CONTAINER VIEW DIMENSIONS

Watch movie 1.3

Repeat the task

Test using different orientations and device sizes

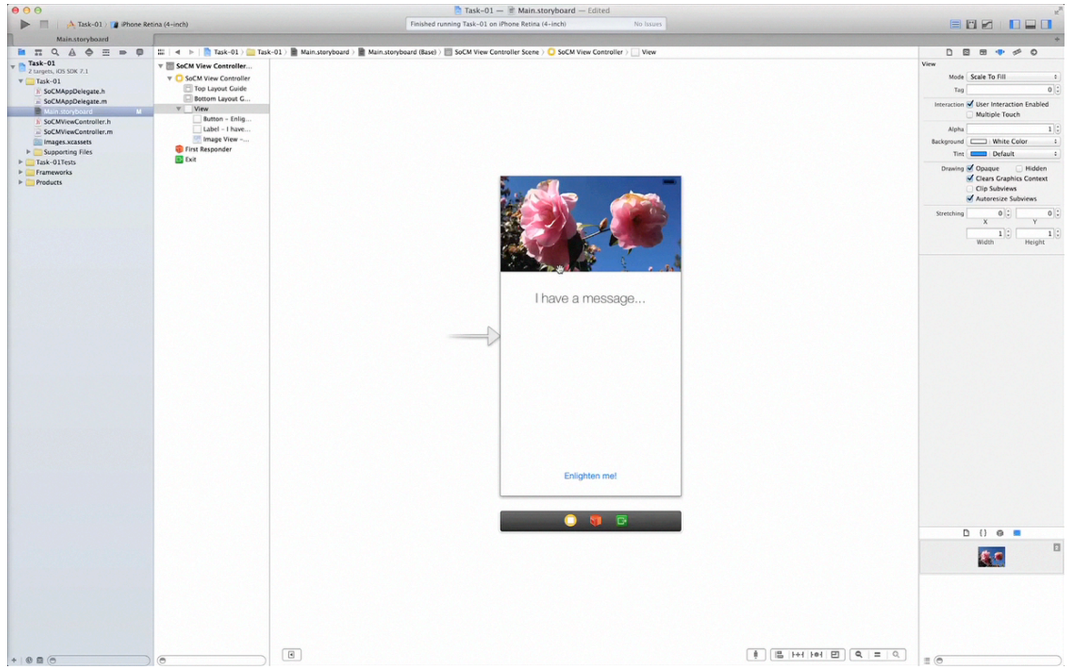
Question 1 of 2

What class is the top level container view?

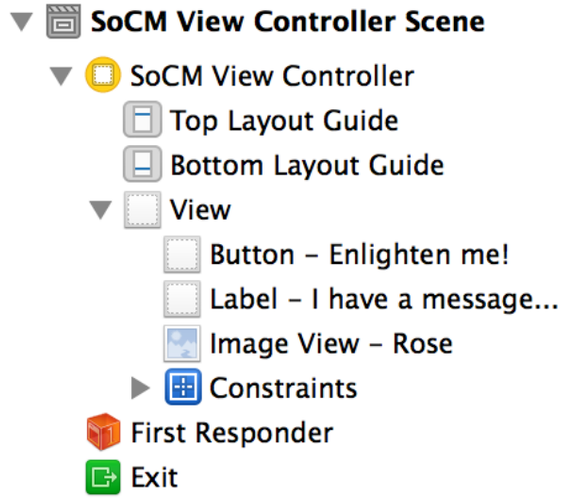
- **A.** UIViewController
- **B.** UIImageView
- **C.** UIView
- **D.** UILayer

◀ Check Answer ▶

Movie Introduction to iOS7 Development.3 Using AUTOLAYOUT to manage changes in parent view size



In this task, we use AUTOLAYOUT to add constraints and rules. These rules define what happens to each of the "subviews" when their "parent/ container view" changes size. See below to see the view hierarchy.



The view hierarchy can be observed and manipulated in StoryBoard. You can use this presentation instead of the WYSIWYG presentation.

TASK 01-04 RESPONDING TO EVENTS USING ACTIONS

In this task, you are going to hook up an event handler.

When the button is tapped, a message will appear in the console.

Watch movie 1.4

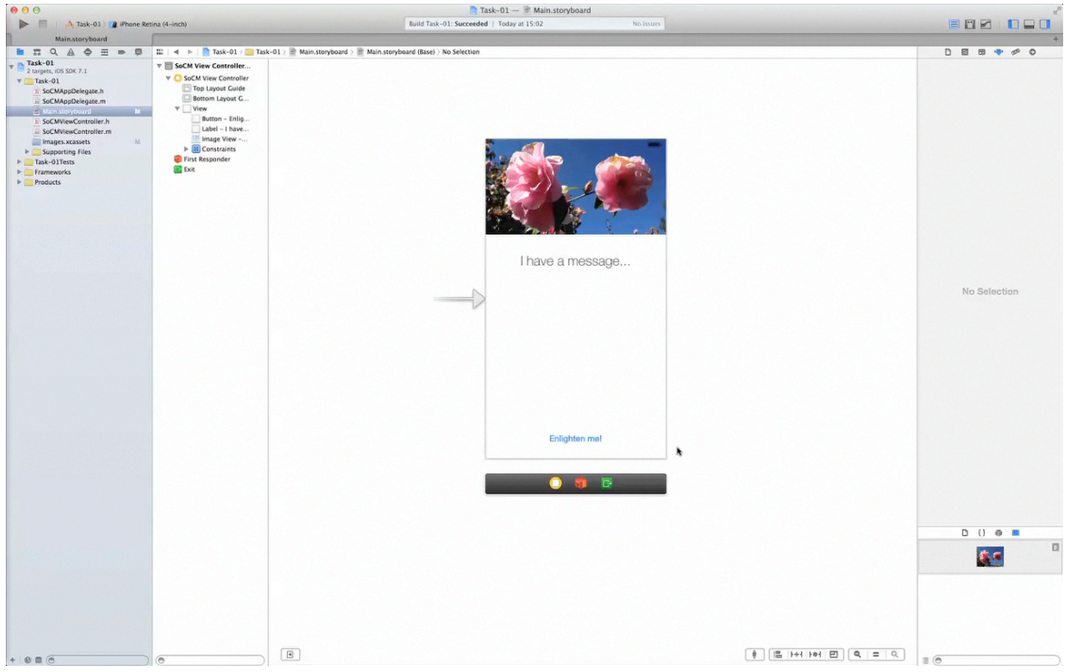
Now hook up your button to write a message in the console.

You event handler should look something like this:

```
- (IBAction)doTap:(id)sender {
    NSLog(@"You tapped the button");
}
```

The text `@"You tapped the button"` refers to a static string of type `NSString`. Anything prefixed with an @ symbol is actually an Objective-C object. Memory management and scope of these objects are handled for you.

Movie Introduction to iOS7 Development.4 Adding an Action (event handler)



*Methods that return a type IBAction are designated as event handlers, or "actions". Right click on a control and these methods will be listed. You can **declare** methods in either the header file (publicly visible) or the implementation file (hidden). You always **define** methods in the implementation file.*

TASK 01-05 REFERENCING OBJECTS USING OUTLETS

In this task, you are going to change the label text. To do this, **you need some reference to the label**.

Watch movie 1.5

Create an **outlet** for your text label

Modify your code to change the text when the button is tapped (as shown in the video).

IMPORTANT:

How do the view objects become instantiated?

In some GUI tools, it's not uncommon to auto generate source code. In XCode (and increasingly on other platforms as well) the GUI tool "StoryBoard" simply generates an XML file that describes the view objects and their relationships.

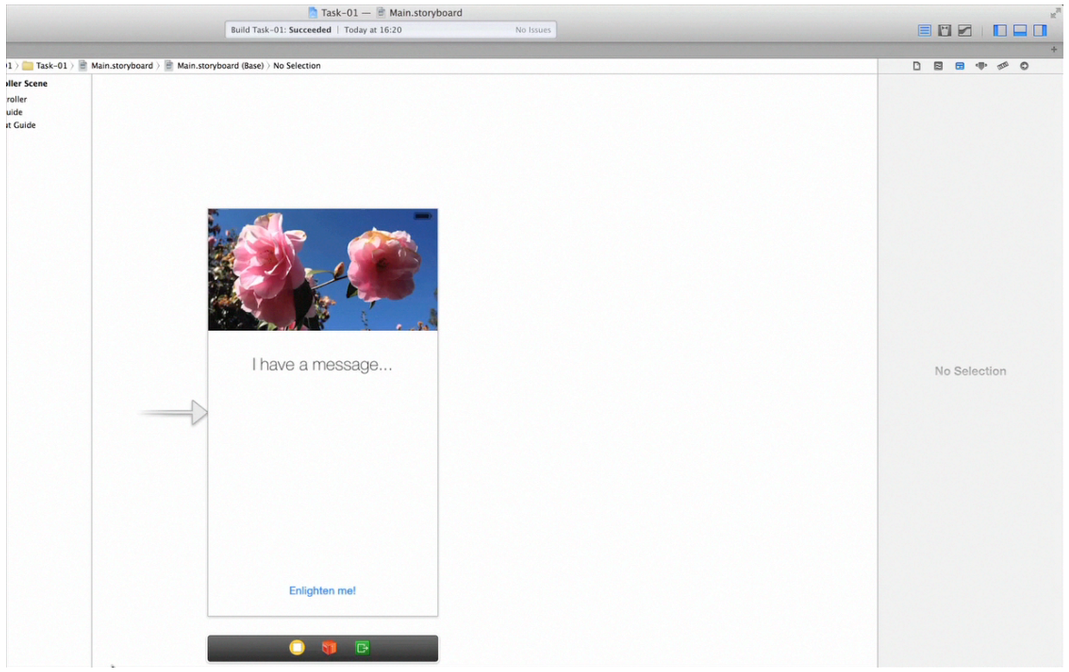
When the project is built, the view objects described in the XML file are instantiated (they actually exist in memory, just as if you'd created them in code) and archived to disk. This is analogous to serialization in Java.

When the view controller decides to load it's view, these objects are loaded from disk and unarchived. All the actions and outlets are then hooked up automatically.

In the view controller, the point at which this has occurred is in the method -(void)**viewDidLoad**.

Are all objects instantiated in this way view objects?

Movie Introduction to iOS7 Development.5 Creating references to objects using Outlets



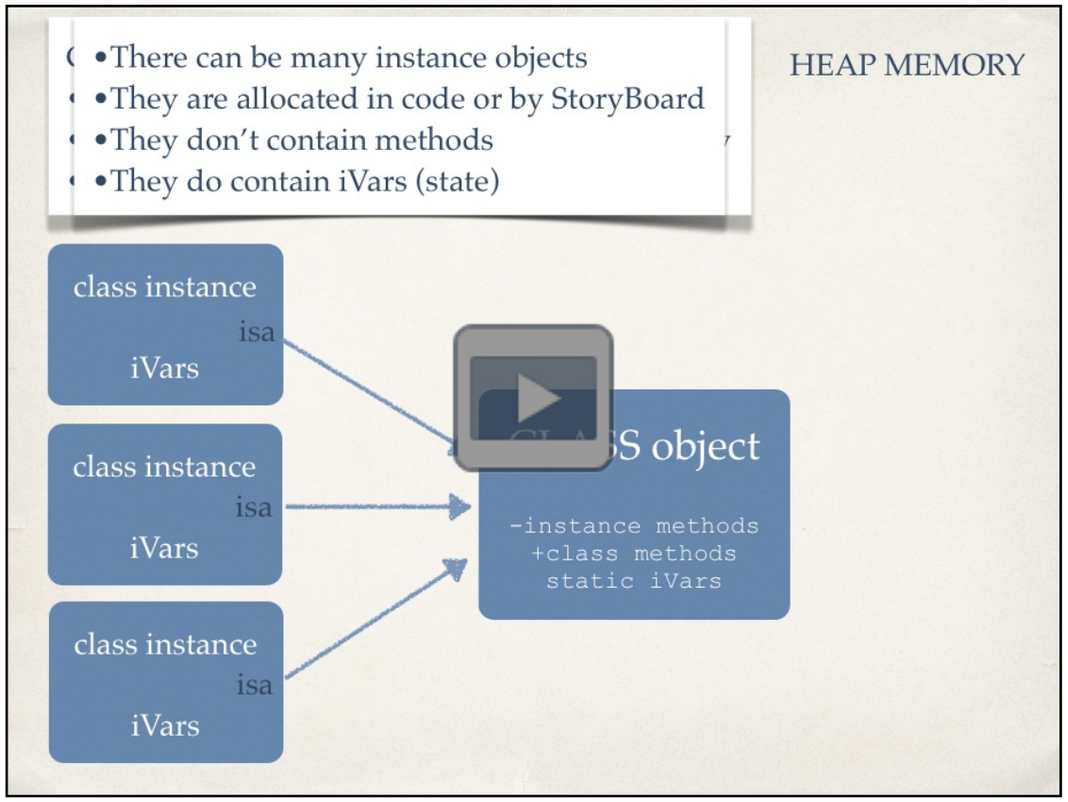
An outlet is a C pointer (an address of an object in memory). Once the views are loaded into memory, it will point to its respective view object, thus making it accessible using code.

The keyword IBOutet is actually defined as a blank. It is purely a flag so that StoryBoard knows it is a pointer to a view object. Right click on the view controller, and all Outlets and Actions will be listed.

A BIT OF THEORY

The tutor will now give a few mini lectures. For your convenience, these are also included here.

Lecture Introduction to iOS7 Development.1 Closer look at ObjectiveC classes



Objective C is an object-orientated extension of the C programming language. Both head and implementation files are needed to define and declare a class. Categories are an interesting feature of the language

Review Introduction to iOS7 Development.1 ObjectiveC classes

Question 1 of 2

Which of the following TRUE?

- **A.** A class contains instance variables
- **B.** A class instance contains all the methods needed to operate on the iVars
- ✓ **C.** All instances of a class contain their own instance variables, but no methods

Check Answer

MODEL VIEW CONTROLLER

Lecture on Model View Controller in the context of iOS development.

Lecture Introduction to iOS7 Development.2 Lecture on MVC

MVC in the context of iOS development is presented here. Later you will use multiple controllers and also observe the model-stack.

Review Introduction to iOS7 Development.2 Model View Controller

Which of the following is TRUE?

- ✓ **A.** The controller creates and destroys the model object
- **B.** The view sends data to the model when it's updated
- **C.** In a data model stack, all controllers have access to the complete model
- **D.** The controller owns the view, but not the model

Check Answer

TASK 01-06 ADDING THE MODEL

The model in MVC is typically instantiated and destroyed by the controller. On this basis, controller is said to “own” the model.

In this next task, you are going to create a simple model object. The model will be an **immutable** array of immutable strings.

The top level object will be of type NSArray.

The strings will be of type NSString

Constructors?

ObjectiveC has no constructors as such. Initialization methods needs to be called explicitly. There are naming conventions for this purpose, but *they are not enforced* (we **will** return to this!) and it depends on how an object is instantiated.

viewDidLoad is an important method in the UIViewController lifecycle. It is called once the view objects are unarchived and the outlets + actions are all hooked up. *viewDidLoad is therefore usually where we initialize the view*

Modify your controller code to match that of Listing 01-06.

Build and run.

Click the button to cycle through the sayings.

Code Listing 01-06

```
#import "SoCMViewController.h"

@interface SoCMViewController ()

@end

@implementation SoCMViewController {
    NSArray* arrayOfWisdom;
    NSUInteger index;
}

- (void)viewDidLoad
{
    [super viewDidLoad];
    // Do any additional setup after loading the view, typically from a nib.
    if (arrayOfWisdom == nil) {
        arrayOfWisdom = @[ @"May the force be with you",
                           @"If things don't change, they will stay as they are",
                           @"A day without sunshine is like night",
                           @"Why does the sun not come out at night when it would be more useful?",
                           @"if you have noticed this notice you will have noticed that this notice is
not worth noticing",
                           @"Always remember you're unique, just like everyone else",
                           @"It's comforting to reminisce with people you don't know"
                           ];
```

TASK 01-07 ACCESSORS

The model so far is a NSArray, and we're working directly with this array object. The controller is keeping a count of which saying is to be displayed using an integer 'index'.

However, for many reasons, it is advisable to create accessors when working with model objects. One of these reasons is that it allows us to use **lazy loading** to initialize out model objects.

Watch Movie 1.7 (**note** - this contains audio)

Review Introduction to iOS7 Development.3 Accessors

Question 1 of 4

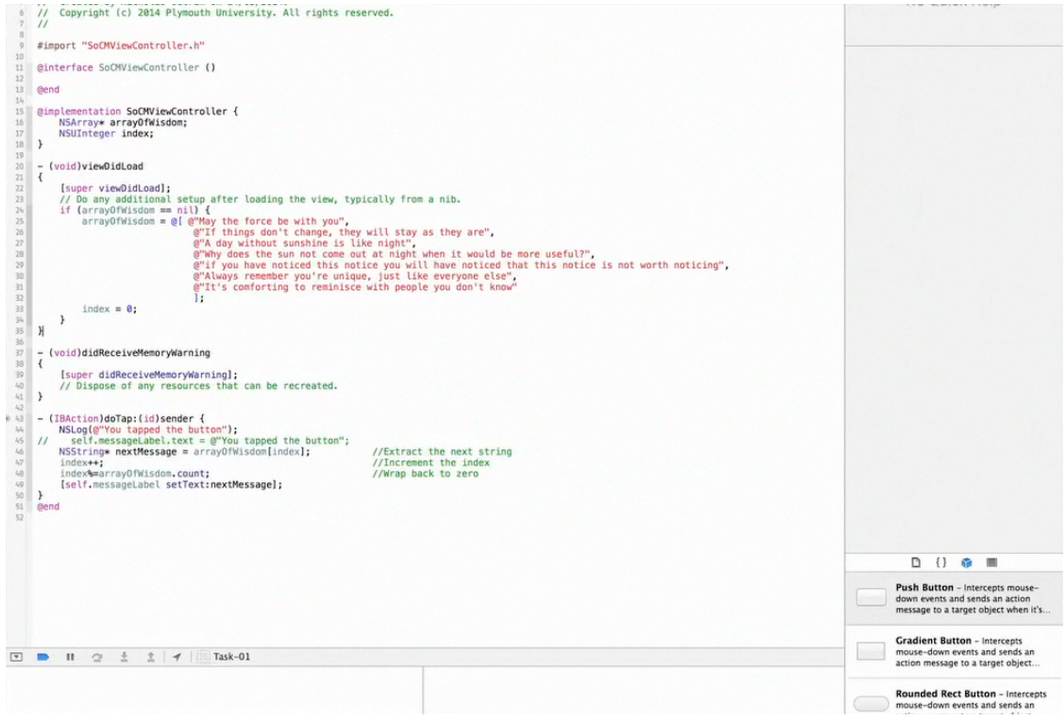
If you had a iVar NSString* name, what would be the name of the setter?

- **A.** setname
- **B.** SetName
- **C.** setName ✓
- **D.** name

◀ Check Answer ▶

Now repeat the steps, and modify your code to use an accessor to access the model object `arrayOfWisdom`. You should use lazy-loading as shown in the video.

Movie Introduction to iOS7 Development.6 Writing Accessors to perform Lazy Loading



Lazy loading is such a common technique, it is worth learning early.

TASK 01-08 PROPERTIES

In the previous task, we created accessor methods. The good news is that this task can be automated using ‘properties’ and the @**property** keyword.

Watch Movie 1.8

Create properties for both `index` and `arrayOfWisdom`

Review Introduction to iOS7 Development.4 Review on properties

Question 1 of 2

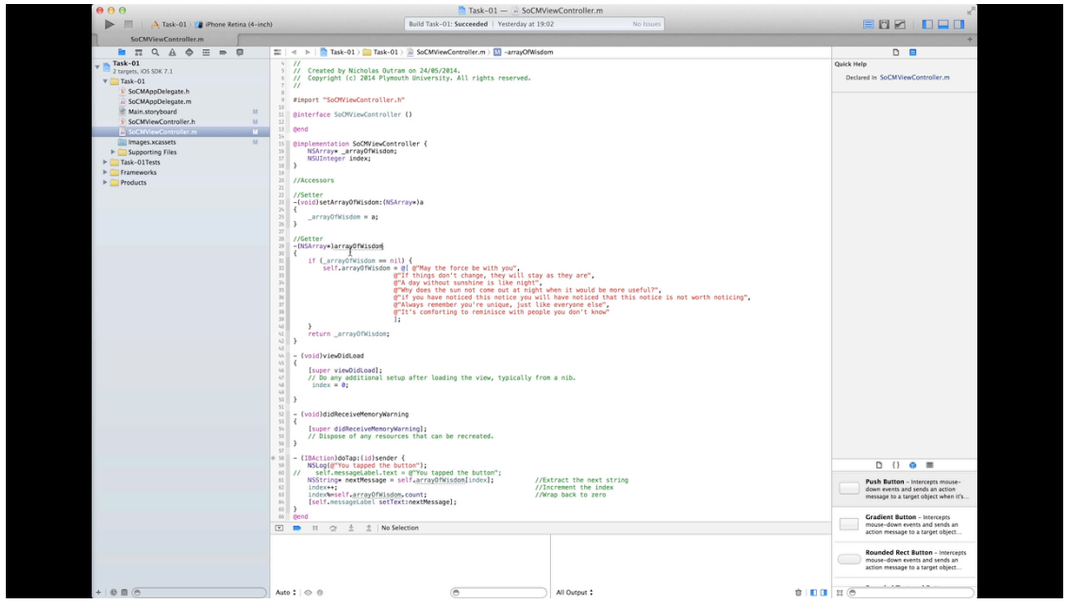
Why is the property for index defined as strong?

- **A.** So it cannot be over written
- **B.** So that the object it points to is retained in memory
- **C.** So that other objects cannot point to the same object

Check Answer

At some point, you should read the Apple documentation on @property.

Movie 1.8 Using properties



You should always get into the habit of using properties as it provides correctly written default accessors.

This is the end of Task 01

IBAction

Defined as void. When used as a return type, this serves two purposes - the method has no return data; XCode knows this method is to be used as an event handler. When you right click on a control, this method will be listed.

Related Glossary Terms

Drag related terms here

Index Find Term

Immutable

An object who's contents is defined at create time and which cannot be subsequently changed.

Related Glossary Terms

Drag related terms here

Index Find Term

Lazy loading

Where an object is not instantiated and initialized until it is first read. This technique has the advantage that if an object only consumed memory if it is actually needed. It also creates a tidy mechanism for initialization that does not depend on the life-cycle of other objects (constructors / initializers).

Related Glossary Terms

Drag related terms here

Index Find Term

Outlet

This term refers to a C pointer that is designated to point to a view object. This view object will be loaded using either StoryBoard or “Nib loading”. The Keywork IBOutlet is defined as blank, so does nothing to the code. However, it does tell XCode that this is a pointer to a view object.

Related Glossary Terms

Drag related terms here

Index Find Term

Section 1 - Task 01 - Single Page App

Property

Using the @property keyword, this creates one or more accessors on a property of an object. A property is often backed by an instance variable, but this is not always necessary (it could be derived from many others).

Related Glossary Terms

Drag related terms here

Index Find Term

Section 1 - Task 01 - Single Page App