

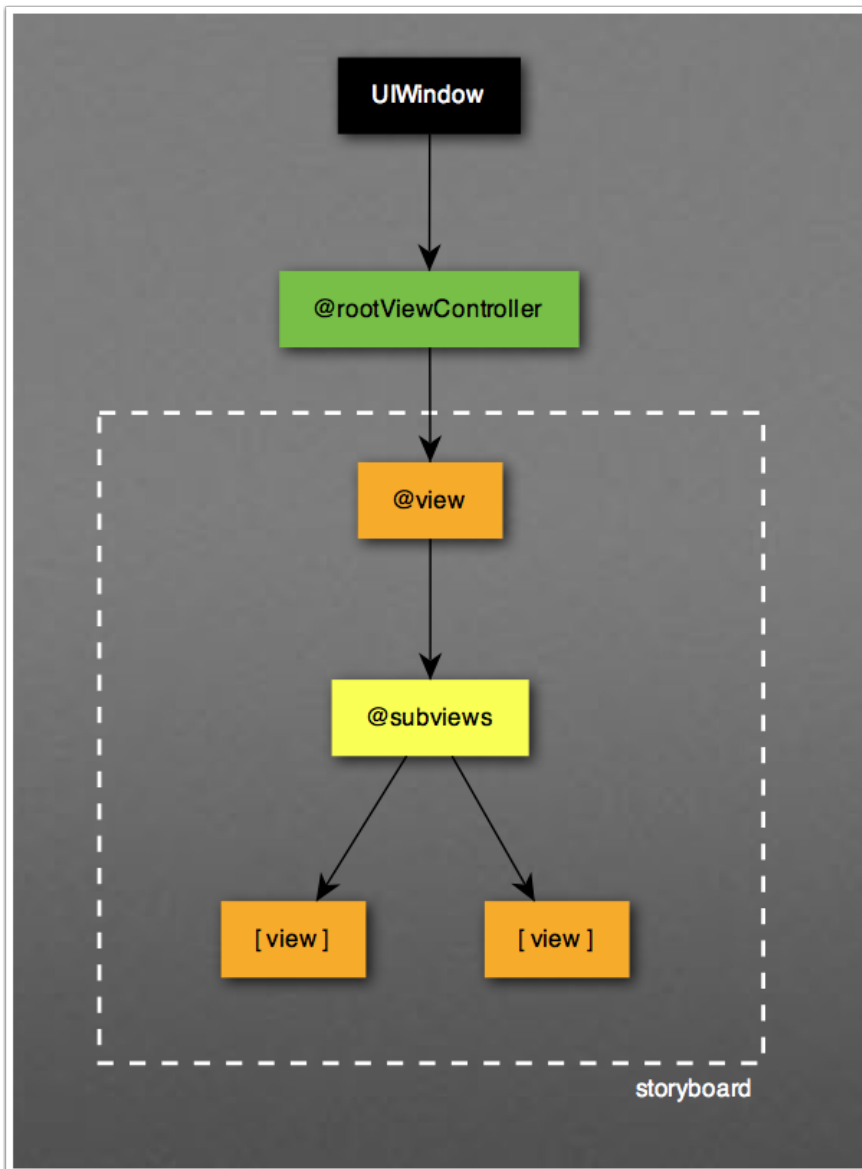
# Why outlets have weak references

---

## Introduction

This is a series of diagrams that visually explain why, when we create properties for outlets in our storyboard views, we give them weak references. The diagrams also show why it's handy to create outlet properties.

## A very simple view/controller hierarchy

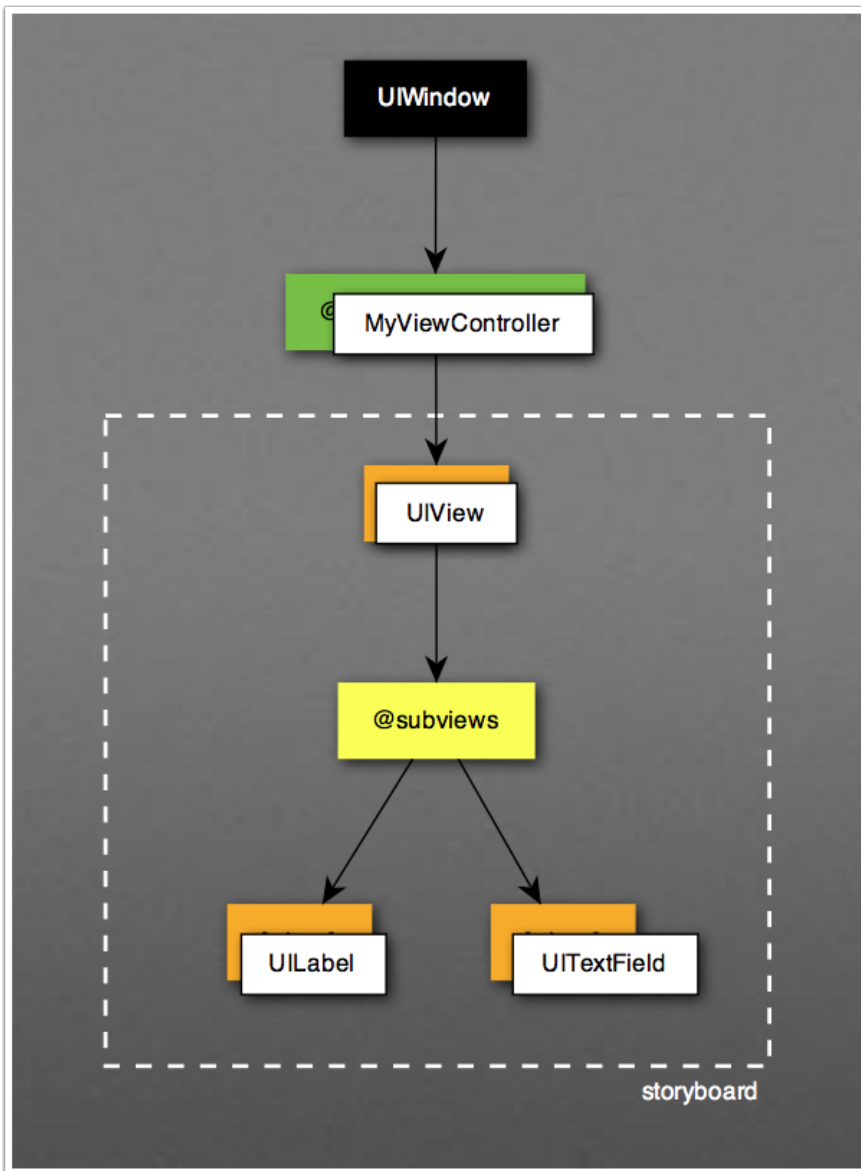


This diagram shows the hierarchical relationships of a very simple view controller and its views. (View controllers are green, views are orange, arrays are yellow, windows are black.)

The window that holds the app has a `rootViewController` property, which must be filled by an instance derived from `UIViewController`. The `rootViewController` has a `view` property, which must be filled by an instance derived from `UIView`. The view has a `subviews` property, which is an array that can be populated by more `UIView`-derived instances. (These include `UILabel`, `UIButton`, etc.)

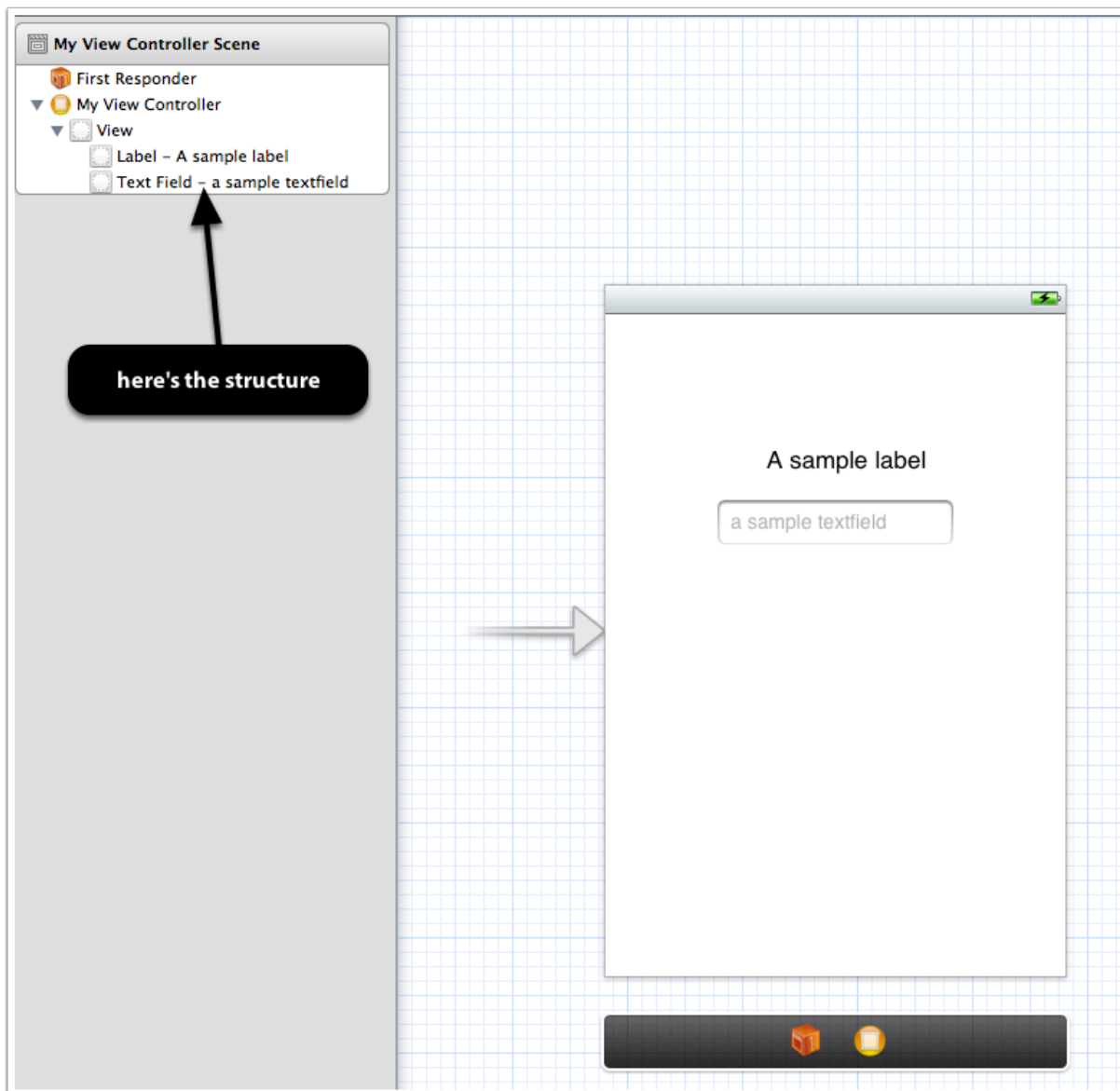
Everything inside the white dotted line would be created through the storyboard.

## The very simple hierarchy with some specifics added in



The boxes in white here show an example of what might actually go here. The rootViewController for the sample is an instance of a UIViewController subclass, called (very imaginatively) MyViewController. Within the storyboard, the base view (just a plain old UIView object) has two interface elements, a label and a text field.

## The very simple hierarchy as it appears in the storyboard



I created a Single View project, named the root view controller `MyViewController`, and in the storyboard dragged out a label and a text field. This creates the structure shown in the diagram above.

In terms of memory retention, here's what this all means: The window has a strong pointer to the `rootViewController` (`MyViewController`), which has a strong pointer to its view (as defined in the storyboard). The view has strong references to each of its subviews. Strong references are used when one object "owns" another object. Ownership follows the hierarchy down from the top.

## The problem of access...

```
// this should be the label
[self.view.subviews objectAtIndex: 0]

// this should be the textfield
[self.view.subviews objectAtIndex: 1]
```

So, now that we've created this hierarchy, there should be a way for the view controller (MyViewController) to talk to the subviews (the label and the textfield). They're in the hierarchy, so there is a way -- it's just messy. Something like what's shown here.

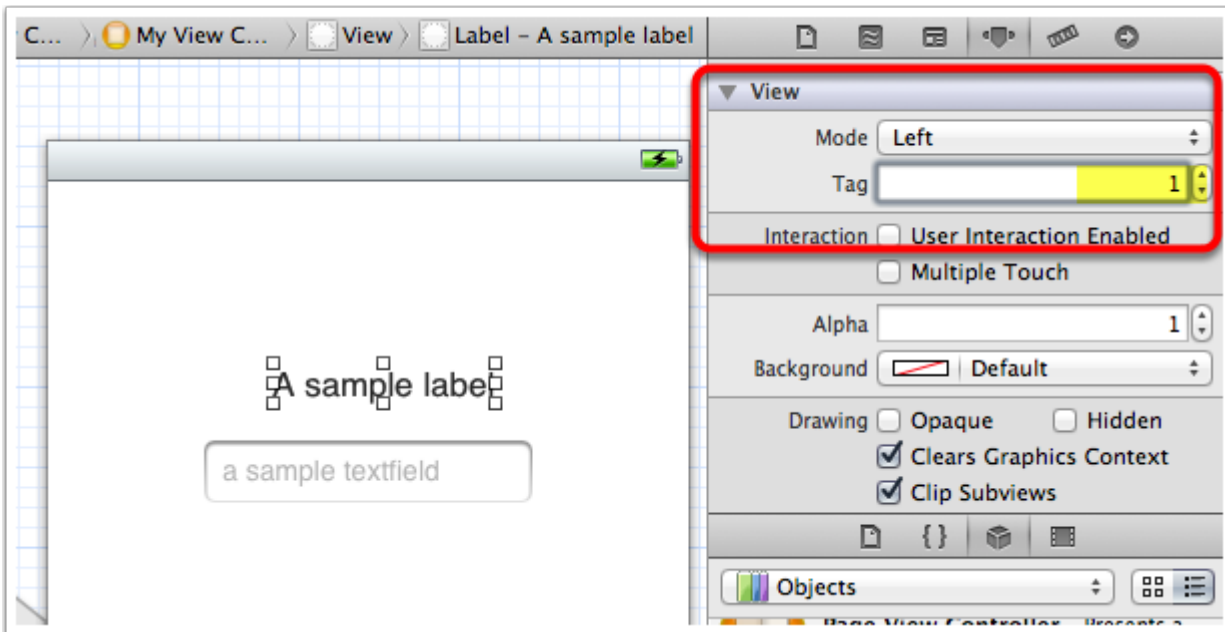
## Trying it out...

```
- (void)viewDidLoad
{
    [super viewDidLoad];

    UILabel *myLabel = (UILabel *)[self.view.subviews objectAtIndex:0];
    UITextField *myTextField = (UITextField *)[self.view.subviews objectAtIndex:1];
    myLabel.text = @"Hello World";
    myTextField.text = @"foobar";
}
```

This bit of code is just fraught with dangers -- if the first view in the subviews array isn't a label, or the second view isn't a textfield, there's a big fat crash in your future. But assuming the hierarchy is exactly as specified, it will work. So it is possible to work through the hierarchy.

## Using tags for access



A more efficient approach would be to tag each subview that we'll need to access later -- like tagging an animal in the wild. This can be done through the storyboard, by using the View section of the Attributes Inspector. In this screenshot, I've selected the label and given it a tag of 1 (tags must be integers, anything unique and above 0 should work).

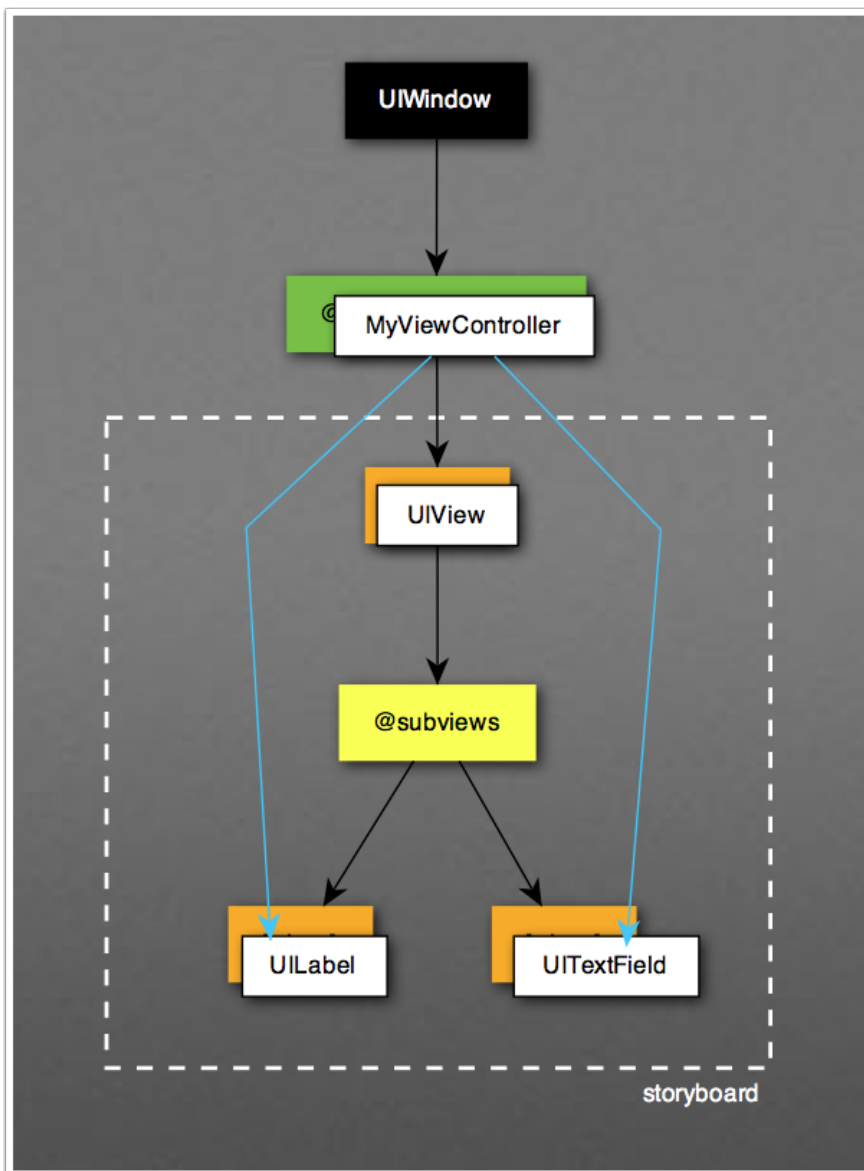
## Trying it out...

```
- (void)viewDidLoad
{
    [super viewDidLoad];

    UILabel *myLabel = (UILabel *)[self.view viewWithTag:1];
    UITextField *myTextField = (UITextField *)[self.view viewWithTag:2];
    myLabel.text = @"Hello World";
    myTextField.text = @"foobar";
}
```

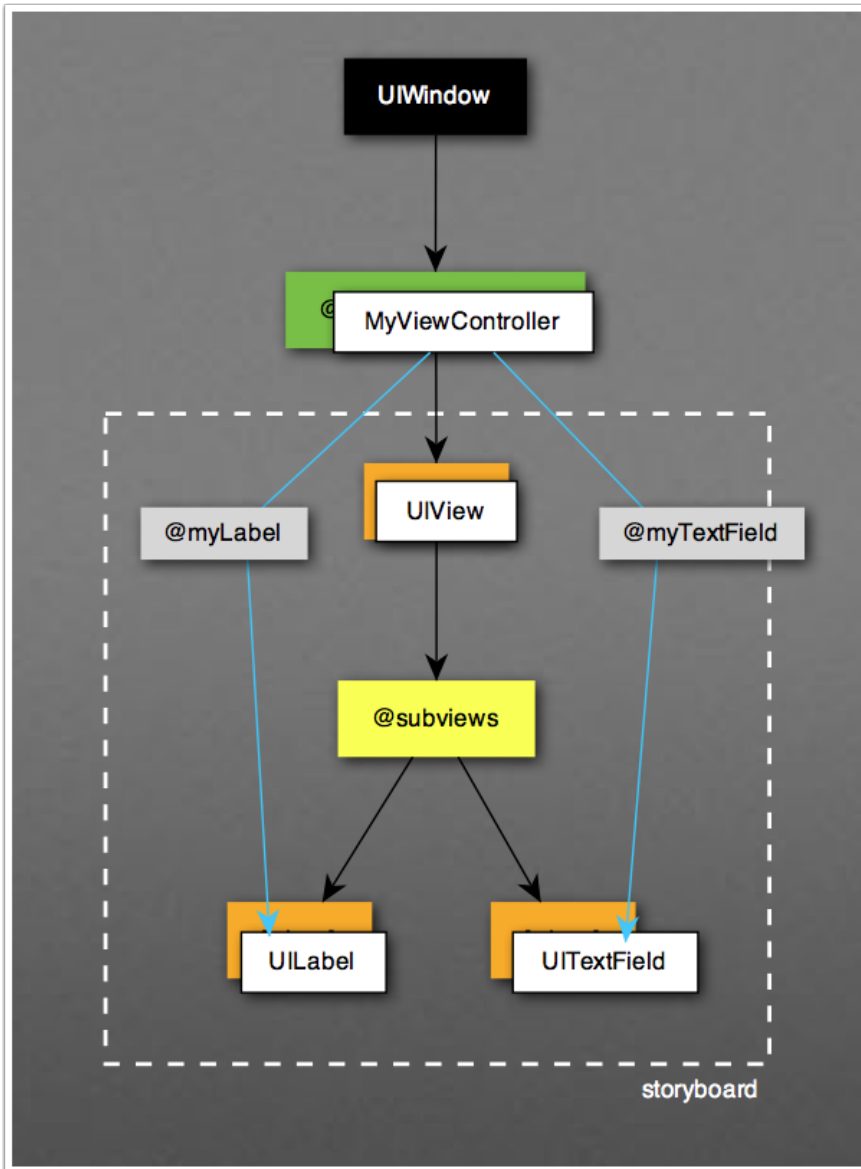
Assuming I gave the textfield a tag of 2, here's how I might access the two subviews in my code.

## What we're trying to do...



Here's the diagram again, showing what we're trying to do, which is get easy access to those two subviews, which could be buried way down in the view hierarchy. Ownership has already been determined by the main hierarchy (black arrows). We just want access (blue arrows).

## Using properties for quick access



The best way to access those subviews is to create a reference to them in the view controller using properties. But remember -- these properties are just handy-dandy references to something that already exists further down in the hierarchy. They don't imply any kind of ownership (we already have that), so they shouldn't be strong references.



## Properties in code

```
@interface MyViewController : UIViewController

@property (nonatomic, weak) UILabel *myLabel;
@property (nonatomic, weak) UITextField *myTextField;

@end
```

Here are the property declarations, showing that they are weak references.

## Connect the properties to the storyboard using IBOutlet

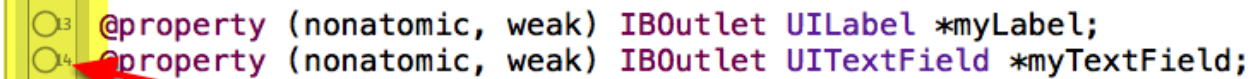
```
@interface MyViewController : UIViewController

@property (nonatomic, weak) IBOutlet UILabel *myLabel;
@property (nonatomic, weak) IBOutlet UITextField *myTextField;

@end
```

The previous code sample is correct and complete. In the sample above, I've added the keyboard IBOutlet, which doesn't add anything to its meaning -- but it does tell Xcode that I will want to hook these properties to objects in the storyboard.

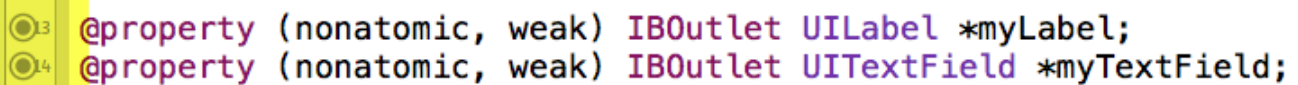
## An Xcode view of IBOutlet in action



```
@property (nonatomic, weak) IBOutlet UILabel *myLabel;  
@property (nonatomic, weak) IBOutlet UITextField *myTextField;
```

Once I've added IBOutlet (IB stands for Interface Builder, which is the part of Xcode that creates storyboards), Xcode displays the little circle in the gutter. This is for linking things to the storyboard.

## IBOutlet properties successfully wired up



```
@property (nonatomic, weak) IBOutlet UILabel *myLabel;  
@property (nonatomic, weak) IBOutlet UITextField *myTextField;
```

Once I've wired the outlets to the storyboard, the circles are filled in.