

# Developing Applications for iOS



## Lab 4: RPN Calculator App (3 of 3)

Radu Ionescu  
raducu.ionescu@gmail.com  
Faculty of Mathematics and Computer Science  
University of Bucharest

# Task 1

**Task:** Adjust the Calculator so that we can set variables without having to change the program. Add a button so that we can re-run the current program.

1. Launch Xcode and go to “File > Open” and select the Xcode project (.xcodproj) inside the “Calculator(2of3)” folder.
2. You can run the application in iOS Simulator and take a look at the new features from the last lab. Notice that we can now make programs that accept variables. But what if we want to run the “x y +” program again with different values for x and y. We cannot set both variables without changing the program.

We should be able to set variables and keep the program in the stack. And it would also be nice if we could see the program that we want to run in some auxiliary display. We are going to address these issues by completing this task.

3. Stop running the application.

# Task 1

**Task:** Adjust the Calculator so that we can set variables without having to change the program. Add a button so that we can re-run the current program.

4. Notice that the methods that set the variables are not changing the program. But we should be able to clear the display right after setting a variable so that we can set another variable with a different value. If we use the Clear button we also flush the current program.

We should separate the functionality of the Clear button. To do this, we are going to add another button.

Make room for this button by resizing the C button to 36 pixels wide and the `display` to 192 pixels wide.

5. Open the Utilities area and drag a button from Object Library to your View next to the C button.
6. Set the button title to “AC” and resize it to a width of 36 pixels. It should look like in the following screenshot.

Calculator.xcodeproj — MainStoryboard.storyboard

Running Calculator on iPhone 5.0 Simulator

No Issues

Run Stop Scheme Breakpoints Editor View Organizer

MainStoryboard.storyboard CalculatorBrain.m

Calculat... View Automatic CalculatorViewController.m -allClearPressed



```

1 //
2 // CalculatorViewController.m
3 // Calculator
4 //
5 // Created by Radu-Tudor Ionescu on 2/27/12.
6 // Copyright (c) 2012 __MyCompanyName__. All rights reserved.
7 //
8
9 #import "CalculatorBrain.h"
10 #import "CalculatorViewController.h"
11
12 @interface CalculatorViewController()
13
14 @property (nonatomic) BOOL userIsInTheMiddleOfEnteringANumber;
15 @property (nonatomic, strong) CalculatorBrain *brain;
16
17 @end
18
19 @implementation CalculatorViewController
20
21 @synthesize display;
22 @synthesize xDisplay;
23 @synthesize yDisplay;
24 @synthesize zDisplay;
25 @synthesize userIsInTheMiddleOfEnteringANumber;
26 @synthesize brain = _brain;
27
28 - (void)viewDidUnload
29 {
30     [self setDisplay:nil];
31     [self setXDisplay:nil];
32     [self setYDisplay:nil];
33     [self setZDisplay:nil];
34     [super viewDidUnload];
35     // Release any retained subviews of the main view.
36     // e.g. self.myOutlet = nil;
37 }
38
39 - (CalculatorBrain *)brain
40 {
41     if (_brain == nil) _brain = [[CalculatorBrain alloc] init];
42     return _brain;
43 }
44
45 - (IBAction)setX
46 {
47     self.xDisplay.text = self.display.text;
48 }
49

```

View

Show Frame Rectangle

Origin X: 0, Y: 20

Width: 320, Height: 460

Autosizing Example

Arrange Position View

Objects

Label

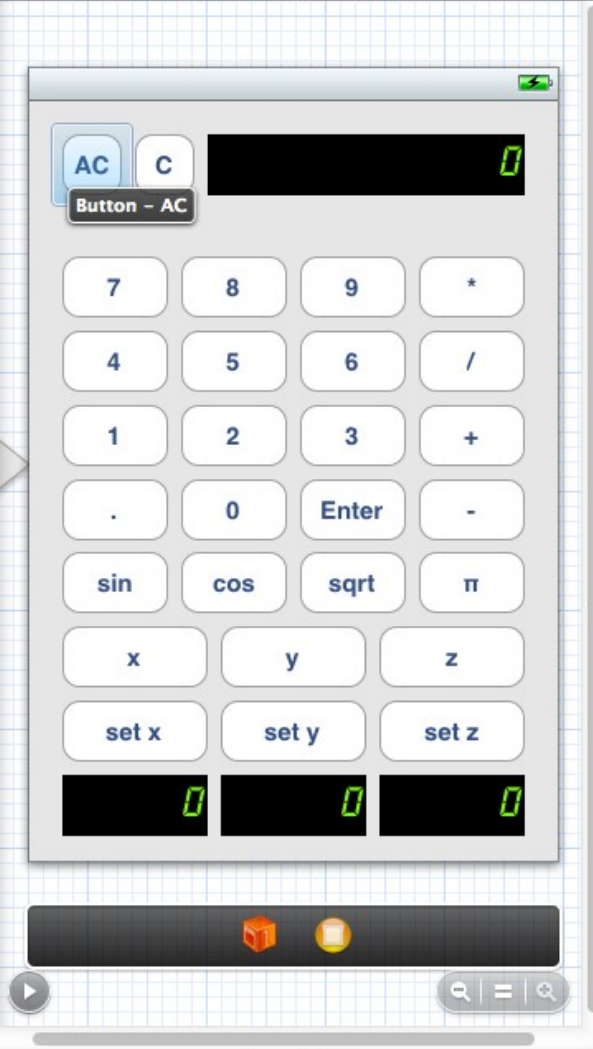
1 2 Text

# Task 1

Task: Adjust the Calculator so that we can set variables without having to change the program. Add a button so that we can re-run the current program.

7. CTRL-drag from the “AC” button in your View to your Controller implementation right before the `clearPressed` method to add an action.
8. Name the new action “allClearPressed” and make sure you select None for Arguments.
9. Copy and paste the code from the `clearPressed` method. This is exactly what `allClearPressed` should do.
10. Then delete the first line of code from the `clearPressed` method that clears the program stack. The C button should only clear the display.

Your Controller should look like in the following screenshot.



```

72 }
73
74 - (IBAction)pointPressed
75 {
76     if (self.userIsInTheMiddleOfEnteringANumber)
77     {
78         NSRange range = [self.display.text rangeOfString:@"."];
79         if (range.location == NSNotFound)
80         {
81             self.display.text = [self.display.text stringByAppendingString:@"."]
82         }
83     }
84     else
85     {
86         self.display.text = @"0.";
87         self.userIsInTheMiddleOfEnteringANumber = YES;
88     }
89 }
90
91 - (IBAction)enterPressed
92 {
93     [self.brain pushOperand:[self.display.text doubleValue]];
94     self.userIsInTheMiddleOfEnteringANumber = NO;
95 }
96
97 - (IBAction)allClearPressed
98 {
99     [self.brain emptyProgramStack];
100    self.display.text = @"0";
101    self.userIsInTheMiddleOfEnteringANumber = NO;
102 }
103
104 - (IBAction)clearPressed
105 {
106     self.display.text = @"0";
107     self.userIsInTheMiddleOfEnteringANumber = NO;
108 }
109
110 - (IBAction)variablePressed:(UIButton *)sender
111 {
112     if (self.userIsInTheMiddleOfEnteringANumber) [self enterPressed];
113
114     NSString *variable = [sender currentTitle];
115     [self.brain pushVariable:variable];
116     self.display.text = variable;
117 }
118
119 - (IBAction)operationPressed:(UIButton *)sender
120 {

```

View

Show Frame Rectangle

X: 0, Y: 20, Width: 320, Height: 460

Origin

Autosizing Example

Arrange Position View

Objects

Label

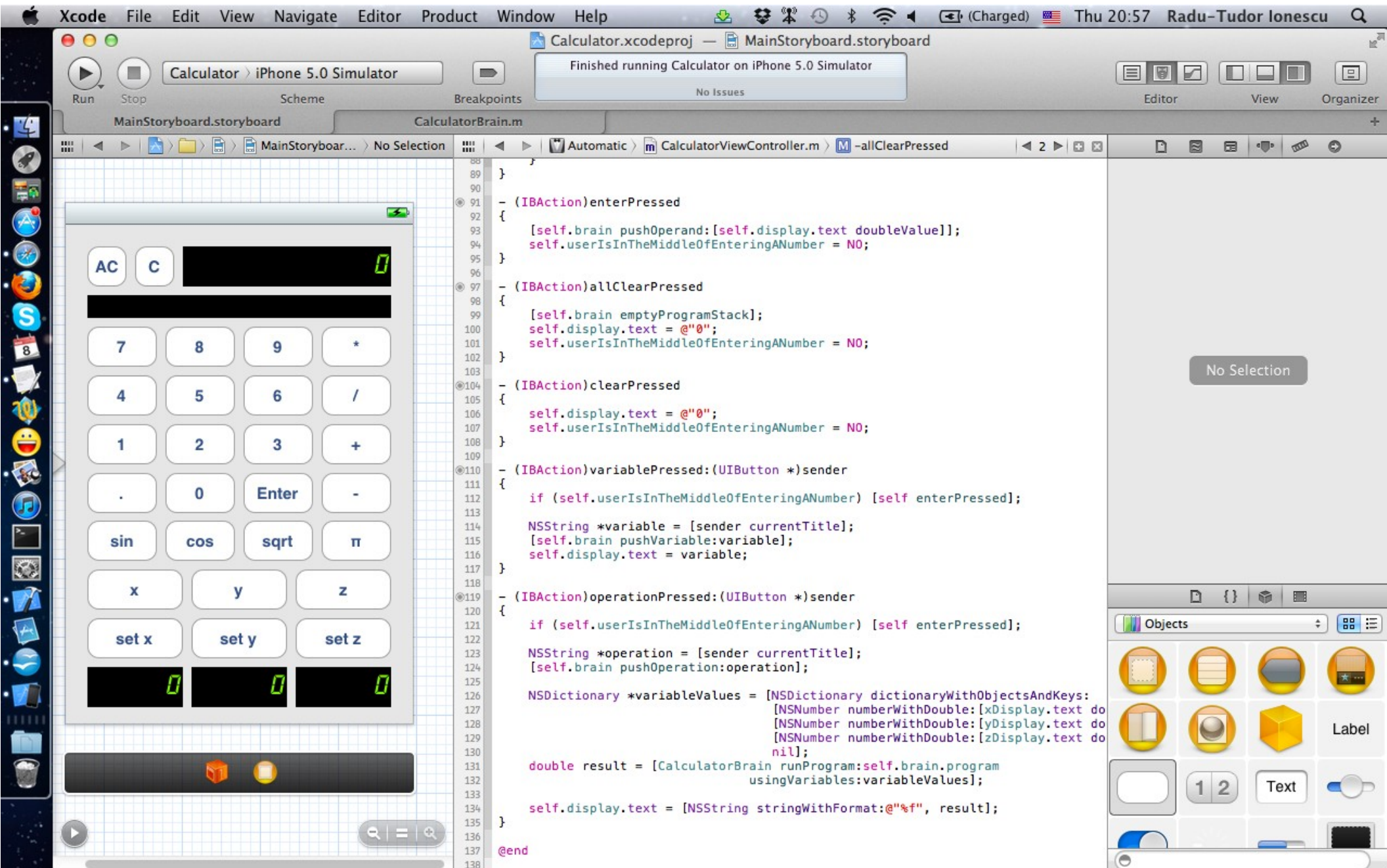
Text

# Task 1

**Task:** Adjust the Calculator so that we can set variables without having to change the program. Add a button so that we can re-run the current program.

11. It's time to add the auxiliary display that will show the current program (or more exactly, what's in the program stack). We should copy and paste the `display` label.
12. Place the copied `UILabel` right under the `display` label and resize it to 280x21 pixels.
13. Change the Font to System 16.0.
14. Clear the text inside this `UILabel`.

Your View should look like in the following screenshot.



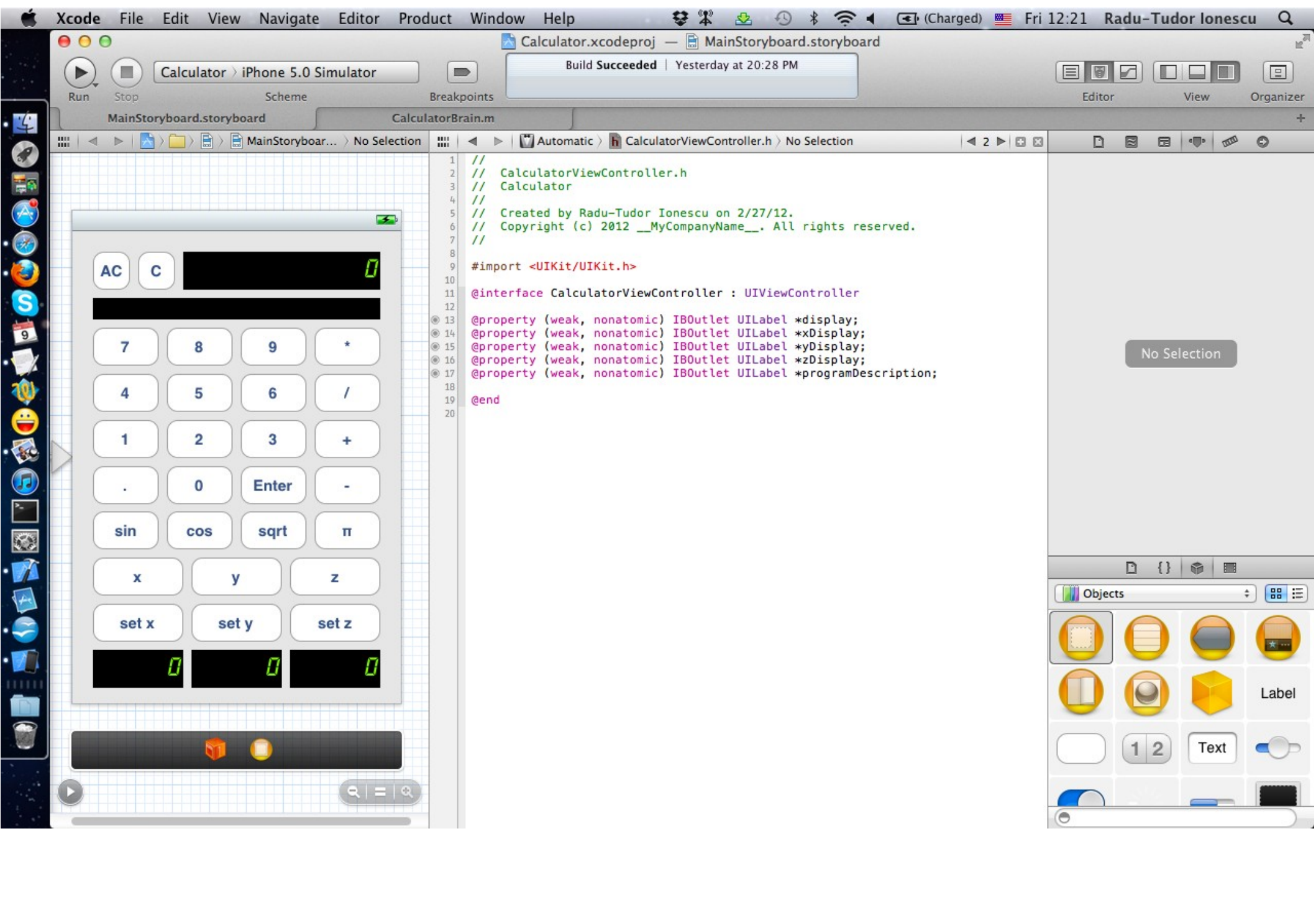


# Task 1

**Task:** Adjust the Calculator so that we can set variables without having to change the program. Add a button so that we can re-run the current program.

15. Open the CalculatorViewController.h in Assistant Editor.
16. CTRL-drag from the UILabel to the header file to create an outlet.
17. Name the outlet “programDescription” and select the Weak Storage for it.

The outlet should look like in the following screenshot.



```
1 //
2 // CalculatorViewController.h
3 // Calculator
4 //
5 // Created by Radu-Tudor Ionescu on 2/27/12.
6 // Copyright (c) 2012 __MyCompanyName__. All rights reserved.
7 //
8
9 #import <UIKit/UIKit.h>
10
11 @interface CalculatorViewController : UIViewController
12
13 @property (weak, nonatomic) IBOutlet UILabel *display;
14 @property (weak, nonatomic) IBOutlet UILabel *xDisplay;
15 @property (weak, nonatomic) IBOutlet UILabel *yDisplay;
16 @property (weak, nonatomic) IBOutlet UILabel *zDisplay;
17 @property (weak, nonatomic) IBOutlet UILabel *programDescription;
18
19 @end
20
```

No Selection

Objects palette containing various UI element icons such as buttons, labels, and text fields.

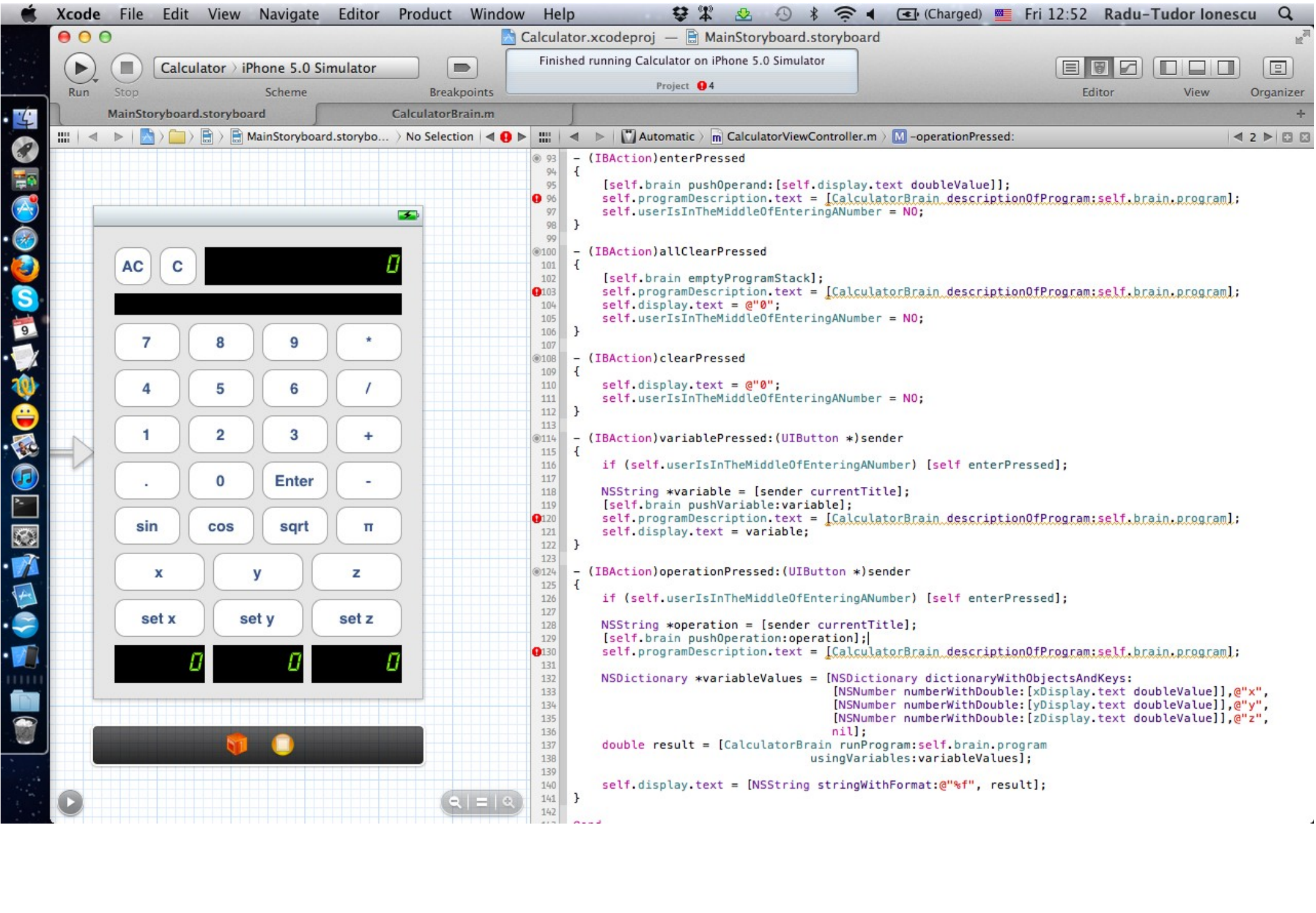
# Task 1

Task: Adjust the Calculator so that we can set variables without having to change the program. Add a button so that we can re-run the current program.

18. Open the CalculatorViewController.m in Assistant Editor.
19. We should update the `programDescription` each time we change the program stack. The program stack changes only when we push something to it (operands, variables and operations) and when we clear the program stack. In each of these cases we should simply send the `descriptionOfProgram:` message the CalculatorBrain that will return an `NSString` with the program description.

Don't worry about the errors. The next step will be to add the `descriptionOfProgram:` method to the CalculatorBrain's public API and implement it.

The Controller implementation should look like in the next screenshot.



# Task 1

Task: Adjust the Calculator so that we can set variables without having to change the program. Add a button so that we can re-run the current program.

20. Switch to the CalculatorBrain.m in tab in Xcode.
21. Declare the public class method `descriptionOfProgram:` that takes a program with the general type `id` and returns a pointer to an `NSString` object.
22. Implement the method so that it returns the description of the program into a `NSString` object. If the program is `nil` this method should also return `nil`. The simplest way to implement this method is to iterate through the program objects and append them to an `NSMutableString` using the `appendFormat:` method. We will have to use introspection again to distinguish between operands (`NSNumber`s) and variables/operations (`NSString`s).

The final CalculatorBrain implementation should look like in the next screenshot.

Xcode File Edit View Navigate Editor Product Window Help

Calculator.xcodeproj — CalculatorBrain.m

Calculator > iPhone 5.0 Simulator

Finished running Calculator on iPhone 5.0 Simulator

No Issues

MainStoryboard.storyboard CalculatorBrain.m

Calculator > Calculator > CalculatorBrain.m > @implementation CalculatorBrain

```
8
9 #import "CalculatorBrain.h"
10
11 @interface CalculatorBrain()
12
13 @property (nonatomic, strong) NSMutableArray *programStack;
14
15 + (BOOL)isOperation:(NSString *)operation;
16 + (double)popTermOffProgramStack:(NSMutableArray *)stack;
17
18 @end
19
20 @implementation CalculatorBrain
21
22 @synthesize programStack = _programStack;
23
24 + (NSString *)descriptionOfProgram:(id)program
25 {
26     if (program == nil) return nil;
27     NSArray *stack;
28     if ([program isKindOfClass:[NSArray class]]) stack = program;
29
30     NSMutableString *programDescription = [NSMutableString stringWithString:@""];
31     for (id stackObject in stack)
32     {
33         if ([stackObject isKindOfClass:[NSString class]])
34         {
35             [programDescription appendFormat:@"% %@",stackObject];
36         }
37         else if ([stackObject isKindOfClass:[NSNumber class]])
38         {
39             [programDescription appendFormat:@"% %.2f",[stackObject doubleValue]];
40         }
41     }
42     return (NSString *)programDescription;
43 }
44
45 + (double)runProgram:(id)program
46 {
47     NSMutableArray *stack;
48     if ([program isKindOfClass:[NSArray class]])
49     {
50         stack = [program mutableCopy];
51     }
52     return [self popTermOffProgramStack:stack];
53 }
54
55 + (double)runProgram:(id)program
56     usingVariables:(NSDictionary *)variableValues
57 {
58     NSMutableArray *stack;
```

```
1 //
2 // CalculatorBrain.h
3 // Calculator
4 //
5 // Created by Radu-Tudor Ionescu on 3/1/12.
6 // Copyright (c) 2012 __MyCompanyName__. All rights reserved.
7 //
8
9 #import <Foundation/Foundation.h>
10
11 @interface CalculatorBrain : NSObject
12
13 @property (nonatomic, readonly) id program;
14
15 + (NSString *)descriptionOfProgram:(id)program;
16 + (double)runProgram:(id)program;
17 + (double)runProgram:(id)program
18     usingVariables:(NSDictionary *)variableValues;
19
20 - (void)emptyProgramStack;
21 - (void)pushOperand:(double)operand;
22 - (void)pushOperation:(NSString *)operation;
23 - (void)pushVariable:(NSString *)variable;
24
25 @end
26
```

# Task 1

**Task:** Adjust the Calculator so that we can set variables without having to change the program. Add a button so that we can re-run the current program.

23. The only thing left to do is to add another button so that we can “run” the current “program”. Switch to MainStoryboard.storyboard tab in Xcode to add this final button.

24. Open the Utilities area.

25. Make room for the “run” button right under the  $\pi$  button. Resize the x, y and z buttons to 64 pixels wide and align them nicely with the buttons above.

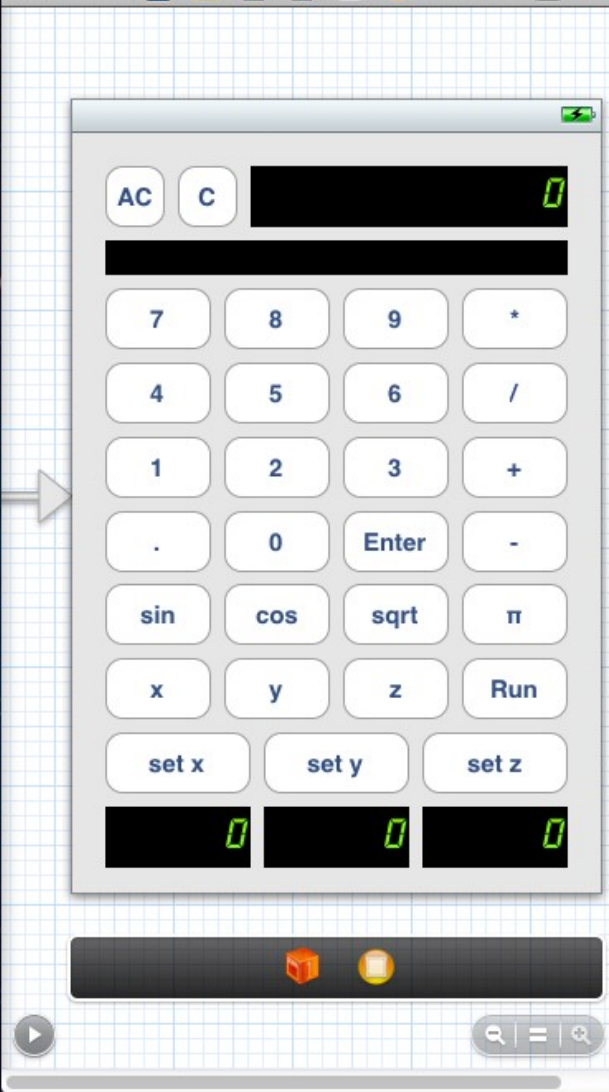
26. Drag an `UIButton` from the Object Library to your View.

27. Set the button title to “Run” and resize it to 64 pixels wide.

Your final View should look like in the next screenshot.

Calculator.xcodeproj — MainStoryboard.storyboard  
Finished running Calculator on iPhone 5.0 Simulator  
No Issues

MainStoryboard.storyboard CalculatorBrain.m  
Automatic CalculatorViewController.m -operationPressed:



```
93 - (IBAction)enterPressed
94 {
95     [self.brain pushOperand:[self.display.text doubleValue]];
96     self.programDescription.text = [CalculatorBrain descriptionOfProgram:self.brain.programDescription];
97     self.userIsInTheMiddleOfEnteringANumber = NO;
98 }
99
100 - (IBAction)allClearPressed
101 {
102     [self.brain emptyProgramStack];
103     self.programDescription.text = [CalculatorBrain descriptionOfProgram:self.brain.programDescription];
104     self.display.text = @"0";
105     self.userIsInTheMiddleOfEnteringANumber = NO;
106 }
107
108 - (IBAction)clearPressed
109 {
110     self.display.text = @"0";
111     self.userIsInTheMiddleOfEnteringANumber = NO;
112 }
113
114 - (IBAction)variablePressed:(UIButton *)sender
115 {
116     if (self.userIsInTheMiddleOfEnteringANumber) [self enterPressed];
117
118     NSString *variable = [sender currentTitle];
119     [self.brain pushVariable:variable];
120     self.programDescription.text = [CalculatorBrain descriptionOfProgram:self.brain.programDescription];
121     self.display.text = variable;
122 }
123
124 - (IBAction)operationPressed:(UIButton *)sender
125 {
126     if (self.userIsInTheMiddleOfEnteringANumber) [self enterPressed];
127
128     NSString *operation = [sender currentTitle];
129     [self.brain pushOperation:operation];
130     self.programDescription.text = [CalculatorBrain descriptionOfProgram:self.brain.programDescription];
131
132     NSDictionary *variableValues = [NSDictionary dictionaryWithObjectsAndKeys:
133         [NSNumber numberWithDouble:[xDisplay.text doubleValue]],
134         [NSNumber numberWithDouble:[yDisplay.text doubleValue]],
135         [NSNumber numberWithDouble:[zDisplay.text doubleValue]],
136         nil];
137
138     double result = [CalculatorBrain runProgram:self.brain.program
139                     usingVariables:variableValues];
140
141     self.display.text = [NSString stringWithFormat:@"%f", result];
142 }
143
144 }
```

View

- Mode: Scale To Fill
- Tag: 0
- Interaction:  User Interaction Enabled,  Multiple Touch
- Alpha: 1
- Background: [Color Picker]
- Drawing:  Opaque,  Hidden,  Clears Graphics Context,  Clip Subviews,  Autoresize Subviews
- Stretching: X: 0, Y: 0, Width: 1, Height: 1

Objects

- Image icons
- Label
- Text



# Task 1

**Task:** Adjust the Calculator so that we can set variables without having to change the program. Add a button so that we can re-run the current program.

28. CTRL-drag from the “run” button in your View to the Controller implementation right before the `@end` keyword to create an action for this button.
29. Name the action “runPressed” and select None for Arguments.
30. The `runPressed` method is similar to `operationPressed:`, only that we don't have to push any operation on the stack. If the user is in the middle of entering a number that input will be cleared (not pushed on the stack) and replaced with the result of running the “program”. To let things work properly we also have to set the `userIsInTheMiddleOfEnteringANumber` to NO.

Basically, we need to build the `variableValues` dictionary and send the `runProgram:usingVariableValues:` message to the `CalculatorBrain`. See how to implement the `runPressed` method in the next screenshot.

Xcode File Edit View Navigate Editor Product Window Help

Calculator.xcodeproj — MainStoryboard.storyboard

Finished running Calculator on iPhone 5.0 Simulator

No Issues


Calculator > iPhone 5.0 Simulator

Run Stop Scheme Breakpoints Editor View Organizer

MainStoryboard.storyboard CalculatorBrain.m

MainStoryboard.storyboard (English) > No Selection

Automatic > CalculatorViewController.m > M --runPressed



```
108 - (IBAction)clearPressed
109 {
110     self.display.text = @"0";
111     self.userIsInTheMiddleOfEnteringANumber = NO;
112 }
113
114 - (IBAction)variablePressed:(UIButton *)sender
115 {
116     if (self.userIsInTheMiddleOfEnteringANumber) [self enterPressed];
117
118     NSString *variable = [sender currentTitle];
119     [self.brain pushVariable:variable];
120     self.programDescription.text = [CalculatorBrain descriptionOfProgram:self.brain.program];
121     self.display.text = variable;
122 }
123
124 - (IBAction)operationPressed:(UIButton *)sender
125 {
126     if (self.userIsInTheMiddleOfEnteringANumber) [self enterPressed];
127
128     NSString *operation = [sender currentTitle];
129     [self.brain pushOperation:operation];
130     self.programDescription.text = [CalculatorBrain descriptionOfProgram:self.brain.program];
131
132     NSDictionary *variableValues = [NSDictionary dictionaryWithObjectsAndKeys:
133                                     [NSNumber numberWithInt:[xDisplay.text doubleValue]],@"x",
134                                     [NSNumber numberWithInt:[yDisplay.text doubleValue]],@"y",
135                                     [NSNumber numberWithInt:[zDisplay.text doubleValue]],@"z",
136                                     nil];
137     double result = [CalculatorBrain runProgram:self.brain.program
138                     usingVariables:variableValues];
139
140     self.display.text = [NSString stringWithFormat:@"%f", result];
141 }
142
143 - (IBAction)runPressed
144 {
145     NSDictionary *variableValues = [NSDictionary dictionaryWithObjectsAndKeys:
146                                     [NSNumber numberWithInt:[xDisplay.text doubleValue]],@"x",
147                                     [NSNumber numberWithInt:[yDisplay.text doubleValue]],@"y",
148                                     [NSNumber numberWithInt:[zDisplay.text doubleValue]],@"z",
149                                     nil];
150     double result = [CalculatorBrain runProgram:self.brain.program
151                     usingVariables:variableValues];
152
153     self.display.text = [NSString stringWithFormat:@"%f", result];
154     self.userIsInTheMiddleOfEnteringANumber = NO;
155 }
156
157 @end
```

# Task 1

Task: Adjust the Calculator so that we can set variables without having to change the program. Add a button so that we can re-run the current program.

31. Run the application in iOS Simulator. Press 3 “set x” and C. Then press 4 “set y” and C. Now let's try to run the  $\sqrt{x^2 + y^2}$  equation. Press  $x^2 + y^2 + \sqrt{\phantom{x}}$  to put this equation in the program stack. Press Run and check that the result is  $5 = \sqrt{3^2 + 4^2}$ . Now press C 7 “set x” C 24 “set y” to change the values of x and y. Press Run and check that the result is  $25 = \sqrt{7^2 + 24^2}$  this time.

Now you can play with your calculator and solve equations!

# Assignment 1

Assignment: Remember that you were advised to rename the instance variable of a `@property` by prefixing it with underscore. You should do this for the `@synthesized` properties in your project.

Hints: To rename the instance variable associated to a `@property` you have to add `"= _propertyName"` in the `@synthesize` declaration of that property.

Note that this will break the `operationPressed:` and `runPressed` methods because we are not using the getters of `xDisplay`, `yDisplay` and `zDisplay` outlets. You should fix this by using the getters (call `self.xDisplay` instead of `xDisplay`).

It's very good if you noticed this issue right when we implemented the `operationPressed:` and `runPressed` methods. This mistake was left on purpose to show the importance of prefixing your instance variables with underscore.

# Assignment 2\*\*

Assignment: Re-implement the `descriptionOfProgram:` method to display the passed program in a more user-friendly manner. Specifically,

- It should display all single-operand operations using “function” notation. For example, 10 sqrt should display as `sqrt(10)`.
- It should display all multi-operand operations using “infix” notation if appropriate, else function notation. For example, 3 Enter 5 + should display as `(3 + 5)`.
- Any no-operand operations, like  $\pi$ , should appear unadorned. For example,  $\pi$ .
- Variables should also appear unadorned. For example, `x`.

Any combination of operations, operands and variables should display properly. It might be that there are multiple things on the stack. If so, separate them by commas in the output with the top of the stack first, for example 3 E 5 E would display as “5, 3”.

## Assignment 2\*\*

Hints: You will almost certainly want to use recursion to implement `descriptionOfProgram:` (just like we did to implement `runProgram:`). You might find it useful to write yourself a `descriptionOfTopOfStack:` method too (just like we wrote ourselves a `popOperandOffStack:` method to help us implement `runProgram:`). If you find recursion a challenge, think “simpler”, not “more complex”. Your `descriptionOfTopOfStack:` method should be less than 20 lines of code and will be very similar to `popOperandOffStack:`.

One of the things your `descriptionOfProgram:` method is going to need to know is whether a given string on the stack is a two-operand operation, a single-operand operation, a no-operand operation or a variable (because it gives a different description for each of those kinds of things).

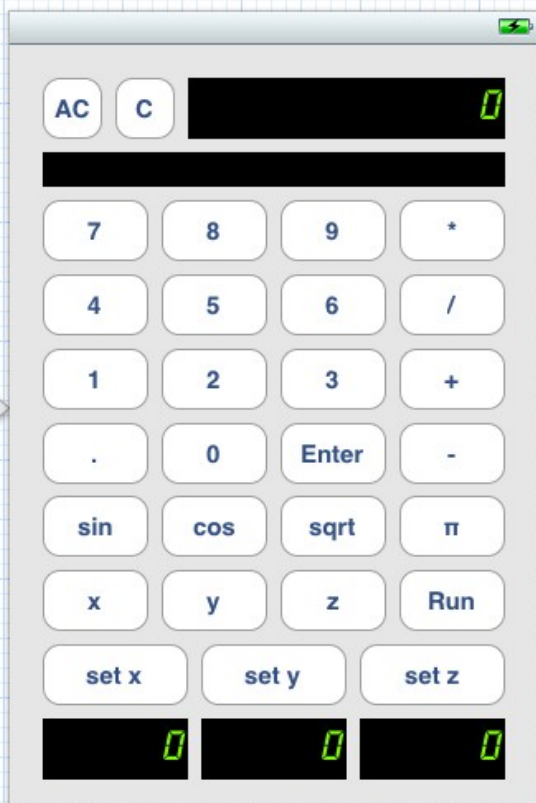
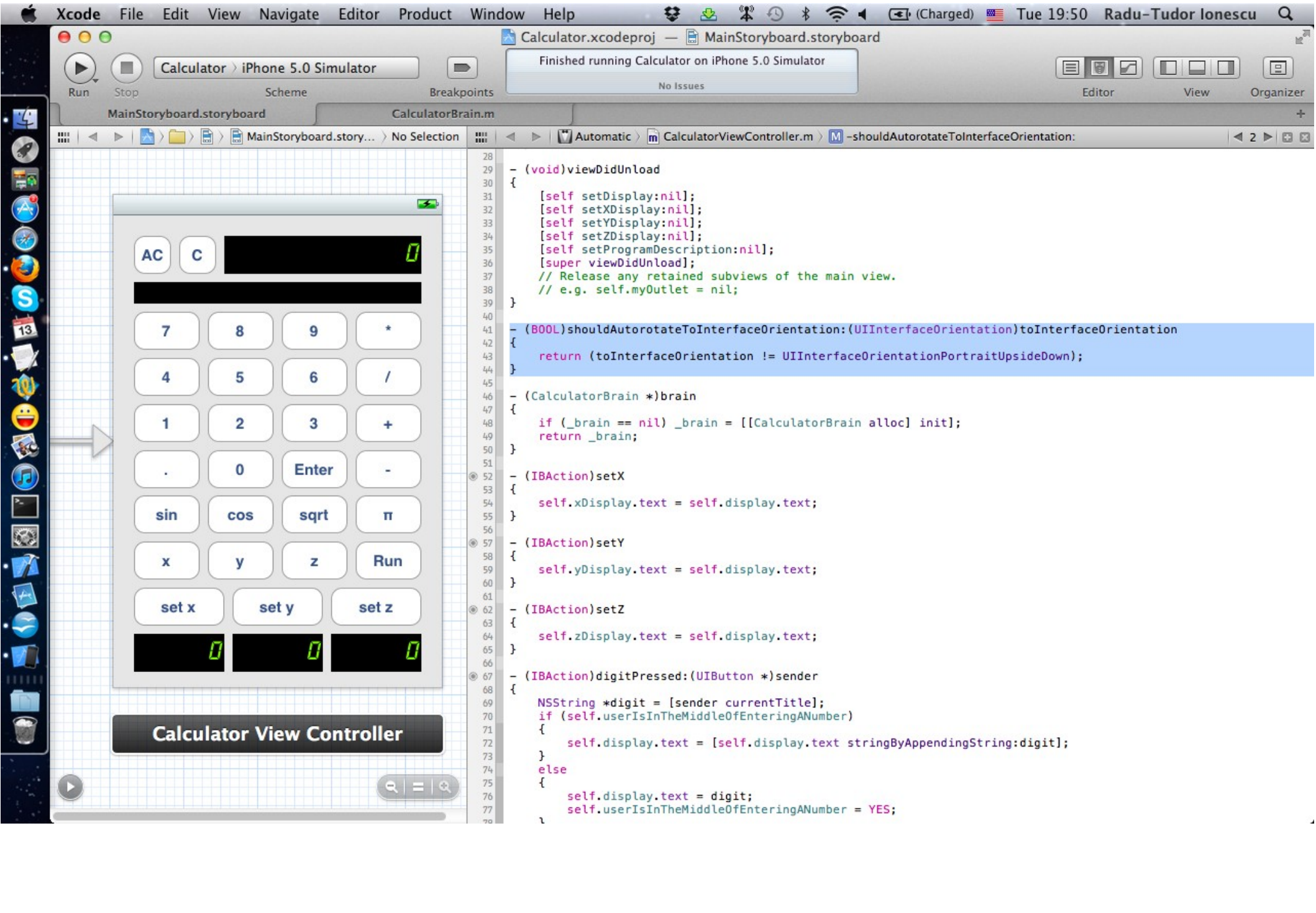
## Task 2

Task: Allow your Calculator to display its View in portrait and in landscape mode using autorotation.

1. Open the MainStoryboard.storyboard tab in Xcode.
2. Select the CalculatorViewController.m in Assistant Editor.
3. Implement the `shouldAutorotateToInterfaceOrientation:` method so that your View will autorotate to all orientations besides the portrait upside down orientation. You will have to return YES for supported device orientations and NO for the portrait upside down orientation. Place this method right after `viewDidLoad:`.

The implementation of this method can be viewed on the next slide.

4. Run the application in iOS Simulator and notice how the View changes when you rotate the device. To change device orientations on iOS Simulator use the CMD + Left Arrow or CMD + Right Arrow shortcut keys (or look in the Simulator's Hardware menu for the Rotate Left or Rotate Right options).



```
28
29 - (void)viewDidLoad
30 {
31     [self setDisplay:nil];
32     [self setXDisplay:nil];
33     [self setYDisplay:nil];
34     [self setZDisplay:nil];
35     [self setDescription:nil];
36     [super viewDidLoad];
37     // Release any retained subviews of the main view.
38     // e.g. self.myOutlet = nil;
39 }
40
41 - (BOOL)shouldAutorotateToInterfaceOrientation:(UIInterfaceOrientation)toInterfaceOrientation
42 {
43     return (toInterfaceOrientation != UIInterfaceOrientationPortraitUpsideDown);
44 }
45
46 - (CalculatorBrain *)brain
47 {
48     if (_brain == nil) _brain = [[CalculatorBrain alloc] init];
49     return _brain;
50 }
51
52 - (IBAction)setX
53 {
54     self.xDisplay.text = self.display.text;
55 }
56
57 - (IBAction)setY
58 {
59     self.yDisplay.text = self.display.text;
60 }
61
62 - (IBAction)setZ
63 {
64     self.zDisplay.text = self.display.text;
65 }
66
67 - (IBAction)digitPressed:(UIButton *)sender
68 {
69     NSString *digit = [sender currentTitle];
70     if (self.userIsInTheMiddleOfEnteringANumber)
71     {
72         self.display.text = [self.display.text stringByAppendingString:digit];
73     }
74     else
75     {
76         self.display.text = digit;
77         self.userIsInTheMiddleOfEnteringANumber = YES;
78     }
79 }
```

Calculator View Controller



## Task 2

Task: Allow your Calculator to display its View in portrait and in landscape mode using autorotation.

5. Stop running the application.
6. We must adjust the View so that it displays nicely in landscape mode too.

Let's open the Utilities area.

7. Select the Size Inspector to set up springs and struts so that the `display` and `programDescription` labels adjust their size to fit the screen width.

The following screenshots will show you how to do this.

8. Run the application and check that the labels resize when you change the device orientation.
9. Stop running the application.

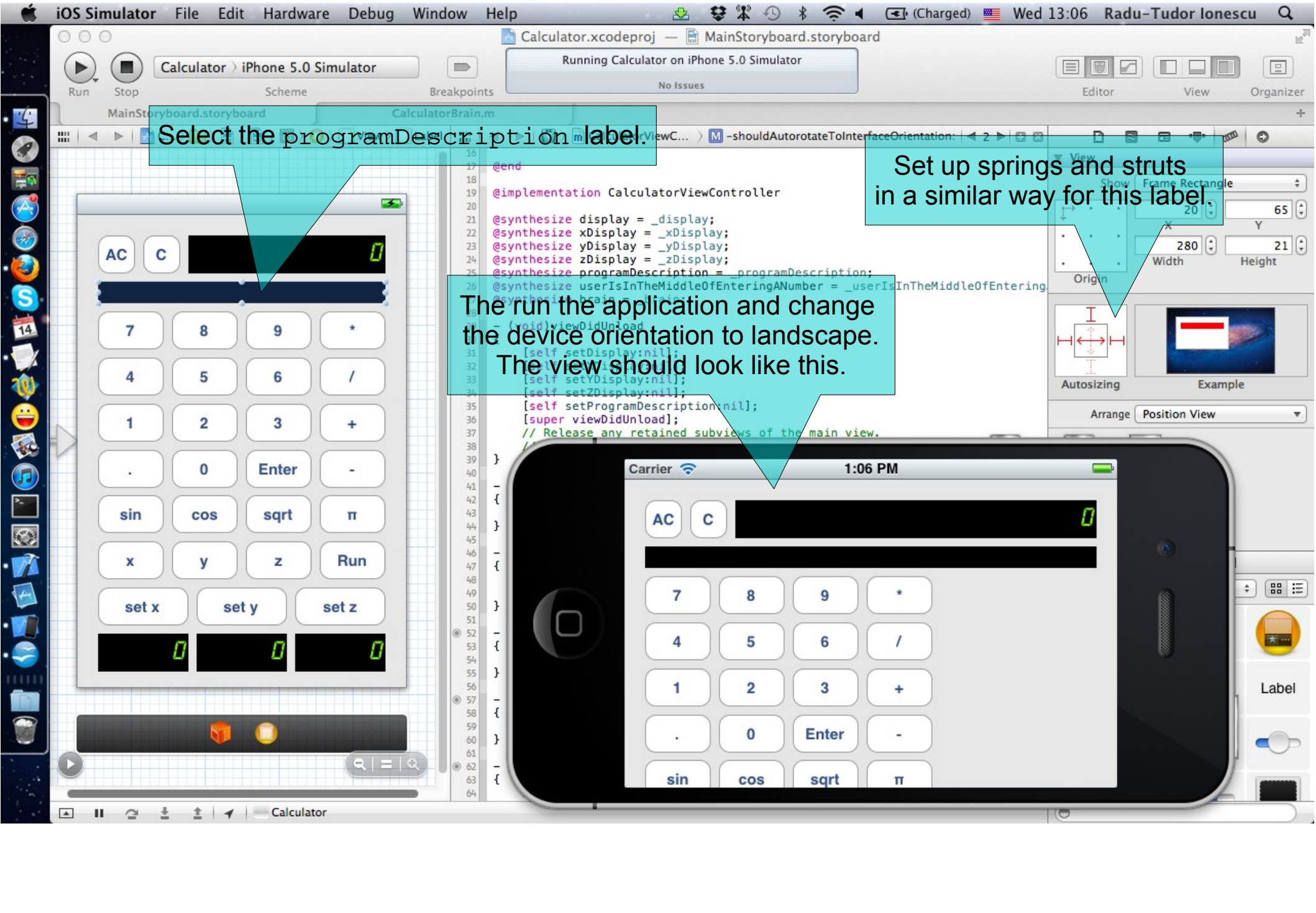
Select the Size Inspector from here.

Make sure the display label is selected.

```
16
17 @end
18
19 @implementation CalculatorViewController
20
21 @synthesize display = _display;
22 @synthesize xDisplay = _xDisplay;
23 @synthesize yDisplay = _yDisplay;
24 @synthesize zDisplay = _zDisplay;
25 @synthesize programDescription = _programDescription;
26 @synthesize userIsInTheMiddleOfEnteringANumber = _userIsInTheMiddleOfEnteringANumber;
27 @synthesize brain = _brain;
28
29 -(void)viewDidUnload
30 {
31     [self setDisplay:nil];
32     [self setXDisplay:nil];
33     [self setYDisplay:nil];
34     [self setZDisplay:nil];
35     [self setProgramDescription:nil];
36     [super viewDidUnload];
37     // Release any retained subviews of the main view.
38     // e.g. self.myOutlet = nil;
39 }
40
41 -(BOOL)shouldAutorotateToInterfaceOrientation:(UIInterfaceOrientation)interfaceOrientation
42 {
43     return (toInterfaceOrientationSupported != YES);
44 }
45
46 -(CalculatorBrain *)brain
47 {
48     if (_brain == nil) _brain = [[CalculatorBrain alloc] initWithProgramDescription:@"Calculator"];
49     return _brain;
50 }
51
52 -(IBAction)setX
53 {
54     self.xDisplay.text = self.display.text;
55 }
56
57 -(IBAction)setY
58 {
59     self.yDisplay.text = self.display.text;
60 }
61
62 -(IBAction)setZ
63 {
64     self.zDisplay.text = self.display.text;
65 }
66
```

Set up springs and struts like this. It means the display will keep its position and it will resize horizontally on autorotation.

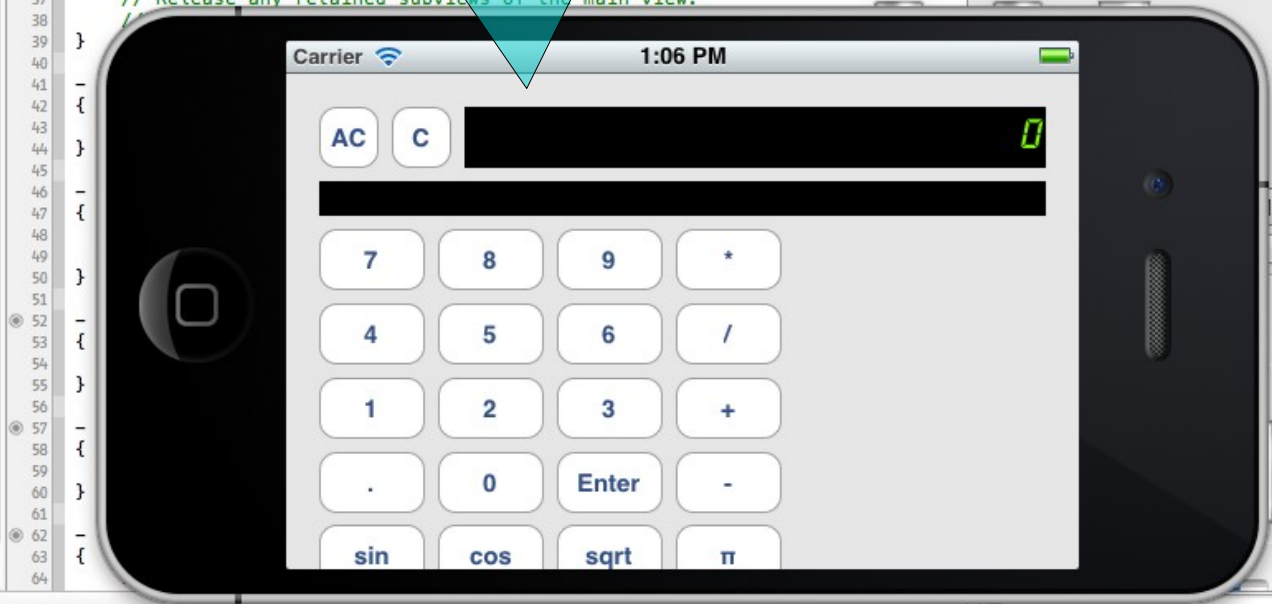
The View panel shows the 'Frame Rectangle' with X: 108, Y: 20, Width: 192, and Height: 37. The 'Autosizing' section shows a diagram with horizontal and vertical struts. The 'Objects' panel shows a selected 'Label' object.



Select the programDescription label.

Set up springs and struts in a similar way for this label

The run the application and change the device orientation to landscape. The view should look like this.



## Task 2

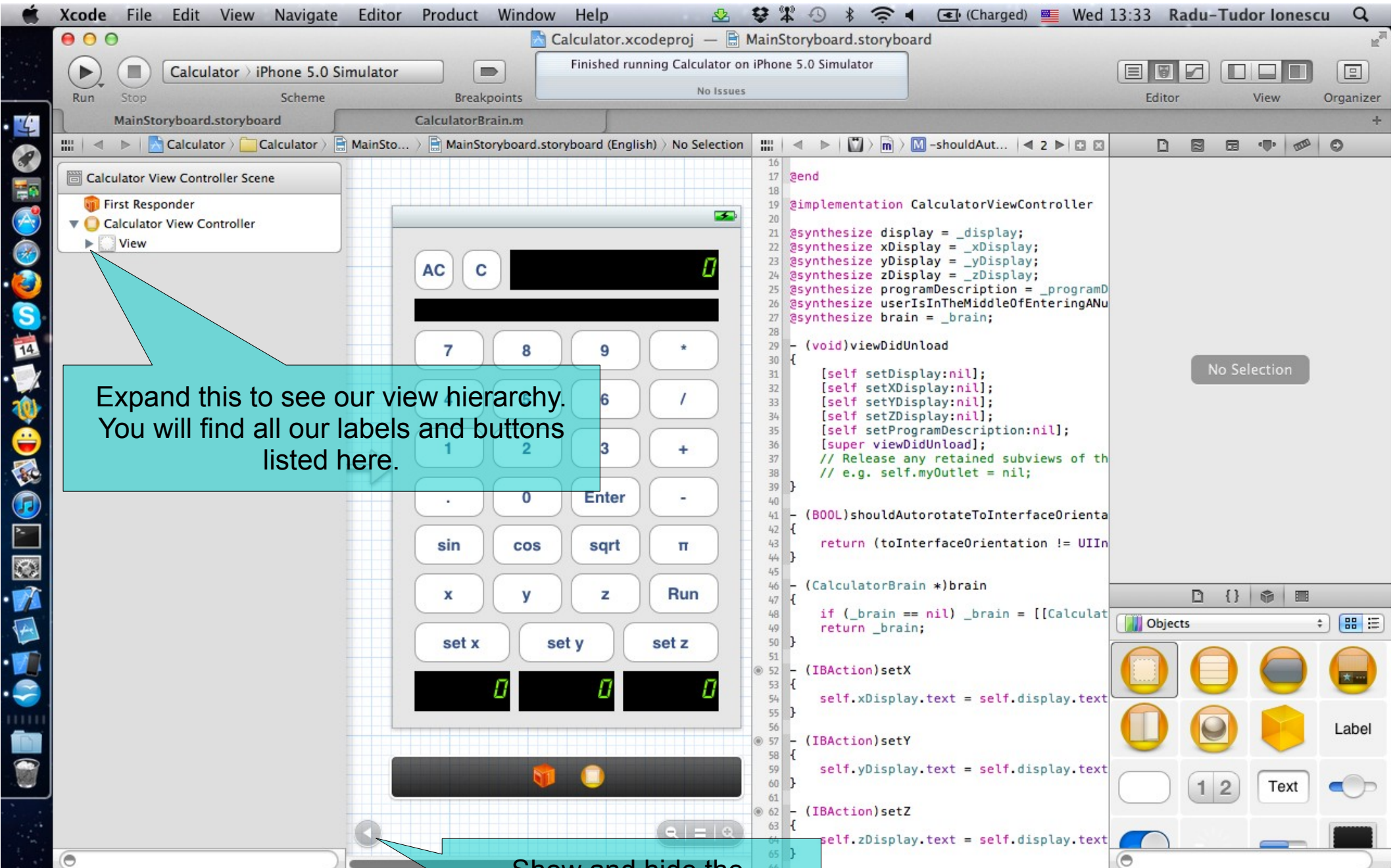
Task: Allow your Calculator to display its View in portrait and in landscape mode using autorotation.

10. We should adjust the buttons so that they fit inside the screen when the device is in landscape mode. However, we would like to keep the View unchanged when the device is in portrait mode.

We will divide the buttons and place them in two subviews. The first subview will contain the first 4 by 4 buttons (that is the digit buttons, “.”, “Enter”, “\*”, “/”, “+” and “-”). This subview will display on the upper side of the screen in portrait mode and on the left side of the screen in landscape mode.

The second subview will contain all the other buttons and the labels that display the x, y and z variables values. This subview will display on the lower side of the screen in portrait mode and on the right side of the screen in landscape mode.

Open the Document Outline in order to add and set these subviews properly. Also expand the `UIViEW` to see our current view hierarchy.



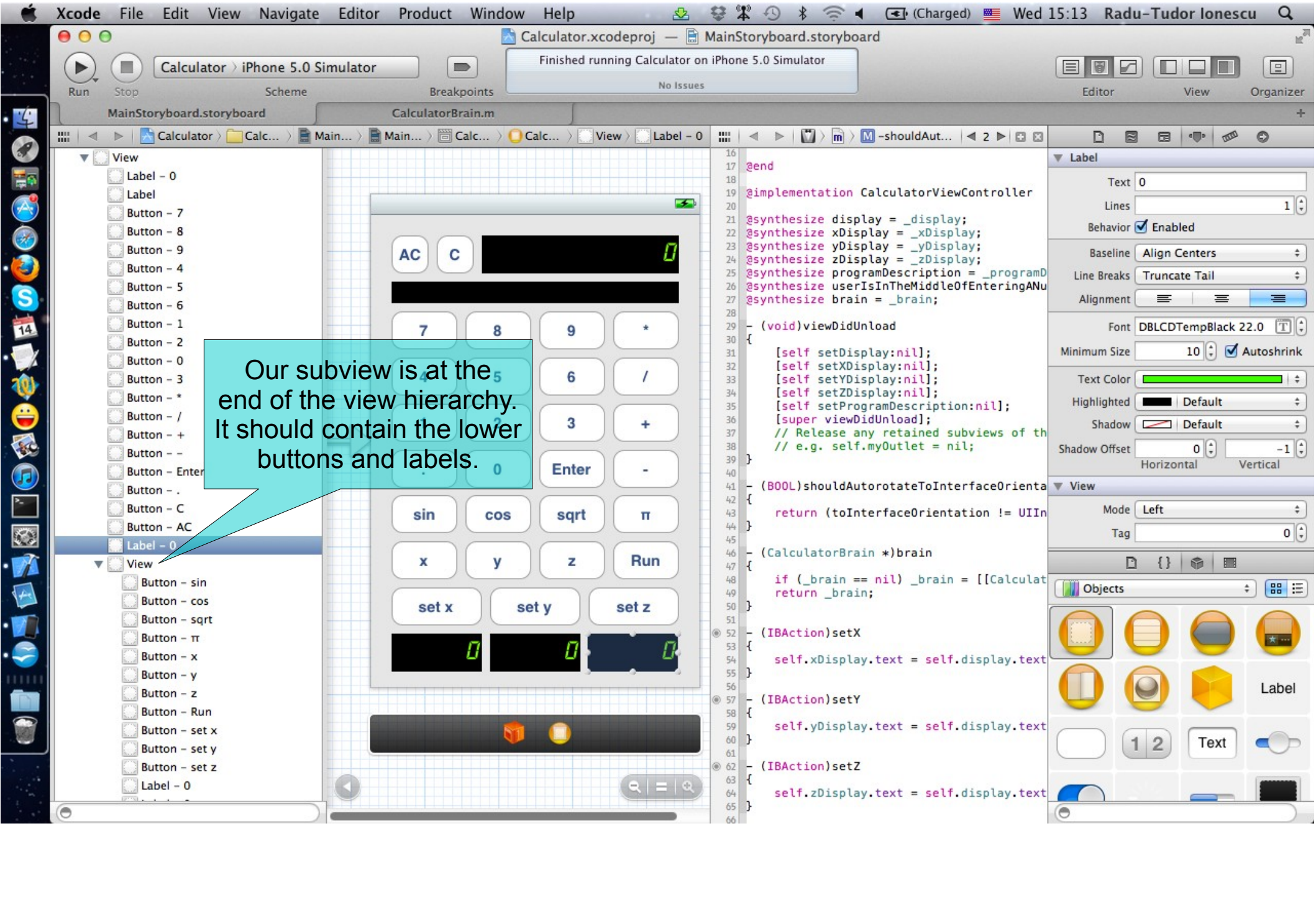
Expand this to see our view hierarchy. You will find all our labels and buttons listed here.

Show and hide the Document Outline from here.

## Task 2

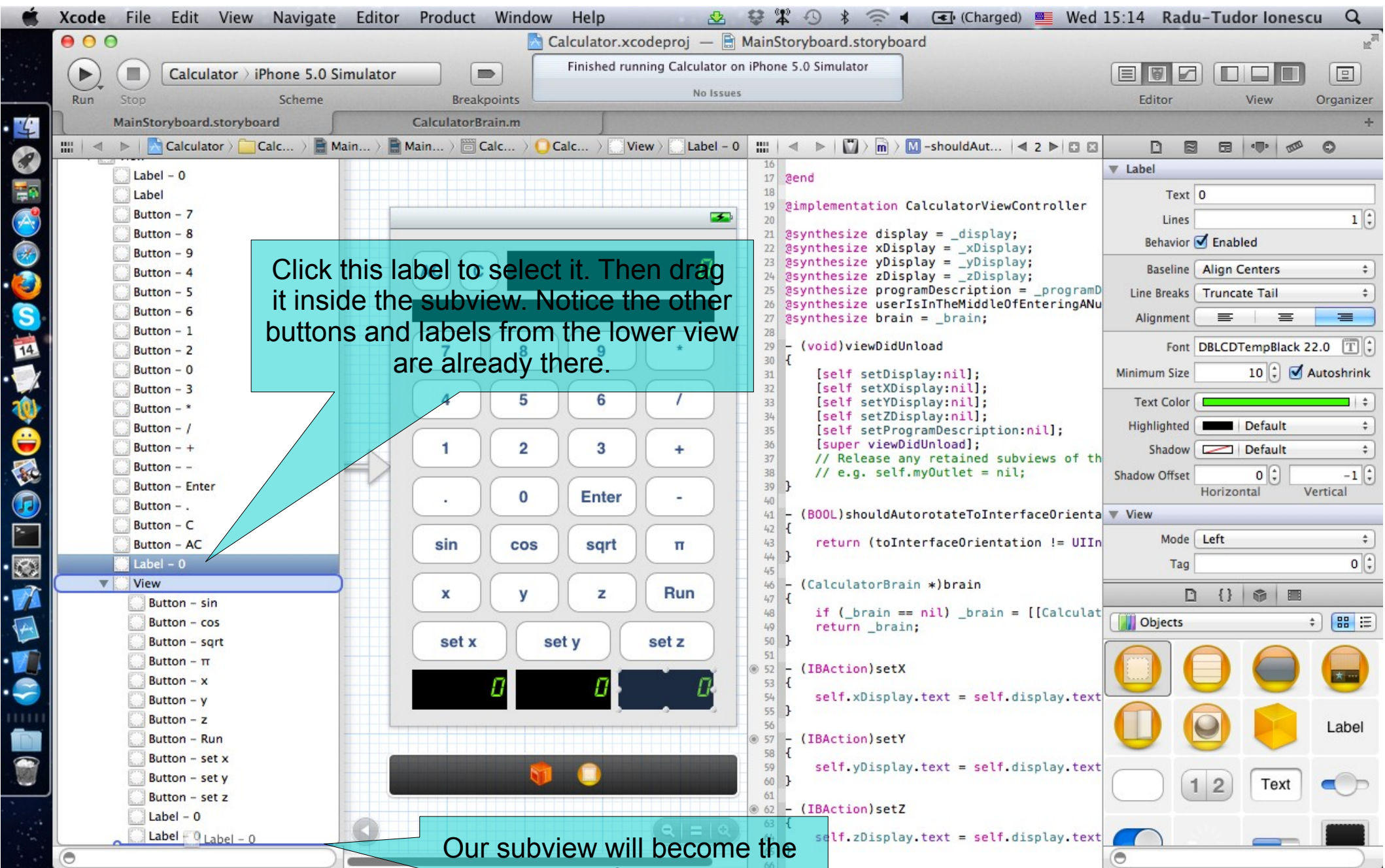
Task: Allow your Calculator to display its View in portrait and in landscape mode using autorotation.

11. Drag and drop an `UIView` from the Object Library.
12. In Size Inspector set its frame to `X=0, Y=273, Width=320` and `Height=186`. Note that this is the second subview that will contain the lower buttons.
13. Select the Attributes Inspector and set the Background to Clear color. This means we want our subview to have a transparent background. Deselect the Opaque checkbox (our view isn't opaque anymore). Note that selecting Opaque for non-transparent or non-translucent views is a performance optimization.
14. Make sure this subview is at the end of the view hierarchy. If not, in Document Outline click and drag this subview to the end of the hierarchy.
15. In Document Outline drag the lower buttons in the subview. You will have to reposition each button to its original position.



Our subview is at the end of the view hierarchy. It should contain the lower buttons and labels.

```
16
17 @end
18
19 @implementation CalculatorViewController
20
21 @synthesize display = _display;
22 @synthesize xDisplay = _xDisplay;
23 @synthesize yDisplay = _yDisplay;
24 @synthesize zDisplay = _zDisplay;
25 @synthesize programDescription = _programD
26 @synthesize userIsInTheMiddleOfEnteringANU
27 @synthesize brain = _brain;
28
29 - (void)viewDidLoad
30 {
31     [self setDisplay:nil];
32     [self setXDisplay:nil];
33     [self setYDisplay:nil];
34     [self setZDisplay:nil];
35     [self setProgramDescription:nil];
36     [super viewDidLoad];
37     // Release any retained subviews of th
38     // e.g. self.myOutlet = nil;
39 }
40
41 - (BOOL)shouldAutorotateToInterfaceOrientation
42 {
43     return (toInterfaceOrientation != UIIn
44 }
45
46 - (CalculatorBrain *)brain
47 {
48     if (_brain == nil) _brain = [[Calculat
49     return _brain;
50 }
51
52 - (IBAction)setX
53 {
54     self.xDisplay.text = self.display.text
55 }
56
57 - (IBAction)setY
58 {
59     self.yDisplay.text = self.display.text
60 }
61
62 - (IBAction)setZ
63 {
64     self.zDisplay.text = self.display.text
65 }
66
```



Click this label to select it. Then drag it inside the subview. Notice the other buttons and labels from the lower view are already there.

Our subview will become the superview of this label when you drag it here.

```
16 @end
17
18
19 @implementation CalculatorViewController
20
21 @synthesize display = _display;
22 @synthesize xDisplay = _xDisplay;
23 @synthesize yDisplay = _yDisplay;
24 @synthesize zDisplay = _zDisplay;
25 @synthesize programDescription = _programD
26 @synthesize userIsInTheMiddleOfEnteringANU
27 @synthesize brain = _brain;
28
29 - (void)viewDidLoad
30 {
31     [self setDisplay:nil];
32     [self setXDisplay:nil];
33     [self setYDisplay:nil];
34     [self setZDisplay:nil];
35     [self setProgramDescription:nil];
36     [super viewDidLoad];
37     // Release any retained subviews of th
38     // e.g. self.myOutlet = nil;
39 }
40
41 - (BOOL)shouldAutorotateToInterfaceOrientation
42 {
43     return (toInterfaceOrientation != UIIn
44 }
45
46 - (CalculatorBrain *)brain
47 {
48     if (_brain == nil) _brain = [[Calculat
49     return _brain;
50 }
51
52 - (IBAction)setX
53 {
54     self.xDisplay.text = self.display.text
55 }
56
57 - (IBAction)setY
58 {
59     self.yDisplay.text = self.display.text
60 }
61
62 - (IBAction)setZ
63 {
64     self.zDisplay.text = self.display.text
65 }
66
```

**Label**

Text: 0

Lines: 1

Behavior:  Enabled

Baseline: Align Centers

Line Breaks: Truncate Tail

Alignment: [Left] [Center] [Right]

Font: DBLCDTempBlack 22.0

Minimum Size: 10  Autoshrink

Text Color: [Color Picker]

Highlighted: [Color Picker] Default

Shadow: [Color Picker] Default

Shadow Offset: 0 [Horizontal] -1 [Vertical]

**View**

Mode: Left

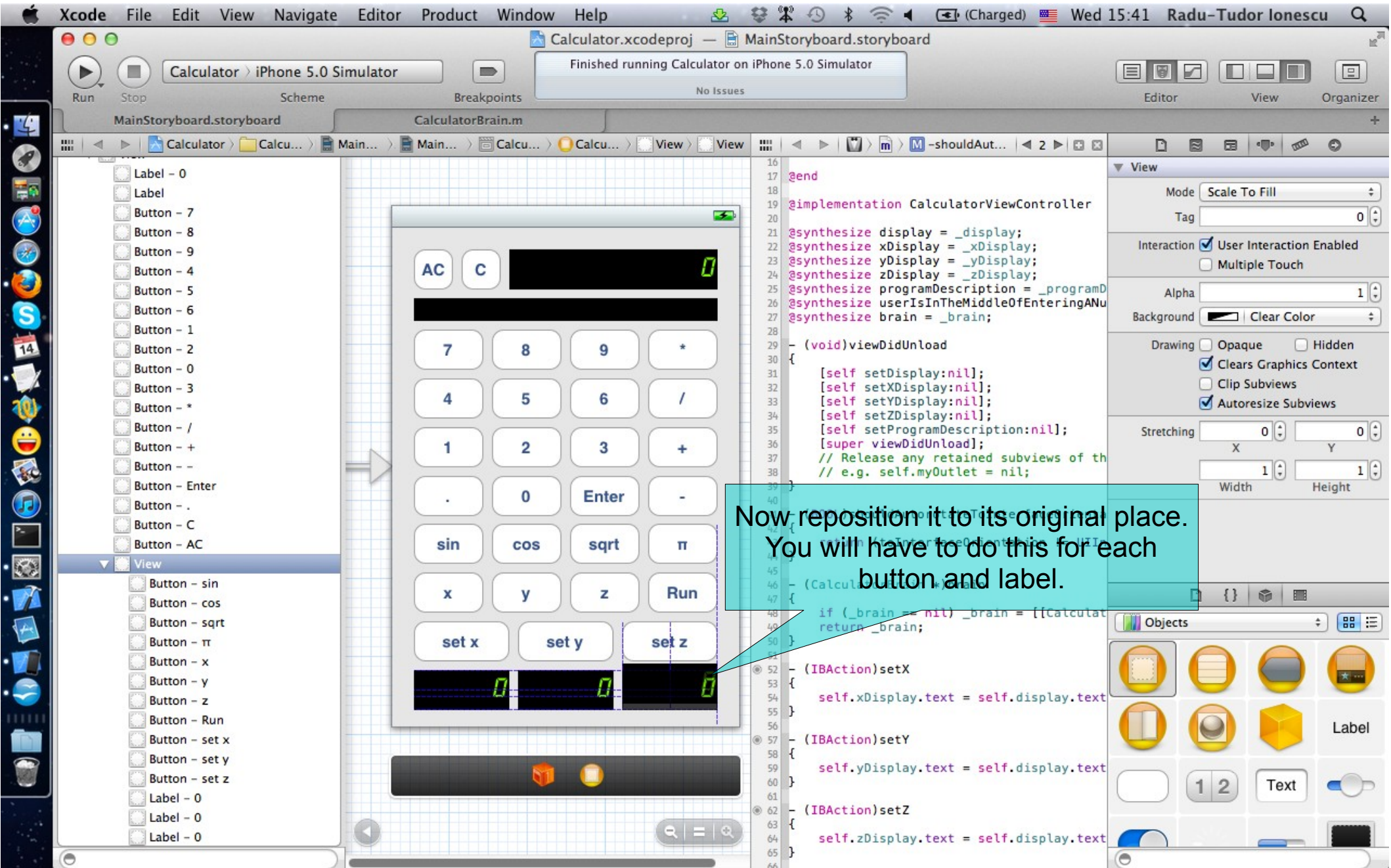
Tag: 0

**Objects**

[Library icons]

Label





Now reposition it to its original place. You will have to do this for each button and label.

## Task 2

Task: Allow your Calculator to display its View in portrait and in landscape mode using autorotation.

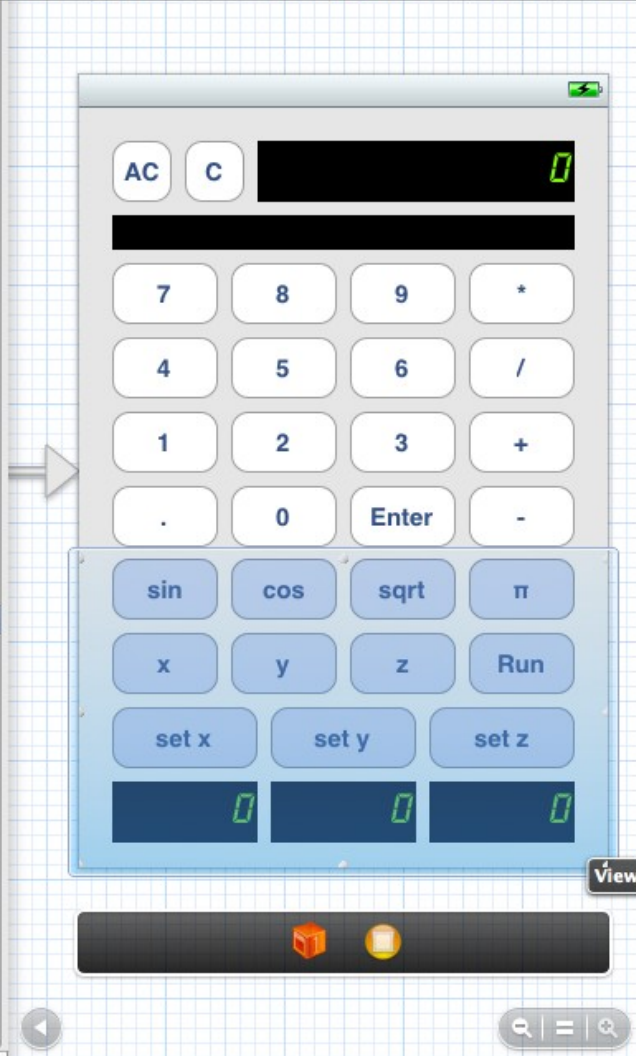
16. Select the CalculatorViewController.h in Assistant Editor.
17. CTRL-drag from this subview to your Controller interface to create an outlet.
18. Name this outlet “lowerRightView” and select the Weak Storage for it.

The next screenshot shows how this `@property` should look like.

Calculator.xcodeproj — MainStoryboard.storyboard  
Attaching to Calculator  
No Issues

Calculator > iPhone 5.0 Simulator  
Run Stop Scheme Breakpoints

- Label - 0
- Label
- Button - 7
- Button - 8
- Button - 9
- Button - 4
- Button - 5
- Button - 6
- Button - 1
- Button - 2
- Button - 0
- Button - 3
- Button - \*
- Button - /
- Button - +
- Button - -
- Button - Enter
- Button - .
- Button - C
- Button - AC
- View
  - Button - sin
  - Button - cos
  - Button - sqrt
  - Button -  $\pi$
  - Button - x
  - Button - y
  - Button - z
  - Button - Run
  - Button - set x
  - Button - set y
  - Button - set z
  - Label - 0
  - Label - 0
  - Label - 0



Automatic > CalculatorViewController.h > lowerRightView

```
1 //
2 // CalculatorViewController.h
3 // Calculator
4 //
5 // Created by Radu-Tudor Ionescu on 2/27/12.
6 // Copyright (c) 2012 __MyCompanyName__. All rights reserved.
7 //
8
9 #import <UIKit/UIKit.h>
10
11 @interface CalculatorViewController : UIViewController
12
13 @property (weak, nonatomic) IBOutlet UIView *lowerRightView;
14 @property (weak, nonatomic) IBOutlet UILabel *display;
15 @property (weak, nonatomic) IBOutlet UILabel *xDisplay;
16 @property (weak, nonatomic) IBOutlet UILabel *yDisplay;
17 @property (weak, nonatomic) IBOutlet UILabel *zDisplay;
18 @property (weak, nonatomic) IBOutlet UILabel *programDescription;
19
20 @end
21
```

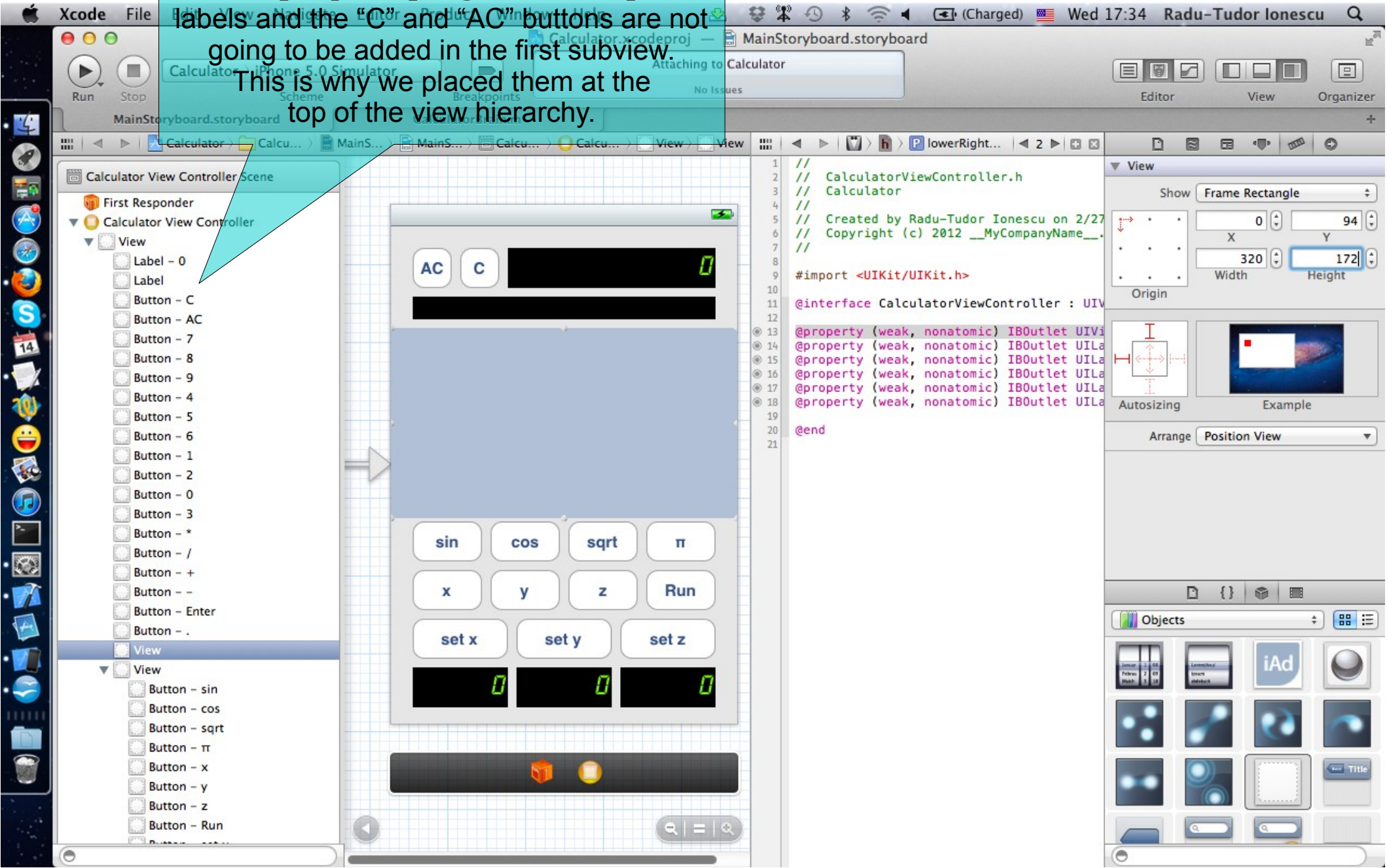
## Task 2

Task: Allow your Calculator to display its View in portrait and in landscape mode using autorotation.

19. In Document Outline we should place the `display` and `programDescription` labels at the beginning of the view hierarchy.
20. Do the same thing for the “C” and “AC” buttons.
21. Drag another `UIView` from the Object Library.
22. Make sure you place it before the `lowerRightView` in the view hierarchy.
23. Select the Size Inspector and set the frame to `X=0, Y=94, Width=320` and `Height=172`.

It should look like in the next screenshot.

The display and programDescription labels and the "C" and "AC" buttons are not going to be added in the first subview. This is why we placed them at the top of the view hierarchy.



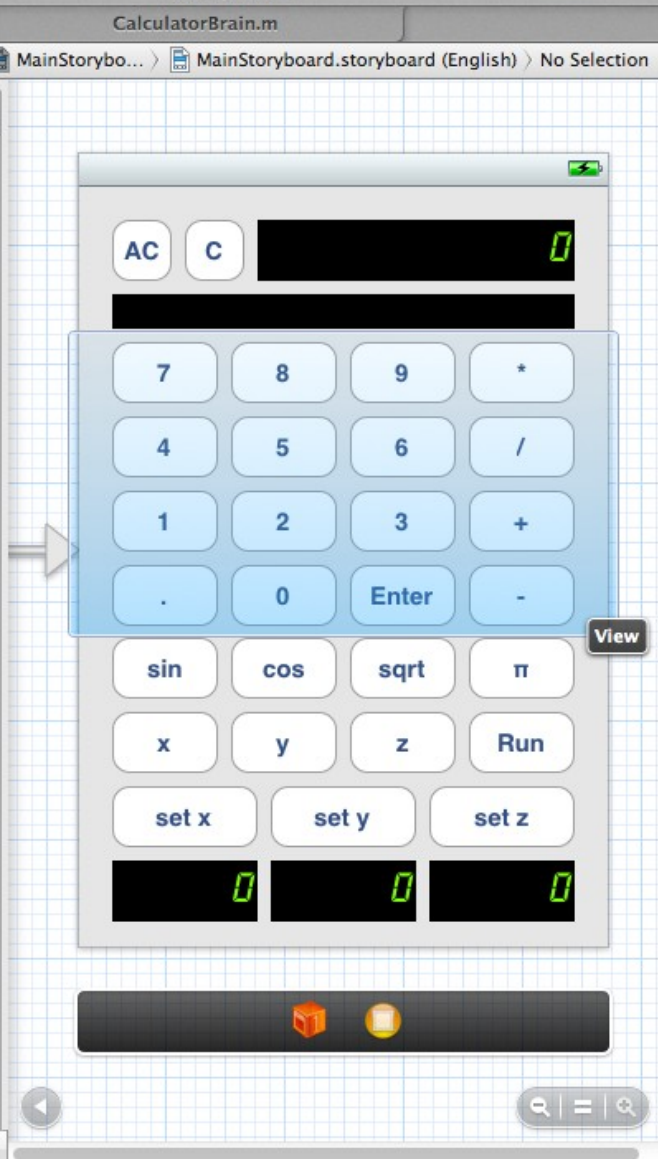
## Task 2

Task: Allow your Calculator to display its View in portrait and in landscape mode using autorotation.

24. Select the Attributes Inspector and set the Background to Clear color. Also deselect the Opaque checkbox.
25. In Document outline drag the digit buttons and reposition them to their original place. Do the same thing for “.”, “Enter” and the remaining operation buttons.
26. Select the CalculatorViewController.h in Assistant Editor.
27. CTRL-drag from this subview to your Controller interface to create an outlet.
28. Name this outlet “upperLeftView” and select the Weak Storage for it.

Everything should look like as in the next screenshot.

- Calculator View Controller Scene
  - First Responder
    - Calculator View Controller
      - View
        - Label - 0
        - Label
        - Button - C
        - Button - AC
        - View
          - Button - 7
          - Button - 8
          - Button - 9
          - Button - 4
          - Button - 5
          - Button - 6
          - Button - 1
          - Button - 2
          - Button - 3
          - Button - 0
          - Button - \*
          - Button - /
          - Button - +
          - Button - -
          - Button - .
          - Button - Enter
        - View
          - Button - sin
          - Button - cos
          - Button - sqrt
          - Button -  $\pi$
          - Button - x
          - Button - y
          - Button - z
          - Button - Run



```
1 //
2 // CalculatorViewController.h
3 // Calculator
4 //
5 // Created by Radu-Tudor Ionescu on 2/27/12.
6 // Copyright (c) 2012 __MyCompanyName__. All rights reserved.
7 //
8
9 #import <UIKit/UIKit.h>
10
11 @interface CalculatorViewController : UIViewController
12
13 @property (weak, nonatomic) IBOutlet UIView *upperLeftView;|
14 @property (weak, nonatomic) IBOutlet UIView *lowerRightView;
15 @property (weak, nonatomic) IBOutlet UILabel *display;
16 @property (weak, nonatomic) IBOutlet UILabel *xDisplay;
17 @property (weak, nonatomic) IBOutlet UILabel *yDisplay;
18 @property (weak, nonatomic) IBOutlet UILabel *zDisplay;
19 @property (weak, nonatomic) IBOutlet UILabel *programDescription;
20
21 @end
22
```

## Task 2

Task: Allow your Calculator to display its View in portrait and in landscape mode using autorotation.

29. This is all we can do from Interface Builder to adjust our View for different device orientations automatically. Next we are going to resize the `upperLeftView` and the `lowerRightView` from code so that the buttons inside them display nicely on all device orientations. We can hide the Document Outline now.

30. Open the Controller's implementation file in Assistant Editor.

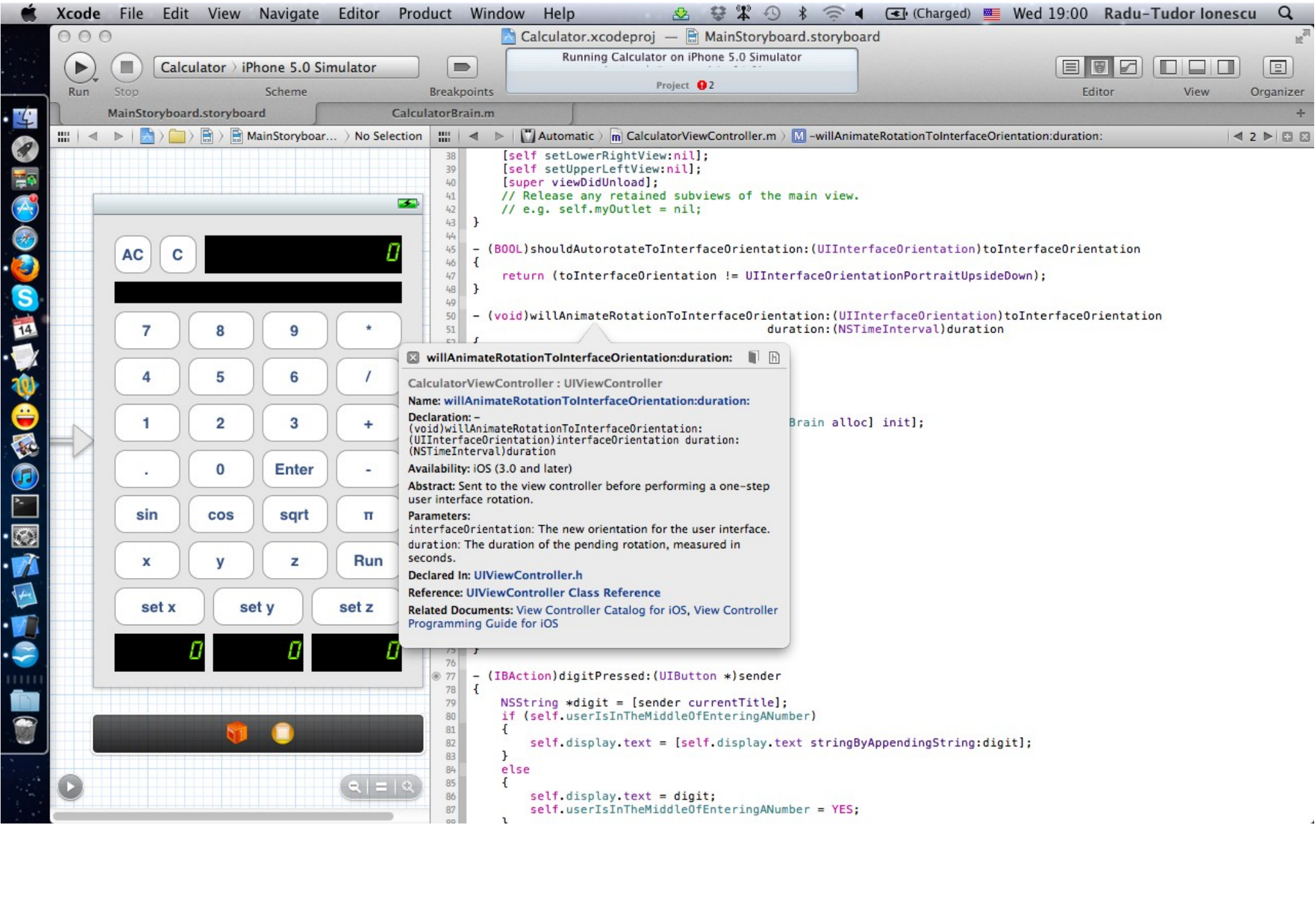
31. We should implement the method that is called from within the animation block used to rotate the view:

```
willAnimateRotationToInterfaceOrientation:duration:
```

Add the implementation for this method right after the `shouldAutorotateToInterfaceOrientation:` method.

32. Hold down the option key and click on the method signature to see additional information about it.





```
38 [self setLowerRightView:nil];
39 [self setUpperLeftView:nil];
40 [super viewDidUnload];
41 // Release any retained subviews of the main view.
42 // e.g. self.myOutlet = nil;
43 }
44
45 - (BOOL)shouldAutorotateToInterfaceOrientation:(UIInterfaceOrientation)toInterfaceOrientation
46 {
47     return (toInterfaceOrientation != UIInterfaceOrientationPortraitUpsideDown);
48 }
49
50 - (void)willAnimateRotationToInterfaceOrientation:(UIInterfaceOrientation)toInterfaceOrientation
51     duration:(NSTimeInterval)duration
```

**willAnimateRotationToInterfaceOrientation:duration:**

CalculatorViewController: UIViewController

**Name:** willAnimateRotationToInterfaceOrientation:duration:

**Declaration:** - (void)willAnimateRotationToInterfaceOrientation:(UIInterfaceOrientation)interfaceOrientation duration:(NSTimeInterval)duration

**Availability:** iOS (3.0 and later)

**Abstract:** Sent to the view controller before performing a one-step user interface rotation.

**Parameters:**  
interfaceOrientation: The new orientation for the user interface.  
duration: The duration of the pending rotation, measured in seconds.

**Declared In:** UIViewController.h

**Reference:** UIViewController Class Reference

**Related Documents:** View Controller Catalog for iOS, View Controller Programming Guide for iOS

```
75
76
77 - (IBAction)digitPressed:(UIButton *)sender
78 {
79     NSString *digit = [sender currentTitle];
80     if (self.userIsInTheMiddleOfEnteringANumber)
81     {
82         self.display.text = [self.display.text stringByAppendingString:digit];
83     }
84     else
85     {
86         self.display.text = digit;
87         self.userIsInTheMiddleOfEnteringANumber = YES;
88     }
89 }
```

## Task 2

Task: Allow your Calculator to display its View in portrait and in landscape mode using autorotation.

33. Now we need to adjust the View for every device orientation. Note that there are four defined orientations that correspond to the four general ways that the iPhone can be held:

```
UIInterfaceOrientationPortrait  
UIInterfaceOrientationPortraitUpsideDown  
UIInterfaceOrientationLandscapeLeft  
UIInterfaceOrientationLandscapeRight
```

We will not support the portrait upside down orientation. We can also group the landscape orientations because the View will be the same for both landscape left and landscape right orientations.

Thus, we will have two different ways of displaying our View (portrait and landscape). We only have to adjust the frame of the `upperLeftView` and `lowerRightView`.

Make sure your code is similar to the code highlighted in the next screenshot.

Xcode File Edit View Navigate Editor Product Window Help

Calculator.xcodeproj — MainStoryboard.storyboard

Build Succeeded | Yesterday at 19:15 PM

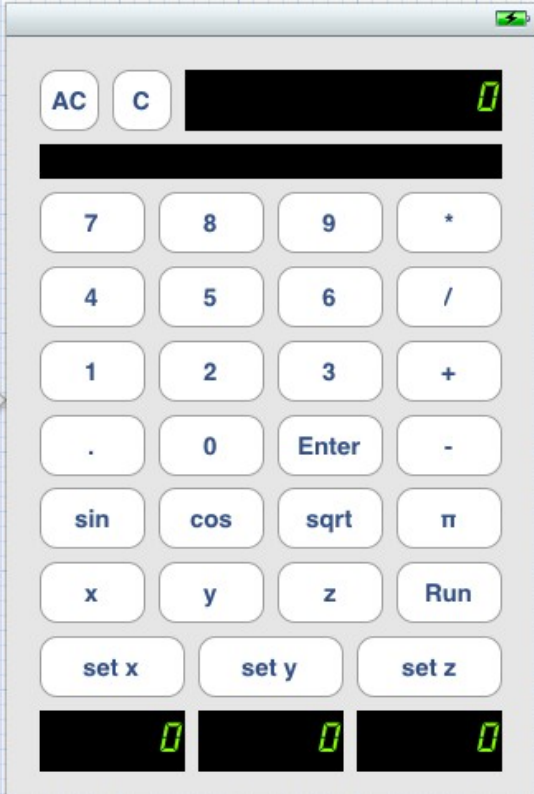
No Issues

Calculator > iPhone 5.0 Simulator

Run Stop Scheme Breakpoints Editor View Organizer

MainStoryboard.storyboard CalculatorBrain.m

MainStoryboard... > No Selection Automatic > CalculatorViewController.m > -willAnimateRotationToInterfaceOrientation:duration:



```
51 }
52
53 - (BOOL)shouldAutorotateToInterfaceOrientation:(UIInterfaceOrientation)toInterfaceOrientation
54 {
55     return (toInterfaceOrientation != UIInterfaceOrientationPortraitUpsideDown);
56 }
57
58 - (void)willAnimateRotationToInterfaceOrientation:(UIInterfaceOrientation)toInterfaceOrientation
59     duration:(NSTimeInterval)duration
60 {
61     if (toInterfaceOrientation == UIInterfaceOrientationPortrait)
62     {
63         self.upperLeftView.frame = CGRectMake(0, 90, 320, 172);
64         self.lowerRightView.frame = CGRectMake(0, 273, 320, 186);
65     }
66     else if (toInterfaceOrientation == UIInterfaceOrientationLandscapeLeft ||
67             toInterfaceOrientation == UIInterfaceOrientationLandscapeRight)
68     {
69         self.upperLeftView.frame = CGRectMake(5, 100, 240, 172);
70         self.lowerRightView.frame = CGRectMake(235, 100, 240, 172);
71     }
72 }
73
74 - (CalculatorBrain *)brain
75 {
76     if (_brain == nil) _brain = [[CalculatorBrain alloc] init];
77     return _brain;
78 }
79
80 - (IBAction)setX
81 {
82     self.xDisplay.text = self.display.text;
83 }
84
85 - (IBAction)setY
86 {
87     self.yDisplay.text = self.display.text;
88 }
89
90 - (IBAction)setZ
91 {
92     self.zDisplay.text = self.display.text;
93 }
94
95 - (IBAction)digitPressed:(UIButton *)sender
96 {
97     NSString *digit = [sender currentTitle];
98     if (self.userIsInTheMiddleOfEnteringANumber)
99     {
100         self.display.text = [self.display.text stringByAppendingString:digit];
101     }
```

## Task 2

Task: Allow your Calculator to display its View in portrait and in landscape mode using autorotation.

34. Run the application in iOS Simulator. Notice that the buttons align nicely on the landscape orientation, but they are too wide and still don't fit on the screen (this time horizontally).

We need to solve this issue by rescaling the `upperLeftView` and `lowerRightView`.

35. Stop running the application.

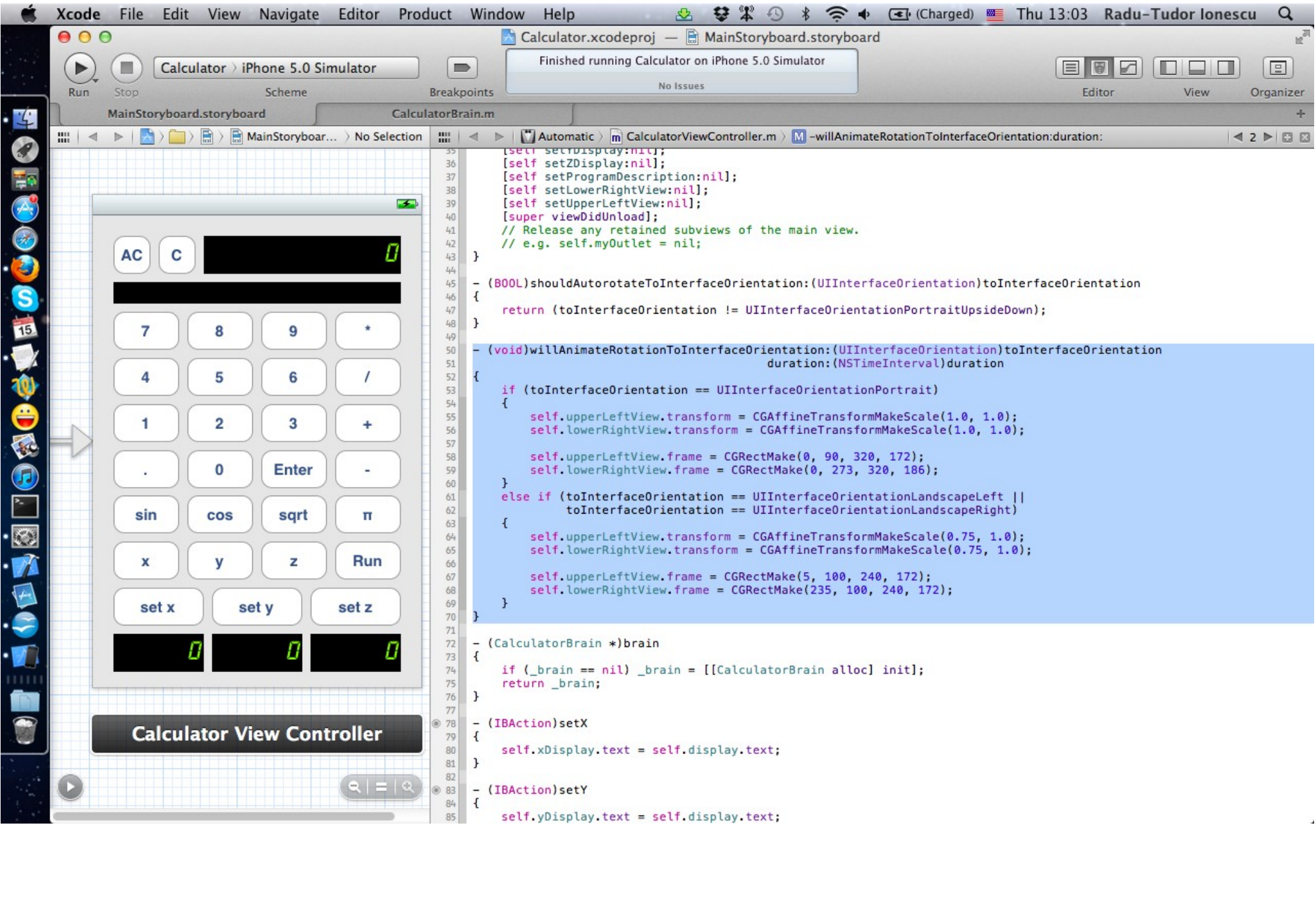
## Task 2

Task: Allow your Calculator to display its View in portrait and in landscape mode using autorotation.

36. `UIView` objects have a `transform` @property that enables us to make affine transformations. Thus, we can translate, scale or rotate the any view by setting its `transform`. The Core Graphics framework has a few helper C functions that start with `CGAffineTransformMake...` and allow us to quickly create transformations (that are C structs, not Objective-C objects) from scratch. There are also other helper functions to manipulate existing transformations that have the `CGAffineTransform...` prefix.

We are going to use the `CGAffineTransformMakeScale` function to rescale the `upperLeftView` and the `lowerLeftView`. We only have to rescale them horizontally to 75% in landscape mode, but we have to rescale them back to 100% on portrait mode. We also have to transform the subviews before setting their `frame` (because rescaling can modify the `frame`).

The next screenshot shows how your code should look like.



## Task 2

Task: Allow your Calculator to display its View in portrait and in landscape mode using autorotation.

37. All done! Run the application in iOS Simulator and try different device orientations. Your View should now look good in any device orientation (portrait or landscape).

Also notice the automatic animation that iOS does for us when the device orientation changes.

38. Stop running the application.

## Task 3

**Task:** Add gestures for the most important actions of your Calculator.

1. We are going to add a swipe gesture recognizer for the Clear button and a tap gesture recognizer for the Enter button.

Open the MainStoryboard.storyboard tab in Xcode.

2. Open the Object Library and select the List View. Search for a Swipe Gesture Recognizer in the Object Library.
3. Drag a Swipe Gesture Recognizer over the View (make sure the view highlights as in the next screenshot when you drag it).
4. Notice the Gesture Recognizer appears next to the CalculatorViewController object in the bar placed right under the View.
5. Right-click on the Gesture Recognizer and make sure that gestureRecognizers in Referencing Outlet Collections is associated with the View.



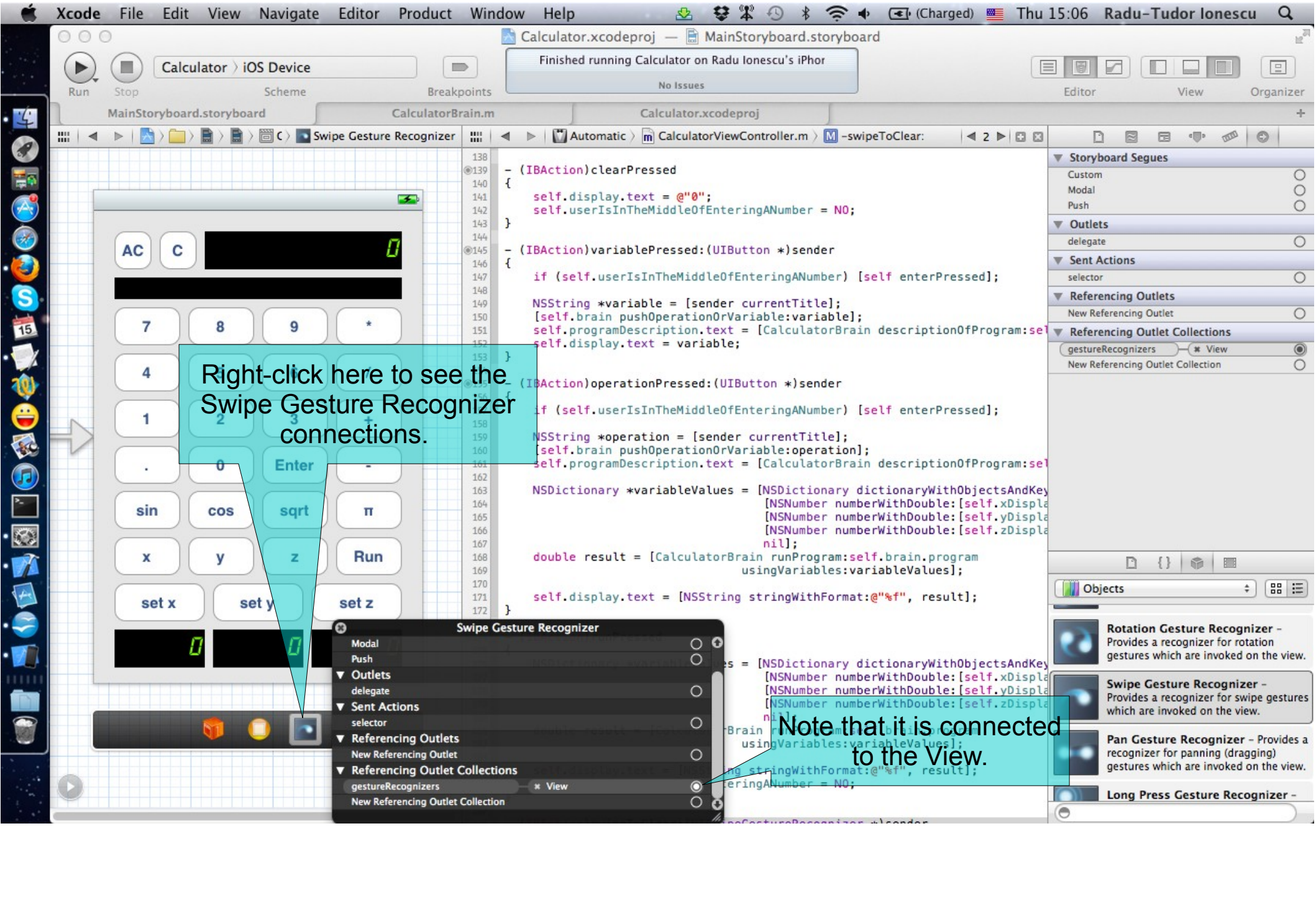
Make sure the View is highlighted.  
This also creates an association between the recognizer and your View. Basically, we want to recognize gestures on the entire screen.

Drag the Swipe Gesture Recognizer from the Object Library to your View.

```
138  
139 - (IBAction)clearPressed  
140 {  
141     self.display.text = @"0";  
142     self.userIsInTheMiddleOfEnteringANumber = NO;  
143 }  
144  
145 - (IBAction)variablePressed:(UIButton *)sender  
146 {  
147     if (self.userIsInTheMiddleOfEnteringANumber) [self enterPressed];  
148  
149     NSString *variable = [sender currentTitle];  
150     [self.brain pushOperationOrVariable:variable];  
151     self.programDescription.text = [CalculatorBrain descriptionOfProgram:self.brain.program];  
152     self.display.text = variable;  
153 }  
154  
155 - (IBAction)operationPressed:(UIButton *)sender  
156 {  
157     if (self.userIsInTheMiddleOfEnteringANumber) [self enterPressed];  
158  
159     NSString *operation = [sender currentTitle];  
160     [self.brain pushOperationOrVariable:operation];  
161     self.programDescription.text = [CalculatorBrain descriptionOfProgram:self.brain.program];  
162  
163     NSDictionary *variableValues = [NSDictionary dictionaryWithObjectsAndKeys:  
164         [NSNumber numberWithInt:[self.xDisplay.text intValue]],  
165         [NSNumber numberWithInt:[self.yDisplay.text intValue]],  
166         [NSNumber numberWithInt:[self.zDisplay.text intValue]],  
167         nil];  
168     double result = [CalculatorBrain runProgram:self.brain.program  
169         usingVariables:variableValues];  
170  
171     self.display.text = [NSString stringWithFormat:@"%f", result];  
172 }  
173  
174 - (IBAction)runPressed  
175 {  
176     NSDictionary *variableValues = [NSDictionary dictionaryWithObjectsAndKeys:  
177         [NSNumber numberWithInt:[self.xDisplay.text intValue]],  
178         [NSNumber numberWithInt:[self.yDisplay.text intValue]],  
179         [NSNumber numberWithInt:[self.zDisplay.text intValue]],  
180         nil];  
181     double result = [CalculatorBrain runProgram:self.brain.program  
182         usingVariables:variableValues];  
183  
184     self.display.text = [NSString stringWithFormat:@"%f", result];  
185     self.userIsInTheMiddleOfEnteringANumber = NO;  
186 }  
187  
188 - (IBAction)swipeToClear:(UITapGestureRecognizer *)sender
```

Objects

- Rotation Gesture Recognizer - Provides a recognizer for rotation gestures which are invoked on the view.
- Swipe Gesture Recognizer - Provides a recognizer for swipe gestures which are invoked on the view.
- Pan Gesture Recognizer - Provides a recognizer for panning (dragging) gestures which are invoked on the view.
- Long Press Gesture Recognizer -



Right-click here to see the Swipe Gesture Recognizer connections.

Note that it is connected to the View.

```
138  
139 - (IBAction)clearPressed  
140 {  
141     self.display.text = @"0";  
142     self.userIsInTheMiddleOfEnteringANumber = NO;  
143 }  
144  
145 - (IBAction)variablePressed:(UIButton *)sender  
146 {  
147     if (self.userIsInTheMiddleOfEnteringANumber) [self enterPressed];  
148  
149     NSString *variable = [sender currentTitle];  
150     [self.brain pushOperationOrVariable:variable];  
151     self.programDescription.text = [CalculatorBrain descriptionOfProgram:self.brain.program];  
152     self.display.text = variable;  
153 }  
154  
155 - (IBAction)operationPressed:(UIButton *)sender  
156 {  
157     if (self.userIsInTheMiddleOfEnteringANumber) [self enterPressed];  
158  
159     NSString *operation = [sender currentTitle];  
160     [self.brain pushOperationOrVariable:operation];  
161     self.programDescription.text = [CalculatorBrain descriptionOfProgram:self.brain.program];  
162  
163     NSDictionary *variableValues = [NSDictionary dictionaryWithObjectsAndKey:  
164         [NSNumber numberWithInt:[self.xDisplay.text intValue]],  
165         [NSNumber numberWithInt:[self.yDisplay.text intValue]],  
166         [NSNumber numberWithInt:[self.zDisplay.text intValue]],  
167         nil];  
168     double result = [CalculatorBrain runProgram:self.brain.program  
169                     usingVariables:variableValues];  
170  
171     self.display.text = [NSString stringWithFormat:@"%f", result];  
172 }
```

Swipe Gesture Recognizer

- Modal
- Push
- Outlets
  - delegate
- Sent Actions
  - selector
- Referencing Outlets
  - New Referencing Outlet
- Referencing Outlet Collections
  - gestureRecognizers  \* View
  - New Referencing Outlet Collection

Storyboard Segues

- Custom
- Modal
- Push

Outlets

- delegate

Sent Actions

- selector

Referencing Outlets

- New Referencing Outlet

Referencing Outlet Collections

- gestureRecognizers  \* View
- New Referencing Outlet Collection

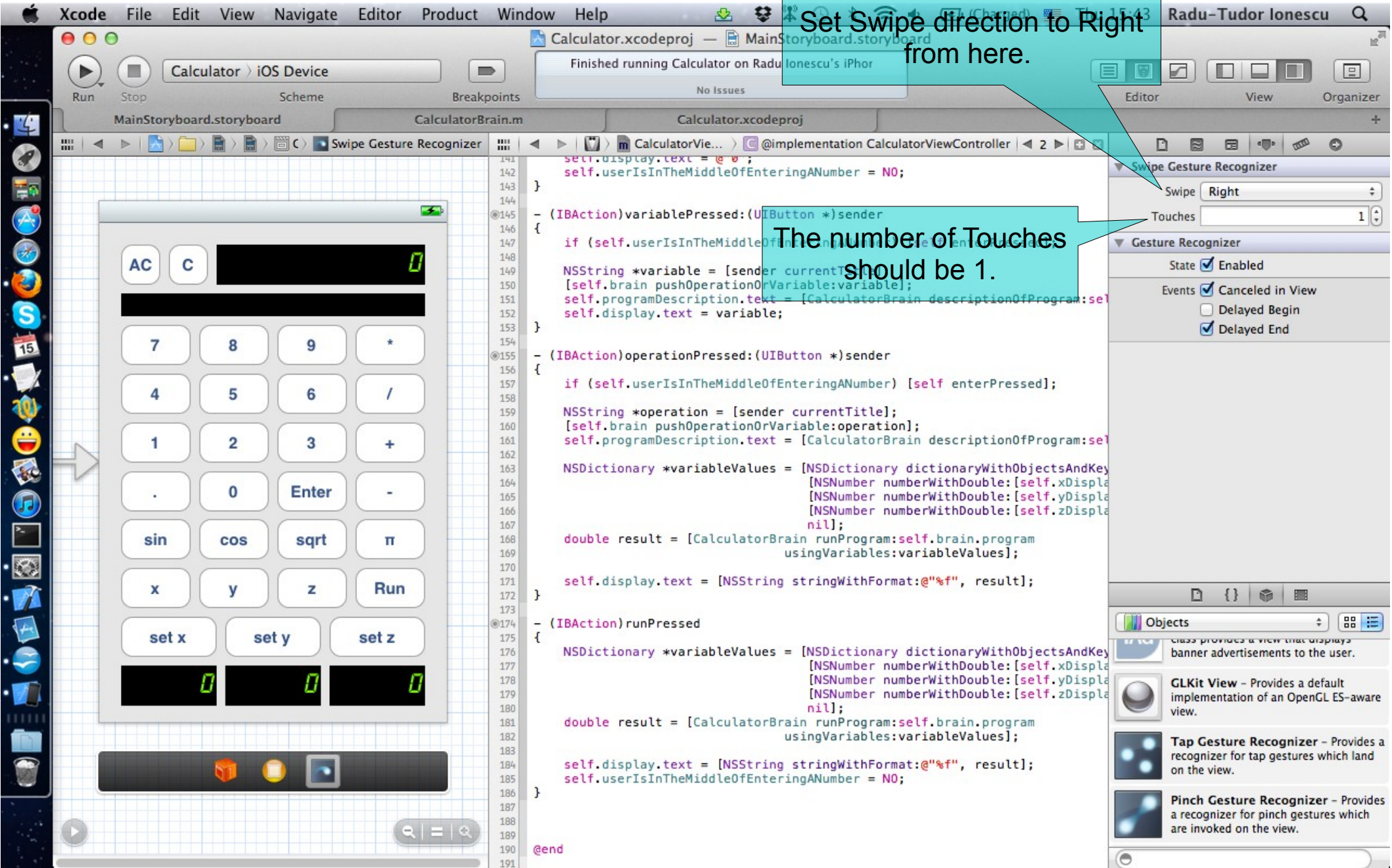
Objects

- Rotation Gesture Recognizer - Provides a recognizer for rotation gestures which are invoked on the view.
- Swipe Gesture Recognizer - Provides a recognizer for swipe gestures which are invoked on the view.
- Pan Gesture Recognizer - Provides a recognizer for panning (dragging) gestures which are invoked on the view.
- Long Press Gesture Recognizer -

## Task 3

Task: Add gestures for the most important actions of your Calculator.

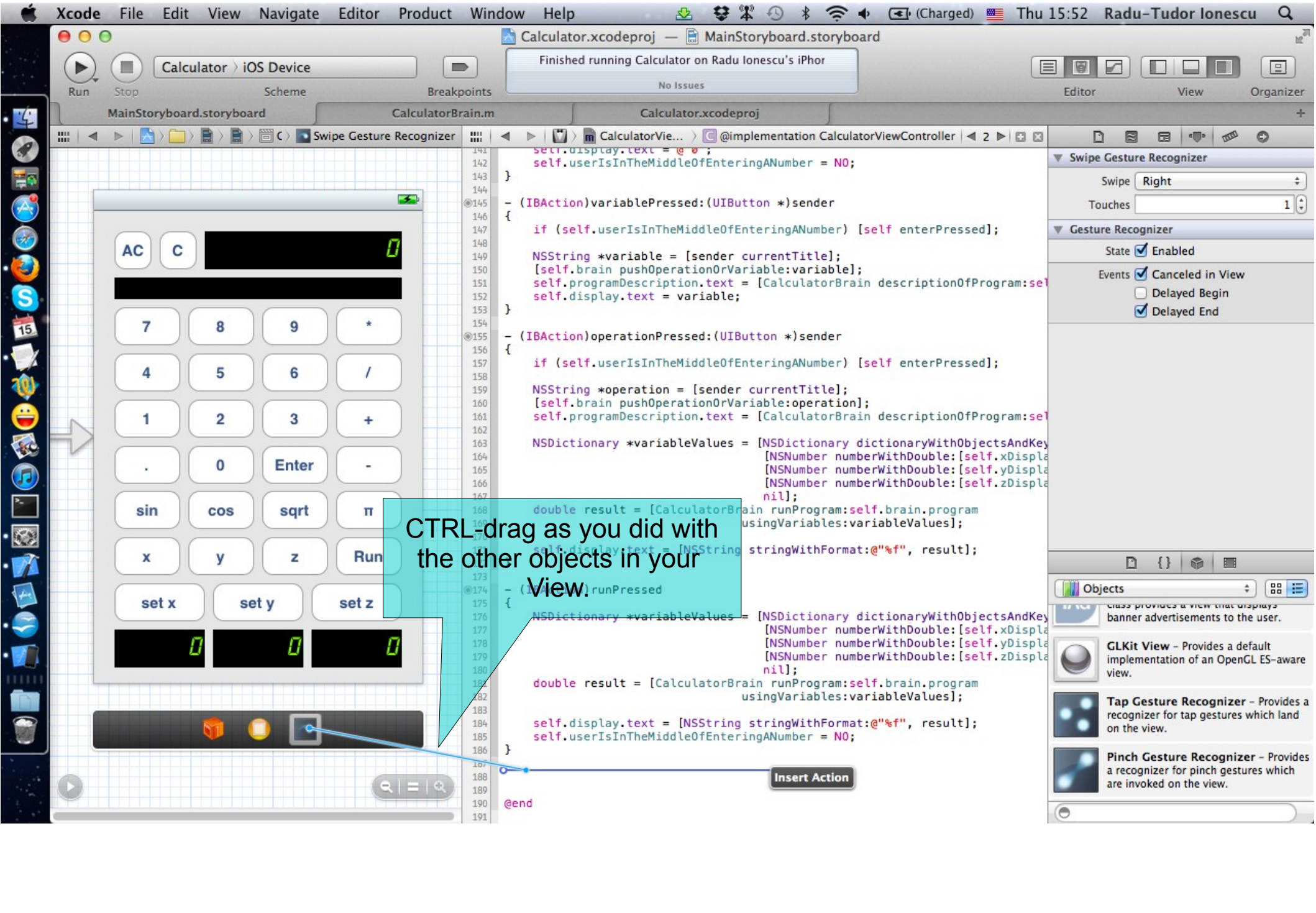
6. Open Attributes Inspector in Utilities area.
7. Select the Swipe Gesture Recognizer.
8. We want to clear the `display` with a single swipe from left to right. For this, we need to set the Swipe direction to Right.
9. Also make sure Touches (the number of fingers to make the gesture) is 1.



## Task 3

Task: Add gestures for the most important actions of your Calculator.

10. We can now assign an action to the gesture recognizer. Make sure CalculatorViewController.m is opened in Assistant Editor.
11. Scroll down to the bottom of the Controller's implementation (we are going to add the action there).
12. CTRL-drag from the Swipe Gesture Recognizer to the Controller's implementation right before `@end`.



CTRL-drag as you did with the other objects in your View.

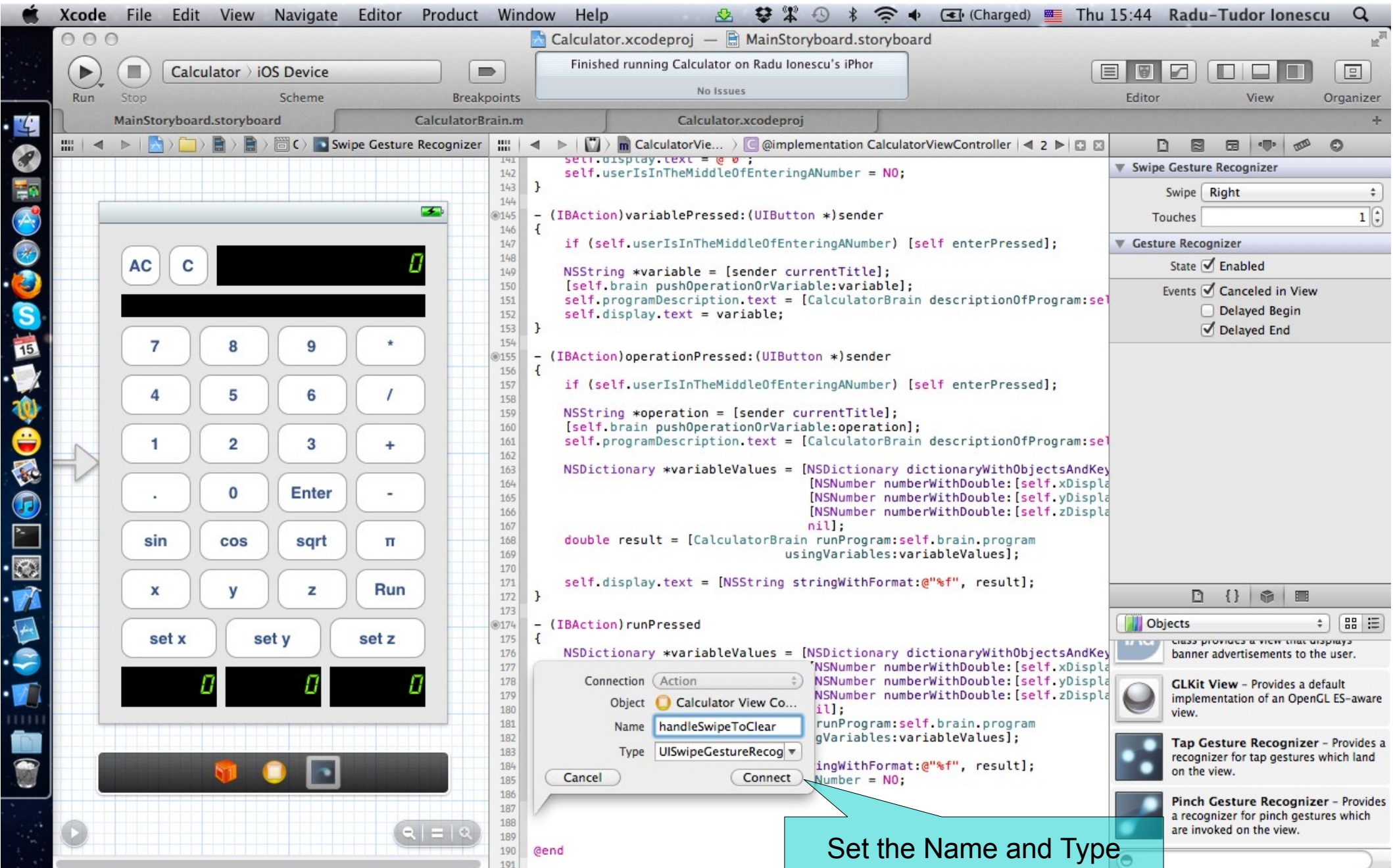
Insert Action

## Task 3

**Task:** Add gestures for the most important actions of your Calculator.

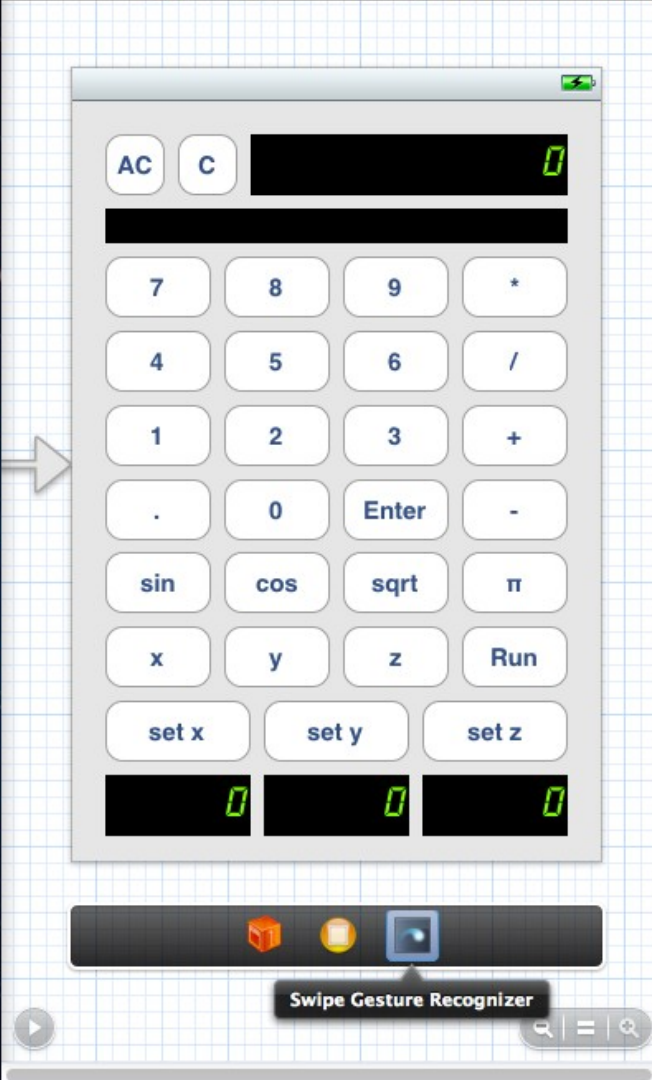
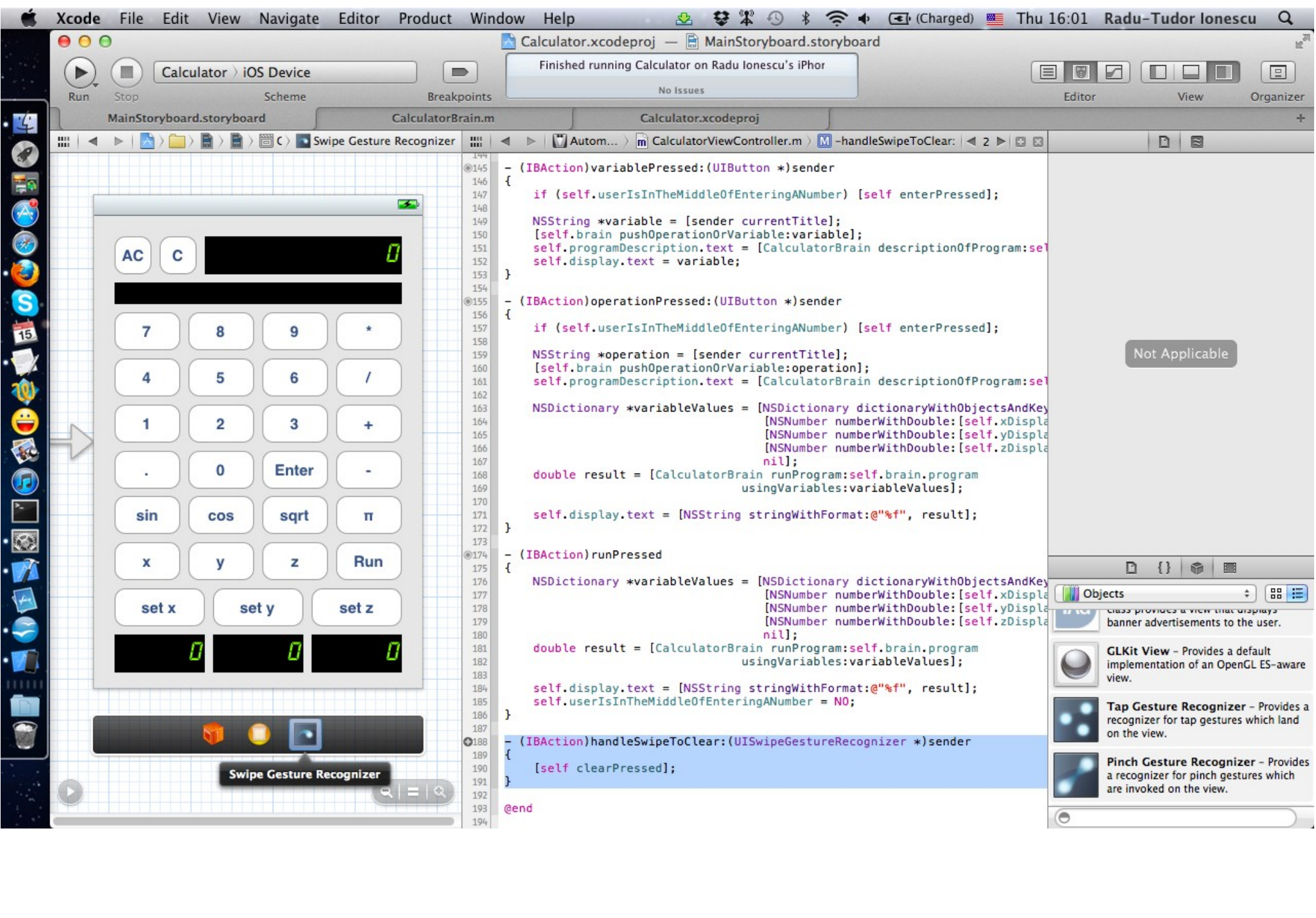
13. In the pop-up window that appears after CTRL-dragging set the method's name to “handleSwipeToClear”.
14. Select `UISwipeGestureRecognizer` for the argument Type.
15. Click Connect.
16. In the implementation of the `handleSwipeToClear:` we are going to simply send the `clearPressed` message on `self`.

The following two slides shows how to perform these steps.



Set the Name and Type and then click Connect.





```
145 - (IBAction)variablePressed:(UIButton *)sender
146 {
147     if (self.userIsInTheMiddleOfEnteringANumber) [self enterPressed];
148
149     NSString *variable = [sender currentTitle];
150     [self.brain pushOperationOrVariable:variable];
151     self.programDescription.text = [CalculatorBrain descriptionOfProgram:self
152     self.display.text = variable;
153 }
154
155 - (IBAction)operationPressed:(UIButton *)sender
156 {
157     if (self.userIsInTheMiddleOfEnteringANumber) [self enterPressed];
158
159     NSString *operation = [sender currentTitle];
160     [self.brain pushOperationOrVariable:operation];
161     self.programDescription.text = [CalculatorBrain descriptionOfProgram:self
162
163     NSDictionary *variableValues = [NSDictionary dictionaryWithObjectsAndKey
164     [NSNumber numberWithInt:[self.xDisplay
165     [NSNumber numberWithInt:[self.yDisplay
166     [NSNumber numberWithInt:[self.zDisplay
167     nil];
168
169     double result = [CalculatorBrain runProgram:self.brain.program
170     usingVariables:variableValues];
171
172     self.display.text = [NSString stringWithFormat:@"%f", result];
173 }
174
175 - (IBAction)runPressed
176 {
177     NSDictionary *variableValues = [NSDictionary dictionaryWithObjectsAndKey
178     [NSNumber numberWithInt:[self.xDisplay
179     [NSNumber numberWithInt:[self.yDisplay
180     [NSNumber numberWithInt:[self.zDisplay
181     nil];
182
183     double result = [CalculatorBrain runProgram:self.brain.program
184     usingVariables:variableValues];
185
186     self.display.text = [NSString stringWithFormat:@"%f", result];
187     self.userIsInTheMiddleOfEnteringANumber = NO;
188 }
189
190 - (IBAction)handleSwipeToClear:(UISwipeGestureRecognizer *)sender
191 {
192     [self clearPressed];
193 }
194 @end
```

Not Applicable

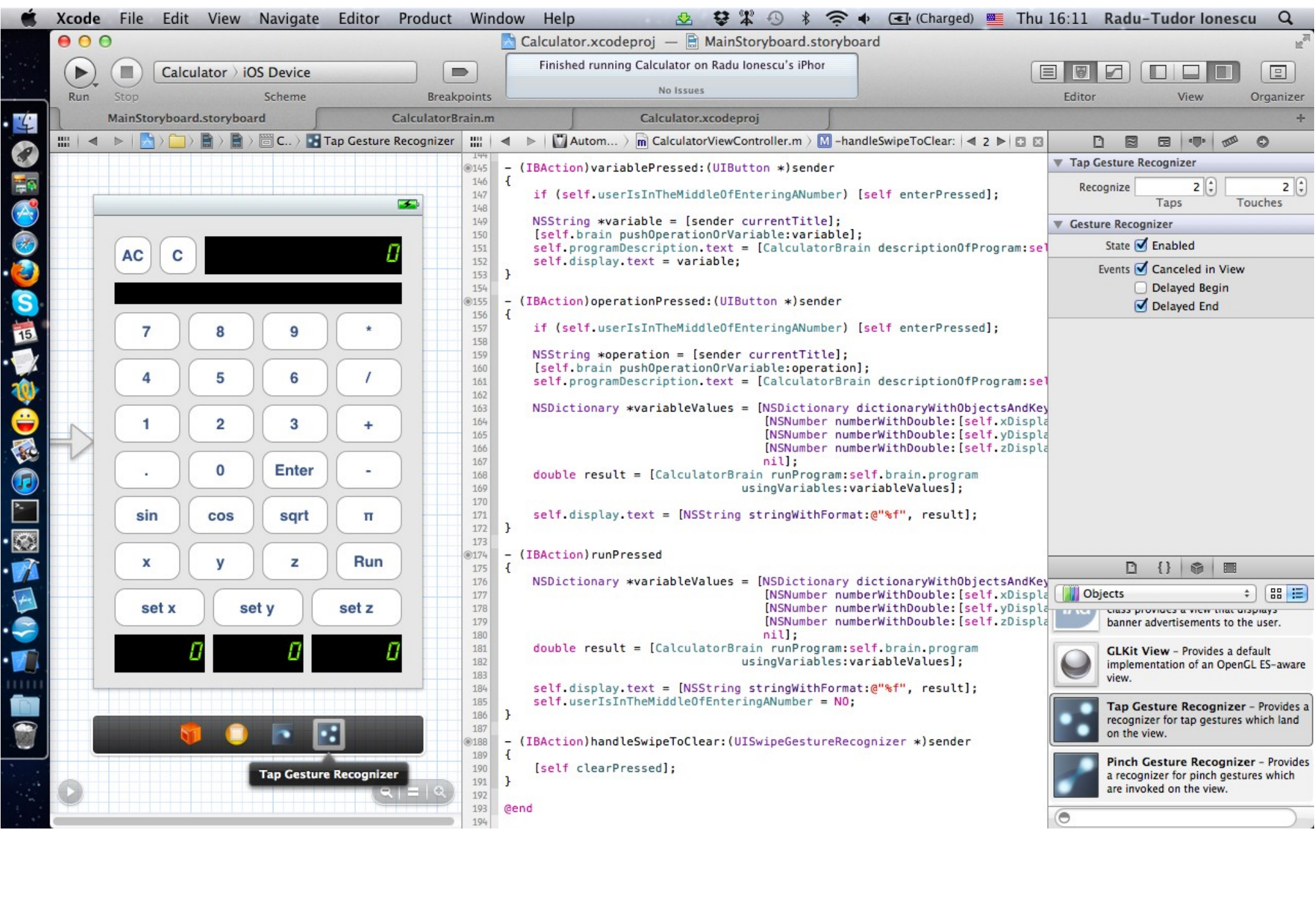
Objects

- class provides a view that displays banner advertisements to the user.
- GLKit View - Provides a default implementation of an OpenGL ES-aware view.
- Tap Gesture Recognizer - Provides a recognizer for tap gestures which land on the view.
- Pinch Gesture Recognizer - Provides a recognizer for pinch gestures which are invoked on the view.

## Task 3

**Task: Add gestures for the most important actions of your Calculator.**

17. In a similar way we are going to add a Tap Gesture Recognizer. Search for a Tap Gesture Recognizer in Object Library and drag it to your View. Again, make sure the entire View is highlighted when you do this.
18. Notice the Tap Gesture Recognizer will be placed right next to the Swipe Gesture Recognizer. Right-click on the Tap Gesture Recognizer and check that the View is a referencing outlet for this gesture recognizer.
19. We are going to assign a double tap gesture with two fingers for Enter. Open Attributes Inspector in Utilities area.
20. Set up the recognizer so that it recognizes a double Tap gesture with 2 Touches. The following slide shows how to set up the Tap Gesture Recognizer.



Storyboard view of a calculator app. The calculator has a display showing '0', buttons for digits 0-9, operations (\*, /, +, -), trigonometric functions (sin, cos, sqrt, pi), and variable keys (x, y, z, set x, set y, set z). A 'Tap Gesture Recognizer' is attached to the bottom of the calculator interface.

```
145 - (IBAction)variablePressed:(UIButton *)sender
146 {
147     if (self.userIsInTheMiddleOfEnteringANumber) [self enterPressed];
148
149     NSString *variable = [sender currentTitle];
150     [self.brain pushOperationOrVariable:variable];
151     self.programDescription.text = [CalculatorBrain descriptionOfProgram:set];
152     self.display.text = variable;
153 }
154
155 - (IBAction)operationPressed:(UIButton *)sender
156 {
157     if (self.userIsInTheMiddleOfEnteringANumber) [self enterPressed];
158
159     NSString *operation = [sender currentTitle];
160     [self.brain pushOperationOrVariable:operation];
161     self.programDescription.text = [CalculatorBrain descriptionOfProgram:set];
162
163     NSDictionary *variableValues = [NSDictionary dictionaryWithObjectsAndKey:
164                                     [NSNumber numberWithInt:[self.xDisplay.text intValue]],
165                                     [NSNumber numberWithInt:[self.yDisplay.text intValue]],
166                                     [NSNumber numberWithInt:[self.zDisplay.text intValue]],
167                                     nil];
168
169     double result = [CalculatorBrain runProgram:self.brain.program
170                    usingVariables:variableValues];
171
172     self.display.text = [NSString stringWithFormat:@"%f", result];
173 }
174
175 - (IBAction)runPressed
176 {
177     NSDictionary *variableValues = [NSDictionary dictionaryWithObjectsAndKey:
178                                     [NSNumber numberWithInt:[self.xDisplay.text intValue]],
179                                     [NSNumber numberWithInt:[self.yDisplay.text intValue]],
180                                     [NSNumber numberWithInt:[self.zDisplay.text intValue]],
181                                     nil];
182
183     double result = [CalculatorBrain runProgram:self.brain.program
184                    usingVariables:variableValues];
185
186     self.display.text = [NSString stringWithFormat:@"%f", result];
187     self.userIsInTheMiddleOfEnteringANumber = NO;
188 }
189
190 - (IBAction)handleSwipeToClear:(UISwipeGestureRecognizer *)sender
191 {
192     [self clearPressed];
193 }
194 @end
```

Tap Gesture Recognizer configuration panel. Recognize: 2 Taps, 2 Touches. State: Enabled. Events: Canceled in View, Delayed End. Objects panel below lists: class provides a view that displays banner advertisements to the user, GLKit View - Provides a default implementation of an OpenGL ES-aware view, Tap Gesture Recognizer - Provides a recognizer for tap gestures which land on the view, Pinch Gesture Recognizer - Provides a recognizer for pinch gestures which are invoked on the view.

## Task 3

**Task: Add gestures for the most important actions of your Calculator.**

21. Let's assign an action to the gesture recognizer. CTRL-drag from the Tap Gesture Recognizer to the Controller's implementation right before `@end`.
22. In the pop-up window that appears after CTRL-dragging set the method's name to "handleTapToEnter".
23. Select `UITapGestureRecognizer` for the argument Type.
24. Click Connect.
25. In the implementation of the `handleTapToEnter:` we are going to simply send the `enterPressed` message on `self`.

The following slide shows how to implement the `handleTapToEnter:` method.

Calculator iPhone 5.1 Simulator

Running Calculator on iPhone 5.1 Simulator No Issues

MainStoryboard.storyboard CalculatorBrain.m Calculator.xcodeproj

Tap Gesture Recognizer

Automatic CalculatorViewController.m -handleTapToEnter:



```

149 NSString *variable = [sender currentTitle];
150 [self.brain pushOperationOrVariable:variable];
151 self.programDescription.text = [CalculatorBrain descriptionOfProgram:self.brain.program];
152 self.display.text = variable;
153 }
154
155 - (IBAction)operationPressed:(UIButton *)sender
156 {
157     if (self.userIsInTheMiddleOfEnteringANumber) [self enterPressed];
158
159     NSString *operation = [sender currentTitle];
160     [self.brain pushOperationOrVariable:operation];
161     self.programDescription.text = [CalculatorBrain descriptionOfProgram:self.brain.program];
162
163     NSDictionary *variableValues = [NSDictionary dictionaryWithObjectsAndKeys:
164                                     [NSNumber numberWithInt:[self.xDisplay.text intValue]],
165                                     [NSNumber numberWithInt:[self.yDisplay.text intValue]],
166                                     [NSNumber numberWithInt:[self.zDisplay.text intValue]],
167                                     nil];
168
169     double result = [CalculatorBrain runProgram:self.brain.program
170                    usingVariables:variableValues];
171
172     self.display.text = [NSString stringWithFormat:@"%f", result];
173 }
174
175 - (IBAction)runPressed
176 {
177     NSDictionary *variableValues = [NSDictionary dictionaryWithObjectsAndKeys:
178                                     [NSNumber numberWithInt:[self.xDisplay.text intValue]],
179                                     [NSNumber numberWithInt:[self.yDisplay.text intValue]],
180                                     [NSNumber numberWithInt:[self.zDisplay.text intValue]],
181                                     nil];
182
183     double result = [CalculatorBrain runProgram:self.brain.program
184                    usingVariables:variableValues];
185
186     self.display.text = [NSString stringWithFormat:@"%f", result];
187     self.userIsInTheMiddleOfEnteringANumber = NO;
188 }
189
190 - (IBAction)handleSwipeToClear:(UISwipeGestureRecognizer *)sender
191 {
192     [self clearPressed];
193 }
194
195 - (IBAction)handleTapToEnter:(UITapGestureRecognizer *)sender
196 {
197     [self enterPressed];
198 }

```

Not Applicable

Objects

- class provides a view that displays banner advertisements to the user.
- GLKit View - Provides a default implementation of an OpenGL ES-aware view.
- Tap Gesture recognizer - Provides a recognizer for tap gestures which land on the view.
- Pinch Gesture Recognizer - Provides a recognizer for pinch gestures which are invoked on the view.

## Task 3

**Task: Add gestures for the most important actions of your Calculator.**

26. Run the application in iOS Simulator. Type in 25 then double tap anywhere on the screen with two fingers (make sure you don't press any button by mistake) to Enter it in the program.

Then type 36 and press the “set x” button.

Swipe from left to right anywhere on the screen to clear the Calculator's display. Then press x and +. You will get the result of  $25 + x = 61$ .

Note that the gestures also work in landscape mode (especially the swipe gesture).

All done! This is the final version of the RPN Calculator app.

# Assignment 3

Assignment: Add another swipe gesture recognizer for the AC button. It should recognize a gesture similar to the one used for the Clear button only that it should be a swipe with two fingers.

Hints: You should drag this gesture recognizer for the Object Library. Don't forget to set the number of Touches tot 2. Add an action for this gesture recognizer and name it `handleSwipeToClearAll:`. Call `allClearPressed` when the gesture is recognized.

# Assignment 4

Assignment: Add a pinch gesture recognizer for the Run button. It should recognize a pinch-close gesture.

Hints: You should drag the Pinch Gesture Recognizer for the Object Library. Add an action for this gesture recognizer and name it `handlePinchToRun:`. Call `runPressed` when the gesture is recognized.



**Congratulations!**