

Developing Applications for iOS



Lab 3: RPN Calculator App (2 of 3)

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

Task 1

Task: Add the following 4 operation buttons: sin (calculates the sine of the top operand on the stack), cos (calculates the cosine of the top operand on the stack), sqrt (calculates the square root of the top operand on the stack), π (calculates the value of π).

1. Launch Xcode and go to “File > Open” and select the Xcode project (.xcodeproj) inside the “Calculator(1of3)” folder.
2. Run the application in iOS Simulator and give it a look. Notice the clear button “C” and the floating point button “.” from assignments 1 and 2 in Lab 2. Type in 1.25 E 3.75 + (where E means Enter). The result should be 5.0. Clear and compute $24 * 26$.
3. Stop running the application.
4. Look for the `clearPressed` and `pointPressed` methods inside `CalculatorViewController.m` to see how they are implemented.

Task 1

Task: Add the following 4 operation buttons: `sin` (calculates the sine of the top operand on the stack), `cos` (calculates the cosine of the top operand on the stack), `sqrt` (calculates the square root of the top operand on the stack), `π` (calculates the value of π).

5. Add four more buttons for the `sin`, `cos`, `sqrt` and `π` operations. Place them under the other buttons and align them to look nice. Note that you should copy and paste an operation button since we want our new operations to send the same `operationPressed: action`.

The next screenshot shows how your View should look like.

6. Switch to the `CalculatorBrain.m` tab.

We will modify the implementation of the `performOperation:` method to also compute these new operations. Note that `sin()`, `cos()` and `sqrt()` are functions in the normal BSD Unix C library. Feel free to use them to calculate sine, cosine and squared root.

Task 1

Task: Add the following 4 operation buttons: `sin` (calculates the sine of the top operand on the stack), `cos` (calculates the cosine of the top operand on the stack), `sqrt` (calculates the square root of the top operand on the stack), `π` (calculates the value of π).

7. Implement `sin`, `cos` and `sqrt` as unary operations.
8. Implement `π` as an operation which takes no arguments off of the operand stack.

Take a look at the next screenshot to check that your code is ok.

9. Run the application and test the new operations. Type in `π E 2 / sin`. The result should be 1.0 (that is $\sin(\pi/2)$). Note that `3 π *` should put three times the value of π into the `display` on your calculator, so should `3 E π *`, so should `π 3 *`. Perhaps unexpectedly, `π E 3 * +` would result in 4 times π being shown. You should understand this is the case because `π` is an operation which takes no arguments off of the operand stack, not a new way of entering an operand into the `display`.

```
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 > M -performOperation:
42 {
43     NSNumber *operandObject = [self.operandStack lastObject];
44     if (operandObject) [self.operandStack removeLastObject];
45     return [operandObject doubleValue];
46 }
47
48 - (double)performOperation:(NSString *)operation
49 {
50     double result = 0;
51
52     if ([operation isEqualToString:@"+"]
53     {
54         result = [self popOperand] + [self popOperand];
55     }
56     else if ([@"*" isEqualToString:operation])
57     {
58         result = [self popOperand] * [self popOperand];
59     }
60     else if ([operation isEqualToString:@"-"]
61     {
62         double subtrahend = [self popOperand];
63         result = [self popOperand] - subtrahend;
64     }
65     else if ([operation isEqualToString:@"/"]
66     {
67         double divisor = [self popOperand];
68         if (divisor) result = [self popOperand] / divisor;
69     }
70     else if ([operation isEqualToString:@"sin"])
71     {
72         result = sin([self popOperand]);
73     }
74     else if ([operation isEqualToString:@"cos"])
75     {
76         result = cos([self popOperand]);
77     }
78     else if ([operation isEqualToString:@"sqrt"])
79     {
80         result = sqrt([self popOperand]);
81     }
82     else if ([operation isEqualToString:@"n"])
83     {
84         result = 3.14159268;
85     }
86
87     [self pushOperand:result];
88     return result;
89 }
90
91 @end
92
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 - (void)emptyOperandStack;
14 - (void)pushOperand:(double)operand;
15 - (double)performOperation:(NSString *)operation;
16
17 @end
18
```

Task 2

Task: Extend the CalculatorBrain with the capacity to run programs.

1. This task requires a major revision of the CalculatorBrain. In order to make our calculator run programs we have to add some public API in the CalculatorBrain. We also have to allow operations to be pushed and popped off the stack. Thus, we should rename the `operandStack` to `programStack`. (don't worry about errors in the CalculatorViewController at this time, we will solve them later).
2. Next, we need to add the public API needed to run programs. In order to run a program we need two things: a copy of the program stack that describes the program and a method that takes a program stack and runs the program.

Let us declare a read-only public `@property` that returns a copy of the `programStack`. Name this property `program`. The type of this property should be `id`, since we want our CalculatorBrain internal implementation of the “program” to be independent of the public API.

The next screenshot shows how to declare the `program` property.

The readonly @property will generate only the getter method. Thus, you will not be able to set this property through a setter method. In general, you can modify a readonly property by accessing it's instance variable from your implementation. By default properties are readwrite, so you don't have to add this keyword explicitly.

```
8 #import "CalculatorBrain.h"
9
10 @interface CalculatorBrain()
11
12 @property (nonatomic, strong) NSMutableArray *programStack;
13
14 @end
15
16 @implementation CalculatorBrain
17
18 @synthesize programStack = _programStack;
19
20 - (NSMutableArray *)programStack
21 {
22     if (_programStack == nil)
23     {
24         _programStack = [[NSMutableArray alloc] init];
25     }
26     return _programStack;
27 }
28
29 - (void)emptyProgramStack
30 {
31     [self.programStack removeAllObjects];
32 }
33
34 - (void)pushOperand:(double)operand
35 {
36     NSNumber *operandObject = [NSNumber numberWithDouble:operand];
37     [self.programStack addObject:operandObject];
38 }
39
40 - (double)popOperand
41 {
42     NSNumber *operandObject = [self.programStack lastObject];
43     if (operandObject) [self.programStack removeLastObject];
44     return [operandObject doubleValue];
45 }
46
47 - (double)performOperation:(NSString *)operation
48 {
49     double result = 0;
50
51     if ([operation isEqualToString:@"+"]
52     {
53         result = [self popOperand] + [self popOperand];
54     }
55     else if ([@"*" isEqualToString:operation])
56     {
57         result = [self popOperand] * [self popOperand];
58     }
59 }
```

```
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 - (void)emptyProgramStack;
16 - (void)pushOperand:(double)operand;
17 - (double)performOperation:(NSString *)operation;
18
19 @end
20
```

Task 2

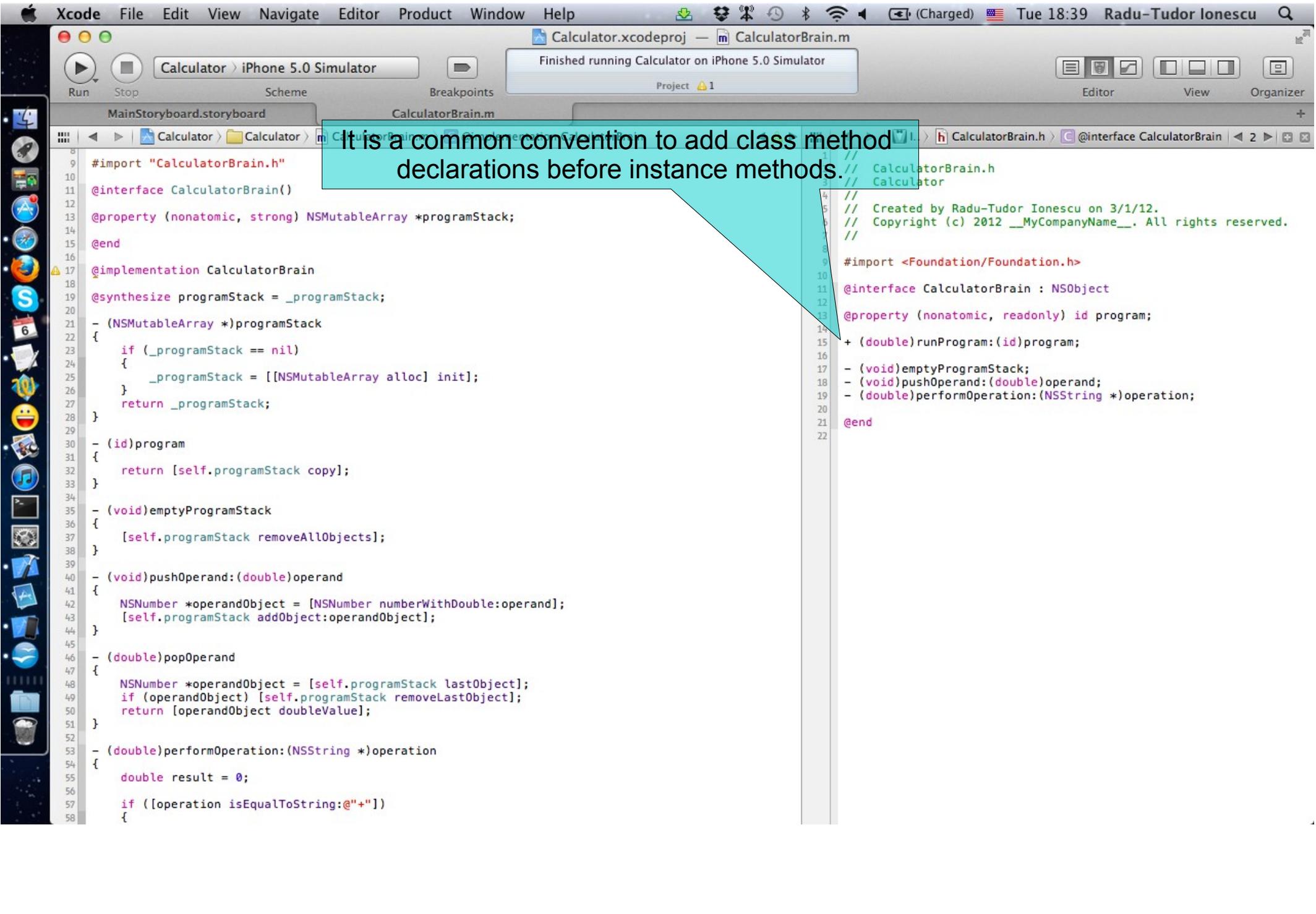
Task: Extend the CalculatorBrain with the capacity to run programs.

3. The value returned by the `program @property` is rather calculated than stored (it's a copy of the `programStack`). Instead of using `@synthesize` to generate the getter, we have to implement it ourself.

Make sure you return **a copy** of the `programStack` and not directly the `programStack`. Add this implementation after the implementation of the `programStack` getter.

4. Now let us add the `runProgram:` method declaration (inside `CalculatorBrain.h`) that runs programs. It should take an argument with general type `id` that is a program. Notice that this method is (and should be) independent of any `CalculatorBrain` instance. This is why we should declare it as a class method.

The following screenshot shows how your code should look by now.



It is a common convention to add class method declarations before instance methods.

```
8 #import "CalculatorBrain.h"
9
10 @interface CalculatorBrain()
11
12 @property (nonatomic, strong) NSMutableArray *programStack;
13
14 @end
15
16 @implementation CalculatorBrain
17
18 @synthesize programStack = _programStack;
19
20 - (NSMutableArray *)programStack
21 {
22     if (_programStack == nil)
23     {
24         _programStack = [[NSMutableArray alloc] init];
25     }
26     return _programStack;
27 }
28
29 - (id)program
30 {
31     return [self.programStack copy];
32 }
33
34 - (void)emptyProgramStack
35 {
36     [self.programStack removeAllObjects];
37 }
38
39 - (void)pushOperand:(double)operand
40 {
41     NSNumber *operandObject = [NSNumber numberWithDouble:operand];
42     [self.programStack addObject:operandObject];
43 }
44
45 - (double)popOperand
46 {
47     NSNumber *operandObject = [self.programStack lastObject];
48     if (operandObject) [self.programStack removeObject];
49     return [operandObject doubleValue];
50 }
51
52 - (double)performOperation:(NSString *)operation
53 {
54     double result = 0;
55
56     if ([operation isEqualToString:@"+"])
```

```
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 + (double)runProgram:(id)program;
16
17 - (void)emptyProgramStack;
18 - (void)pushOperand:(double)operand;
19 - (double)performOperation:(NSString *)operation;
20
21 @end
22
```

Task 2

Task: Extend the `CalculatorBrain` with the capacity to run programs.

5. Implement the `runProgram:` method (inside `CalculatorBrain.m`) so that it runs the program received as argument.

We should test if the `program` is an `NSArray` using introspection then make a `mutableCopy` of it so that we can operate on the stack.

To operate on the stack we can use another recursive function that will be very similar to the current `performOperation:` method, only that it will take the mutable copy of the `program`. Let us name this method `popTermOffProgramStack:` (we will implement it in a second as a class method). Note that we cannot call instance methods from inside class methods.

The following screenshot shows how your code should look by now. Note that sending the `popTermOffProgramStack:` message to the `CalculatorBrain` class generates an error since we haven't implemented it.

The image shows the Xcode IDE interface with two source files open: CalculatorBrain.m and CalculatorBrain.h. The CalculatorBrain.m file contains the implementation of the Calculator class, including the runProgram method which is currently unimplemented. A callout box points to the runProgram method signature in CalculatorBrain.m, stating: "This method is not implemented yet. Note that self means the CalculatorBrain class here."

```
1 //
2 // CalculatorBrain.m
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 "CalculatorBrain.h"
10
11 @interface CalculatorBrain()
12
13 @property (nonatomic, strong) NSMutableArray *programStack;
14
15 @end
16
17 @implementation CalculatorBrain
18
19 @synthesize programStack = _programStack;
20
21 + (double)runProgram:(id)program
22 {
23     NSMutableArray *stack;
24     if ([program isKindOfClass:[NSArray class]])
25     {
26         stack = [program mutableCopy];
27     }
28     return [self popTermOffProgramStack:stack];
29 }
30
31 - (NSMutableArray *)programStack
32 {
33     if (_programStack == nil)
34     {
35         _programStack = [[NSMutableArray alloc] init];
36     }
37     return _programStack;
38 }
39
40 - (id)program
41 {
42     return [self.programStack copy];
43 }
44
45 - (void)emptyProgramStack
46 {
47     [self.programStack removeAllObjects];
48 }
49
50 - (void)pushOperand:(double)operand
51 {
```

```
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 + (double)runProgram:(id)program;
16
17 - (void)emptyProgramStack;
18 - (void)pushOperand:(double)operand;
19 - (double)performOperation:(NSString *)operation;
20
21 @end
22
```

Task 2

Task: Extend the `CalculatorBrain` with the capacity to run programs.

6. First we should declare the `popTermOffProgramStack:` method as a private class method (inside `CalculatorBrain.m`).

The argument of this method is an `NSMutableArray` that is a (or a part of a) program `stack`.

The next screenshot shows how `popTermOffProgramStack:` declaration should look like.

Note that you could directly implement this method (and any “private” method in general) without declaring it in the interface, but you would have to add the implementation before any other method that calls it. Otherwise you will get compiler errors for every call that precedes the method's implementation. To avoid this problem it is better to declare it in the Class Extension.

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

Project 1

MainStoryboard.storyboard CalculatorBrain.m

Calculator > Calculator > CalculatorBrain.m > @implementation CalculatorBrain

```
1 //
2 // CalculatorBrain.m
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 "CalculatorBrain.h"
10
11 @interface CalculatorBrain()
12
13 @property (nonatomic, strong) NSMutableArray *programStack;
14
15 + (double)popTermOffProgramStack:(NSMutableArray *)stack;
16
17 @end
18
19 @implementation CalculatorBrain
20
21 @synthesize programStack = _programStack;
22
23 + (double)runProgram:(id)program
24 {
25     NSMutableArray *stack;
26     if ([program isKindOfClass:[NSArray class]])
27     {
28         stack = [program mutableCopy];
29     }
30     return [self popTermOffProgramStack:stack];
31 }
32
33 - (NSMutableArray *)programStack
34 {
35     if (_programStack == nil)
36     {
37         _programStack = [[NSMutableArray alloc] init];
38     }
39     return _programStack;
40 }
41
42 - (id)program
43 {
44     return [self.programStack copy];
45 }
46
47 - (void)emptyProgramStack
48 {
49     [self.programStack removeAllObjects];
50 }
51 }
```

CalculatorBrain.h

```
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 + (double)runProgram:(id)program;
16
17 - (void)emptyProgramStack;
18 - (void)pushOperand:(double)operand;
19 - (double)performOperation:(NSString *)operation;
20
21 @end
22 }
```

Task 2

Task: Extend the CalculatorBrain with the capacity to run programs.

7. We should implement the `popTermOffProgramStack:` method (inside CalculatorBrain.m) as a class method.

Because this method is very similar to the `performOperation:` instance method we are going to copy and paste the code from this method.

The method should work like this: we pop a term of the stack and we test whether it is an operand (`NSNumber`) or operation (`NSString`) using introspection. For operands we simply return the `doubleValue`, while for operations we do the appropriate action. Instead of calling `popOperand` instance method as in the `performOperation:` method, we should recursively call `popTermOffProgramStack:` to pop terms off the program stack.

The next screenshot shows how `popTermOffProgramStack:` should look like.

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

Project 2

MainStoryboard.storyboard CalculatorBrain.m

Calculator > Calculator > CalculatorBrain.m > +popTermOffProgramStack:

```
31 + (double)popTermOffProgramStack:(NSMutableArray *)stack
32 {
33     double result = 0;
34     id topOfStack = [stack lastObject];
35     if (topOfStack) [stack removeLastObject];
36     if ([topOfStack isKindOfClass:[NSNumber class]])
37     { // We have an operand. Simply return its double value.
38         result = [topOfStack doubleValue];
39     }
40     else if ([topOfStack isKindOfClass:[NSString class]])
41     { // We have an operation. Test the operation type and do the appropriate action.
42         NSString *operation = topOfStack;
43         if ([operation isEqualToString:@"+"]
44             {
45             result = [self popTermOffProgramStack:stack] +
46                 [self popTermOffProgramStack:stack];
47             }
48         else if ([@"*" isEqualToString:operation])
49         {
50             result = [self popTermOffProgramStack:stack] *
51                 [self popTermOffProgramStack:stack];
52         }
53         else if ([operation isEqualToString:@"-"])
54         {
55             double subtrahend = [self popTermOffProgramStack:stack];
56             result = [self popTermOffProgramStack:stack] - subtrahend;
57         }
58         else if ([operation isEqualToString:@"/"])
59         {
60             double divisor = [self popTermOffProgramStack:stack];
61             if (divisor) result = [self popTermOffProgramStack:stack] / divisor;
62         }
63         else if ([operation isEqualToString:@"sin"])
64         {
65             result = sin([self popTermOffProgramStack:stack]);
66         }
67         else if ([operation isEqualToString:@"cos"])
68         {
69             result = cos([self popTermOffProgramStack:stack]);
70         }
71         else if ([operation isEqualToString:@"sqrt"])
72         {
73             result = sqrt([self popTermOffProgramStack:stack]);
74         }
75         else if ([operation isEqualToString:@"n"])
76         {
77             result = 3.14159268;
78         }
79     }
80     return result;
81 }
```

```
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 + (double)runProgram:(id)program;
16
17 - (void)emptyProgramStack;
18 - (void)pushOperand:(double)operand;
19 - (double)performOperation:(NSString *)operation;
20
21 @end
22
```

Task 2

Task: Extend the CalculatorBrain with the capacity to run programs.

8. We should change the `performOperation:` implementation to only push operations on the stack. Thus, we should rename it to `pushOperation:`. The method should return `void` since we don't need to return the result of the current operation anymore. Now we do this by “running” the Calculator's program using `runProgram:`.

To add the operation on the stack we simply call `addObject:` on the `programStack` (the same thing we do when we push operands). Note that operations are `NSString` objects and operands are `NSNumber` objects. We actually used this difference to determine whether we have an operand or an operation using introspection in the `popTermOffProgramStack:` method.

The next screenshot shows how the new `pushOperation:` implementation should look like.

9. We no longer need the `popOperand` method so we can delete it.

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

Run Stop Scheme Breakpoints Editor View Organizer

MainStoryboard.storyboard CalculatorBrain.m

Calculator > Calculator > CalculatorBrain.m > M -pushOperation:

```
66 {
67     result = sin([self popTermOffProgramStack:stack]);
68 }
69 else if ([operation isEqualToString:@"cos"])
70 {
71     result = cos([self popTermOffProgramStack:stack]);
72 }
73 else if ([operation isEqualToString:@"sqrt"])
74 {
75     result = sqrt([self popTermOffProgramStack:stack]);
76 }
77 else if ([operation isEqualToString:@"n"])
78 {
79     result = 3.14159268;
80 }
81 }
82 return result;
83 }
84
85 - (NSMutableArray *)programStack
86 {
87     if (_programStack == nil)
88     {
89         _programStack = [[NSMutableArray alloc] init];
90     }
91     return _programStack;
92 }
93
94 - (id)program
95 {
96     return [self.programStack copy];
97 }
98
99 - (void)emptyProgramStack
100 {
101     [self.programStack removeAllObjects];
102 }
103
104 - (void)pushOperand:(double)operand
105 {
106     NSNumber *operandObject = [NSNumber numberWithDouble:operand];
107     [self.programStack addObject:operandObject];
108 }
109
110 - (void)pushOperation:(NSString *)operation
111 {
112     [self.programStack addObject:operation];
113 }
114
115 @end
116
```

Includes > CalculatorBrain.h > M -pushOperation:

```
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 + (double)runProgram:(id)program;
16
17 - (void)emptyProgramStack;
18 - (void)pushOperand:(double)operand;
19 - (void)pushOperation:(NSString *)operation;
20
21 @end
22
```

Task 2

Task: Extend the CalculatorBrain with the capacity to run programs.

10. Update the CalculatorViewController to use the new methods defined by the CalculatorBrain Model.

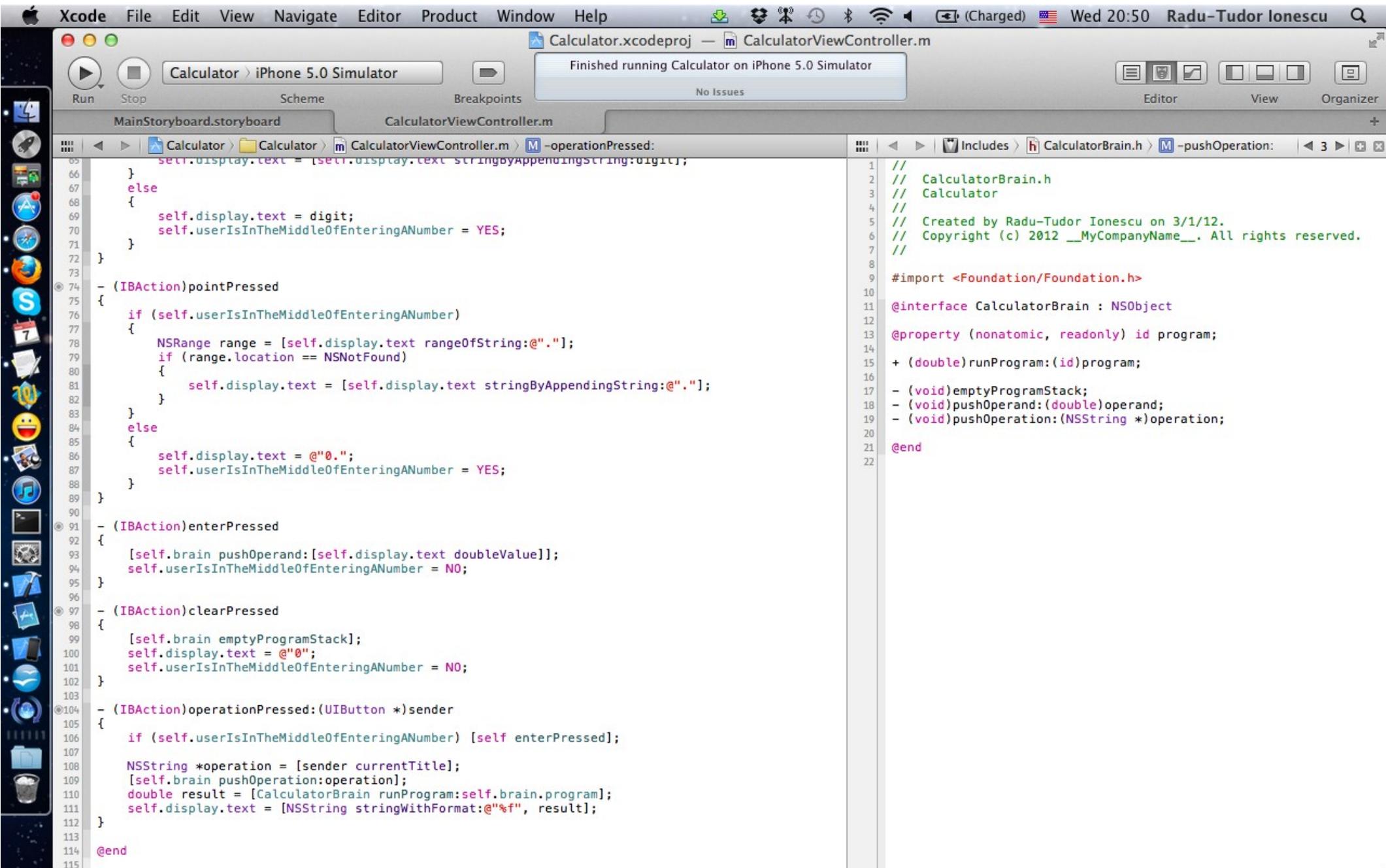
First we need to update the `clearPressed` method that calls `emptyOperandStack` instead of `emptyProgramStack`.

11. Next we should update the `operationPressed:` method that calls `performOperation:` instead of the new `pushOperation:` method.

We also need to get the result of the current operation (since `pushOperation:` doesn't return it anymore). In order to do this we need to use the `runProgram:` class method. Note that we use the class method `class` to get the `Class` of the `self` instance object.

The next screenshot shows how your Controller should look like.

12. Run the application in iOS simulator and check that it works the same as before.



Task 3

Task: Add the capability to your CalculatorBrain to accept variables as operands (in addition to still accepting doubles as operands).

1. Go to the MainStoryboard.storyboard tab in Xcode.
2. Open the Utilities area.
3. You will need new public API in your CalculatorBrain to support this new feature, but let us focus on the variables first.

A variable will be specified as an `NSString` object. It will be pushed on the stack as other `double` operands. The value of the variable will be used when we “run” the program.

To simplify our implementation, we will define only three variables `x`, `y` and `z`. We have to define three buttons to set the value of these variables and three buttons that will let us introduce variables in the “program”.

Add an `UIButton` from the Object Library and place it under the other Calculator's buttons.

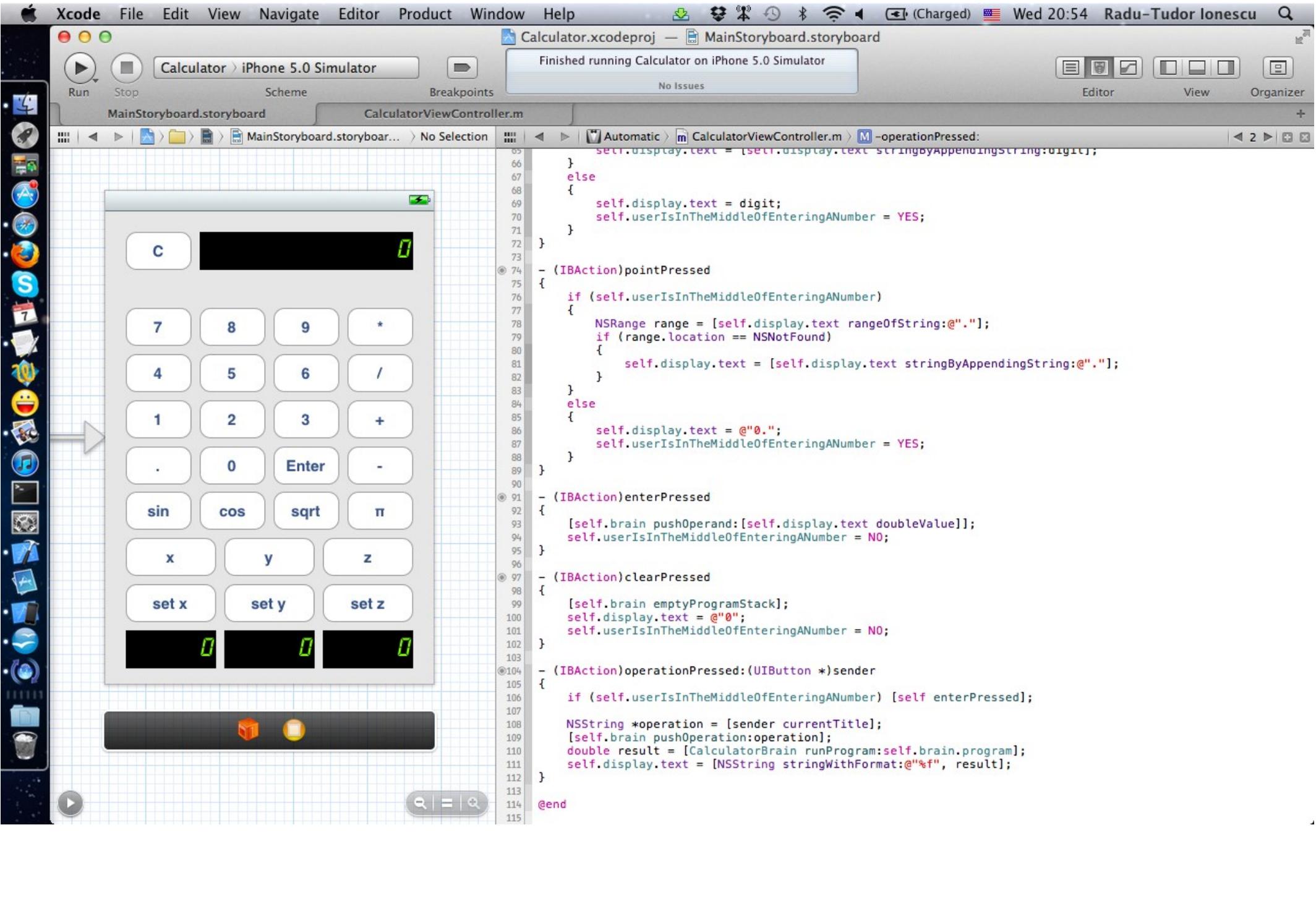
Task 3

Task: Add the capability to your CalculatorBrain to accept variables as operands (in addition to still accepting doubles as operands).

4. Resize it to 88 pixels wide.
5. Copy and paste this button 5 times and place them on two rows (3 buttons per row).
6. Set their title to “x”, “y”, “z”, “set x”, “set y”, “set z”, respectively.
7. We should also use three displays that will show the current value of x, y and z variables, respectively.

The easy way to do this is to copy the `display` label three times and resize the copies to 88 pixels wide.

Your View should now be similar to the one presented in the next screenshot.

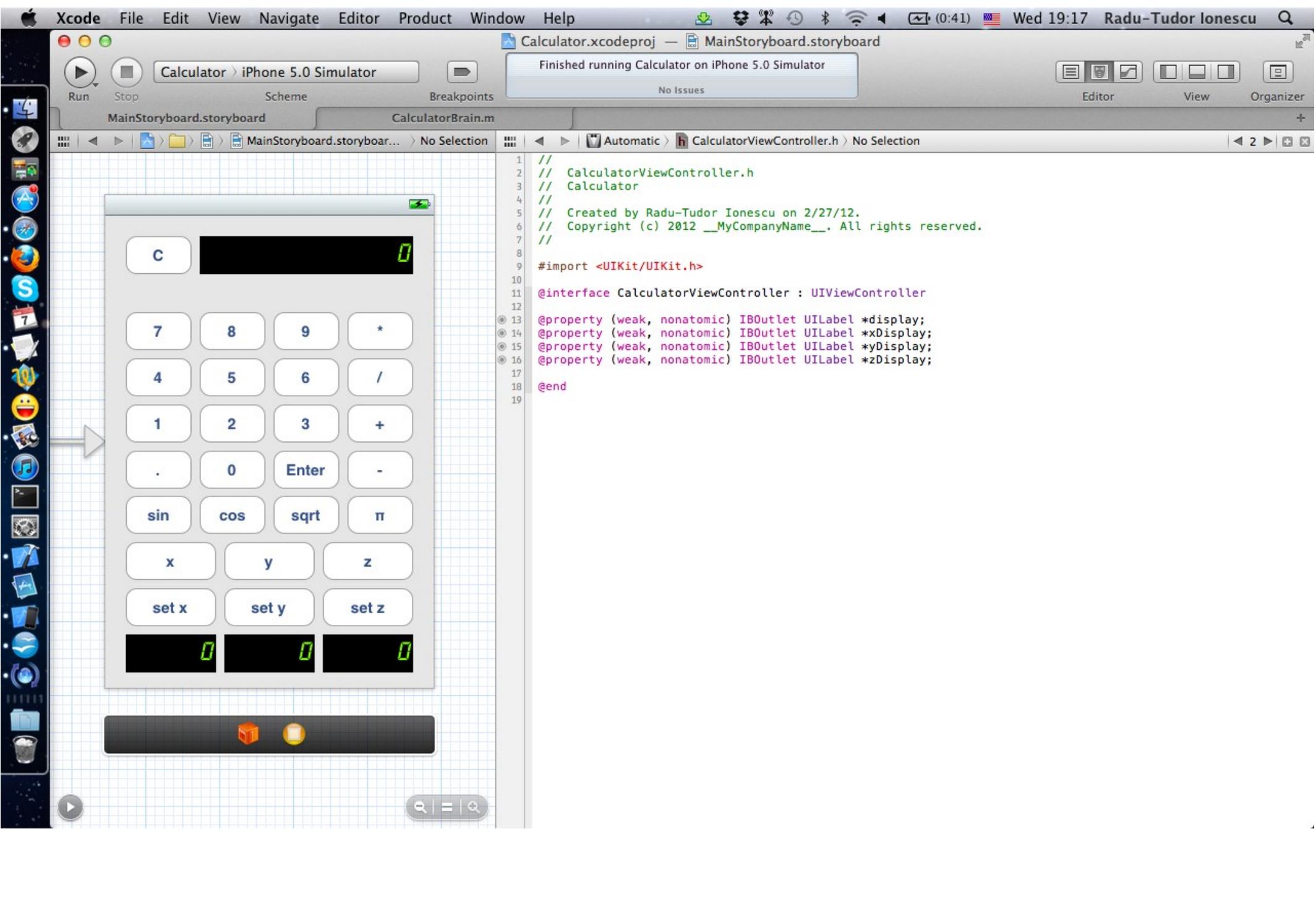


Task 3

Task: Add the capability to your CalculatorBrain to accept variables as operands (in addition to still accepting doubles as operands).

8. Open CalculatorViewController.h in Assistant Editor.
9. CTRL-drag from the first label in your View to you Controller header to add an outlet for that UILabel. Name this outlet “xDisplay”. Make sure you add it as a weak @property.
10. Do the same thing for the other two labels. Name them “yDisplay” and “zDisplay” respectively.

The next screenshot shows how your Controller's header should look like.



Finished running Calculator on iPhone 5.0 Simulator

No Issues



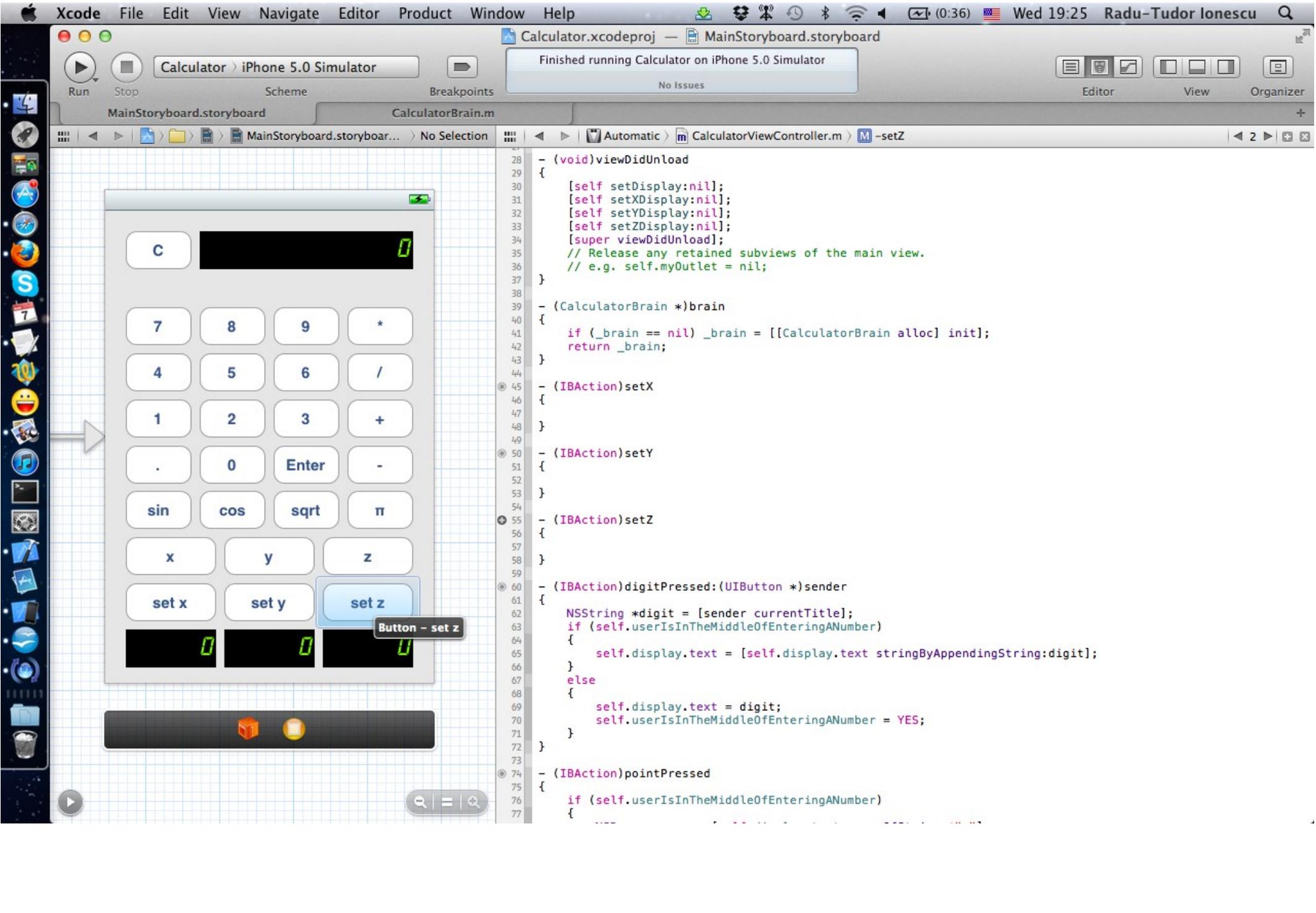
```
1 //
2 // CalculatorViewController.h
3 // Calculator
4 //
5 // Created by Radu-Tudor Ionescu on 2/27/12.
6 // Copyright (c) 2012 __MyCompanyName__. All rights reserved.
7 //
8
9 #import <UIKit/UIKit.h>
10
11 @interface CalculatorViewController : UIViewController
12
13 @property (weak, nonatomic) 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
18 @end
19
```

Task 3

Task: Add the capability to your CalculatorBrain to accept variables as operands (in addition to still accepting doubles as operands).

11. Next we should add the actions for the “set” buttons. Open CalculatorViewController.m in Assistant Editor.
12. CTRL-drag from the “set x” button to your Controller's implementation to associate an action for this button.
13. Name the action “setX” and make sure you select None for Arguments.
14. In a similar way add two more actions for the “set y” and “set z” buttons. Name the actions “setY” and “setZ”, respectively.

The next screenshot gives you a hint of how your Controller's implementation should look like.



```
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 }
48 }
49
50 - (IBAction)setY
51 {
52 }
53 }
54
55 - (IBAction)setZ
56 {
57 }
58 }
59
60 - (IBAction)digitPressed:(UIButton *)sender
61 {
62     NSString *digit = [sender currentTitle];
63     if (self.userIsInTheMiddleOfEnteringANumber)
64     {
65         self.display.text = [self.display.text stringByAppendingString:digit];
66     }
67     else
68     {
69         self.display.text = digit;
70         self.userIsInTheMiddleOfEnteringANumber = YES;
71     }
72 }
73 }
74 - (IBAction)pointPressed
75 {
76     if (self.userIsInTheMiddleOfEnteringANumber)
77     {
```

Task 3

Task: Add the capability to your CalculatorBrain to accept variables as operands (in addition to still accepting doubles as operands).

15. To set the x variable we simply copy the current `display` text and set it to the `xDisplay` text. Make sure you use dot notation for the `xDisplay` setter.

Also do **NOT** try to copy the `display` label itself and assign it to `xDisplay`. Doing so will replace the `xDisplay` outlet with the `display` outlet. Thus, we will have no way to reach the label for the x variable. The point is we only need the text inside the `display` `UILabel`.

16. Implement the `setY` and `setZ` methods in a similar way.

The next screenshot shows how the `setX`, `setY` and `setZ` methods should be implemented.

Xcode File Edit View Navigate Editor Product Window Help

Calculator.xcodeproj — MainStoryboard.storyboard

Running Calculator on iPhone 5.0 Simulator

No Issues

Editor View Organizer

MainStoryboard.storyboard CalculatorBrain.m

MainStoryboard.storyboard... > No Selection Automatic > CalculatorViewController.m > M -setZ



```
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
50 - (IBAction)setY
51 {
52     self.yDisplay.text = self.display.text;
53 }
54
55 - (IBAction)setZ
56 {
57     self.zDisplay.text = self.display.text;
58 }
59
60 - (IBAction)digitPressed:(UIButton *)sender
```

Calculator

Task 3

Task: Add the capability to your CalculatorBrain to accept variables as operands (in addition to still accepting doubles as operands).

17. Next we should add the actions for the “x”, “y” and “z” buttons.

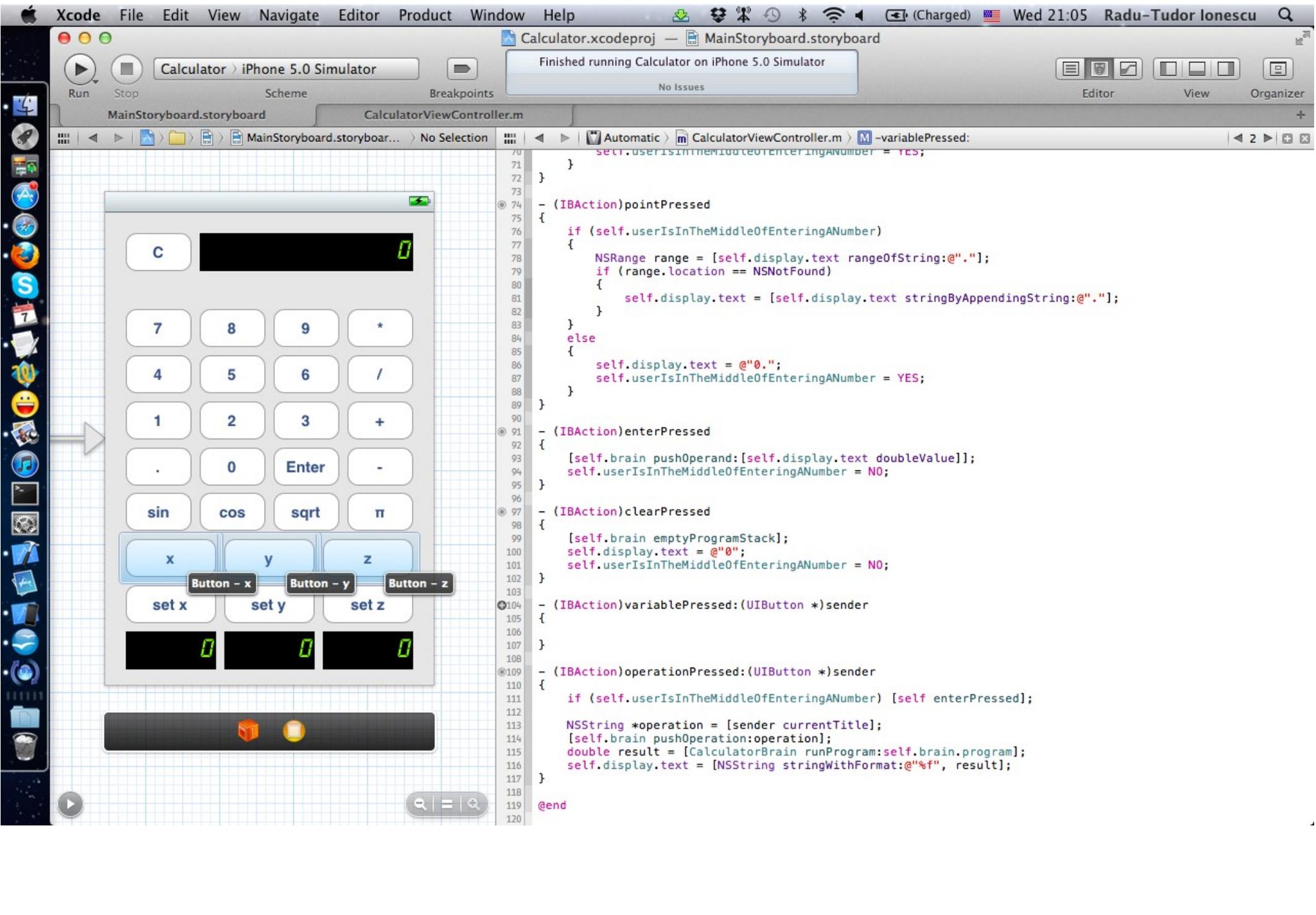
CTRL-drag from the “x” button to your Controller's implementation to associate an action for this button. Place this method before the `operationPressed:` method.

18. Name the action “variablePressed” and make sure you select Sender for Arguments.

19. Using CTRL-drag from the “y” and “z” buttons you can associate the same `variablePressed:` action. This works only if you save the Controller first (use CMD + S to do this).

20. Change the sender type to be `UIButton *` instead of `id`.

The next screenshot gives you a hint about how your Controller's implementation should look like by now.



Task 3

Task: Add the capability to your CalculatorBrain to accept variables as operands (in addition to still accepting doubles as operands).

21. Next we should implement the `variablePressed: action`. It works pretty much the same as the `operationPressed: method`.

If the user is in the middle of entering a number we should press ENTER (by calling `enterPressed`) before adding the variable on the stack.

We need to get the variable name by looking at the sender's `currentTitle`. After this, we can push the variable on the stack using the `pushVariable: method` of the brain. We are going to add this method to our Model in a minute.

Then we have to put the variable name in the `display`.

See how to implement this method on the next screenshot.

Xcode File Edit View Navigate Editor Product Window Help

Calculator.xcodeproj — MainStoryboard.storyboard

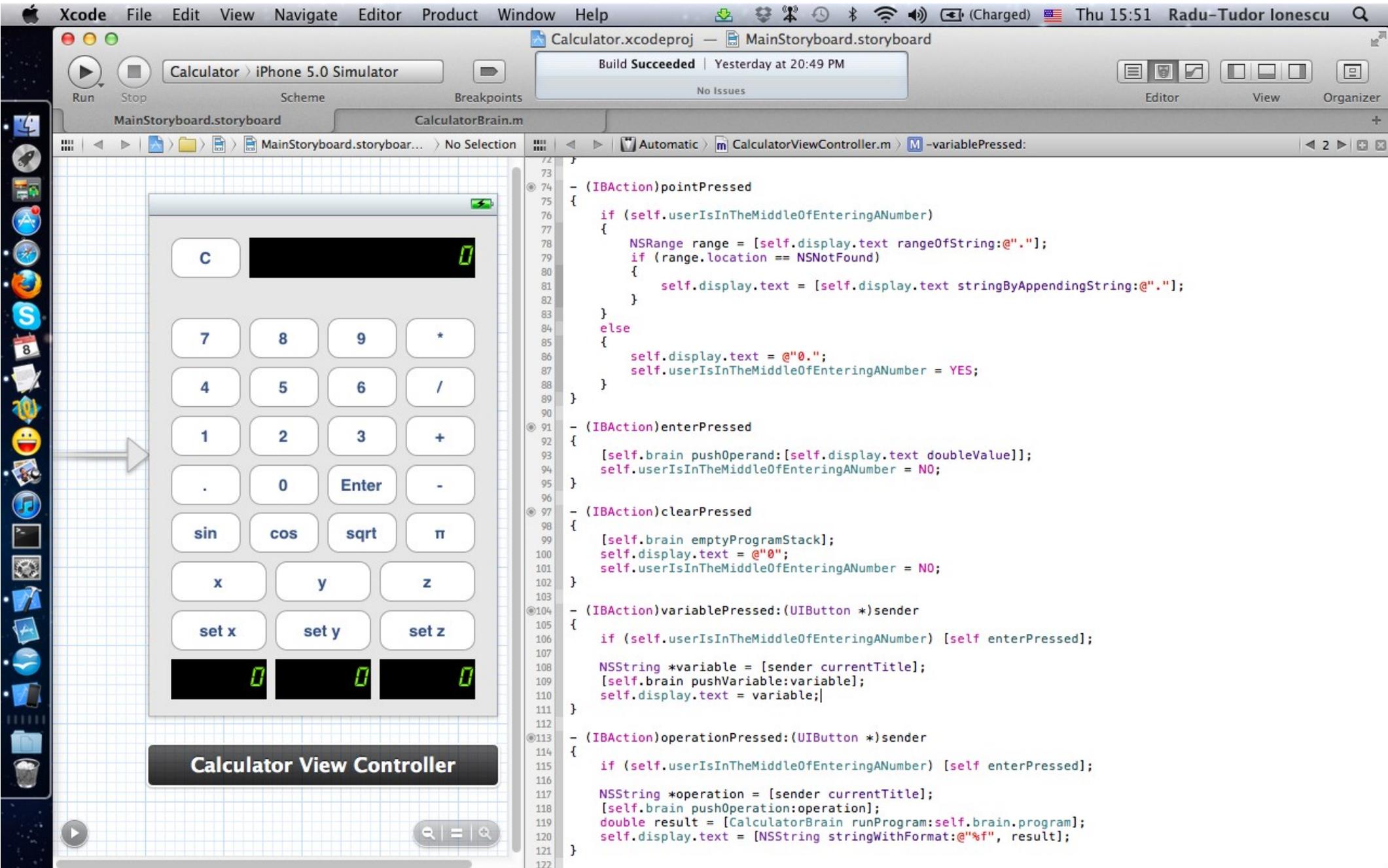
Build Succeeded | Yesterday at 20:49 PM
No Issues

Calculator > iPhone 5.0 Simulator

Run Stop Scheme Breakpoints Editor View Organizer

MainStoryboard.storyboard CalculatorBrain.m

MainStoryboard.storyboard... > No Selection Automatic > CalculatorViewController.m > M -variablePressed:



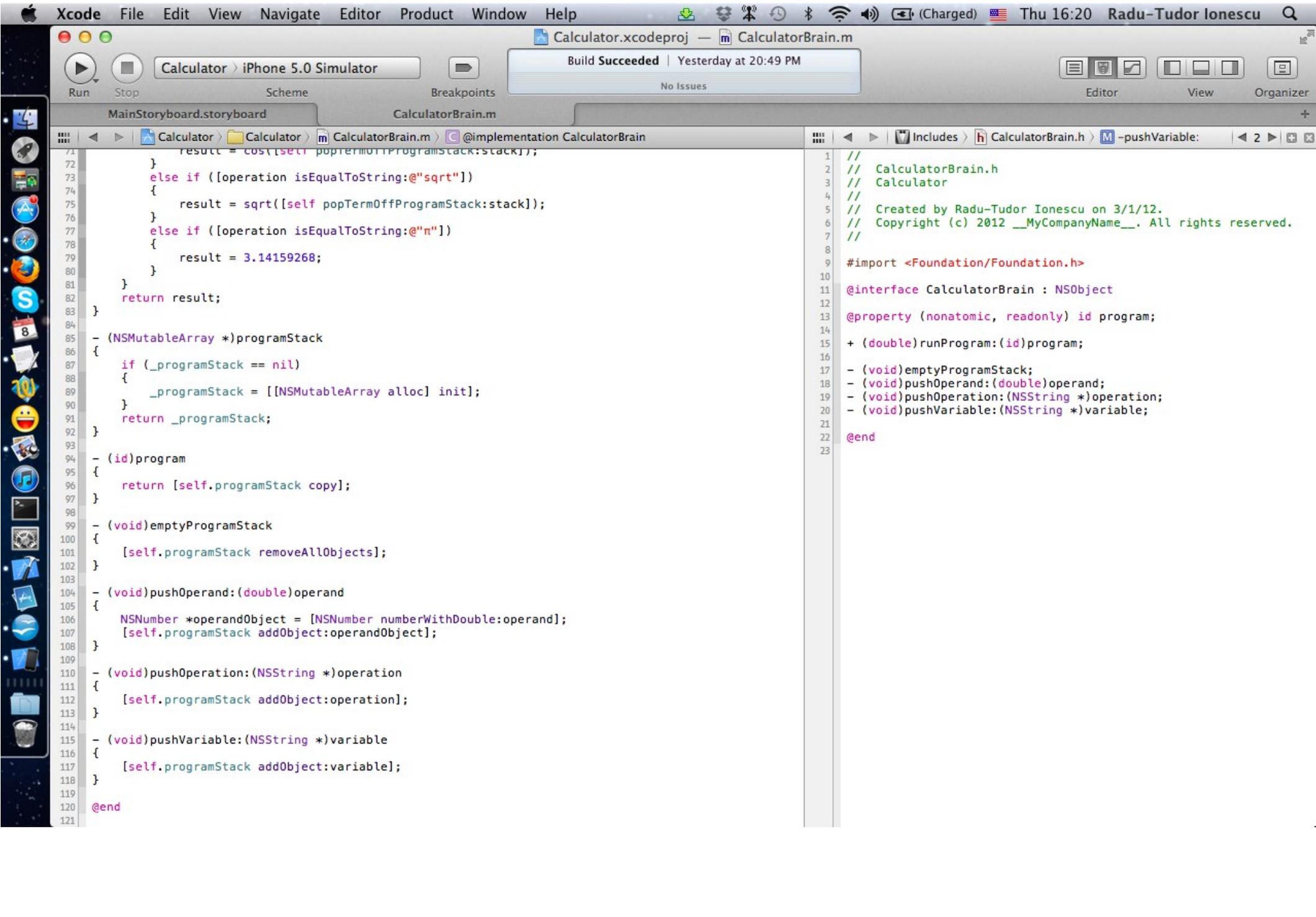
```
72 }
73
74 - (IBAction)pointPressed
75 {
76     if (self.userIsInTheMiddleOfEnteringANumber)
77     {
78         NSRange range = [self.display.text rangeOfString:@"."];
79         if (range.location == NSNotFound)
80         {
81             self.display.text = [self.display.text stringByAppendingString:@"."];
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)clearPressed
98 {
99     [self.brain emptyProgramStack];
100    self.display.text = @"0";
101    self.userIsInTheMiddleOfEnteringANumber = NO;
102 }
103
104 - (IBAction)variablePressed:(UIButton *)sender
105 {
106     if (self.userIsInTheMiddleOfEnteringANumber) [self enterPressed];
107
108     NSString *variable = [sender currentTitle];
109     [self.brain pushVariable:variable];
110     self.display.text = variable;
111 }
112
113 - (IBAction)operationPressed:(UIButton *)sender
114 {
115     if (self.userIsInTheMiddleOfEnteringANumber) [self enterPressed];
116
117     NSString *operation = [sender currentTitle];
118     [self.brain pushOperation:operation];
119     double result = [CalculatorBrain runProgram:self.brain.program];
120     self.display.text = [NSString stringWithFormat:@"%f", result];
121 }
122 }
```

Task 3

Task: Add the capability to your CalculatorBrain to accept variables as operands (in addition to still accepting doubles as operands).

22. Open the CalculatorBrain.m tab in Xcode and let's add the `pushVariable:` method.
23. First we should declare it in the public API of the CalculatorBrain.
24. The implementation of the method is identical to the `pushOperation:` implementation. We simply add the variable to the `programStack` with the `addObject:` method.

See how to declare and implement this method on the next screenshot.



Task 3

Task: Add the capability to your CalculatorBrain to accept variables as operands (in addition to still accepting doubles as operands).

25. Now we can add variable to the program, but we have to find a way to supply the values of the variables when the “program” is “run.” We must add a new version of the `runProgram:` class method with the following signature:

```
+ (double)runProgram:(id)program  
    usingVariables:(NSDictionary *)variableValues;
```

The keys in the passed `variableValues` dictionary are `NSString` objects corresponding to the names of variables, and the values in the dictionary are `NSNumber` objects specifying the value of the corresponding variable.

See how to declare this method on the next screenshot.

Xcode File Edit View Navigate Editor Product Window Help

Calculator.xcodeproj — CalculatorBrain.m

Build Succeeded | Yesterday at 20:49 PM

Project 1

Calculator > iPhone 5.0 Simulator

Run Stop Scheme Breakpoints

MainStoryboard.storyboard CalculatorBrain.m

Calculator > Calculator > CalculatorBrain.m > @implementation CalculatorBrain

```
1 //
2 // CalculatorBrain.m
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 "CalculatorBrain.h"
10
11 @interface CalculatorBrain()
12
13 @property (nonatomic, strong) NSMutableArray *programStack;
14
15 + (double)popTermOffProgramStack:(NSMutableArray *)stack;
16
17 @end
18
19 @implementation CalculatorBrain
20
21 @synthesize programStack = _programStack;
22
23 + (double)runProgram:(id)program
24 {
25     NSMutableArray *stack;
26     if ([program isKindOfClass:[NSArray class]])
27     {
28         stack = [program mutableCopy];
29     }
30     return [self popTermOffProgramStack:stack];
31 }
32
33 + (double)popTermOffProgramStack:(NSMutableArray *)stack
34 {
35     double result = 0;
36     id topOfStack = [stack lastObject];
37     if (topOfStack) [stack removeLastObject];
38     if ([topOfStack isKindOfClass:[NSNumber class]])
39     { // We have an operand. Simply return its double value.
40         result = [topOfStack doubleValue];
41     }
42     else if ([topOfStack isKindOfClass:[NSString class]])
43     { // We have an operation. Test the operation type and do the appropriate action.
44         NSString *operation = topOfStack;
45         if ([operation isEqualToString:@"+"]
46             {
47             result = [self popTermOffProgramStack:stack] +
48                 [self popTermOffProgramStack:stack];
49         }
50         else if ([@"*" isEqualToString:operation])
51         {
```

CalculatorBrain.h

```
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 + (double)runProgram:(id)program;
16 + (double)runProgram:(id)program
17     usingVariables:(NSDictionary *)variableValues;
18
19 - (void)emptyProgramStack;
20 - (void)pushOperand:(double)operand;
21 - (void)pushOperation:(NSString *)operation;
22 - (void)pushVariable:(NSString *)variable;
23
24 @end
25
```

Task 3

Task: Add the capability to your CalculatorBrain to accept variables as operands (in addition to still accepting doubles as operands).

26. It is time to add a private helper class method `isOperation:` that is useful in distinguishing between variables and operations in our implementation of `runProgram:usingVariableValues:`.

Declare this method in the private Class Extension.

27. To implement the `isOperation:` method we will use the `@"+,-,/,*,sin,cos,sqrt,` "constant string (that enumerates all operations) to search for the given `operation`. The method will return `NO` if the location of the `operation` in the constant string is `NSNotFound` and `YES` otherwise.

See how to this method should look like on the next screenshot.

Xcode File Edit View Navigate Editor Product Window Help

Calculator.xcodeproj — CalculatorBrain.m

Calculator > iPhone 5.0 Simulator

Build Succeeded | Yesterday at 20:49 PM

No Issues

MainStoryboard.storyboard CalculatorBrain.m

Calculator > Calculator > CalculatorBrain.m > @implementation CalculatorBrain

Includes > CalculatorBrain.h > No Selection

```
1 //
2 // CalculatorBrain.m
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 "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 + (double)runProgram:(id)program
25 {
26     NSMutableArray *stack;
27     if ([program isKindOfClass:[NSArray class]])
28     {
29         stack = [program mutableCopy];
30     }
31     return [self popTermOffProgramStack:stack];
32 }
33
34 + (BOOL)isOperation:(NSString *)operation
35 {
36     NSRange operationRange = [@"+,-,/,*,sin,cos,sqrt,n" rangeOfString:operation];
37     if (operationRange.location == NSNotFound) return NO;
38     return YES;
39 }
40
41 + (double)popTermOffProgramStack:(NSMutableArray *)stack
42 {
43     double result = 0;
44     id topOfStack = [stack lastObject];
45     if (topOfStack) [stack removeLastObject];
46     if ([topOfStack isKindOfClass:[NSNumber class]])
47     { // We have an operand. Simply return its double value.
48         result = [topOfStack doubleValue];
49     }
50     else if ([topOfStack isKindOfClass:[NSString class]])
51     { // We have an operation. Test the operation type and do the appropriate action.
```

Task 3

Task: Add the capability to your CalculatorBrain to accept variables as operands (in addition to still accepting doubles as operands).

28. We can implement `runProgram:usingVariableValues:` without modifying the method `popOperandOffStack:` at all.

We have to iterate through the `programStack` objects and detect the variables (that are `NSString` objects) using introspection. Because operations are also `NSStrings`, we have to check that `isOperation:` method returns `NO` before we can replace the variable name in the `programStack` with the corresponding values from the `variableValues` dictionary.

In this case it is very useful to use the `NSMutableArray` method `replaceObjectAtIndex:withObject:`. Note that you cannot call this method inside a `for-in` type of enumeration (since you don't have the index in that case): you would need a `for` loop that is iterating by index through the array.

See how to implement this method on 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 > +runProgram:usingVariables:

```
20 @implementation CalculatorBrain
21
22 @synthesize programStack = _programStack;
23
24 + (double)runProgram:(id)program
25 {
26     NSMutableArray *stack;
27     if ([program isKindOfClass:[NSArray class]])
28     {
29         stack = [program mutableCopy];
30     }
31     return [self popTermOffProgramStack:stack];
32 }
33
34 + (double)runProgram:(id)program
35     usingVariables:(NSDictionary *)variableValues
36 {
37     NSMutableArray *stack;
38     if ([program isKindOfClass:[NSArray class]])
39     {
40         stack = [program mutableCopy];
41     }
42
43     for (int i = 0; i < stack.count; i++)
44     {
45         id stackObject = [stack objectAtIndex:i];
46         if ([stackObject isKindOfClass:[NSString class]])
47         {
48             NSString *operationOrVariable = stackObject;
49             if ([self isOperation:operationOrVariable] == NO)
50             {
51                 NSNumber *variableValue = [variableValues objectForKey:operationOrVariable];
52                 if (variableValue == nil) variableValue = [NSNumber numberWithInt:0];
53                 [stack replaceObjectAtIndex:i withObject:variableValue];
54             }
55         }
56     }
57     return [self popTermOffProgramStack:stack];
58 }
59
60 + (BOOL)isOperation:(NSString *)operation
61 {
62     NSRange operationRange = [@"+,-,/,*,sin,cos,sqrt,n" rangeOfString:operation];
63     if (operationRange.location == NSNotFound) return NO;
64     return YES;
65 }
66
67 + (double)popTermOffProgramStack:(NSMutableArray *)stack
68 {
69     double result = 0;
```

Includes > CalculatorBrain.h > No Selection

```
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 + (double)runProgram:(id)program;
16 + (double)runProgram:(id)program
17     usingVariables:(NSDictionary *)variableValues;
18
19 - (void)emptyProgramStack;
20 - (void)pushOperand:(double)operand;
21 - (void)pushOperation:(NSString *)operation;
22 - (void)pushVariable:(NSString *)variable;
23
24 @end
25
```

Task 3

Task: Add the capability to your CalculatorBrain to accept variables as operands (in addition to still accepting doubles as operands).

29. We can now use the new method to run programs that takes an `NSDictionary` of variables with associated values.

Return to the `MainStoryboard.storyboard` tab in Xcode to change the `operationPressed:` method.

30. We have to build a dictionary with the variable names (as keys) and associated values. The `NSDictionary` class method `dictionaryWithObjectsAndKeys:` is great for this task. It takes a `nil`-terminated list of objects and keys separated by comma.

31. We can send the `runProgram:usingVariableValues:` message to the `CalculatorBrain` class and pass the previously created dictionary for the second argument.

The next screenshot shows how the new `operationPressed:` implementation should look like.



Calculator View Controller

```
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)clearPressed
98 {
99     [self.brain emptyProgramStack];
100    self.display.text = @"0.";
101    self.userIsInTheMiddleOfEnteringANumber = NO;
102 }
103 }
104 - (IBAction)variablePressed:(UIButton *)sender
105 {
106     if (self.userIsInTheMiddleOfEnteringANumber) [self enterPressed];
107
108     NSString *variable = [sender currentTitle];
109     [self.brain pushVariable:variable];
110     self.display.text = variable;
111 }
112 }
113 - (IBAction)operationPressed:(UIButton *)sender
114 {
115     if (self.userIsInTheMiddleOfEnteringANumber) [self enterPressed];
116
117     NSString *operation = [sender currentTitle];
118     [self.brain pushOperation:operation];
119
120     NSDictionary *variableValues = [NSDictionary dictionaryWithObjectsAndKeys:
121                                     [NSNumber numberWithInt:[xDisplay.text doubleValue]],@"x",
122                                     [NSNumber numberWithInt:[yDisplay.text doubleValue]],@"y",
123                                     [NSNumber numberWithInt:[zDisplay.text doubleValue]],@"z",
124                                     nil];
125
126     double result = [CalculatorBrain runProgram:self.brain.program
127                     usingVariables:variableValues];
128
129     self.display.text = [NSString stringWithFormat:@"%f", result];
130 }
131 }
132 @end
```

Task 3

Task: Add the capability to your CalculatorBrain to accept variables as operands (in addition to still accepting doubles as operands).

32. Run the application in iOS Simulator and use the new features. Press 36 then “set x”. Press C 25 then “set y”. Press C x y +. The calculator should display 61 (that is $36 + 25$). Notice we don't have to press Enter after we add a variable.

It's great! 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. Of course, we could set x and y to the new values and then retype the “x y +” program. This would work, but it's not right!

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 in the next lab.

Assignment 1

Assignment: Adjust the `CalculatorBrain runProgram:` class method so that it runs programs with variables. Each variable should be replaced with zero for this method. Test this method by placing it in the `operationPressed:` method instead of the method that “runs” “programs” using the actual variable values.

Hints: There are two options for this assignment. You can either call the `runProgram:usingVariables:` and pass `nil` instead of a dictionary with variable values, or you can iterate through the stack objects and replace all variables with zero.

You would have to use `[NSNumber numberWithInt:0]` in the latter case.

Assignment 2

Assignment: Note that the `pushOperation:` and `pushVariable:` methods in `CalculatorBrain` do the same thing. You should replace them with a single method called `pushOperationOrVariable:`.

Hint: You would also have to change the `variablePressed:` and `operationPressed:` to push things on the stack using the new `pushOperationOrVariable:` method.

Congratulations!