**Homework 6 Instructions and Notes**

**Homework Review**

**Homework 4 – write code for the remaining Homework 3 sample tests – (rental car)**

**(Homework 4 Challenge – write a set of tests that define the behavior of Calculator)**

**Homework 5 – choose a set of tests that define our calculator (not done)**

**Homework 6 – write code in the CalcController to create the behavior described in the set of tests**

# *Assignment:*

*In this assignment, we will be implementing the calculator by writing code in CalculatorController.  Ideally, once we’ve made all of the tests for CalculatorController run green, the calculator itself (via the UI) should behave correctly.*

# Writing CalculatorController:

**WARNING**: This assignment is a lot harder than previous assignments.  Specifically, this assignment is the first assignment where the code you must write interacts to a great extent with the internal state of an object.

The object in question is your CalculatorController.  How you choose to represent the “internal state” of the calculator controller is up to you -- but it needs to include the number the user is entering at the moment, whatever number or prior result they may have already entered, and whatever operator they may have entered between them.

*Conversion Operations: For the purposes of this assignment, please use the following code to convert between types:*

*// To convert a char “input” to a string:* ***input.ToString()***

*// To convert a char “input” to a decimal:* ***Decimal.Parse(input.ToString())***

*// To convert an integer “input” to a char:* ***input.ToString()[0]***

*// To convert a single-character string “input” to a char:* ***input[0]***

*// To convert a string “input” to a decimal:* ***Decimal.Parse(input)***

# *Writing CalculatorController:*

***WARNING****: This assignment is a lot harder than previous assignments.  Specifically, this assignment is the first assignment where the* ***code you must write interacts to a great extent with the internal state of an object.***

*The object in question is your CalculatorController.  How you choose to* ***represent the “internal state” of the calculator controller*** *is up to you -- but it needs to include the* ***number the user is entering at the moment, whatever number or prior result they may have already entered, and whatever operator they may have entered between them.***

***Example:*** *If the user enters “3”, “+”, and “4”, the CalculatorController needs to be in a different state than if the user enters “3”, “-”, and “4”.  In both cases, the calculator’s display (the result of the GetOutput() method) should show “4”.  But in the first case, if the user enters “=” as the next input, the calculator should display “7”, and in the second case, if the user enters “=” as the next input, the calculator should display “-1”.*

***The calculator has “internal state”.  It “remembers” what values, and what operators, were previously entered.*** *[AEH: I understand the concept of “state”. But iis it something that “happens” because of C#, or do I need to write code to create states? It sounds like I need to add variables until it keeps track of the previous value entered by the user and/or the result of the last calculation (last =? Last sub-computation?)] That internal state is not generally represented by the external state of the calculator.  In both of the cases above, the calculator displays “4”, and if the user enters a “2” as the next character, the calculator will display “42”.  But in the first case, it will internally be prepared to calculate “3 + 42” when the user enters “=”, and in the second case, it will internally be prepared to calculate “3 - 42” when the user enters “=”.*

*The calculator’s internal state must include the “previous value” (the previous value entered by the user, or the result of the most recent calculation).  It also must include the most recently entered operation (+, -, \*, or /).  And, it needs to include the number the user is entering, or has most recently entered.*

***Example:*** *If the user enters “3 + 4 =”, the calculator should display “7”.  But it still needs to remember that the current operation is “plus”, and that the right-hand operand of that operation was “4”...because if the user enters “=” again, the calculator should display “11”.  Try it in WinCalc if you don’t believe me.  :-)*

*This is a lot of very confusing behavior.  That’s why, rather than expressing it in a written “functional specification” document, we have expressed it as a very large number of automated tests.  As you write CalculatorController, you will see some tests pass and other tests suddenly fail again.  You’ll know your work is done when all of the tests pass...and you should have reasonable confidence that your calculator will work correctly when they do.*

**--------------------------------------------**

**Homework 4 Challenge Mode - Tests**

**How to Create Tests to Define Calc Behavior**

Ah -- the "gold standard" for the behavior of your calculator is the "calc" program (the built-in calculator program that comes with Windows). If you're not sure how "a calculator" should behave, just run "calc" and see what its behavior is when you put in various combinations of inputs.

For myself, I tend to think of the simplest cases first. For example, I'll write the tests for addition before I write those for subtraction, multiplication, and division -- and for single-digit addition before multi-digit addition. So I'd start by listing the things I want my calculator to be able to do, in simplest-first order:

"Enter a single-digit number": "3" -> "3"

"Enter a multi-digit number": "3", "7" -> "37"

"Add two single-digit numbers that add up to a single-digit number": "1", "+", "3", "=" -> "4"

"Add two single-digit numbers that add up to a multi-digit number": "9+8=" -> "17"

"Add two multi-digit numbers": "13+26=" -> "39"

"Add three numbers": "1+3+4=" -> "8"

"Add three multi-digit numbers": "12+43+17="72"

And so on.

Once I've handled all of the "happy path" cases I can think of, I go back and think of the "weird" cases. E.g.

"Division by zero should give an error": "7/0=" -> "Cannot divide by zero"

(I had to run "calc" and see what it did in that case, to be sure how to handle that)

"Entering multiple operators in a row just uses the latest one entered": "2+\*3=" -> "6"

(again, I ran "calc" to see how it handled this case)

And that's basically how I plan to come up with the test cases for CalculatorController.

Common Errors:

There were a number of errors that showed up across multiple suggested tests.  Rather than address them individually in the aggregation process, I just fixed them where they showed up, and will enumerate them here so you know to be careful of them.

1. Forgetting to include the "=" after mathematical operations.  The calculator does not display the result of the calculation until you hit "=", because it can't know that you are done entering the number until you hit "=" or some other operator.
2. Expecting the “-” operator to enter a negative number.  Many tests required a behavior relative to negative numbers that does not match the WinCalc implementation.  3--2 (three minus negative two) should, mathematically, be “5”.  WinCalc, however, ignores all but the last operator in a series of operators, and interprets that equation as “3-2”, or “1”.  Similarly, 5\*-3 is “2” (five minus 3), not “-15” (5 times negative 3).
3. Iterating through the characters from 0 to 9, adding them to the controller and asserting that the output after entering each character was that character.  That’s not how WinCalc behaves.  If you enter '0', the output will be "0".  If you then enter '1', the output will be "1"  But if you then enter '2', the output will be "12", not 2".
4. Converting a number (e.g. 3) to a character (e.g. '3') by casting it to type "char".  This does not actually work.  The character whose numeric value is 3 is not '3' -- the numeric value of the character '3' is actually 51 (the ASCII value for '3').  Google "ASCII" for more details.  To convert from the integer 3 to the character '3', you need to use one of the following approaches:
   1. intVal.ToString()[0]   // Convert the integer to a string, then get the string’s 1st character
   2. (char)((int)'0' + intVal)   // Convert the character '0' to an integer, add the digit value to be converted, and convert it back to a string