CIS 3309 Lab Assignment 3 – Chapters 6-7

**(Last revised December 28, 2019. In the interest of time and because I assume you have already done some work with Try-Catch exception handling, we will require only Exercises 6.1 and 6.2X be done for this lab. You will be using exception handling as discussed in Chapter 7 in later Lab Projects, anyway.) All entered data should be validated before using in a calculation.**

Please read Chapters 6 and 7, and understand the material before Lab 3 begins.

Chapters 6 and 7 introduce procedures, event handlers, exceptions and data validation.

* Chapter 6 covers procedures and event handlers.
* Chapter 7 covers exceptions and data validation.

The lab assignment is to complete exercises as indicated below. Note that some of these exercises involve enhancing the exercises from Chapter 5, so these need to have been done already.

## Exercise 6.1: Enhance Future Value

After completing Step 7, rework the CalculateFutureValue method by making it a void function and adding a fourth parameter representing the future value amount to be returned by this method. Remember that this future value argument needs to be treated as a reference argument. Why?

Then complete this exercise by following steps 8-13.

A couple of things to be noted: 1) how to write functions with output arguments, and 2) the importance of putting as little code as possible in your event handlers, and instead moving the details required in an event handler to separate functions called by your event handlers.

## Exercise 6.2 X: Federal Income Tax (with a lookup table)

In this exercise, you’ll use a search or lookup table to calculate the federal income tax that is owed for a taxable income amount entered by the user.



Below is the 2015 table for the federal income tax on individuals that you should use for calculating the tax. Find an IRS table on the web for 2021 tax rates. Note that this will affect the income tax owed as listed in the above figure. Calculate the revised tax by hand and compare it to the answer provided by your project. They should match. If you stick with this table, the correct amount should be $4841.25

|  |  |
| --- | --- |
| **Taxable income** | **Income tax** |
| **Over…** | **But not over…** |  | **Of excess over…** |
| $0 | $9,225 | $0 plus 10% | $0 |
| $9,225 | $37,450 | $922.50 plus 15% | $9,225 |
| $37,450 | $90,750 | $5,156.25 plus 25% | $37,450 |
| $90,750 | $189,300 | $18,481.25 plus 28% | $90,750 |
| $189,300 | $411,500 | $46,075.25 plus 33% | $189,300 |
| $411,500 | $413,200 | $119,401.25 plus 35% | $411,500 |
| $413,200 |  | $119,996,25 plus 39.6% | $413,200 |

1. Start a new project named TaxCalculator in the Extra Exercises\Chapter 06\TaxCalculator directory.
2. Add labels, text boxes, and buttons to the default form and set the properties of the form and its controls so they appear as shown above. When the user presses the Enter key, the Click event of the Calculate button should fire. When the user presses the Esc key, the Click event of the Exit button should fire.
3. Add the usual “Enter Name” label, text box and OK button and ensure that a non-blank name is entered. Make sure that at least the taxable income label and text box are disabled until the non-blank name is entered.
4. Create an event handler for the Click event of the Exit button that closes the form.
5. Write a method (to be called by the Calculate button event handler which validates the taxableIncome entered by the user. This value must be a decimal and it must be non-negative.
6. Create another event handler for the Click event of the Calculate button which carries out the required calculation. Write the code for calculating and returning the tax owed for any valid taxableIncome amount. This code should be written in a separate method called taxTableLookup which takes as its input a taxableIncome value (a decimal), locates the table entry for the correct range for this input, and computes and returns the taxOwed using the information for the located range. (Thus for an income of $100,000, the tax owed would be $18,481.25 plus 28% of ($100,000 - $90,750) or $21,071.25.) Don’t forget to then display the taxable amount and the tax owed in the Click event.
7. To test this code, use income values of 8700 and 35350, which should display taxable amounts of 870 and 4840.50. Also test for invalid tax amounts, including a negative amount.

## The Exercises (below) are not to be done for Spring 2021

## Exercise 7-1: Enhance Invoice Total

## (First complete Exercise 7.1 in the text)

## Once you have inserted the try-catch statement into your program, if for any reason, the first use of the variable subtotal (in computing discountAmount) is flagged with an unassigned local variable error, see if you can find a work-around. If all else fails, insert the following code line into the catch clause:

subtotal = 0m;

## Exercise 7-2 X: A Simple Calculator

In this exercise, you’ll create a form that accepts two operands and an operator from the user and then performs the requested operation.



1. Start a new project named SimpleCalculator.
2. Add labels, text boxes, and buttons to the default form and set the properties of the form and its controls so they appear as shown above. When the user presses the Enter key, the Click event of the Calculate button should fire. When the user presses the Esc key, the Click event of the Exit button should fire.
3. Code a private method named Calculate that performs the requested operation and returns a decimal value. This method should accept the following arguments:

Argument Description

**decimal operand1** The value entered for the first operand.

**string operator1** One of these four operators: +, -, \*, or /.

**decimal operand2** The value entered for the second operand.

1. Create an event handler for the Click event of the Calculate button. This event handler should get the two numbers and operand the user enters, call the Calculate method to get the result of the calculation, display the result rounded to four decimal places, and move the focus to the Operand 1 text box.
2. Create an event handler for the Click event of the Exit button that closes the form.
3. Create an event handler that clears the Result text box if the user changes the text in any of the other text boxes.
4. Test the application to be sure it works correctly.
5. Once this has been done correctly, add a try-catch statement in the btnCalculate\_Click event handler that will catch any exceptions that occur when the statements in that event handler are executed. If an exception occurs, display a dialog box with the error message, the type of error, and a stack trace. Test the application by entering a nonnumeric value for one of the operands.
6. Add three additional catch blocks to the try-catch statement that will catch a FormatException, an OverflowException, and a DivideByZeroException. These catch blocks should display a dialog box with an appropriate error message.
7. Test the application again by entering a nonnumeric value for one of the operands. Then, enter 0 for the second operand as shown above to see what happens.
8. Once this has been done correctly, code methods named IsPresent, IsDecimal, and IsWithinRange that work like the methods described in chapter 7 of the book.
9. Code a method named IsOperator that checks that the text box that’s passed to it contains a value of +, -, \*, or /.
10. Code a method named IsValidData that checks that the Operand 1 and Operand 2 text boxes contain a decimal value between 0 and 1,000,000 (non-inclusive) and that the Operator text box contains a valid operator.
11. Delete all of the catch blocks from the try-catch statement in the btnCalculate\_Click event handler except for the one that catches any exception. Then, add code to this event handler that performs the calculation and displays the result only if the values of the text boxes are valid.
12. Test the application to be sure that all the data is validated properly.
13. Now you are done.