

Programming Logic - Beginning

152-101

Unit 4 – Catching Exceptions Lab

5 points

Name _____

Score _____ / 5

Update Value

[Make all corrections and resubmit to earn update points](#)

Overview

As you have no doubt experienced, it is very easy to crash your program by entering text where the program expects a number or even leaving numeric TextBoxes empty. Other run-time errors can also occur (divide by 0). In this lab you'll learn a simple technique to intercept these run-time errors so your program doesn't crash.

1. Use your Bowling program from Lab 2. You will need the Welcome Screen that you linked to the main program in Lab 2.
2. Open the Bowling project, open the window in Design View and then open another tab in Code View.

Catching and Handling Exceptions

The technique we will employ for error checking uses a programming tool called Try-Catch. Most programming languages include some form of Try-Catch. The Try-Catch construct allows you to *try* a chunk of code. If a run-time error occurs (technical term: *an exception is thrown*), Try-Catch senses it, *catches* it and allows you to insert code to *handle* the error. The program never crashes.

There are limits to the effectiveness of this technique (described below), but at least your program won't crash any more. In Programming Logic-Intermediate you'll learn more advanced error handling techniques.

3. Open Code View for frmBowling and, if you need to, expand the btnCalculate_Click event.
4. After the variable declarations, before the transfer of inputs:
Type `try` and then press the Tab key twice. Visual Studio inserts a Try-Catch block into your code.
5. Move the code that was originally in the method (except the variable declarations) (about 18 lines of code including comments) underneath the `try` command.
6. Press the Tab key (with all that code still highlighted) to correctly indent the code under `try`.

If any statement in the `try` block causes an exception to be thrown, the rest of the code in the `try` block is skipped and the code jumps to the `catch` block and executes the code there. If no errors occur, the `catch` block is not executed—it is skipped.

In this program, the only statements that **could** cause a run-time error are the three Parse statements. We could have surrounded only the Parse statements with the Try-Catch. However, if any of the inputs fail, the rest of the program is basically useless so it shouldn't execute.

7. Next, we have to code the `catch` block. Remove the `throw;` command entered by Visual Studio and replace it with the following:

```
MessageBox.Show("One of your inputs contains an error.\n" +  
                "Please check your inputs and calculate again.");
```

There is no (easy) way to distinguish which of the inputs has the error. That's what you'll learn how to do in Programming Logic-Intermediate.

The `MessageBox.Show` method in its simplest form displays a single string. If you look closely, you'll see these two strings are concatenated into one.

The `\n` is the C# *new line* character. When the string displays, it will be split into two lines before the word *Please*.

If you wish, you may insert your own, more *colorful* error message.

The `MessageBox.Show` method has other options, but we'll leave that for another unit.


8. Run the program, clear the test data and click the Calculate button. The `MessageBox` should appear. Note how the error message appears on two lines.

Testing to ensure the Try-Catch works is why you were required to include one *error* test in your test plans.

9. Return to the code and modify the method description for `btnCalculate_Click`. Add the following line: *If an error occurs, a dialog box is displayed letting the user know.*

Program Documentation

In Unit 2, you included program documentation in the Welcome Screen and States program windows themselves. Now however, there is more than one window in the project. Which one should include the program documentation? Answer: neither—we're going to move the documentation to a separate file.

10. Click the drop down arrow next to the Add New Item button on the toolbar  (2nd from the left). Select the Add New Item option.
11. In the dialog box that appears, choose Code File. Before clicking Add, change the name of the new code file to `_Documentation`. Note there is an underscore before the name. This will ensure the (important) documentation file appears first in the Solution Explorer.
12. Click the Add button.
13. The `_Documentation.cs` file is just an empty file. This suits our purpose because all we're going to add to it are comments.

14. On the first line, start a block comment (/*). Press Enter a couple of times and enter the symbols to end the block comment (*). Place your cursor on the line below the start of the block comment. Press Tab.

Insert the following documentation. Insert your name and the current date where appropriate. C# will automatically add a * to every new line. You may leave it or remove it.

```
Project Name:      Bowling
Programmer Name:   insert your name here
Due Date:         insert the project due date here

Purpose:          This application accepts three bowling scores from
                  the user. It calculates and displays the
                  series (total) and average of the three games.
                  It also determines and displays the next bowling
                  date.

Change Log:
    201x-mm-dd: Initial program submittal
```

This documentation should look familiar—it's the same documentation we used in Unit 2 just stored in a separate file. Remember, our standards dictate that the application purpose be grammatically correct.

Now that our project has more than one window, our school standards require that you give a brief description of each window as part of the program documentation. The standards also require that the windows be listed in alphabetical order.

15. The window descriptions should appear just before the Change Log. Move the Change Log entry down a couple of lines to prepare for the window descriptions.
16. First, we'll enter the description of the About window. Include this description (as an example) even if you did do that part of Lab 2.

```
window:   wdwAbout
Purpose:  This window displays information about the programmer and the
          program.
```

Note that window descriptions must be complete, grammatically correct sentences (standards).

17. Now enter the information for wdwBowling and wdwWelcome.

```
Window:   wdwBowling
Purpose:  This is the application's main window. It accepts three bowling
          scores from the user. It calculates and displays the series (total)
          and average of the three games. It also determines and displays the
          next bowling date.

Window:   wdwWelcome
Purpose:  This window welcomes the user to the program. It also provides the
          user a means to cancel program execution.
```

Note the main windows description is very similar to the program's purpose (as you might expect).

18. If you are in the lab with the instructor present, demonstrate the following (in order). If the instructor is not available, zip and email your copy of the Bowling application.

- ☐ Run the program and click the Clear button.
- ☐ Click the Calculate button. MessageBox should appear.
- ☐ Open the _Documentation.cs file for review.
- ☐ Exit the program and claim your points.