Programming Exercise 03
Exception Handling

C# Step by Step

June 13, 2018

This activity consists of four programming exercises. The following exercises are open book and open note. You are free to use any written documentation you wish. However, these are individual exercises, and you cannot consult with each other in writing your programs.

This programming exercise has four parts consisting of four requirements. The grade for each requirement is indicated, for a maximum of 100 points. At a minimum, your program must compile successfully and run.

This exercise builds on Exercise 1, Mathematical Formulas. You may have noticed that some inputs do not work, and that sometimes users make errors in inputs. In this exercise, we will address some of these issues.

Format exceptions: 70 points For these formulas to work, you must enter a number consisting of digits — alphabetical and punctuation characters will not work. For these three parts, add a try/catch block for each formula that handles a FormatException error.

Numbers less than zero: 80 points Additionally, for these formulas to return valid values, input must be greater than zero. You cannot have a circle with a negative radius, or a triangle with a negative side! Add a try/catch block for a custom exception using a if/else filter or some other technique to guard aghainst users entering numbers equal to or less than zero.

Finally block: 90 points Modify your program to add a finally block. This actually doesn't have to do anything, as we haven't covered network connections, file handles, etc. Just have it print a message such as, "Your number is okay."

Variable overflow: 100 points In the examples in the book, the exception handling techniques al terminate the program with an error message. In production software, you should notify your user of the error and allow recovery from the error. In other words, the exception should not terminate the program, but notify the user of the kind of error and allow the user to recover. As you saw in Exercise 02, methods can recursively call themselves, which would be a proper solution here. Unfortunately, you cannot call the parent method from a *catch* block. For full points, implement this functionality. Your output might perhaps look something like this:

```
Part 1, circumference and area of a circle.
Enter an integer for the radius: badinput
You must enter a valid number.
Okay
Enter an integer for the radius: -1
Your number is out of range
Okay
Enter an integer for the radius: 3
Okay
The circumference is 18.8495559215388
The area is 28.2743338823081
```