Scalable Data Infrastructures - Project 03

Overview

In this activity, you'll use functions to create a rudimentary calculator that will allow you to enter two numbers and an operation to receive an outcome based on the operation entered. I recommend you read over the instructions carefully and create a problem analysis to ensure you're covering all requirements.

Instructions

As in all projects, it's best to start with a problem analysis. While one is not required to be submitted for this code exercise, it would benefit you greatly to begin by examining the requirements and creating a bulleted list of the things you'll need to complete the code. You can then begin by creating a new solution with the following naming format: *LastName_FirstName_PR03*.

The project should contain the default Main method as we've seen in previous projects. You're going to create custom methods that will be called from the Main method. These custom methods will perform specific tasks to validate user input and perform a calculation on the numbers entered.

You'll ask the user for input in the Main method. Each time the user enters something, you'll validate the input by calling one of two validation methods. Each method will return a value, so think about what is needed in order to capture that value in the Main method.

The first validation method should accept a string data type, which will be the number entered by the user. A loop inside the method will use TryParse and ask the user to try again if they did not enter a number. Once the user input is validated, you will return that number to the Main method. This method will be used to validate both numbers, thus, it will be called twice. Keep in mind that the user should be allowed to enter floating point numbers, not just whole numbers.

The second validation method should also accept a string data type, which will be the math operator entered by the user. This time the loop should verify that the user entered something and validate that what was entered is a proper mathematical operator (+, -, *, /). Once validated, the method will then return the string of that operator.

Once the input is validated, you'll send all three values into a third custom method. Thus, the third custom method will accept the two numbers and the mathematical operator as arguments. Using the operator string, you'll use a conditional to determine what mathematical operation needs to be performed. You'll then perform that operation using the two numbers and return the outcome to the Main method. The number returned from the method should be stored in a return value variable and then output to the console in some meaningful way so that a user can see what numbers have been used, what operation was performed, and what the outcome is.

Something to keep in mind: the code should always output meaningful data. Thus, if a user were to try to divide by zero, the code should not throw an error; it should give the user useful feedback. How might you ensure that this happens?

When complete, compress all the files **into a single zip file** as *lastname_firstname_lab5.zip* and upload it to this activity.

Things to Consider

Look to your problem analysis to verify that you've included all the requirements of the problem. If you ask your instructor or a lab specialist for assistance on the code, he/she is going to ask to see your problem analysis first. If you didn't do that, we are going to tell you to complete that task before we will look at your code.

Make sure you're including comments at the top of the code to include your name, class and term, and the assignment name, and that you have meaningful comments for each line of code in the project.

Finally, remember that you must compress the entire project folder for submission. Submitting only the Program.cs file or the .sln file will result in a 0 for the activity.

Rubric: Project 03

Minimum Project Requirements

These requirements must be satisfied before any points are awarded. Failing to meet these requirements will result in a zero (0) grade.

1. Project must run when instructor compiles it.
2. The submission must be submitted in the proper format as defined in the FSO activity.
3. You will lose 5 points if the project does not follow the naming convention described in the activity's documentation.

Topic	%	Excellent (100%)	Acceptable (80%)	Good (50%)	Fair (25%)	Poor (0%)
Coding						
Comments	**5**	Comments exist at the top of the code to include name, class and term, assignment, and date, and each line of the code is properly commented.	Missing the initial comments with name, class and term, assignment, and date, but the rest of the code is commented properly.	Missing up to four line comments, but some comments present.	Missing more than four comments	No comments in the code.
Syntax	**10**	There are no syntax errors, including correct line and **formatting according to the style taught**.	There are no syntax errors, but the code does not follow the style taught.	Project code contains minor syntax errors but is easily fixed.	Project code contains more major syntax errors but are easily fixed.	Project code does not run.
Methods	**25**	All required methods are created and include all the requirements of the method signature, including a descriptive, PascalCase, active-verb name with correct return and parameter.	A custom method is created, but the name is not descriptive or not an active verb or not in PascalCase.	Missing up to one required method.	Missing more than one required method, but there is a custom method in the project.	**No custom method exists in the submitted project. Zero (0) for the entire exercise.**
Arguments/Parameters	**15**	All required arguments/parameters are present and of the correct type with descriptive names.	One missing argument/parameter or incorrect data types used.	Missing more than one required argument/parameter.		The custom methods contains no parameters.
Returns	**15**	All required return values are present, assigned, and of the correct type. All returned values are stored in return value variables.	One missed, unassigned, or mistyped return values, or up to one returned value not stored in a variable.	No returned values are stored in return value variables.		The custom methods do not return values.
Validation	**10**	The user inputs are validated with the proper loop, parsed to the proper data type, and the validation is encapsulated in the code block of the required custom methods.	The user input is not converted/parsed to the correct data type.	User input is validated with a conditional, not a loop.	User input validation is contained in the Main method, not in the custom method.	The user input is not validated.
Method Calls	**10**	The methods are properly invoked with the correct argument data being sent to the custom methods and with a returned value variable in place to capture the value returned from the methods.	The methods are invoked, but missing at least one returned value variable to hold returned values or missing at least one argument in the method calls.	The methods are invoked, but no arguments are sent into the methods or not returned values are properly stored in the Main method.		The custom method is not invoked.
Output	**10**	Output is meaningful and done in the Main method.	Output is done in the Main method, but the text is not meaningful to the user.			The output is not done in the Main method.