

CIS 345 – Business Information Systems Development II – Fall 2014

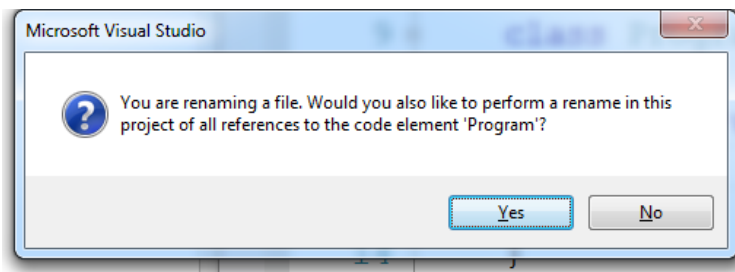
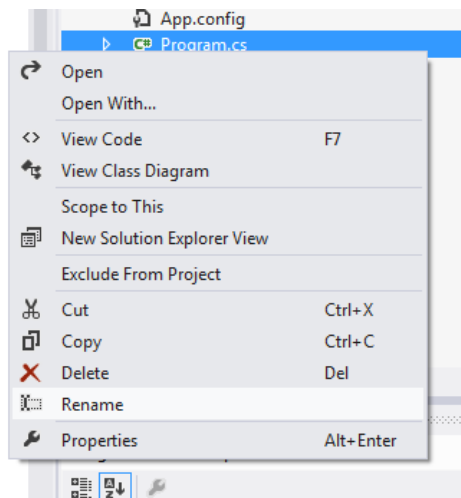
Assignment 2: Methods

Due on Blackboard on Friday, September 19 by 10 PM

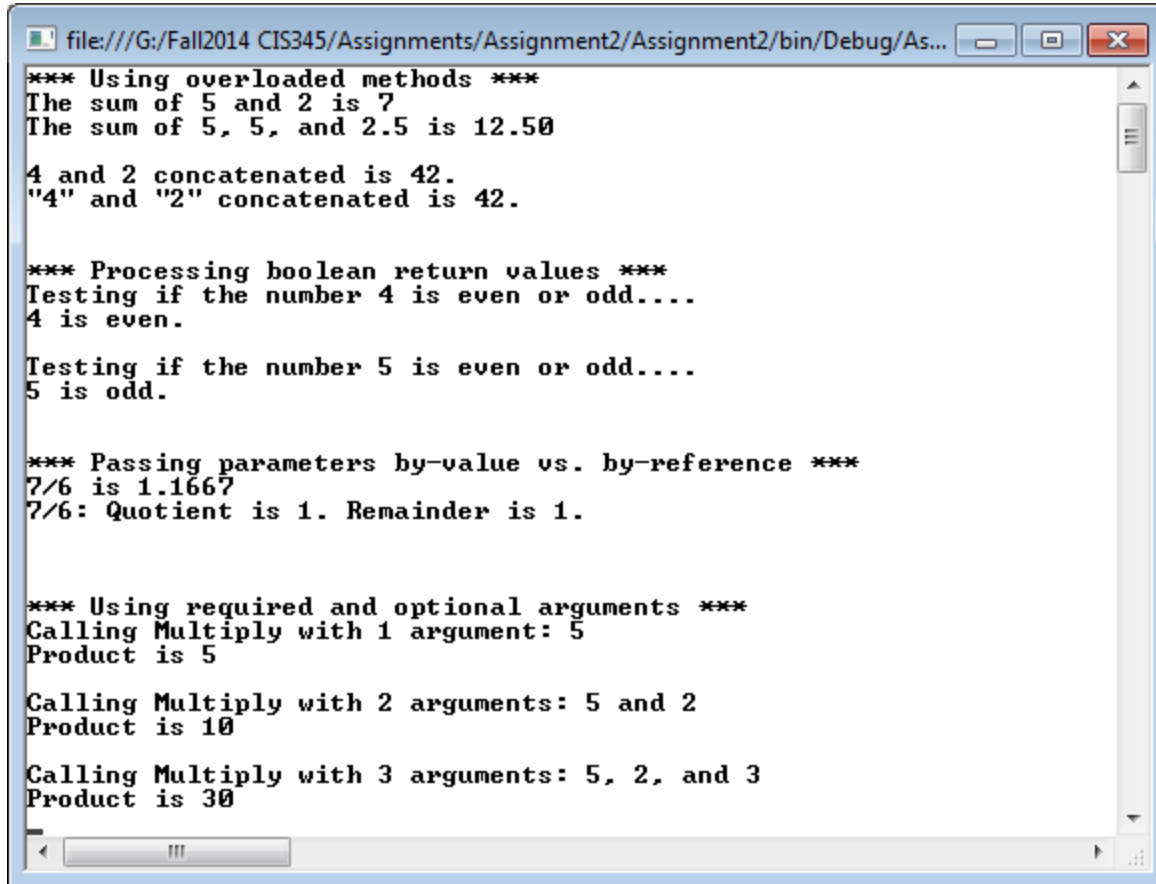
Skills developed: Writing methods, writing overloaded methods, using parameters, calling overloaded methods, passing arguments, returning values, using returned values, passing by-value vs by-reference, using optional arguments.

Create a new Visual Studio 2012/13 project. Start with the project name “Assignment2”.

Rename your class name from Program.cs to Methods.cs by right-clicking on the filename in Solution Explorer and clicking rename. Accept its recommendation to change the name of the class throughout the project code.



Sample Output



```
file:///G:/Fall2014 CIS345/Assignments/Assignment2/Assignment2/bin/Debug/As...
*** Using overloaded methods ***
The sum of 5 and 2 is 7
The sum of 5, 5, and 2.5 is 12.50

4 and 2 concatenated is 42.
"4" and "2" concatenated is 42.

*** Processing boolean return values ***
Testing if the number 4 is even or odd....
4 is even.

Testing if the number 5 is even or odd....
5 is odd.

*** Passing parameters by-value vs. by-reference ***
7/6 is 1.1667
7/6: Quotient is 1. Remainder is 1.

*** Using required and optional arguments ***
Calling Multiply with 1 argument: 5
Product is 5

Calling Multiply with 2 arguments: 5 and 2
Product is 10

Calling Multiply with 3 arguments: 5, 2, and 3
Product is 30
```

Guide on notation and capitalization style

CIS 345 will follow the following Microsoft .NET Framework naming conventions. Other languages and platforms have different conventions, which we will not use or mix with .NET conventions.

- Class, method, and *public* variable names are in PascalCase, i.e. first letter of each word in uppercase, e.g. “BackColor”, “FontSize”
- Parameter names and *local* variables are in camelCase, i.e. first letter of first word in lower case, then first letter of each subsequent word in upper case, e.g., “backColor”, “fontSize”
 - Compound words are treated as one word, e.g. “username”, “filename”
- Variable and method names are not abbreviated – they are spelled out in full.
- Data types are not included as part of the name (Hungarian notation is not used).

Create and complete the following methods in your `Methods.cs` class. All methods should be static. You MUST use the variable and method names that are given to you. You MUST follow the exact sequence of the parameters given to you.

Understand what a method is supposed to do before you write it!

1)

Method Name: Add

Access modifier: Private

Parameters: 2 integers, called number1 and number2.

Return value: integer

Purpose: This method returns the sum of two integers

In the body of the method block,

- Return the sum of the two parameters.

2)

Method Name: Add

Access modifier: Private

Parameters: 3 Parameters as listed below.

- 2 integers, called number1 and number2
- 1 double, called number3

Return value: double

Purpose: This method calculates the sum of two integers and a double and returns the total as a double.

In the body of the method block,

- Return the sum of the three parameters. *Check: is your return value of type double? Do you need to convert it to a double explicitly when there are 2 integers and 1 double?*

3)

Method Name: Concatenate

Access modifier: Private

Parameters: 2 strings, called string1 and string2

Return value: string

Purpose: This method joins two strings

In the body of the method block,

- Return the two strings joined together. You can join the two strings using the plus (+) operator. *The plus operator is overloaded too – it behaves differently based on input type!*

4)

Method Name: Concatenate

Access modifier: Private

Parameters: 2 integers, called number1 and number2

Return value: string

Purpose: This method converts two integers to strings, joins them, and returns one joined string.

In the body of the method block,

- Declare a string, called outputText. Initialize it with a blank string.
- Call the Concatenate method and pass it number1 and number2 as arguments. Use .ToString() after the integer variables to convert the numbers to string within the method call.
 - Store the return value in outputText.
 - *This concatenate method is making a call to another overloaded Concatenate method that accepts string parameters.*
- Return outputText.

5)

Method Name: IsEven

Access modifier: Private

Parameters: integer, called number

Return value: boolean

Purpose: This method returns a true or a false indicating if the number passed to it is even or not.

In the body of the method block,

- Using the % operator, test if number is even. If it is, return **true**. Otherwise, return false.

6)

Method Name: Divide

Access modifier: Private

Parameters: 2 integers, called number1 and number2.

Return value: double

Purpose: This method returns the quotient after dividing two numbers.

In the body of the method block,

- Declare a double called quotient. Initialize it with a value of 0.0.
- Divide number1 by number2. When you are dividing the two numbers, *cast* number1 as a double i.e. force it to be used as a double, by preceding it with the cast operator, i.e. "(double)" (no quotes needed). Assign the resulting value to the variable, quotient.
- Return quotient.

7)

Method Name: Divide

Access modifier: Private

Parameters: 4 Parameters as listed below

2 integers, called number1 and number2

2 integers, called quotient and remainder (both passed by reference using ref keyword)

Return value: integer

Purpose: This method divides two numbers and stores the result in parameter variables that have been passed to it by reference.

In the body of the method block,

- Divide number1 by number 2. Assign it to the quotient parameter. Make sure you are performing integer division i.e. do not cast/convert any number into a double.
- Use the remainder operator (%) on number1 and number2 and assign the result to the remainder parameter.
- Return the number zero. *This return value doesn't mean anything right now. In the future, we will be able to utilize this return value to send error codes if something goes wrong, e.g. division by zero, etc.*

8)

Method Name: Multiply

Access modifier: Private

Parameters: 3 integers, with the following characteristics:

number1, required parameter

number2, optional parameter, default value of 1

number3, optional parameter, default value of 1

Return value: integer

Purpose: This method returns the product of the three parameters.

In the body of the method block,

- Multiply all the three parameters and store the product in a variable called product (you will need to declare the variable).
- Return product.

9) Method Name: Main

In the body of the Main method block,

Call the two Add methods:

- Call the Add method – provide it with arguments, 5 and 2. Store the return value in an integer variable called additionResult (declare it). Output the sum using a WriteLine e.g. “The sum of 5 and 2 is 7.”
- Call the Add method – provide it with arguments, 5, 5, and 2.5.
 - Do not store the return value in a variable. Instead, use it inline within a WriteLine.
 - The WriteLine statement should display the total on the Console e.g., “The sum of 5 , 5 and 2 is 12.50” Obtain the value of 12.5 from the return value of the method call. Use place-holders to position the return value within the string. Also, within the place-holder, format the value as a float with two decimal places (use :f2).

Call the Concatenate methods and use the method calls inline within a WriteLine statement:

- Call Concatenate and provide it with 4 and 2 as integers. Use the return value within the WriteLine statement (using placeholders). The output on the Console should say “4 and 2 concatenated is 42.”
- Call Concatenate and provide it with “4” and “2” as strings. Use the return value within the WriteLine statement (using placeholders). The output on the Console should say “ “4” and “2” concatenated is 42.” *Precede your inner quote characters with escape character (\) to ensure you can put a quote within quotes.*

Call the IsEven method:

- Declare a boolean variable, numberIsEven. Initialize it with the value false.
- Write a statement to the console stating that you are going to test if 4 is even or odd.
- Call IsEven and supply it with 4 as an argument. Store the return value in numberIsEven.
- Write a statement to the Console window stating “4 is ”. Complete the rest of the sentence by using an if statement
 - Test if numberIsEven is true or not. If it is true, write “even”, otherwise write “odd.”
- Repeat the testing for even numbers using a value of 5.

Call the first Divide method:

- Call the Divide Method - provide it with arguments, 7 and 6. Store the return value in a double variable called divisionResult (declare it). Print divisionResult to the Console window. Show 4 decimal places.

Call the other overloaded Divide method:

- Declare two integer variables, number1 and number2. Initialize them with values 7 and 6 respectively.
- Declare two integer variables, quotient and remainder. Initialize both to zero.
- Call the Divide method with arguments: number1, number2, AND the variables quotient and remainder.
 - Pass number1 and number2 “by value.”
 - Pass quotient and remainder “by reference.”
 - Store the return value in divisionResult (*don't declare it again!*).
- Print the quotient and the remainder to the Console window. Use the two variables you supplied as arguments. *The return value is not used.*

Demonstrate using optional arguments. Call the Multiply method three times, with a varying number of arguments.

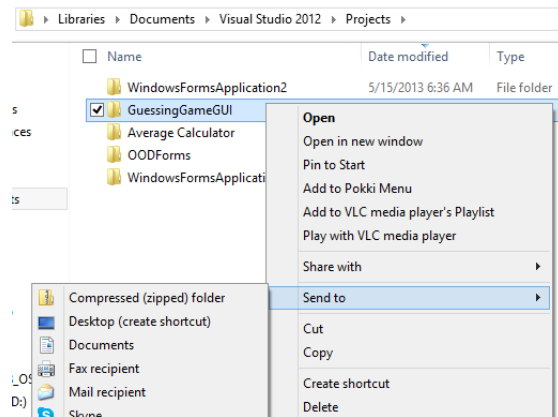
- Call the Multiply method – provide it with one argument, 5. Store the return value of the method in an integer variable called product (declare it). Display the product on the console using a WriteLine. Be sure to list how many arguments you used.
- Call the Multiply method – provide it with two arguments, 5 and 2. Compute and display the product as above. *Don't declare product again.*
- Call the Multiply method – provide it with three arguments, 5, 2, and 3. Compute and display the product as above. *Don't declare product again.*
- Finally, make sure you insert a ReadLine or ReadKey to make the program pause at the end.

Submission Instructions

NOTE CHANGE OF SUBMISSION PROCEDURE BELOW

Submission should be made using a zip file that contains all of the Visual Studio C# project files. You will need to **zip the entire project folder** along with the .sln and .suo files. The folder will automatically contain the class source files as well as the executable file that is generated in \ProjectName\bin\Debug folder. Upload file to the Blackboard assignment drop box.

Zip the entire top-level folder by right-clicking the folder and selecting Send to | Compressed (zipped) folder.



Using built-in windows zip tools: <http://windows.microsoft.com/en-us/windows/compress-uncompress-files-zip-files>

Make sure you check the following. Your grade is dependent on all these criteria being met.

- You have included your name as a comment within your class.
 - e.g. “ // Assignment 2, Jane C. Smith, CIS 345, Tuesday 9:00 AM”
- Class file is called Methods.cs (rename from Program.cs).
- Your Visual Studio project is called Assignment2.
- Zip filename is: **FirstNameLastName_Assignment2.zip**

Verify your zip file before you submit

- Check for actual class files being present in the folder before you zip it.
- Check your zip file size after zipping – if it is 1K, it likely contains only a shortcut.
- Uncompress your zip file before submitting and verify that files are present.
- Download your zip file after submitting, uncompress, and again verify that your files are present.

This takes an extra couple of minutes. Please do it if your grade is important to you. If you do this, you will not end up submitting a bad file. If you submit an empty file, or one containing only a shortcut, or a bad zip file, or a bad project file, you will receive a score of zero and your only recourse will be to do the makeup assignment at the end of the semester.