Programming Assignment 4.2

**Assignment Description**

You're really excited about the four different kinds of methods we can have, so you want to implement all of them. Because you also recently learned about console app classes, you also want to put those methods into a separate class (instead of in the **Program** class).

In this assignment, you'll implement all four kinds of methods in a class.

**Why do we care?**

You'll write all kinds of different methods as you develop games, so it's important to practice doing that.

**Getting Started**

Download the appropriate zip file for your OS below and unzip the file somewhere on your computer.

**Windows Programming Assignment 4.2 Materials** ZIP File

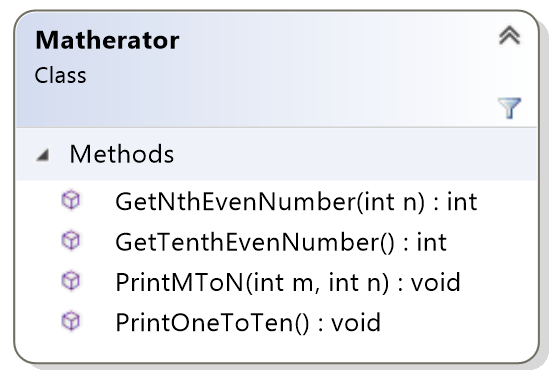
**Mac Programming Assignment 4.2 Materials** ZIP File

Open the project in Visual Studio.

**Important:** You MUST only add code as indicated by the comments in the Matherator.cs file. If you don't, you're virtually guaranteed to fail all the test cases in the automated grader.

**Requirements**

For this assignment, you need to implement a **Matherator** class that exposes the methods shown in the class diagram below:



The Visual Studio project I provided to you contains the **Matherator** class (with the method bodies unimplemented) and a **Program** class that tests the **Matherator** class. You shouldn't change the Program.cs file at all (if you do, you'll break the automated grader), all your work in this assignment is in the Matherator.cs file.

You've probably noticed that these are the four kinds of methods we can have. Remember that you can call one method from another; you should do that as much as possible to avoid duplicating code.

The **GetNthEvenNumber** method gets the nth even number (based on the **n** parameter) and returns it. This method assumes 2 is the first even number. Note that this method doesn't print that number, it returns it from the method.

The **GetTenthEvenNumber** method gets the tenth even number and returns it. This method assumes 2 is the first even number. Note that this method doesn't print that number, it returns it from the method.

The **PrintMToN** method prints the numbers from m to n, inclusive, based on the **m** and **n** parameters.

The **PrintOneToTen** method prints the numbers from 1 to 10.

The best approach is to implement one of the **Matherator** methods, then test it by running the code, then move on to the next **Matherator** method, and so on. I implemented the methods in the following order: **PrintOneToTen**, **PrintMToN**, **GetTenthEvenNumber**, **GetNthEvenNumber**.

**Required Output Format**

For the two **Matherator** methods that print (rather then return a value), print each value followed by a single space, then call the **Console.Writeline** method when you've printed all the values. For example, calling the **PrintOneToTen** method should yield the following output:

**1 2 3 4 5 6 7 8 9 10**

**Note: The last value in your output (10 for the example above) must be followed by a single space**. Trust me, it’s easier that way, so the automated grader assumes you do that! The Program.cs file I provided makes sure your output from calling the methods that return a value is printed in the correct format.

We’d typically label our output to tell the user what those numbers are, but that will just confuse the automated grader. You must print ONLY in the format described above. For the two methods that return a value (rather than printing values), the Program.cs file I provided prints that values properly.

**Running Your Code**

Because of the code I included to work with the automated grader on Coursera, when you run your program the command prompt window will open and it will sit there doing nothing. To make your code run, type in 3 integers with a single space between each one and press the <Enter> key; your code should then run so you can check your output.

You'll provide input using the following format:

**<operation> <first argument <second argument>**

The first input value tells what operation to perform: 1 for get the nth even number (where n is the value of the first argument), 2 for get the tenth even number (neither argument is used), 3 for print the numbers from m to n (where m is the first argument and n is the second argument), and 4 for print the numbers from 1 to 10 (neither argument is used).

The second and third input values are the arguments to use when calling the method for the selected operation. Only one of the methods requires both arguments, but it's easier for the automated grader to always process the same input format and just ignore the input values it doesn't need.

For example, your input could be

**3 2 4**

to print the numbers from 2, 3, and 4.

You can actually run your code again if you want to by typing in 3 integers with a single space between each one and pressing the <Enter> key again. When you’re ready to stop running your code, type q (for quit).

Here's what running the code multiple times with different inputs should look like. The first line is your input line with the operation and the two arguments, the second line is your output line, and so on:

**3 2 4**

**2 3 4**

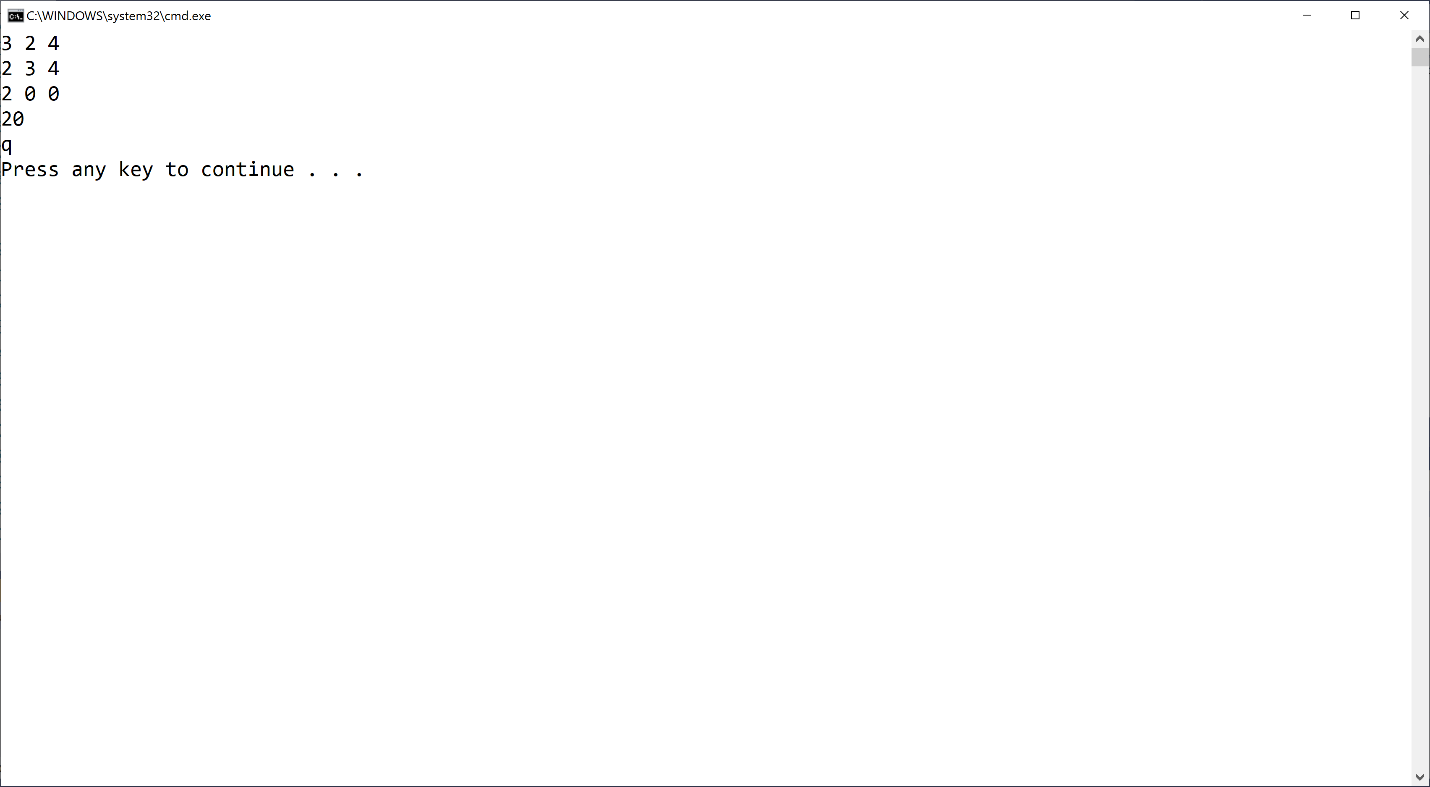
**2 0 0**

**20**

**q**

**Important Note:** The Coursera formatting makes it look like there's a blank line between each of the lines above, but there's not. The above output should be exactly 5 lines of input/output.

The image below shows my console window when I run the code multiple times as described above:



If your output doesn't match the image above EXACTLY (no extra words, characters, spaces, or blank lines) you'll fail all the test cases in the automated grader. **Remember, even though you can't see it above, each output line has a single space at the end.**

Mac users: You may or may not have an extra blank line after the q. That's fine.

**Test Case Inputs**

The automated grader uses the following set of test case inputs to test your code (1 test case per line):

**1 5 0**

**1 10 0**

**1 100 0**

**2 0 0**

**3 2 4**

**3 -4 -2**

**3 1 1**

**3 -1 0**

**3 -5 3**

**4 0 0**

You should figure out what the expected results are for each test case input and make sure your code is generating those expected results in the required format before submitting your code for grading.

**Common Problems**

Historically, lots of people post about grading errors on this assignment (because their code is wrong). Here are a couple things you should check:

1. You have to start with the Visual Studio project I gave you in the assignment materials and add your code in the appropriate places in the Matheratos.cs file. The code I gave you includes the appropriate structure for the automated grader to work. If you don't do this, you'll almost definitely fail all the test cases in the automated grader
2. I give you an example in this assignment (and all assignments) for what output you should get when you run the code multiple times. Be sure to try that example; if your code doesn't generate output exactly as shown in the example, your code isn't working properly

**Submitting Your Solution**

Go to the LMS and submit your project to **Assignment 4.2**