SOLID Principles

* Single Responsibility Principle – each class should have one reason to change member data.
* Open/Closed Principle – classes should be closed to modification but open for extension.
* Liskov Substitution Principle – base classes should be substitutable for their derived classes.
* Interface Segregation Principle – do not force classes to implement interface methods that they do not use.
* Dependency Inversion Principle – high-level modules should not depend on low-level details.

SOLID Principles violated in the original FizzBuzz

* SRP violated – there are four possible outputs for this program, so four conditions on which the code can change. This violates SRP which states that the class should have only one reason to modify member data.
* OCP violated – a modification to the output would require an additional “else if” statement, which modifies the core function. This violates OCP, since the output should have the ability to be extend output without modifying the core function.
* LSP violated – there is no inheritance or interfaces used in this method. A program with that modifies the output based on a specific condition should use an interface or an abstract class.
* ISP violated - the for loop implements an “else” condition, by default making a case for a condition that should not exist. A SOLID-compliant program should make conditions for specific cases and employ a default virtual method implementation if a default behavior is desired.
* DIP violated – everything in the program depends on a single for loop implementation. This makes a specific condition difficult to test.

Refactor Strategies

I will use the **Strategy pattern** to implement a strategy for reading different conditions (for example: *readModulusStrategy*). This interface will be responsible for handling different conditions of the logic of the FizzBuzz loop.

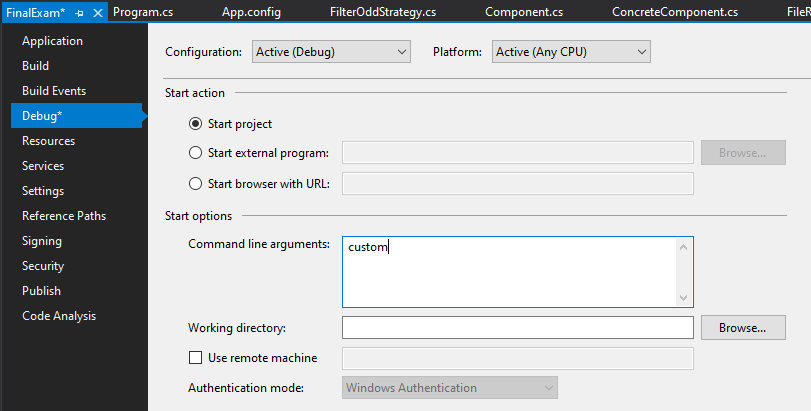
Each output is always going to be a *Console.WriteLine: some string*. This is a perfect opportunity to implement the **Decorator pattern**, as I will be decorating the outputs.

Client Configuration

Note: the program is shipped with default FizzBuzz output.

To load a custom configuration:

1. In the FinalExam program click on “Properties” and navigate to the “Debug” tab;
2. Add “custom” to the Command Line Arguments window.
3. Program will read your custom configuration saved on the config.txt file the FinalExam folder.



Configuration File Setup

The main configuration is read from **config.txt** file in the FinalExam folder.

First 5 numbers select the following options

* Output type:
  + 0 for console
  + 1 for to output to file *output.txt* in the FinalExam folder
* Start Index:
  + Selects start index; must be smaller than the total number of integers on the list
* Stop Index:
  + Selects stop index
* Filtering:
  + 0 for default
  + 1 filter out odd numbers
  + 2 filter out even numbers
* Printing:
  + 0 for default
  + 1 for reverse

After the ### comments, user may enter custom multipliers and words associated with them:

* Foo,3
* Bar,5
* Test,13

There is no limit on the number of custom multipliers and words, but they must be entered in the order and format shown above

After the last ### comments, user may enter any number of integers to be processed by FizzBuzz