CS 3201 – Fall 2020
Assignment 2, 20 pts.
Due: Monday, Sep. 21st at 9a.

Objectives

- Refactor code to remove code smell
- Add new functionality to existing code
- Create and use properties and methods within a class and to allow object collaboration
- Write basic UI code
- Use LINQ to perform queries and analysis on a data set.

Notes

- Please read through all the requirements before you begin, so you know what the program is supposed to do when you are done with the assignment.
- Make sure you use ReSharper to verify your code meets the style requirements.
    - Details on how to use ReSharper are available in Moodle.
- **Any program that does not compile will receive a zero (0). Partial credit is not possible for any program that does not compile.**

Getting started

1. The goal of this assignment is to improve and increase the functionality of the COVID-19 analysis project from Assignment 1. The starting point of this project is a correct and completed Assignment 1. You will use your submission from Assignment 1 as a starting point for this assignment, if there were bugs and/or code smell issues in your Assignment 1 submission, they must be fixed or full credit cannot be earned for this assignment.

2. The new and improved COVID-19 analysis project will have improvements in the following areas:

    a. Remove all bugs and add missing functionality from Assignment 1.

    b. Refactor the code to make use of best practices in class design and clean code that eliminates code duplicity, promotes reuse, and is code that is efficient, readable, and understandable.

    c. The ability to handle the ability to handle missing or incorrect data in the file.

    d. When loading a file, if data is already loaded, prompt the user if they want to replace or merge the loaded file into the existing data. Additionally, when merging the data, prompt the user on how to handle any duplicated days.

    e. Modify the output.

    f. Allow the user to adjust the threshold values to count positive cases below and above specified threshold values.

    g. Use LINQ, as appropriate, to perform queries and analysis on a data set.

Requirements

Requirement #1: Complete, debug, and refactor A1

1. Complete assignment 1 by adding missing functionality and removing all bugs, so that the assignment works according to the Assignment 1 specifications.

    a. Note: If your project had errors or missing functionality, those issues must be addressed in this submission. If these bugs still exist, major points will be deducted from the functionality score.

2. Refactor your Assignment 1 submission to eliminate all class design and code smell items that have been discussed in class, presented in the book: *Clean Code*, and/or noted in any specific feedback.

    Items that need to be addressed include, but are not limited to, the following:

    - Separation of concerns - placing functionality and corresponding state information in appropriate model, view, datatier, etc. namespaces and classes.
    - Making sure the implementation adheres to best practices, in particular, in regards to SRP and DRY.
    - Eliminate data duplicity within and between collaborating classes.
    - Address readability, e.g.,
        - Appropriate use of enums and/or constants for magic numbers
        - Formatting and ReSharper issues
        - Appropriate number of levels of code within a method
        - Appropriate use of white space
        - Writing self-documenting code
        - Adding required and appropriate documentation

Requirement #2: Loading files

1. The program needs to be capable of the following functionality when loading a file.

   Note: Some of this may already be handled in your Assignment 1 submission.

   a. A file may have multiple states of data within it. This occurred in Assignment 1. For this assignment all summary output data will still be for **Georgia only** with the exception of the lines with errors in them that must be displayed. Error lines will be displayed for any line regardless of the state.

   b. A file may be missing individual days and/or an entire month's worth of data may be missing for a state.

   c. An individual line may have corrupt or missing data. If this is the case, the line should be skipped and continue to read in the rest of the file. There should be functionality added to the UI so that the user can view the lines with errors including the line number in the file.

   d. When loading a file, if an existing file is already loaded in to the program, the user should be prompted if they want to replace the existing data or merge the new file into the existing data.

      i. If the user is merging into the existing data and a duplicate day is found, then the user should be prompt if they want to keep the existing day's data or replace the day.

         As a part of this, the user should be able to signify if they want the selected action applied for all duplicated days that are encountered when loading the file.

Requirement #3: Modifying the output

1. The output should be modified as follows:

   Note: All output will be for GA only.

   Note: This summary information is slightly modified from Assignment 1.

   i. The date of the first positive test in Georgia.
   ii. The highest number of positive tests and the date it occurred on.
   iii. The highest number of negative tests and the date it occurred on.
   iv. The highest number of tests on a day and the date it occurred on.
   v. The highest number of deaths and the date it occurred on.
   vi. The highest number of hospitalization and the date it occurred on.
   vii. The highest percentage of positive tests and the date it occurred on.
   viii. The average number of positive tests per day since the first positive test.
   ix. The overall positivity rate of all tests.
   x. The number of days from the first positive test with the number of positive tests greater than 2,500. (This value can now be modified by the user. See Requirement #4.)
   xi. The number of days from the first positive test with the number of positive tests less than 1,000. (This value can now be modified by the user. See Requirement #4.)
   xii. A positive-case count histogram. (New - see Requirement #5)

   The monthly breakdown must be displayed after the above summary information and it needs to display the same information as before which is noted below. The first month displayed will be the first month with a positive test for GA.

   i. The highest number of positive tests and the date it occurred on.
      1. This is inclusive of the first date of a test for a month (positive or negative test) and will include all days of data loaded after the first date of a test for that month.
   ii. The lowest number of positive tests and the date it occurred on.
      1. This is inclusive of the first date of a test for a month (positive or negative test) and will include all days of data loaded after the first date of a test for that month.
   iii. The highest number of total tests and the date it occurred on.
   iv. The lowest number of total tests and the date it occurred on.
   v. The average number of positive tests per day for the month
      1. This is inclusive of the first date of a test for a month (positive or negative test) and will include all days of data loaded after the first date of a test for that month.
   vi. The average number of total tests per day for the month.
      1. This is inclusive of the first date of a test for a month (positive or negative test) and will include all days of data loaded after the first date of a test for that month.

The monthly summary output must be modified as follows:

i. It should start with the first month that has a positive or negative test and continue through the last month with a positive or negative test.
ii. The year should be displayed next to the month name along with the number of days of loaded data for the month.
iii. If a month has no data, then still display the header, but nothing else.
iv. If there are multiple days with the same value found then display each day that value occurred on.

Example output for the monthly breakdown

March 2020 (26 days of data):
Highest # positive tests: 532 occurred on the 29th.
Lowest # positive tests: 0 occurred on the 2nd and 23rd.
Highest # total tests: 12,345 occurred on the 12th.
Lowest # total tests: 3,210 occurred on the 2nd.
Average # positive tests: 151.38
Average # total tests: 9,525.00

April 2020 (30 days of data):
Highest # positive tests: 753 occurred on the 1st, 5th, and 17th.
Lowest # positive tests: 252 occurred on the 1st.
Highest # total tests: 18,712 occurred on the 22nd.
Lowest # total tests: 8,251 occurred on the 3rd.
Average # positive tests: 552.40
Average # total tests: 13,205.42

May 2020 (0 days of data):

…

2. Formatting:
    a. Numbers should be displayed with commas to denote thousands separators.
    b. All values that are decimal values should be displayed to two decimal places.

Requirement #4: User inputs upper and lower thresholds

1. Add functionality to the UI so that the user can modify the lower and upper positive case thresholds.

2. The default threshold values should be 1,000 and 2,500, respectively.

Requirement #5: Positive case breakdown added to the overall summary

1. For the overall summary, create and display a positive cases breakdown as follows:

    a. The number of positive cases should be divided into 500 case segments, e.g., 0 to 500, 501 to 1,000, 1,001 to 1,500, …,. The count of the number of positive cases in each segment should be displayed. This count starts from the first day with a test and goes through the last day with a test.
    The last "bucket" should include the highest positive test count. For example, if the highest positive test count was 4,853 the last "bucket" will be from 4,501 to 5,000.

    Make sure all output is aligned in columns as shown below.

    Example output is below: (The numbers are not based on the data set.)

```
      0 -   500: 17
    501 - 1,000: 35
  1,001 - 1,500: 21
  1,501 - 2,000:  7
  2,001 - 2,500:  0
  2,601 - 3,000: 15
  3,001 - 3,500: 11
  3,501 - 4,000:  6
  3,001 - 3,500:  9
  3,501 - 4,000: 12
  4,001 - 4,500:  8
  5,501 - 5,000:  2
```

Requirement #6: LINQ

1. Use LINQ, as appropriate, to perform queries and analysis on a data set.

End of requirements

Data format

Data format is same as before, except there could be corrupt or missing data on a line.

The CSV file is in the following format.

```
date,state,positiveIncrease,negativeIncrease,deathIncrease,hospit
alizedIncrease
20200816,AK,106,4732,0,0
20200816,AL,853,8554,2,0
```

Submission

- Use ReSharper to clean up the code and address all ReSharper warnings.
- Make sure all required documentation is provided and complete.
- Add a notes.txt file to the project and note any missing functionality or bugs.
- Make sure you clean your solution (Build → Clean Solution) and then zip up your project which needs to include the solution file into a ZIP file called *YourName*A2.zip and submit the assignment by the due date.

Grading breakdown - 20 points (12 pts. – functionality; 8 pts – implementation)

- *Any program that does not compile will receive a 0. Partial credit is not possible for any program that does not compile.*
- *If a program is only partially complete and a category cannot be accurately assessed, you will not receive full credit for that category.*
- *One point will be deducted from your grade, if your submission does not follow the naming convention.*
- *No rubric will be used for the functionality scores. It will be graded on a continuous scale.*

Implementation rubric

If a program is only partially complete and a category cannot be accurately assessed, you will not receive full credit for that category.

	Exceptional	Acceptable	Amateur	Unsatisfactory
Readability	2 pts.	1 pt.	NA	0 pts.
Implementation	4 pts.	3 pts.	2 pt.	1 pts.
Documentation	2 pts.	NA	1 pt.	0 pts.

Grading description

	Exceptional	Acceptable	Amateur	Unsatisfactory
Readability	The program is exceptionally well organized, very easy to follow, and there are not any ReSharper warnings.	The code is fairly easy to read and there are a few ReSharper warnings that were not legitimately explained as to why they still exist.		The code is poorly organized and very difficult to read. Many ReSharper warnings remain.
Implementation	The code follows best practices in OO design & program development. (OODD) The code could be reused as a whole or each routine could be reused.	Most of the code follows best practices in OODD development. Most of the code could be reused in other programs.	Some of the code follows best practices in OODD Some parts of the code could be reused in other programs.	The code does not follow best practices in OODD or violates very basic OODD principles. The code is not organized for reusability.
Documentation	The required documentation is well written and clearly explains what the code is accomplishing. No redundant inline commenting.		The required documentation is there, but not complete or is only somewhat useful in understanding the code.	The documentation is poor and very incomplete.