CS 3201 – Fall 2020
Assignment 4, 30 pts.
Due: Monday, Nov. 9th at 9a.

Objectives

- Add new functionality to existing code.
- Determine an approach to solve a problem and implement that approach in code.
- Refactor existing code to reduce code smell and employ good class design principles.
- Apply clean coding practices.
- Use a version control system to share and work on a software project with other developers.
- Use issue tracking and a wiki to help manage a software project.
- Write basic UI code that incorporates data binding and the MVVM design pattern.
- Write data to a file using serialization.

Notes

- This assignment will be a group project. The groups for this assignment are posted in Moodle. The specifications of how the group will interact and work on this assignment are also included below.
- Please read through all the requirements you begin, so you know what the application is supposed to do when you are done with the assignment.
- Make sure you use ReSharper to verify your code meets the style requirements.
- **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

Getting started #1: Github

1. Select one of your group member projects as a starter for this project and create a private repository for it within GitHub.

    a. Choose one person to be the team lead. The team lead will need to be the person who will create the initial repository.

    b. Name your project and the repository: Group*X*CovidAnalysis, where *X* is your group letter.

    c. Make sure each group member has access to the repository.

    d. Share the GitHub repository with (yoder531) dyoder@westga.edu.

Getting started #2: Renaming application

1. Change the title of the application to: Covid-19 Analysis by *List the last name of each group member*

Getting started #3: Refactor and debug

1. Refactor your code to eliminate all class design and code smell items that have been discussed in class, presented in the book: *Clean Code*, the video: *Clean Code: Writing Code for Humans*, 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, io, 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.
   - Use appropriate constructs to query a data set, e.g., LINQ.
   - 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

2. Add functionality and remove bugs based on the requirements given in assignment 1, 2, and 3.

   a. Note: If your selected starter project had errors or missing functionality, those issues must be addressed in this submission.

   b. **You will lose points off the top of your grade if it still has missing functionality or bugs based on previous requirements.**

3. Complete the required unit testing from Assignment 3 so that each method being tested has **complete coverage**.

Functional requirements (15 pts.)

Requirement #1: Unit testing (1 pt.)

1. Add unit testing for five more methods so that there will be unit testing for a total of 10 model or statistic type methods – the five required in Assignment 3 and an additional five (5) for Assignment 4.

    a. Make sure the unit testing provides complete coverage for each method that testing.

    b. The methods selected for unit testing need to have some type of logic or perform some type of calculation. Unit tests for a method that is like a getter only do not count toward this requirement.

    c. At the top of the unit testing class file, list out the test cases that are implemented in the file.

2. Make sure to format the unit testing code using ReSharper, but you do not need to worry about using ReSharper warnings on files within the test project.

Requirement #2: Input validation and guide the user (2 pts.)

1. Add functionality so that the program prevents invalid input. This will include not allowing a user to enter text when a number is required, the number entered is in a valid range, cannot enter an invalid date, making sure the user enters a required value, etc.

2. Add functionality that enables and disables controls appropriately to help guide the user as the user interacts with the UI.

3. Make sure the user can cancel any action that has been started.

Requirement #3: Read, store, and analyze new data; modify summary output (2 pts.)

1. Modify the functionality so that the reading and writing of data will handle the **new** data format with a new column of data.

    New data format

    ```
    Data with each record on its own line.
    ```

    Example of the first few lines of data in a file:

    ```
    date,state,positiveIncrease,negativeIncrease,hospitalizedCurre
    ntly,hospitalizedIncrease,deathIncrease
    10/25/2020,AK,529,6320,58,0,0
    10/25/2020,AL,1079,5863,922,0,0
    10/25/2020,AR,797,7315,618,11,15
    ```

    Note the following:

    - The date is now specified in a new format.
    - Some columns are in a new order.
    - A new column of data exists.

2. When reading the data in, if the value is negative, it should be set to 0. If data is missing, it should be assumed to be 0. If it has text where a number is required or is invalid date then it should be considered an error line.

3. With the new data, modify the summary output so that is will also output the average current hospitalizations for the state and the max current hospitalizations and the date it occurred on for the selected state.

4. With the new data, modify the monthly output so that it will display the max and min current hospitalizations for the month and the day on which it occurred.

Requirement #4: Specific state (1 pt.)

1. Add functionality so that the user can specify what state of data to display. The user must be able to select this in a dropdown box.

Requirement #5: XML file format (2 pts.)

1. Add functionality to the application so that the user can also save and read the covid data as an XML file. After this, the two file formats the application will support will be CSV and XML.

    a. You must use XML serialization for this requirement.

    b. You may assume that first time the program loads a file it will be a CSV file, and then you should be able to save and load an XML file in the format that your program can handle.

Requirement #6: List and detail view (3 pts.)

1. Modify the UI so that it will have a list of all the days of data loaded for the selected state and a corresponding detail view where the user can view and edit an individual day that is selected.

    a. This must be implemented using data binding.
    b. Each list item must display the date and the number of positive tests for that day.

2. There should still be a way for the user to view a summary of the covid data that has the overall summary and monthly breakdown displayed. This doesn't have to all be done on the same page. This could be down by displaying the summary data on a separate page that the user can view through button invocation.

Requirement #7: Edit and delete day (2 pts.)

1. Add functionality so that the user can edit or delete the selected day.

Requirement #8: MVVM (2 pts.)

1. Use the MVVM design pattern with the data binding for the implementation of the list and detail view of the covid data.

End of functional requirements

Github and version control requirements

1. Each team member must make at least five commits.

2. Within the repository there must be at least one merge within the repository. If you are on a team of three (3), there needs to be at least two merges where each team member has code that is involved in the merge.

3. In the Wiki do the following:

    a. Have one team member give a quick overview of the project and include a picture of the application running.

        i. A short summary will suffice.

    b. Have another team member add links to the four assignment specifications for each iteration of this project.

    c. Add a completion section that states each individual item that was successfully completed for this iteration.

    d. Make sure your team uses appropriate headings within the wiki to offset the required sections.

4. In *Issue Tracking* do the following:

    a. Setup issues for any bugs and clean code refactoring that need to be addressed within the "starter" project.

    b. Setup issues for each of the additional requirements for the project.

    c. As a team assign specific issues to each team member.

    d. Make sure each team member resolves the issues within Issue Tracking as they are fixed.

    e. Any outstanding issues must be noted as unresolved issues within Issue Tracking.

5. Once you have completed the assignment tag the completed assignment changeset with the following label Assignment4Submission.

Moodle submission requirements

In Moodle, each team member needs to do the following:

1. Specify an estimate of the percentage of work each team member did, e.g., John – 40%, Sally – 60%

2. State what you specifically implemented in the project.

3. Give a statement or two summarizing your team member's contribution.

4. Assign yourself and your team member(s) a grade on a 0-100 scale.

   Note: Your team member(s) will not see any feedback that you provide in Moodle.

Have the team lead zip the tagged version and submit your completed project to Moodle by the due date. Verify the name of the exported file is: *GroupX*Covid19Analysis.zip. Note: I will actually clone the project from GitHub, but I want to make sure I have a zipped version of the project.

Grading breakdown – Total 30 points.

Any program that does not compile will receive a 0. Partial credit is not possible for any program that does not compile.

- 15 pts. – Functionality
- 10 pts. – Implementation and UI design
- 3 pts. – Use of version control, wiki and issue tracking.
- 2 pts. – Group contribution based on commits in GitHub and evaluation from team member(s).
    - **If individually you do not contribute any work on this project, you will receive a 0 on this assignment regardless of what the rest of your group receives.**

No rubric will be used for the functionality scores. It will be graded on a continuous scale.

The implementation rubric is on the next page.

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.
GUI design	2 pts.	1 pt.	NA	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.
GUI Design	The GUI follows standard design practices; has a good flow and layout.	GUI follows most, but not all, best practices, but there are some flow and design issues.		GUI interface provides functionality, but flow and design are not well thought out.