Cpt S 321 – Exam 2
Spring 2021
25 points total
Total pages: 4

Name (First Last):	Ryan Rochleau
WSU ID:	011577056

Read the instructions carefully:

- This is an individual exam: you are **not allowed to communicate** with anyone regarding the exam questions.
- If something is unclear, ask for clarifications via the BB Discussion (no code allowed). If it is still unclear after my answer, then write down any assumptions that you are making.
- Make sure you download your midterm code in a clean directory to ensure that it works.
- No late submissions are allowed.
- **What to submit (failure the follow the submission instructions below will result in receiving 0 automatically for the exam):**

- Commit a .PDF version of this document with your answers and your code in your repository for the in-class exercises in a branch "MidTerm2". Tag what you want to be graded with "MT2_DONE" tag.
- Your GitLab readme file should contain a summary of the features that you implemented and the ones that are missing.
- Also commit in that same repository a short video capturing your screen where 1) you show us how you download your code from the GitLab repository in a clean directory, and 2) you execute

your application from that clean download and you show us that it runs.

Please read the entire question carefully before you start working.

You are contacted by a company to build an application that will allow users to create shapes based on an input sequence of characters and then manipulate them. As a start, the application should support three shapes: circles, squares, and rectangles. More will definitely come later so your design should allow easily adding those.

For example, the sequence "c s c r" corresponds to the user wanting to create a circle, a square, a circle, and a rectangle – the order matters and you will see why. The first shape will always be created with a default size (of your choice but can be changed by the user). The next shape in the sequence will be created by doubling the default size. The third shape – by tripling the default size, etc. The sequence of shapes entered by the used should be stored in an appropriate data structure. The user can manipulate the shapes in different ways.

In the prototype that you are implementing, the user should be able to:

- Change default size for shapes.
- List the shapes that the user has created with all their associate information (ex., type, size, character).
- View the history of the shape creation sequences. In other words, list all sequences of shapes that she/he has created since the application is running.
- Adding a sequence of shapes. This means that the sequence should be added to the history and the shapes should be created.
- Alter the history by deleting or modifying a sequence in the history. The created shapes need to be updated accordingly.
- Compute the area of a sequence of shapes, which is the cumulative area of all existing shapes.
- Filter shapes on specific criteria, such as the area of the shape. For example, the user might want to know what are all shapes whose area is less than or equal to a certain threshold. (Hint: you must use LINQ and lambda expressions to do

this.)

Using TDD, implement a prototype in C# that would fit the needs of the client as described above. Note: You do not have to worry about a Graphical User Interface for now. You can build a prototype in a console application (Choose "Console App (.NET Core)" or "Console App (.NET Framework)" when you are creating the project) to demonstrate the required functionalities. **NOTE: Tag with "TDD" the commits that contain tests before implementing the functionality.**

Grading schema and point breakdown (25 points total):

- 10 points: Fulfill all the requirements above with no inaccuracies in the output and no crashes.
- 3 points: For a "healthy" version control history, i.e., 1) the prototype should be built iteratively, 2) every commit should be a cohesive functionality, 3) the commit message should concisely describe what is being committed, 4) you should follow TDD – i.e., write and commit tests first and then implement and commit the functionality.
- 4 points: Code is clean, efficient and well organized.
- 2 points: Quality of identifiers.
- 2 points: Existence and quality of comments.
- 2 points: Existence and quality of test cases. Normal cases and edge cases are both important to test.
- 1 point: summary of the features implemented and features missing (if any).
- 1 point: Video showing that you run the application from a clean download.

General Homework Requirements	
Quality of Version Control	• Should be built iteratively (i.e., one feature at a time, not in one huge commit). • Each commit should have cohesive functionality. • Commit messages should concisely describe what is being committed. • TDD should be used (i.e, write and commit tests first and then implement and commit functionality). • Use of a .gitignore. • Commenting is done alongside with the code (i.e, there is commenting added in each commit, not done all at once at the end).

Quality of Code	• Each file should only contain one public class. • Correct use of access modifiers. • Classes are cohesive. • Namespaces make sense. • Code is easy to follow. • StyleCop is installed and configured correctly for all projects in the solution and all warnings are resolved. If any warnings are suppressed, a good reason must be provided. • Use of appropriate design patterns and software principles seen in class.
Quality of Identifiers	• No underscores in names of classes, attributes, and properties. • No numbers in names of classes or tests. • Identifiers should be descriptive. • Project names should make sense. • Class names and method names use PascalCasing. • Method arguments and local variables use camelCasing. • No Linguistic Antipatterns or Lexicon Bad Smells.
Existence and Quality of Comments	• Every method, attribute, type, and test case has a comment block with a minimum of <summary>, <returns>, <param>, and <exception> filled in as applicable. • All comment blocks use the format that is generated when typing "///" on the line above each entity. • There is useful inline commenting in addition to comment blocks that explains how the algorithm is implemented.
Existence and Quality of Tests	• Normal, boundary, and overflow/error cases should be tested for each feature. • Test cases should be modularized (i.e, you should have a separate test case for each feature or scenario that you test - do not combine them into one large test case).