Cpt S 321 – Final Exam
Spring 2021
30 points total
Total pages: 4

Name (First Last):	Lucas Da Silva
WSU ID:	11631988

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):

1. Commit a .PDF version of this document with your answers and your code in your repository for the in-class exercises in the same branch as your second mid-term exam. Tag what you want to be graded with "FinalExam_DONE" tag. If you mistakenly committed in the wrong repository you must inform us prior to the deadline.
2. Your GitLab readme file should be updated to contain a summary of the features that you implemented for this exam and the ones that are missing.
3. In your GitLab readme file, share a link to 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.

The company that you were contacted by for the shape application came back to you. They were pleased with the prototype you built (how can they not be?!?!?) but they have a few changes that they would like you to make. The client listed the changes as follows:

1. The application should support two additional shapes: trapezium (to be used with the character “t”) and pentagon (with the character “p”).
2. Shapes can have a color and a border. Borders can have different thickness and pattern (solid line, dotted line, dashed line, etc.). Both color and shape should have default values of your choice, but the user should be able to change them for each instance of a shape. For example, the user might want to have a bleu and a red trapezium.
3. The user should be able to filter shapes on color and on thickness.
4. The application should allow the user to undo/redo all operations since the application has been last started. This means that there is no need to think about making the undo/redo stack persistent.
5. The application should be able to save and load data from an XML file.

Your main menu for the user interface must support the following options as shown here:

1. Change Default Shape Size
2. List Created Shapes
3. View Shape Creation History
4. Add a Sequence
5. Delete a Sequence
6. Modify a Sequence
7. Compute Total Area of a Sequence
8. Filter Shapes Based on Criteria
9. Undo
10. Redo
11. Save to XML File
12. Load from XML File

The design of the sub-menus to carry out each of these tasks are up to you, but they must be intuitive. For example, we would expect that when a user chooses option 6, Modify a Sequence, they are first prompted to select a sequence to modify and are then presented with the modification options (i.e., adding a shape, deleting a shape, changing a shape's type, changing a shape's color, changing a shape's border thickness, changing a shape's border pattern).

Using TDD, build on your previous prototype and adapt it to the needs of the client as described above. This implies that you must have your mid-term 2 completed before you start working on this exam. Note: You do not have to worry about a Graphical User Interface. **NOTE: Use the keyword "TDD" in the commits that contain tests before implementing the functionality.**

Grading schema and point breakdown (25 points total):

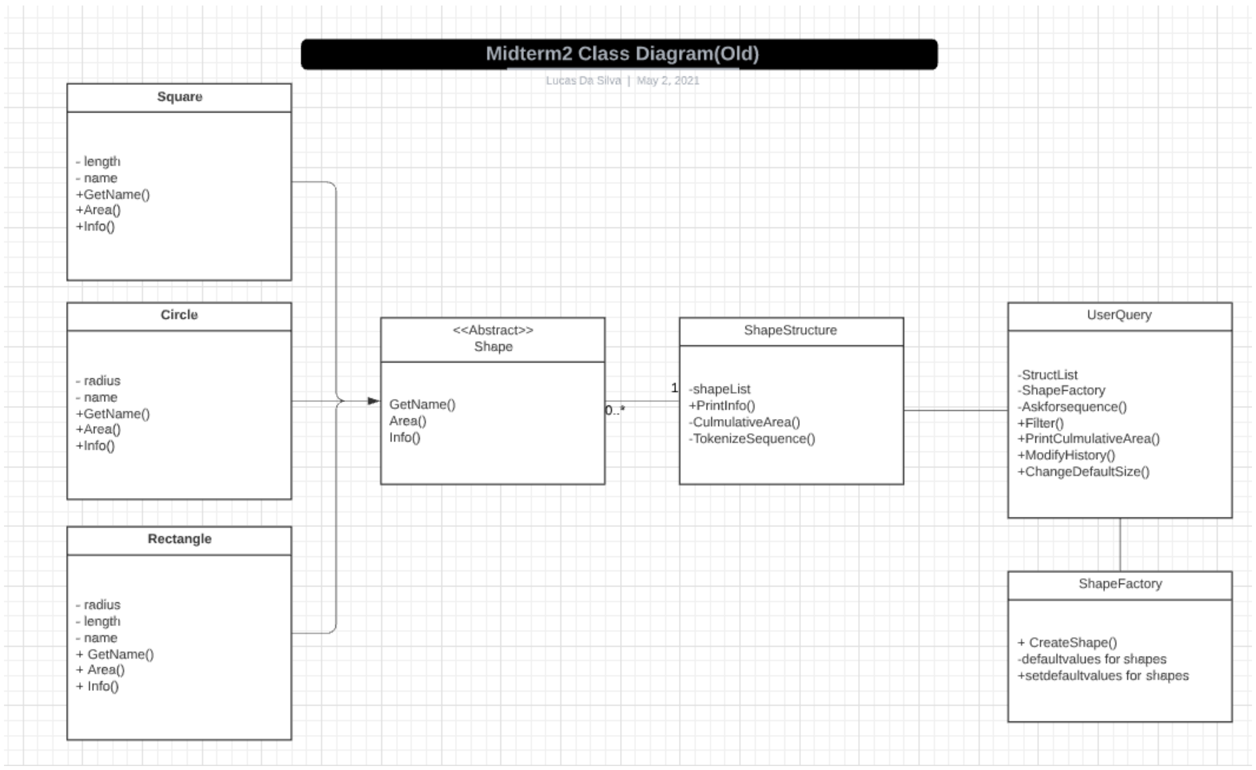
- **10 points**: Fulfill all the requirements above with no inaccuracies in the output and no crashes.
- **5 points**: A design document (you can use the end of this word document for this) showing 1) a class diagram that represents your project before you start (i.e., what you had for mid-term two), 2) a summary of how you plan to implement the requested features by describing the changes you need to make on your design and any rationale behind the choices you make, 3) a class diagram that represents your project at the end, i.e., after your final commit.
- **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).

DESIGN DOCUMENT:

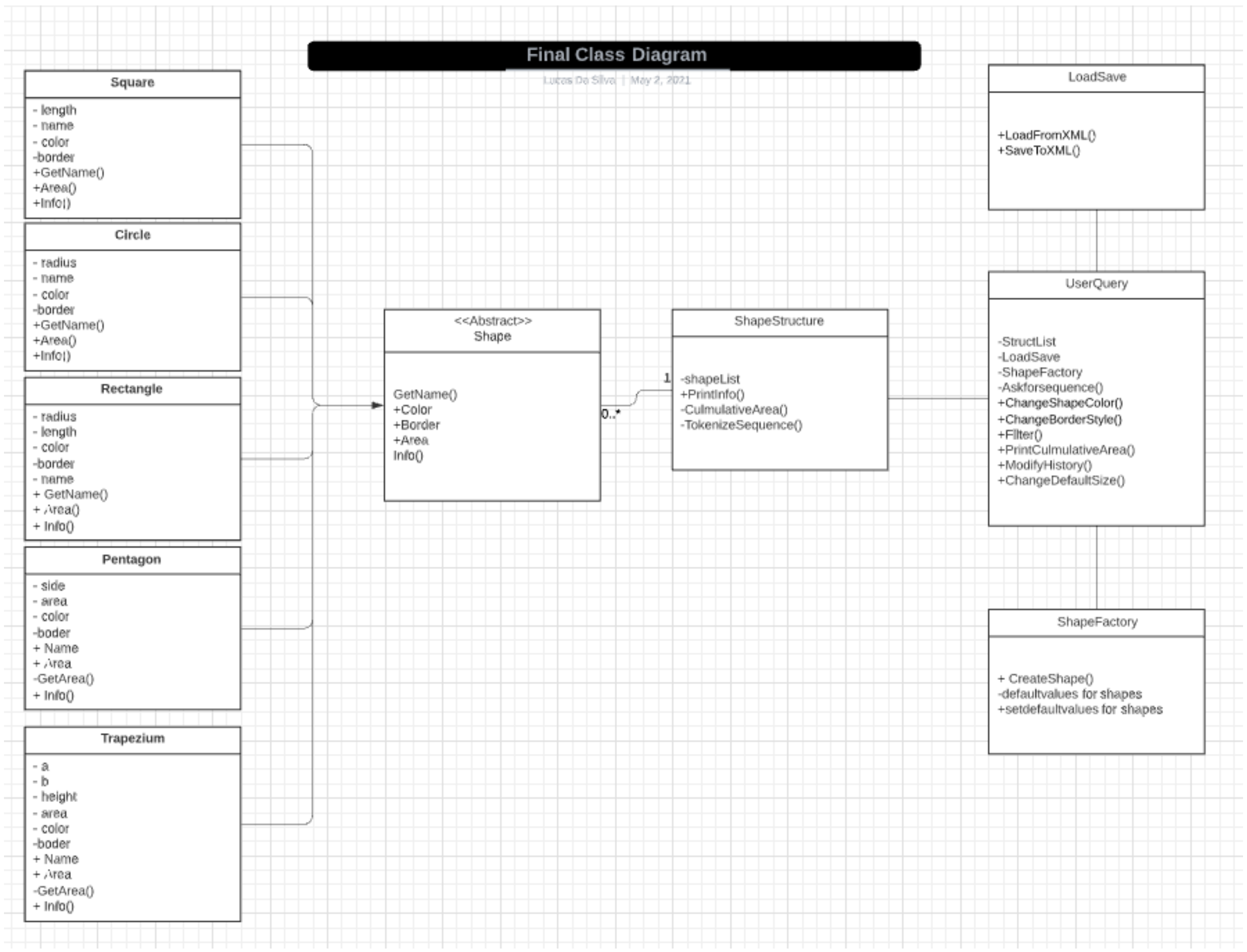
1) **a class diagram that represents your project before you start (i.e., what you had for mid-term two)**



2) **a summary of how you plan to implement the requested features by describing the changes you need to make on your design and any rationale behind the choices you make**

   a. To add the extra shapes, ill simply just create new shape classes and make them inherit the Shape class
   b. To add the addition of color and border, I will have to add Color and Border property to the Shape class itself, and change that for all the other classes that inherit from Shape
   c. To do the load and save feature, I think I will simply have it be in a different class and call it from the user query class

3) a class diagram that represents your project at the end, i.e., after your final commit.


Final Class Diagram
Lucas Da Silva | May 2, 2021
Square
- length
- name
- color
-border
+GetName()
+Area()
+Info()
Circle
- radius
- name
- color
-border
+GetName()
+Area()
+Info()
Rectangle
- radius
- length
- color
-border
- name
+ GetName()
+ Area()
+ Info()
Pentagon
- side
- area
- color
-boder
+ Name
+ Area
-GetArea()
+ Info()
Trapezium
- a
- b
- height
- area
- color
-boder
+ Name
+ Area
-GetArea()
+ Info()
<<Abstract>>
Shape
GetName()
+Color
+Border
+Area
Info()
0..*
1
ShapeStructure
-shapeList
+PrintInfo()
-CulmulativeArea()
-TokenizeSequence()
LoadSave
+LoadFromXML()
+SaveToXML()
UserQuery
-StructList
-LoadSave
-ShapeFactory
-Askforsequence()
+ChangeShapeColor()
+ChangeBorderStyle()
+Filter()
+PrintCulmulativeArea()
+ModifyHistory()
+ChangeDefaultSize()
ShapeFactory
+ CreateShape()
-defaultvalues for shapes
+setdefaultvalues for shapes