Cpt S 321 – Final Exam
Fall 2023
30 points total
Total pages: 4

First name:	
Last name:	
WSU ID:	

Read the instructions carefully until the end before asking questions:

- 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 Canvas Discussion (no code allowed). If it is still unclear after my answer, then write down any assumptions that you are making.
- At the end, make sure you download your exam code in a clean directory to ensure that it works.
- No late submissions are allowed.
- **What to submit (failure to follow the submission instructions below will result in receiving 0 automatically for the exam):**

1. In your **in-class exercises** GitLab repository **create a new branch called** "FinalExam" and commit the following:
   - i) a .PDF version of your class diagram. The design document must be placed in a folder "Design_Documents" at the root of your project and the name must contain your name and a description of the diagram (ex.: "Venera_Arnaoudova-final-ClassDiagram.pdf").
   - ii) your code.

   Tag what you want to be graded with "FinalExam_DONE" tag.
2. Your GitLab readme file must:
   - explain what features you implemented for the exam and the ones that you are missing.
   - provide a link to a 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 the different features that you implemented.
3. Answer questions 1, 2.1, 2.2, and 2.3 in this document and submit it via Canvas as a **.PDF file**. Name it in the following format: "**FirstName_LastName-final-answers.pdf**"

Q1. (5 points) Consider the GRASP, SOLID, and Design Patterns (DP) that we have learned in class. Select 2 patterns that belong to different categories (ex. one DP and 1 SOLID principle) and illustrate how the two different patterns can work towards the same high-level goal. Also discuss the similarities and differences of how the two patterns achieve that goal.

Pattern 1: Category:

Pattern 2: Category:

High-level goal of the two patterns:

Similarities regarding how the two patterns achieve that goal:

Differences regarding how the two patterns achieve that goal:

Q2. (25 points) There are 3 sub-questions following the description of the problem.

You are contacted by a Banking company to build a desktop application in C# that will allow their clients and employees to connect to the online banking system. The features that the company currently needs are as follows:

- Authenticate: users (clients and employees) should be able to login before any of the other features below are made available to them. For authentication, the company will try several third party libraries and will select from those what will be used. This means that your design should accommodate that and that you do not need to implement the actual authentication.
- Check status: users should be able to check the status of all their accounts. Saving accounts have interests that clients gain as well as a minimum amount that should be available on the account. Checking the status consists of:
    - Checking accounts: show the account number, current balance and the last 5 transactions.

    - Saving accounts: show the account number, current balance, interest rate, total amount of interest gained for the year, and the last 5 transactions on the account.
- Transfer funds: users should be able to transfer money from one account to another (assume for now that only accounts from this bank can be used for transfer). A user can transfer funds from one of their own accounts to either their own or someone else's account. Of course, the transfer is valid and should happen only if sufficient funds are available. Thus, the transaction consists of 3 actions: 1) check for available sufficient funds, 2) withdraw the funds from one account, and 3) add funds into the receiving account. The user should be able to undo the last transaction within the next 5 minutes of placing the transaction (i.e., within 5 minutes after clicking on the transfer funds button). Employees, can cancel the transaction within the first 24h.

You need to design and implement a prototype with a GUI using C# and WinForms.

Q.2.1: Fill the table below indicating all principles/patterns/good practices that we learned about in class that you plan to use for your design.

Principle/Pattern name (type) Ex.: "Polymorphism (GRASP)"	**Classes that are involved** Ex.: "Animal, Cat, Dog"	**Additional comment** Ex.: Cat and Dog inherit from Animal

Q.2.2: Draw the class diagram for your design to show how different classes are connected. Also highlight any additional design choices that you have made that were not discussed in the previous questions by annotating your diagram with comments. Use draw.io to make the class diagram, export it as a **.PDF file (or a .PNG)**, and include it in your submission. Name the file in

the following format: "**FirstName_LastName-diagram.pdf**". Provide a direct link to your class diagram here:

Q.2.3: Implement your design using best coding practices in C# in and provide a link to your GitLab tag (make sure me and the TAs are maintainers) here:

Grading schema and point breakdown for Q2 (25 points total):

- **8 points**: The design is easy to maintain and extend and make use of good design principles and practices. Both questions Q2.1 and Q2.2 will be used to evaluate the design. A design document showing a class diagram that represents your design should be present in your repository. Feel free to add other types of diagrams if need be. Feel free to use any software (such as draw.io) to make the class diagram – don't forget to export it as a .PDF to include in your submission.
- **4 points**: Your software fulfills 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, and 3) the commit message should concisely describe what is being committed.
- **3 points**: Code is clean, efficient and well organized. This includes identifying opportunities for refactoring your code and applying those refactorings. Make sure that the refactoring commits clearly describe the type of refactoring that is applied.
- **2 points**: Quality of identifiers.
- **2 points**: Existence and quality of comments.
- **3 points**: Existence and quality of test cases. Normal cases and edge cases are both important to test.

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. • 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).