HW5 – Putting Principles, Practices, and Patterns to Work

Estimated time: 18-24 hours

Objectives

- Use all the principles, practices, and patterns learned thus far to improve an existing system.
- Learn to recognize and avoid common pitfalls.

Overview

For this assignment, you will select one of following as a starting point: 1) what you built for HW2, HW3, or HW4, 2) a working system that you built for some other class, or 3) something that you built for a work assignment. Then, you will improve the functionality and quality of that system by

- Adding at least one **significant** feature.
- Improving the abstraction, modularity, and encapsulation of its design and implementation.
- Applying design patterns not previously used in that system, that are appropriate and whose application would improve the quality of the design and implementation.
- Making the unit testing more thorough.

Even though you need to add some functionality, don't go overboard with new functionality. Do not try to add too many new features or too big of a feature. The primary purpose of the assignment is to add one significant feature and to improve the quality of an existing system in meaningful and defendable ways. Limit your time to 24 hours.

Instructions

1. Decide which previous system you want to improve. Save a snapshot of that system in an "old" directory or zip file so we can compare the changes during the review.
2. Review your old design and implementation.
3. Identify areas that need improvement. Look for common pitfalls that you (or the developer, if it wasn't you) may have fallen into.
4. Refactor the design towards design patterns that can help you improve the system's abstraction, modularity, and encapsulation.
5. Don't use a design pattern just for the sake of doing something different. Any refactoring toward a design pattern must be justified.
6. Look for interesting opportunities to extend the functionality of the system without dramatically increasing its complexity.

7. Update (or create) the design document (UML **Class**, **Interaction**, and **State Chart** diagrams), so they accurately communicate the design of the system.
8. Implement your changes.
9. Test all non-GUI components **thoroughly**.
10. Create a meaningful README file
11. Write up a short report that explains your improvements and your rationale for each one, as well as your design and insights.

IMPORTANT: You are responsible for choosing the scope this assignment. Choose wisely. Limit your time to 24 hours of meaningful effort.

Submission Instructions

Zip up your entire solution, including test cases and sample input files, in an archive file called CS5700_hw5_<*fullname*>.zip, where fullname is your first and last names. Then, submit the zip file to the Canvas system.

Grading Criteria

Criteria	Points
A short report that introduces your project, discusses your improvements, their rationale, your design, and insights. The report should also include your class diagrams.	0 - 10
A clear and concise design, described in UML Class Diagrams, Interaction Diagrams, and State Charts that 1) reflects good abstraction, modularity, and encapsulation, 2) includes the appropriate application of patterns, and 3) will guide the construction of the software such that all functional requirements will be met.	0 - 40
A working implementation that is true to the design	0 - 40
Meaningful, executable unit test cases for not UI components that provide good coverage of critical components from a path-testing perspective.	0 - 30
Deductions (apply top of scores given above): • Poor modularity • Poor abstraction • Poor encapsulation • Violation of best practices discuss in class to date • Missed opportunities to apply appropriate design patterns • Late (up to -36 points) • Did not complete a code review – an automatic -120. • Cheating – copying someone else's work or 3rd party code (in partial or full), without giving attribution to the source. This will be an automatic -240 – twice the	-240 - 0

maximum points.	