HW2 – Encapsulation, Observer, Template Method, and More

Estimated time: 14-20 hours

Objectives

- Become familiar with the principle of encapsulation.
- Become more familiar with the principles of abstraction and modularity.
- Become familiar with the application of more patterns, including Template Method, Observer, and Factory Method.
- Become familiar with using UML Interaction Diagrams to communicate inter-object behaviors.
- Become more familiar with using UML Class Diagrams to communicate the structural aspects of a system's design.
- Become more familiar with unit testing techniques.

Overview

For this assignment, you will enhance your Drone Flier and implement a Simulator to replace the initial simulator provided with HW1. You will have some freedom to decide which specific enhancements you make to your Drone Flier.

As you design and implement your new Drone Flier and your own Simulator, you will need to pay particular attention to the principle encapsulation, as well as look for opportunities to improve your modularity and abstraction. You will also need the Template Method Pattern and either the Observer or Factory Method or both.

Drone Flier Enhancements

Table 1 lists possible enhancements for the Drone Flier. You need to implement all those that have an "M" in the *Value* column and choose from the others a subset such that the total value is equal to or greater than 7. Do not try to implement all the possible enhancements.

#	Enhancement	Concepts that could be applied	Value
1	Implement the tracking of periodic status messages from the drone, so the flier always has a relatively up-to-date snapshot of the drone's state.	Intra-process concurrency using threads or asynchronous event handling	M
2	Use drone state information to improve the execution of missions.	The AME Principles	M
3	Implement an internal estimation of the drone's position based on completed maneuvers.	Strategy Pattern	M
4	Implement a graphic user interface that displays the drone's state, a front view of the drone's position, and a side view of its position.	Observer Pattern, GUI development	3
5	Allow missions to be loaded from files, which can be in one of several different formats (e.g., CSV, JSON, and XML).	Strategy Pattern, Template Method Pattern	2

6	Implement potential configuration values (e.g., port numbers, timeouts, maximum retries, etc.) so they can be easily changed at runtime.	Settings or Property files, Command-argument parsing	1
7	Allow missions to include actions that do not involve drone communications, e.g., the displaying a message to the user, input from the user, waiting for some state-related condition to be true, etc.	Strategy Pattern, Template Method Pattern	2
8	Allow other developers to easily add new types of mission actions, without requiring them to change your code.	Strategy Pattern, Template Method Pattern, and maybe the Factory Method Pattern	1
9	Ensure that your component responsible for network communications can be used by both the Drone Flier and Simulator.	The AME principles, class libraries, dependencies, and build instructions	1
10	Ensure that your components responsible for message serialization, de-serialization, and validation can be used by both the Drone Flier and Simulator.	The AME principles, the Factory Method Pattern, class libraries, dependencies, and build instructions	1
11	Ensure that your component responsible for drone state can be used by both the Drone Flier and Simulator.	The AME principles, class libraries, dependencies, and build instructions	1

The Requirements below contains more details about each of the above items.

Simulator

Your Simulator will need to do all that the original Simulator did, plus manage the state of the simulated drone and send a periodic status message the Drone Flier (on port 8890). Some incomplete sample code will be provided to you as a starting point for the Simulator. Some of this code may not represent good abstraction, modularization, or encapsulation – feel free to adapt and improve it as you see necessary. The purpose of sample code is simply to illustrate at least one technique working with threads and to provide a starting for a Drone State component.

Instructions

To complete this assignment, you should do the following:

1. Create a new Git repository of HW2.
2. Study the requirements the Drone Flier enhancements listed in Table 1 and for the Simulator.
3. Decide which enhancements you want to make to your Drone Flier.
4. Determine how to change your HW1 design and update your design diagrams accordingly.
    a. Think seriously about refactoring out components that could be reused in the Simulator into their own project so they can be easily integrated into both the Drone Fly and Simulator.
    b. Use both Class Diagrams and Interaction Diagrams to document your design.
5. Implement and test your new Drone Flier.
    a. You do not need to write executable unit-test cases for user-interface components.
6. Test your program with a real drone (optional).
7. Design your own Simulator, using the sample code for ideas as you see fit.
    a. Use both Class Diagrams and Interaction Diagrams to document your design.

8. Implement and test your Simulator.
9. Commit and push all changes to your Git repository frequently.
10. Write up a short report that includes your design diagrams and insights or lessons learned. Not counting the diagrams, the report should be **no more than 500 words**. Save your report as PDF file, called HW2-Report.pdf, in the root directory your Git repository. Commit and push.
11. Be prepared to share your repository, if asked or if you would like assistance from the instructor or TA.
12. Zip your Git repository and submit it to Canvas by the due date/time.
13. Complete a code review with the instructor or TA.

Note: committing and pushing your git repository frequently not only provides you with a backup of your work but it will also document of a consistent effort over time. At any time during the assignment or the grading of the assignment, the instructor and TA may ask to see you commit log.

Requirements

1. Drone Flier Enhancement #1 (mandatory) – Implement the tracking of periodic status messages from the drone, which are sent 100 ms, so the flier always has a relative up-to-date snapshot of the drone's state.
    1.1. After entering command mode, the real drone or your Simulator will send status messages to port 8890 at the same IP Address that the command message was sent from.
    1.2. The Drone Flier should retrieve these messages, independent of whatever it else it might be doing, and use them to update its knowledge of the drone's state.
2. Drone Flier Enhancement #2 (mandatory) - Use drone state information to improve the execution of missions.
    2.1. The execution of a mission should be aborted and the drone should be landed, if the drone's battery drops below a very-low-battery threshold.
    2.2. A flip maneuver should be automatically converted to a left, right, forward, or back maneuver that moves the same distance, if the drone's battery drops below a low-battery threshold.
    2.3. The execution of a mission should be aborted and the drone should be landed, if the drone's temperature exceeds a high-temperature threshold.
3. Drone Flier Enhancement #3 (mandatory) – implement an internal estimation of the drone's position based on completed maneuvers.
    3.1. When executing a maneuver that changes the drone's position and orientation, calculate a new position and orientation after successful execution of that maneuver. See the sample code for the managing drone state for ideas on how to do that.
    3.2. Reset the drone's position and orientation at appropriate times, e.g., before takeoff or after landing.
4. Drone Flier Enhancement #4 (optional) – Implement a graphic user interface that displays drone's state, plus front and side views of the drone's position.
    4.1. The graphical user interface must show, in real-time, at least a side view and front view of the drone's location in a 3x3 meter box.

4.1.1. The initial position of the drone when it takes off is the center of the bottom of the box, with its orientation facing to the front of the box.

4.1.2. When the drone takes off, it moves straight up in the box.

4.1.3. From takeoff to landing, the side view must show the position of the drone in the box as if the user was looking through the left side of the box.

4.1.4. Also, from takeoff to landing, the front view must show the drone position as if the user was looking through the front of box.

4.1.5. If the drone moves out of the box, an error should be displayed.

4.1.6. If the drone moves back into the box, the error should be cleared.

4.2. The graphical user interface must show current state information for the drone.

4.3. The graphic user interface may allow the user to change certain configuration parameters, like drone address and port number – this is not essential for Enhancement #4.

4.4. The graphic user interface may allow the user to select a mission, start the mission, and abort a mission – this is not essential for Enhancement #4.

5. Drone Flier Enhancement #5 (optional) – Allow missions to be loaded from files, which can be in one of several different formats (e.g., CSV, JSON, or XML)

5.1. The Drone Flier should be able to load a mission from a file.

5.2. The loading feature must support at least two different file formats.

6. Drone Flier Enhancement #6 (optional) – Ensure potential configuration values (e.g., port numbers, time outs, maximum retries, etc.) are implemented so they can be easily changed at runtime.

6.1. All configuration values should have defaults defined in a convenient and maintainable way.

6.2. All configuration values should changeable through command-line arguments or user input.

7. Drone Flier Enhancement #7 (optional) – Allow missions to include actions that do not involve drone communications. Here are some ideas from which at least three must be implemented.

7.1. An action could cause certain information to be displayed to the user.

7.2. An action could ask the user to enter some value, like a distance to move.

7.3. An action could pause the mission until the drone's state satisfies some condition.

7.4. An action could pause the mission for a specific amount of time.

7.5. An action could be a branching statement based on some condition involving the drone's state.

7.5.1. The true (or false) branch could go to another action in the mission.

7.5.2. The true (or false) branch could start a new mission.

8. Drone Flier Enhancement #8 (optional) – Allow other developers to easily add new types of mission actions, without requiring them to change your code.

8.1. The Drone Fly must be designed with the Open/Closed principle in mind so other developers can extend the types of actions that can used in a mission.

9. Drone Flier Enhancement #9 (optional) – Ensure that the component responsible for network communications can be used by both the Drone Flier and Simulator.

9.1. The Flier (and Simulator) must be structured to use a common class library that contains a component that provides a convenient abstraction for send/receiving messages.

10. Drone Flier Enhancement #10 (optional) – Ensure that the component(s) responsible for message serialization, de-serialization, and validation can be used by both the Drone Flier and Simulator.

   10.1. The Flier (and Simulator) must be structured to use a common class library that contains components that provides convenient abstractions for messages.
11. Drone Flier Enhancement #11 (optional) – Ensure that the component responsible for drone state can be used by both the Drone Flier and Simulator.
   11.1. The Flier (and Simulator) must be structured to use a common class library that contains a component that provides a convenient abstraction for the drone's state.

Submission Instructions

Zip up your entire solution, including test cases and sample input files, in a zip archive file called CS5700_hw2_<*fullname*>.zip, where fullname is your first and last names. Then, submit the zip file to the Canvas system.

Grading Criteria

Criteria	**Points**
A clear and concise design, described in UML Class and Interaction Diagrams, 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 implement that is true to the design and meets all functional requirements.	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
A short report that discusses your design and insights that you uncovered during the design, implementation, or testing. The report should also include your class diagrams.	0 - 10
Deductions (apply top of scores given above): • Poor modularity • Poor abstraction • Violation of best practices discuss in class to date • No attempt to find an appropriate application of the Strategy pattern • 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 maximum points.	-240 - 0