Assignment 2

Start on 28 January, complete by Sunday, 6 February end of day Montreal time zone. Covers Chapter 2-3 of the book.

Problem Statement

Using the principles, mechanisms, and techniques seen in Chapter 2 and Chapter 3 of the book, design and write the code necessary to meet the following requirements. We have provided an incomplete code base as a starter. You are free to make any modification that you judge as necessary. *Your solution should continue to respect principles of good design seen in prior chapters.*

1. Update the baseline code to satisfy the following requirements. Each movement of the drone should specify the direction a drone can move, the speed at which it moves, and the distance it moves. The 6 possible directions include forward, backward, left, right, up, and down. The three possible speeds are low, moderate, and high, and the distance should be a positive value. Each movement can be either recorded or not recorded. When the recording function is turned on, the video can be saved as one of MP4, MOV, WMV, AVI, FLV, or MKV formats. [2 points]

NOTES:

- To simulate the move being executed, you can print statements to the console. For e.g., `System.out.println("Move forward ... distance at ... speed")`.
- Using print statements is only to simulate actual behaviour of the application that is out of scope of this assignment (such as some functionality or for obtaining input that would normally require a GUI). In general, printing to the console is not a means to interact with the client and should be used internally for development only.

2. Add the support for drone tricks in your design, which are composed of the specific combination of movements that a drone can perform. Every trick should have a name of the trick (e.g., *take off*, *land*, *pucker*, *spindive*, etc.) and a sequence of movements. For example, a *take off* trick can be described as fly up at low speed -> fly up at moderate speed; and a *pucker* trick can be described as fly up -> turn left -> turn left -> turn left -> turn left -> fly down (at any speed and for any distance). You can refer to more tricks here. For the scope of this assignment, only movements defined in Problem Statement 1 are considered. The client should be able to configure the trick to be recorded or not. Please implement the **take off** and **pucker** trick examples and one more trick of your choice. [5 points]
3. Add a `Flight` class to your program. The flight class should allow a client to group a series of tricks into a flight. It should also provide a way to allow a client to query the number of unique movement directions (i.e., forward, backward, left, etc.) in a `Flight` object. [5 points]
4. Allow a client to compare different flights either based on the number of tricks it contains, or the number of unique movements, in ascending order. To illustrate your design, represent three different flights in your `RunDrone.java` file, and sort them using different comparison

methods. [5 points]
5. Use **class** diagram to illustrate the key design decisions in this assignment. [3 points]

Deliverables

1. The submission must include a zip file of the following:
   1. The source code of your solution in a `src` folder; It should include a `RunDrone.java` file (located in the `src` folder) with a `main` method that exercises the main scenarios of the code;
   2. A pdf file of a short description (max. 200 words) of the design of your solution (major decisions, techniques used, trade-offs, etc.), supported by UML **class** diagram created with JetUML or other tools.
2. Submit your work on MyCourses -> Assignment -> Assignment2.

Rules of Carrying Out Assignment

- The focus of the assignment is to evaluate your application of design techniques while respecting all the requirements in the problem statement. To this end, you are expected to come up with an original design when the requirements don't specify a particular design.
- The goal of this assignment is to arrive at a solution having explored the design space and to understand the trade-offs.
- Remember that there's no single optimal solution. However, you should be able to justify the design choices you made during the assignment both by your code and your description file.
- Try to think of different use cases for your code, and how your solution could accommodate them. Writing your own client code can help identify design issues. For example, you can use real world scenarios to approach the problem statements. However, you don't need to implement features other than those in the problem statement, and you are not expected to implement a perfect solution for all real world scenarios.

Assessment

The TA evaluation of your solution will focus on three aspects:

1. How well does the delivered code satisfy the requirements in the problem statement?
2. How well are the design concepts, principles, and techniques discussed so far are applied and explained in the description file (including the trade-offs of different options)?
3. How readable is the source code (through its style and documentation)?

Policy on Code Reuse

1. All design activities are prepared so as to be completed without any need to reuse external code; **"External code" means any code not part of the course samples or Java/JavaFX class libraries;** If you are pining for the a perfect score, please note that using external code will not give you any kind of competitive advantage to help you reach this level.
2. You can reuse external code only for implementation of functionalities that clearly go beyond the requirements; All major design decisions required to fulfill the requirements should be yours and yours only. In case of doubt either refrain from using external code or clarify the situation with your teaching assistant (TA). If you reuse external code it is your responsibility to ensure it does not take away from your required contributions.
3. If you reuse external code it is your responsibility to understand how it works; External code included with a solution is considered part and parcel of the solution: any bug or design flaw caused by external code will be considered a problem with the solution.
4. If you reuse external code you must clearly identify it and locate it in a separate package named `external`, and code files must clearly bear the origin of the external code.
5. If you reuse external code you are responsible for looking up its license and respecting it (in particular any attribution clause).
6. If you reuse external code you are responsible for documenting how it works to the extent where it integrates with your solution. The TAs can be assumed to have knowledge of the Java class libraries, but not external code.
7. Use of external code must be credited anywhere it appears. For example, if you paste a code fragment from Stack Overflow into a comment, this needs to be credited even if it might not appear in the final solution.
8. Any unaccredited contribution of external code will be considered a breach of academic integrity. In brief, reusing code is a big responsibility, not to be taken lightly. Please read the article Surviving Software Dependencies for more insights.