CS 350 Task 3.*x*: Project Proof of Concept Component

This task serves as a proof of concept for the key component in the larger architecture to come, which we presumably identified after Tasks 1 and 2. The specification resides in the provided Javadoc. This document is an overview of the component and the process of creating it.

Overview

This component provides basic creational, structural, and behavioral capabilities on a single simplistic airplane:

- Class `Airplane` defines a airplane that can change its position in three-dimensional space.
- Class `Mover` defines the mechanism for changing the state of an airplane.

Part I: Questions

Submit this part to the Task 3.*x* Pre Due link in a single plain text document.

Read and attempt to understand this document and the Javadoc with respect to what you know about the project at this point. State any questions you have. There is no format except for a blank line between questions. If you have no questions, submit a statement to this effect.

Part II: Requirements and Applicability

Submit this part to the Task 3.*x* Post Due link in a single plain text document with the given headings. It is a separate document from Part I.

Inferred Requirements

Use the Javadoc to reverse engineer the data and control back to the behavior it enables. You do not have to be formal in the requirements (e.g., cross-reference indexing or use of *shall*), but be sure to produce a list of everything that the Javadoc would check off as the corresponding solution. Do not just list the methods. Remember, in the forward engineering, requirements do not state such things or in such form.

Inferred Applicability

Review the 26 elements from Task 1 and list by name those that would directly benefit from the capabilities provided here. In other words, in place of `Airplane`, which other classes would be appropriate with `Mover` (give or take; it need not correspond exactly). For each, briefly indicate why.

Part III: Implementation

Implement the solution *exactly as specified*. Submissions that fail to compile for any reason receive no credit.

If any part of the *read-understand-plan-execute-verify-reflect* process is broken, identify and try to fix it. The complexity of this solution at a 210 CS level and high school math level. You are responsible for this prerequisite material. There are understandable and expected reasons for not getting all the correct results in your first attempt, but there are no legitimate reasons for it not to compile or run, at least.

The line counts from my solution are in square brackets for general reference. Your solution may differ, but not substantially.

Submit your two Java source files to the Task 3.*x* Actual Due link individually, not zipped.