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. Lecture 23 provides an overview. The specification resides in the provided Javadoc.

Overview

This component provides basic creational, structural, and behavioral capabilities for modeling a simplistic aircraft:

- Class `Aircraft` defines an aircraft that can change its position and orientation in three-dimensional space.
- Class `State` defines the mechanism for changing the state of the aircraft.

Part I: Questions

This part helps you clarify any uncertainty you may have about the specifications. Your questions need to be appropriate for a subject-matter expert or the customer; e.g., do not ask when the task is due or how many points it is worth.

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 Testing Plan

Inferred Requirements

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

For example, from the fictitious method `getTime()`, you could infer a requirement that there needs to be way to get the current time (assuming this is what the Javadoc states it does). Most of these requirements will be very straightforward because a single requirement translates to a single unit of code structure like a method (and vice versa). Pay special attention to those that are not straightforward. If you cannot determine what it satisfies, then you do not actually understand what it does.

Testing Plan

For each class, state for each method in the order of the Javadoc how you would verify that your solution is correct. Keep this general. Do not specify exact values or conditions unless it is unavoidable. Rather, explain in a sentence or two what would guide your actual test.

For example, for `getTime()`, you could state that the result should be the same value that you provided to `setTime()`, which could likewise be tested with `getTime()`.

Part III: Implementation

Implement the solution *exactly as specified*. Submissions that fail to compile for any reason will 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 CS 210 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 on your first attempt, but there are no legitimate reasons for it not to compile or run, at least. Attention to detail is critical. Compilers show no mercy.

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

Deliverables

Submit Parts I and II to the Task 3.*x* Pre Due link in a single plain text document.

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