1. What type of a design pattern (creational, structural, behavioural, custom) did you employ in this assignment? Why?

We chose the structural pattern, **the Adapter Design Pattern**. This pattern was necessary because each vehicle class (such as Boat, Car, and Airplane) had its own interface that was incompatible with the others. The Adapter pattern allowed us to create a unified way of interacting with different vehicle types without modifying the existing code. This was critical for enabling the simulation to function cohesively and ensure smooth integration of legacy vehicle classes into a consistent interface.

1. Describe a software design pattern that you have used before this course, even if you were not aware of it.

One of our team members once tried to implement Redis. Although not fully aware of it at the time, the Singleton pattern was likely used. Redis is typically accessed through a single instance of the client across the entire application to ensure efficient resource usage and connection management. By using a Singleton for Redis, it ensures that only one instance of the Redis client exists throughout the application, optimizing memory and network connections.

1. Do you feel standard design patterns are of great assistance, or great hindrance, in software design? Give advantages and disadvantages.

Advantages:

* 1. Code Reusability: Design patterns offer proven solutions to common problems, making it easier to reuse code and avoid reinventing the wheel.
  2. Improved Communication: Patterns provide a shared vocabulary that developers can use to communicate design decisions and implementation strategies more effectively.
  3. Maintainability: They make the code more organized and easier to maintain by providing a clear structure.
  4. Scalability: They allow for extending systems without major overhauls, especially creational and structural patterns.

Disadvantages:

* 1. Overuse: Applying patterns in situations where they aren't necessary can lead to overly complex code, making the system harder to understand and maintain.
  2. Learning Curve: Understanding and implementing design patterns effectively requires knowledge and experience, which can be challenging for beginners.
  3. Rigid Designs: Sometimes patterns can lead to rigid designs that may be harder to adapt to changes or new requirements if the wrong pattern is applied.

1. What are some major drawbacks of the Singleton design pattern?
   1. Global State: Since the Singleton pattern introduces a global instance, it can lead to hidden dependencies across the system, making it harder to track the flow of data and debug.
   2. Testing Difficulties: Singleton instances can make unit testing more challenging, as they introduce global state that might be difficult to isolate or mock.
   3. Concurrency Issues: In multi-threaded environments, managing the Singleton instance's lifecycle and ensuring thread-safety can become complex, potentially leading to race conditions.
   4. Tight Coupling: The Singleton pattern can lead to tight coupling between the Singleton and the classes that use it, making it harder to refactor the code or switch to another implementation.