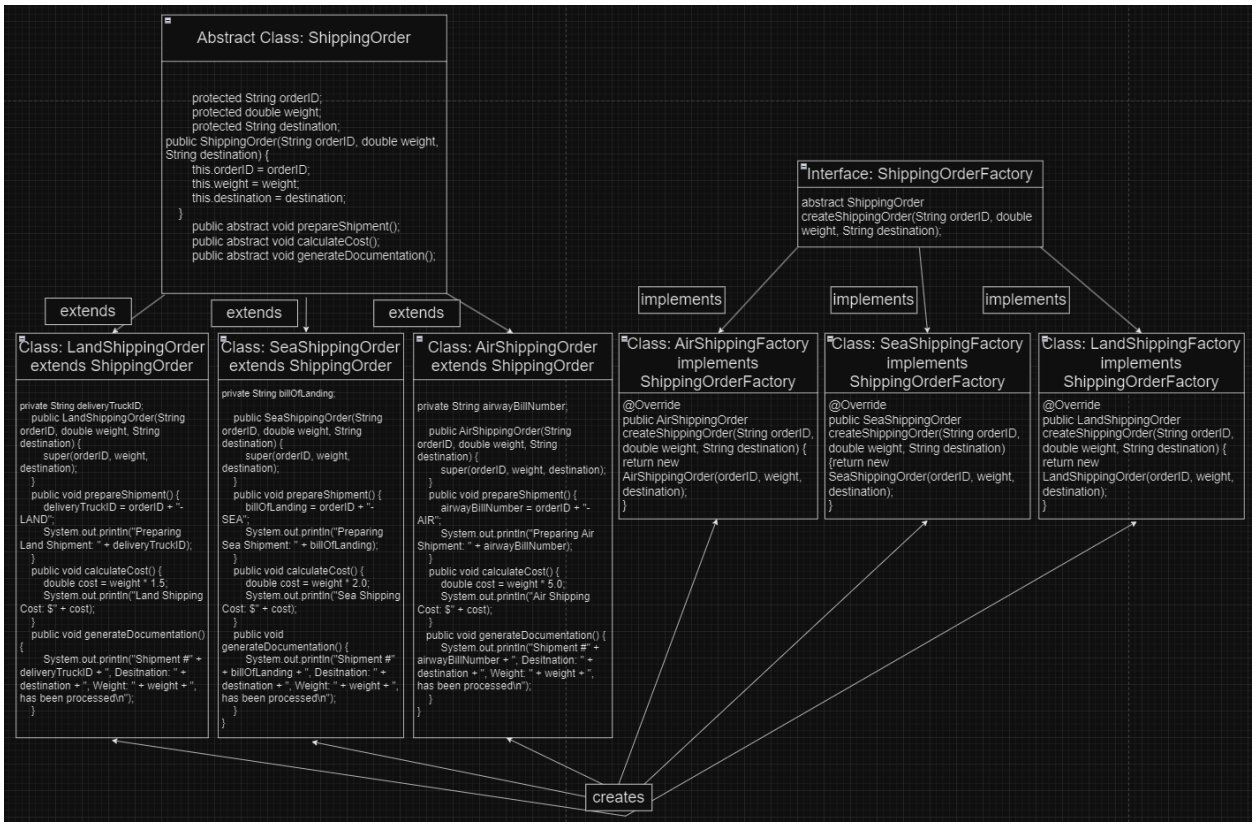
Group Members: Kyle Langenderfer, Andrew Reardon, Alex Ivary

Problem 1

Design:



Explanation:

The factory pattern is the most suitable design pattern because:

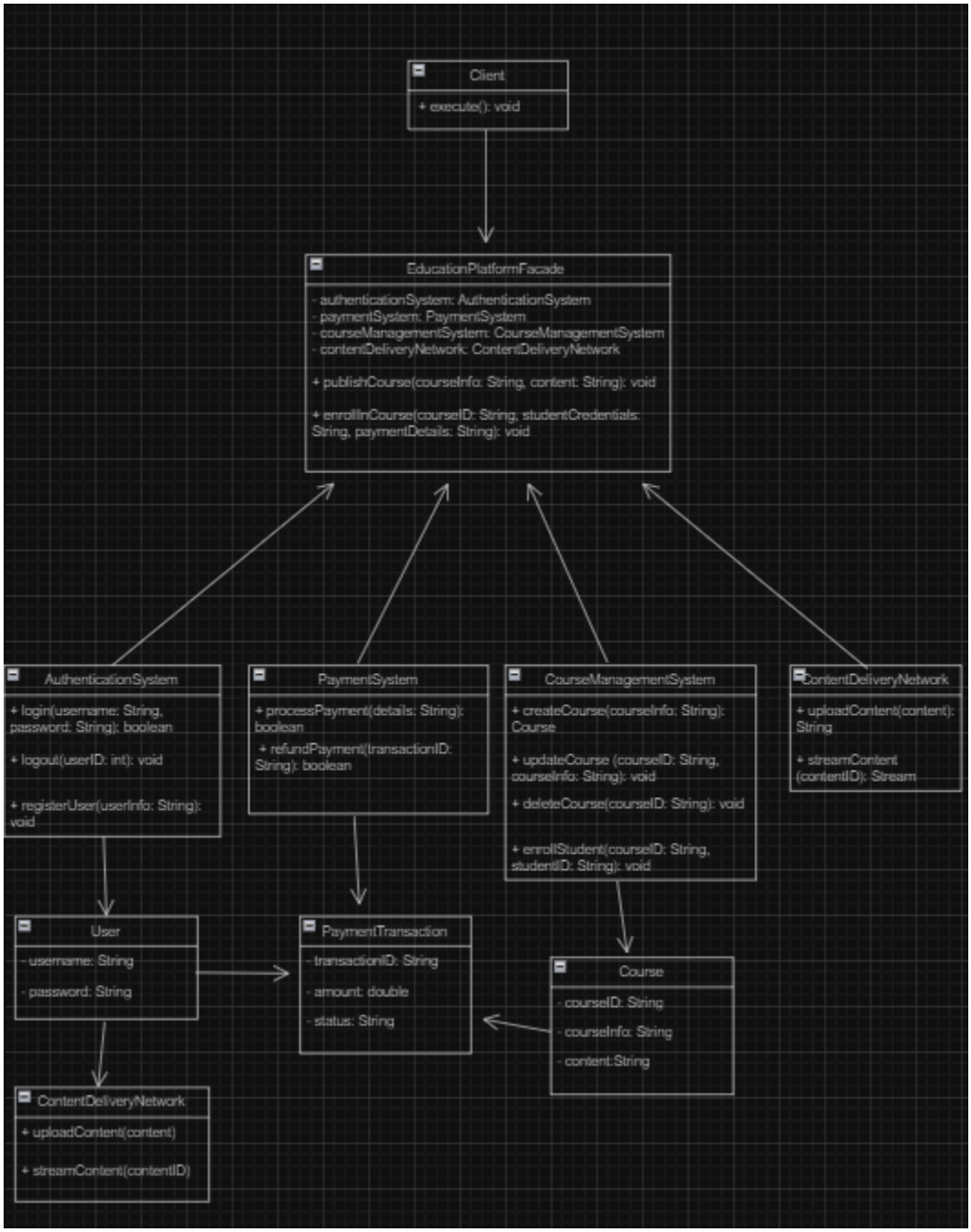
- The client code remains independent of the specific classes.
- Adding new shipping methods (e.g., DroneShippingOrder) requires minimal changes to the client.
- If a new shipping type is added, we can introduce a new concrete factory (e.g., DroneShippingOrderFactory) and its associated class without modifying existing code.
- Existing client code can continue to work seamlessly with the new factory implementation.

Why other patterns are less suitable:

- **Builder Pattern** - The builder pattern is less suitable for this scenario because shipping orders in this system don't have a super complex construction process with stages. Instead, the factory pattern fits better because it directly produces initialized objects, making it more straightforward
- **Facade Pattern** - The facade pattern is less suitable because it is typically used for simplifying subsystem interactions and is too broad for this scenario. Alternatively, the factory pattern is better for this scenario because it focuses on polymorphic object creation which is specifically a requirement for this system.

Problem 2

Design:



Explanation:

- The Facade pattern best fits this scenario because the platform comprises multiple subsystems.
- The client only interacts with the EducationPlatformFacade which provides high-level methods like publishCourse and enrollInCourse. This removes the need for the client to handle the complexities of interacting directly with multiple subsystems.
- The facade acts as a single point of access, encapsulating the internal workings of the subsystem. This reduces coupling and ensures that changes in the subsystem APIs don't impact code
- Streamlined: The facade coordinates interactions between subsystems, ensuring that processes like course publishing and enrollment are seamlessly integrated with no need for the client to manage the communication between them manually

Why other patterns are less suited for this

- Mediator Pattern
    - It's not suitable because the subsystems are not intended to communicate with each other independently. Instead, the goal is to provide a unified interface to the client making the facade a better choice
- Adapter Pattern
    - The subsystem APIs and client needs are not incompatible. The facade pattern is better because it abstracts and simplifies existing APIs rather than modifying or adapting them.