Certainly! Here are some advanced object-oriented assignment questions in Go (Golang) to help you master concepts such as interfaces, structs, struct embedding, and method signatures.

1. Understanding Structs and Interfaces

Question:

Create a `Vehicle` interface that has methods `StartEngine()` and `StopEngine()`. Implement two structs, `Car` and `Bike`, that satisfy the `Vehicle` interface. Write a function `OperateVehicle` that takes a `Vehicle` as an argument and calls its methods.

- **Objective:** Understand the use of interfaces and how different structs can implement the same interface.

2. Struct Embedding and Composition

Question:

Create a struct `Person` with fields `Name` and `Age`. Create another struct `Employee` that embeds `Person` and adds an additional field `EmployeeID`. Write methods for `Person` and `Employee` to display information about them. Demonstrate how to access the embedded struct's fields and methods.

- **Objective:** Learn how to use struct embedding for composition and field/method access in Go.

3. Interfaces and Polymorphism

Question:

Define an interface `Shape` with methods `Area()` and `Perimeter()`. Implement `Shape` interface in two structs, `Circle` and `Rectangle`. Write a function `PrintShapeDetails` that takes a `Shape` interface and prints its area and perimeter.

- **Objective:** Gain a deeper understanding of polymorphism through interfaces.

4. Method Sets and Receivers

Question:

Create a struct `Counter` with a field `Value`. Write methods `Increment()` and `Decrement()` with both pointer and value receivers. Create an interface `Countable` with methods `Increment()` and `Decrement()`. Explain the difference in behavior when implementing the interface using the value receiver vs. pointer receiver.

- **Objective:** Explore method sets and the implications of value vs. pointer receivers.

5. Interfaces and Struct Methods

Question:

Define an interface `Notifier` with a method `Notify()`. Create a struct `User` with fields `Name` and `Email`. Implement `Notifier` in `User` by writing a method `Notify()` that prints a notification message. Then, create another struct `Admin` that embeds `User` and adds a field `Level`. Override `Notify()` in `Admin` to include the level in the notification message. Demonstrate how polymorphism works in this scenario.

- **Objective:** Understand how struct embedding and method overriding interact with interfaces.

6. Using Interfaces for Dependency Injection

Question:

Create an interface `Storage` with methods `Save(data string)` and `Load() string`. Implement two structs, `FileStorage` and `MemoryStorage`, that satisfy the `Storage` interface. Write a function `UseStorage` that accepts a `Storage` interface and demonstrates saving and loading data. Explain how this setup can be used for dependency injection.

- **Objective:** Learn how to use interfaces for dependency injection to make code more flexible and testable.

7. Interface Segregation and Real-World Simulation

Question:

Define a set of interfaces for a multimedia application: `Player` (methods: `Play()`, `Pause()`), `Recorder` (method: `Record()`), and `Streamer` (method: `Stream()`). Implement these interfaces in different structs like `MediaPlayer`, `StreamingDevice`, and `RecordingDevice`. Write a function that simulates a scenario where a `MediaPlayer` can be used both as a `Player` and `Recorder`, while a `StreamingDevice` can be used as a `Player` and `Streamer`.

- **Objective:** Apply interface segregation to simulate real-world scenarios and understand the importance of cohesive interfaces.

8. Composition Over Inheritance

Question:

Create an interface `Flyer` with a method `Fly()`. Create two structs `Bird` and `Airplane` that implement `Flyer`. Create another struct `FlyingMachine` that

embeds `Airplane` and `Bird`. Demonstrate how you can use `FlyingMachine` to call the `Fly` method of both embedded structs and discuss how this approach differs from traditional inheritance.

- **Objective:** Understand the principle of composition over inheritance and its application in Go.

9. Chaining Methods with Structs

Question:

Design a struct `Builder` that has methods to set various properties (`SetName()`, `SetAge()`, `SetAddress()`) and returns the modified struct itself (method chaining). Write a function `Build()` that finalizes the build process and returns a completed `Person` struct.

- **Objective:** Practice creating fluent interfaces and method chaining with structs.

10. Implementing the Strategy Pattern

Question:

Create an interface `PaymentStrategy` with a method `Pay(amount float64)`. Implement two structs, `CreditCard` and `PayPal`, that satisfy the `PaymentStrategy` interface. Write a struct `ShoppingCart` that has a method `Checkout(strategy PaymentStrategy)` and demonstrate using different payment strategies.

- **Objective:** Understand how to implement design patterns like Strategy using interfaces.

11. Extending Interfaces and Structs

Question:

Create an interface `Logger` with methods `LogInfo(message string)` and `LogError(message string)`. Implement this interface in a struct `SimpleLogger`. Then create an extended interface `AdvancedLogger` that includes `Logger` and adds a method `LogDebug(message string)`. Implement `AdvancedLogger` in another struct `DetailedLogger`. Write functions that demonstrate how `SimpleLogger` and `DetailedLogger` can be used interchangeably where appropriate.

- **Objective:** Explore interface extension and how to use it effectively in Go.

12. Interface Type Assertions and Type Switching

Question:

Create an interface `Animal` with a method `Speak()`. Implement `Animal` in structs `Dog`, `Cat`, and `Cow`. Write a function `IdentifyAnimal` that takes an `Animal` and uses type assertions and type switches to identify the type of animal and print a specific message.

- **Objective:** Learn how to work with type assertions and type switches for dynamic type handling.

13. Exploring Empty Interfaces and Type Safety

Question:

Create a function `Describe(data interface{})` that accepts an empty interface and uses reflection to print the type and value of `data`. Write code to demonstrate how `Describe` can handle different types (`int`, `string`, `struct`, `slice`), and discuss the trade-offs between using empty interfaces and type safety.

- **Objective:** Understand the role of the empty interface and reflection in Go.

14. Implementing the Observer Pattern

Question:

Define an interface `Observer` with a method `Update(data string)`. Create another interface `Subject` with methods `RegisterObserver(o Observer)`, `DeregisterObserver(o Observer)`, and `NotifyObservers()`. Implement these interfaces in structs `WeatherData` and `WeatherDisplay`. Demonstrate how to create an observer pattern using these interfaces and structs.

- **Objective:** Learn how to implement the Observer pattern in Go.

15. Method Receivers and Interface Satisfaction

Question:

Create a struct `Account` with a method `Deposit(amount float64)` that uses a pointer receiver and a method `Balance()` that uses a value receiver. Define an interface `Bank` with methods `Deposit(amount float64)` and `Balance() float64`. Explain why the interface can only be satisfied by `*Account` and not `Account`.

- **Objective:** Deepen understanding of method receivers and their impact on interface satisfaction.

These questions should help solidify your understanding of advanced object-oriented concepts in Go, focusing on practical applications and real-world scenarios. Good luck with your learning journey!