REQ3

We have decided to create a Monologue interface that provides a contract for monologue items. The concrete implementations of this interface are passed into the MonologueAction class which has the responsibility of executing the action of listening to a monologue. Additionally, we have introduced a Subscription class to allow items to be subscribed to. This design adheres to the following principles:

Single Responsibility Principle (SRP):
Each class in the system has a single responsibility. The MonologueAction class is solely responsible for executing the action of listening to a monologue. Additionally, the Monologue interface is responsible for representing a monologue item. Finally, the Subscription class is responsible for managing the subscription of items. This ensures that each class is focused and cohesive, making the system easier to understand and maintain.

Open-Closed Principle (OCP):
By making Monologue an interface, the system is open for extension but closed for modification. New implementations of Monologue can be added without changing the MonologueAction class by simply implementing the interface. This allows for new monologue items to be added to the system without affecting existing code. Similarly, the Subscription class allows for new subscription-related functionality to be added and different subscription types to be assigned to items without modifying the existing code. This promotes code reusability and extensibility.

Liskov Substitution Principle (LSP):
All concrete implementations of the Monologuable interface can be substituted for each other without affecting the correctness of the system. This is because they all adhere to the same contract defined by the interface. This allows for polymorphism and flexibility in the system.

Interface Segregation Principle (ISP):
Monologuable interface is designed to be minimal and specific to the needs of the MonologueAction class. This ensures that classes that implement the interface only need to implement the methods that are relevant to them, and not any unnecessary methods.

Dependency Inversion Principle (DIP):
Monologue is passed into MonologueAction via Constructor Injection, which allows the MonologueAction class to depend on abstractions rather than concrete implementations. This

reduces coupling between classes and makes the system more modular and testable.

Connascence of Algorithm:
This type of connascence occurs when multiple components must agree on a particular algorithm. MonologueAction class doesn't need to know the specific algorithm used by the Monologuable instance it stores to generate the monologue. It just calls the monologue method. The algorithm is encapsulated within the Monologuable implementations, reducing the connascence of algorithm.

Data Clumps & God Class:
Initially, there was no Subscription class and all subscription-related data was stored in the Astley class. This led to a God Class that had too many responsibilities and was difficult to maintain. By introducing the Subscription class, the data clumps were separated into a separate class, encapsulating the subscription-related data and behavior. This makes the system more modular and easier to maintain.

However, there are some potential drawbacks to this design. Firstly, if Astley is the only monologue item, having an interface might be overkill. Secondly, if Astley is the only monologue item that requires subscription, it may be overkill to have a separate Subscription class. Finally, if the system grows and more monologue items are added, the codebase could become larger and more complex to manage.