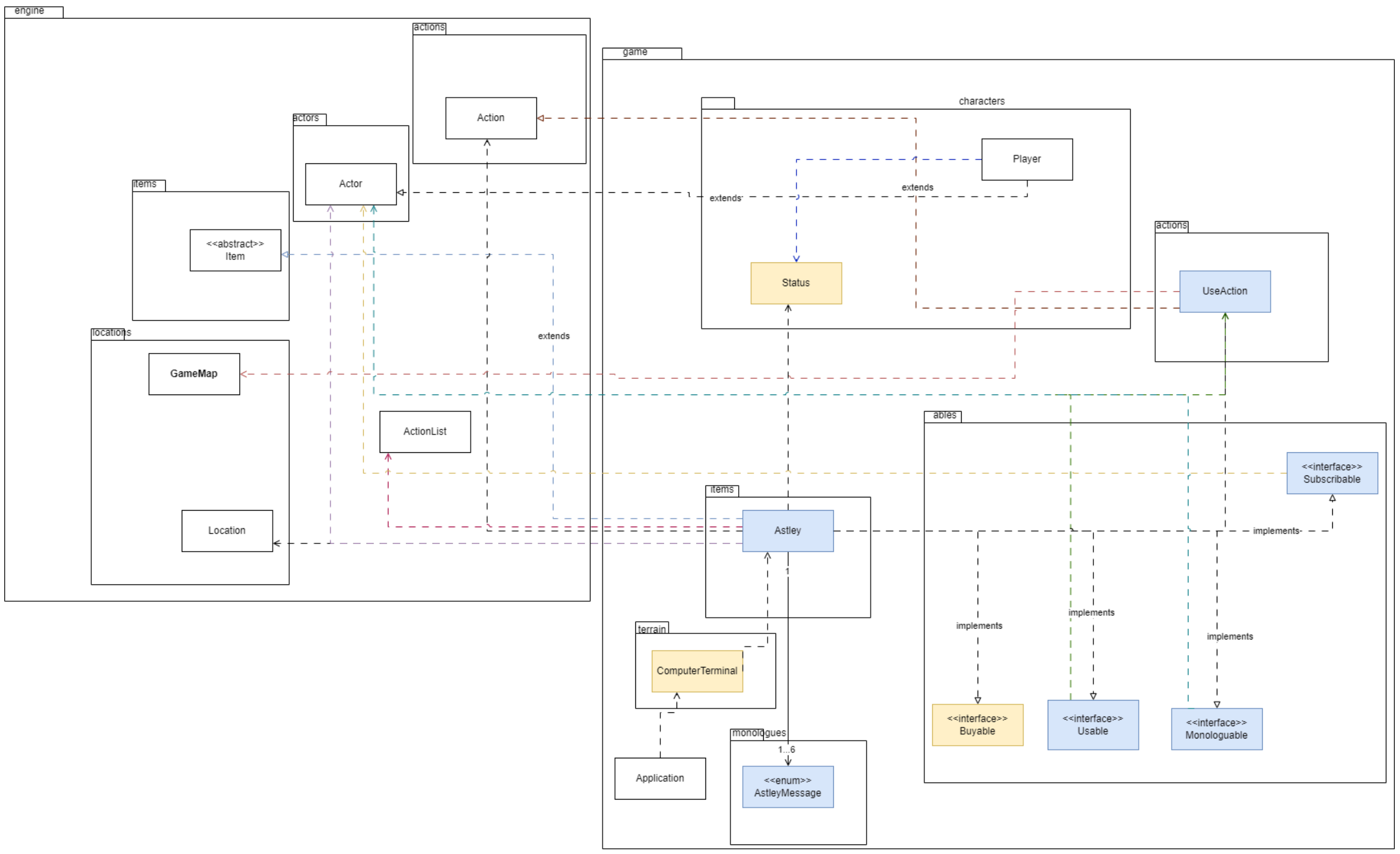
Assignment 3 Design Rationale

Requirement 3: dQw4w9WgXcQ

UML Diagram:



- Blue classes are newly made
- Yellow classes existing from Assignment 2

Design Goal:

To add the AI device "Astley" into the intern's inventory, allowing it to be bought, managed through a subscription system, and provide random monologues based on the intern's status. The design should also be flexible enough to easily add similar features for other characters or items in the future.

Design Decision:

I created new interfaces for Usable, Subscribable, and Monologuable to be implemented by Astley.
A new Usable interface was created instead of reusing the Consumable interface as Consumable denotes an item that is consumed and removed from inventory, but Usable denotes any item or interactable item that can be used repeatedly. Monologuable denotes anything that would have an enum of preset messages that can be generated. Subscribable has 2 methods, checkActive, and paySubscription. CheckActive checks for the conditions required for subscription to be active, in this case, that Astley is in inventory, as well as whether the subscription fee has been paid. PaySubscription checks the owner balance and then pays if sufficient. This ensures that in the future, any new items (or other) can be Usable, Subscribable, and Monologuable and be implemented with ease.

Alternative Design:

One alternative approach could involve coding the Astley class with all the checks for the active subscription, as well as the payment of the subscription within the same method. Instead of using multiple methods, a subscription() method would instead be called, that checks if Astley is in inventory, whether the owner has enough balance to pay the subscription, etc.

Analysis of Alternative Design:

The alternative design is not ideal because it violates various Design and SOLID principles:

1. Single Responsibility Principle (SRP):

Violation: Combining all subscription-related checks and actions into a single subscription() method causes the method to handle multiple responsibilities: verifying inventory presence, checking balance, deducting the fee, and managing subscription status.

Explanation: This violates SRP because a method should have only one reason to change. By having a single method manage multiple responsibilities, any change in the subscription logic or balance checking will require modifying this method, making the code harder to maintain and more error-prone.

2. Open/Closed Principle (OCP):

Violation: A single subscription() method that handles all aspects of subscription logic would need to be modified whenever new subscription conditions or actions are introduced.

Explanation: the class is not closed for modification. Any new functionality related to subscriptions would require changes to the existing method, increasing the risk of introducing bugs and making it harder to extend the system.

3. Interface Segregation Principle (ISP):

Violation: The alternative design does not leverage interfaces effectively, leading to a monolithic method that combines various actions.

Explanation: it forces a single method to handle diverse responsibilities that could be separated into different interfaces. Proper use of interfaces would allow specific functionalities (like checking balance or managing inventory) to be handled separately, making the system more modular and easier to manage.

4. Dependency Inversion Principle (DIP):

Violation: The subscription() method may depend on concrete details rather than abstractions, making the system less flexible.

Explanation: High-level modules (like the Astley class) should depend on abstractions (interfaces) rather than concrete implementations. By combining all logic in a single method, the design becomes less modular and harder to adapt to changes.

Final Design:

In our design, we closely adhere to [relevant design principles or best practices, e.g., SOLID, DRY, etc.]:

1. **Single Responsibility Principle (SRP):**

    - **Application:** Each class and interface in the design has a single responsibility. Astley handles functionality specific to the AI device, while interfaces like Usable, Subscribable, Monologuable define the distinct capabilities.

    - **Why:** This ensures any changes in one part of the code will not affect the other, making it easily extensible.

2. **Open/Closed Principle (OCP):**

    - **Application:** OCP means open for extension, closed for modification. The use of interfaces for each specific type of “able” allows for differing conditions for those instances. E.g. Subscribable has checkActive and paySubscription methods, but they are undefined as different Subscribables might have different conditions for fees, or conditions for being held as active.

    - **Why:** Easier to extend

    - **Pros/Cons:** More scalable, but needs careful consideration of which interfaces to create

3. **Interface Segregation Principle (ISP):**

    - **Application:** Astley uses multiple interfaces instead of just being coded for a singular Astley device

    - **Why:** more modular code

    - **Pros/Cons:** Easier to implement any instances that will have similar functionality, but requires careful management of interfaces.

4. **Liskov Substitution Principle (LSP):**

- **Application:** ensures that any class implementing an interface can be used interchangeably without affecting the correctness of the program. For example, any class that implements the Monologuable interface can have a monologue based on the different actors.
- **Why:** ensures that any subclasses can implement the base classes or interfaces without affecting the designed behaviour of the code
- **Pros/Cons:** promotes polymorphism

Conclusion:

Overall, our chosen design provides a robust framework for [achieving the assignment objective]. By carefully considering [relevant factors, such as design principles, requirements, constraints, etc.], we have developed a solution that is [efficient, scalable, maintainable, etc.], paving the way for [future enhancements, extensions, optimizations, etc.].