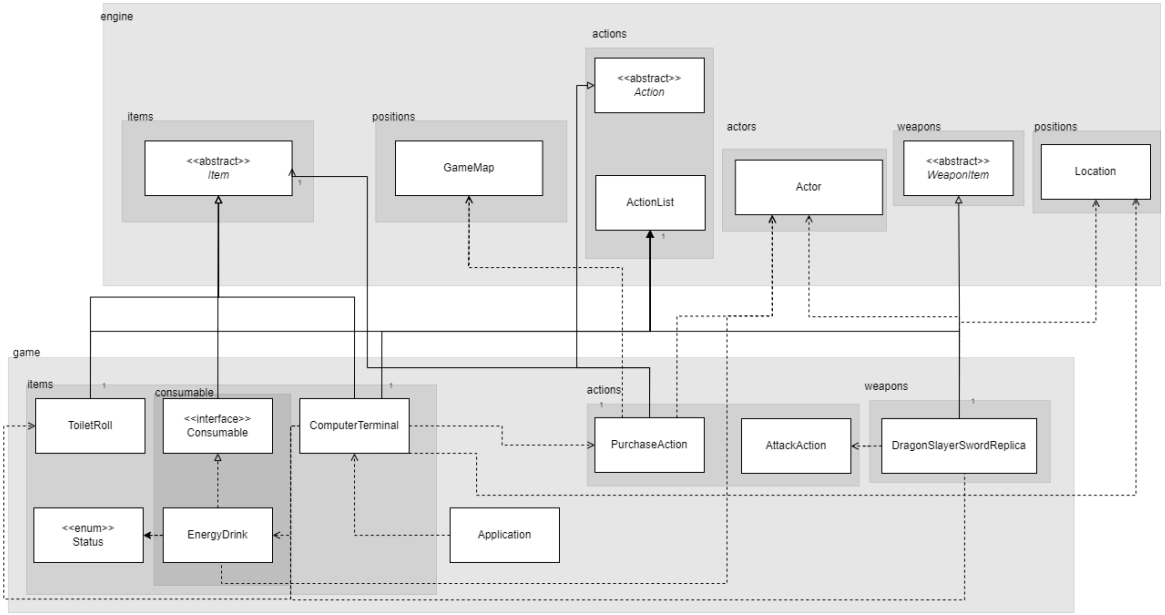
Assignment 2 Design Rationale
REQ4 : Static Factory's Staff Benefits

UML



Design Goal

1. **Enhance Player Experience:** Introduce a Computer Terminal feature that offers players the ability to purchase items and weapons.
2. **Promote Randomness:** Implement randomized pricing and chance-based events to introduce variability and randomness.
3. **Maintain Code Quality:** Ensure that the design adheres to principles of modularity, encapsulation, and code reuse to facilitate easy maintenance and future expansion of the game.

Design Decision

The proposed system is designed to enhance the functionality of a game by introducing new elements, Computer Terminal that allows players to purchase items.

EnergyDrink extends Item and implements Consumable – making it a consumable item. It extends consumable as the actor can use it up. It inherited the Item class constructor to set its unique attributes and has its own methods, defining its prices. It also inherits the consume and allowableActions class to perform the consume action, as well as implement the effects of consuming it to the actor.

ToiletRoll extends Item and has its own method, getToiletPaperPrice, to determine its price.

DragonSlayerSwordReplica extends WeaponItem. Like MetalPipe, it has its own constructor, and allowableActions to enable the attack when it is acquired.

PurcaseAction

Alternative Design

An alternative design could be to implement everything within the Computer Terminal itself without the use of the PurchaseAction.

Analysis of Alternative Design

This is considered a bad design choice as it would violate the SRP Principle and the DRY Principle:

- **Violation of SRP:** The **ComputerTerminal** class would take on multiple responsibilities, including user interaction and purchasing logic, leading to code that is harder to maintain and extend.
- **Violation of DRY:** The logic for purchasing items would be duplicated within the **ComputerTerminal** class, leading to maintenance challenges and potential inconsistencies.

Final Design

The implementation follows the following principles

Single Responsibility Principle (SRP)

Each class focuses on a specific aspect of the game, such as representing game entities, defining behaviors, or implementing actions. For example, the **ComputerTerminal** class handles interactions with the computer terminal, while the **PurchaseAction** class manages the purchase process while overall also promotes modularity and encapsulation. Each class is responsible for a distinct aspect of the game's functionality, enhancing code organization and maintainability.

High cohesion is also achieved by ensuring that each class has a single, well-defined purpose. For instance, the **EnergyDrink** class handles the behavior of the energy drink item, with related functionalities grouped together.

Inheritance and Polymorphism

Inheritance is utilized in item classes such as **EnergyDrink**, **ToiletRoll**, and **DragonSlayerSwordReplica**, which inherit from the **Item** class. This allows them to inherit common attributes and behaviors while specializing their functionality.

Polymorphism is demonstrated through method overriding, such as in the **execute()** method of the **PurchaseAction** class, where different implementations are provided based on the type of item being purchased.

Don't repeat Yourself (DRY)

The DRY principle is followed by encapsulating common functionalities like purchasing items in the **PurchaseAction** class. This class contains the logic for purchasing items, which can be reused across different item types without duplication.

Flexibility and Extensibility

The design allows for easy extension by adding new item subclasse. For example, additional items can be integrated into the Computer Terminal by creating new subclasses of **Item** and using the **PurchaseAction**, promoting flexibility and extensibility.

The use of randomized pricing for items adds variability to the game and allows for the introduction of chance-based discounts.

Pros & Cons

Pros

- Clear organization and separation of concerns enhance code readability and maintainability.
- The design promotes code reusability and extensibility, allowing for easy addition of new items and actions.

Cons

- Code Duplication: The execute method in the PurchaseAction class contains some duplicated code.

Conclusion

The implementation overall follows the SOLID principle, mainly SRP and OCP, effectively minimising duplicates through the DRY principle. While there are room for improvement, the expected outcome to introduce and manage consumable items in the game was achieved.