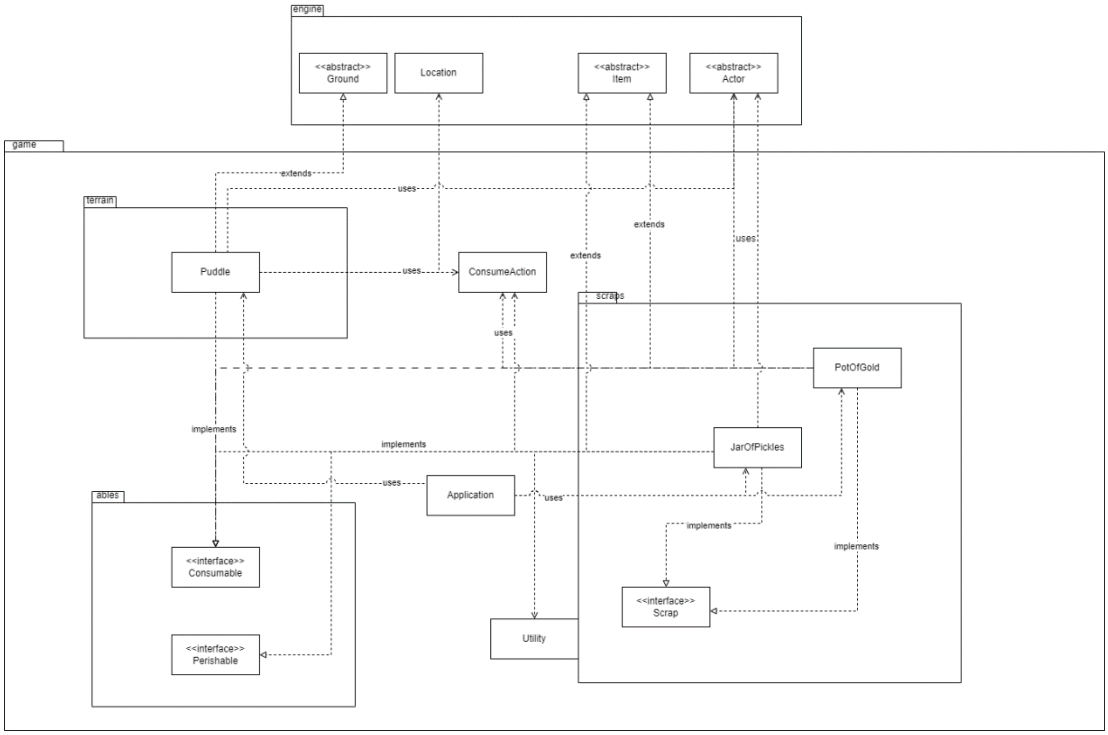
Assignment 2 Design Rationale

Requirement 3: More Scraps

UML Diagram:



Design Goal:

The goal for this requirement was to add new types of Scrap to the moon that the Intern can interact with. These include a Jar of Pickles and a Pot of Gold. Additionally, now Puddles can be consumed. These must be implemented following the SOLID principles, so using the extensible program interfaces already in place.

Design Decision:

A new interface, Perishable, was created for anything that can expire/deteriorate. This checks whether the instance has expired, and ensures the effects of expiration is decided specific to each item, because different items could have different conditions for expiry.

For both new Scraps, allowableActions() method provides a list of Actions, specifically Consume Actions. Puddle is also made to have the allowableActions() method, because it is now consumable.

JarOfPickles class is responsible for the instantiation of the item JarOfPickles, which is both consumable and perishable. Expired() checks whether the item has expired, and in effect(), if the item has expired, the specific effects are applied to the actor upon consumption.

PotOfGold class is responsible for the instantiation of the item PotOfGold that can be consumed to add to the balance of the actor. This implements Consumable interface as well.

Puddle class is a class created originally to extend Ground class only, as a type of terrain. But with the new requirement, it now also implements Consumable interface, so it can be consumed to increase the maximum HP of the actor. However, the returnItem() method returns null, because Puddle is not an item.

Analysis of Chosen Implementation

Single-responsibility Principle

Perishable is only responsible for assigning the expired() method in perishable items.

Open-closed Principle

Making puddle a Consumable, just returning null in returnItem(), instead of editing the code within Consumable.

Interface Segregation Principle

Creating a new Perishable interface follows the ISP as it clearly separates a new possible type of item.

Dependency Inversion Principle

Puddle, JarOfPickles, and PotOfGold class implements Consumable interface, instead of hard-coding what happens when consumed.

All new classes in requirement 3 follow the DIP by using abstraction to instantiate specific types of items.

Alternative Implementations for items affecting health

Another possible implementation for items that can affect health would be to add 2 more interfaces, Healable, and Damageable.

Healable could have a method: healed(); to determine what happens if the item heals the actor, and vice versa for Damageable. But it would just be redundant, as there is already

heal() and hurt() methods within the actual Actor class, violating the Interface Segregation Principle.

Or a static method could have been implemented to check whether actor is conscious, within the Consumable interface.

This would not be optimal as it doesn't call for the method to be implemented in all Consumable items, so if it were left out of the instance of an item, let's say a new item called Slime, which slowly reduces player HP over time, when HP reaches 0, the player would still be alive.

Alternative implementations for Consumption of Puddle

The returnItem() in Puddle could be said to violate the principle of interface segregation, as it returns null, but the alternative would be to create a new interface like GroundConsumable to separate from ItemConsumable, or to remove the returnItem method from the current Consumable interface.

The first alternative would be redundant, as GroundConsumable would now have the same methods as ItemConsumable only without returnItem, causing over-abstraction.

The 2nd alternative would be inefficient, as we would then have to create a method to access the item to be able to add or remove it from the actor's inventory every time a new item is created.