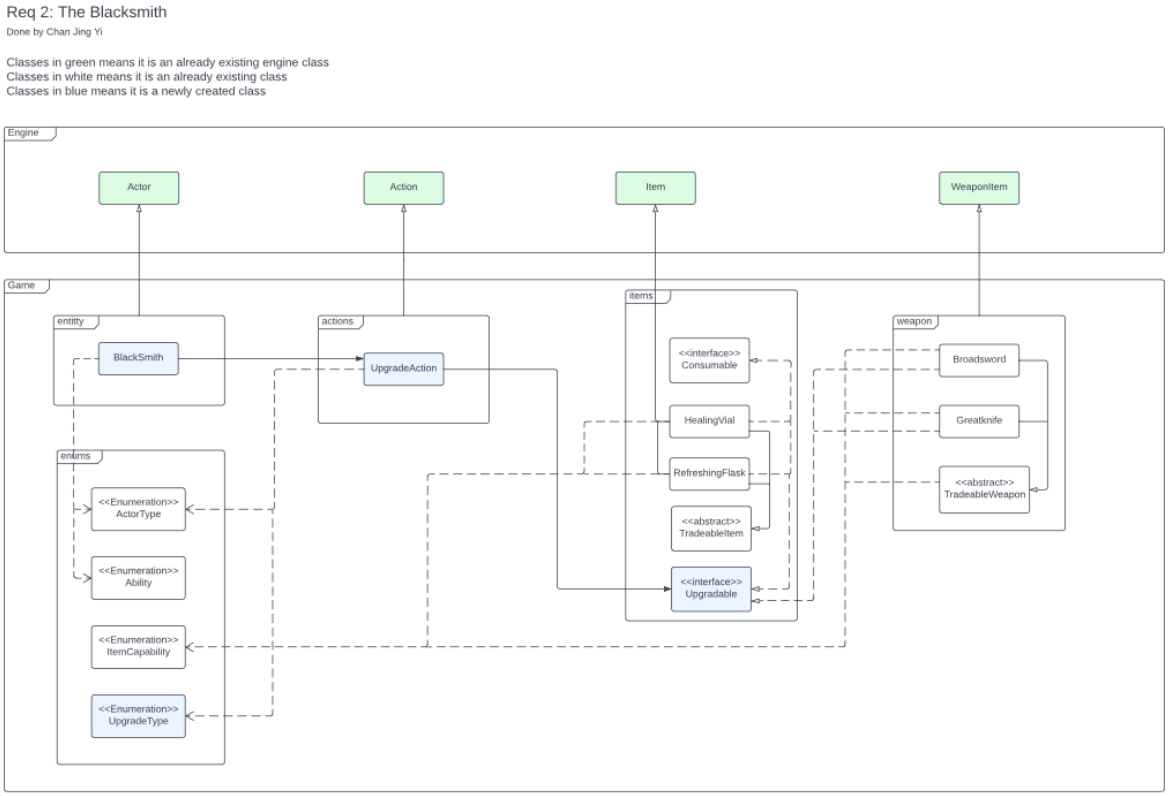
A3 Design Rationale Req 2 (Done by Jing Yi)



Problem and Solution

The upgrade feature allows players to improve the effectiveness of certain items by trading runes with service from the Blacksmith. My initial attempt was to create an abstract “UpgradableItem” class. However, I encountered a problem during the implementation as there was an abstract “TradeableItem” class that was previously created in Assignment 2, Java does not support multiple inheritance, which means we couldn't extend both “UpgradableItem” and “TradeableItem” at the same time.

To overcome this limitation, the decision was made to create an “Upgradable” interface, which could be implemented by any item that supports upgrades. This approach provided the flexibility needed to separate the concerns of tradeability and upgradability. By implementing the “Upgradable” interface, items could indicate that they are eligible for upgrades without being constrained by the limitations of Java's inheritance model.

How it works

The upgrade mechanism is simple. Upon upgrading, the item's attributes are directly altered. This is achieved by modifying the internal state of the item. For instance, in the case of a Broadsword, the damage is increased by a fixed amount for each upgrade. This approach allows for a dynamic and customizable upgrade system, ensuring that upgrades are reflected immediately in the item's behaviour.

To keep track of an item's upgrade status, a boolean flag ‘isUpgraded’ was introduced. This flag serves as a simple indicator to prevent multiple upgrades of the same item if it is a “upgrade once” type of item. Additionally, an ‘upgradeCount’ variable was introduced to keep a record of how many upgrades have been performed on an “upgrade multiple times” type of item. This information can be used to calculate the cumulative effect of upgrades on the item's attributes.