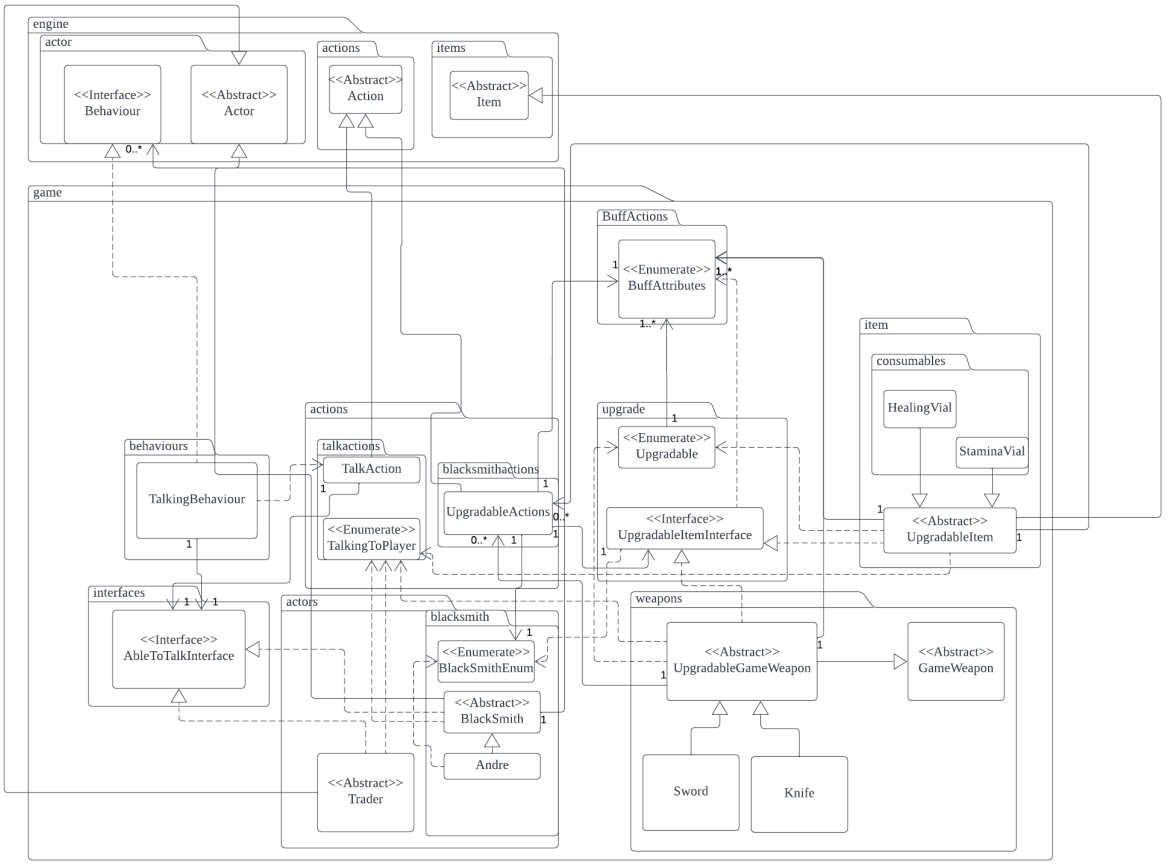
Design rationale of Assignment 3 REQ 2 and 3

REQ 2 design rationale



Design Decision

The talking functionalities originally from Trader has is also now in AbleToTalkInterface to represent that the player can talk the actor for more actions. By doing this, the BlackSmith class which has similar talking functionalities can implement from that interface or else the DRY principle could be violated and make it easy to manipulate attributes which any entities implementing AbleToTalkInterface have.

The blacksmith class is an abstract class because not all blacksmiths are the same, this allows for blacksmiths with a different mechanic to be implemented by extending from the BlackSmith class. The Andre class is meant to represent the blacksmith in requirement 2.

Upgradable is used to make it easier to inject the possible stats that can be increased to an upgradable item. Without this, if 2 weapons have the same 4 stats that can be raised for example, using the upgradable enum is less tedious than writing the same 4 stats again to the other weapon.

All items which are upgradable implements the UpgradeItemInterface, such as UpgradeGameWeapon which extends from GameWeapon and UpgradableItem. Naturally,

sword and knife now extends from UpgradeGameWeapon and the vials extend from UpgradableItem because they are upgradable. This makes it easy to create new upgradable items and weapons by extending from one of these classes or else many repetitions for each upgradable item or weapon, leading to violating DRY principle.

UpgradableGameWeapon and UpgradableItem stores the upgradable actions when encountering a certain blacksmith and the BuffAttributes as a hashmap which leads to a value representing the stat increase. If the upgradable actions or the hashmap are not stored in these classes, repetition of declaring both of them will occur every time, leading to violating DRY principle.

An UpgradableActions will present itself as an option when the correct blacksmith, the item to upgrade, holds the correct attribute which it can be buffed by, if the blacksmith has the capability from TalkingToPlayer to represent that the player is talking to this specific blacksmith and if the level cap is not reached. The reason that the upgradable items offer the UpgradableActions instead of the blacksmith is because instanceof would need to be used to find the correct item to buff. The reason for such an implementation is so that swords that can only have its hit rate upgraded will not have the option to have its attack upgraded. More reasons are because not all blacksmiths may offer the same upgrades and to not display the upgrades if the user finds it annoying that it is popping up everytime especially if the blacksmith offers many upgrades so the player must talk to the blacksmith first.

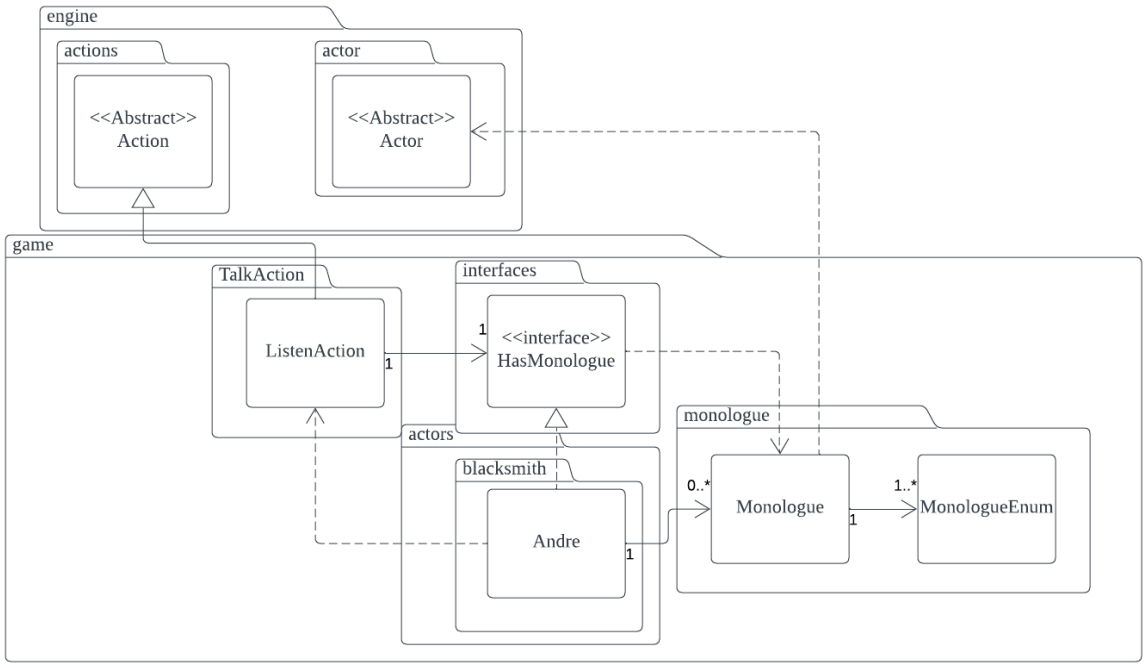
Design Principles

All classes in this requirement have their own responsibilities to adhere to SRP. UpgradableGameWeapon adheres to the Open Closed principle by extending and adding more functionalities from GameWeapon. The Open Closed principle is also followed for all classes extending from another class.

AbleToTalkInterface is used to adhere to the DI principle because the TalkAction will not need to be modified if there is another talkable actor with a different way of interacting with the player such as the player might need to do certain achievement before the actor might talk). This interface will prevent coupling due to not having to do switch cases to check which actor is talking to the player and how the interaction works.

The other interface, UpgradableItemInterface is also used to adhere to the DI principle because the UpgradeAction will not need to be modified if another type of BuffAttributes is added to accommodate how the buff will affect the upgraded item or weapon which leads to coupling. Instead, through the interface, the UpgradeAction only needs to call the UpgradableItemInterface's raiseBuffAttributeBy every time no matter how many types of BuffAttributes there are.

REQ 3 design rationale



Design Decision

Now Andre class has monologues so this class implements HasMonologue. Without the HasMonologue, obtaining the list of monologues will be difficult or would be a hacky implementation such as instanceof may need to be used to get the list of Monologues from Andre class for the ListenAction class. Monologue class will also be its own class because it will represent monologues with/without conditions which is based on the capability the player has inside of MonologueEnum. With the monologue class, ListenAction will not have to check because the monologue class does the checking instead through the sayMonologue() function. Without the monologue class, the ListenAction class will have a hard time knowing when to use the alternate dialogue or default dialogue without a tedious implementation.

Design Principle

Once again, all classes in this requirement have their own responsibilities to adhere to SRP. HasMonologue assists Monologue class to adhere to the DI principle between ListenAction and the Andre class because it helps to obtain the list of Monologue classes which is what helps prevent coupling to check for conditions for an alternate dialogue to occur instead through the Monologue's sayMonologue() function which does the checking.