Design Rationale

By: MA_AppliedSession1_Group3

REQ 2: The Blacksmith

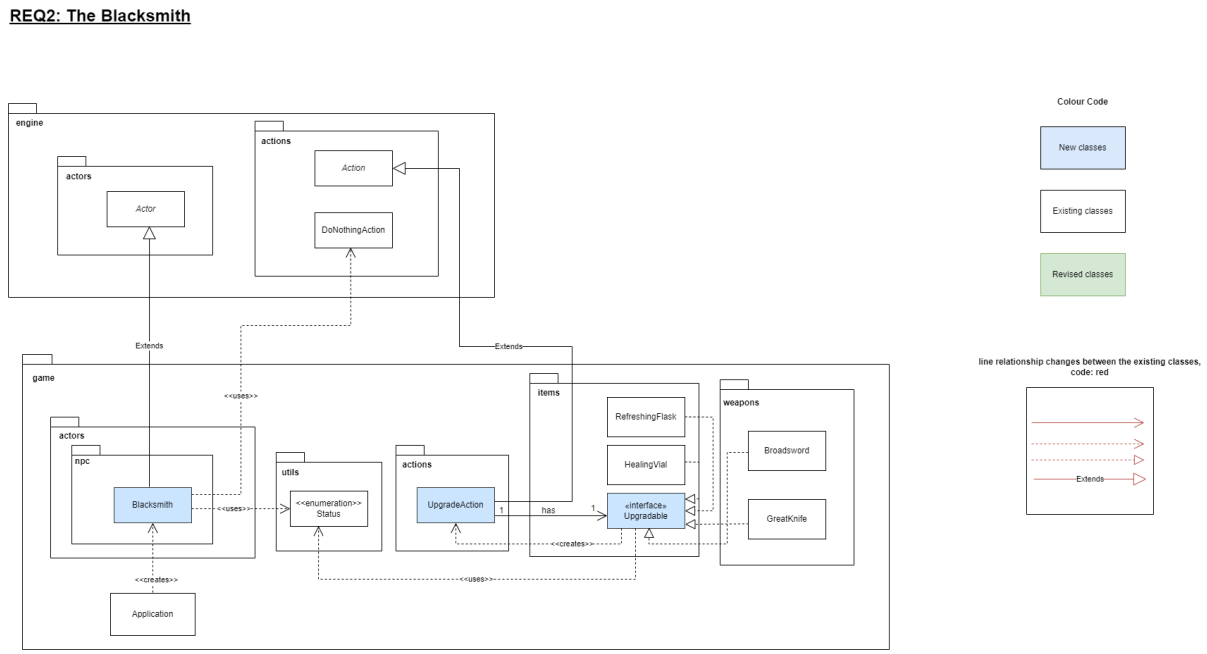

Figure 2: UML class diagram of REQ2 - The Blacksmith

The provided diagram shows the entire system implementation for Requirement 2, The Blacksmith. The primary goal of this design is to create an actor, The Blacksmith, that allows the players to upgrade the items in their item inventories with different prices. Thus, new classes and interfaces are introduced in the extended system: Blacksmith, UpgradeAction, and Upgradable. These classes will interact with the existing classes in the extended system.

To achieve the goal of the design, we created the Blacksmith class and the UpgradeAction class to execute their own responsibilities. This implementation follows the Single Responsibility Principle (SRP) because each class only focuses on their responsibilities. For instance, the UpgradeAction class only upgrades the items while not doing other work for the items. This makes the maintenance of the system easier as it would be more efficient to change the responsibility of a class without affecting the others.

Moreover, we decided to create an Upgradable interface in our extended system. This interface is implemented by the HealingVial, Broadsword, RefreshingFlask, and GreatKnife classes to make them upgradable. This design effectively aligns with the Don't Repeat Yourself (DRY) principle and prevents the code smell by using the polymorphism (Upgradable item) in the UpgradeAction class. Besides, the polymorphism with (Upgradable item) also obeys the Liskov Substitution Principle (LSP) and the Open/Close Principle (OCP) as these classes can execute the upgradedBy() methods in their own classes with different

implementations, while do not affects the implementation of upgradedBy() in the Upgradable interface.

Additionally, the design aligns with the Interface Segregation Principle (ISP) and the Dependency Inversion Principle (DIP) with the creation of the Upgradable interface. For example, the GiantHammer does not implement the Upgradable interface as it cannot be upgraded, while some item and weapon classes can be upgraded with the implementation of the Upgradable interface, which follows the ISP. For DIP, the UpgradeAction class depends on the abstraction layer of Upgradable interface (item.upgradedBy) instead of the concrete classes such as the HealingVial class. Besides, with the private field of the “Upgradable item” in the UpgradeAction class, the encapsulation for the information hiding is considered.

On the other hand, the usage of the enumeration ‘Status.UPGRADE_PERSON’ and ‘Status.UPGRADED’ might be the drawback of the design as this will be a little touch of the code smell. However, the usage of the enumeration can replace the ‘instanceof’ operator which might add extra dependencies and violate the Open/Closed principle (OCP). Besides, the complexity of the classes would significantly increase if more related interfaces or abstract classes are added. Ultimately, we decided to continue with this current design.