Design Rationale

Requirement 1

Requirement 2

Requirement 3

Requirement 4

This UML class diagram will showcase the system in the types of classes in the game. The goal of the class system is to create a system that allows the player to choose a class that has different weapons and skills while allowing the possibility of extending it.

First of all the player is an actor in the game, Therefore the Player class will extend the Actor class to inherit all its properties. Each player can also choose between 3 different classes, so 3 concrete classes called Samurai, Bandit and Wretch are created that will be associated with the player class. To avoid compromising the Dependency Inversion Principle, I created an abstract Class class to handle all the things that a class can do and let the 3 concrete classes extend it while the Player class associates with the Class class. By doing so, the game may extend the type of classes with ease due to the abstraction. Moreover, to ensure the consistency in the name of the class, the enum class ClassType is there to ensure it. Alternatively, we can just use a String to represent the class type but to prevent runtime error, an enum class is recommended.

In the game, each class has their own unique signature weapon. Therefore I have created 3 concrete classes to handle the weapons attributes and methods called Uchigatana, GreatKnife and Club class that extends from WeaponItem abstract class since all weapons are a weapon item so an inheritance is preferred for code reusability. Furthermore, since a weapon is also an item, the WeaponItem abstract class also extends from the Item abstract class. Each weapon may or may not have a skill which is a capability, so it has an association with the CapabilitySet class to handle everything about the skill. Alternatively, we can create a new class called skill action that each weapon has to provide the player the ability to do a skill if the conditions have been met. However, by creating a skill action which relates closely to the capability set this will violate the Do Not Repeat Principle. Furthermore, since a player can choose to use their fist to fight, it has a dependency with the Intrinsic Weapon class.

The player class is associated with the WeaponItem class instead of the 3 weapon concrete class to reduce coupling. Hence, if there are additional weapons we do not need to fix the Player class to accommodate the new weapons Therefore, we can obey the Dependency Inversion Principle and the Open-Closed Principle since there is no need to create more if-else to check the type of weapon which is not recommended since it involves extra dependency and modification to the code.

Requirement 5