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.

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 DIP, I created an abstract Class class to handle all the things that a class can do and let the 3 concrete classes extend it. To ensure the consistency in the name of the class, the enum class ClassType is there to ensure it so there is no runtime error.

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 share the same attributes and methods 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.

The player class has an association to the WeaponItem class instead of the 3 weapon concrete class to reduce coupling. Moreover, if there are additional weapons we do not need to fix the Player class to accommodate the new weapons. Therefore, we can obey the DIP principle and the O/CP.

Pros:
Reduce class dependency
Obey DIP,O/CP and SRP principle

Cons: The game package will be bigger as more extension is added since no new class is added to the engine package

Requirement 5