Design Rationale REQ 5

<<interface>> Tradable implemented by MagicalItem and Wrench. To follow open-closed principle, the MagicalItem and Wrench classes share the same method but different implementations. By using an interface as an abstraction, it can open for extension of the same method but different implementation.

MagicalItem---*<<create>>*--->*BuyAction* and *Wrench*---*<<create>>*---> *BuyAction*. The MagicalItem and Wrench are the objects that give the Actor(Player) an action to buy. We can simply create BuyAction inside the Player class, however by doing this, we need to know which object the player wants to buy, it will require additional dependency between Player and MagicalItem and Wrench. Besides, we also need to check whether the object allows the player to buy or not, checking the object classes using if-else statements will also increase dependency. To align our design with the Reduce Dependency Principle, we discard this alternative and use a different approach that is shown in the class diagram above.

Coin has the Status.Spend. It will be added into its capability set that will be used when the player wants to buy items.

WalletManager has a list of tradable items that were bought by the player. It is a static class with a private constructor. The reason for doing this is because we only want an instance of WalletManager and the only way to get the instance is using the getInstance() method. WalletManager allows us to keep track with item bought by player and deduct the price of item from its wallet balance