Design Rationale for Requirement 2

The diagram represents an object-oriented system for the Traders and Runes in the game.

The player's runes are managed by a RuneManager class, which is responsible for increasing and decreasing the number of runes the player has, represented by a 1 to 1 association. When an enemy is killed by the player, the RuneManager class handles awarding runes to the player. This is represented by the association relationship between Enemy and RuneManager.

The trader Merchant Kale is implemented as its own class MerchantKale, which extends the Trader abstract class. Trader itself is a subclass of the Actor abstract class, as it shares common attributes and methods. Trader has an association of 1 to many with the Item abstract class, to represent the items it is willing to buy or sell.

The player can buy and sell items and weapons from Traders through the PurchaseAction and SellAction classes respectively, which extend the Action abstract class. Both actions interact with the RuneManager class to update the player's rune count after the purchase/sale.

The use of abstract classes to represent Traders adheres to the Open-closed Principle, allowing new Traders to be added without modifying existing code. It also adheres to the DRY principle by reusing code through inheritance in each Trader.

The use of a RuneManager class to update the player's runes adheres to the Single Responsibility Principle, by handling the changes in rune count in its own class.

Changes in Assignment 2

RuneManager uses a singleton constructor to ensure only one instance of it exists at a time. An advantage of this approach is that the RuneManager instance can be accessed by any other class. A disadvantage is that the implementation (the singleton constructor) is more complex, and can cause problems if done incorrectly.

The buying and selling of weapons with Traders is implemented using the TradeableWeapon class, which declares the purchasing and or selling price of a weapon, and contains an instance of the weapon being traded. Using interfaces to represent purchasable and sellable weapons was considered, but ultimately decided against. The reason for this is that the chosen approach allows different Traders to have different purchasing and selling prices for weapons, as the purchasing and selling prices are not tied to the weapon classes themselves. A disadvantage to this approach is that more code is needed to implement the TradeableWeapon class, as well as create an instance of it for each weapon a Trader is willing to buy.