REQ4: The Wayward Seller of Wares

Implementation A: PurchasableWeapon Interface and WeaponItem Hierarchy

We created an abstract WeaponItem class to represent all weapon items. This class implements the PurchasableWeapon interface, which declares the method applyPurchaseEffect(...). Each purchasable weapon (e.g., Broadsword, Dragonslayer Greatsword, Katana) extends WeaponItem and overrides the applyPurchaseEffect method to implement side-effects triggered upon purchase. These effects are tailored to the selling merchant and may include stat boosts, healing, or spawning new entities.
This approach encapsulates purchase behavior in each weapon, avoiding type checking in the BuyWeaponAction class. The design adheres to the Open/Closed Principle by allowing new weapons to be introduced without modifying existing logic. The BuyWeaponAction operates on the abstraction (PurchasableWeapon), not the concrete weapon types, supporting Dependency Inversion. This ensures loose coupling between the buying mechanism and individual item logic.

Pros	Cons
Respects Open/Closed and DIP: purchase logic lives in weapons, not the action class	Adds minor complexity by requiring both an abstract class and an interface
Keeps weapon effects close to the weapon class, supporting cohesion	Some duplication may occur between weapons (e.g., similar healing logic)
Easy to add new weapons or effects without touching BuyWeaponAction	Assumes all purchasable weapons can and should extend WeaponItem

Implementation B: Unifying BuyWeaponAction with Dynamic Seller Effects

We implemented a single BuyWeaponAction class to handle all purchases. When triggered, the action checks the player’s runes, adds the weapon to their inventory, and delegates the post-purchase effects to the weapon’s applyPurchaseEffect(...) method.
The BuyWeaponAction is fully generic. It does not differentiate between weapons or merchants itself. It can simply pass the seller's name to the weapon, which determines what effect to apply. All logic related to side effects (e.g., spawning an Omen Sheep, restoring stamina, increasing max health) is encapsulated in the weapon classes.
This implementation follows the Single Responsibility Principle by limiting the BuyWeaponAction’s role to transaction execution. It also supports polymorphism, since the

action works with any PurchasableWeapon. All merchant-specific effects remain modular and isolated.

Pros	**Cons**
Keeps action logic simple and reusable	The seller is passed as a String, which is prone to typos
Encourages consistent item behavior across merchants	Scaling to more merchants/items may increase complexity in applyPurchaseEffect
Easy to link actions in the merchants' allowableActions()	Slightly limits flexibility compared to more complex trading systems