Design Rationale REQ3

Implementation & Why:

- Created Tradeable interface
    - Defines functions all tradeable items must have
    - Adheres to Interface Segregation Principle (ISP) has only classes that implement this interface need to add the functionality needed.
    - Has getters and setter for price, adheres to Open Closed Principle (OCP). The setters allow different Tradeable Items in different actor's inventories to have various prices. This makes it easy to set the selling prices for different Trader actors in the future.
    - Has methods to implement price variations or scams with Trader. As every item will have different chances and price variations, its best that this is implemented in all classes that implement Tradeable. Otherwise, the action handling selling and buying would have countless If-else statements to handle all tradeable items.
- Created TradeableItem class: child of Items
    - A wrapper class for Item, adds price field and helps identify a tradeable item from a non-tradeable one by using capabilities
    - Adheres to DRY as Item's functionality is extended instead of being re-written
    - Constructor takes price in, and the price set for each TradeableItem child is the price the player sells it to other merchant Actors. Since the player is the main Actor, coding in this method means we don't have to recode prices when player sells to other merchants.
- Created TradeableWeaponItem class: child of WeaponItem
    - A wrapper class for WeaponItem, adds price field and helps identify a tradeable item from a non-tradeable one by using capabilities
    - Adheres to DRY as Item's functionality is extend instead of being re-written
    - Since WeaponItem extends Item, but because its an engine class, I can't overwrite WeaponItem to extend from TradeableItem. Hence, I had to make this class. This is slightly repetitive, however, it's the best option with the assignment restrictions in place.

    - Constructor takes price in, and the price set for each TradeableWeaponItem child is the price the player sells it to other merchant Actors. Since the player is the main Actor, coding in this method means we don't have to recode prices when player sells to other merchants.

- Created a Trader class: child of Actor
    - Adds implementation to allow buying and selling from any Trader actor by using Trader capability enum.
    - Adheres to DRY, if we were to create a new merchant actor, we could simply extend from Trader without copying code. Keeps future implementation in mind.
    - Has a function (getItems()) that checks the player's and trader's inventory for tradeable items and creates a list of possible items to sell and buy .
        - Downcasting is used convert Item to Tradeable type. Downcasting is only used once and minimized as much as possible. However, it is a necessity as otherwise Item doesn't have the functionality and field to handle pricing.
- Created a Traveller class: child of Trader
    - Follows DRY as implementation is already in Trader, hence, Traveller doesn't need to define any functionality.
    - Adds items that the Traveller can sell to inventory
- Created TradeCharacteristics enum class:
    - Defines how the item is traded and if it can't be scammed
    - All TradeableItem and TradeableWeaponItem would define the type of scam (if any) in their getScamType() method where it is NOT_SCAMMABLE by default.
    - Used in SellAction to modify price based on the TradeCharacteristics. For example, the traveller steals the RefreshingFlask, while it steals runes for BroadSword.
- Created SellAction class: child of Action
    - An action that allows an actor to buy a Tradeable from another actor
    - Checks if an item's price can be modified by chance, allowing for any TradeableItem to be modified/scammed.
    - Follows DRY as only one class is needed for buying/selling functionality regardless of which actor is selling or buying

- Modified Player to have Trade capability

Pros:

- Tradeable interface can be used as a data type, meaning if other tradeable classes are added in the future (ex: TradeableArmor), code doesn't need to be modified

- Each Tradeable item type has a setter, allowing different merchant Actors to have different selling prices for Items. This allows me to use the same class without changing code, hence, following DRY.

- TradeableItem & TradeableWeaponItem extend Item/WeaponItem and utilise the base code given as much as possible to ensure functionality is implemented within the given environment

- Very efficient and concise code for Traveller with only a constructor.

Cons:

- Downcasting used in Trader to convert Item to Tradeable type, however, as pointed out earlier, downcasting was the only possible way to access price and methods of Tradeable as they are not present in Item. Furthermore, downcasting to minimized and only used in one instance so that its easily trackable.

Extendibility:

- Trader allows multiple merchant/seller actors to be created in the future with little to no effort. This follows OCP as child classes of Trader are also open for modification without changing Trader itself.

- New merchant classes simply need to add items to be sold in inventory, making for very efficient code

- Adding additional methods to Tradeable interface ensures all children must implement, minimising chances of logic errors where some tradeable classes have methods but others don't.

- Each tradeable class handles its own price modification, meaning the functionality of each item can be unique and doesn't rely on any other class which adheres to single responsibility principle (SRP).