Item

Pokeball extends Item. Pokeball concrete class inherits Item abstract class. Pokeball has an attribute of Pokemon type because each Pokeball will contain a single Pokemon. There is an infinite number of Pokeball CaptureAction will instantiate a Pokeball with the corresponding target of the CaptureAction if CaptureAction is successful.

CaptureAction---<<create>>--->Pokeball. CaptureAction will instantiate a Pokeball with the corresponding Pokemon of the CaptureAction. The Pokeball with the Pokemon will be instantly added to the Player's inventory (ArrayList of item) to prevent any other Actors from picking up the Pokeball.

ReleaseAction---<<remove>>--->Pokeball. ReleaseAction will remove a Pokeball with the corresponding Pokemon from the Player's Inventory. The Pokeball will not be dropped to prevent other Actors from picking up the Pokeball.

Pokeball---<<contains>>--->Pokemon. Each instance of Pokeball will have an instance of Pokemon inside it. Pokeball has an association relationship with Pokemon.

Pokefruit extends Item. Pokefruit concrete class inherits Item abstract class. Pokefruit has an attribute of Element which is a Enum to determine which type of Pokefruit as there are 3 types of Pokefruit which is Fire Pokefruit, Water Pokefruit and Grass Pokefruit. Player will be able to pick up or drop the Item because of the inherited method from Item abstract class.

Pokefruit---<<has>>--->Element. Each Pokefruit has a single element as its attribute to determine which type of Pokefruit it is. There is a association relationship between them.

FeedAction---<<removes>>--->Candy. FeedAction removes the selected pokefruit from the Player's inventory. FeedAction knows about Candy.

Candy extends Item. Candy concrete class inherits Item abstract class. Candy will be instantiated upon a successful CaptureAction and dropped to the ground at the location where CaptureAction succeeded. Players will be able to pick up or drop the Candy because of the inherited method from Item abstract class.

CaptureAction---<<create>>--->Candy. CaptureAction will instantiate a single Candy and drop it at the location of the CaptureAction. There is a dependency relationship between them.

TradeAction---<<remove>>--->Candy. TradeAction will remove Candy from the Player's inventory on a successful trade. The amount of Candy removed depends on the Item involved in the TradeAction.

While designing the Items, the Pokeball, Pokefruit and Candy all inherit from the parent abstract Item class. This is to allow every class that inherits Actor abstract class to be able to store these items with the ArrayList of items inherited in Actor abstract class. This satisfies the Open-closed Principle as Pokeball, Pokefruit and Candy all can be extended without having to modify the Item class and they can have their own details in their subclasses.

If we tried to create Pokefruit, Pokeball and Candy as standalone classes, it would violate the Open-closed principle. This is because the Actor abstract class that contains an ArrayList of Items as it attributes needs to be modified with a separate list for each item when extending to its subclasses as a list of items cannot be used in this case.

NurseJoy

NurseJoy extends Actor. NurseJoy extends the actor abstract class and has an ArrayList of items which acts as the items she is tradable with. NurseJoy has a HashMap to determine the price of the items with Item as key and price as value.

NurseJoy---<<has>>--->Item. NurseJoy has another HashMap to determine the price of the items with Item as key and price as value. A single NurseJoy can have a Map of unlimited items.

TradeAction extends Action. TradeAction extends the Action abstract class. This is to allow the subclasses which inherit the Actor class to be able to access this class through ActionList in the Actor class that they inherited from. On a successful trade, the targeted item to trade will be instantiated and picked up by the Player immediately. On a failed trade, an error message will be printed.

TradeAction---<<has>>--->NurseJoy and **TradeAction---<<has>>--->Player.** TradeAction has an attribute of NurseJoy as NurseJoy and Player are the only characters that are allowed to trade. Specifying the attribute as NurseJoy and Actor allows the class to differentiate between the inventory of the trader(NurseJoy) and the inventory of the Player.

TradeAction---<<has>>--->Item. TradeAction has an attribute of Item as it allows the user to choose an item from the trader's(NurseJoy) inventory as the trade target. The TradeAction and Item have an association relationship.

TradeAction---<<remove>>--->Candy. TradeAction will remove Candy from the Player's inventory on a successful trade. The amount of Candy removed depends on the Item involved in the TradeAction. The relationship between TradeAction and Candy is shown through the relationship between TradeAction and Item.

TradeAction---<<knows>>--->PickupAction. TradeAction knows about PickupAction because the Player will pick up the Item immediately after a successful trade.

The implementation above adheres to the Single Responsibility Principle and Open-closed Principle of the SOLID principles. This is because we can avoid using multiple if-else statements for the TradeAction to determine the price of each item and the result of the trade. The price of each item can be determined by using a HashMap in NurseJoy we are able to check on every type of item and their respective price without having to change any attributes in the Item abstract class. Single Responsibility Principle can be fulfilled because the price for trading is contained within the NurseJoy and TradeAction class.

An alternative for the implementation of NurseJoy is adding an attribute of price for the Item abstract class so any of its subclasses will have a price for trading. However, this implementation method is less desirable as it violates the Single Responsibility principle and as the attribute only used for trading is now inherited by all the item's subclasses and some of the subclasses of Item are not tradable. Besides that, this implementation requires modification on Item abstract class for it to work and Open-closed principle does not allow that.