Design rationale:

Pokefruit**,** Candy **and** Pokeball **extend** Item. Generalisation allows Pokefruit , candy and Pokeball to inherit properties of the Item class in game engine. Similarly, Player **and** NurseJoy **extend** Actor**.** Generalisation allows reusability of code by reducing repetition of shared attributes that define a player and a nurse. In both cases, generalisation allowed us to distinguish responsibilities of classes, thereby exhibiting the Single responsibility principle.

Pokefruit class implements a Pokefruit enumeration class. Pokefruit types – fire, water and grass – are staying constant in the game. As such implementation of enumeration for Pokefruit types will decrease the likelihood of errors due to typos and make it easy to add more types of Pokefruits in the future should the game develop.

Pokeball **has an association with abstract class** Pokémon**.** Each Pokeball stores one Pokémon as per game specification. Pokemon **extends** AffectionManager**.** AffectionManager updates players' affection points depending on the type of fruit they feed to the Pokemon. Candy **extends** CandyManager**.** CandyManager helps manage the candy balance i.e., updating candy balance after picking new candy, dropping candy or trading with nurse joy. In both these cases, construction of an additional class with a unique purpose – managing affinity and managing candy – ensures that the design abides the single responsibility principle.

Player **has an association with** Candy. Each player has more zero or more candies stored in his inventory for trading. Player **has a dependency with** DropItemAction, PickItemAction and TradeAction. The player implements these methods to pick up, drop or trade candy. **Interface** Tradeable **implements** TradeAction. TradeAction **has a dependency with** Tradable. It allows one item to be traded for one or more items. Pokefruit **and** Pokemon **implement tradable interface** as they both qualify as items eligible to be traded off for candies. Alternatively, Pokefruit, could have been extended from Pokemon, or vice versa, however, we chose not to go with approach as it increases the likelihood of coupling. Instead, the addition of the Tradable interface ensure that the tradable methods are implemented without any unforeseen coupling in the methods. Therefore, our proposed design follows the Liskov's substitution principle. Another poor alternative approach would be to add dependencies between Tradable items and TradeAction, however, this approach would make it harder in the future to increase the scalability of the game if more tradable items arise. Our approach also allows us to reduce the number of dependencies in the design and thus aligns with dependency inversion principle as it allows easy the reusability of the code by allowing addition more tradable items without interfering with the implementation of the broader method.

NurseJoy **has a dependency relationship with the** TradeAction which allows her to trade Pokefruit or Pokemon for candies. It also allows NurseJoy to easily offer items she is able to trade depending on the candies the player has.