Elements:
The pokemon element should be handled by an Element enum class and an ElementHandler to compare, check compatibility and handle all element-related interactions. This is because elements are separate from Environment and Pokemon, but both interact with and thus would be reasonable to be processed with an enum class, therefore, ensuring integrity and synchronisation. Ground and Pokemon class will save an element attribute to store the elements of the object. This is already implemented in the game package and engine package by assigning items, actors and ground a CapabilitySet object.

Environment:
The ground class will be an abstract class with a subclass of different types of ground (Lava, Crater,...), and for some subclass in ground is spawning ground where spawn the pokemon and drop pokefruit, so these subclass will implement the interface SpawningGround and the subclass of spawning grounds will generate to the superclass Ground so there will a generalisation between the superclass Ground and interface SpawningGround. Also the SpawnPokemonAction has a dependency relationship between SpawningGround as well and allows the action of spawning pokemons and dropping poke fruit.

so there will be an implementation between the interface SpawningGround and spawning ground(Crater, Waterfall,…), and Pokefruit will have a dependency with spawning ground.

Pokemon:
The pokemon subclass is the superclass for the pokemon (Charmander, Squirtle, …) which is an abstract class of actor class and can contain additional attributes and methods unique for Pokeons. The Item subclass Pokeball can maintain a Pokemon class object to be summoned by SummonPokemonAction.execute().

The Special attack and intrinsic attack of the pokemon will be handled using the weapon system. As the game engine already implemented attack action using weapons and items, implementing special attacks as a weapon and an item makes the design simpler and more compatible with the system. The new Abstract class SpecialWeapon is considered an Item auto-equipped when being in the environment with the same element type. It contains 3 instantiable classes according to the pokemon types as in the requirements, which is capable of AttackAction.execute() sequence as being of Weapon interface and an Item.

The SpecialAttackManager class will handle the above special attack by assigning the special weapons to the pokemons' respective inventory when they are in environments with the same attribute. It is to avoid all Ground or Pokemon classes having to maintain an attribute to execute this functionality.

The affection mechanism will be handled by the AffectionManager class. Though this is a violation of the Single Responsibility principle, it would make the design simpler and more succinct if a single class with a single instance per game were to handle the affection mechanisms than having all Pokemon bearing an affection attribute or class.

Game time:
This game has game time inside, so we need a game perception manager to manage the time, and the class TimePerceptionManager will implement the method in the interface

TimePerception, and also associate to the enum class of game period, because the game time is switched between only day and night every five turns, so it will have two subclass DAYTIME and NIGHTTIME. Also ground and actor will have a time perception of the world, so these two classes will be associated with TimePerceptionManager

Items:
Pokefruit, Candy and Pokeball extends Item and Player and Nurse Joy extend Actor and Pokefruit and Pokeball extend abstract class TradableItem. Generalisation allows child classes to inherit properties of the parent class in the game engine. In all the instances mentioned, Inheritance allows reusability of code by reducing repetition of shared attributes that define a parent class i.e., Item, Actor and TradableItem. Generalisation makes it easier to distinguish responsibilities of classes, thereby exhibiting the Single responsibility principle.

We decided to implement an abstract class TradableItem which stores the attributes required for a trade. The TradeAction class has an association with TradableItem. This class is responsible for the transaction that takes place between Nurse joy and Player shown with the dependencies between the two. Alternatively, Pokefruit could have been extended from Pokemon, or vice versa, however, we chose not to go with this approach as it increases the likelihood of coupling. Instead, the addition of the Tradable class to ensure that the tradable methods are implemented without any unforeseen coupling in the methods. Therefore, our proposed design follows 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 reusability of the code by allowing addition more tradable items without interfering with the implementation of the broader method.

Pokeball has an association with abstract class Pokémon. Each Pokeball stores one Pokémon as a Pokeball cannot exist without a Pokemon inside. AffectionManager demonstrates an association with abstract class Pokemon. AffectionManager updates players’ affection points depending on the type of fruit they feed to the Pokemon.
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 CandyManager. Each player has one CandyManager. 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 an association with Tradable. It allows one item to be traded for one or more items.

Pokefruit and Pokeball are subclasses of the Item classes in order to allow them to have actions and capabilities. Pokeball will contain the CapturePokemonAction, ChangeAffectionAction and SummonPokemonAction in their ActionList which will handle its interaction with Pokemons. Similarly, Pokefruit also contains ChangeAffectioAction so that the execution is compatible with the game engine. The elements of the Pokefruit are implemented through the CapabilitySet in the game engine.

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