Assignment 2 Analysis

1. *How does your design implement the four pillars of OOP (abstraction, encapsulation, inheritance and composition, and polymorphism)?*

Our abstract class, PartyMember, defines abstract methods which are implemented in both Pokemon and Egg subclasses. This demonstrates the use of abstraction within our design.

For encapsulation, our classes implement many private methods not available to the user and variables which are accessed by getters and setters. A good example of encapsulation within our design is the _level_up() method in the Pokemon class. This method is not called directly by the end user, rather the Pokemon "levels up" as it gains experience (using the add_xp() public method). This method is hidden from the public interface and is integrated into the use of the class.

Our design also incorporates inheritance, as the Pokemon and Egg classes are subclasses of PartyMember, and they inherit all of PartyMember's properties and methods.

As for composition, we decided to take our design in a slightly different direction than expected. Our PartyManager and PartyMember are in a composition relationship, because a PartyMember is created only by PartyManager, and not independently. This means one "Party" can have many PartyMembers, and without a "Party" (A PartyManager instance), you cannot create any PartyMembers (In other words, you cannot have a party member if you do not have a party to put them in).

We incorporate polymorphism in many methods throughout our project, most notably in our PartyManager class, within the "get member" methods (get_all_members_by_elemental_type(), get_member_by_type(), etc). In these methods we construct a list of all the available members from the PC storage and the party, and then access their member_type() method to determine that member's type (Egg or Pokemon). Accessing this method is a demonstration of polymorphism because it is implemented in both Egg and Pokemon, and depending on the class type, returns a different value.

2. *Why are your classes good abstractions (i.e., models) of the real-world entities they represent?*

Well, our classes aren't *technically* real world entities, but we did a fairly accurate depiction of the video game entities that they do represent. Essentially, one PartyManager entity is one player in the the game "Pokemon." The player can have a party up to 6 Pokemon, as well as a PC storage unit which stores their Pokemon when they are not in their party. Its possible for the Pokemon to be an Egg instead of a Pokemon, which hatches after a certain amount of steps. Once the egg hatches, it's replaced in their party as the Pokemon that was growing within that

egg. Pokemon in storage and in the party have a number of stats that are represented by the numerous properties of each instance. Most are automatically generated, for example base health points and base battle stats.

The functionality that is included in the real Pokemon games is not fully represented in our class, but we do implement some features such as egg-hatching and leveling up.