Assignment 2 – Analysis

Group 32: Jimmy Ho – Josh Wong

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

Abstraction

Abstraction describes the act of information hiding. Through abstraction, users are only presented with the things that they need to be concerned about. Abstraction can be defined as the process of encapsulating information with separate public and private interfaces. Our design implements abstraction through using public and private variables and methods.

Encapsulation

Encapsulation is the act of hiding functional details, and is used to restrict access to methods or variables. The public interface is important to design as it is difficult to change later. Our design implements encapsulation through allowing the user to only be able to use public attributes and methods, while hiding private variables and methods.

Inheritance and Composition

Inheritance allow new objects to take on the properties of existing objects. This can be done by a child class inheriting methods and properties of a parent class. In the case of our project, we implement inheritance by allowing the GymLeader and RegularTrainer classes to take the methods and properties of the AbstractTrainer.

Composition can be defined as collecting several objects to compose a new object. Aggregation is similar to composition but unlike composition, the child object can exist independent of the parent. In our implementation, AbstractTrainers can exist independently of TrainerManager through an aggregate relationship.

Polymorphism

Polymorphism describes the ability to redefine methods form the base class in the derived class. In our design, the AbstractTrainer super class has abstract methods that are defined by the derived classes GymLeader and RegularTrainer. These methods include get_type() and get_details().

Question 2: Why are your entity classes good abstractions (i.e., models) of the real-world entities?

Our entity classes are good abstractions of the real world because they only present users with the information that they need to see. For example, TrainerStats shows the important trainer statistics that a developer or user would need to know, without showing how the information was generated. The TrainerManager class gives all the functionality needed to manage trainers

without extra methods, since AbstractTrainer, GymTrainer, and RegularTrainer classes take care of the specific trainer details.