Name: Quang Vinh Le

ID: 104097488

Object-Oriented Programming (OOP) Principles

Task 2

1. Explain the four principles of object oriented programming. For each of the principles, refer to a piece of work that you have completed for this unit and explain how it demonstrates the principle.

- Object-oriented programming (OOP) relies on four core principles: encapsulation, inheritance, polymorphism, and abstraction. Now let's delve into each of these principles with explanations and examples:

1. **Encapsulation:** Encapsulation in object-oriented programming refers to the practice of wrapping data (attributes) and the methods (functions) that operate on that data into a single unit, which is the class. This concept encourages data hiding and controlled access. In my code, The "Arena" class demonstrates encapsulation. It encapsulates the private field "_enemie"s, which is a list of enemies, along with methods like "AddEnemy", "Attack", and "AttackAll". These methods provide controlled access to the "_enemies" data and ensure that external code cannot directly manipulate it, which is a key aspect of encapsulation.
2. **Inheritance:** Inheritance is a principle in OOP where a new class (a subclass or derived class) can inherit properties and behaviors from an existing class (a superclass or base class). It promotes code reuse and the creation of specialized classes based on a common template. In my code, the "RegularEnemy" and "InvincibleEnemy" classes inherit from the "Enemy" class. The "Enemy" class defines the common structure for all types of enemies, including the abstract method "GetHit". By inheriting from the "Enemy" class, "RegularEnemy" and "InvincibleEnemy" inherit the common behavior of being able to be "hit" in some way, while they can also extend or override this behavior to suit their unique characteristics.
3. **Polymorphism**: Polymorphism is the capability of objects of different classes to respond to the same method or message in a way that is specific to their individual types. It enhances flexibility and dynamic behavior in your code. In my code, polymorphism is demonstrated through the "GetHit" method. While the "Enemy" class defines "GetHit" as an abstract method, the concrete implementations of "GetHit" in the "RegularEnemy" and "InvincibleEnemy" classes allow me to invoke "GetHit" on an instance of "Enemy" without knowing its exact type. This dynamic dispatch ensures that the appropriate version of "GetHit" is called based on the actual type of the enemy, allowing different enemy types to respond to damage in their own unique ways.

4. **Abstraction**: Abstraction simplifies complex systems by emphasizing what an object does while hiding how it does it. It involves defining high-level interfaces or abstract classes, promoting code reusability, and providing a clear, simplified view of objects. In my code, the “Enemy” class serves as an abstract representation of an enemy, defining the essential behavior that all enemies share, which is the “GetHit” method. By making Enemy an abstract class and providing a common interface for all enemy types, I am abstracting away the specific details of each enemy type while ensuring they all adhere to a consistent set of behaviors and attributes.

-These principles collectively form the cornerstone of OOP, enabling developers to create structured, modular, and highly adaptable software systems.