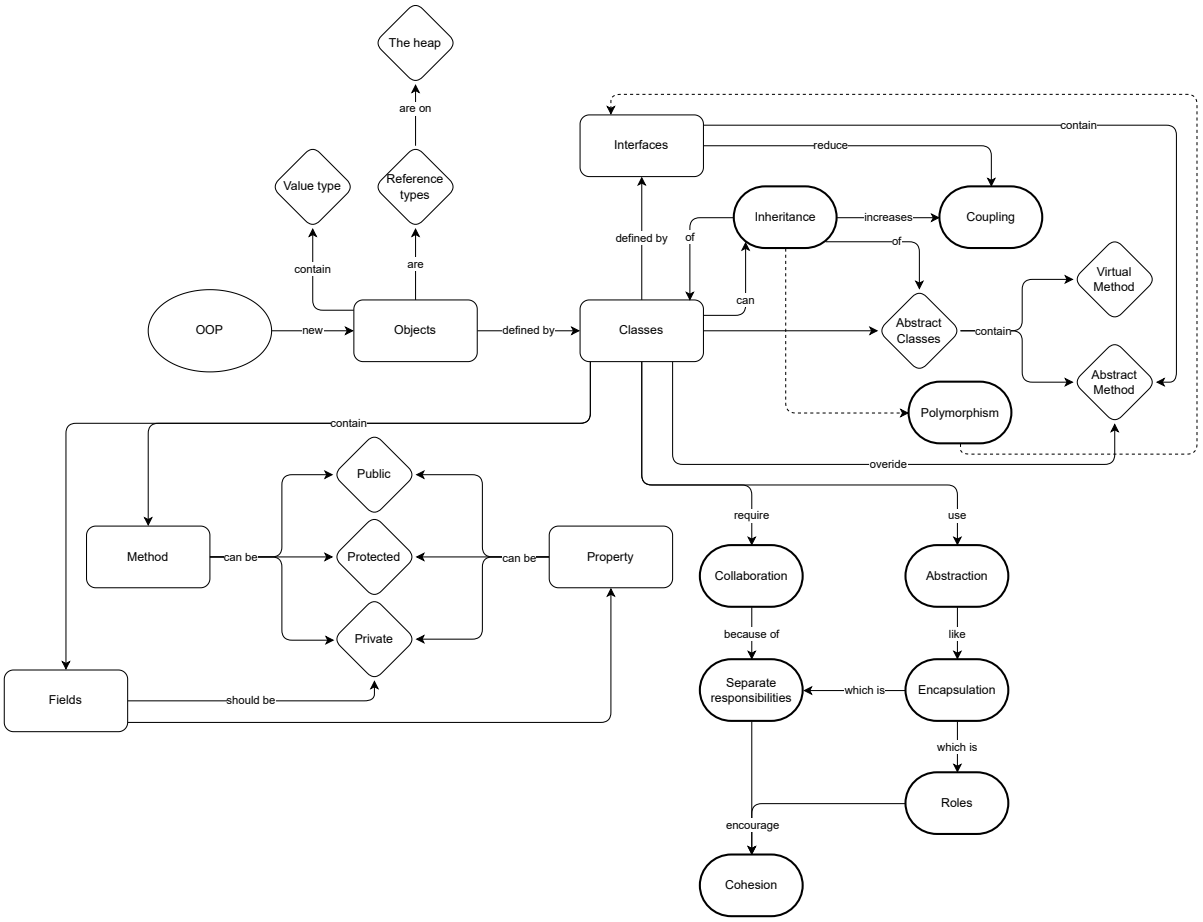
Name: Tran Duc Anh Dang Student ID: 103995439

Object-Oriented Programming Report

A programming paradigm known as object-oriented programming, or OOP, puts more emphasis on the objects than the logic and function that go into the creation of software. Since programmers may alter and arrange items inside programmed, this kind of programming is appropriate for complex, often updated, and large-scale applications. The image below depicts the gist of OOP ideas.



The foundation of object-oriented programming is, as the name implies, objects. The concept of OOP objects is similar to real-world objects, which have knowledge and capabilities. When applying this concept to programming, programmers build objects using classes. A class has fields for data representation and methods for object functionality. To put it simply, a class serves as a model for items that exhibit comparable traits and behaviours. Many object-oriented languages, including C#, utilise the term new to generate an object from a regular class; this object is referred to as an instance.

There are 4 principles: Abstraction, Encapsulation, Inheritance and Polymorphism

1. Abstraction: The principle of abstraction states that objects should merely know certain facts and do certain actions, leaving out the specifics of how they learn and operate. This is also known as "designing the items." To be more precise, classifications, roles, responsibilities, and collaborations are the four fundamental properties of objects that abstraction helps establish. The definition of the thing is referred to as classification. Each programme object need to have a function, a purpose, and

the duties necessary to carry out that role. The relationship between items is also defined by abstraction, and this is what partnerships mean.

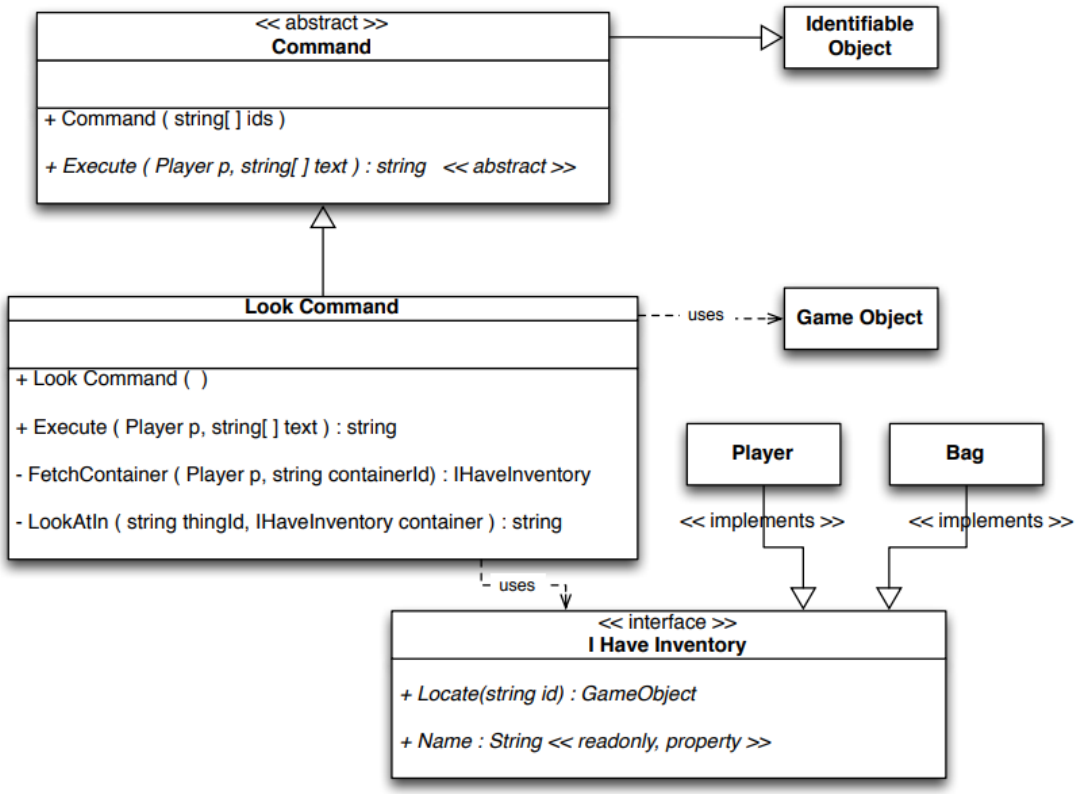
2. Encapsulation: Encapsulation is the concept of managing the information accessibilities of things by encapsulating data and functionality inside those objects. Although methods and properties allow for the usage of objects, the internal workings of these must be secured. This is because using things only requires understanding their function, and tampering with them unintentionally can have devastating consequences. Access to programmed classes can be changed by using access modifiers.

3. Inheritance: In essence, inheritance creates a class from another class. All members of the parent base class may be inherited by the child derived class. To increase code reuse, developers utilise inheritance to construct a family of classes. Abstract classes and interfaces are introduced in relation to inheritance. Because they partially include members without any particular logic, abstract classes are designed to be inherited. Although classes may also derive from interface, the goal of utilising interface is to give classes that don't have the same features as the class family additional functionality. Although an interface can have numerous base classes, a class can only have one base class.

4. Polymorphism: The concept of polymorphism relates to a single object with many forms, which is achieved using inheritance. This means that the child class has both its type and its parent’s type, and developers can change its type flexibly throughout the program.

These principles have been used in earlier weekly activities to carry out straightforward yet efficient programmed. The Case Study, SwinAdventure, is an example.

Name: Tran Duc Anh Dang Student ID: 103995439



Abstraction is used in the UML diagram above from SwinAdventure to organise the classes, such as Command should execute. The development of the LookCommand, which not only executes but also fetches a container and looks at objects, makes use of encapsulation. However, only one method is made available to the public while the others are hidden because execute is the Look Command's principal function. Since a family of objects has been developed from the Identifiable Object class using inheritance and polymorphism, the use of objects is more flexible and adaptable. The person acts as a game item upon being located, and acts like IHaveInventory when locating other items.

Roles: An object's functions and what has been assigned to, expected of, or required of it are represented by a role, which is a reference to the system object. A role is an interchangeable term for a group of connected duties.

Responsibility: Responsibilities is an employee or an obligation of the classifier. The duties placed on an object's conduct are connected to responsibilities. These include carrying out an action or having certain knowledge.

Coupling: The term coupling describes the reliance on and connection between the two groups. If there is little connection between the two classes, updating the code in one won't effect the other. On the other hand, strong coupling should be avoided in software development and OOP design in general since it makes it harder to maintain and modify the code because the two classes are so closely coupled.

Name: Tran Duc Anh Dang Student ID: 103995439

Cohesion: A class's cohesion is defined as its functionality. Low cohesiveness in a class indicates that it contains a wide range of methods and functions without any focus on what it should be doing. High cohesiveness, on the other hand, indicates that the class is focused on what it should be accomplishing, which was the original purpose of the class.

Therefore, a high Cohesion and low Couple should represents a good OOP design structure.