OOP Principles

COS 20007 – Oriented Object Programming

Nguyen Manh Dung

20/5/2023

Oriented Object Programming Principles

Oriented Object programming have four keys concepts, which is:

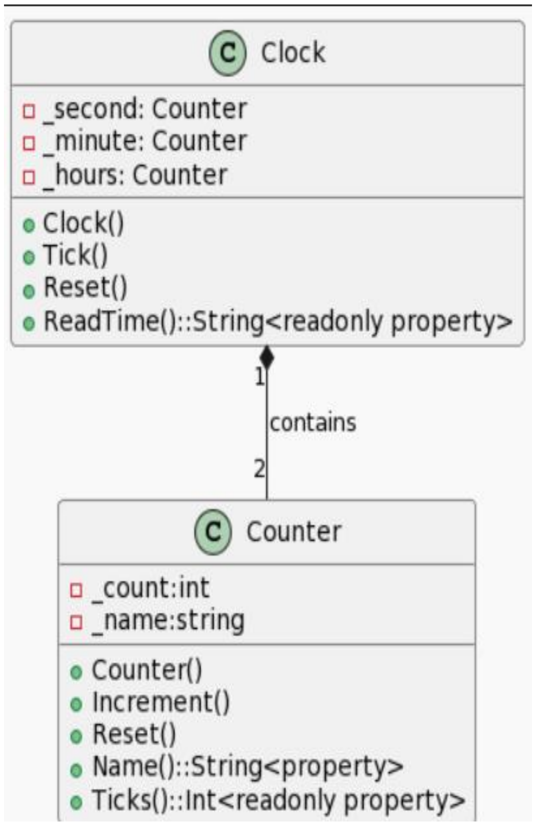
- **Encapsulation**
- **Abstraction**
- **Inheritance**
- **Polymorphism**

1.Encapsulation

Encapsulation is the process of combining a method and a data member into a single entity (or class).

Encapsulation is attained when each object keeps a private state inside of a class. Other objects cannot directly access this state. Users can only utilize a restricted number of methods, which are public functions.

As a result, an object manages its own state using methods, and no other class is permitted to interfere unless explicitly permitted. If you want to interact with the object, use the available methods. You cannot, however, change the status (by default).



For instance, we have a Clock and a Counter class in the Clock program that we created in pass task 3.3. _count and _name, two private variables, serve as the Counter class's state variables. Additionally, it has three functions that utilise these variables: Counter(), Increment(), and Reset(). While public methods like Increment() and Reset() can be used in the main application to change the internal state of the Counter class, you cannot directly alter the values that _count and _name retain.

Encapsulation is demonstrated in this case.

2.Abstraction

The principle of abstraction emphasizes limiting the knowledge and functions of objects without considering their specific circumstances of acquisition and application. It is often referred to as "designing the items." Abstraction helps construct objects with four key qualities: classifications, roles, responsibilities, and collaborations. Classification defines how something is defined, while each software object has a role, a function, and the responsibilities necessary to fulfill that role. Collaborations indicate that the relationship between objects is also determined through abstraction.

3. Inheritance

Inheritance is the process of creating a class based on another class. The child derived class inherits every member of the parent base class. Developers use inheritance to establish a family of classes and promote code reuse. Abstract classes and interfaces are introduced in relation to inheritance. Abstract classes are meant to be inherited as they partially define members without any specific logic. While classes can also inherit from interfaces, interfaces provide additional functionality to classes that may not share the same characteristics as the class family. A class can have only one base class, but an interface can be implemented by multiple classes.

4. Polymorphism

Polymorphism refers to the ability of a single object to exhibit multiple forms. This means that a child class can change its type at any point during the program and possesses both its own type and the type of its parent. These principles have been applied in previous weekly events to execute simple yet effective programming. An example is the Case Study, SwinAdventure.

Role

A role is a reference to a system object; it describes the function of the object and what has been assigned to it or is expected of it. A role is an interchangeable term for a group of connected duties.

Responsibilities

The classifier's obligation or contractor is responsibility. The duties placed on an object's behavior are connected to responsibilities. These include carrying out an action or being aware of certain facts.

Collaboration

Collaborations are procedures where two or more things share information. They consist of interactions between roles and objects (and occasionally both).

Coupling

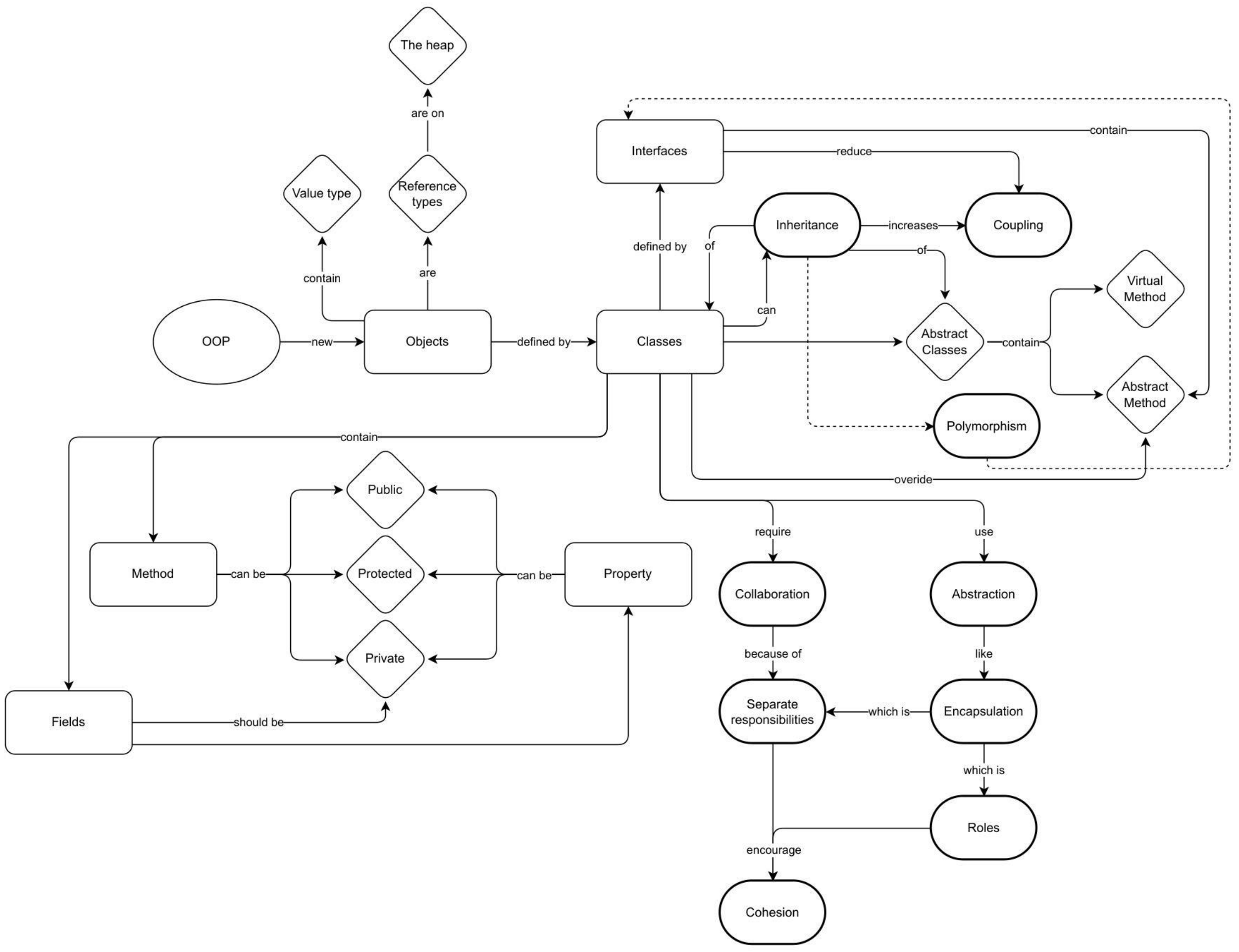
The term coupling describes the reliance on and connection between the two groups. If there is little connection between the two classes, updating the code in one won't effect the other. On the other hand, strong coupling should be avoided in software development and OOP design in general since it makes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

Cohesion

A class's functioning is referred to as its cohesion. Low cohesiveness in a class refers to a class's vast range of methods and functions, which lacks any focus on what the class is intended to do. As opposed to this, strong coherence indicates that the class is focused on what it should be accomplishing, which was the original purpose of the class.

- ⇨ Good OOP design should have high cohesion and low coupling

Concepts Map


The heap
are on
Value type
Reference types
contain
are
OOP
new
Objects
defined by
Classes
Interfaces
defined by
reduce
contain
Inheritance
increases
Coupling
of
of
can
Abstract Classes
contain
Virtual Method
Abstract Method
Polymorphism
overide
contain
Method
can be
Public
Protected
Private
can be
Property
Fields
should be
require
use
Collaboration
Abstraction
because of
like
Separate responsibilities
which is
Encapsulation
which is
Roles
encourage
Cohesion