COS 20007

Task 6.2

Duc Thuan Tran

104330455

I. Introduction

OOP (Object Oriented Programming) is programming paradigm that focuses on arranging code using the concept of "objects.". Every object contains both information in the form of attributes and functionality in the form of methods. OOP allows user to organize code in a way that allows for easy maintenance, growth, and recycling.

II. Key Principles

1. Encapsulation

Definition: Encapsulation is the process of hiding an objects inside activities from outside access and exposing only a public interface for interaction with other objects. This idea ensures that an object's inside state is changed in a controlled way, which aids in code reliability.

Example: Imagine we are crafting a user profile for a social media application. We keep important data in the "UserProfile" class, such as the user's name, email, birthdate, and list of friends. We use encapsulation to protect this information from unauthorized access by identifying the data properties as private. We then make public methods available for updating user information, controlling their friend list, and retrieving their name. This approach to encapsulating ensures the security and honesty of user data while allowing for accessing and changing.

Relate the principles to program: Encapsulation is represented in the Drawing.cs (4.1) by the private values _shapes and _background, which are encased by public properties such as SelectedShapes and Background. These attributes govern access to the underlying data, allowing users to retrieve or alter it in a secure manner. This ensures that the Drawing class's internal state is not directly exposed and can be maintained appropriately.

2. Inheritance

Definition: Inheritance is the process that allows one class to reuse or inherit properties and actions from another class, promoting code reuse and system organization.

Example: Imagine we are building a zoo management system. We begin by creating a base class called "Animal," which contains properties such as "name," "species," and methods such as "eat" and "sleep" We may now develop animal classes such as "Lion" and "Elephant" that derive from the "Animal" class. These specialized classes inherit the base class's general animal attributes and methods while also introducing their own distinct qualities. The "Lion" class, for example, might have a method named "hunt" while the "Elephant" class would have a method called "run" This use of inheritance allows us to reuse the common animal-related functionality while customizing it for different types of animals in the zoo.

Relate the principles to program: Inheritance is represented in the MultipleShape (4.1). The base class, Shape, defines common characteristics as well as abstract methods. MyCircle, MyLine, and MyRectangle are three derived classes that inherit from Shape and expand its capabilities with unique properties and methods for creating certain forms. By centralizing shared behaviour in the base class and allowing specialized classes to adapt it, inheritance increases code reusability and organization. This technique adheres to object-oriented concepts, allowing for more efficient code management.

3. Polymorphism

Definition: Polymorphism is the ability of objects to respond to the same function or message in different ways, which is a crucial element in object-oriented programming that improves code flexibility and extension.

Example: Imagine about a "Vehicle" hierarchy with a base class "Vehicle" and derived classes like "Car" and "Bicycle." The "StartEngine" technique is unique to each vehicle. In this case, polymorphism can be used by running the "StartEngine" function on various vehicle objects. A car's engine may require fuel injection and ignition to start, but a bicycle requires only pedalling. Polymorphism allows us to use a similar way to start the engine, but the behaviour varies depending on the vehicle type, demonstrating how objects respond differently to the same method call.

Relate the principles to program: Polymorphism is represented in the Swin Adventure (4.2). In a class hierarchy, method overriding is an example of polymorphism. For example, the GameObject class defines the "FullDescription" field, which is overridden in subclasses such as Player and Item. Though they all have the same parent class, every subclass offers a different way to implement "FullDescription." This demonstrates polymorphism, which frees objects of various classes from the need for interfaces to display unique behaviors within the same hierarchy.

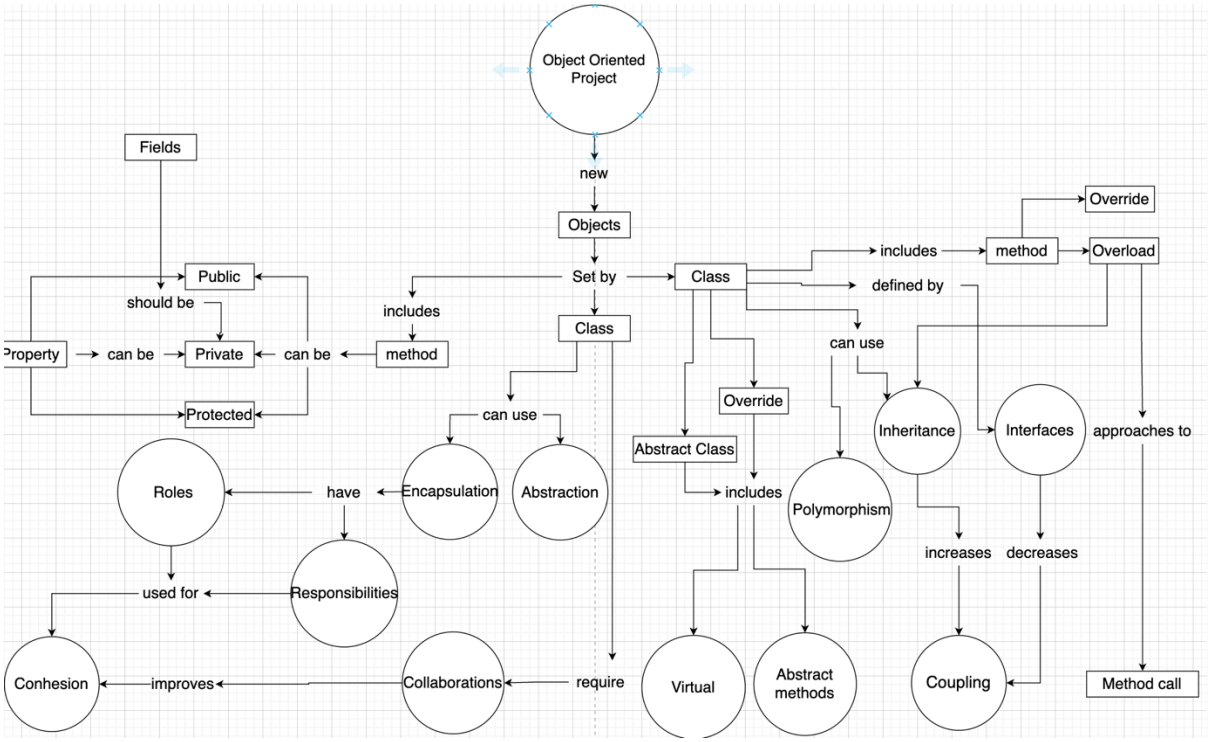
4. Abstraction

Definition: Abstraction is the process of distilling an object's core features and behaviours while leaving out the complicated execution details. This method simplifies complexity within systems, making them more understandable and approachable.

Example: Imagine we are creating a financial application. We may have different types of accounts, such as savings accounts, checking accounts, and investment accounts. To abstract the basic functionalities, we can design an abstract class named "Account" that contains essential characteristics like account number, account holder's name, and amount. Each account type, such as "SavingsAccount" and "CheckingAccount," can then derive from the fundamental "Account" class, adding its own features and functions. This abstraction allows us to concentrate on the fundamental attributes and functions of all accounts while abstracting away implementation details, resulting in a more modular and maintainable code.

Relate the principles to program: In the MultipleShape (4.1), consider drawing shapes as an example. This procedure is abstracted by the code, allowing all shapes to be drawn using a single 'Draw' method. This abstraction simplifies the Drawing class and allows us to treat shapes consistently, focusing on the 'what' (drawing shapes) rather than the 'how' (particular drawing details)."

III. Concept map


Object Oriented Project
new
Objects
Set by
Class
Fields
Public
should be
Property
can be
Private
can be
Protected
includes
method
Class
can use
Encapsulation
Abstraction
Roles
have
Responsibilities
used for
Conhesion
improves
Collaborations
require
includes
method
Override
Overload
defined by
can use
Override
Abstract Class
includes
Polymorphism
Inheritance
Interfaces
approaches to
increases
decreases
Virtual
Abstract methods
Coupling
Method call