OBJECT-ORIENTED PROGRAMMING CONCEPTS

104309099 – Justin Nguyen

1. Abstraction

Definition:

The idea of abstraction is to hide the implementation details and concentrate on the attributes' functioning in order to simplify complex systems. It enables us to build models or representations of actual processes or things that include the necessary data and behaviors. In addition to encouraging modularity and managing complexity, abstraction offers a straightforward and easy-to-use interface for working with objects.

Example:

In the 4.1 ShapeDrawing program, we do not need to know exactly the design specification of the Draw(), DrawOutline() or IsAt(pt) methods in the Shape class, we only know how the methods work by calling the methods in MyLine, MyRectangle or MyCircle. It gives us a clean separation between what we need to know (the public interface) and how the methods work (the private implemention).

2. Inheritance

Definition:

Through the concept of inheritance, a class can inherit fields and methods from another class. The class that is inheriting is referred to as the child class (or subclass), while the class that is being inherited is referred to as the parent class (or superclass). A derived class is a specialized version of the base class; it also supports the idea of a "is-a-type-of" relationship. It encourages code flexibility, adaptation, and extensibility.

Example:

In Iteration tasks, the Player class inherits functions and properties from the GameObject class such as Name, FullDescription, etc.

3. Encapsulation

Definition:

The idea of encapsulation is combining data and associated processes into a single entity. While a number of public functions and methods are available for interaction, direct access to the object's data is restricted. Other classes are unable to alter the object's data without express permission; these functions act as an interface for doing

so. Encapsulation enforces controlled access to an object's internal state, which maintains data integrity and encourages the development of modular, reusable code.

Example:

In the CounterClock task, the Clock class contains private properties such as _seconds, _hours, and _minutes and methods such as Clock(), Tick(), and Reset(). While you cannot change the values of the private properties, you can call the methods like Tick() or Reset() to change the internal state of the Clock class. It prevents accidental or unauthorized changes by enabling the integrity of the clock's data and giving the user control over how the clock's attributes are accessed and changed.

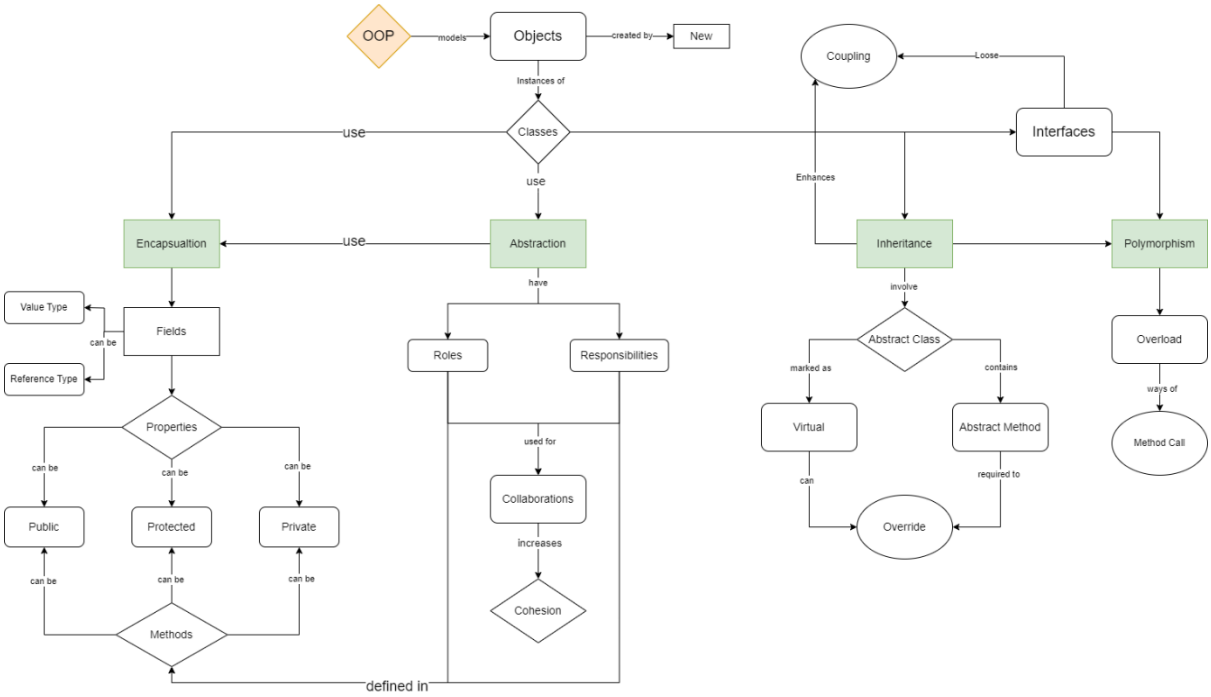
4. Polymorphism

Definition:

The idea of polymorphism enables us to consistently interact with objects irrespective of their particular kinds. It can be dynamic (achieved by overriding a method) or static (achieved by overloading a method). Because polymorphism treats distinct objects as instances of a shared interface, it allows utility classes, such collection classes, to deal with objects of multiple kinds while also providing flexibility and code reusability. This flexibility allows writing more generic and reusable code because it can handle different objects without needing to be modified or rewritten for each type, while this extensibility allows adding new functionality to the code by introducing new classes without having to modify the code of the parent class.

Example:

In the Drawing tasks, the Shape class handles methods like Draw(), DrawOutLine(), IsAt(), etc. The subclasses such as MyLine, MyRectangle and MyCircle inherit the methods from the Shape class and override the method by implement unique features for each shape. When we want to draw a shape or an outline, the corresponding methods will be called. It is possible to treat objects of different classes as belonging to the same superclass as long as the provided methods are followed.


OOP
models
Objects
created by
New
Instances of
Classes
use
use
use
Encapsualtion
Abstraction
Coupling
Loose
Interfaces
Enhances
Inheritance
Polymorphism
Value Type
Reference Type
can be
Fields
Properties
can be
can be
can be
Public
Protected
Private
can be
can be
can be
Methods
have
Roles
Responsibilities
used for
Collaborations
increases
Cohesion
defined in
involve
Abstract Class
marked as
contains
Virtual
Abstract Method
can
required to
Override
Overload
ways of
Method Call