OBJECT-ORIENTED PROGRAMMING CONCEPTS

103812143 - Luan Nguyen

1. Abstraction

Abstraction is a concept in object-oriented programming that involves simplifying complex systems by focusing on the roles, and responsibilities of properties and hiding unnecessary details that how these properties were created. It allows us to create models or representations of real-world objects or processes with only the necessary information and behaviours needed for a particular context. Abstraction helps in managing complexity, promoting modularity, and providing a clear and simplified interface for interacting with objects.

Example:

The UML diagram would be the best example for the application of abstraction that in the single class of the UML diagram, we only know the roles, and responsibilities of fields, methods, and properties on that and how they collaborate, just ignoring how complex the functions were, we only need to know that there is a method in that class and we can use that method as well as the properties or fields.

2. Inheritance

Inheritance is one of the 4 core concepts of OOP that allows the class to inherit properties and behaviours from another class. It promotes code flexibility, adaptability and extensibility, and supports the concept of "is-a" relationship, where a derived class is a specialized version of the base class.

Example: In Iteration tasks, Player class is the child class of the GameObjects class that inherit all the function and properties from GameObjects class such as Name,

FullDescription,….

3. Encapsulation

Encapsulation is one of 4 fundamental concepts in object-oriented programming (OOP) that involves **bundling data** and related operations into a single unit. It ensures that the internal state of an object is kept private within its class. Encapsulation acts as a **protective barrier**, preventing direct access to the object's data, but allowing interaction through a set of public functions or methods. These functions serve as an interface for manipulating the object's data, and other classes cannot modify it without explicit permission. Encapsulation provides data integrity and promotes modular, reusable code by enforcing controlled access to an object's internal state.

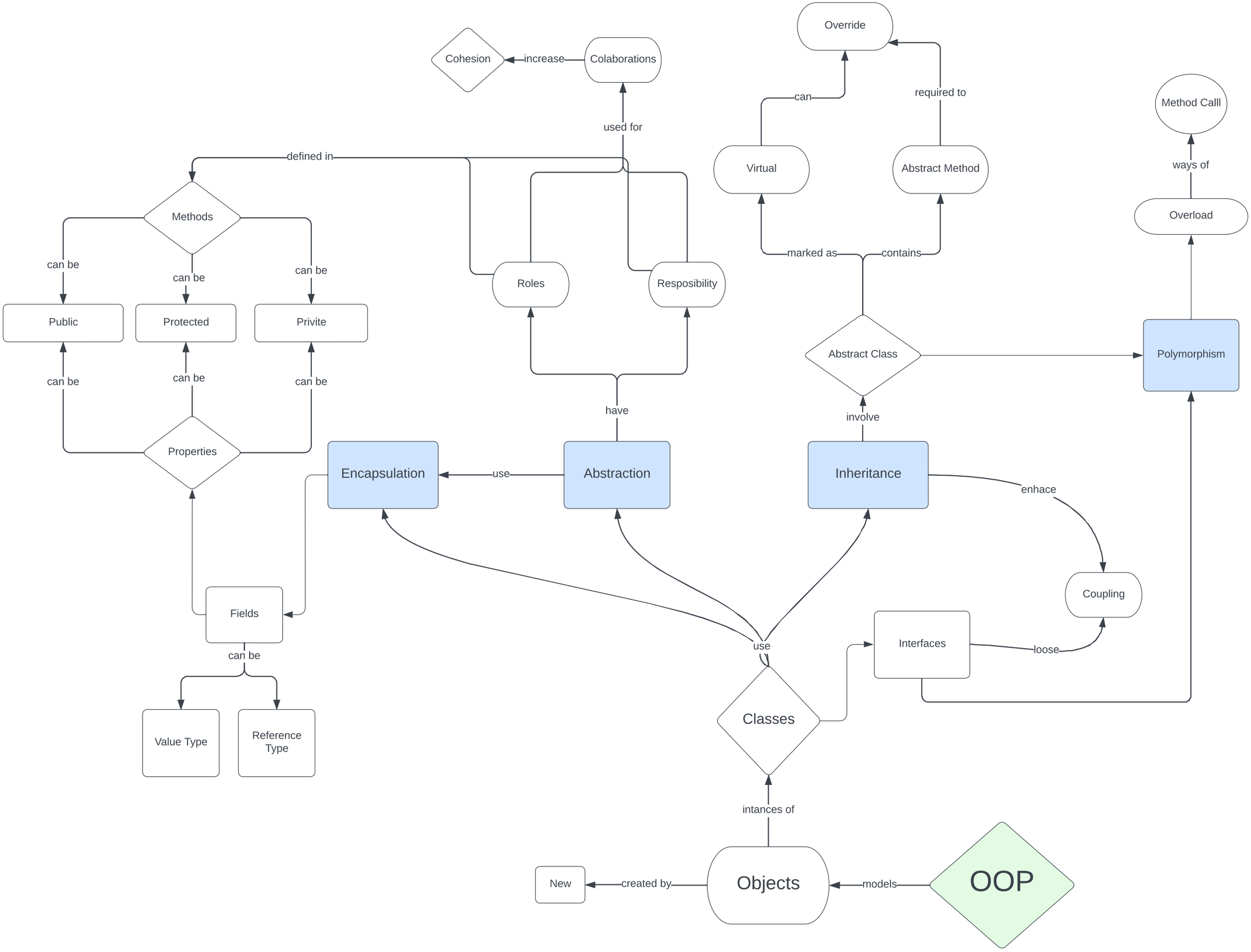
Example: Consider the Clock Task in the portfolio, the private properties such as _hours, _minutes, _ seconds are created that can be use in other public functions it also ensures that the integrity of the clock's data is maintained. It allows for better control over how the clock's properties are modified and accessed, preventing unintended or unauthorized changes.

4. Polymorphism

Polymorphism is one of the core concepts of OOP which means “many forms”. It allows us to interact with objects in a consistent way, regardless of their specific types. It can be static, achieved through method overloading, or dynamic, achieved through method overriding in inheritance. Polymorphism provides **flexibility**, code **reusability** by treating different objects as instances of a common interface, and **adaptability** allows utility classes, such as collection classes, to work with objects of different types. This flexibility allows writing more generic and reusable code, as it can handle different objects without needing to be modified or rewritten for each type while extensibility allows adding new functionality to the code by introducing new classes without having to modify the existing code that relies on the superclass or interface.

Example: Considering the Shapes in Drawing Task. The Shapes can have many forms such as circles, Rectangles, Line. While they share the same function and properties they are all

inherit from the Shape class(child of the Shape class). So, they can share the same function with each other and the shapes(Draw(), DrawOutLine(), IsAt(),…)


Override
Cohesion
increase
Colaborations
can
required to
Method Calll
used for
Virtual
Abstract Method
ways of
defined in
Overload
Methods
marked as
contains
can be
can be
can be
Roles
Resposibility
Abstract Class
Polymorphism
Public
Protected
Privite
can be
can be
can be
have
involve
Properties
Encapsulation
use
Abstraction
Inheritance
enhace
Coupling
Fields
use
Interfaces
loose
can be
Value Type
Reference Type
Classes
intances of
New
created by
Objects
models
OOP