Introduction

OOP is a programming paradigm that focuses on organizing code into objects, which are instances of classes. Each object contains data (in the form of attributes) and behaviour (in the form of methods). OOP provides a way to structure code that is easy to maintain, extend, and reuse. In this document, we are going to read about the four key principles of OOP.

Key Principles

1. **Encapsulation**: Encapsulation is the practice of hiding the internal workings of an object from the outside world, and only exposing a public interface that other objects interact with. This principle helps to ensure that the internal state of an object is only modified in a controlled wat, which can make code easier to maintain.

   Example:
   Suppose we are developing a banking application. One of the objects we might define as "BankAccount" class, which contains data about the customer's account balance, acc number, etc. to protect this information from unauthorized access, we may make the data attributes private, and only expose public methods for depositing, withdrawing, and checking the acc balance.

   Relation to previous program:
   In the drawing program, we can see that we implement this in our "MyLine" class. It has a private float of both ends of the lines which can only be accessed and modified through the properties.

2. **Inheritance**: Inheritance is the process by which one class can inherit the properties and methods of another class. This allows for code reuse and can make it easier to create and maintain large software systems.

   Example:
   Suppose we are developing a video game that contains many different types of enemies. We might define a base "Enemy" class. Then, we could create sub-classes for each type of enemy, such as "Goblin" and "Archer", which inherit from the base "Enemy" class but add their own unique properties and behaviours. This way we can reuse code and avoid duplicating common functionality across multiple classes.

   Relation to previous program:
   In ShapeDrawing, we had a "MyRectangle" class which inherits all the features from our "Shape" class. This means that "MyRectangle" gets all the features of our "Shape" class. Now "MyRectangle" knows its location, its size, its colour and if it is selected. All these features are inherited from the "Shape" class.

3. **Polymorphism**: Polymorphism is the ability of objects to take on many forms. In OOP, this typically refers to the ability of different objects to responds to the same message in different ways. Polymorphism can make code more flexible and easier to extend.

   Example:
   Suppose we have a “Player” class that can attack enemies. We might define an “Attack” method that takes an “Enemy” object as a parameter. Because each type of enemy has its own attack behaviour, we can implement polymorphism by defining a different “Attack” method for each type of enemy. This way, when the player attacks an enemy, the appropriate “Attack” method is called based on the type of enemy being attacked.

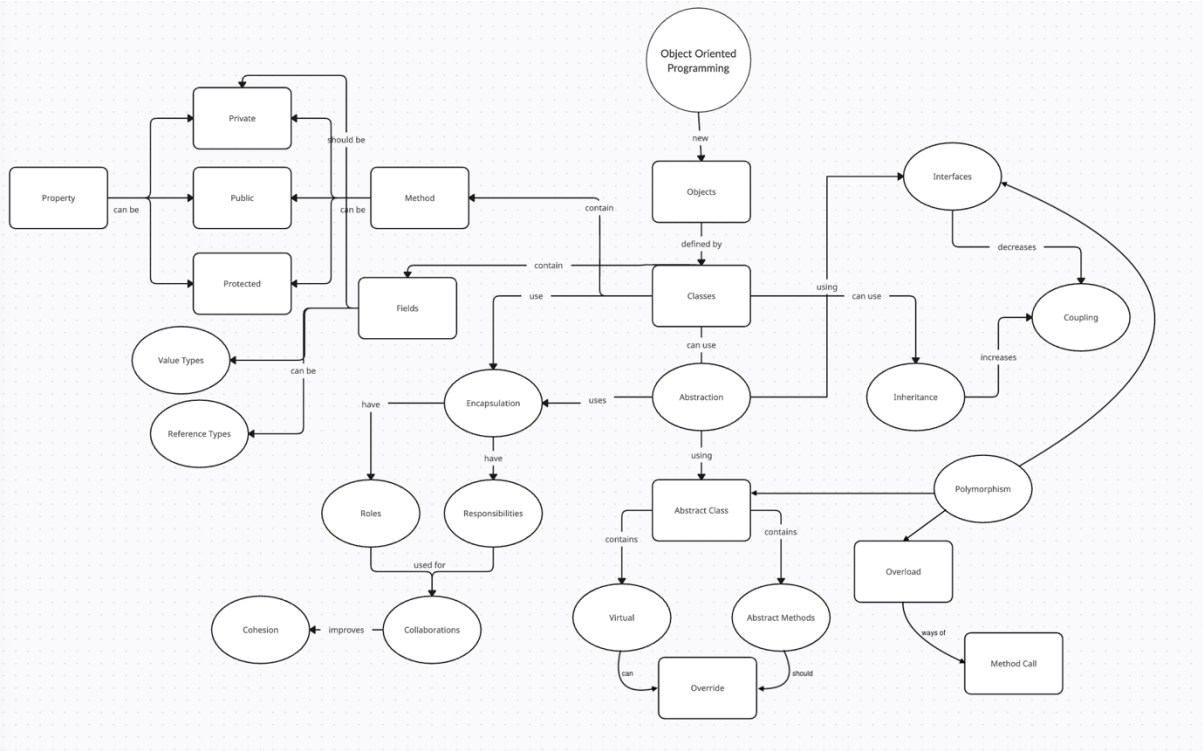
   Relation to previous program:
   In SwinAdventure, we had an interface “IHaveInventory” which has a Locate method. All objects using the interface had their own implementation, this shows how polymorphism works. It allows the object to use the same interface but using the method in their own way.

4. **Abstraction**: Abstraction is the practice of focusing on the essential features of an object, while ignoring its implementation details. This principle can help to simplify complex systems and make them easier to understand.

   Example:
   Suppose we are developing a music player application. We might define a “Song” class that contains data about the song, such as the title, artist, and duration. To make the code more abstract, we could define an abstract “Media” class that contains common properties and methods for all types of media, such as songs, albums, and playlists. Then, we could define a sub-class for each type of media, which inherits from the base “Media” class but adds its own unique properties and behaviours. This way, we can focus on the essential features of each type of media, while ignoring the implementation details.

   Relation to previous program:
   In DrawingProgram, we had an abstract class “Shape”. Which has methods such as “Draw”, “IsAt”, and “DrawOutLine” which all types of shapes must have (rectangle, circle, line). This means that “Shape” will tell the shapes to draw their own shape, but only the shapes themselves will know how to draw themselves.

Concept Map


Object Oriented Programming
new
Objects
defined by
Classes
Property
can be
Private
Public
Protected
should be
can be
Method
contain
Fields
contain
use
Value Types
Reference Types
can be
Encapsulation
have
uses
Abstraction
can use
using
Interfaces
decreases
Coupling
using
can use
Inheritance
increases
Polymorphism
have
Roles
Responsibilities
used for
Cohesion
improves
Collaborations
Abstract Class
contains
contains
Virtual
Abstract Methods
can
should
Override
Overload
ways of
Method Call