First and foremost, based on my knowledge and research, there are four distinct principles of Object-Oriented Programming (OOP): Abstraction, Encapsulation, Inheritance, and Polymorphism. Moreover, I will describe them carefully and in detail with examples of previous tasks.

A. Abstraction

From my perspective, "Abstraction" in the OOP can be understood in the following case. For example, if you are considered generating a shipping application, the attribute ***_color*** may not be important or be ignored. However, when implementing the selling interface with various products, the ***_color*** is important to the customer. Similarly, in the task ***ShapeDrawing***, the ***Shape*** class includes ***IsAt()***, ***DrawOutline()***, and ***Draw()*** methods. Moreover, according to the UML design, the three mentioned methods are also abstract, forcing any shape classes inherited from ***Shape*** including ***MyRectangle***, ***MyCircle***, and ***Myline*** to provide their own unique implementations for these methods.

B. Encapsulation

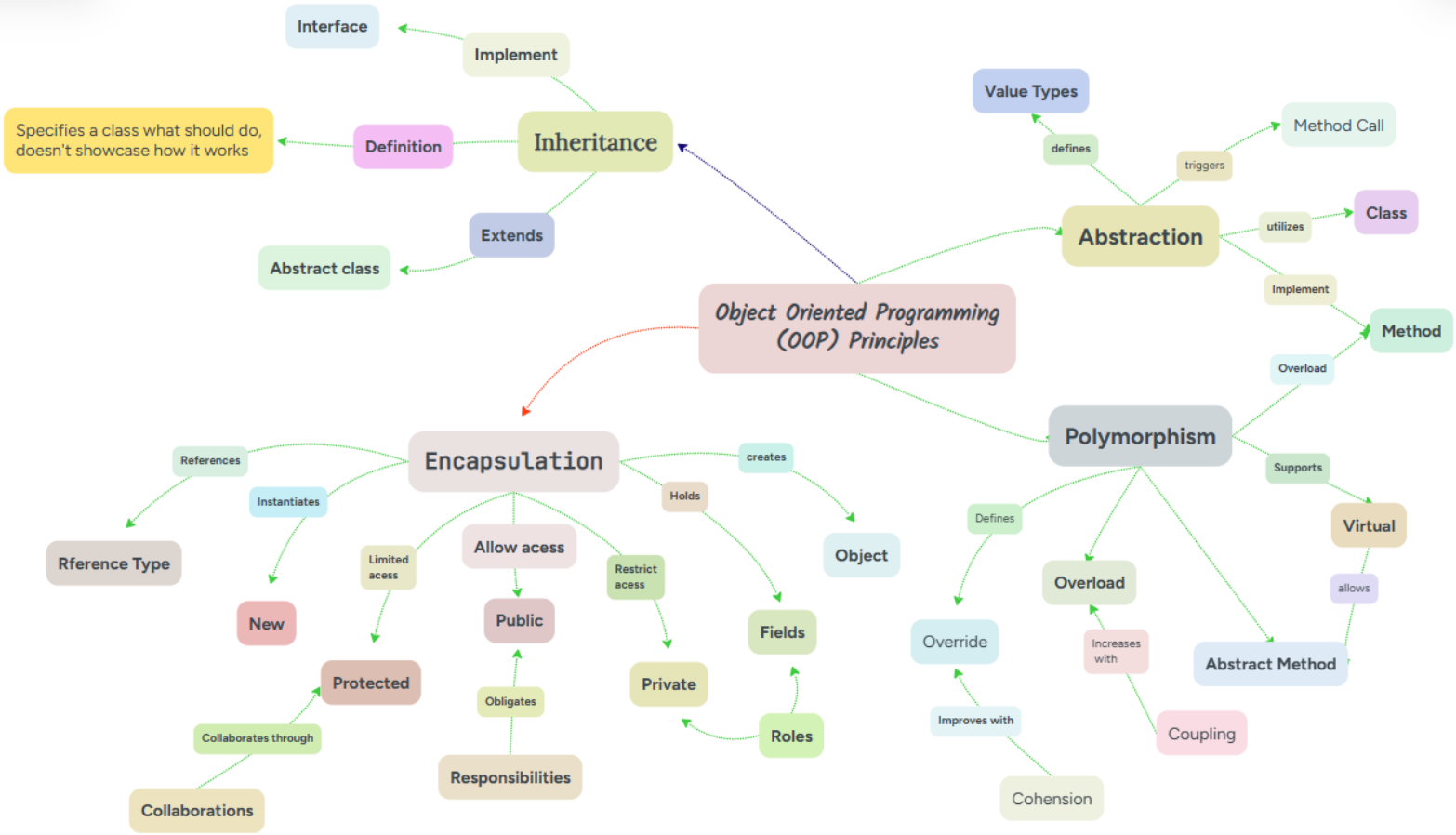
In my opinion, "Encapsulation" serves as the principle of bundling data (attributes) and the methods that operate on that data within a single unit (a class) and restricting direct access to the internal state. In essence, the principle preserves data integrity and mitigates unintended external interactions. For instance, in the ***ShapeDrawing*** task, attributes such as ***Color***, ***_x*** and ***_y*** are set as private, meaning that other classes can only access and modify these attributes through public methods provided by ***Shape*** class.

C. Inheritance

In my point of view, when learning the "inheritance" principle, there are two crucial definitions which students must understand: parent and child classes. Furthermore, a child (derived) class can inherit all members from its parent (base) class. Specifically, ***MyRectangle***, ***My***Circle, and ***Myline*** clasess (child clasess) inherit from ***Shape*** class (parent class), meaning that they share basic ***Shape*** abilities but still draw themselves seperately.

D. Polymorphism

Based on my knowledge and researh, "Polymorphism" principle means "having multiple forms" or "many shapes" in Greek. For example, I am writing a program contains ***Move()*** method and then I can utilize the above method for others such as ***Cat()*** and ***Dog().*** Similarly, polymorphism allows different objects, such as circles and rectangles, to have distinct drawing methods. While they all belong to the Shape class, their drawing methods vary based on specific attributes like height, width, or length.


Object Oriented Programming (OOP) Principles
Inheritance
Implement
Interface
Definition
Specifies a class what should do, doesn't showcase how it works
Extends
Abstract class
Abstraction
defines
Value Types
triggers
Method Call
utilizes
Class
Implement
Method
Polymorphism
Overload
Supports
Virtual
allows
Abstract Method
Defines
Override
Improves with
Cohension
Overload
Increases with
Coupling
Encapsulation
References
Rference Type
Instantiates
New
creates
Object
Holds
Fields
Limited acess
Protected
Collaborates through
Collaborations
Allow acess
Public
Obligates
Responsibilities
Restrict acess
Private
Roles