I have removed everything like "poetry" and focus on the formal definition and actual description based on my research and understanding.

RELATING TO SHAPE DRAWING PROGRAM – WHAT I HAVE LEARNED AND DID IN THE UNIT

These four OO principles have been used in many weekly tasks to carry out an efficient program. And the example that I have in shape drawing is one of them, and I have described each of the principles by four examples below.

ABSTRACTION

Definition: Definition: Abstraction is the purposeful suppression or hiding of some details of a process or artifact to emphasize other aspects, details, or structure (Timothy A Budd, 2002).

Understanding: Abstraction argues that the details of how objects learn, and function should be left out, and that they should only know certain facts and perform certain actions. It identifies primary classifications, roles, responsibilities, and collaborations. By doing so, it streamlines the design process. Classification is the process of defining an object. Each program object must have a role, a function, and the responsibilities required to fulfil that role. Partnerships mean that the relationship between things is also defined by abstraction.

Example: Consider a Drawing Program. The Shape Class has methods like IsAt, DrawOutline, and Draw. The exact workings of these methods are implemented in MyLine, MyRectangle, and MyShape classes. This division allows for a clean interaction with the Shape Class, making it less complicated.

ENCAPSULATION

Definition: We use the term encapsulation to mean that there is a strict division between the inner and the outer view (Timothy A Budd, 2002).

Understanding: Encapsulation emphasizes the principle that while objects' methods and properties are accessible for use, their internal mechanics should be protected from direct interference. This safeguard ensures that users interact with an object based on its functionalities without inadvertently disrupting its internal processes. We can manage the accessibility levels of various components within a class by using access modifiers. Encapsulation, in essence, aids in the preservation of data integrity and the mitigation of unintended external interactions.

Example: In the Drawing Program, the Shape Class has private attributes like Color, and coordinates _x and _y. Other classes can only access or modify these through the public methods provided by the Shape Class.

POLYMORPHISM

Definition: Polymorphism is the art of flexibility, allowing methods to wear different hats based on the object they mingle with, creating a harmonious interface for various types. Polymorphism allows the shared code to be tailored to the specific circumstances of individual data types (Timothy A Budd, 2002).

Understanding: Polymorphism enables flexibility in software design. It allows shared code to be tailored to cater to individual data types, ensuring both adaptability and uniformity. Essentially, a child class can inherit functionalities from its parent class while also maintaining its unique methods. This feature grants developers the liberty to manipulate an object's type seamlessly within the program.

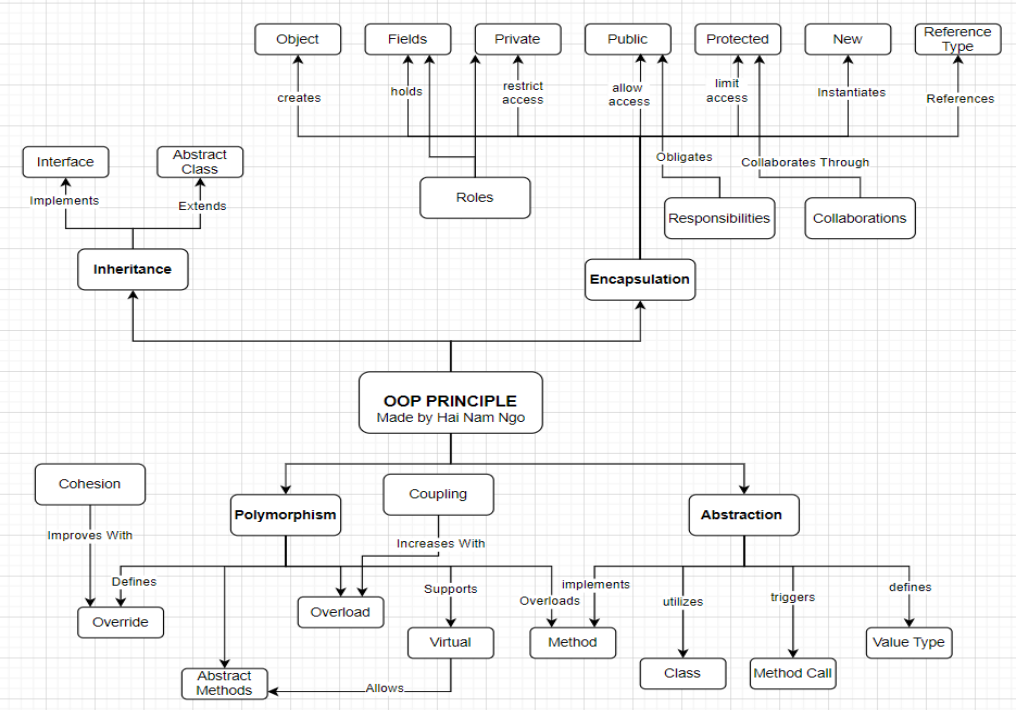
Example: Using the Shape class, polymorphism allows different objects, such as circles and rectangles, to have distinct drawing methods. While they all belong to the Shape class, their drawing methods vary based on specific attributes like height, width, or length.

INHERITANCE

Definition: The idea of inheritance is classes can be organized into a hierarchical inheritance structure. A child class (or subclass) will inherit attributes from a parent class higher in the tree (Timothy A Budd, 2002).

Understanding: Inheritance allows one class to be formed based on another. A child (or derived) class can inherit all members from its parent (or base) class. Developers leverage inheritance to create a structured hierarchy of classes, enhancing code reusability. Alongside inheritance, abstract classes and interfaces come into play. Abstract classes are intended for inheritance as they might have members without specific implementations. While classes can be inherited from interfaces to gain extra functionalities not tied to the class hierarchy, it's essential to note the difference: a class may implement multiple interfaces but can only inherit from a single base class.

Example: In the Drawing Program, the Shape Class is the parent, from which MyLine, MyRectangle, and MyCircle inherit. While they get general methods from Shape, each of them might have additional methods or variations due to their distinct attributes.



REFERENCES

Timothy A Budd. (2002). *Information for an introduction to object-oriented programming 3rd ed.*

Web.engr.oregonstate.edu.

https://web.engr.oregonstate.edu/~budd/Books/oopintro3e/info/ReadMe.html