Name: Trung Kien Nguyen

Student ID: 104053642

COS20007

Object-Oriented Programming

REPORT

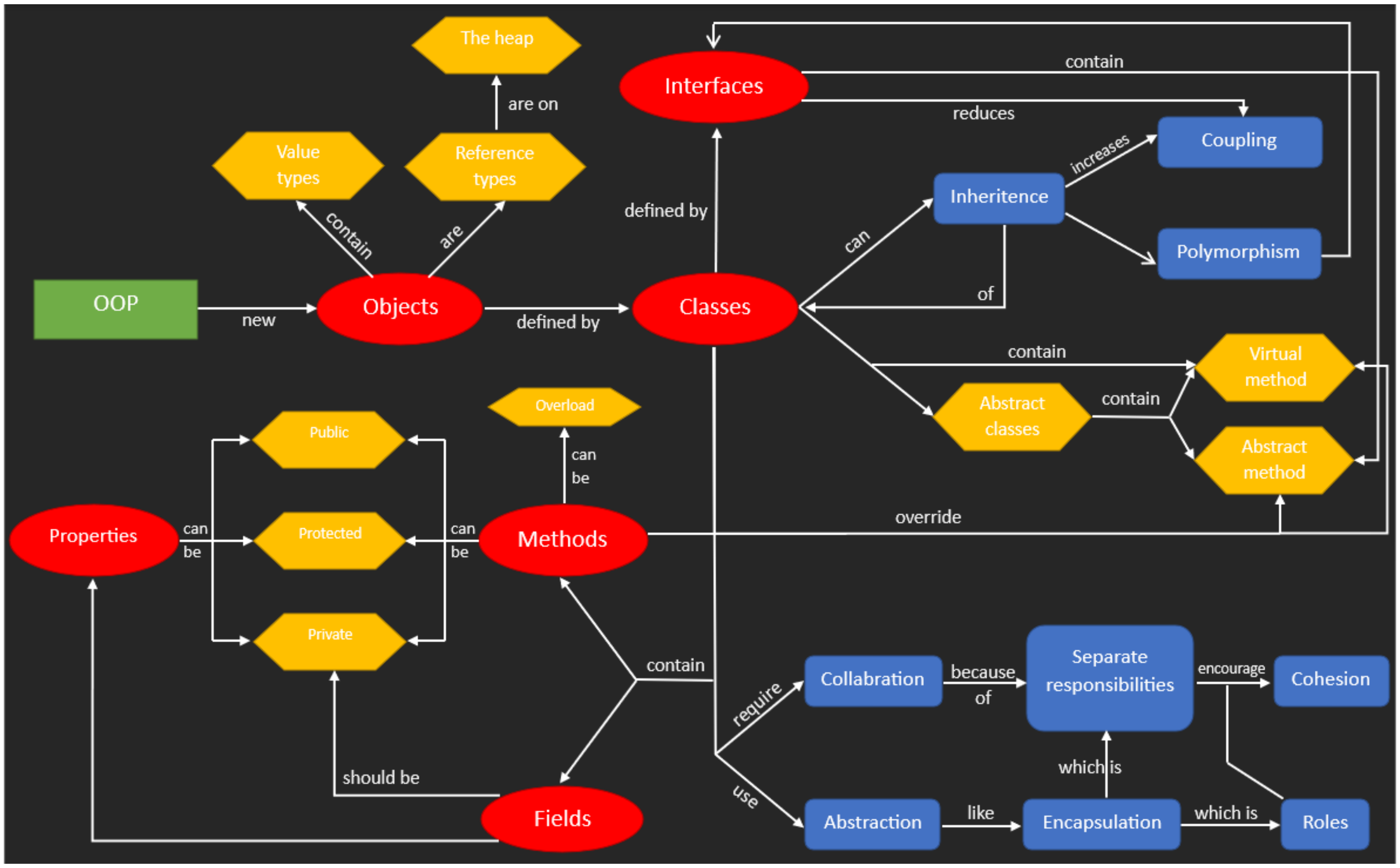
Key Object Oriented Concepts

TABLE OF CONTENTS

DEFINITION AND MAIN CONCEPT

The term of "Object-oriented programming" is based on the concept of "objects", and puts a great deal of emphasis on the objects than the logic and function when creating a program/application. The definition of object in OOP is defined as a self-contained entity that contains both data (fields and properties) and behavior (methods) that operate on that data.

In OOP, programs are organized around objects and their interactions. The main idea is to encapsulate data and behavior in a single entity for better code abstraction, modularity and reuse. Also, because programmers can change and rearrange elements within a program, OOP is well suited for large, complex, and frequently updated applications. The following diagram shows the main concepts of OOP:



The OOP Concept map

In summery, OOP provides a way to organize and structure your code in a more modular, maintainable, and scalable way. OOP concept, and its key principles as well, is popularly and widely used by high level programming languages such as C#, Java, C++, Python, ... , and is considered a powerful and flexible way to build programs/applications.

KEY PRINCIPLES

There are four key principles of Object-oriented Programming (OOP), including ***Abstraction***, ***Encapsulation***, ***Inheritance***, and ***Polymorphism***.

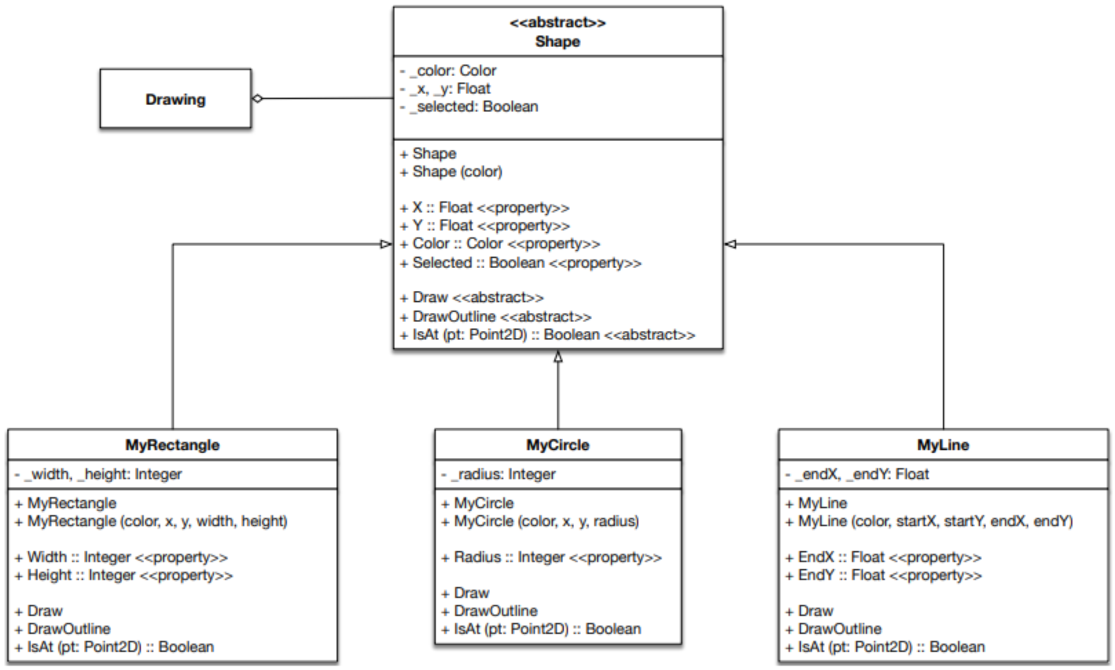
1. **Abstraction:** Abstraction basically involves reducing complex systems or processes to simpler, more manageable components. In OOP, this term is used to create classes that publicing necessary properties and methods for others to see or use, while hiding unnecessary details. Thus, abstraction principle focus only on the essential characteristics of an object, while ignoring its irrelevant or distracting details. It is closely related to encapsulation principle.

2. **Encapsulation:** This OOP's principle simply means is to encapsulate of data and code within an object. Encapsulation allows an object to control its own data (e.g, fields), and to provide access to only properties and methods necessary to interact with that data. This keeps the object's internal state remains consistent and protected from external interference of the program/application.

3. **Inheritance:** This principle allows a class (child class/subclass) to inherit "protected" and "public" properties and methods from another class (parent class). Inheritance is often used to promotes code reuse and helps simplify the design of the programs/applications. Inheritance relationships are usually expressed as "is-a" relationships, with subclasses being a more specialized type of parent class, and is related to the principle Polymorphsim.

4. **Polymorphism:** Polymorphism refer to the ability of objects to take on different forms or behaviors, depending on the context in which they are used. In other words, this means that the child class has both its type and its parent's type. Furthermore, polymorphism promotes readability, flexibility, and extensibility in OOP program/application designs.

Let take an example of the code I have done in the task *4.1P - Drawing Program - Multiple Shape Kinds*


Drawing
<<abstract>>
Shape
- _color: Color
- _x, _y: Float
- _selected: Boolean
+ Shape
+ Shape (color)
+ X :: Float <<property>>
+ Y :: Float <<property>>
+ Color :: Color <<property>>
+ Selected :: Boolean <<property>>
+ Draw <<abstract>>
+ DrawOutline <<abstract>>
+ IsAt (pt: Point2D) :: Boolean <<abstract>>
MyRectangle
- _width, _height: Integer
+ MyRectangle
+ MyRectangle (color, x, y, width, height)
+ Width :: Integer <<property>>
+ Height :: Integer <<property>>
+ Draw
+ DrawOutline
+ IsAt (pt: Point2D) :: Boolean
MyCircle
- _radius: Integer
+ MyCircle
+ MyCircle (color, x, y, radius)
+ Radius :: Integer <<property>>
+ Draw
+ DrawOutline
+ IsAt (pt: Point2D) :: Boolean
MyLine
- _endX, _endY: Float
+ MyLine
+ MyLine (color, startX, startY, endX, endY)
+ EndX :: Float <<property>>
+ EndY :: Float <<property>>
+ Draw
+ DrawOutline
+ IsAt (pt: Point2D) :: Boolean

- Abstraction: The “Shape” abstract class provides an abstraction of a generic shape, which allows specific shapes, including “MyRectangle”, “MyCircle” and “MyLine”, to inherit from it and implement their own specific versions of its abstract methods. “MyRectangle”, for example, also provides an abstraction of a rectangle, publicing the properties and methods (“X”, “Y”, “Width”, “Height”, “Draw()”, “DrawOutline()”, “IsAt()”) necessary to define a rectangle, while hiding its internal implementation details.

- Encapsulation: The "Shape" class encapsulates the data of a shape, e.g. rectangle, including its "Color", "X", "Y", and "Selected" properties from the parent class, and its own “Width”, “Height”. These properties also include getter and setter methods for encapsulated access to their values.

- Inheritance: The "MyRectangle", "MyCircle", and "MyLine" classes inherit from the "Shape" abstract class, which is a clearly example of inheritance in action. By inheriting from the abstract class "Shape", these subclasses inherit its public properties ("Color", "X", "Y", and "Selected"), and abstract methods ("Draw()", "IsAt()", and "DrawOutline()") that is overidden by those three with their own implementations specific to rectangles, circles, and lines respectively.

- Polymorphism: The "MyRectangle", "MyCircle", and "MyLine" classes demonstrate the fourth principle by overriding the "Draw()", "IsAt()", and "DrawOutline()" methods inherited from "Shape". This means that any code that expects a "Shape" object can also use a "MyRectangle", "MyCircle", or "MyLine" object in its place, since those are the types of Shape. In other word, if there is a method that takes a "Shape" object as a parameter, it can be passed a "MyRectangle" object instead as "MyRectangle" is a child class of "Shape", as presented in the “Main()” method.

REFERENCES

[1] ChatGPT. Retrieved from: https://chat.openai.com/