Ta Quang Tung - 104222196 – COS20007 - 7.1P - Key Object Oriented Concepts

Object-oriented programming: A programming paradigm in which programs are made of objects that contain data and functionality and interact with each other. *Example: an OOP program for student management can contain Student, Course, and Timetable objects.*

class: A class defines a blueprint for an object. It specifies what kind of data and functions an object should have. *Example: a Course class can declare a name and course ID.*

object: An object is an entity that contains data and functionality created from a class. *Example: a Course object can have a name of "OOP" and ID of "COS20007".*

field: A field is a variable declared directly in a class. Objects store their data in fields. *Example: a Student object can have fields such as name and class.*

method: A method is a function defined in a class. The functionality of an object is contained in methods. *Example: a Student object can contain a method to register for a class.*

new: The new operation creates an object from a class. *Example: Student student = new Student() creates a new student object and assigns its reference to the "student" variable.*

method call: Calling a method simply means using (or invoking) it. *Example: To tell a student to register for a class, call its Register method.*

value type: A variable of a value type contains the actual instance of that type [1]. *Examples of value types are integers, floats, booleans.*

reference type: A variable of a reference type contains a reference to the instance of that type [1]. *Examples of reference types are arrays, lists, and class types.*

Abstraction: Abstraction involves designing classes by identifying the essential features of objects. It also means to hide away the internal implementation of an object and present only what is needed to use it. *Example: Anything that has the ability to track and read time, tick by 1 second, and reset is a Clock. A clock object can have ReadTime, Tick, and Reset methods. Objects that call these methods on the outside do not have to care about how they are implemented.*

Encapsulation: Encapsulation means to keep the data and methods of an object private to prevent external access unless explicitly specified. *Example: a Clock object can expose a public method for reading time but keep its time field private so that outsiders cannot change it directly.*

public: An access modifier that allows fields and methods to be accessible outside a class [2]. *Example: the public Draw method of Shape objects allows it to be called from outside.*

private: An access modifier that restricts data and method access to inside a class [3]. *Example: the private _x field of a Shape cannot be accessed outside of Shape.*

protected: An access modifier that restricts data and method access to the class and classes that inherits from it [4]. *Example: the protected _x field of a Shape is accessible inside Shape and classes that inherit from Shape such as Circle or Line.*

Inheritance: Inheritance allows classes to reuse the logic of other classes. *Example: Circle, Square, and Line classes can all reuse the logic of a base Shape class.*

interface: An interface defines a set of properties and methods that a class must implement. *Example: a class that implements the IHaveInventory interface must contain the properties and methods declared in IHaveInventory such as Fetch and Name.*

abstract class: An abstract class is meant to be inherited and cannot create objects of its own. *Example: Shape can be marked as abstract because it does not define any specific Shape and is meant to be inherited.*

abstract method: An abstract method contains no implementation and is meant to be overriden by children classes. *Example: Shape can contain an abstract Draw method that must be explicitly implemented by Circle, Rectangle, and Line.*

virtual: A virtual method contains implementation but can be overriden by children classes. *Example: GameObject contains the virtual property FullDescription which has some basic implementation but can be overriden for more specific outputs.*

override: An override method contains the extended implementation of a virtual or abstract method. *Example: Player, which inherits from GameObject, can override the base implementation of FullDescription to return more detailed information about the player.*

Polymorphism: Polymorphism allows objects of different derived classes to be treated as objects of the base class. *Example: objects of Circle, Rectangle, and Line can all be stored in an array of Shapes.*

overload: An overloaded method provides a different set of parameters so that the same method name can be used with different parameters. *Example: Console.WriteLine can be used with string, integer, or boolean parameters thanks to overloading [5].*

Role: A role is the purpose of an object in a program. It is also defined as a group of responsibilities [6]. *Example: a Drawing object's role is to represent a drawing on the screen, which is made of shapes.*

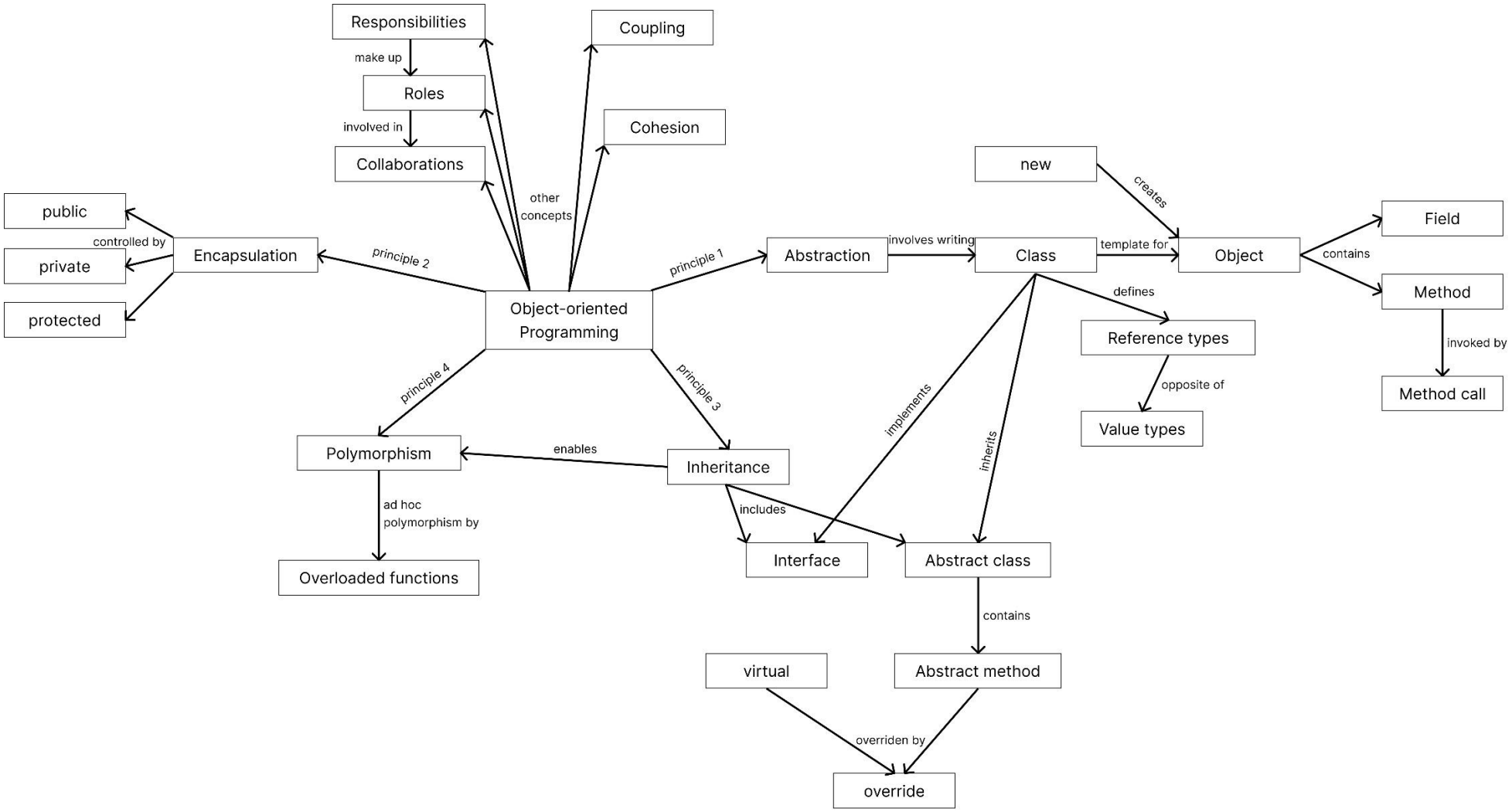
Responsibility: A responsibility is a duty that an object performs to fulfil its role. *Example: a Drawing object's responsibilities are to store shapes, store selected shapes, and draw all shapes onto the screen.*

Collaboration: Collaboration refers to the interaction between objects in the program. Objects need to work together to produce the desired output. *Example: a Drawing object will ask all shapes within it to draw themselves onto the screen.*

Coupling: Coupling refers to how dependent two classes are on one another. Low coupling means that major changes in one class will not affect the other. *Example: Although a Drawing object can contain many kinds of shapes, it depends only on the common methods declared in the abstract class Shape. It does not care how the different concrete Shapes are implemented. Therefore, major changes in Circle, Rectangle, or Line do not affect Drawing.*

Cohesion: Cohesion refers to how related the methods in a class are. High cohesion means that the methods are closely related and have a narrow focus. *Example: a Drawing object should not contain code that dictates how individual Shapes should be drawn, or code that determines how different Shapes should be selected.*

Concept map


Responsibilities
make up
Roles
involved in
Collaborations
Coupling
Cohesion
other concepts
public
controlled by
private
protected
Encapsulation
principle 2
Object-oriented Programming
principle 1
Abstraction
involves writing
Class
new
creates
template for
Object
Field
contains
Method
invoked by
Method call
defines
Reference types
opposite of
Value types
principle 4
principle 3
Polymorphism
enables
Inheritance
implements
inherits
ad hoc polymorphism by
Overloaded functions
includes
Interface
Abstract class
contains
virtual
Abstract method
overriden by
override

References

[1] Microsoft. (2022, September 29). *Value types—C# reference*. https://learn.microsoft.com/en-us/dotnet/csharp/language-reference/builtin-types/value-types

[2] Microsoft. (2022, January 25). *public keyword—C# Reference*. https://learn.microsoft.com/en-us/dotnet/csharp/language-reference/keywords/public

[3] Microsoft. (2022, October 27). *private keyword—C# Reference*. https://learn.microsoft.com/en-us/dotnet/csharp/language-reference/keywords/private

[4] Microsoft. (2022, January 25). *protected keyword—C# Reference*. https://learn.microsoft.com/en-us/dotnet/csharp/language-reference/keywords/protected

[5] Microsoft. (n.d.). *Console.WriteLine Method (System)*. Retrieved June 25, 2023, from https://learn.microsoft.com/en-us/dotnet/api/system.console.writeline?view=net-7.0

[6] Wirfs-Brock, R., & McKean, A. (2002). *Object Design: Roles, Responsibilities, and Collaborations*.