Swinburne University of Technology

COS20007 Object Oriented Programming

6.2P - Key Object Oriented Concepts

PDF generated at 00:04 on Thursday 4th May, 2023

Name: Trung Kien Nguyen

Student ID: 104053642

SWINBURNE UNIVERSITY OF TECHNOLOGY

COS20007

Object-Oriented Programming

REPORT

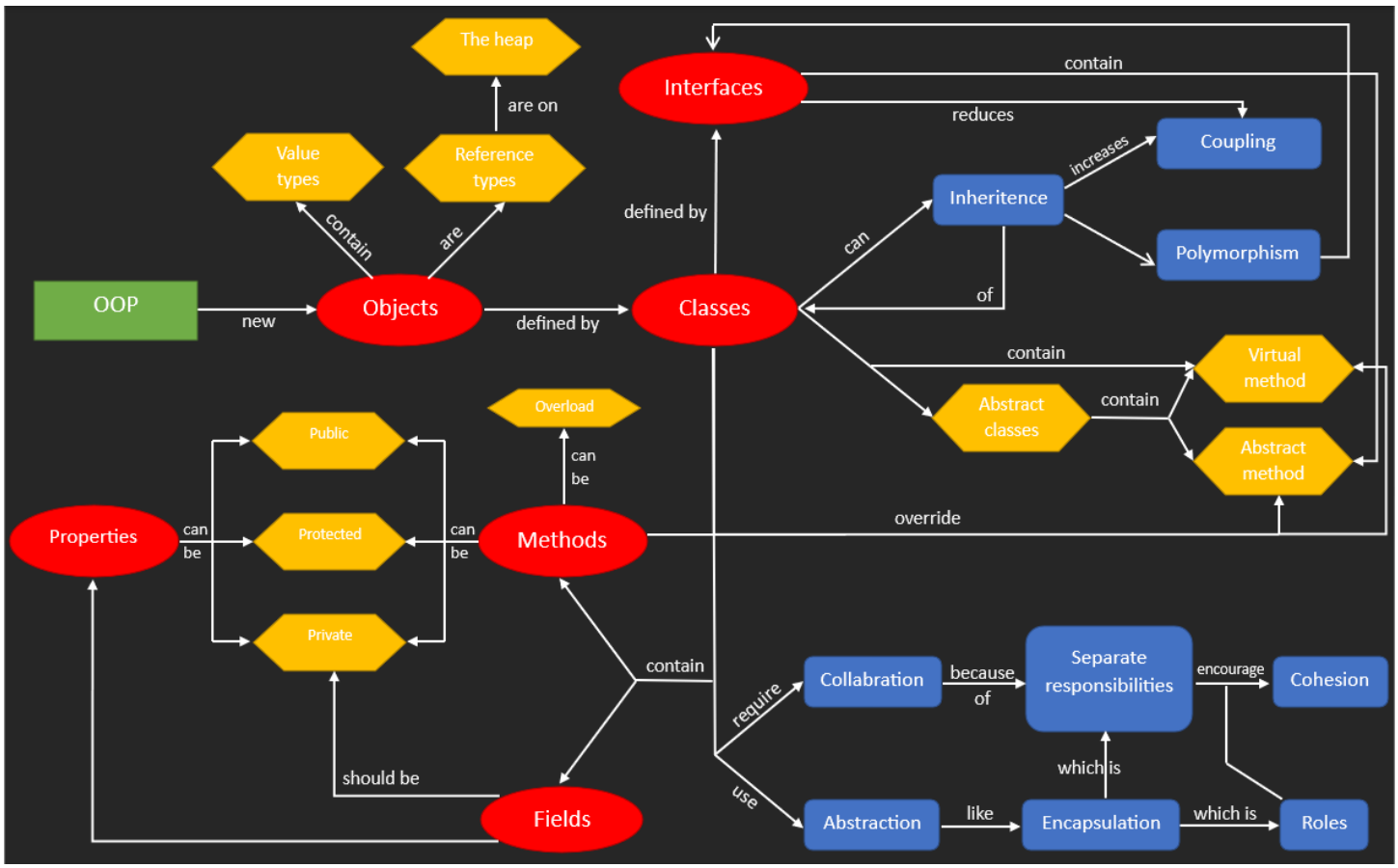
Key Object Oriented Concepts

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

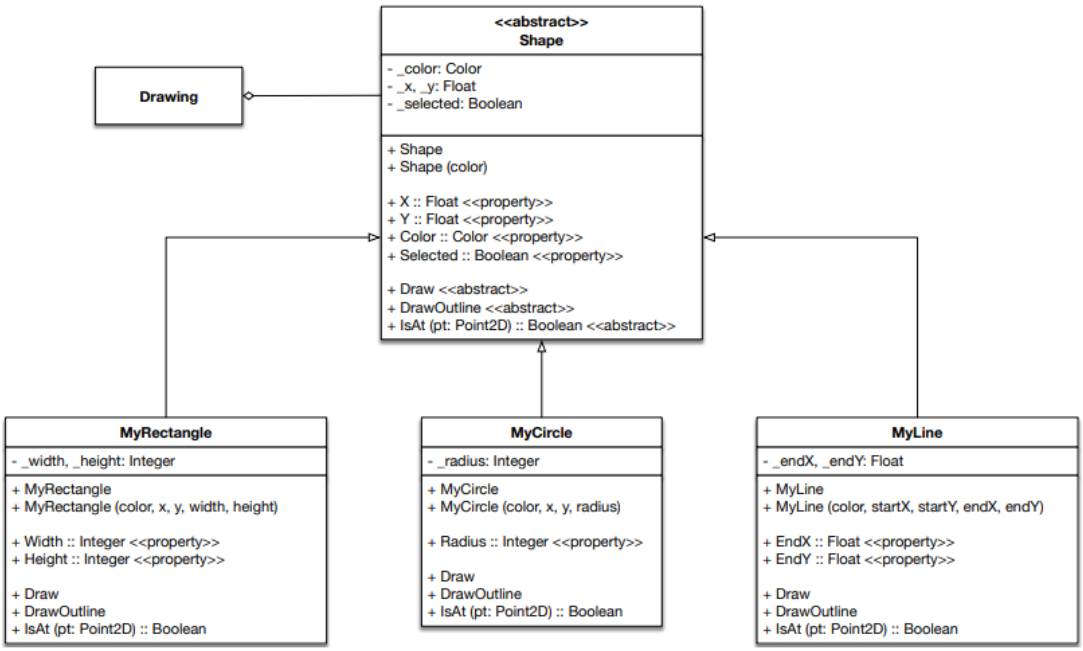
There are four key principles of Object-oriented Programming (OOP), including ***Abstraction***, ***Encapsulation***, ***Inheritance***, and ***Polymorphism***.

1. **Abstraction:** Abstraction involves reducing complex systems or processes to simpler, more manageable components. In OOP, abstraction is used to create classes that represent real-world objects or concepts, while hiding unnecessary details. Abstraction allows us to focus on the essential characteristics of a system or object, while ignoring irrelevant or distracting details. Abstraction is closely related to encapsulation, as both principles involve hiding implementation details from external users or clients.

2. **Encapsulation:** This principle refers to the bundling of data and code within a single unit, or object. Encapsulation allows an object to control its own data, and to provide access to only the methods necessary to interact with that data. This keeps the object's internal state remains consistent and protected from external interference.

3. **Inheritance:** This principle allows a class (child class/subclass) to inherit properties and methods from another class (parent class). Inheritance is usually used to promotes code reuse and helps simplify the design and implementation of complex systems. Inheritance relationships are usually expressed as "is-a" relationships, with subclasses being a more specialized type of parent class.
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. Polymorphism allows different objects to respond to the same message or method call in different ways, based on their class or type. 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*



- 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, encapsulating the properties and methods necessary to define a rectangle, while hiding its internal implementation details.

- Encapsulation: The "Shape" class encapsulates the data of a rectangle, including its "Color", "X", "Y", and "Selected" properties, and its methods "Draw", "IsAt", and "DrawOutline". The 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 them with their own implementations specific to rectangles, circles, and lines respectively.

- Polymorphism: The "MyRectangle", "MyCircle", and "MyLine" classes demonstrate polymorphism 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".

OTHER CONCEPTS

In Object-Oriented Programming, there are some other important concepts, including:

- Collaboration:
    - In OOP, classes are designed to interact and collaborate with each other to achieve a larger goal or task. This collaboration is necessary because no class can fully accomplish a complex task on its own. Instead, different classes with unique functionalities and responsibilities work together to achieve a larger goal.
    - Collaboration in OOP can take different forms, such as method calls, message passing, and data sharing between objects. For example, one class may need to call a method, or access a property of another class to complete a specific task or requirement of the program. This is done by creating objects of the classes and then using them to exchange messages or call methods to communicate with each other.
- Responsibilities:
    - The concept of “responsibilities” in OOP refers to the tasks or functionalities for which a class is responsible. Each class in the application, or program, is designed to have specific responsibilities or areas of expertise, which help to organize and modularize its codebase, making it easier to update, maintain, and extend.
    - A class's responsibilities can be defined by the methods and properties it contains, and the interactions it has with other classes. A well-designed class should have a clear and concise purpose, with its responsibilities being easy to understand and implement.
- Roles:
    - Different kinds of objects that can be constructed from a single class are referred to as "roles" in OOP. By exposing different interfaces, properties, and methods on various objects, a class can define several roles.
    - Roles help programmers reuse code more effectively and build more adaptable and flexible systems. Programmers can design objects that can interact with other objects in various ways based on the role by declaring numerous roles in a single class.
- Coupling:
    - Coupling in OOP refers to the level to which one class is dependent on another class. If one class relies heavily on another class, then the two classes are said to be tightly coupled. If one class has little or no reliance on another class, then the two classes are said to be loosely coupled.
    - Tightly coupled classes are usually more difficult to maintain, as changes to one class may require changes to other classes that depend on it. On the other hand, separated, or loosely coupled

classes are easier to update, extend and maintain, since changes to one class have minimal impact on other classes.

- Cohesion:
    - OOP's "cohesion" idea describes the degree to which a class's constituent parts are interconnected and cooperate to accomplish a single, clearly defined task or goal. While a class with low cohesion tries to do too many things at once, a class with high cohesion is concentrated on a single, distinct responsibility or functionality.
    - Normally, high cohesion in a class makes it more easier to read and understand, and also to update, maintain and extend, as each field, property and method and is related to specific functionality, and making changes to one part of the class is less likely to affect other parts of that class.

SOME TERMINOLOGIES

In OOP, there are some significant terminologies, such as:

- "Value type" and "Reference type": These are two types of data in OOP
    - Value types data types, which are allocated on the stack, store their values directly in memory. This implies that a copy of a value type's value is created whenever it is provided as a parameter or assigned to a variable. Integers, float numbers, and boolean values are a few forms of values.
    - Reference types, on the other hand, hold a reference or pointer to the memory location where the actual object is stored. Reference types are allocated on the heap, making them to have a longer lifespan and can be accessed from multiple parts of a program/application. When a reference type is assigned to a variable or passed as a parameter, only its reference or memory address is copied, not the object itself. Examples of reference types include objects, arrays, and strings.

- "Abstract Class" and "Abstract Method":
    - In OOP, an abstract class is a class that cannot be instantiated, meaning you cannot create objects directly from it. Instead, it serves as a blueprint for creating derived classes. It is able to contain both abstract and non-abstract methods, and can also have fields, properties, and constructors, depending on the program/application requirement.
    - An abstract method, which can only be declared inside an abstract class, or interface, is a method that has no implementation in the abstract class, and is marked with the "abstract" keyword, meaning that any class that derives from the abstract class must provide an implementation for the abstract method. This method has to be

overriden by the subclass inherited from the abstract class.

- “private”, “public” and “protected: In OOP, access modifiers like private, public, and protected are used to control the visibility and accessibility of classes, methods, and properties within a program:
    - Public: A public member can be accessed from anywhere in the program, including other classes and namespaces.
    - Private: A private member can only be accessed from within the same class, while not accessible to any other classes or namespaces.
    - Protected: A protected member can **only** be accessed from within the same class and its inherited classes.

- “virtual”: In OOP, "virtual" is a keyword that can be used to define a method or property in a base class that can be overridden by derived classes.

- “override”: In OOP, "override" is a keyword that is used to indicate that a method or property in a derived class is intended to replace a method or property in the base class with the same name and signature.

- “overload”: In OOP, "overload" is a term used to describe the ability to define multiple methods or constructors in a class with the same name but different parameters.
    - Method overloading is a technique used to provide multiple methods with the same name but different parameter lists, allowing the method to perform different operations depending on the type and number of arguments passed in.
    - Overloading is a useful technique that can help to make code more flexible and easier to use, by allowing methods to be called with different argument types or numbers of arguments, without having to provide different method names.

REFERENCES

[1] Object-oriented programming. Retrieved from: *https://en.wikipedia.org/wiki/Object-oriented_programming*

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