7.1

Key Object-Oriented Concepts

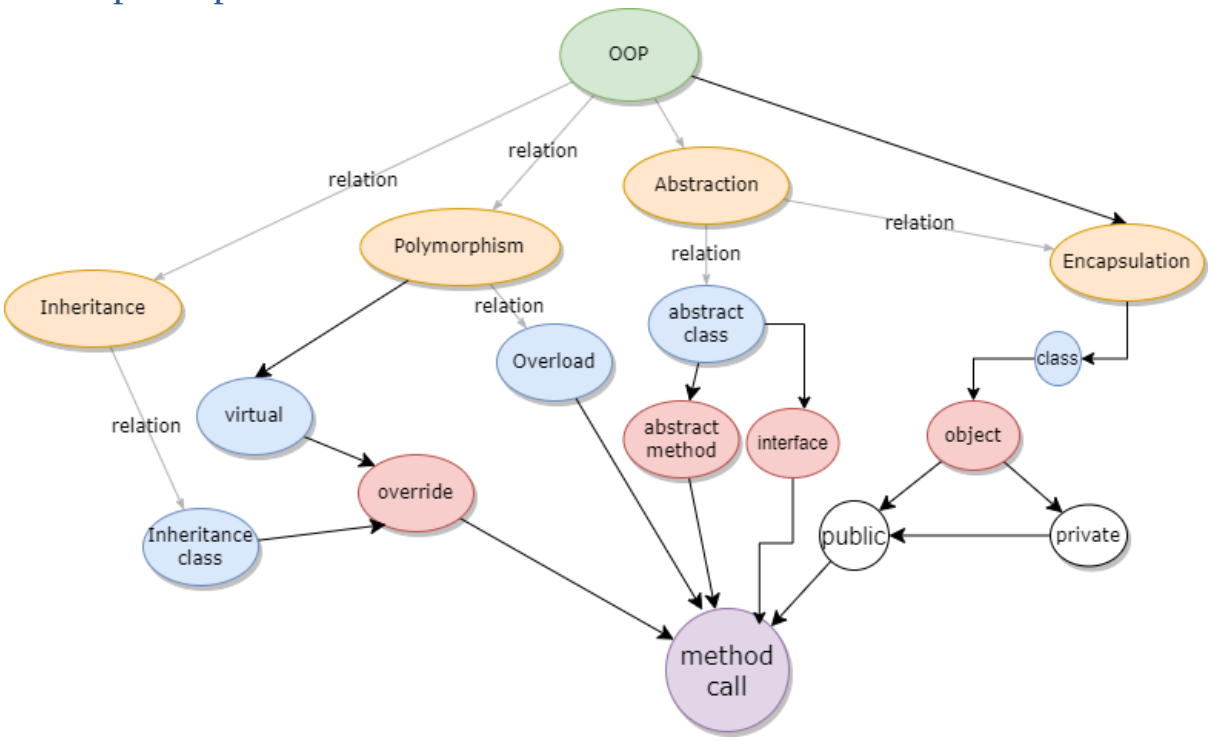
Thanh Minh

COS20007 Object Oriented Programming

Principles and Concepts

- **Abstraction**: it is the process of OOP to display the common and important features to the users and hide the complex details of the program. It helps to reduce the complexity of the program and also minimize the code . It also can be achieved with the combinations of encapsulation which will help the written code cleaner and easy to maintainable. In the example of pets, we only focus of how it can help us with daily tasks or entertain us rather than understand deeply about how it can do that.
- **Encapsulation**: it is the process of preventing the direct access to the data so that prevent the data is modified accidentally. It only allow the access through the defined functions which improve the data security and help to minimize the code , the code can be reused. For example, in the pets' case is that when call method for the cat to move, you need the data attributes like toys, food, …
- **Inheritance**: it is the ability to give other class to inherit the attributes and items from the parent class. It can help us to reuse the code and reduce the redundancy for minimizing the error in code can occur while coding. For example in the pets case, we can have parent class call Pets, for the child classes, they can be called as Cat, Dog, Hamster,.. and inherit the properties of the parent class.
- **Polymorphism**: it is the ability of the objects can be in multiple forms depend on the context. It can be achieved through the method call override, which subclass provides its own implementation of a method already present in the parent class. Another method can be used is overloading where the class has many methods with the same name, but the parameters are different. For example, in pets' case, cats can have various types Ragdoll, Persian… but it will eat different types of food.
- **Interface**: is the ability of the class which it only know what it should do instead of how it should do. It is used as the common way to interacting with the different objects. For example, in the payment method for billing system, we only know about methods for each payment Visa, Mastercard but don't know how it work.
    - **Roles**: are the behaviors and the attributes of the objects in the context
    - **Responsibilities**: this term refer to the functions of the object to perform
    - **Collaborations**: refer to the interactions of the objects.
    - **Coupling**: refer to how one class depend on another class.
    - **Cohesion**: refer to how different classes related to each other
    - **Class**: it defines the properties of the objects
    - **Object**: refer to the instance of a class and is defined by class
    - **Method**: refer to the function use object.
    - **Value type**: the type contains its data and store the value in the memory location
    - **Reference type**: refer to the address of where the data is stored.
    - **Abstract class**: it is used as a base for other classes and contain the abstract methods.
    - **Abstract method**: is the method defined in abstract class without body.
    - **Private**: refer to the restricts access to members of a class
    - **Public**: refer to the public access to members of a class
    - **Protected**: refer to the protected access within the class and other used classes
    - **Overload**: refer to the method allow multiple methods with same name but different types and used in the same class
    - **Override**: refer to the method allow the derived class to implement the method which has defined in the base class
    - **Virtual**: refer to the method can be overridden by the derived class

Concept Map



- OOP contain 4 concepts: **inheritance**, **polymorphism**, **abstraction**, encapsulation.
- The **inheritance class** can be in relations with the **override** class.
- **Class** contains the properties and behaviors of the **objects.**
- **Objects** can be **private** and **public.**
- The **public object** can be used within the **method call.**
- **Interface** is part of the **abstract** which the class can implement.
- **Abstract class** contain **abstract methods.**