
# ĐẠI HỌC XÂY DỰNG HÀ NỘI

## BM CNPM – KHOA CNTT

# Bài 9

## Kế thừa và đa hình

# Nội dung

- Nạp chồng phương thức
- Nạp chồng constructor
- Kế thừa
- Kế thừa constructor
- Ghi đè
- Lớp sealed
- Tính đa hình trong C#

# Nạp chồng phương thức 1-2

- Nạp chồng phương thức (Overloading method) là khả năng của một lớp cho phép định nghĩa nhiều phương thức cùng tên nhau
- Loại phương thức dạng này phải thỏa mãn các điều kiện sau:
  - Có tên giống nhau
  - Khác nhau về số lượng tham số
  - Cùng số lượng tham số nhưng phải khác nhau về kiểu tham số

# Nạp chồng phương thức 2-2

Ví dụ

```
class NapChong
{
    static int Sum(int a)
    {
        int s = 0;
        for (int i = 1; i <= a; i++)
        {
            s += i;
        }
        return s;
    }
    static int Sum(int a, int b)
    {
        int s = 0;
        for (int i = a; i <= b; i++)
        {
            s += i;
        }
        return s;
    }
    static int Add(int a, int b)
    {
        return a + b;
    }
    static double Add(double a, double b)
    {
        return a + b;
    }
}
```

# Nạp chồng constructor

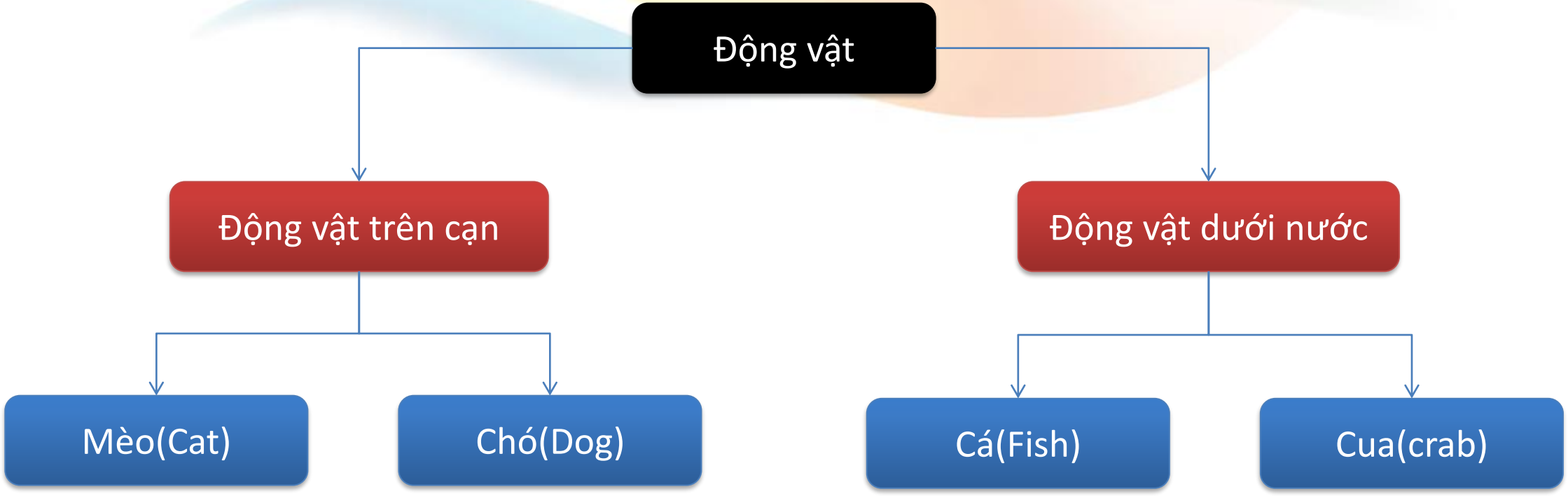
Ví dụ

```
class Student
{
    private string id;
    private string name;
    private double mark;
    public Student()
    {
    }
    public Student(string id,string name,double mark)
    {
        this.id = id;
        this.name = name;
        this.mark = mark;
    }
    public Student(string name, double mark)
    {
        this.name = name;
        this.mark = mark;
    }
}
```

# Kế thừa 1-3

- Kế thừa là cơ chế cho phép định nghĩa một lớp mới kế thừa từ lớp cha
- Sau đó xây dựng thêm các thuộc tính và phương thức riêng của lớp đó
- Lớp cha trong sự kế thừa được gọi là lớp cơ sở (base class)
- Lớp con trong sự kế thừa được gọi là lớp dẫn xuất (Derived class)
- Derived class được kế thừa tất cả những thành phần của base class ngoại trừ những thành phần là private

# Kế thừa 2-3




**Mô hình kế thừa của các loài đông vật**

# Kế thừa 3-3

Ví dụ

```
class Animal
{
    public void Eat()
    {
        Console.WriteLine("Dong vat an mot vai thu");
    }
    public void DoSomething()
    {
        Console.WriteLine("Dong vat lam mot vai thu");
    }
}
class Cat : Animal
{
    public void Actions()
    {
        Eat();
        DoSomething();
    }
}
```

# Bổ từ "protected"

- Bổ từ "protected" sử dụng cho các thành viên của lớp cơ sở, nó quy định các thành viên đó có thể được truy ở lớp dẫn xuất.

Ví dụ

```
class Animal
{
    protected string food;
    protected string activity;
}
class Cat : Animal
{
    public void Show()
    {
        Console.WriteLine("Meo an:" + food);
        Console.WriteLine("Meo :" + activity);
    }
}
```

# Từ khóa "base"

- Từ khóa "base" cho phép bạn truy cập tới các biến và phương thức của lớp cơ sở từ lớp dẫn xuất, do khi kế thừa các biến hoặc phương thức ở lớp cơ sở có thể bị định nghĩa lại ở lớp dẫn xuất.

Ví dụ

```
class Animal
{
    public void Eat() { Console.WriteLine("Animal eating"); }
}
class Dog : Animal
{
    public void Eat() { Console.WriteLine("Dog eating"); }
    public void DoAction()
    {
        Dog d = new Dog();
        d.Eat();
        base.Eat();
    }
}
```

# Từ khóa "new" 1-2

- Từ khóa "new" ngoài việc dùng như một toán tử để khởi tạo đối tượng, no còn có thể được sử dụng như một bổ từ trong C#.
- Khi lớp con kế thừa từ lớp cha và tạo ra một phương thức giống phương thức lớp cha, lúc này từ khóa new được sử dụng để tạo ra một phiên bản mới của phương này so với phương thức lớp cha, hay chúng ta có thể nói phương thức này sẽ làm ẩn và thay thế phương thức lớp cha.

# Từ khóa "new" 2-2

Ví dụ

```
class Employee
{
    int id=10;
    string name="Tran Thu Ha";
    int age = 20;
    protected void Display()
    {
        Console.WriteLine("Emp Id:" + id);
        Console.WriteLine("Emp Name:"+ name);
        Console.WriteLine("Age:" + age);
    }
}
class Department:Employee
{
    int depid = 20;
    string depname = "Information technology";
    //đây là phiên bản mới của phương thức display ở lớp cha
    public new void Display()
    {
        base.Display();
        Console.WriteLine("Dep Id:" + depid);
        Console.WriteLine("Dep Name:" + depname);
    }
}
```

# Kế thừa constructor 1-2

- Các phương thức khởi tạo sẽ được gọi khi thể hiện của lớp được tạo, trong C# bạn không thể kế thừa phương thức khởi tạo giống các phương thức thường, tuy nhiên bạn có thể gọi constructor lớp cơ sở trong lúc tạo constructor của lớp dẫn xuất.

Ví dụ

```
class Animal
{
    string name;
    public Animal(string name)
    {
        this.name = name;
    }
}
class Canime : Animal
{
    public Canime()
        : base("Lion")//gọi constructor lớp cơ sở
    {
        Console.WriteLine("Derived Canime");
    }
}
```

# Kế thừa constructor 2-2

Ví dụ

```
class Person
{
    string id;
    string name;
    string address;
    string phone;
    public Person(string id, string name, string address, string phone)
    {
        this.id = id; this.name = name; this.address = address; this.phone = phone;
    }
}
class Staff : Person
{
    int salary;
    public Staff(string id, string name, string address, string phone, int salary)
        : base(id, name, address, phone) //gọi constructor lớp cơ sở
    {
        this.salary = salary;
    }

}
```

# Ghi đè phương thức 1-3

- Ghi đè phương thức (Overriding method) là khi phương thức đã xuất hiện ở lớp cha và xuất hiện tiếp ở lớp con.
- Khi đối tượng thuộc lớp con gọi phương thức thì sẽ chọn lựa và chạy theo phương thức trong lớp con.
- Nếu lớp con không có phương thức đó thì mới lên kiếm ở lớp cha để chạy.
- Phương thức ghi đè có cùng tên, cùng tham số truyền vào, cùng kiểu giá trị trả về với phương thức ở lớp cha!

# Ghi đè phương thức 2-3

- Để thực thi ghi đè phương thức, bạn cần khai báo phương thức trong lớp cơ sở sử dụng từ khóa **virtual**. Trong lớp dẫn xuất, bạn cần khai báo phương thức ghi đè với từ khóa **override**. Đây là điều bắt buộc với bất kỳ phương thức **virtual** nào.

# Ghi đè phương thức 3-3

Ví dụ

```
public class Employee
{
    //khai báo tên và lương cơ bản
    public string name;
    protected decimal basepay;
    //khai báo constructor
    public Employee(string name, decimal basepay)
    {
        this.name = name;
        this.basepay = basepay;
    }
    // khai báo phương thức ảo có thể được ghi đè ở lớp con
    public virtual decimal CalculatePay()
    {
        return basepay;
    }
}
// khai bao lớp nhân viên bán hàng kế thừa từ lớp Employee.
public class SalesEmployee : Employee
{
    // khai báo thêm trường thưởng
    private decimal salesbonus;
    // Khai báo constructor và gọi constructor lớp cha
    public SalesEmployee(string name, decimal basepay,
              decimal salesbonus) : base(name, basepay)
    {
        this.salesbonus = salesbonus;
    }
    // Ghi đè phương thức CalculatePay
    public override decimal CalculatePay()
    {
        return basepay + salesbonus;
    }
}
```

# Lớp cô lập (sealed class)

- Lớp sealed là lớp không cho phép các lớp khác kế thừa, khi khai báo lớp bạn bổ sung từ khóa sealed vào trước từ khóa class.

Ví dụ

```
//khai báo lớp sealed
sealed class Product
{
    string id;
    string name;
    int price;
    int quantity;
    public Product(string id, string name, int price, int quantity)
    {
        this.id = id;
        this.name = name;
        this.price = price;
        this.quantity = quantity;
    }
}
//biên dịch máy sẽ báo lỗi dòng này
class Pen:Product
{

}
```

# Tính đa hình trong C#

| Đa hình tại thời điểm biên dịch (compile-time) | Đa hình tại thời điểm chạy (Run- time) |
|---|---|
| Được triển khai thông qua việc nạp chồng phương thức (overloading method) | Được triển khai thông qua việc ghi đè phương thức (overriding method) |
| Được thực thi tại thời điểm biên dịch, từ đó trình biên dịch biết phương thức nào sẽ thực thi phụ thuộc vào số tham số và kiểu dữ liệu | Được thực thi tại thời điểm chạy, do đó trình biên dịch không biết phương thức của lớp cơ sở hay lớp dẫn xuất sẽ được thực thi. |
| Được coi như dạng đa hình tĩnh (static polymorphism) | Được coi như dạng đa hình động (dynamic polymorphism) |

# Question & Answer