Bài 04

Kế thừa và đa hình

Lớp trừu tượng và giao diện

Mục tiêu “Kế thừa và đa hình”

- 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

DEVMASTER.VN

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

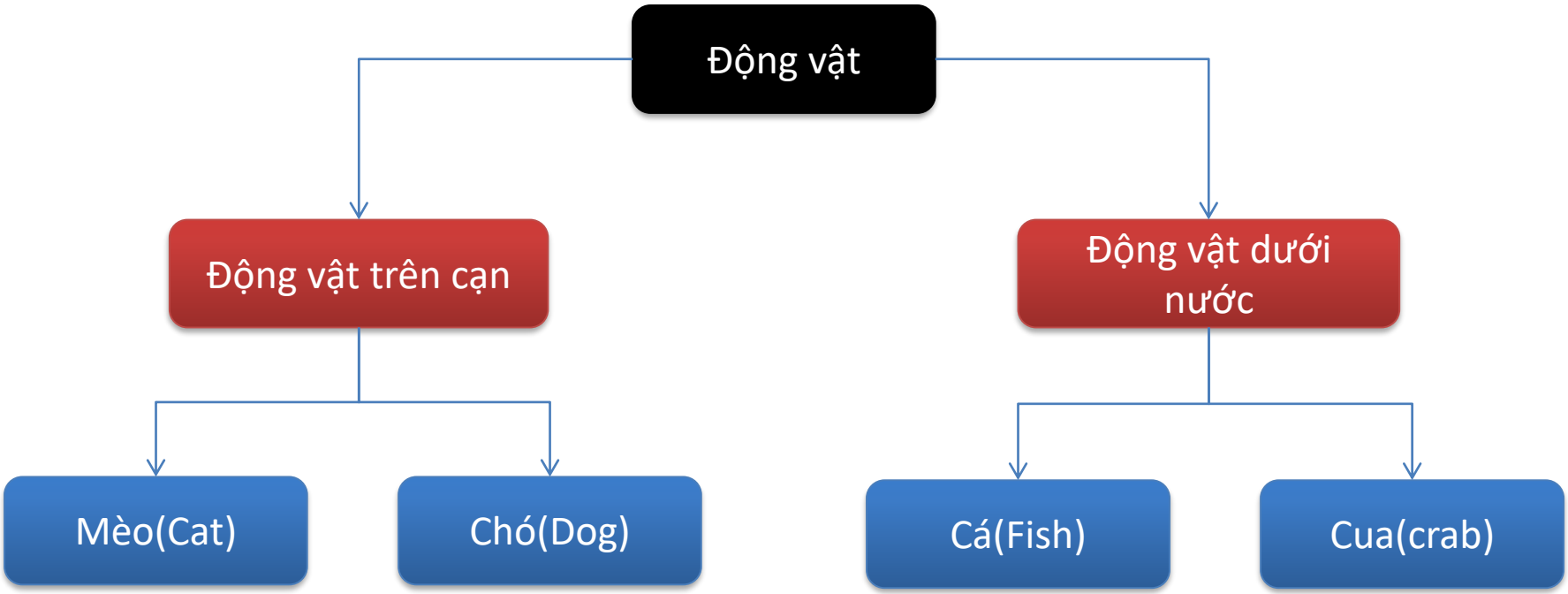
Ví dụ

```
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 đồng 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, nó 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
- Từ khóa new được sử dụng để tạo ra một phiên bản mới của phương so với phương thức lớp cha
- Chúng ta có thể nói phương thức này sẽ làm ẩn và thay thế phương thức lớp cha.

Từ khóa “new” 2-2

DEVMASTER.VN

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

DEVMASTER.VN

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

DEVMASTER.VN

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 là khả năng một thực thể có cách ứng xử khác nhau trong những tình huống khác nhau.
- Đa hình trong C# thể hiện ở hai loại phương thức Overloading và Overidding.

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)

Mục tiêu "Lớp trừa tượng và giao diện"

- Lớp trừu tượng
- Giao diện
- Thực thi giao diện
- Đa kế thừa giao diện
- Thực thi giao diện tường minh
- Kế thừa giao diện
- Sự khác nhau giữa lớp trừu tượng và giao diện

Lớp trừu tượng 1-3

- Lớp trừu tượng (abstract class) là một lớp cơ sở chưa hoàn thành
- Bên trong lớp có một số phương thức trừu tượng chỉ khai báo mà chưa triển khai nội dung.
- Lớp trừu tượng không cho phép tạo thể hiện, nhưng nó cho phép kế thừa.
- Các lớp con được kế thừa
  - Phải thực thi tất cả các phương thức trừu tượng được khai báo trong lớp cơ sở
  - Nếu không thì phải khai báo là abstract

Lớp trừu tượng 2-3

- **Cú pháp**

```
public abstract class <tên_lớp>
{
        //khai báo các phương thức trừu tượng
        <phạm_vi> abstract <kiểu_dữ_liệu> <tên_phương_thức>([tham_số]);
        //các phương thức không trừu tượng khác
}
```

- **Lưu ý:** Khi thực thi các phương thức trừu tượng ở lớp cơ sở tại lớp con, bạn cần thêm từ khóa override vào tương tự việc ghi đè trong kế thừa.

Lớp trừu tượng 3-3

DEVMASTER.VN

Ví dụ

```
//khai báo lớp trừu tượng
public abstract class Person
{
    //phương thức không trừu tượng
    public void Speak()
    {
        Console.WriteLine("Moi nguoi deu noi Tieng Viet");
    }
    //phương thức trừu tượng
    public abstract void DoWork();
    public abstract void EnvironmentWork();
}
class Employee : Person
{
    //thực thi các phương thức trừu tượng ở lớp cơ sở
    public override void DoWork()
    {
        Console.WriteLine("Lam viec rat vat va");
    }

    public override void EnvironmentWork()
    {
        Console.WriteLine("Lam viec tai cac nha may rat doc hai");
    }
}
```

Giao diện (Interface) 1-4

- **Mục đích:**
  - Giải quyết vấn đề đa kế thừa trong C#
  - Mặc định một lớp trong C# chỉ cho phép kế từ một lớp khác, nhưng nó có thể thực thi từ nhiều giao diện.
- **Giao diện là gì?:**
  - Là ràng buộc, giao ước đảm bảo cho các lớp hay các cấu trúc sẽ thực hiện một điều gì đó.
  - Khi một lớp thực thi một giao diện, lớp này phải thực thi tất cả các phương thức của giao diện. Đây là một bắt buộc mà các lớp phải thực hiện.

Giao diện (Interface) 2-4

- **Cú pháp:**

```
interface <tên_giao_diện>
{
    //khai báo các thành viên
}
```

Ví dụ

```
//khai báo giao diện
public interface IStorable
{
    void Read();
    void Write(Object data);
}
```

Lưu ý

- Không được sử dụng bất kỳ phạm vi truy cập nào khi khai báo các thành viên của giao diện(mặc định là public)
- Không được sử dụng từ khóa abstract khi khai báo các thành viên giao diện (mặc định là abstract)

Thực thi giao diện

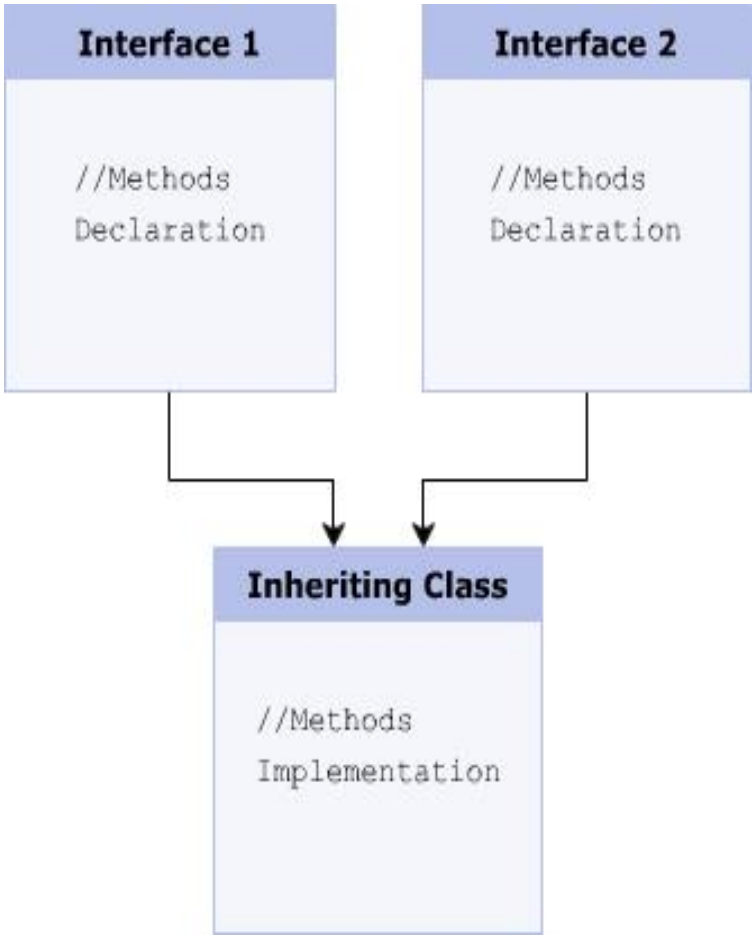
Ví dụ

```
//khai báo giao diện
public interface IStorable
{
     void Read();
     void Write(Object data);
}
//khai báo lớp thực thi từ giao diện
public class Document : IStorable
{
    #region "thực thi các phương thức từ giao diện IStorable"
    public void Read()
    {
        Console.WriteLine("Doc du lieu tu tai lieu");
    }

    public void Write(object data)
    {
        Console.WriteLine("Ghi du lieu "+data+" vao tai lieu");
    }
    #endregion
}
```

Giao diện và đa kế thừa 1-2

- Một lớp có thể thực thi từ nhiều giao diện, khi thực thi thì mỗi giao diện cách nhau bởi dấu phẩy.
- Một lớp thực thi từ nhiều giao diện thì phải thực thi tất cả các phương thức trừu tượng được khai báo trong các giao diện
- Từ khóa override không được sử dụng trong khi thực thi các phương thức trừu tượng của giao diện.



Giao diện và đa kế thừa 2-2

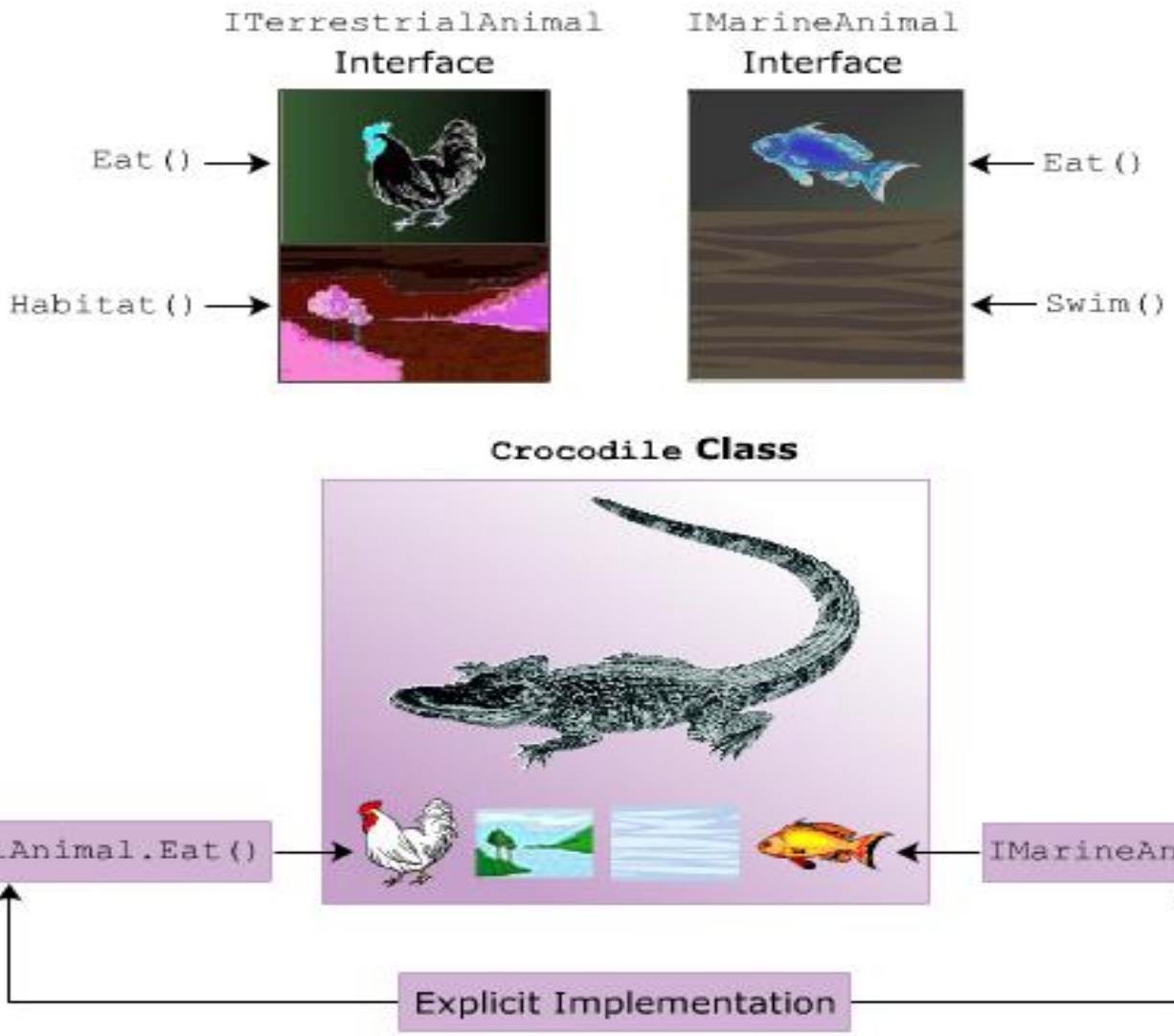
DEVMASTER.VN

Ví dụ

```
//khai báo giao diện 1
public interface ICalculator1
{
    int Add(int a, int b);
    int Mul(int a, int b);
}
public interface ICalculator2
{
    int Sub(int a, int b);
    int Div(int a, int b);
}
//thực thi đa giao diện
class Caculation : ICalculator1, ICalculator2
{
    public int Add(int a, int b)
    {
        return a + b;
    }
    public int Mul(int a, int b)
    {
        return a* b;
    }
    public int Sub(int a, int b)
    {
        return a - b;
    }
    public int Div(int a, int b)
    {
        return a / b;
    }
}
```

- Khi các giao diện khai báo các phương thức thành viên trùng tên nhau
  - Lúc này lớp thực thi từ các giao diện đó phải thực thi tường minh các phương thức trùng tên nhau
  - Trường hợp này gọi là thực thi giao diện tường minh

Thực thi giao diện tường minh 1-2



Thực thi giao diện tường minh 2-2

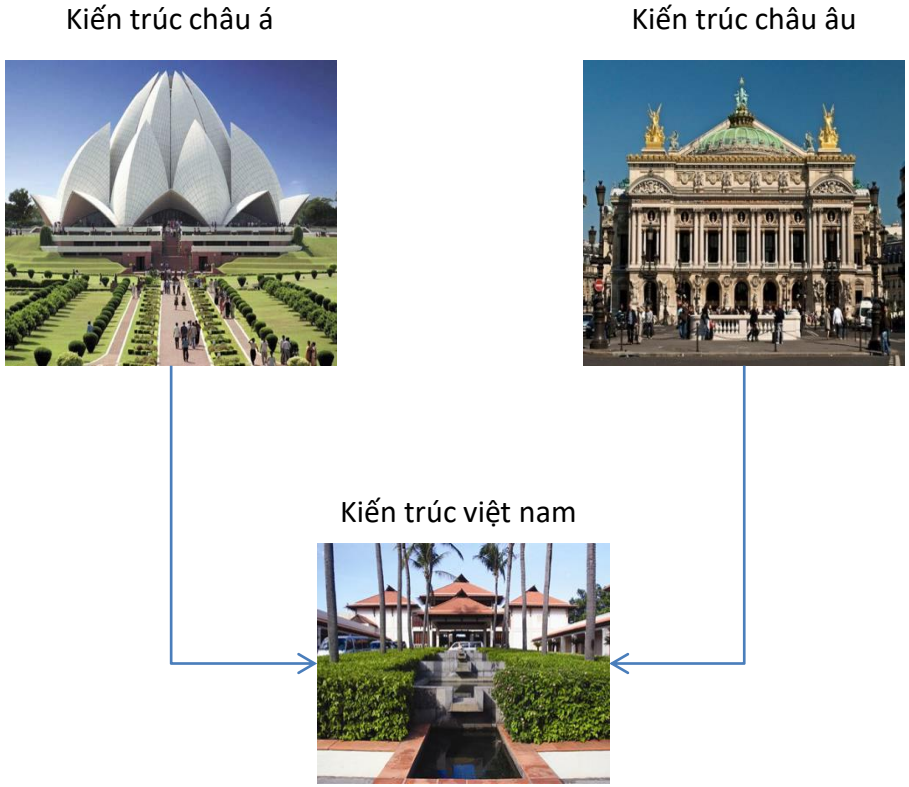
DEVMASTER.VN

Ví dụ

```
//ví dụ thi giao diện tường minh
interface ITerrestrialAminal
{
    void Eat();
}
interface IMarineAnimal
{
    void Eat();
}
class Crocodie : ITerrestrialAminal, IMarineAnimal
{
    //thực thi tường minh 2 phương thức của 2 giao diện
    void ITerrestrialAminal.Eat()
    {
        Console.WriteLine("Eat on Terrestrial");
    }
    void IMarineAnimal.Eat()
    {
        Console.WriteLine("Eat on Marine");
    }
    //public các phương thức ra ngoài khi cần sử dụng
    public void EatTerrestrial()
    {
        ITerrestrialAminal ta = this;
        ta.Eat();
    }
    public void EatMarine()
    {
        IMarineAnimal ma = this;
        ma.Eat();
    }
}
```

Kế thừa giao diện 1-2

- Một giao diện có thể kế thừa từ nhiều giao diện khác, nhưng không thể thực thi chúng, việc thực thi sẽ được giao cho các lớp.



Kế thừa giao diện 2-2

DEVMASTER.VN

Ví dụ

```
//ví dụ về kế thừa giao diện
interface IEurope
{
    void BuildAncient();
}
interface IAsia
{
    void BuildModern();
}
interface IVietNam:IEurope, IAsia
{
    void BuildMixed();
}
class BuildingKeangnam : IVietNam
{

    public void BuildMixed()
    {
        Console.WriteLine("Kien truc pha tron");
    }

    public void BuildAncient()
    {
        Console.WriteLine("Kien truc co kinh");
    }

    public void BuildModern()
    {
        Console.WriteLine("Kien truc hien dai");
    }
}
```

Lớp trừu tượng và giao diện

Abstract Classes	Interfaces
Lớp trừu tượng có thể kế thừa từ một lớp hoặc nhiều giao diện	Giao diện có thể kế thừa từ nhiều giao diện nhưng không thể kế thừa từ lớp
Lớp trừu tượng có thể có phương thức triển khai nội dung	Giao diện chỉ chứa các phương thức hoàn toàn trừu tượng
Phương thức trong lớp trừu tượng được thực thi sử dụng từ khóa override	Phương thức trong giao diện không cần sử dụng từ khóa override
Lớp trừu tượng là tùy chọn tốt khi bạn cần thực thi các phương thức chung và khai báo phương thức trừu tượng	Giao diện là tùy chọn tốt khi bạn cần khai báo các phương thức trừu tượng
Lớp trừu tượng có thể khai báo constructor và destructor	Giao diện không thể khai báo constructors hoặc destructor

Hỏi Đáp