

LẬP TRÌNH HƯỚNG ĐỐI TƯỢNG

1. MỘT SỐ KHÁI NIỆM CƠ BẢN	2
a. Đối tượng là gì?	2
b. Lớp là gì?	2
c. Sự khác nhau giữa đối tượng và lớp	2
2. PHÂN TÍCH SƠ BỘ VỀ CLASS	3
a. Cấu trúc của 1 class:	3
1. <i>Thuộc tính:</i>	3
2. <i>Phương thức:</i>	10
<i>Overloaded constructor</i>	12
3. CÁC TÍNH CHẤT CỦA LẬP TRÌNH HƯỚNG ĐỐI TƯỢNG	14
a. Tính đóng gói (Encapsulation)	14
b. Tính kế thừa (Inheritance)	18
c. Tính đa hình (Polymorphism)	29
d. Tính trừu tượng (Abstraction)	36

LẬP TRÌNH HƯỚNG ĐỐI TƯỢNG

1. MỘT SỐ KHÁI NIỆM CƠ BẢN

a. Đối tượng là gì?

Đối tượng là những sự vật, sự việc có những tính chất, đặc tính, hành động giống nhau, ta gom chúng lại thành đối tượng giống trong thực tiễn cuộc sống. VD các học sinh đều mang những đặc điểm như đặc điểm ngoại hình, thông tin học sinh (Mã học sinh, họ tên, địa chỉ, lớp, ...). Vậy nên ta có thể gộp chung các học sinh lại để thể hiện trong lập trình hướng đối tượng dưới dạng 1 đối tượng (Object) có tên là HocSinh với những đặc điểm chung được mô tả trong đó.

Một đối tượng bao gồm 2 thông tin: **thuộc tính** và **phương thức**.

- **Thuộc tính** là những thông tin, đặc điểm của đối tượng. VD: con người có các đặc điểm như mắt, mũi, tay, chân, màu mắt, màu da, ...
- **Phương thức** là những thao tác, hành động mà đối tượng đó có thể thực hiện. VD: một người có thể thực hiện các hành động như nói, đi, ăn, uống, ...

b. Lớp là gì?

Một lớp là một kiểu dữ liệu bao gồm các thuộc tính và các phương thức được định nghĩa từ trước. Đây là sự trừu tượng hóa của đối tượng. Khác với kiểu dữ liệu thông thường, một lớp là một đơn vị (trừu tượng) bao gồm sự kết hợp giữa các phương thức và các thuộc tính. Hiểu nôm na hơn là các đối tượng có các đặc tính tương tự nhau được gom lại thành một lớp đối tượng.

c. Sự khác nhau giữa đối tượng và lớp

Lớp ở đây bạn có thể hiểu như là các khuôn mẫu, đối tượng là các thực thể được thể hiện dựa trên khuôn mẫu đó.

Vậy lớp Người theo như lý thuyết là các khuôn mẫu, vậy đối tượng được thể hiện dựa trên khuôn mẫu trên là gì?

Ví dụ ta nói đến **lớp Người**:

- **Đặc điểm, thông tin:** Họ tên , ngày sinh, chiều cao, cân nặng, màu mắt, màu da, ... (Tóm lại đây là những thông tin chung, cơ bản nhất mà một người có).
- **Các hành động:** nói chuyện, ăn, ngủ, đi, chạy, nhảy, ... (Các hành động chung, cơ bản nhất mà một con người có).

Để trả lời câu hỏi trên, các bạn có thể xem VD sau:

ĐỐI TƯỢNG (OBJECT) HỌC SINH

Học sinh là một đối tượng được thể hiện dựa trên khuôn mẫu trên bởi học sinh bản chất cũng là người, mà nếu là một người thì sẽ mang đầy đủ các đặc điểm, hành động của lớp Người được mô tả ở trên. Tuy nhiên với đối tượng học sinh sẽ có điểm khác biệt so với lớp người, thứ mà khiến nó được coi là 1 thể hiện của lớp Người đó chính là những thông tin riêng của học sinh (VD: Mã học sinh, tên trường học, tên lớp, điểm thi, ...).

Một lớp sẽ chỉ thể hiện những điểm chung nhất của các sự vật liên quan đến lớp đó. Đối tượng sẽ là những biến thể được kế thừa những thông tin cơ bản của lớp, và mang trong mình những đặc điểm riêng biệt, không thuộc về lớp.

2. PHÂN TÍCH SƠ BỘ VỀ CLASS

a. Cấu trúc của 1 class:

1. Thuộc tính:

a. Field (Trường dữ liệu):

Cú pháp: *<phạm vi truy cập> <kiểu dữ liệu> <tên biến>;*

Ví dụ:

Java:

```
private String maHocSinh;  
private double diemTB;  
private String maLop;
```

C#:

```
private string _maHocSinh;  
private double _diemTB;  
private string _maLop;
```

b. Property (Thuộc tính):

Property dùng để quản lý sự truy cập đến field. Dùng property để đảm bảo field không bị gán các giá trị không hợp lệ.

Thuộc tính (property) là sự kế thừa của các field, chúng sử dụng các cơ chế get, set (getter, setter) để hỗ trợ việc trao đổi dữ liệu với các private field.

+ *Getter ở đây là cơ chế giúp xử lý đầu ra của dữ liệu, nó sẽ chạy khi ta thực hiện lấy dữ liệu ra để tính toán hoặc hiển thị bằng cách gọi thuộc tính.*

+ *Setter là cơ chế giúp xử lý đầu vào của dữ liệu, nó sẽ chạy khi ta thực hiện nhập dữ liệu cho thuộc tính.*

Ví dụ:

Java:

```
private String maHocSinh;  
private double diemTB;  
private String maLop;  
  
public String getMaHocSinh() {  
    return maHocSinh;  
}  
  
public void setMaHocSinh(String maHocSinh) {  
    this.maHocSinh = maHocSinh;  
}  
  
public double getDiemTB() {  
    return diemTB;  
}  
  
public void setDiemTB(double diemTB) {  
    this.diemTB = diemTB;  
}  
  
public String getMaLop() {  
    return maLop;  
}  
  
public void setMaLop(String maLop) {  
    this.maLop = maLop;  
}
```

|

Với Java, việc truy cập và trao đổi dữ liệu với các private field sẽ phải thông qua các cơ chế get, set như ví dụ trên, vì Java không có thuộc tính nên các cơ chế nhập, xuất (về bản chất là các hàm hỗ trợ nhập xuất) sẽ thay thế cho thuộc tính giúp các class bên ngoài có thể tương tác được với các private field.

C#:

```
private string _maHocSinh;  
private double _diemTB;  
private string _maLop;  
  
0 references  
public string MaHocSinh  
{  
    get { return _maHocSinh; }  
    set { _maHocSinh = value; }  
}  
  
0 references  
public double DiemTB  
{  
    get { return _diemTB; }  
    set { _diemTB = value; }  
}  
  
0 references  
public string MaLop  
{  
    get { return _maLop; }  
    set { _maLop = value; }  
}
```

Với C#, việc trao đổi dữ liệu với các private field sẽ do các thuộc tính đảm nhận, các thuộc tính hình thức gần giống như các field, cũng có phạm vi truy cập, cũng có kiểu dữ liệu và tên biến. Tuy nhiên thuộc tính sẽ có thêm các cơ chế get, set để hỗ trợ class bên ngoài giao tiếp với các private field.

Các bạn có thể thắc mắc, việc thiết lập các cơ chế nhập xuất có vẻ hơi thừa bởi nếu viết như trên, các class bên ngoài hoàn toàn có thể truy cập và thay đổi trực tiếp dữ liệu của đối tượng giống như việc truy cập trực tiếp vào các field, chỉ khác là tốn công hơn một chút vì phải qua getter và setter.

Java:

```
public class HocSinh extends Nguoi {  
    private String maHocSinh = "HS1";  
    private double diemTB = 5;  
    private String maLop = "12A1";  
}
```

```
public class Main {  
    public static void main(String[] args) {  
        HocSinh hocSinh = new HocSinh();  
        hocSinh.setMaHocSinh("HS2");  
        hocSinh.setDiemTB(10);  
        hocSinh.InThongTinHocSinh();  
    }  
}
```

Main (1) ×

```
"C:\Program Files\Java\jdk-11.0.9\bin\java.exe" "-jav  
HocSinh{maHocSinh='HS2', diemTB=10.0, maLop='12A1'}
```

C#:

2 references

```
class HocSinh : Nguoi  
{  
    private string _maHocSinh = "HS1";  
    private double _diemTB = 5;  
    private string _maLop = "12A1";  
}
```

0 references

```
class Program  
{  
    0 references  
    static void Main(string[] args)  
    {  
        HocSinh hocSinh = new HocSinh();  
        hocSinh.MaHocSinh = "HS2";  
        hocSinh.DiemTB = 10;  
        hocSinh.InThongTinHocSinh();  
    }  
}
```

Microsoft Visual Studio Debug Console

```
HocSinh{maHocSinh='HS2', diemTB=10, maLop='12A1'}
```

Các bạn có thể thấy ở ví dụ trên, ban đầu dữ liệu của học sinh được đặt với mã là HS1, điểm thi 5 và các dữ liệu này đang ở dạng private (class bên ngoài không thể truy cập được). Tuy nhiên với sự xuất hiện của các cơ chế get, set, việc thay đổi dữ liệu lại dễ hơn bao giờ hết, mình chứng cho việc đó là bạn hoàn toàn có thể thay đổi dữ liệu của học sinh đổi mã học sinh và điểm của học sinh đó dễ dàng.

Vậy có lẽ getter và setter vô dụng vậy sao?

Câu trả lời là không, việc truy cập được như trên là do ta chưa tận dụng được khả năng quản trị dữ liệu của chúng, với getter và setter, ta hoàn toàn có thể khống chế dữ liệu vào ra, về bản chất ta vẫn cho class bên ngoài tương tác với dữ liệu bên trong, tuy nhiên tương tác thế nào, thực hiện ra làm sao thì hoàn toàn phải theo các quy tắc ta đặt ra trong getter và setter. Ví dụ để truy cập và đổi điểm của học sinh, ta sẽ yêu cầu nhập tài khoản, mật khẩu, nếu đúng thì mới cho thay đổi điểm thành điểm mới.

Java:

```
public void setDiemTB(double diemTB) {
    Scanner sc = new Scanner(System.in);
    System.out.printf("Nhap tai khoan: ");
    String taiKhoan = sc.nextLine();
    System.out.printf("Mat khau: ");
    String matKhau = sc.nextLine();
    if (taiKhoan.equals("admin") && matKhau.equals("admin")) {
        this.diemTB = diemTB;
    } else {
        System.out.println("Ban khong co quyen thay doi diem");
    }
}
```

Kết quả:


```
Nhap tai khoan: asdf
Mat khau: asdf
Ban khong co quyen thay doi diem
HocSinh{maHocSinh='HS2', diemTB=5.0, maLop='12A1'}
```

C#:

```
1 reference
public double DiemTB
{
    get { return _diemTB; }
    set
    {
        Console.Write("Nhap tai khoan: ");
        string taiKhoan = Console.ReadLine();
        Console.Write("Mat khau: ");
        string matKhau = Console.ReadLine();
        if (taiKhoan.Equals("admin") && matKhau.Equals("admin"))
        {
            _diemTB = value;
        }
        else
        {
            Console.WriteLine("Ban khong co quyen thay doi diem");
        }
    }
}
```

```
Nhap tai khoan: asdf
Mat khau: asdf
Ban khong co quyen thay doi diem
HocSinh{maHocSinh='HS2', diemTB=5, maLop='12A1'}
```

Các bạn có thể thấy, mình đã yêu cầu nhập tài khoản, mật khẩu trong quá trình nhập dữ liệu do setter quản lý, hiểu đơn giản là với setter, ta có một đội vệ sĩ canh cổng, khi có người chuyển dữ liệu vào trong sẽ bắt buộc phải qua cổng này, và các vệ sĩ setter sẽ đóng vai trò kiểm soát dữ liệu đầu vào đó xem có phù hợp với quy tắc mà ta đề ra hay không

rồi mới đưa ra quyết định xử lý với dữ liệu đó. Vậy nên nếu ta thiết lập quy tắc, ai muốn vào (set) hay ra (get) thì phải tuân theo quy tắc đó, còn nếu không thiết lập, mở cổng tự do như ví dụ ban đầu thì class bên ngoài có thể thoải mái ra vào xảy ra cũng là điều hiển nhiên.

2. Phương thức:

a. Phương thức khởi tạo:

Phương thức khởi tạo (constructor) là một phương thức đặc biệt, nó được dùng để khởi tạo và trả về đối tượng của lớp mà nó được định nghĩa. Constructor sẽ có tên trùng với tên của lớp mà nó được định nghĩa và chúng không được định nghĩa một kiểu giá trị trả về.

Constructor bao gồm 2 loại: Constructor có tham số và constructor không tham số.

Java:

```
//Constructor không tham số  
public HocSinh() {  
}
```

```
//Constructor có tham số  
public HocSinh(String maHocSinh, double diemTB, String maLop) {  
    this.maHocSinh = maHocSinh;  
    this.diemTB = diemTB;  
    this.maLop = maLop;  
}
```

C#:

```

//Constructor không tham số
1 reference
public HocSinh()
{
}

//Constructor có tham số
0 references
public HocSinh(string maHocSinh, double diemTB, string maLop)
{
    _maHocSinh = maHocSinh;
    _diemTB = diemTB;
    _maLop = maLop;
}

```

Constructor được sử dụng khi ta tiến hành khởi tạo đối tượng, VD như việc khởi tạo đối tượng ở hàm main:

Java:

```
HocSinh hocSinh = new HocSinh();
```

C#:

```
HocSinh hocSinh = new HocSinh();
```

`new HocSinh()` ở đây chính là việc ta gọi đến hàm khởi tạo không tham số để khởi tạo 1 đối tượng `hocSinh`.

Note: Khi ta không khai báo hàm khởi tạo nào trong class thì mặc định khi chạy class đó sẽ tự tạo ngầm 1 hàm khởi tạo mặc định là hàm khởi tạo không tham số để sử dụng.

Tương tự với hàm khởi tạo có tham số, các bạn truyền giá trị vào cho hàm khởi tạo tương tự như truyền tham số vào hàm bình thường.

```
HocSinh hocSinh2 = new HocSinh( maHocSinh: "HS1", diemTB: 8, maLop: "12A2");
```

Overloaded constructor

Khi ta định nghĩa nhiều constructor cho một đối tượng và mỗi constructor sẽ có các tham số khác nhau cả về số lượng tham số lẫn kiểu dữ liệu của tham số đó thì công việc đó sẽ gọi là overloaded constructor.

VD:

```
public class Student {  
  
    private String name;  
  
    private int age;  
  
    public Student(int age) {  
        this.age = age;  
    }  
  
    public Student(String name) {  
        this.name = name;  
    }  
  
    public Student(String name, int age) {  
        this.name = name;  
        this.age = age;  
    }  
}
```

Các bạn có thể thấy, cùng là hàm khởi tạo, nhưng mỗi hàm lại khác nhau về tham số truyền vào, chỉ giống tên hàm và phạm vi truy cập, đó chính là việc chúng ta đang overloaded constructor.

NOTE: this ở đây là biến tham chiếu tới thành phần của lớp hiện tại, việc truyền tham số trùng tên với field hay property của class sẽ gây ra sự

nhằm lẫn nên ta dùng thêm this đặt phía trước thành phần thuộc về class.

b. Phương thức thường:

Phương thức thường ở đây chính là các phương thức, các hàm xử lý bên trong 1 class ngoài constructor (hàm khởi tạo).

Java:

```
public void NhapThongTinHocSinh() {  
    Scanner sc = new Scanner(System.in);  
    System.out.printf("Nhap ma hoc sinh: ");  
    maHocSinh = sc.nextLine();  
    System.out.printf("Nhap diem trung binh: ");  
    diemTB = sc.nextDouble();  
    System.out.printf("Nhap ma lop: ");  
    maLop = sc.nextLine();  
}
```

```
public void InThongTinHocSinh() {  
    System.out.println("HocSinh{" +  
        "maHocSinh='" + maHocSinh + '\\'  
        ", diemTB=" + diemTB +  
        ", maLop='" + maLop + '\\'  
        '}');  
}
```

C#:

```
0 references
public void NhapThongTinHocSinh()
{
    Console.Write("Nhap ma hoc sinh: ");
    _maHocSinh = Console.ReadLine();
    Console.Write("Nhap diem trung binh: ");
    _diemTB = double.Parse(Console.ReadLine());
    Console.Write("Nhap ma lop: ");
    _maLop = Console.ReadLine();
}

0 references
public void InThongTinHocSinh()
{
    Console.WriteLine("HocSinh{" +
        "maHocSinh='" + _maHocSinh + '\'' +
        ", diemTB=" + _diemTB +
        ", maLop='" + _maLop + '\'' +
        '}');
}
```

3. CÁC TÍNH CHẤT CỦA LẬP TRÌNH HƯỚNG ĐỐI TƯỢNG

a. Tính đóng gói (Encapsulation)

Các thuộc tính và phương thức của đối tượng cần cho việc giải quyết bài toán sẽ được đóng gói vào một kiểu dữ liệu là class nhằm che giấu thông tin khỏi các đối tượng bên ngoài. Việc truy cập vào dữ liệu trong class sẽ được quyết định bởi access modifier (phạm vi truy cập) của các thuộc tính, phương thức. Với class ta sẽ có 2 loại access modifier là public và default nhưng với biến, method (phương thức) thì ta có 4 loại access modifier (public, protected, default, private).

Link tham khảo phạm vi truy cập:

C#: [Các loại phạm vi truy cập trong Lập trình hướng đối tượng | How Kteam](#)

Java: [Thao tác trên đối tượng và phạm vi truy cập trong Java \(freetuts.net\)](#)

Ví dụ:

Java:

```
public class Nguoi {  
  
    //PRIVATE  
    private String hoTen;  
    private Integer tuoi;  
    private String diaChi;  
    private String soCMT;  
    private String soDienThoai;  
  
    //PUBLIC  
    public void NhapThongTin() {  
        Scanner sc = new Scanner(System.in);  
        System.out.printf("Nhap ho ten: ");  
        hoTen = sc.nextLine();  
        System.out.printf("Nhap tuoi: ");  
        tuoi = sc.nextInt();  
        System.out.printf("Nhap dia chi: ");  
        diaChi = sc.nextLine();  
        System.out.printf("Nhap so CMT: ");  
        soCMT = sc.nextLine();  
        System.out.printf("Nhap so dien thoai: ");  
        soDienThoai = sc.nextLine();  
    }  
  
    public void HienThiThongTin() {  
        System.out.println("Nguoi{" +  
            "hoTen='" + hoTen + '\\'' +  
            ", tuoi=" + tuoi +  
            ", diaChi='" + diaChi + '\\'' +  
            ", soCMT='" + soCMT + '\\'' +  
            ", soDienThoai='" + soDienThoai + '\\'' +  
            '}');  
    }  
}
```

C#:

```
0 references
class Nguoi
{
    #region PRIVATE
    private string _hoTen;
    private int _tuoi;
    private string _diaChi;
    private string _soCMT;
    private string _soDienThoai;
    #endregion

    #region PUBLIC
    0 references
    public void NhapThongTin()
    {
        Console.Write("Nhap ho ten: ");
        Console.Write("Nhap tuoi: ");
        Console.Write("Nhap dia chi: ");
        Console.Write("Nhap so CMT: ");
        Console.Write("Nhap so dien thoai: ");
    }

    0 references
    public void HienThiThongTin()
    {
        Console.WriteLine("Nguoi{" +
            "hoTen='" + _hoTen + '\'' +
            ", tuoi=" + _tuoi +
            ", diaChi='" + _diaChi + '\'' +
            ", soCMT='" + _soCMT + '\'' +
            ", soDienThoai='" + _soDienThoai + '\'' +
            '}');
    }
    #endregion
}
```


Trong đoạn code trên, tính đóng gói được thể hiện thông qua các field (trường dữ liệu) và các phương thức NhapThongTin() và HienThiThongTin() được gói vào trong class Nguoi. Bạn không thể truy cập đến các private data hoặc gọi đến các private methods của class từ bên ngoài class đó.

Java:

```
package JavaPractice.Java00PBasic.JavaLearn;
```

```
public class Main {  
    public static void main(String[] args) {  
        Nguoi nguoi = new Nguoi();  
        nguoi.hoTen = "Nguyen Van A";  
    }  
}
```

'hoTen' has private access in 'JavaPractice.Java00PBasic.JavaLearn.Nguoi'
Create field 'hoTen' in 'Nguoi' Alt+Shift+Enter More actions... Alt+Enter

JavaPractice.Java00PBasic.JavaLearn.Nguoi
private String hoTen

C#:

```
using System;  
  
namespace OOP_Test  
{  
    0 references  
    class Program  
    {  
        0 references  
        static void Main(string[] args)  
        {  
            Nguoi nguoi = new Nguoi();  
            nguoi.hoTen = "Nguyen Van A";  
        }  
    }  
}
```

class System.String
Represents text as a sequence of UTF-16 code units.

CS0122: 'Nguoi.hoTen' is inaccessible due to its protection level

Có thể nói, tính đóng gói (Encapsulation) là cơ chế của che giấu dữ liệu bởi chúng được lớp che giấu đi (ở dạng private) và đảm bảo các dữ liệu đó được truy cập và sử dụng đúng mục đích thông qua các hàm và phương thức ở dạng public mà class cung cấp. Đó là lý do bạn không thể truy cập đến các thuộc tính và phương thức private trong class. Như ở VD trên, các field được đặt ở dạng private dẫn đến các class bên ngoài chỉ được phép truy cập các dữ liệu đó thông qua 2 phương thức `NhapThongTin()`, `HienThiThongTin()` được đặt ở dạng public mà không thể truy cập trực tiếp dữ liệu từ các field đó.

Tính đóng gói được thể hiện khi có các thành phần mang trạng thái là private ở trong 1 class và những đối tượng khác không thể truy cập trực tiếp vào các dữ liệu private này. Thay vào đó chúng chỉ có thể gọi các hàm mang phạm vi public để tiến hành trao đổi dữ liệu với các thành phần private đó.

b. Tính kế thừa (Inheritance)

Tính kế thừa cho phép xây dựng một lớp mới (lớp con) dựa trên các định nghĩa của lớp đã có (lớp cha). Lớp cha ở đây có thể chi sẻ các dữ liệu về thuộc tính, phương thức cho các lớp con, các lớp con sẽ không cần phải định nghĩa lại các nội dung đó. Ngoài ra, lớp con có thể định nghĩa thêm các thành phần riêng biệt của nó và mở rộng các thành phần kế thừa để phục vụ mục đích mà lớp con đó đang thực thi. Điều này giúp ta có thể tái sử dụng mã nguồn một cách tối ưu nhất, tận dụng được các mã nguồn đã được định nghĩa trước.

Ví dụ:

Java: Lớp HocSinh kế thừa lớp Nguoi thông qua từ khoá extends

```
package JavaPractice.Java00PBasic.JavaLearn;

import java.util.Scanner;

public class HocSinh extends Nguoi {
    private String maHocSinh;
    private double diemTB;
    private String maLop;

    public void NhapThongTinHocSinh() {
        Scanner sc = new Scanner(System.in);
        System.out.printf("Nhap ma hoc sinh: ");
        maHocSinh = sc.nextLine();
        System.out.printf("Nhap diem trung binh: ");
        diemTB = sc.nextDouble();
        System.out.printf("Nhap ma lop: ");
        maLop = sc.nextLine();
    }

    public void InThongTinHocSinh() {
        System.out.println("HocSinh{" +
            "maHocSinh='" + maHocSinh + '\'' +
            ", diemTB=" + diemTB +
            ", maLop='" + maLop + '\'' +
            '}');
    }
}
```

```
package JavaPractice.Java00PBasic.JavaLearn;
```

```
public class Main {
```

```
    public static void main(String[] args) {
```

```
        HocSinh hocSinh = new HocSinh();
```

```
        hocSinh.NhapThongTin();
```

```
        hocSinh.NhapThongTinHocSinh();
```

```
        hocSinh.HienThiThongTin();
```

```
        hocSinh.InThongTinHocSinh();
```

```
    }
```

```
}
```

C#: Lớp HocSinh kế thừa lớp Nguoi thông qua dấu ":"

```
using System.Text;
using System.Threading.Tasks;

namespace OOP_Test
{
    2 references
    class HocSinh : Nguoi
    {
        private string _maHocSinh;
        private double _diemTB;
        private string _maLop;

        1 reference
        public void NhapThongTinHocSinh()
        {
            Console.Write("Nhap ma hoc sinh: ");
            _maHocSinh = Console.ReadLine();
            Console.Write("Nhap diem trung binh: ");
            _diemTB = double.Parse(Console.ReadLine());
            Console.Write("Nhap ma lop: ");
            _maLop = Console.ReadLine();
        }

        1 reference
        public void InThongTinHocSinh()
        {
            Console.WriteLine("HocSinh{" +
                "maHocSinh='" + _maHocSinh + '\'' +
                ", diemTB=" + _diemTB +
                ", maLop='" + _maLop + '\'' +
                '}');
        }
    }
}
```

```

using System;

namespace OOP_Test
{
    0 references
    class Program
    {
        0 references
        static void Main(string[] args)
        {
            HocSinh hocSinh = new HocSinh();
            hocSinh.NhapThongTin();
            hocSinh.NhapThongTinHocSinh();

            hocSinh.HienThiThongTin();
            hocSinh.InThongTinHocSinh();
        }
    }
}

```

Ở ví dụ trên, lớp HocSinh được kế thừa từ lớp Nguoi, trong lớp HocSinh định nghĩa thêm các thông tin khác như maHocSinh, diemTB, maLop và 2 phương thức NhapThongTinHocSinh() và InThongTinHocSinh(). Tuy nhiên ở hàm Main, các bạn có thể thấy lớp hocSinh vẫn có thể gọi được 2 phương thức NhapThongTin() và HienThiThongTin() ở lớp Nguoi đó chính là do lớp HocSinh đang được kế thừa các phương thức đó từ lớp cha là lớp Nguoi.

- **Một số lưu ý khi kế thừa:**

1. Lớp con có thể kế thừa thuộc tính hay phương thức từ lớp cha hay không còn phụ thuộc vào phạm vi truy cập của các thuộc tính hay phương thức đó. Nếu phạm vi truy cập của thuộc tính hay phương thức đó là public hoặc protected thì lớp con có thể kế thừa được, nhưng nếu là private thì không.

VD:

Trong lớp `Nguoi`, gán field `hoTen` là "Nguyen Van A", đặt phạm vi truy cập là `protected`.

Java:

```
public class Nguoi {  
  
    //PRIVATE  
    protected String hoTen = "Nguyen Van A";  
    private Integer tuoi = 30;  
    private String diaChi = "Ha Noi";  
    private String soCMT = "123456";  
    private String soDienThoai = "123456";  
}
```

C#:

```
1 reference  
class Nguoi  
{  
    #region PRIVATE  
    protected string _hoTen = "Nguyen Van A";  
    private int tuoi = 30;  
    private string diaChi = "Ha Noi";  
    private string soCMT = "123456";  
    private string soDienThoai = "123456";  
    #endregion  
}
```

Tại lớp `HocSinh`, tạo 1 hàm `DoiThongTin` thực hiện gán `hoTen` bằng tên mới là `Duong Van B`.

Java:

```
public class HocSinh extends Nguoi {  
    private String maHocSinh;  
    private double diemTB;  
    private String maLop;  
  
    public void DoiThongTin() {  
        hoTen = "Duong Van B";  
        //      tuoi = 30;  
    }  
}
```

C#:

```
2 references  
class HocSinh : Nguoi  
{  
    private string _maHocSinh;  
    private double _diemTB;  
    private string _maLop;  
  
    0 references  
    public void DoiThongTin()  
    {  
        _hoTen = "Duong Van B";  
    }  
}
```


Kết quả:

Java:

```
public class Main {  
    public static void main(String[] args) {  
        HocSinh hocSinh = new HocSinh();  
        hocSinh.DoithongTin();  
        hocSinh.HienThiThongTin();  
    }  
}
```

Main (1) x

```
"C:\Program Files\Java\jdk-11.0.9\bin\java.exe" "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA\bin\idea-agent-1.0.jar" -Dfile.encoding=UTF-8 -Djava.awt.headless=true -Djava.class.path=C:\Program Files\Java\jdk-11.0.9\bin\java.exe  
Nguoi{hoTen='Duong Van B', tuoi=30, diaChi='Ha Noi', soCMT='123456', soDienThoai='123456'}
```

C#:

```
0 references  
class Program  
{  
    0 references  
    static void Main(string[] args)  
    {  
        HocSinh hocSinh = new HocSinh();  
        hocSinh.DoithongTin();  
        hocSinh.HienThiThongTin();  
    }  
}
```

Microsoft Visual Studio Debug Console

```
Nguoi{hoTen='Duong Van B', tuoi=30, diaChi='Ha Noi', soCMT='123456', soDienThoai='123456'}  
  
D:\Work\CodeTest\CodeTest\OOP_Test\bin\Debug\net5.0\OOP_Test.exe (process 14632) exited with 0  
To automatically close the console when debugging stops, enable Tools->Options->Debugging->  
lose the console when debugging stops.  
Press any key to close this window . . .
```

Kết quả in ra hoTen sẽ là chuỗi họ tên mới là Duong Van B.

Java:

```
public class HocSinh extends Nguoi {
    private String maHocSinh;
    private double diemTB;
    private String maLop;

    public void DoiThongTin() {
        hoTen = "Duong Van B";
        tuoi = 50;
    }

    public
    Sca JavaPractice.JavaOOPBasic.JavaLearn.Nguoi
    Sys private Integer tuoi = 30
    maHocSinh = sc.nextLine();
}
```

C#:

```
1 reference
public void DoiThongTin()
{
    _hoTen = "Duong Van B";
    _tuoi = 50;
}

0 references
public v
{
    Console.WriteLine("Nhap ma hoc sinh: ");
}
```

Nếu trong hàm DoiThongTin() ta viết thêm gán `tuoi = 50` thì sẽ báo lỗi vì phạm vi truy cập không hợp lệ.

Qua ví dụ trên ta có thể thấy khi đặt phạm vi truy cập là `protected` thì class HocSinh có thể truy cập và thay đổi dữ liệu của trường hoTen ở class Nguoi được, các trường còn lại của lớp Nguoi thì không do phạm vi truy cập của chúng là `private`.

2. Một số chú ý khi kế thừa với constructor:

Java:

```
public class Nguoi {  
  
    //PRIVATE  
    protected String hoTen = "Nguyen Van A";  
    private Integer tuoi = 30;  
    private String diaChi = "Ha Noi";  
    private String soCMT = "123456";  
    private String soDienThoai = "123456";  
  
    public Nguoi() {  
    }  
  
    public Nguoi(String hoTen, Integer tuoi, String diaChi, String soCMT, String soDienThoai) {  
        this.hoTen = hoTen;  
        this.tuoi = tuoi;  
        this.diaChi = diaChi;  
        this.soCMT = soCMT;  
        this.soDienThoai = soDienThoai;  
    }  
  
    public class HocSinh extends Nguoi {  
        private String maHocSinh = "HS1";  
        private double diemTB = 5;  
        private String maLop = "12A1";  
  
        public String getMaHocSinh() {  
            return maHocSinh;  
        }  
  
        public void setMaHocSinh(String maHocSinh) {  
            this.maHocSinh = maHocSinh;  
        }  
  
        public double getDiemTB() { return diemTB; }  
  
        //Constructor không tham số  
        public HocSinh() {  
        }  
  
        //Constructor có tham số  
        public HocSinh(String hoTen, Integer tuoi, String diaChi, String soCMT, String soDienThoai, String maHocSinh,  
            double diemTB, String maLop) {  
            super(hoTen, tuoi, diaChi, soCMT, soDienThoai);  
            this.maHocSinh = maHocSinh;  
            this.diemTB = diemTB;  
            this.maLop = maLop;  
        }  
    }  
}
```

Với Java, khi thiết kế hàm khởi tạo cho lớp con là lớp HocSinh mà muốn gán thêm các thông tin của lớp cha là lớp Nguoi thì trong hàm khởi tạo của lớp con, ta cần gọi thêm đến hàm khởi tạo của lớp Nguoi (lớp cha) để gửi các thông tin sang cho lớp cha. Cụ thể là khi khởi tạo ở lớp con, ta sẽ cho phép nhập thêm các thông tin của lớp cha để khởi tạo (hoTen, tuoi, diaChi, soCMT, soDienThoai). Tuy nhiên các thông tin này lại không được phép khởi tạo ở lớp con, vậy nên ta sẽ sử dụng thêm **super** (*1 biến tham chiếu sử dụng để tham chiếu đến lớp cha gần nhất*) để gọi đến hàm khởi tạo của lớp cha và gửi dữ liệu khởi tạo sang đó `super(hoTen, tuoi, diaChi, soCMT, soDienThoai);`. Các tham số truyền vào hàm khởi tạo của lớp con hiện tại sẽ bao gồm các thông tin của lớp con và lớp cha, với các thông tin của lớp con ta sẽ gán trực tiếp như viết hàm khởi tạo bình thường, còn các thông tin của lớp cha sẽ được gửi đi khởi tạo thông qua biến tham chiếu super.

C#:

```
0 references
public Nguoi()
{
}

0 references
public Nguoi(string hoTen, int tuoi, string diaChi, string soCMT, string soDienThoai)
{
    HoTen = hoTen;
    Tuoi = tuoi;
    DiaChi = diaChi;
    SoCMT = soCMT;
    SoDienThoai = soDienThoai;
}
```

```

//Constructor không tham số
1 reference
public HocSinh()
{
}

//Constructor có tham số
0 references
public HocSinh(string MaHocSinh, double DiemTB, string MaLop, string hoTen, int tuoi, string diaChi, string soCMT,
    string soDienThoai) : base(hoTen, tuoi, diaChi, soCMT, soDienThoai)
{
    this.MaHocSinh = MaHocSinh;
    this.DiemTB = DiemTB;
    this.MaLop = MaLop;
}

```

Với C#, khi thiết kế hàm khởi tạo cho lớp con là lớp HocSinh mà muốn gán thêm các thông tin của lớp cha là lớp Nguoi thì trong hàm khởi tạo của lớp con, ta cần gọi thêm đến hàm khởi tạo của lớp Nguoi (lớp cha) để gửi các thông tin sang cho lớp cha. Cụ thể là khi khởi tạo ở lớp con, ta sẽ cho phép nhập thêm các thông tin của lớp cha để khởi tạo (hoTen, tuoi, diaChi, soCMT, soDienThoai). Tuy nhiên các thông tin này lại không được phép khởi tạo ở lớp con, vậy nên ta sẽ sử dụng thêm **base** để tiến hành tham chiếu đến hàm khởi tạo của lớp cha `base(hoTen, tuoi, diaChi, soCMT, soDienThoai)`. Định danh **base** trong ngôn ngữ lập trình C# là một từ khoá trong ngôn ngữ lập trình C# cho phép người sử dụng truy cập đến các thông tin của lớp cha từ lớp con. Các tham số truyền vào hàm khởi tạo của lớp con hiện tại sẽ bao gồm các thông tin của lớp con và lớp cha, với các thông tin của lớp con ta sẽ gán trực tiếp như viết hàm khởi tạo bình thường, còn các thông tin của lớp cha sẽ được gửi qua lớp cha khởi tạo thông qua biến tham chiếu **base**.

c. Tính đa hình (Polymorphism)

Tính đa hình là một hành động có thể được thực hiện bằng nhiều cách khác nhau. Hiểu một cách đơn giản: Đa hình là khái niệm mà hai hay nhiều lớp có những phương thức giống nhau nhưng có thể thực thi theo những cách thức khác nhau.

Một ví dụ thực tế về tính đa hình: Ta có 2 con vật là chó và mèo, cả chó và mèo đều thuộc lớp động vật và đều có thể phát ra tiếng kêu. Tuy nhiên với chó thì tiếng kêu sẽ là “gâu gâu” còn với mèo lại là “meo meo”.

Thông qua ví dụ trên, các bạn có thể thấy điểm chung của chó, mèo đều thuộc 1 lớp cha là lớp động vật, cùng có hành động (phương thức) kêu, tuy nhiên tiếng kêu của chó và mèo là khác nhau. Đó chính là tính đa hình.

Ví dụ:

Java:

Ở class `Nguyen`, tạo thêm 1 phương thức `Di()` thực hiện in ra “Di bo”

```
public class Nguyen {  
  
    //PRIVATE  
    protected String hoTen = "Nguyen Van A";  
    private Integer tuoi = 30;  
    private String diaChi = "Ha Noi";  
    private String soCMT = "123456";  
    private String soDienThoai = "123456";  
  
    public void Di(){  
        System.out.println("Di bo");  
    }  
}
```

Class `HocSinh` kế thừa class `Nguyen` cũng tạo thêm 1 phương thức `Di()` thực hiện in ra “Di hoc”

```

public class HocSinh extends Nguoi {
    private String maHocSinh = "HS1";
    private double diemTB = 5;
    private String maLop = "12A1";

    public void Di(){
        System.out.println("Di hoc");
    }
}

```

Class NhanVien kế thừa class Nguoi, tạo phương thức Di() thực hiện in ra "Di lam"

```

public class NhanVien extends Nguoi {
    public void Di() {
        System.out.println("Di lam");
    }
}

```

Kết quả sau khi thực hiện ở hàm main:

```

public class Main {
    public static void main(String[] args) {
        Nguoi nguoi = new Nguoi();
        Nguoi hocSinh = new HocSinh();
        Nguoi nhanVien = new NhanVien();

        System.out.println("Phương thức Di của Nguoi: ");
        nguoi.Di();
        System.out.println("Phương thức Di của HocSinh: ");
        hocSinh.Di();
        System.out.println("Phương thức Di của NhanVien: ");
        nhanVien.Di();
    }
}

```

Main (1) ×

```

"C:\Program Files\Java\jdk-11.0.9\bin\java.exe" "-javaagent:C:
Phương thức Di của Nguoi:
Di bo
Phương thức Di của HocSinh:
Di hoc
Phương thức Di của NhanVien:
Di lam

```

Ở hàm main mình thực hiện khởi tạo đối tượng có kiểu là `Nguoi` dưới 3 dạng là `Nguoi`, `HocSinh` và `NhanVien`. Tương ứng ta thực hiện gọi phương thức `Di()` của 3 đối tượng được khởi tạo ở trên, kết quả in ra sẽ ra 3 hình thức đi của từng đối tượng: `nguoi` khởi tạo dưới dạng `Nguoi()` sẽ hiển thị ra "Di bo", khởi tạo dưới dạng `HocSinh()` sẽ hiển thị "Di hoc" và khởi tạo dưới dạng `NhanVien()` sẽ hiển thị ra "Di lam".

Ứng với ví dụ về chó và mèo phía trên, các bạn có thể thấy `nguoi`, `hocSinh`, `nhanVien` đều thuộc lớp `Nguoi`, tuy nhiên phương thức `Di()` của từng đối tượng sẽ là khác nhau. Đó là thể hiện của tính đa hình.

NOTE: Với trường hợp trong phương thức `Di()` của lớp con muốn gọi đến phương thức `Di()` của lớp cha, ta chỉ cần dùng `super` để gọi đến phương thức `Di()` của lớp cha.

```
public class NhanVien extends Nguoi {  
    public void Di() {  
        super.Di();  
        System.out.println("Di lam");  
    }  
}
```

Kết quả khi chạy chương trình: phương thức `Di()` của `nhanVien` có hiển thị thêm thông tin `Di()` của lớp cha là `Nguoi`.

```
public class Main {  
    public static void main(String[] args) {  
        Nguoi nhanVien = new NhanVien();  
  
        System.out.println("Phuong thuc Di cua NhanVien: ");  
        nhanVien.Di();  
    }  
}
```

Main (1) x

"C:\Program Files\Java\jdk-11.0.9\bin\java.exe" "-javaagent:C:
Phuong thuc Di cua NhanVien:
Di bo
Di lam

C#:

Ở class `Nguoi`, tạo thêm 1 phương thức ảo (virtual) `Di()` thực hiện in ra "Đi bo"

```
9 references
class Nguoi
{
    #region PRIVATE
    private string hoTen = "Nguyen Van A";
    private int _tuoi = 30;
    private string _diaChi = "Ha Noi";
    private string _soCMT = "123456";
    private string _soDienThoai = "123456";
    #endregion

    5 references
    public virtual void Di()
    {
        Console.WriteLine("Đi bo");
    }
}
```

Ở class `HocSinh`, tạo thêm phương thức ghi đè (override) phương thức `Di()` của lớp cha

```
3 references
class HocSinh : Nguoi
{
    private string _maHocSinh = "HS1";
    private double _diemTB = 5;
    private string _maLop = "12A1";

    4 references
    public override void Di()
    {
        Console.WriteLine("Đi hoc");
    }
}
```

Ở class NhanVien, tạo phương thức ghi đè (override) phương thức Di() của lớp cha

```
1 reference
class NhanVien : Nguoi
{
    4 references
    public override void Di()
    {
        Console.WriteLine("Di lam");
    }
}
```

Kết quả thực hiện ở hàm main:

```
0 references
static void Main(string[] args)
{
    Nguoi nguoi = new Nguoi();
    Nguoi hocSinh = new HocSinh();
    Nguoi nhanVien = new NhanVien();

    Console.WriteLine("Phuong thuc Di cua Nguoi: ");
    nguoi.Di();
    Console.WriteLine("Phuong thuc Di cua HocSinh: ");
    hocSinh.Di();
    Console.WriteLine("Phuong thuc Di cua NhanVien: ");
    nhanVien.Di();
}
```

Microsoft Visual Studio Debug Console

```
Phuong thuc Di cua Nguoi:
Di bo
Phuong thuc Di cua HocSinh:
Di hoc
Phuong thuc Di cua NhanVien:
Di lam
```

Ở hàm main mình thực hiện khởi tạo đối tượng có kiểu là Nguoi dưới 3 dạng là Nguoi, HocSinh và NhanVien. Tương ứng ta thực hiện gọi phương thức Di() của 3 đối tượng được khởi tạo ở trên, kết quả in ra sẽ ra 3 hình thức đi của từng đối tượng: nguoi khởi tạo dưới dạng Nguoi() sẽ hiển thị ra "Di bo", khởi tạo dưới dạng HocSinh() sẽ hiển thị "Di hoc" và khởi tạo dưới dạng NhanVien() sẽ hiển thị ra "Di lam".

Ứng với ví dụ về chó và mèo phía trên, các bạn có thể thấy `Nguoi`, `hocSinh`, `nhanVien` đều thuộc lớp `Nguoi`, tuy nhiên phương thức `Di()` của từng đối tượng sẽ là khác nhau. Đó là thể hiện của tính đa hình.

NOTE:

+ Để phân biệt rõ được phương thức `Di()` nào là của lớp cha, phương thức `Di()` nào của lớp con, các bạn cần dùng thêm các từ khoá **virtual** cho phương thức của lớp cha và **override** với phương thức ghi đè ở lớp con. Nhờ vậy mà khi khởi tạo chương trình sẽ nhận diện được phương thức nào của lớp nào. Nếu không có thì mặc định sẽ lấy phương thức của lớp cha (vì kiểu dữ liệu của đối tượng đang là lớp cha).

+ Với trường hợp trong phương thức `Di()` của lớp con muốn gọi đến phương thức `Di()` của lớp cha, ta chỉ cần dùng `base` để gọi đến phương thức `Di()` của lớp cha.

```
1 reference
class NhanVien : Nguoi
{
    3 references
    public override void Di()
    {
        base.Di();
        Console.WriteLine("Di lam");
    }
}
```

Kết quả khi chạy chương trình: phương thức `Di()` của `nhanVien` có hiển thị thêm thông tin `Di()` của lớp cha là `Nguoi`.

```
0 references
static void Main(string[] args)
{
    Nguoi nhanVien = new NhanVien();

    Console.WriteLine("Phuong thuc Di cua NhanVien: ");
    nhanVien.Di();
}
```

Microsoft Visual Studio Debug Console

Phuong thuc Di cua NhanVien:
Di bo
Di lam

d. Tính trừu tượng (Abstraction)

Trừu tượng có nghĩa là tổng quát hoá một cái gì đó lên, không cần chú ý chi tiết bên trong là gì, tuy nhiên người ta vẫn có thể hiểu mỗi khi nghe về nó.

Ví dụ: Bạn chạy xe ga có hành động tăng ga để xe tăng tốc, thì chức năng tăng ga là đại diện cho sự trừu tượng. Người dùng chỉ biết là tăng ga thì xe tăng tốc, không cần biết bên trong nó làm thế nào, qua những công đoạn nào thì mới tăng tốc được.

Trong OOP, tính trừu tượng là định nghĩa tại class cha các thuộc tính, phương thức cần cho việc giải quyết vấn đề mà mọi class con phải tuân theo. Từ khoá nhận diện tính trừu tượng là thông qua **Interface** hoặc **Abstract**.

1. Abstract:

Java:

Abstract class (Lớp trừu tượng)

```
public abstract class Nguoi {  
  
    //PRIVATE  
    protected String hoTen = "Nguyen Van A";  
    private Integer tuoi = 30;  
    private String diaChi = "Ha Noi";  
    private String soCMT = "123456";  
    private String soDienThoai = "123456";  
  
    public void Di() { System.out.println("Di bo"); }  
  
    public Nguoi() {}  
}
```

Thêm từ khoá abstract vào class Nguoi để biến lớp Nguoi thành lớp cơ sở.

Khi đặt abstract vào lớp, ta sẽ không thể khởi tạo đối tượng từ lớp đó, thay vào đó ta sẽ phải khởi tạo thông qua lớp con kế thừa từ lớp abstract.

```
Ngnoi nguoi = new Ngnoi();
```

'Ngnoi' is abstract; cannot be instantiated

Make 'Ngnoi' not abstract Alt+Shift+Enter

More actions... Alt+Enter

```
JavaPractice.Java00PBasic.JavaLearn.Ngnoi  
@Contract(pure = true)  
public Ngnoi()
```

Inferred annotations: @org.jetbrains.annotations.Contract(pure = true)

Khi khởi tạo bằng lớp Ngnoi sẽ báo lỗi Ngnoi hiện tại là lớp cơ sở, không thể khởi tạo.

```
public static void main(String[] args) {  
    Ngnoi nguoi = new Ngnoi();  
    Ngnoi nguoi1 = new HocSinh();  
}
```

Khi khởi tạo bằng lớp HocSinh là con của lớp Ngnoi sẽ không xảy ra lỗi.

Abstract Method (Phương thức trừu tượng)

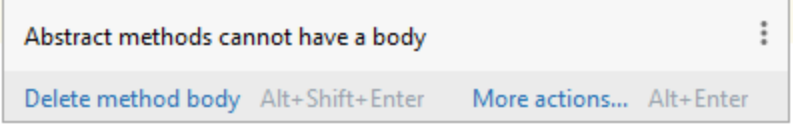
Ở lớp Ngnoi, ta thực hiện chuyển phương thức Di() thành phương thức trừu tượng (abstract method)

```
public abstract class Ngnoi {  
  
    //PRIVATE  
    protected String hoTen = "Nguyen Van A";  
    private Integer tuoi = 30;  
    private String diaChi = "Ha Noi";  
    private String soCMT = "123456";  
    private String soDienThoai = "123456";  
  
    public abstract void Di();  
}
```

Một phương thức được chuyển thành phương thức trừu tượng thông qua từ khoá abstract sẽ không có thân hàm, chỉ có phần khai báo bao gồm:

<phạm vi truy cập> abstract <kiểu dữ liệu> <tên hàm>(<tham số nếu có>);

```
public abstract void Di();  
public abstract void TestMethod(){  
    |  
}
```



2 related problems

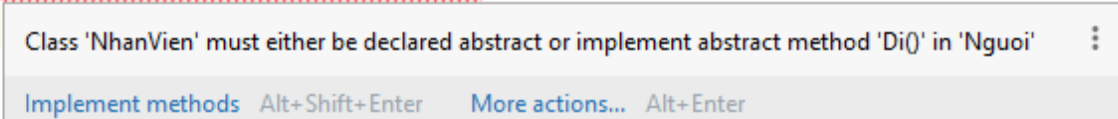
Nếu phương thức abstract có thân hàm sẽ báo lỗi như hình.

Với phương thức Di() tại lớp Nguoi được chuyển thành phương thức trừu tượng, lúc này mọi lớp con kế thừa lớp Nguoi sẽ **PHẢI** định nghĩa phương thức Di() đó:

```
public class HocSinh extends Nguoi {  
    private String maHocSinh = "HS1";  
    private double diemTB = 5;  
    private String maLop = "12A1";  
  
    public void Di(){  
        System.out.println("Di hoc");  
    }  
}
```

Lớp HocSinh kế thừa lớp Nguoi có định nghĩa phương thức Di() nên không xảy ra lỗi gì.

```
public class NhanVien extends Nguoi {  
    |  
}
```



Lớp NhanVien kế thừa lớp Nguoi nhưng không định nghĩa phương thức Di() nên chương trình sẽ báo lỗi như hình.

C#:

Abstract class (Lớp trừu tượng)

```
6 references
abstract class Nguoi
{
    #region PRIVATE
    private string hoTen = "Nguyen Van A";
    private int _tuoi = 30;
    private string _diaChi = "Ha Noi";
    private string _soCMT = "123456";
    private string _soDienThoai = "123456";
    #endregion

    4 references
    public virtual void Di()
    {
        Console.WriteLine("Đi bo");
    }
}
```

Thêm từ khoá abstract vào class Nguoi để biến lớp Nguoi thành lớp cơ sở.

Khi đặt abstract vào lớp, ta sẽ không thể khởi tạo đối tượng từ lớp đó, thay vào đó ta sẽ phải khởi tạo thông qua lớp con kế thừa từ lớp abstract.

```
Nguoi nguoi = new Nguoi();
```

Nguyen.Nguoi() (+ 1 overload)

CS0144: Cannot create an instance of the abstract type or interface 'Nguoi'

Show potential fixes (Alt+Enter or Ctrl+.)

Khi khởi tạo bằng lớp Nguoi sẽ báo lỗi Nguoi hiện tại là lớp cơ sở, không thể khởi tạo.

```

0 references
static void Main(string[] args)
{
    Ngươi nguoi = new Ngươi();
    Ngươi nguoi1 = new HocSinh();
}

```

Khi khởi tạo bằng lớp HocSinh là con của lớp Ngươi sẽ không xảy ra lỗi.

Abstract Method (Phương thức trừu tượng)

Ở lớp Ngươi, ta thực hiện chuyển phương thức Di() thành phương thức trừu tượng (abstract method)

```

abstract class Ngươi
{
    #region PRIVATE
    private string hoTen = "Nguyen Van A";
    private int _tuoi = 30;
    private string _diaChi = "Ha Noi";
    private string _soCMT = "123456";
    private string _soDienThoai = "123456";
    #endregion

    3 references
    public abstract void Di();
}

```

Một phương thức được chuyển thành phương thức trừu tượng thông qua từ khoá abstract sẽ không có thân hàm, chỉ có phần khai báo bao gồm:

<phạm vi truy cập> abstract <kiểu dữ liệu> <tên hàm>(<tham số nếu có>);


```
3 references
public abstract void Di();
0 references
public abstract void TestMethod()
{
}

void Nguoi.TestMethod()
CS0500: 'Nguoi.TestMethod()' cannot declare a body because it is marked abstract
```

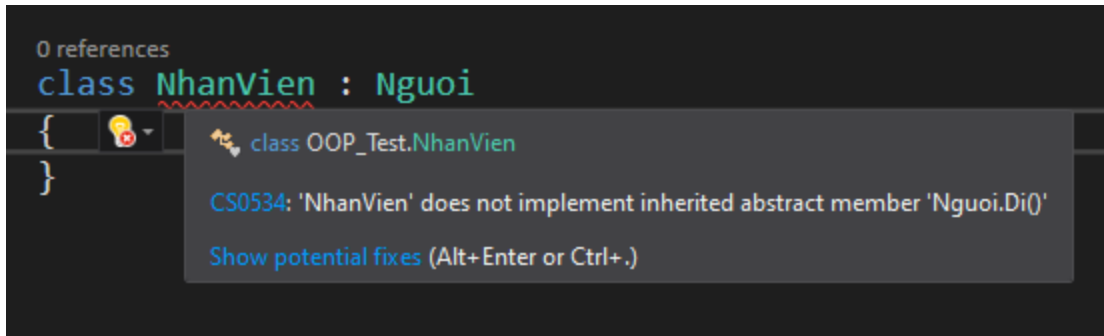
Nếu phương thức abstract có thân hàm sẽ báo lỗi như hình.

Với phương thức Di() tại lớp Nguoi được chuyển thành phương thức trừu tượng, lúc này mọi lớp con kế thừa lớp Nguoi sẽ **PHẢI** định nghĩa phương thức Di() đó:

```
class HocSinh : Nguoi
{
    private string _maHocSinh = "HS1";
    private double _diemTB = 5;
    private string _maLop = "12A1";

    1 reference
    public override void Di()
    {
        Console.WriteLine("Đi học");
    }
}
```

Lớp HocSinh kế thừa lớp Nguoi có định nghĩa phương thức Di() nên không xảy ra lỗi gì.



Lớp NhanVien kế thừa lớp Ngươi nhưng không định nghĩa phương thức Di() nên chương trình sẽ báo lỗi như hình.

Một lớp là lớp cơ sở sẽ chỉ cho phép các lớp khác kế thừa các thông tin từ nó, hiểu đơn giản nó sẽ là khuôn mẫu, là lớp chung cho các lớp khác. VD lớp DongVat sẽ là lớp chung, mang các thông tin là khuôn mẫu cho các lớp như Cho, Meo, Ga, ... là các lớp con kế thừa những đặc tính chung của DongVat.

2. Interface:

Một interface được hiểu như một khuôn mẫu mà mọi lớp thực thi nó đều phải tuân theo. Interface sẽ định nghĩa phần **"làm gì"** (khai báo) và lớp thực thi interface này (implement) sẽ định nghĩa chi tiết phần **"làm thế nào"** (nội dung).

Đặc điểm của interface:

- Chỉ chứa khai báo, không chứa định nghĩa (giống như phương thức abstract). Tuy nhiên phương thức trong interface khi khai báo sẽ không cần từ khoá abstract.
- Việc ghi đè 1 thành phần trong interface cũng không cần từ khoá override.
- Không cần khai báo phạm vi truy cập cho các thành phần bên trong interface. Các thành phần này sẽ mặc định là public bởi interface là

khuôn mẫu cho lớp khác thực thi công việc chứ nó không tự thực thi nên phải để là public.

- Interface không chứa các thuộc tính (property) hoặc các biến.
- Interface không có constructor và destructor.
- Các lớp có thể thực thi (implement) cùng lúc nhiều interface.
- Một interface có thể kế thừa nhiều interface khác nhưng không thể kế thừa class.

Ví dụ về Interface:

Java:

Interface

```
public interface IHocSinhService {  
    void ThemHocSinh(HocSinh hocSinh);  
  
    void SuaHocSinh(HocSinh hocSinh);  
  
    void XoaHocSinh(int hocSinhID);  
  
    void ChuyenLop(int hocSinhID, int lopID);  
  
    List<HocSinh> LayDanhSachHocSinh();  
}
```

Lớp HocSinhService thực thi interface IhocSinhService

```

public class HocSinhService implements IHocSinhService {
    @Autowired
    HocSinhRepository hsRep;
    @Autowired
    LopRepository lopRep;

    public boolean KiemTra(int lopID) {...}

    public void CapNhatSiSo(int lopID) {...}

    @Override
    public void ThemHocSinh(HocSinh hocSinh) {...}

    @Override
    public void SuaHocSinh(HocSinh hocSinh) {...}

    @Override
    public void XoaHocSinh(int hocSinhID) {...}

    @Override
    public void ChuyenLop(int hocSinhID, int lopID) {}

    @Override
    public List<HocSinh> LayDanhSachHocSinh() {...}
}

```

Lớp HocSinhService thực thi (implement) IHocSinhService sẽ phải mang đầy đủ, chính xác thông tin các hàm trong IHocSinhService, và thực thi phần thân hàm cho các phần định nghĩa tại IHocSinhService.

```

public class HocSinhService implements IHocSinhService {
    @Autowired
    HocSinhRepository hsRep;
    @Autowired
    LopRepository lopRep;

    public boolean KiemTra(int lopID) {...}

    public void CapNhatSiSo(int lopID) {...}

    @Override
    public void ThemHocSinh(HocSinh hocSinh) {
        hsRep.save(hocSinh);
        CapNhatSiSo(hocSinh.getLop().getId());
    }

    @Override
    public void SuaHocSinh(HocSinh hocSinh) {
        // TODO Auto-generated method stub
        if (KiemTra(hocSinh.getLop().getId())) {
            HocSinh currentHS = hsRep.findById(hocSinh.getId()).get();
            int oldClass = currentHS.getLop().getId();
            currentHS = hocSinh;
            hsRep.save(currentHS);
            CapNhatSiSo(oldClass);
            CapNhatSiSo(hocSinh.getLop().getId());
        }
    }
}

```

C#:

Interface

```
1 reference
interface IHocSinhService
{
    2 references
    IEnumerable<HocSinh> GetStudentList();
    2 references
    HocSinh GetStudentListById(int hocSinhId);
    2 references
    HocSinh AddNewStudent(HocSinh hocSinh);
    2 references
    HocSinh UpdateStudent(HocSinh hocSinh);
    2 references
    HocSinh DeleteStudent(int hocSinhId);
}
```

Lớp HocSinhService thực thi interface IHocSinhService

```
3 references
public class HocSinhService : IHocSinhService
{
    15 references
    private QLHocSinhDbContext dbContext { get; }
    1 reference
    public HocSinhService()...
    1 reference
    public void CapNhatSiSoChoLop(int lopId)...
    3 references
    public void CapNhatSiSo()...
    2 references
    public HocSinh AddNewStudent(HocSinh hocSinh)...
    2 references
    public HocSinh DeleteStudent(int hocSinhId)...

    2 references
    public IEnumerable<HocSinh> GetStudentList()...

    2 references
    public HocSinh GetStudentListById(int hocSinhId)...

    2 references
    public HocSinh UpdateStudent(HocSinh hocSinh)...
}
```

Lớp HocSinhService thực thi (implement) IHocSinhService sẽ phải mang đầy đủ, chính xác thông tin các hàm trong IHocSinhService, và thực thi phần thân hàm cho các phần định nghĩa tại IHocSinhService.

Ví dụ:

```
3 references
public class HocSinhService : IHocSinhService
{
    15 references
    private QLHocSinhDbContext dbContext { get; }
    1 reference
    public HocSinhService(...)
    1 reference
    public void CapNhatSiSoChoLop(int lopId) ...
    3 references
    public void CapNhatSiSo(...)
    2 references
    public HocSinh AddNewStudent(HocSinh hocSinh)
    {
        dbContext.HocSinhs.Add(hocSinh);
        dbContext.SaveChanges();
        CapNhatSiSo();
        return hocSinh;
    }
    2 references
    public HocSinh DeleteStudent(int hocSinhId)
    {
        var currentHocSinh = dbContext.HocSinhs.Find(hocSinhId);
        if (currentHocSinh != null)
        {
            dbContext.HocSinhs.Remove(currentHocSinh);
            dbContext.SaveChanges();
            CapNhatSiSo();
        }
        return currentHocSinh;
    }
}
```

Link tham khảo thêm về tính trừu tượng:

[Tính trừu tượng trong lập trình hướng đối tượng với Java | How Kteam](#)

[Interface trong Lập trình hướng đối tượng | How Kteam](#)

[Java Core - Tính trừu tượng \(Abstract\) - YouTube](#)