## Lớp trong java

Lớp là một tập hợp các đối tượng có những thuộc tính và phương thức giống nhau. Ta ví dụ về lớp car sẽ có các đối tượng moto, bike …. Chúng đều có các thuộc tính như speed, limit speed. Vậy đối tượng là gì ?

### **Đối tượng trong java**

Một thực thể có trạng thái và hành vi thì được gọi là một đối tượng. Ví dụ xe máy có trạng thái chạy, nghỉ, có các hành vi như bật đèn, tắt đèn…

Một đối tượng có có ba đặc điểm chính:

* Trạng thái: Là các thuộc tính của đối tượng.(trường)
* Hành vi: Là các phương thức của đối tượng.
* Danh tính: Là tên của đối tượng đó.

#### **Các cách để tạo một đối tượng trong java**

* Dùng từ khóa new . Cái này chắc các bạn cũng dùng nhiều rồi.
* Sử dụng phương thức newInstance()
* Sử dụng phương thức clone()
* Sử dụng phương thức factory

Chúng ta sẽ dùng cách tạo từ khóa new để tạo một đối tượng, còn những cách khác ta sẽ học ở những bài sau.

Ví dụ:

|  |  |
| --- | --- |
| 0  1  2  3  4  5  6  7  8  9  10  11  12  13 | class Car{      int a = 10;  }  class Oto extends Car {      int a = 20;  }  class Test{      public static void main(String[] args){          Oto A = new Oto();          System.out.println(A.a);      }  } |

##### **Sự khác nhau giữa đối tượng và lớp trong java**

|  |  |  |
| --- | --- | --- |
| **STT** | **Đối tượng** | **Lớp** |
| 1 | Là thể hiện của một lớp. | Là một khuôn mẫu để tạo ra các đối tượng. |
| 2 | Đối tượng là một thực thể có thật và có các đặc điểm riêng biệt. | Lớp là một tập hợp các đối tượng có những tính chất giống nhau. |
| 3 | Là một thực thể vật lý. | Là một thực thể logic. |
| 4 | Thường được tạo ra bởi từ khóa new. | Được khai báo bằng từ khóa class. |
| 5 | Đối tượng được tạo ra nhiều lần. | Lớp chỉ được khai báo một lần. |
| 6 | Đối tượng sẽ được cấp bộ nhớ khi nó được tạo ra. | Không được cấp bộ nhớ. |
| 7 | Có nhiều cách để tạo ra một đối tượng. | Chỉ có một cách khai báo đó là dùng từ khóa class. |

## Tính đóng gói trong java

**Tính đóng gói trong java** là kỹ thuật ẩn giấu thông tin và hiển thị ra thông tin liên quan. Mục đích chính của đóng gói trong java là giảm thiểu mức độ phức tạp phát triển phần mềm.

Đóng gói cũng được sử dụng để bảo vệ trạng thái bên trong của một đối tượng. Thông qua các phương thức set,get ta có thể thay đổi các giá trị thuộc tính và lấy giá trị từ chúng. Điều này làm cho chương trình dễ quản lý hơn và có thể kiểm soát dữ liệu tốt hơn.

Ví dụ về tính đóng gói

}

|  |  |
| --- | --- |
| 0  1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18 | public class HinhChuNhat {      private int rong; // biến intance      private int dai; // biến intance        public void setRong(int rong){          this.rong = rong;      }      public void setDai(int dai){          this.dai = dai;      }      public int getRong(){          return this.rong;      }      public int getDai(){          return this.dai;      }  } |

Mặc dù các bạn không thể trực tiếp thay đổi giá trị của hai biến rong và dai nhưng có thể thông qua phương thức setRong , setDai để thay đổi giá trị của chúng. Hoặc thông qua hai phương thức getRong và getDai để lấy ra giá trị của chúng.

|  |
| --- |
| public class DemoJava {        public static void main(String[] args) {          HinhChuNhat HCN = new HinhChuNhat();          HCN.setRong(3);          HCN.setDai(4);          System.out.println("HCN co chieu rong: " + HCN.getRong() );          System.out.println("HCN co chieu dai: " + HCN.getDai() );      }  }; |

Sau khi chạy sẽ có kết quả

|  |  |
| --- | --- |
| 0  1  2  3 | HCN co chieu rong: 3  HCN co chieu dai: 4 |

### **Phạm vi truy cập**

Trong java cung cấp thành phần gọi là phạm vi truy cập của một lớp, một phương thức hay một thuộc tính. Gồm có 4 loại:

* **default (mặc định)**: Nếu như bạn không khai báo phạm vi truy cập thì Java sẽ hiểu mặc định phạm vi truy cập là default. Với default thì trong lớp (class) đó và trong gói (package) đó mới có thể nhìn thấy được.
* **private (riêng tư)**: Nếu như bạn khai báo phạm vi truy cập là private thì chỉ duy nhất trong lớp (class) đó mới có thể nhìn thấy được.
* **public (công khai)**: Nếu như bạn khai báo phạm vi truy cập là public thì mọi thứ từ lớp (class), gói (package), lớp con (subclass) đề có thể nhìn thấy được.
* **protected (được bảo vệ)**: Nếu như bạn khai báo phạm vi truy cập là protected thì lớp (class) đó, gói (package) đó, lớp con (subclass) đó đều có thể nhìn thấy được.

Bài tập: Xây dựng một lớp HocSinh gồm các thuộc tính tên, tuổi, địa chỉ. Các phương thức set get tên,tuổi, địa chỉ của lớp HocSinh.

## Tính kế thừa trong java

* **Kế thừa trong java** là sự liên quan giữa hai class với nhau, trong đó có class cha (superclass) và class con (subclass). Khi kế thừa class con được hưởng tất cả các phương thức và thuộc tính của class cha. Tuy nhiên, nó chỉ được truy cập các thành viên public và protected của class cha. Nó không được phép truy cập đến thành viên private của class cha.
* Khi kế thừa từ một lớp đang tồn tại bạn có sử dụng lại các phương thức và thuộc tính của lớp cha, đồng thời có thể khai báo thêm các phương thức và thuộc tính khác.

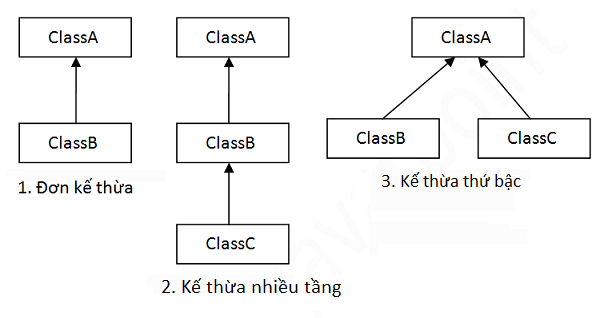
### **Cú pháp kế thừa**

|  |  |
| --- | --- |
| 0  1  2  3  4 | class Subclass-name extends Superclass-name {     //methods and fields  } |

Ví dụ về kế thừa trong java

|  |  |
| --- | --- |
| 0  1  2  3  4  5  6  7  8  9  10 | class Point{      private int x;      private int y;    }  class Circle extends Point{      private int r;    } |

#### **Trong java có support các kiểu kế thừa sau**

Các kiểu kế thừa trong java

Ví dụ về đơn thừa kế

|  |  |  |
| --- | --- | --- |
| 0  1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18 | | class A{      public void hello(){          System.out.println("Hello!");      }  }  class C extends A{      public void helloJava(){          System.out.println("Hello Java!");      }  }  class Test{      public static void main(String[] args){          C obj = new C();          obj.hello();          obj.helloJava();      }  } |
| 0  1  2  3 | Hello!  Hello Java! | | |

Ví dụ về thừa kế nhiều tầng

|  |  |  |
| --- | --- | --- |
| 0  1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24 | | class A{      public void hello(){          System.out.println("Hello!");      }  }  class B extends A{      public void hellWorld(){          System.out.println("Hello World!");      }  }  class C extends B{      public void helloJava(){          System.out.println("Hello Java!");      }  }  class Test{      public static void main(String[] args){          C obj = new C();          obj.hello();          obj.hellWorld();          obj.helloJava();      }  } |
| 0  1  2  3  4 | Hello!  Hello World!  Hello Java! | |

Ví dụ về thừa kế thứ bậc

|  |  |  |
| --- | --- | --- |
| 0  1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24 | | class A{      public void hello(){          System.out.println("Hello!");      }  }  class B extends A{      public void hellWorld(){          System.out.println("Hello World!");      }  }  class C extends A{      public void helloJava(){          System.out.println("Hello Java!");      }  }  class Test{      public static void main(String[] args){          C obj = new C();          obj.hello();          obj.helloJava();          //obj.helloWorld();      }  } |
| 0  1  2  3 | Hello!  Hello Java! | |

Để giảm thiểu sự phức tạp và đơn giản hóa ngôn ngữ, đa kế thừa không được support trong java.

Hãy suy xét kịch bản sau: Có 3 lớp A, B, C. Trong đó lớp C kế thừa từ các lớp A và B. Nếu các lớp A và B có phương thức giống nhau và bạn gọi nó từ đối tượng của lớp con, như vậy khó có thể xác đinh được việc gọi phương thức của lớp A hay B.

Vì vậy lỗi khi biên dịch sẽ tốt hơn lỗi khi runtime, java sẽ print ra lỗi “compile time error” nếu bạn cố tình kế thừa 2 class.

|  |  |  |
| --- | --- | --- |
| 0  1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17 | | class A{      public void eat(){          System.out.println("Eating");      }  }  class B{      public void eat(){          System.out.println("Not eat");      }  }  class C extends A,B{      public static void main(String[] args){          C obj = new C();          obj.eat();      }  } |
| 0  1  2 | Compilation failed; | |

Bài tập:

1. Viết một class point có các thuộc tính vị trí (x,y) và các phương thức set,get để làm việc với hai thuộc tính đó.
2. Viết một class circle kế thừa từ class point. Có thêm thuộc tính r (bán kính) và các phương thức tính chu vi, diện tích của hình tròn đó.

Tính trừu tượng trong java

Tính trừu tượng là một tiến trình ẩn các cài đặt chi tiết và chỉ hiển thị tính năng tới người dùng. Sử dụng tính trừu tượng giúp chúng ta chỉ tập trung vào trọng tâm thay vì quan tâm đến cách nó thực hiện.

Có hai cách để sử dụng tính trừu tượng:

* Sử dụng lớp abstract
* Sử dụng interface

### **Lớp trừu tượng trong java**

Một lớp abstract trong java thì phải có ít nhất một phương thức abstract ( phương thức ảo ). Phương thức abstract là phương thức được khai báo với từ khóa abstract và không có tính triển khai.

|  |  |
| --- | --- |
| 0  1  2  3  4 | abstract class Car{      abstract void display();  } |

Ở ví dụ trên phương thức display() là một phương thức abstract.

Lưu ý:

* Phương thức abstract thì không có dấu {} ở phía sau. Nếu có dấu {} thì chương trình tự hiểu phương thức đó là một phương thức trống chứ không phải không có trình triển khai.
* Một constructor không thể khai báo abstract.
* Một phương thức abstract không thể khai báo với từ khóa final.

#### **Cách kế thừa một lớp abstract**

Một lớp kế thừa lớp abstract nếu không muốn là lớp abstract thì phải overriding (ghi đè) tất cả các phương thức abstract của lớp cha.

Các bạn xem ví dụ sau.

|  |  |  |
| --- | --- | --- |
| 0  1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30 | | abstract class HinhHoc{      abstract float dientich();      abstract float chuvi();  }  class HinhTron extends HinhHoc{      float r;      HinhTron(float r){          this.r = r;      }      @Override      float dientich() {          return 3.14f\*r\*r;      }        @Override      float chuvi() {          return 2\*r\*3.14f;      }      public void display(){          System.out.println("Chu vi: "+ chuvi());          System.out.println("Dien tich: "+ dientich());      }  }  public class TEST {      public static void main(String[] args){          HinhTron hinhTron = new HinhTron(2);          hinhTron.display();      }  }    Out put: |
| 0  1  2  3 | Chu vi: 12.56  Dien tich: 12.56 | | |

Mình tạo ra một lớp abstract HinhHoc và một lớp HinhTron kế thừa từ lớp HinhHoc. Trong lớp HinhTron mình tiến hành overriding lại các phương thức abstract của lớp HinhHoc.

Một interface trong java có tất cả các phương thức đều là abstract hết. Các class triển khai từ một interface.

## Cách dùng interface trong java

Các bạn xem ví dụ dưới đây

|  |  |  |
| --- | --- | --- |
| 0  1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16 | | interface animal{       void eating();  }  class Dog implements animal{      @Override      public void eating() {          System.out.println("Dog eating");      }  }  public class TEST {      public static void main(String[] args){          Dog dog = new Dog();          dog.eating();      }  } |
| 0  1  2 | Dog eating | |

Trong java trình biên dịch tự động thêm từ khóa public và abstract trước phương thức của interface và các từ khóa public, static và final trước các thành viên dữ liệu.

Ta có các quy tắc như sau:

* Một interface không thể có constructor hay bạn không thể khởi tạo một interface.
* Tất cả các phương thức của interface đều là abstract.
* Một interface có thể được kế thừa từ một interface khác.
* Một interface không thể được kế thừa từ một lớp.
* Một class chỉ được kế thừa từ một class khác, nhưng một class có thể được triển khai từ nhiều interface.
* Bạn phải dùng từ khóa implements để triển khai interface.

### **Kế thừa trong interface**

Bạn có thể triển khai class từ một interface đã được kế thừa.

|  |  |  |
| --- | --- | --- |
| 0  1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26 | | interface animal{       void eat();  }  interface dog extends animal{      void drink();  }  class BabyDog implements dog{        @Override      public void eat() {          System.out.println("Eating");      }        @Override      public void drink() {          System.out.println("Dinking");      }  }  public class TEST {      public static void main(String[] args){          BabyDog dog = new BabyDog();          dog.eat();          dog.drink();      }  } |
| 0  1  2  3 | Eating  Dinking | | |

Trong ví dụ trên interface dog được kế thừa từ interface animal. Sau đó class BabyDog triển khai interface dog và phải overriding lại hai phương thức eat và drink.

#### **Lồng interface**

Một interface có thể chứa một interface khác, đó gọi là lông interface.

|  |  |  |
| --- | --- | --- |
| 0  1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18 | | interface animal{      interface dog{          void drink();      }  }  class BabyDog implements animal.dog{      @Override      public void drink() {          System.out.println("Drinking");      }  }  public class TEST {      public static void main(String[] args){          BabyDog dog = new BabyDog();          dog.drink();      }  }    Out put: |
| 0  1  2 | Drinking | |

Sự khác nhau giữa Interface và Abstract class.

|  |  |  |
| --- | --- | --- |
| **STT** | **Interface** | **Abstract class** |
| 1 | Chỉ có phương thức abstract. | Có phương thức abstract và non-abstract. |
| 2 | Có thể đa kế thừa. | Chỉ có thể kế thừa từ một class. |
| 3 | Interface chỉ có các biến **static và final**. | Abstract class có các biến **final, non-final, static and non-static**. |
| 4 | Sử dụng từ khác interface để khai báo. | Sử dụng từ khóa abstract để khai báo. |

### 1. Object trong Java

Object (đối tượng) là một thực thể vật lý, có thể là một con vật, một đồ vật… Ví dụ như ngôi nhà, máy bay, xe máy, con người…

Mỗi đối tượng sẽ có đặc trưng riêng của nó:

* Trạng thái của đối tượng: thể hiện ở giá trị của các biến trong class (các field của đối tượng), ví dụ cái xe màu gì, bao nhiêu phân khối, giá tiền…
* Hành vi: các method (phương thức) của Class, hay được hiểu là các hành động của đối tượng. Ví dụ cái xe có thể chạy, phát tiếng còi, phát ánh đèn…
* Định danh: việc nhận diện đối tượng được triển khai thông qua một ID duy nhất. Giá trị của ID là không thể nhìn thấy với người dùng bên ngoài. Nhưng nó được sử dụng nội tại bởi JVM để nhận diện mỗi đối tượng một cách duy nhất.

Ví dụ: Có một đối tượng bút tên là Thiên Long, có màu đen, … được xem như là trạng thái của nó. Nó được sử dụng để viết, do đó viết là hành vi của nó.

### 2. Class trong Java

Trong Java nói riêng và trong lập trình hướng đối tượng nói chung thì Class (lớp) được hiểu là một nhóm các đối tượng có các đặc điểm chung.

* Ví dụ class xe là một nhóm các đối tượng có bánh xe và dùng làm phương tiện duy chuyển trên đường bộ.

Class là một mô hình chi tiết để bạn sử dụng tạo ra các Object. Class định nghĩa tất cả các thuộc tính và các phương thức cần thiết của một Object.

* Ví dụ class XeOto gồm các thuộc tính (màu sắc, phân khối, tốc độ…) thì ta có thể hiểu đây là thiết kế của một chiếc XeOto, và khi tạo một chiếc xe ô tô (object) từ bản thiết kế đó ta sẽ có nhiều lựa chọn khác nhau: màu sắc (màu trắng, màu vàng, màu xanh, màu đen), phân khối (phân khối 5.2L, 6.3L), tốc độ (300km/h, 400km/h)…

Việc tạo ra các đối tượng phải tuân theo bản thiết kế (class) đã định nghĩa trước đó:

* Ví dụ Class XeOto được thiết kế không có cánh quạt thì thì đối tượng (ví dụ: xe ô tô màu đen, phân khối 5.2L và tốc độ 300km/h) được tạo sẽ không thể nào có cánh quạt.

### 3. Abstract class trong Java

Abstract class hay còn gọi là lớp trừu tượng đơn giản được xem như một Class cha cho tất cả các Class có cùng bản chất. Do đó mỗi lớp dẫn xuất (lớp con) chỉ có thể kế thừa từ một lớp trừu tượng. Bên cạnh đó nó không cho phép tạo instance, nghĩa là sẽ không thể tạo được các đối tượng thuộc lớp đó.

Một lớp được khai báo với từ khóa abstract là lớp trừu tượng (abstract class).

Lớp trừu tượng có thể có các phương thức: Abstract method hoặc Non-abtract method.

Abstract method là method trống không có thực thi.

Non-abtract method là method có thực thi.

Lớp trừu tượng có thể khai báo 0, 1 hoặc nhiều method trừu tượng bên trong.

Không thể khởi tạo 1 đối tượng trực tiếp từ một class trừu tượng.

### 4. Interface trong Java

Interface được xem như một mặt nạ cho tất cả các Class cùng cách thức hoạt động nhưng có thể khác nhau về bản chất. Từ đó lớp dẫn xuấtcó thể kế thừa từ nhiều lớp Interface để bổ sung đầy đủ cách thức hoạt động của mình (đa kế thừa - Multiple inheritance).

Interface không phải là class.

Interface chỉ chứa những method/properties trống không có thực thi.

Interface giống như một khuôn mẫu, một khung để để các lớp implement và follow.

Các Class có thể kế thừa nhiều interface.

Interface là một contract, các class implement phải triển khai các method theo như interface đã định nghĩa.

### 5. Tổng kết

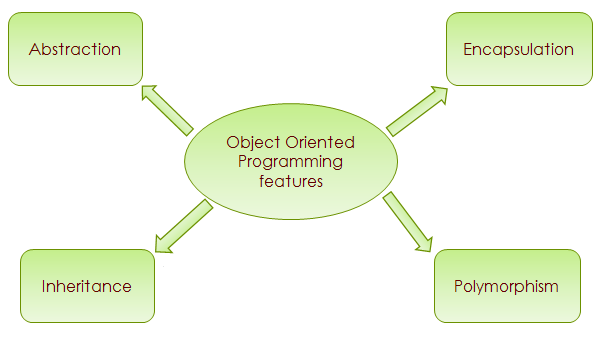
Class là một mô hình chi tiết để bạn sử dụng tạo ra các Object. Class định nghĩa tất cả các thuộc tính và các phương thức cần thiết của một Object.

Mỗi Object phải thuộc một Class nào đó. Và một Object là một thể hiện của Class. Tất cả các Object thuộc về cùng một Class có cùng các thuộc tính và các phương thức.

Bạn không nên nhầm lẫn khi nói về việc một class được implement hay extend.

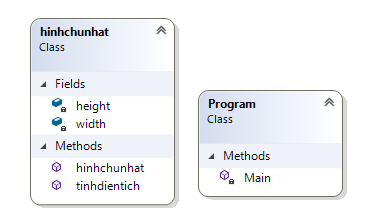
Bạn chỉ có thể thừa kế (extend) từ một class và chỉ có thể hiện thực (implement) các chức năng (interface) cho class của mình. Theo cách ngắn gọn, quan hệ giữa một class khi thừa kế một abstract class được gọi là is-a, và một class khi hiện thực một interface được gọi là can-do (hoặc –able).

Lập trình hướng đối tượng quá quen thuộc rồi bạn nào học lập trình đều phải học, đi phỏng vấn cũng vậy hỏi suốt(chắc cái này tùy vào vị trí tuyển dụng chủ yếu junior chắc chắn sẽ hỏi).nó là nền tảng cho hầu hết các design pattern hiện nay.Bài viết này đúc rút kinh nghiệm thực tế và độ hiểu của mình về OOP. Lập trình hướng đối tượng là một kỹ thuật lập trình cho phép lập trình viên tạo ra các đối tượng trong code để trừu tượng hóa các đối tượng thực tế trong cuộc sống.



### **Tính đóng gói (Encapsulation):**

Là cách để che dấu những tính chất xử lý bên trong của đối tượng, những đối tượng khác không thể tác động trực tiếp làm thay đổi trạng thái  chỉ có thể tác động thông qua các method public của đối tượng đó. Mình sẽ tạo ra 2 class để thể hiện điều này:



xem cách thể hiện bằng code dưới đây : **class hinhchunhat**

using System;

namespace oop

{

class hinhchunhat

{

private int height;

private int width;

public hinhchunhat(int newHeight, int newWidth) {

height = newHeight;

width = newWidth;

}

public int tinhdientich() {

return height \* width;

}

}

}

* height và width ở đây chính là các tính chất (**properties**) của đối  tượng **class** **hinhchunhat**
* **tinhdientich()** là method được public nhằm mục đích tương tác với các đối tượng khác. Tạo một **class Program** với method **static** để run, xem cách tương tác và thay đổi tính chất  của đối tượng thông qua các method public như nào:

using System;

namespace oop

{

class Program

{

static void Main(string[] args)

{

//thay doi properties (height, width) cua doi tuong thong qua method public

hinhchunhat hcn = new hinhchunhat(10, 5);

//lay du lieu thong qua method public

Console.WriteLine("Dien tich cua {0} la: " + hcn.tinhdientich(),hcn);

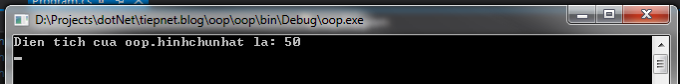
Console.ReadLine();

}

}

}

Output:



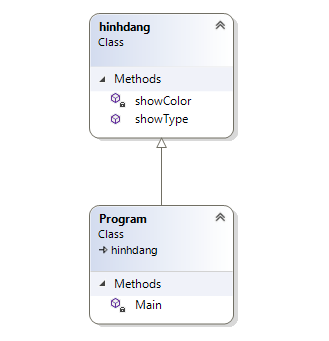
Như vậy khi ta muốn thay đổi các tính chất (properties) không thể tương tác trực tiếp với properties mà phải thông qua các method public được định nghĩa bên trong class

* không thể biết luồng xử lý logic bên trong của đối tượng

### **Tính kế thừa (Inheritance):**

Là kỹ thuật cho phép kế thừa lại những tính năng mà một đối tượng khác đã có, giúp tránh việc code lặp dư thừa mà chỉ xử lý công việc tương tự.

* Kế thừa một cấp (Single level Inheritance): Với một class cha và một class con



using System;

namespace oop

{

class hinhdang

{

private void showColor()

{

Console.WriteLine("Mau hong");

}

public void showType()

{

Console.WriteLine("Day la hinh chu nhat");

}

}

}

using System;

namespace oop

{

class Program : hinhdang

{

static void Main(string[] args)

{

Program pg = new Program();

pg.showType();

//pg.showColor(); khong the truy cap private method

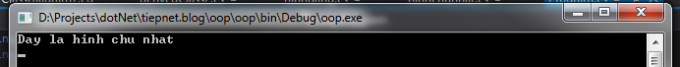
Console.ReadLine();

}

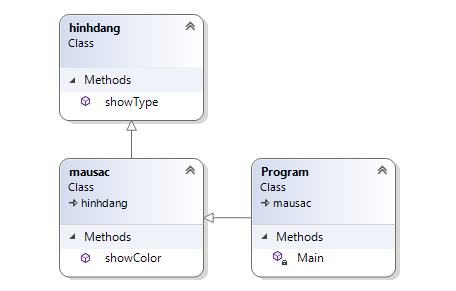
}

}

Output:



Trong class Program không hề có method **showType()** nhưng vẫn có thể truy cập sử dụng nó bằng cách kế thừa lại method của **class hinhdang** **Kế thừa nhiều cấp (Multiple level Inheritance):** Kế thừa  nhiều class.



Ở diagram trên mình viết thêm class mausac và chuyển private  method showColor() sang class mausac  thành public method.

using System;

namespace oop

{

class hinhdang

{

public void showType()

{

Console.WriteLine("Day la hinh chu nhat");

}

}

}

using System;

namespace oop

{

class mausac : hinhdang

{

public void showColor()

{

Console.WriteLine("Mau hong");

}

}

}

using System;

namespace oop

{

class Program : mausac

{

static void Main(string[] args)

{

Program pg = new Program();

pg.showType();

pg.showColor();

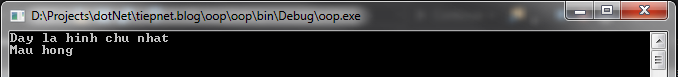
Console.ReadLine();

}

}

}

Output:



**class Program** chỉ kế thừa **class mausac** nhưng vẫn có thể truy cập method **showType()** được viết trong **class hinhdang**, đây chính là hình thức kế thừa nhiều cấp,rất tiện đúng không?

### **Tính đa hình (Polymorphism ):**

Là một đối tượng thuộc các lớp khác nhau có thể hiểu cùng một thông điệp theo cách khác nhau.

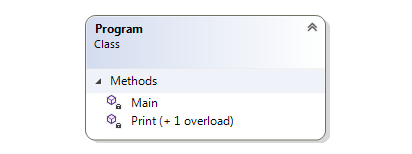
Ví dụ đa hình trong thực tế: Mình có 2 con vật: chó, mèo hai con vật này khi nhận được mệnh lệnh  là **"hãy kêu"** thì chó kêu "gâu gâu",  mèo kêu "meo meo".

Ví dụ trên cả 2 con vật đều hiểu chung một thông điệp **"hãy kêu"** và thực hiện theo cách riêng của chúng.

Trong code để thể hiện tính đa hình có 2 cách:

1. Method Overloading (compile time polymorphism)
2. Method Overriding (run time polymorphism)

* **Method Overloading :** là cách nạp chồng các method có cùng tên nhưng khác tham số



Tạo 1 **class Program** gồm 2 method **Print()** có tham số khác nhau (hai method này được gọi là medthod overloading)

using System;

namespace oop

{

class Program

{

static void Print(object o)

{

Console.WriteLine("Object overload called");

}

static void Print(string a)

{

Console.WriteLine("String overload called");

}

static void Main(string[] args)

{

object o = "hello";

Print(o);

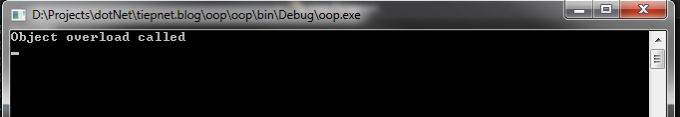
Console.ReadLine();

}

}

}

Output:

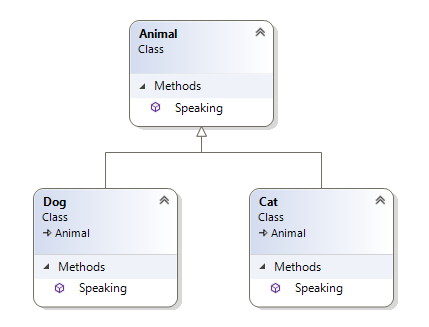


Ở đây **Print(object)** được gọi vì **o** là loại **object**, để xác định method nào được gọi ra chúng xác định bằng loại param hoặc số lượng param chuyền vào.

* **Method Overriding:**  Đây là một phương pháp được ghi đè lại các method ảo của một lớp cha nào đó(được khai báo bằng từ khóa virtual).

Để thể hiện phương pháp này cần dùng 2 từ khóa:

* **virtual** :từ khoá dùng để khai báo 1 phương thức ảo (có thể ghi đè được).
* **override**: từ khoá dùng để đánh dấu phương thức ghi đè lên phương thức của lớp cha.



Tạo ra 3 class **Animal**,**Dog**,**Cat**

using System;

namespace oop

{

class Animal

{

public void Speak()

{

Console.WriteLine("Animal is speaking...");

}

}

}

using System;

namespace oop

{

class Dog: Animal

{

public new void Speak(){

Console.WriteLine("Dog speaks go go");

}

}

}

using System;

namespace oop

{

class Cat:Animal

{

public new void Speak()

{

Console.WriteLine("Cat speaks meo meo");

}

}

}

using System;

namespace oop

{

class Program

{

static void Main(string[] args)

{

Animal dog = new Dog();

dog.Speaking();

Animal cat = new Cat();

cat.Speaking();

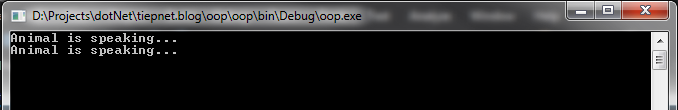
Console.ReadLine();

}

}

}

Output:



Như bạn có thể thấy ở trên mình không dùng đến 2 từ khóa là **virtual** và **override**, mặc dù mình khởi tạo lớp **Dog** và sử dụng method **Speak()** kết quả in ra dữ liệu của method trong lớp **Animal** như vậy không thể hiện được tính đa hình.

Sau đây mình sẽ sử dụng 2 từ khóa **virtual** và **override**:

using System;

namespace oop

{

class Animal

{

public virtual void Speak()

{

Console.WriteLine("Animal is speaking...");

}

}

}

using System;

namespace oop

{

class Dog: Animal

{

override

public void Speak(){

Console.WriteLine("Dog speaks go go");

}

}

}

using System;

namespace oop

{

class Cat : Animal

{

override

public void Speak()

{

Console.WriteLine("Cat speaks meo meo");

}

}

}

using System;

namespace oop

{

class Program

{

static void Main(string[] args)

{

Animal dog = new Dog();

dog.Speak();

Animal cat = new Cat();

cat.Speak();

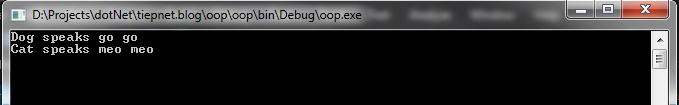
Console.ReadLine();

}

}

}

Output:



Khi sử dụng **virtual** và **override** kết quả đã thể hiện được tính đa hình, lúc này từ khóa **override** đã được ưu tiên và ghi đè phương thức ảo từ lớp cha, khi các đối tượng gọi chung phương thức **Speak()** nó sẽ được trỏ tới phương thức tương ứng của mỗi đối tượng được khởi tạo.

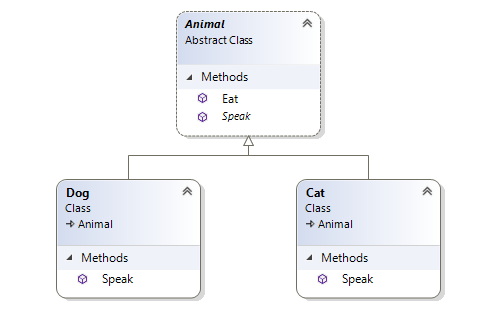
### **Tính trừu tượng(Abstraction):**

Là phương pháp trừu tượng hóa định nghĩa lên những hành động, tính chất của loại đối tượng nào đó cần phải có. Ví dụ khi bạn định nghĩa một lớp động vật(Animal), Animal thì có rất nhiều loại, làm sao để xác định đó là một loại động vật? lúc này bạn sẽ hình dung trong đầu động vật có những tính chất hành vi cơ bản nhất định phải có như ăn, nói khi bất kỳ một developer nào định viết một đối tượng thuộc lớp động vật sẽ kế thừa lại lớp Animal có 2 hành vi ăn, nói,đối tượng được tạo ra có thể khác nhau như chó hoặc mèo nhưng đều có những hành vi của động vật là ăn và nói. **=>** Trong ví dụ trên nhìn vào hành vi ăn và nói của chó và mèo ta có thể khẳng định nó thuộc lớp động vật. Vậy chốt lại rõ làng tính trừu tượng ở đây sinh ra chủ yếu để trừu tượng hóa và định nghĩa các tính chất hành vi phải có để xác định đó là đối tượng gì dựa vào tính chất hành vi của đối tượng. **=>** Các method trừu tượng đều rỗng không thực hiện bất kỳ hành vy nào, hành vy sẽ được triển khai cụ thể do các đối tượng kế thừa. **=>** Viết xong đoạn trên không biết các bạn đọc có hiểu không @@ thấy có vẻ lan man quá, vì chưa có kinh nghiệm @@, nói chung định nghĩa một phần phải thực hành nhiều coder mà, thực hành nhiều tự các bạn sẽ hiểu ra ).

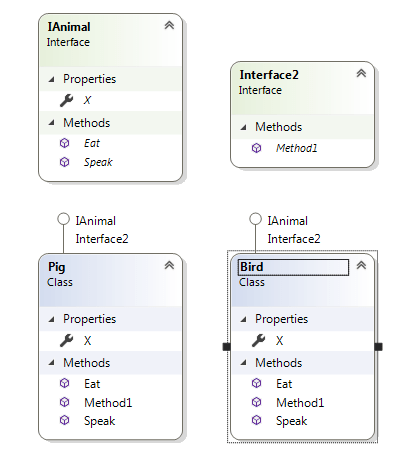
Tiếp tục nhé ^^ trong c# có 2 phương pháp để triển khai tính trừu tượng này:

1. Abstract class

* trong abstract class có 2 loại method:
  + abstract method (là method rỗng không thực hiện gì)
* method thường (là vẫn có logic trả về data hoặc thực thi hành động nào đó, nó được sử dụng cho mục đích dùng chung)



Abstract class Animal và 2 class Dog, Cat kế thừa lại những method được public 2. **Interface** : Khá giống với abstract class nhưng interface không phải là class, trong interface chỉ  có khai báo những method/properties trống không có thực thi, thực thi sẽ được thể hiện trong các lớp kế thừa, interface giống như một cái khung mẫu để các lớp implement và follow.



Tạo 2 interface IAnimal,Interface2 và 2 class Pig,Bird kế thừa 2 interface